Fall 2011

A Machine Learning and Compiler-based Approach to Automatically Parallelize Serial Programs Using OpenMP

Nam Quang Lam
San Jose State University

A Machine Learning and Compiler-based Approach to Automatically Parallelize Serial Programs Using OpenMP

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

By

Nam Quang Lam

December 2011

SAN JOSE STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

A MACHINE LEARNING AND COMPILER-BASED APPROACH TO AUTOMATICALLY PARALLELIZE SERIAL PROGRAMS USING OPENMP

by
Nam Quang Lam

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Teng Moh,	Department of Computer Science	Date
Prof. Ronald Mak,	Department of Computer Science	Date
Dr. Jon Pearce,	Department of Computer Science	Date

APPROVED FOR THE UNIVERSITY

Associate Dean	Office of Graduate Studies and Research	Date

ABSTRACT

Single core designs and architectures have reached their limits due to heat and power walls. In order to continue to increase hardware performance, hardware industries have moved forward to multi-core designs and implementations which introduces a new paradigm in parallel computing. As a result, software programmers must be able to explicitly write or produce parallel programs to fully exploit the potential computing power of parallel processing in the underlying multi-core architectures. Since the hardware solution directly exposes parallelism to software designers, different approaches have been investigated to help the programmers to implement software parallelism at different levels. One of the approaches is to dynamically parallelize serial programs at the binary level. Another approach is to use automatic parallelizing compilers. Yet another common approach is to manually insert parallel directives into serial codes to achieve parallelism.

This writing project presents a machine learning and compiler-based approach to design and implement a system to automatically parallelize serial C programs via OpenMP directives. The system is able to learn and analyze source code parallelization mechanisms from a training set containing pre-parallelized programs with OpenMP constructs. It then automatically applies the knowledge learned onto serial programs to achieve parallelism. This automatic parallelizing approach can be used to target certain common parallel constructs or directives, and its results when combined with a manual parallelizing technique can achieve maximum or better parallelism in complex serial programs. Furthermore, the approach can also be used as part of compiler design to help improve both the speed and performance of a parallel compiler.

ACKNOWLEDGEMENTS

I would like to thank my advisors Dr. Robert Chun and Dr. Teng Moh for their dedication, insight, and guidance throughout the period of this writing project. I also would like to express many thanks to committee member Prof. Ronald Mak for his valuable time to provide support and advice. The three of you are great assets to the university and to the students. Knowledge required to complete this project comes directly from all of your classes.

I also would like to take this opportunity to thank my parents for their encouragement and to especially thank my wife, Mai Nguyen, for her encouragement and continuous support for the completion of this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction and Motivation

Over the past decades, computer hardware designs and architectures have rapidly evolved in both macroscopic and microscopic directions [6]. Detailed descriptions of the evolutions of computer hardware architectures are beyond the scope of this writing project. In other words, the following macroscopic and microscopic views of hardware solutions give a brief introduction but adequate enough to show a big picture of hardware trends that lead to our current explicit parallel programming paradigm.

Macroscopically, in 1981, IBM introduced its first line of personal computers (PCs) to the consumer market [7, 13]. This was a major breakthrough in personal computing industry; this was also the beginning of the era of single node computers or desktop workstations that are still being used today. A stand-alone computer is used mostly by an individual in an office or at home to carry out small daily tasks. Computing power of a single PC is limited and self-contained. In order to be able to run compute-intensive or heavy workload programs, such as scientific applications, multiple computers are put together to share the computational workload of a large application; this is the basic idea of cluster computing. A cluster computer is group of tens to hundreds of workstations or nodes commonly interconnected via a local area network. All nodes are usually very highly coupled [8, 9, 10]. A cluster is an integration of computing resource that has a frontend node to provide access to various services performed by a collection of backend nodes. Thus, a cluster computer appears as a single supercomputing system to the end users. Another technological advancement that also makes use of multiple computer resources to create a virtual supercomputer to achieve high-performance computing environment to run complex and resource intensive

applications is grid computing. A grid is similar to a cluster in a way that both integrate many computers together to work as one; however, a grid is composed of thousands of nodes that are more loosely-coupled and are geographically distributed, resulted in a suitable computing environment for a variety of distributed applications. Grids use special-purpose software libraries called middleware to coordinate the work and resources among the nodes [18]. In short, the grid architecture is reliable and scalable to integrate and utilize multiple resources including, but not limited to, clusters, mass repositories, specialized computers, software, etc. to serve a common goal or various computing purposes [10, 11, 12]. Evolved from the grid is cloud computing, a current hot research topic and also a future information technology for everyone. Cloud computing is a TCP/IP- or Internet-based computing that utilizes most or all aspects of grid computing, utility computing, virtualization, and service-oriented architecture (SOA) [12, 14, 17]. As a result, cloud computing is very scalable, loose coupling, highly secure, service-oriented, and has strong fault tolerance. High-level overview of the cloud architecture, which is transparent or abstracted from the end users, includes cloud service, cloud platform, cloud infrastructure, and cloud storage. With a single point of entry via a thin client, a user is able to access a variety of on-demand services provided by the cloud, including Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [15, 16]. A thin client accessing the cloud is analogous to a terminal accessing the mainframe; however, a subtle difference is that the mainframe provides concurrent data access concurrent programming whereas the cloud provides parallel access to on-demand services at larger scope. In short, macroscopic evolution of hardware architecture directly exposes parallel programming to the programmers.

Microscopically, computer processors in the past decades have gone through a significant evolution in terms of clock frequency and number of transistors. In 1981, when IBM first PC was introduced to the market, it was featured with a single Intel 8088 microprocessor running at 4.77 MHz and containing fewer than 30,000 transistors [13, 19]. However, in 2006, an Intel Core 2 Duo was shining at an amazing clock frequency of 2.9 GHz and having around 291,000,000 of transistors [6]. This correctly validated Moore's law, which stated that number of transistors would double approximately every 18 to 24 months [1, 6]. Yet, everything has its limit. Single core designs and architectures have reached their limits due to heat and power walls. In order to continue to increase hardware performance, hardware industries have moved forward to multi-core designs and implementations, which introduces a new paradigm in parallel computing [1, 6]. Just as macroscopic view, in short, microscopic evolution of hardware architecture also directly exposes parallel programming to the programmers.

Hardware solution has dragged software industries into a higher demand for parallelism [6]. As a result, software programmers must be able to explicitly write or produce parallel programs to fully exploit the potential computing power of parallel processing in the underlying multi-core architectures. Since the hardware solution directly exposes parallelism to software designers, different approaches have been investigated and/or implemented to help the programmers to implement software parallelism at different levels. One of the approaches is to dynamically parallelize serial programs at the binary level [33]; another approach is to use automatic parallelizing compilers [24, 31]. Yet another common approach is to manually insert parallel directives into serial codes to achieve parallelism [1, 2].

This writing project presents a machine learning and compiler-based approach to design and implement a system to automatically parallelize serial C programs via OpenMP directives. The source language for the system implementation is Java, and the target programs are written in the C language. The system is able to analyze and learn source code parallel mechanisms from a training set containing pre-parallelized programs with OpenMP constructs. It then automatically applies the knowledge learned onto serial programs to achieve parallelism. Practically, the project focuses on work-sharing at loop-level because loops take all or most of the execution time of a running program. This automatic parallelizing approach can be used to target certain common parallel constructs or loop directives, and results then combined with a manual parallelizing technique to achieve maximum or better parallelism in complex serial programs. Furthermore, the approach can also be used as part of compiler design; it helps to improve both speed and performance of a parallel compiler.

2. Related Work

Over the years, many different approaches have been researched and implemented to help software developers to write parallel programs or to exploit parallelism for multi-core processors; however, not that many of the approaches are directly related to the work presented in this project. This project proposes a new way to achieve automatic parallelism for the underlying multi-core architectures via OpenMP using the combination of machine learning and compiler-based methods. As a result, it is not an easy task to actually compare the performance of the implementations of other approaches with this project's approach. This section describes some of previous works that are related to this project only on certain aspects but relevant enough to contribute to knowledge base for completing this writing project.

2.1. "Adaptive Multi-versioning for OpenMP Parallelization via Machine Learning" [20]

In general, OpenMP APIs allow programmers to produce parallel programs by manually inserting OpenMP directives or constructs into corresponding serial programs to guide the compiler about which regions of the codes can be executed in parallel [2]. Details of OpenMP are covered in the later Background section. However, OpenMP does not guarantee an optimal execution of the resulting programs. Motivated from this fact, the authors of the paper, Adaptive Multi-versioning for OpenMP Parallelization via Machine Learning [20], presented a mechanism to be integrated into a compiler to adaptively achieve parallelism in a cost-aware manner. The mechanism guarantees that the implemented compiler would be able to properly parallelize the workload of a program to

produce a corresponding optimally performing OpenMP parallel version at runtime for a multi-core system.

Briefly, the proposed solution works as follows. When the compiler encounters a loop in a serial program, it generates multiple OpenMP parallel versions (i.e., multi-versioning) of the loop with various numbers of threads for each version that it can evaluate with various inputs of different sizes, as demonstrated in Figure 1. The compiler then selects a proper number of representative versions using the algorithm shown in Figure 2, which will be embedded into the final executable. With proper mapping features, an appropriate version is selected to execute at runtime.

Actually, it is very difficult or almost impossible to select an appropriate number of threads at compile time for a specific program to be properly executed in a parallel environment that yields an optimal performance. Therefore, a technique called adaptive optimization [20] is recommended to provide best-possible solution for selecting a suitable number of threads at runtime. Adaptive optimization can be dynamic compilation or multi-versioning; yet the authors decided to choose multi-versioning technique because it virtually beats all adaptive static versions in terms of different runtime contexts while “dynamic compilation needs extra time for runtime recompilation of some code regions.”

The authors made use of K-Nearest Neighbor machine learning algorithm [3, 4] for runtime version selection. The basic idea is that when a loop is encountered at runtime, its input sizes are computed and compared with a pre-calculated version’s input size which results in a corresponding number of threads of the closest match embedded version to be chosen. The machine learning technique being used here is less similar to a

switch programming structure with an extra capability to choose a closest match entity. It is not fully implemented as an end-to-end machine learning approach [3, 4] (or at least not mentioned in the paper). Details of machine learning are covered in the later Background section.

One can appreciate that the main contribution of the paper to parallelism is the pre-calculation process to obtain a best possible selection of representative OpenMP versions of loops which will be chosen at runtime to yield an optimal parallel workload allocation. This adaptive multi-versioning technique would definitely help compiler designers to implement a parallel cost-aware compiler for current and future multi-core architectures.

```
// the sequential version
for (i=0; i<N; i++)
    … …  // loop body
```

(A) a given loop in its original sequential form

```
// the OpenMP parallel version
switch (certain conditions) {
  case …: // the tn0 thread version
     #ifndef NOOMP
     omp_set_num_threads(tn0);
     #pragma omp parallel default(none)
     { #pragma omp for
       #endif
       for (i=0;i<N;i++)
          … … // loop body
     } break;
  … …
  // more OpenMP version with various numbers
  // of threads
  case …: // the tnv-1 thread version
     #ifndef NOOMP
     omp_set_num_threads(tnv-1);
     #pragma omp parallel default(none)
     { #pragma omp for
       #endif
       for (i=0;i<N;i++)
          … …  // loop body
     } break;
  … …
  default: // the tnv thread version
     #ifndef NOOMP
     omp_set_num_threads(tnv);
     #pragma omp parallel default(none)
     { #pragma omp for
       #endif
       for (i=0;i<N;i++)
          … …  // loop body
     } break;
}
```

(B) its adaptive multi-versioned OpenMP version

Figure 1. Sequential version (A) and corresponding multiple OpenMP parallel versions (B) [20]

```
// P: the program to be parallelized
// v: a predefined no. of candidate versions
// s: the set of thread numbers selected
generate versions of P by parallelizing it with
     various numbers of threads tns;
test-run each these versions Pis with inputs of
     various sizes, and record their performances;
for each input size {
   find the best performance BP;
}
for each test-run {
   calculate each version's efficiency;
   if (its efficiency reaches a certain level)
     award the corresponding tn a certain points;
}
sort all tns in a temporary list L according to their
     points awarded
s = {}; count =0;
repeat {
   pick the first thread number tn from L;
   if (tn is not too close to any element in s) {
      s = s+{tn}; count++;
   }
} until (count ==v);
generate parallel versions of P according to
   the thread numbers selected in s;
```

Figure 2. Pseudo-code of the Candidate Selection Algorithm [20]

2.2. "Automatic Parallelization of Sequential C Code" [22]

Motivated from the high demand for parallel applications in order to fully take advantages of computing power in cluster environments, the authors of the paper, "Automatic Parallelization of Sequential C Code" [22], presented an automatic parallelization approach to parallelize serial C programs to be executed under cluster-based architectures. This is an ongoing project that still needs help from user inputs to actually carry out the automatic parallelizing process.

The goal of the project is to provide a useful tool to help the users to comprehend the inside of a given program, mainly about loop structures, for parallelism. At a high level overview, the tool works as follows. It analyzes serial source codes and generates a

graphical presentation of the program execution to the users. Based on the presented structure, the users are able to define parallelizable regions of codes to guide the automation process. The result is a Message Passing Interface (MPI) parallel program. MPI implementations allow programmers to achieve parallelism in distributed memory environments [6, 25, 37] that are suitable for cluster computing architectures.

Derived from ideas of compiler design technique and framework (details of compiler design are covered in the later Background section), the project was implemented as shown in Figure 3. First, a programming language utility named ANTLR (ANother Tool for Language Recognition) was used to build the C language parser needed to parse and process source code of a C program and to build a corresponding AST (abstract syntax tree) [22, 40]. Then, a data structure was implemented to classify the source code into basic-blocks in order to construct a graphical presentation of the program workflow, focusing on loop level. A graphical user interface was also designed for various user interactions. Since this is an ongoing project, the authors postponed the Dependency Analysis feature for later development. So, as the last stage of the development, an MPI Workpool model and code generator were designed and implemented to actually put together all the pieces, such as the flow graph, AST, various data structures, user inputs, etc., to properly generate MPI-based parallel programs for cluster computing architectures.

Although this project is ongoing and still relies on user inputs to actually carry out the automatic execution, it has made a great start for automatic parallel programming in cluster-based environments. Specifically, both distributed application developers and

compiler designers can benefit from the ideas, techniques, and tool presented in this paper for current and future parallel program implementations.

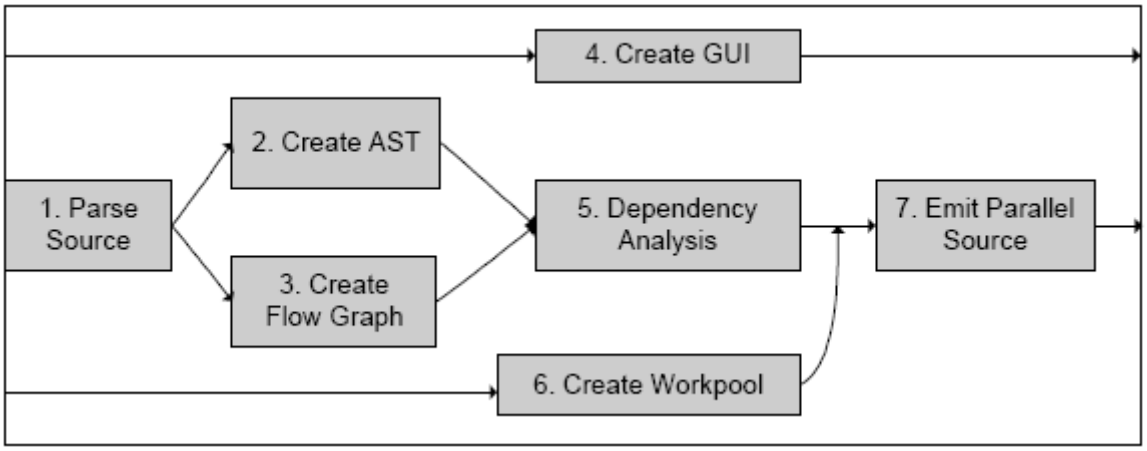


Figure 3. Application Development Model [22]

2.3. "The Algorithm of Parallel Recognition Based on Data Dependence" [27]

In this paper, the authors investigated and presented an algorithm that would be used to recognize parallelizable regions or blocks of code in serial programs based on data dependence. Motivated from the fact that it is difficult and takes time to manually parallelize large and complex serial programs, the goal of the algorithm is to provide a mechanism to be integrated into modern compilers to achieve automatic program parallelization or to design an automatic parallel compiler. The basic idea behind this paper is briefly described as follows.

In order to parallelize a serial program, various parts or blocks of the program are extracted and analyzed for data dependence among them. The authors define that a block "is the program element that there is only one entrance and only one exit. The character of the blocks is that the whole codes of blocks either must be executed or must not be

executed." [27] Data dependence between blocks is recognized based on input and output variables that the blocks have access at the same time. If output variables of a block depend on input variables of another block and vice versa, these blocks have data dependence and cannot be executed or computed in parallel.

Mathematically, data structures that are used to analyze data dependence between blocks compose of various matrices, including input variable matrix M_i, output variable matrix M_o, and dependence matrix M_d. These matrices can be summarized in Table 1 [27].

$M_i(i,j)=\begin{Bmatrix}0, & a_j \notin B_{iI}\\ 1, & a_j \in B_{iI}\end{Bmatrix}$	Input variable matrix M_i sized (m x n) with blocks B_i (rows) and variables a_j (columns), where i = 1,2,3,...m j = 1,2,3,...n 0: a_j does not belong to input variables of B_i 1: a_j belongs to input variables of B_i
$M_o(i,j)=\begin{Bmatrix}0, & a_j \notin B_{iO}\\ 1, & a_j \in B_{iO}\end{Bmatrix}$	Output variable matrix M_i sized (m x n) with blocks B_i (rows) and variables a_j (columns), where i = 1,2,3,...m j = 1,2,3,...n 0: a_j does not belong to output variables of B_i 1: a_j belongs to out variables of B_i
$M_d(i,j)=\begin{Bmatrix}0\\ n(n>0)\end{Bmatrix}$	Dependence matrix M_d with rows and columns are composed of blocks to describe dependency between B_i and B_j. 0: no dependence n: dependence

Table 1. Summary of Data Dependence Matrices

Basically, the goal of the algorithm is to look for dependency status between blocks in serial programs in order to determine whether the blocks can be executed in parallel or not. Major steps of the algorithm can be briefly described as below.

- Step 1: Identify and label the blocks in the source code of a serial program.
- Step 2: Process the variables of the program and build corresponding matrices.
- Step 3: From resulting matrices, determine execution order of the blocks.

This paper has presented a simple but practical algorithm that can be useful to the design of a parallel compiler which can produce parallel programs to be executed on multi-processors environment. However, the authors also indicated that under certain conditions, the algorithm needed to be properly optimized in order to truly achieve parallel performance due to thread scheduling overheads, etc.

2.4. "Mapping Parallelism to Multi-cores: A Machine Learning Based Approach" [26]

As seen in previous sections, there were a variety of methods and technologies that could be used to design and implement parallel programs to fully take advantage of multi-core platform performance. In fact, programmers can write parallel programs from scratch for a particular multi-core platform, or they can manually apply a parallel mechanism to legacy serial programs to run on multi-core systems. In this paper, the authors have introduced a different approach to multi-core programming. In stead of applying parallelization directly to a program, the authors proposed an approach that takes a program that has been already parallelized and maps it to a multi-core processor via machine learning technique.

The paper pointed out that parallelizing a program at source code level is one thing, but efficiently mapping the code to the underlying hardware architecture is another thing. Mapping parallelism to underlying hardware is a very difficult task that is highly platform dependent. Furthermore, hardware architecture is rapidly changing and varies from one generation to the next. Therefore, finding an efficient or best mapping is time consuming and highly depends on various factors and decisions, including tasks or threads scheduling, number of processors, the right amount of parallelism to be exploited, and so on. As a result, it motivated the authors to use machine learning to design an automatic mapping approach for parallel compilers. Main goal of this approach is to automatically predict program scalability to select the correct number of threads and to automatically classify program scheduling policies, which would result in an automatic efficient mapping of parallelism to multi-core processors.

At a high level of overview, the authors made use of two different machine learning techniques called Artificial Neural Network (ANN) [3, 26] and Support Vector Machine (SVM) [3, 26] to build two different learning models to predict program scalability and to classify scheduling policies, respectively. The models were trained using a set of training data that consisted of pre-parallelized programs with selected features and desired mapping decisions. After the models were trained, they would be used to predict and classify a new or unseen parallel program, which resulted in correct mapping decisions for a parallel compiler. Details of machine learning will be covered later in Background section of this paper.

The two figures below are extracted directly from the paper to describe training and predicting processes of the machine learning models.

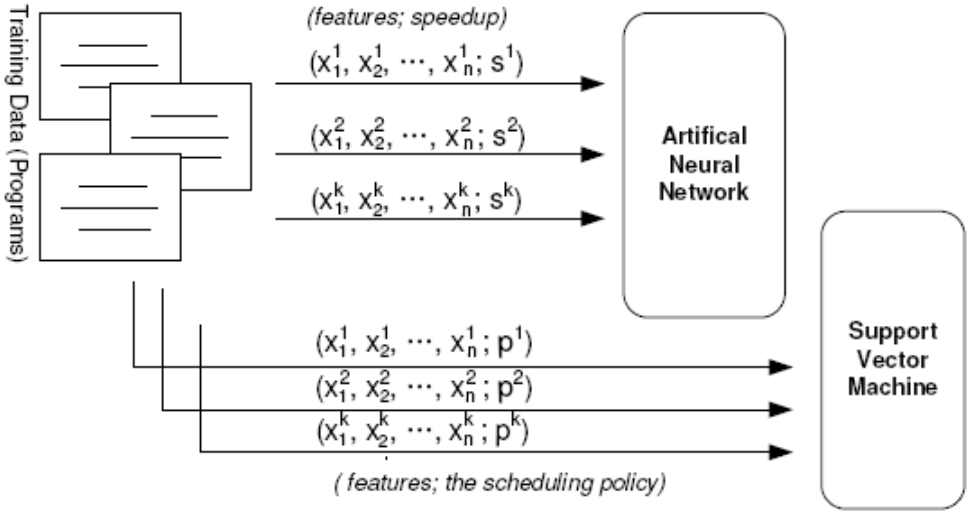


Figure 4. Models Training Process [26]

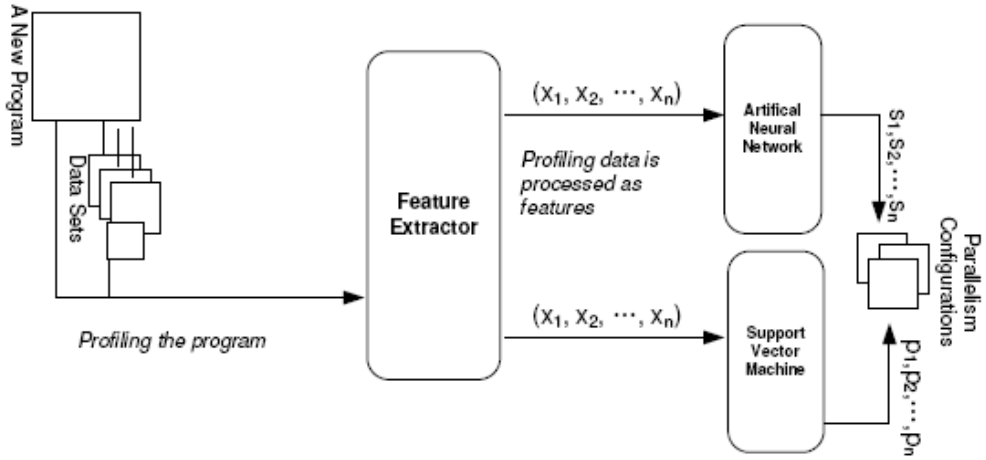


Figure 5. Models Applying Process [26]

The authors did the experiments of the models using different sets of data as well as different types of multi-core platforms featuring both shared-memory and distributed-memory parallel programming models. Their results showed that their approach “has the

best performance on average" and the performance "is stable across programs, data sets, and architectures." [26]

In summary, this article presented two machine learning techniques to build automatic mapping models that are portable and efficient to be integrated into the design of a parallel compiler.

3. Background

3.1. Multi-Core Programming and OpenMP

3.1.1. Parallel Programming Models

Parallel programming is not new. For decades, it has always been a great topic for researchers, computer scientists, or software developers due to its nature in providing exceptional performance gains. In 1989, there was already an investigation in implementing a compiler using an input partitioning technique to exploit parallelism for multiple and independent processors systems [30]. Then in 1990, a functional programming language compiler to generate parallel programs to exploit data-parallelism for massive parallel computers was designed [29]. These are the two examples among many other examples of long history parallel programming. However, it is worth noting that the studies and implementations of high-level parallelism were mainly for commercial multi-processors or distributed systems, not home user computers. One key reason was that single-core processor performance could still be improved by increasing number of transistors and, importantly, by implementing low-level hardware parallelisms, such as Instruction-Level Parallelism (ILP) and hardware pipelining, which are transparent to the software programmers [6].

Nevertheless, low-level implicit hardware parallelizations have reached their limits due to heat and power walls [6], which ended the single-core processors era. In 2005, Intel rolled out a new line of multi-core processors to the consumer market [1]. This has signaled a beginning of high-level explicit parallel programming. As a result, now both system-level and application-level software programmers have to learn to

explicitly write parallel programs in order to exploit the computing power of parallel processing in the underlying multi-core architectures. Generally from hardware design perspective, there are two types of memory architecture [6, 34]: shared-memory architecture and distributed-memory architecture.

Under shared-memory architecture, global memory space is available and shared by all processors; whereas under distributed-memory architecture, each processor has its own memory space. Each design has its own advantages and disadvantages in terms of costs, scalability, complexity, usability, and communication management. Since details of these two hardware designs are beyond the scope of this report, the Figures 6 and 7 show the differences between these two memory models.

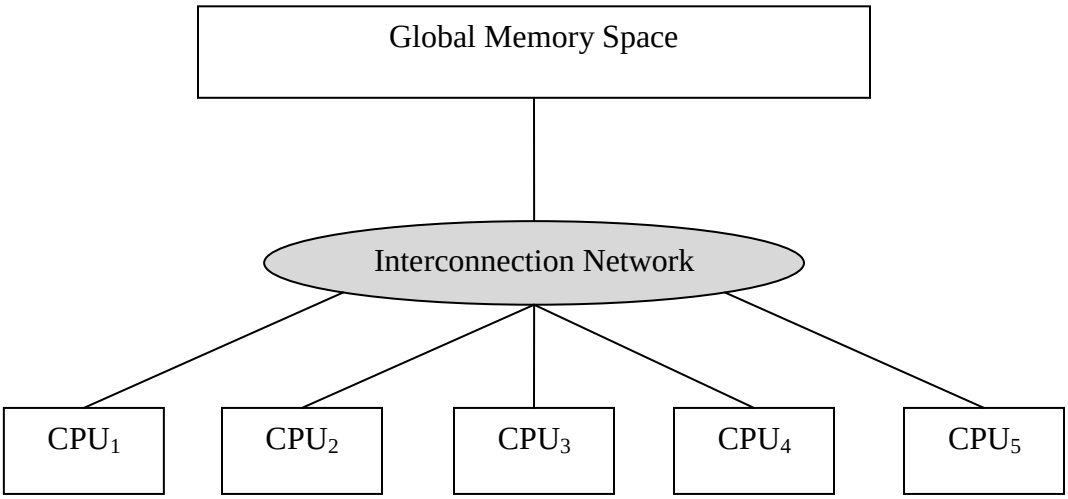


Figure 6. Shared-memory Hardware Architecture

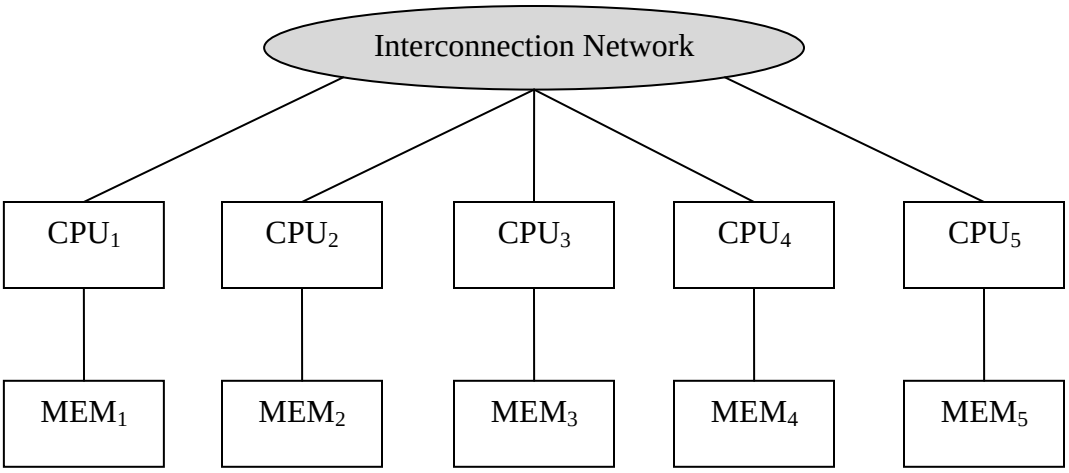


Figure 7. Distributed-memory Hardware Architecture

The two memory architectures have led to two major parallel programming models [6]: shared-memory programming model and distributed-memory (or message passing) programming model. They are briefly described in subsequent sections below according to [2, 6, 34].

Shared-memory parallel programming model:

In this model, a parallel program is executed via multiple independent threads that all have access to their same program address space. These threads are utilized on multi-core processors in a shared-memory architecture. Since data is shared among the threads, proper thread scheduling and synchronization mechanism must be implemented in order to ensure correctness of the program. Specifically, when software programmers write multi-threaded applications, they must be able to handle common multi-threaded problems, such as deadlocks, race conditions, mutual exclusion, and so on. Shared-memory parallel programs can be implemented using OpenMP, a parallel programming API that practically realizes this model. OpenMP is described in depth in its own dedicated subsequent section.

Distributed-memory parallel programming model:

In this model, a parallel program is executed via multiple independent processes each of which has its own address space. These processes are utilized on multi-core processors in a distributed-memory architecture. This distributed-memory model is also called message passing parallel programming model. Since each process operates on its own private address space, the processes communicate data with each other in distributed fashion by explicitly exchanging messages. This implies that data communication requires cooperation between senders and receivers. As a result, software programmers are required to have solid background in distributed computing; specifically, they are responsible for details and correctness of data communication implemented using message passing mechanism. The Message Passing Interface (MPI) standard provides a rich set of library routines that enable software programmers to write efficient distributed-memory parallel code. See Figure 8.

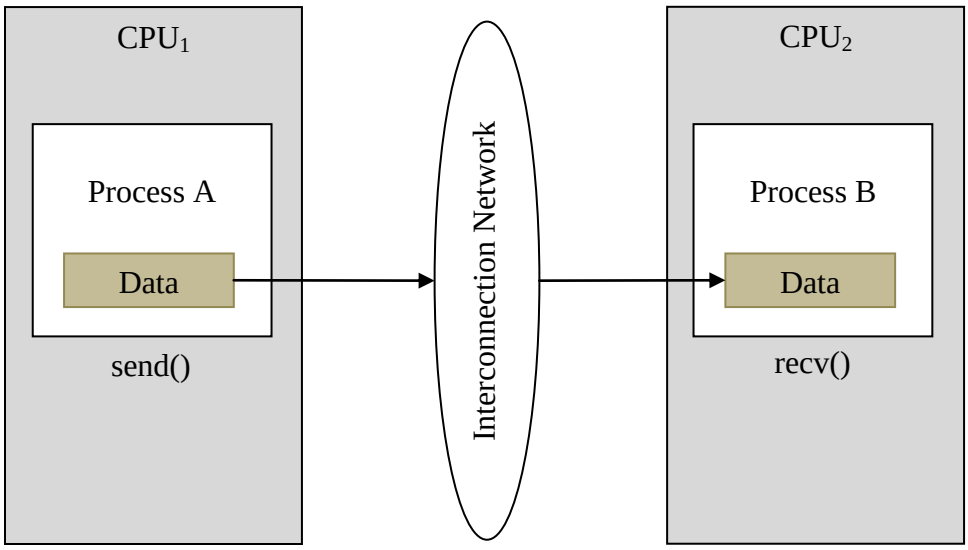


Figure 8. Distributed-memory Message Passing Model

3.1.2. Parallel Programming Patterns

Designing a parallel program requires a programmer first to understand the problem that he or she is trying to solve in parallel. Then, the programmer needs to identify which part of the problem is potentially a good spot to apply parallelization. The programmer has to partition or decompose the problem into smaller chunks or individual tasks that can be executed in parallel [1, 2]. This results in two major parallel programming patterns, which are summarized in the Table 2 and Figures 9 and 10.

Parallel Patterns	Description
Data-level parallelism	This parallel programming pattern realizes parallelism by focusing on the data set associated with a problem. It means that the data set is partitioned into smaller blocks of data, where each block is operated on by an individual thread or process to accomplish a specific task.
Task-level parallelism	Also being called functional-level parallelism, this parallel programming pattern realizes parallelization by focusing on nature of tasks to solve a problem. If the tasks can be performed in parallel, i.e., there has no dependency between them, then they are accordingly scheduled and distributed among threads or processes in the program to be independently executed.

Table 2. Two Major Types of Parallel Programming Patterns

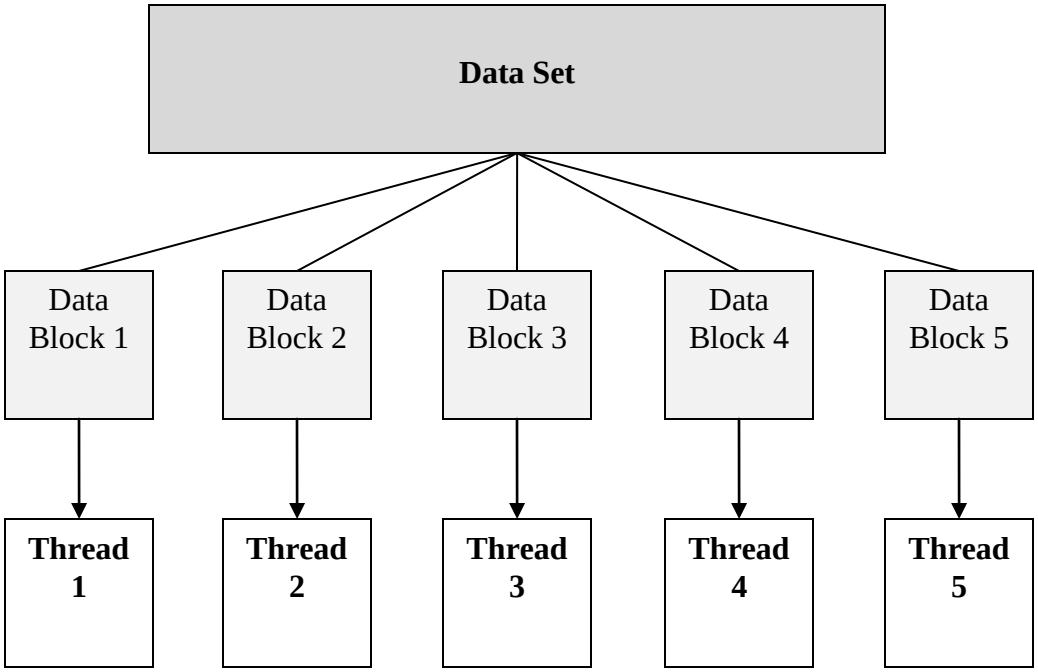


Figure 9. A Typical Data-level Parallelism

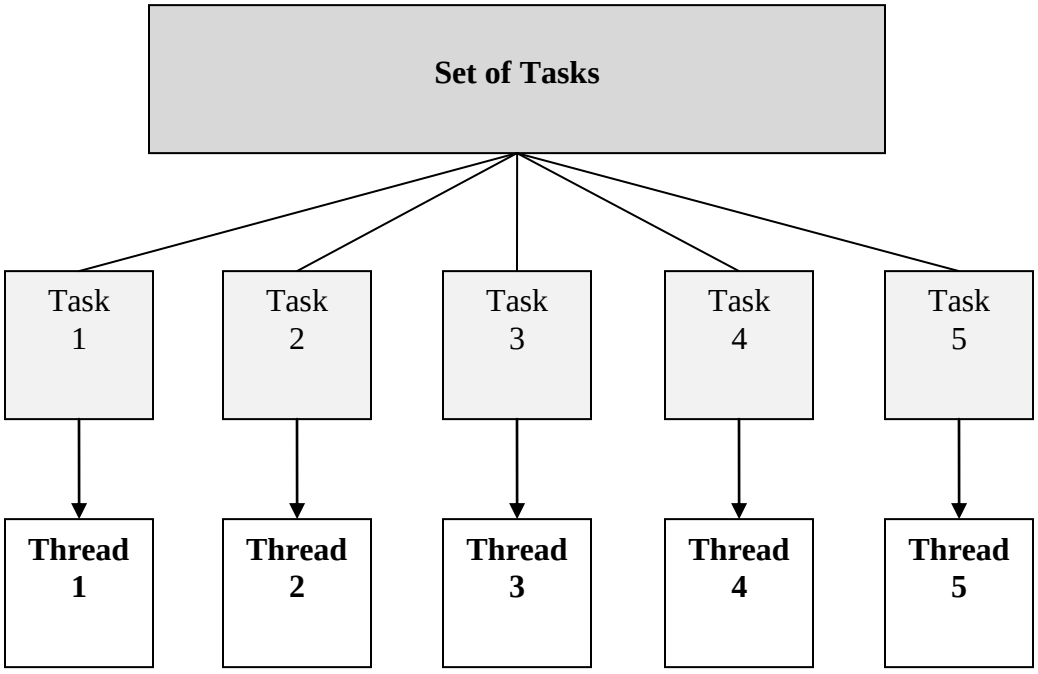


Figure 10. A Typical Task-level Parallelism

3.1.3. Overview of OpenMP

Due to the nonstop increase in the number of shared-memory parallel computers, specially multi-core architectures, OpenMP was developed to implement the shared-memory model. It enables the software programmers the ability to explicitly write parallel software applications that fully exploit and utilize parallel processing power of multi-core processors [1, 2].

Recall that OpenMP (Open Multi-Processing) is a shared-memory Application Programming Interface (API) that implements the shared-memory parallel programming model. OpenMP is not a programming language; it is rather a programming specification that provides application developers a powerful set of compiler directives that are inserted into a serial program written in C, C++, or Fortran to guide a target compiler to generate a corresponding parallel program to run on multi-core systems with minimal changes to the serial code. The OpenMP specification has been created and standardized [2, 42] as a result of mutual agreement among the members of Architectures Review Board (ARB) which now includes "almost all the major computer manufactures, major compiler companies, several government laboratories, and groups of researchers belong to the ARB." [2]

OpenMP is portable, scalable, and easy-to-use. It hides most multi-threaded program development complexities from software programmers so that the programmers can resourcefully concentrate their design and problem analysis for program parallelization. The following section will give a high-level overview of the OpenMP programming or execution model, focus on discussion of certain important features that

are associated with system design and implementation of this writing project, and present some practical OpenMP sample codes.

3.1.4. OpenMP Programming Model and Features

OpenMP provides an explicit, multi-threaded programming model for shared-memory multi-core platforms. In order to manage parallel execution of threads, OpenMP implements the fork-join thread execution model [2, 42]. In this execution model, as illustrated in Figure 11, a program process starts with a single initial thread, called the master thread, which serially executes until it encounters a parallel region. A team of threads is created or forked to share the workload in the parallel region. After the parallel workload execution is completed, this team of threads is released or joined, which leaves only master thread to continue its serial program execution. The entire process repeats when a parallel region is encountered again.

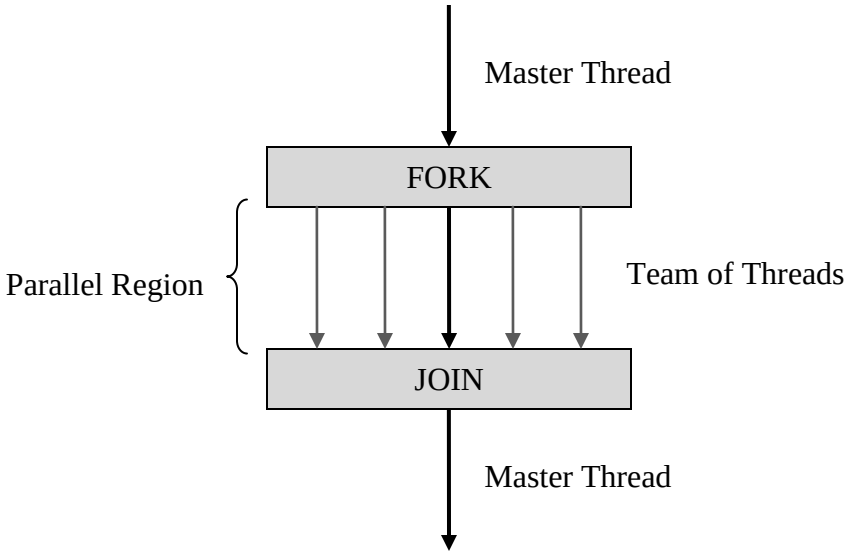


Figure 11. OpenMP Fork-Join Execution Model

Parallel execution in the parallel region can be task-level parallelism or data-level parallelism. General OpenMP syntax structure to initiate a parallel region or parallel construct uses the "*parallel*" directive in a C source file, as shown in Figure 12:

```
#include <omp.h>
int main()
{
    /* some declarations go here */
    int num1, num2;
    /* serial code might be here */
    ...
    /* begin of parallel region */
    #pragma omp parallel shared(num1) private(num2)
    {
        /* code inside this parallel region is
         * executed in parallel by all threads
         */
    }
    /* end of parallel region */
    return 0;
}
```

Figure 12. General OpenMP Syntax Structure

Main features of OpenMP come from its set of compiler directives to create work-sharing constructs. General syntax format of an OpenMP directive is:

#pragma omp directive-name [clause,…] newline

There are two major types of work-sharing constructs, FOR and SECTIONS, which represent data-level parallelism and task-level parallelism, respectively. These two

constructs are created using the "*for*" and "*sections*" directives, which are summarized in Tables 3 and 4 [2, 42].

Work-sharing Construct	Combined Syntax Format in C/C++
FOR	`#pragma omp parallel for [clause...] newline` `schedule(type [,chunk])` `private(list)` `shared(list)` `reduction(operator: list)` `...` *`for_loop`*
SECTIONS	`#pragma omp parallel sections [clause...] newline` `private(list)` `reduction(operator: list)` `...` `{` `[#pragma omp section newline ]` *`structured_block`* `[#pragma omp section newline` *`structured_block ]`* `}`

Table 3. Syntax Format of Two Major OpenMP Work-sharing Constructs

Work-sharing Construct	Description
FOR	This work-sharing construct is initiated using the "*for*" directive. As the name implies, this construct is to apply to loop-level parallelization, specifically, representing data-level parallelism. The construct must be enclosed within a parallel region that has been initiated using the "parallel" directive. The "*for*" directive bears a certain number of clauses to control the parallel execution with the parallel region. OpenMP specification indicates that iterations of a *for* loop that immediately follow this "*for*" directive would be executed in parallel by corresponding team of threads in the region. Distribution behavior of loop iterations depends on the "*schedule*" clause, which will be described, along with other related clauses, in the subsequent paragraphs below.
SECTIONS	This work-sharing construct is initiated using the "*sections*" directive. This construct represents a type of functional- or task-level parallelism. As with the FOR construct, this construct must be enclosed within a parallel region that has been initiated using the "*parallel*" directive. Within a "sections" directive, there are nested "section" directives to specify different portions of code to be executed in parallel by corresponding team of threads in the region. Details of SECTIONS end here because this project is interested only in *for* loops parallelization.

Table 4. Description of Two Major OpenMP Work-sharing constructs

Since the scope of this writing project is to focus on the FOR work-sharing construct, corresponding selective clauses of the OpenMP "*parallel*" and "*for*" directives are described in details in Table 5. These clauses are necessary and important enough to parallelize most common serial C programs.

Commonly-used Clauses	Description
num_threads	*num_threads (integer-expression)* This clause is used to specify number of threads to be created to share workload in parallel region. A common problem associated with this clause is threading overhead when it is wrongly used to create more threads than needed. Program performance associated with the number of threads also depends on the number of cores available in the underlying system. Ideally, the number of threads should equal number of cores available.
shared	*shared (list)* This clause is used to specify that those variables enclosed in the *list* are shared among the threads in parallel region. All threads in the team can asynchronously access, such as read or write, the variables. Users have the responsibility to correctly specify these variables in order to avoid various multi-threaded problems, such as race conditions, deadlocks, etc.

private	*private (list)* This clause is used to specify that those variables enclosed in the list are not shared or are private to each thread in parallel region. That is, each thread in the team is given a unique local copy of each variable so that the thread can safely modify or update the variable. Common usage for this clause is to specify a list of variables whose values are being modified or updated in the parallel region.
default	*default (none|shared)* This clause is used to specify default scope for the variables in parallel region. The default sharing scope for the variable can be either *none* or *shared*. Note that *private* is not an option.
reduction	*reduction (operator:list)* There are some cases where software programmers want to incrementally update and store values of a variable, such as summation problems. In some sense, the variable needs to be shared because all threads in the team need to have access to the value; at the same time, the variable also needs to be private because its value is being modified which might result in race conditions. OpenMP provides the users an easy way to implement this kind of recurrence calculations via the use of *reduction* clause, where *operator* can be an associative operator or a commutative operator that operates on the variables in the *list*. There is no need to specify these variables in a *shared* clause.

schedule	*schedule (type[, chunk-size])* This is the most important clause used in the loop construct. This clause provides the programmers a way to control the behavior or manner of the distribution of number of iterations over the threads in the team, where: *chunk-size* is an optional integer expression that results in a positive integer value. The number of iterations can be divided into chunks, each with size of *chunk-size*, to be distributed among the threads based on *type*. The behavior of chunks assignments depend on schedule *type*, which can be *static*, *dynamic*, *guided*, *runtime*, or *auto*. *static* schedule type is the most straightforward one that allows the programmers to statically assign chunks of iterations to the team of threads in a round-robin manner. *dynamic* schedule type allows the programmers to dynamically assign chunks of iterations to the team of threads. Each thread requests a chunk of iterations, operates on that chunk, and continues to request for a next chunk until no more chunks available to work on. *guided* schedule type is similar to the *dynamic* schedule type in a way that it allows the programmers to dynamically assign chunks of iterations to the team of threads. However, a major difference between the two is that the *guided* schedule decreases the size of each chunk over time. The reason for this is that initial chunk sizes should

	be large in order to reduce the overhead due to imbalanced workload and multi-threading. *runtime* schedule type allows the programmers to delegate chunk assignments or scheduling decisions until run time, in which a schedule type would be set using the environment variable OMP_SCHEDULE. *chunk-size* is not allowed to used together with *runtime* type. *auto* schedule type allows the programmers to delegate chunk assignments or scheduling decisions to the target compiler or a target running system. This is the last option if a programmer cannot select a best possible schedule type at the time of parallelizing a serial C program. Selecting a correct schedule type has a major impact on parallel program performance. Specially, *dynamic* and *guided* schedule types are suitable for loops containing imbalanced or unpredictable workloads.

Table 5. Commonly-used Clauses of the "*parallel*" and "*for*" Directives

By hiding all complexities of underlying parallel problem analysis for serial program parallelization, creating an OpenMP program requires software programmers to follow two basic steps. In the first step, the programmers need to be able to analyze and identify portions of serial code that are potential "hot spots" for program parallelizations, which can be task-level or data-level parallelisms. In the second step, the programmers need to properly select and apply OpenMP compiler directives to achieve parallelism.

The programmers have to do this parallel work incrementally, that is, repeatedly apply OpenMP directives and perform tests, until achieving a best possible or acceptable parallel program performance.

Figure 13 is an example of a simple OpenMP program, called dot-product that illustrates the important concepts of OpenMP.

```
int main()
{
    double sum;
    double a[256];
    double b[256];
    int i;
    int n = 256;
    for ( i = 0; i < n; i++ )
    {
        a[i] = i * 0.5;
        b[i] = i * 2.0;
    }

    sum = 0;
    #pragma omp parallel for schedule(static) \
    private(i) shared(a, b) reduction(+:sum)
    for ( i = 0; i < n; i++ )
    {
        sum = sum + a[i] * b[i];
    }
    printf("sum = %f\n", sum);

    return 0;
}
```

Figure 13. An Example of A Simple OpenMP Program

3.2. Overview of Machine Learning

A short but precise definition of machine learning from the book titled *Introduction to Machine Learning* by Ethem Alpaydin [23] is: "Machine learning is programming computers to optimize a performance criterion using example data or past experience." In other words, machine learning is a scientific method to program a computer to solve a given problem from existing sets of data or sample records. Machine learning has been applied to various fields of study, including, but not limited to, artificial intelligence and data mining, resulted in many practical and intelligent machine learning applications found today, such as weather forecasting, Web data filtering, bioinformatics data pattern recognition, robot behavior optimization, sales target prediction, digital image processing, and so on [3, 4, 23, 21].

Machine learning and data mining are overlapping or inter-related to each other. Data mining is a process of exploring data to discover useful information for various technological and scientific decisions [3, 24], in which machine learning is one of major techniques used in the data discovery process. In fact, this writing project is a combination of machine learning and data mining; where prior knowledge or sets of data are OpenMP C program source files. The terms machine learning and data mining are used interchangeably throughout this report. Figure 14 shows the relationship among machine learning and data mining. [3]

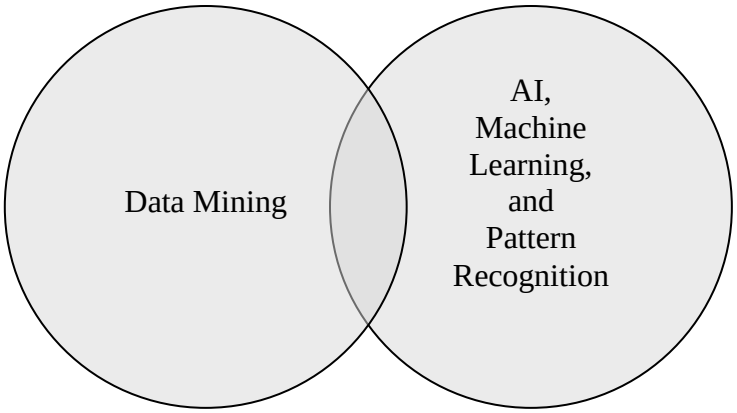


Figure 14. Relationship Among Machine Learning and Data Mining

3.2.1. Concepts and Definitions

Data is everything in machine learning and data mining. An existing set of data represent prior knowledge that is analyzed and learned by a learning algorithm in order to train and build an engine or a model for a specific machine learning task. A sample data set could be a collection of images of a road which would be used to train a robotic vehicle model; another example of a data set could be a collection of Web pages which would be used for target marketing prediction; and many more [21].

Data originally collected in raw format is not so useful to machine learning. In other words, collected data should be preprocessed to filter out noises and/or to extract useful information in order to produce a data set that has direct benefit to a learning algorithm. From this point on, the term "data set" conventionally refers to a ready-to-use set of data, which should have been preprocessed.

A **data set** contains a collection of data objects, which are usually referred to as data records or data instances [3, 4]. Data sets are usually stored either in flat files or in a database system. Each data record is composed of a number of fields where each field is

an **attribute** representing a property of that record. There are four different attribute types: nominal, ordinal, interval, and ratio. Direct definitions of these attribute types are extracted from the book titled *Introduction to Data Mining* by Pang-Ning Tan [3] and are summarized in Table 6.

Attribute type		Description
Categorical (Qualitative)	Nominal	The values of a nominal attribute are just different names; i.e., nominal values provide only enough information to distinguish one object from another. Comparison operations: (=, ≠)
	Ordinal	The values of an ordinal attribute provide enough information to order objects. Comparison operations: (<, >)
Numeric (Quantitative)	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. Comparison operations: (+, -)
	Ratio	For ratio variables, both differences and ratios are meaningful. Comparison operations: (*, /)

Table 6. Summary of Four Different Attribute Types [3]

Data records that are used to train a machine learning model for a specific data mining task are called **training records** or **training instances**. Table 7 contains an example training set for a tax defaulted borrower detection problem.

ID	Refund	Marital Status	Taxable Income	Defaulted Borrower
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Table 7. Sample Training Instances of Tax Defaulted Borrower Problem [3]

Machine learning tasks are categorized into two groups: **predictive** tasks and **descriptive** tasks [3, 21]. Predictive tasks include classification and regression, whereas descriptive tasks include association and clustering. The objective of predictive tasks is to predict the value of a specific attribute of a new data instance based on the availability of other attribute values in the data instance. On the other hand, the objective of descriptive tasks is to discover interesting patterns or relationships in the data. Each task is briefly described below, according to [3, 23, 21], to present enough information to better understand the machine learning task selected for this writing project.

Classification model – The task of this model is to classify a new data record into one of several predefined categorical values of an attribute. This attribute is called the output or target attribute, and its nominal values are called **classes**. An example of a classification problem could be a prediction of risk levels of banking customers. The model, often called the classifier, would learn all provided past information about the

customers, including three known classes of risk levels, such as low-risk, medium-risk, and high-risk. The classifier is now able to predict or classify a new customer into one of the three classes of risk levels based on corresponding information obtained from the new customer. The training data set of this example can be visualized in Figure 15, where each square corresponds to a training instance that belongs to each of the three pre-defined classes, low-risk (‘-‘), medium-risk (‘o’), and high-risk (‘+’). For simplicity, only two input attributes, debt and income, are presented.

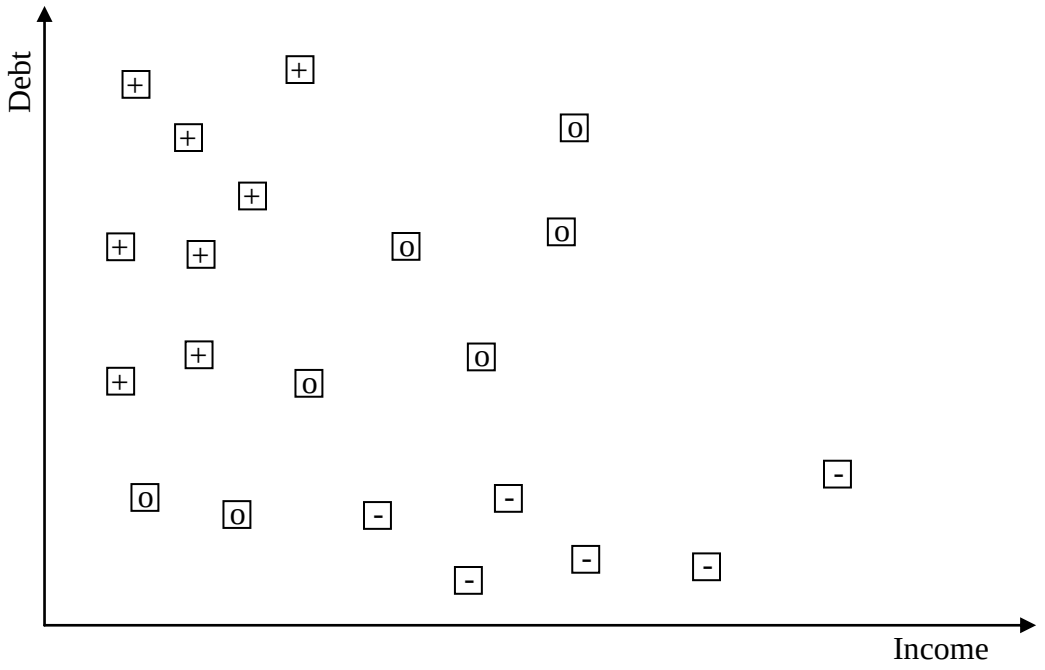


Figure 15. Sample Training Data of the Classification Problem Example

Regression model – The objective of this model is to predict a real value of the target attribute of a new unknown data instance. Since the output attribute value is a number, a continuous value, it is called the regression model. An example of a regression problem could be a prediction of used car prices. Each training data instance for the

model might have a corresponding set of attributes about a used car, such as make, model, year, mileage, and a known price. For simplicity, let X be the input attribute, "Mileage" and Y be the output or target attribute "Price." The training data set of this regression example can be visualized in Figure 16.

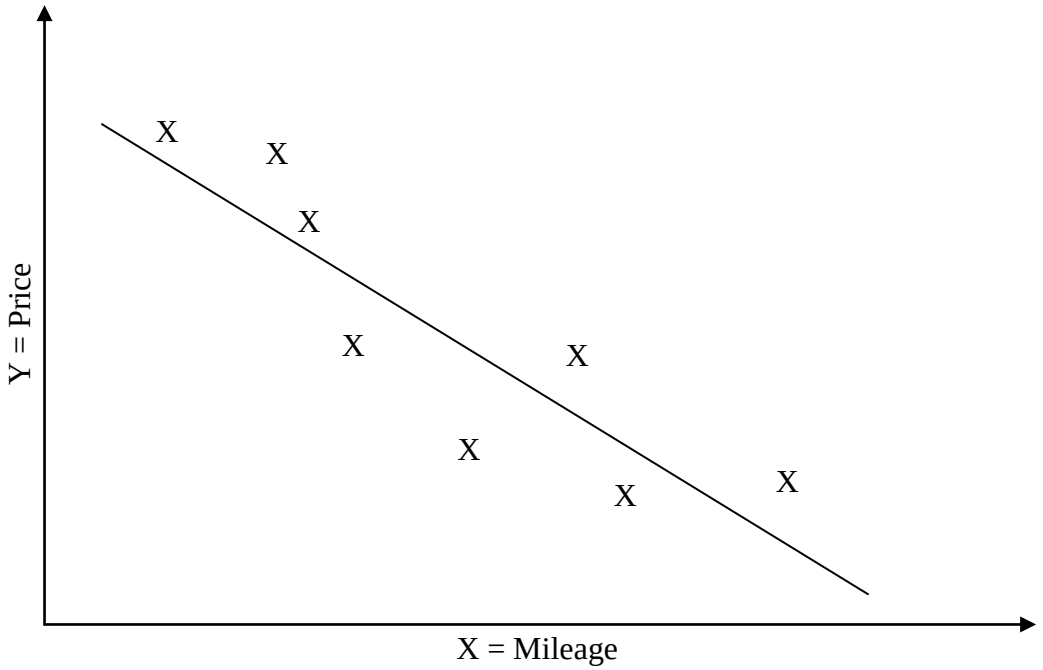


Figure 16. Sample Training Data of the Regression Problem Example

Association and clustering models – These two descriptive tasks are not the scope of this project, so they are briefly described here. An association task aims to discover or to extract certain patterns of associated features in provided data sets, whereas a clustering task is to group related or similar data items into clusters for various information analyses. Figure 17, a snapshot from [3], efficiently visualizes the difference between these two descriptive data mining tasks.

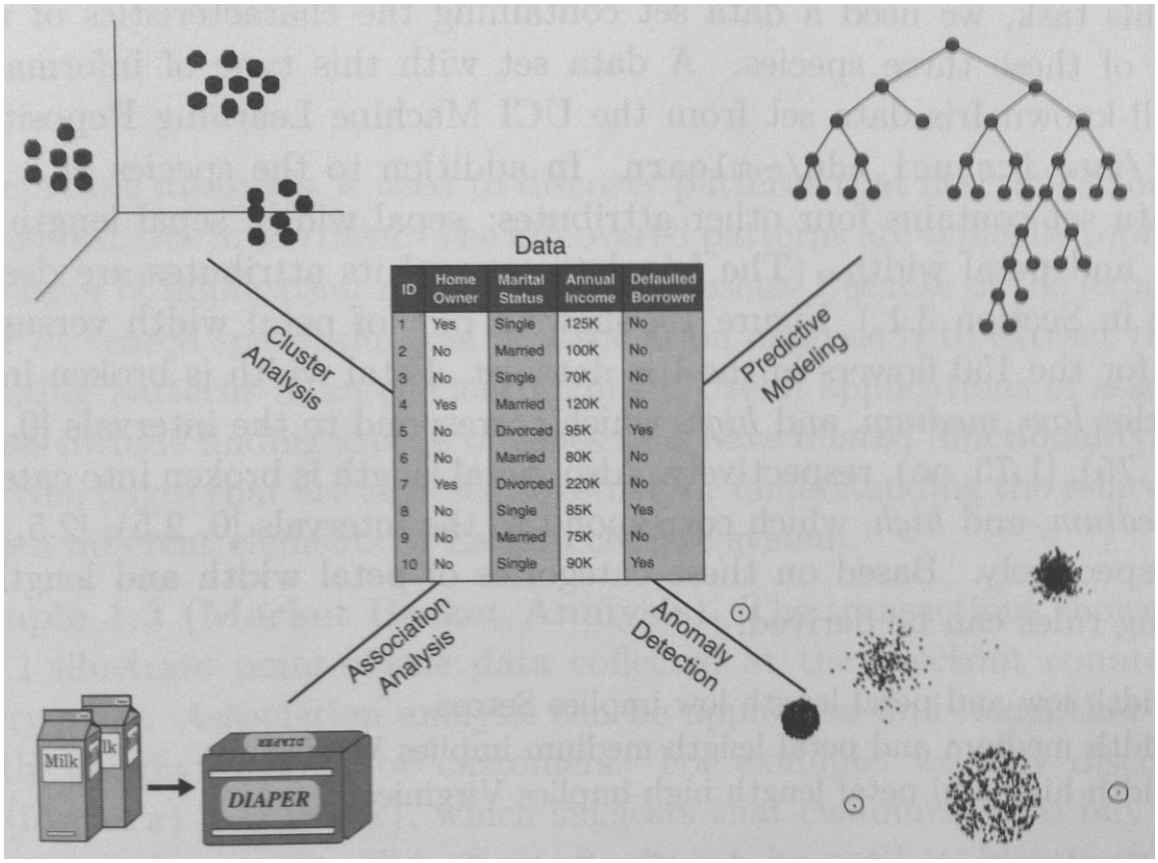


Figure 17. Core Data Mining Tasks [3]

Another important concept in machine learning is the difference between **supervised** learning and **unsupervised** learning. In supervised learning, both input attributes and the correct corresponding output or target attribute of each record are given in a data set. Thus, the purpose of supervised learning is to train a machine learning model using data sets with known correct outputs, the model being supervised, so that the resulting model can be applied to a new record to derive a best possible output or result. In unsupervised learning, the data contains only input attributes, or we have only input data. The task of a machine learning model is to find or discover patterns or relationships among the input data items for various purposes in different fields of study.

3.2.2. End-to-End Process

This section describes a high-level overview of an end-to-end process in data mining. Details of the entire process are dedicated to the Design and Implementation section which presents the process with actual data and explanation. A data mining process consists of a sequence of steps, which can be generalized in Figure 18. The process is similar to Knowledge Discovery in Database (KDD) [3].

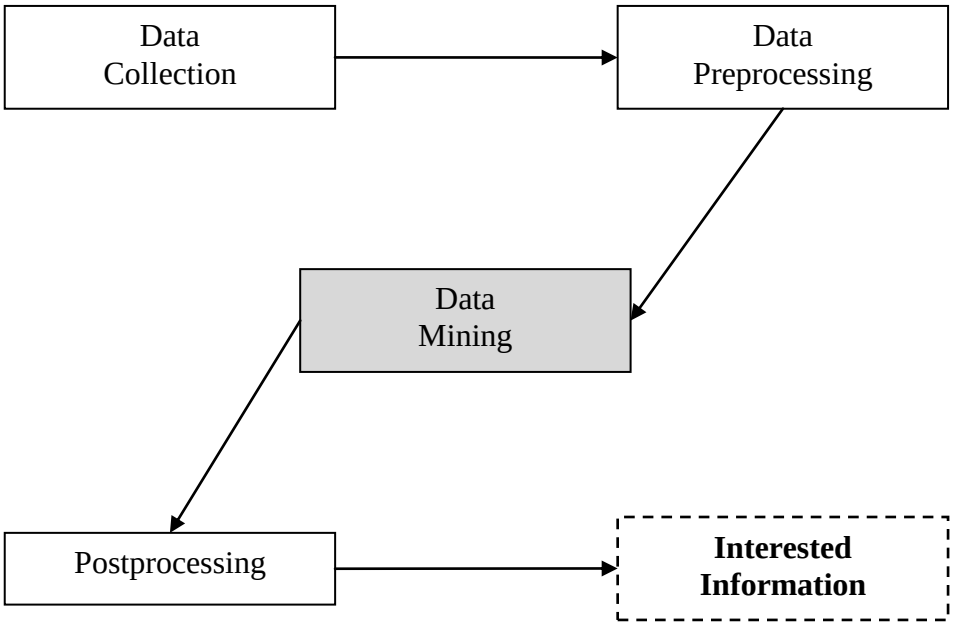


Figure 18. General Process of Data Mining

Data collection is the first important step in data mining or machine learning process. At this step, interested data is collected to solve a given problem. Quality of collected data is important because it is everything that a computing system has or is based upon to build a best possible corresponding machine learning model to solve the problem. Depending on the nature of the problem being solved, the data could be

systematically constructed from natural information saved over time in a database or from various statistical data sources. In general, the various algorithms that are used should tolerate quality problems, which are unavoidable in naturally collected data [3]. However, this is not the case in this writing project because interested data is systematically analyzed and constructed, resulted in high quality data being used for the project.

Data preprocessing is the next important step that involves one or more preprocessing phases, such as data transformation, data filtering, and/or data features extraction. Collected data in the first step is usually called raw data, which might not be suitable to train and build a target machine learning model. Therefore, preprocessing phases are needed to build proper training data sets that are ready for model training and system building.

Data mining is obviously a major step in the entire process where all the intelligence is made. Ready-to-use data sets provided from the previous step are used to train and build a machine learning model for a particular data mining task to solve the problem. A data mining task can be carried out using one of the machine learning techniques, such as decision tree classifiers, rule-based classifiers, instance-based classifiers, support vector machines, and many more [3]. Each technique can be implemented via a variety of machine learning algorithms, including, but not limited to, Hunt's algorithm, sequential recovering algorithm using learn-one-rule function, k-nearest neighbor algorithm, and so on. After the model is built and tested, it is ready to carry out the data mining task.

Postprocessing is the step where the results produced from applying the model in previous step are processed, if it is necessary, to generate meaningful or interested information. This concludes a general data mining process. Interested information now can be used for various scientific and/or technological decisions.

3.2.3. Selected Techniques and Algorithms

There are numerous machine learning techniques. This section covers only two important classification techniques whose resulting models are usually called decision tree classifier and instance-based classifier. The decision tree classifier is presented here as an introduction to the classification task to describe the basic concepts about classification in machine learning and data mining. The instance-based classifier is the learning model chosen for this writing project due to nature of its benefits.

The decision tree classifier:

This classification technique is simple but widely used to solve various data mining problems. In order to explain how a decision tree works, for simplicity, let us consider a weather problem with the sample data set presented in Table 8, where the target attribute is "Play." This weather data set is directly extracted from [4] and contains a small number of training records, which is good for illustration purposes.

Outlook	Humidity	Windy	Play
sunny	high	false	**no**
sunny	high	true	**no**
overcast	high	false	**yes**
rainy	high	false	**yes**
rainy	normal	false	**yes**

rainy	normal	true	**no**
overcast	normal	true	**yes**
sunny	high	false	**no**
sunny	normal	false	**yes**
rainy	normal	false	**yes**
sunny	normal	true	**yes**
overcast	high	true	**yes**
overcast	normal	false	**yes**
rainy	high	true	**no**

Table 8. The Weather Problem Data [4]

From the sample weather training data set, the goal of a decision tree classifier is to learn and build a corresponding decision tree representation or abstraction. To classify a new unknown instance, the classifier walks its decision tree with the input attributes until it reaches a leaf node, which is the target class that the instance belongs to. So, the only question left is how to construct a decision tree with a given training data set. There are "many existing decision tree induction algorithms, including ID3, C4.5, and CART." [3] This paper describes a simple divide-and-conquer [4] algorithm called Hunt's algorithm [3].

The algorithm can be expressed verbally and recursively as follows: start with an attribute ***t*** as a root node. If all the records under ***t*** belong to the same class y_t, then ***t*** becomes a leaf node labeled class y_t. If all the records under ***t*** belong to more than one classes, then split the records into subsets based on attribute values of ***t***; this is also called attribute test condition. This process now can be repeated and recursively applied to each subset of the records.

By applying the algorithm to the weather problem data set, a resulting decision tree is presented in Figure 19.

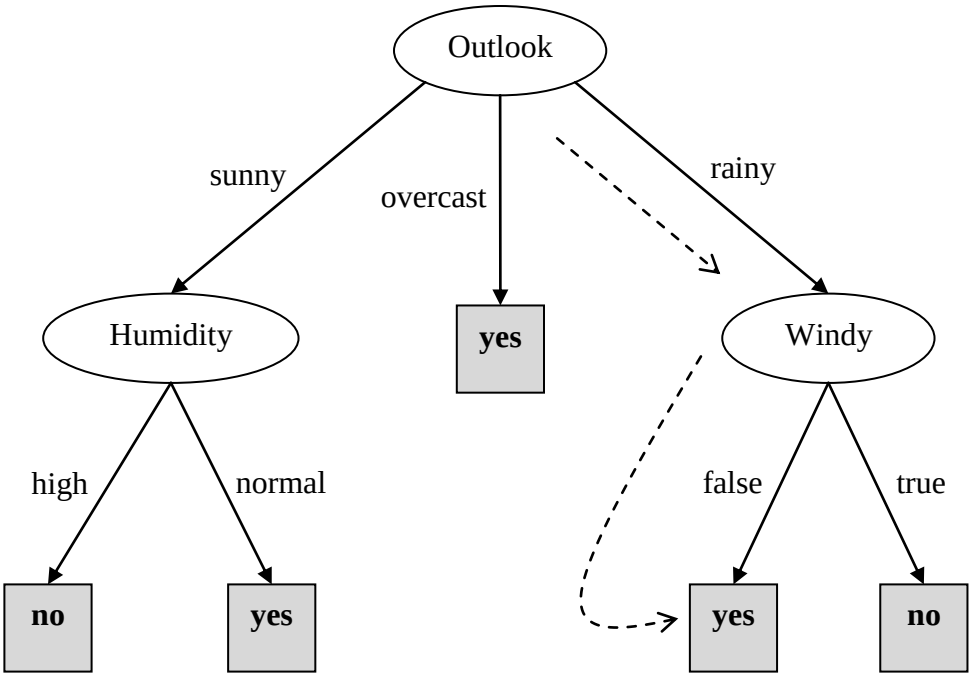


Figure 19. A Decision Tree for the Weather Classification Problem

Given a sample new unknown instance of the weather problem in Table 9, by walking the resulting tree in the figure above, the target class label can be determined via dotted arrows. The record would be classified as **yes**.

Outlook	Humidity	Windy	Play
rainy	normal	false	**???**

Table 9. A New Unknown Instance of the Weather Problem

The instance-based classifier:

The decision tree presented above is an example of **eager learners** because it is designed to immediately learn and build a model from available training records. Instance-based, however, presents an opposite approach. It delays the learning process until it needs to classify a new unknown instance or a test sample record [3, 4]; therefore, it is so called a **lazy learner**.

The idea behind instance-based classifier is that, as the name implies, it classifies a new unknown instance based on available training instances. Thus, knowledge of an instance-based classifier comes directly from the set of training records themselves. The classifier does not have to generalize the training set into some kind of representation or abstraction. The process of building an instance-based model can be thought to be transparent. The model building process is employed via a mechanism that saves or memorizes the training instances along with an associated distance-metric function to compute distances between each training instance and a new test instance. In the simplest form, this new test instance would be classified in the same class of the closest training instance. The number of closest training instances can be 1, 2, 3, …, k; therefore, the most commonly-used algorithm in designing an instance-based classifier is so called the k-nearest neighbor classification algorithm [3].

The **k-nearest neighbor algorithm** can be expressed verbally as follows: Let D be a set of training instances, and compute distance between the test instance and each of the training instances. Then select ***k*** closest training instances to the test instance to create a list of the nearest-neighbors. The test instance is then classified based on the majority

class label found in the list. Figure 20 shows the k-nearest neighbors of a test instance, where k = 1, 2, 3.

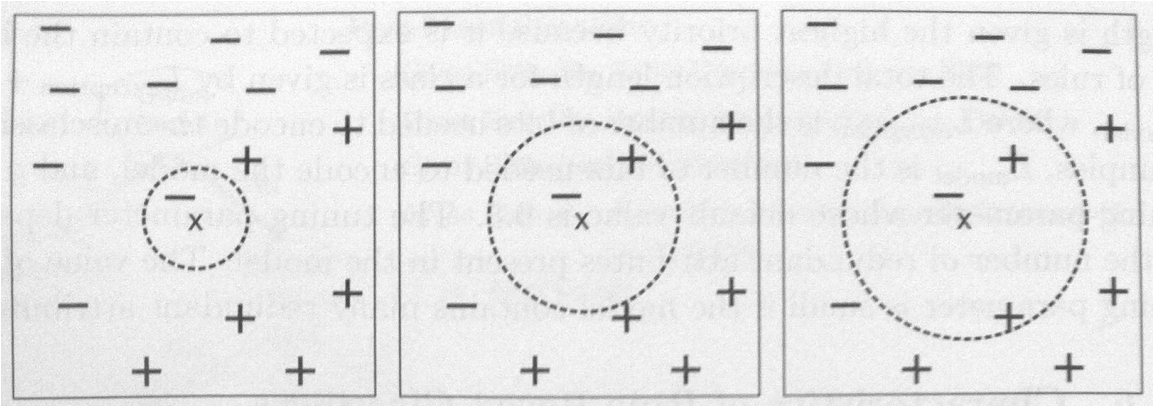


Figure 20. The 1-, 2-, and 3-Nearest Neighbors of an Instance [3]

The only question left is how to compute the distance between any given two instances. The most common method is to use the Euclidean distance formula: Let ***d*** be the distance between two points ***x*** and ***y*** in ***k*** dimensional space. The formula is defined in Figure 21. In relation to the k-nearest neighbors algorithm, ***d*** is the distance between two instances, ***x*** and ***y***, where x_k and y_k are k^{th} attributes of ***x*** and ***y***, respectively.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2},$$

Figure 21. The Euclidean Distance Formula [3]

3.3. Concept of Compiler Design

Just like multi-core OpenMP programming and machine learning, compiler design is another advanced subject that contributes to the inspiration, core design, implementation, and, eventually, completion of this writing project.

A compiler is a translator [5] that translates a program written in a high-level programming language, such as C, C++, or Java, into a low-level machine language that the target machine understands and executes. At a highest level, a compiler can be illustrated in Figure 22.

Figure 22. A High-level View of A Compiler

Compilers are everywhere in programming world and play a vital role in software development. One can imagine that behind a successful programming project, either a high-level application or an embedded system, the core contribution of this extremely important special tool, the compiler, should not be taken for granted [5].

Details in depth of an end-to-end compiler design process is not the purpose of this literature review. The following subsections present fundamental but essential concepts of a compiler design framework that practically applies to the design and implementation of this writing project.

3.3.1. General Framework of a Typical Compiler

At a high-level overview, a typical compiler begins its work by scanning a program written in the source language to construct tokens according to the language's lexical specification. The compiler then parses the sequences of tokens to build the corresponding parse tree, also called Abstract Syntax Tree (AST), according to the language's syntactic specification or grammar. The AST is then processed to produce executable target machine code, which is also sometimes called binary code or object code. As a result, a compiler can be practically broken down into three different parts or phases [5]: the front-end, the intermediate tier, and the back-end, as illustrated in Figure 23. Details of each phase are described in each of the next three subsections.

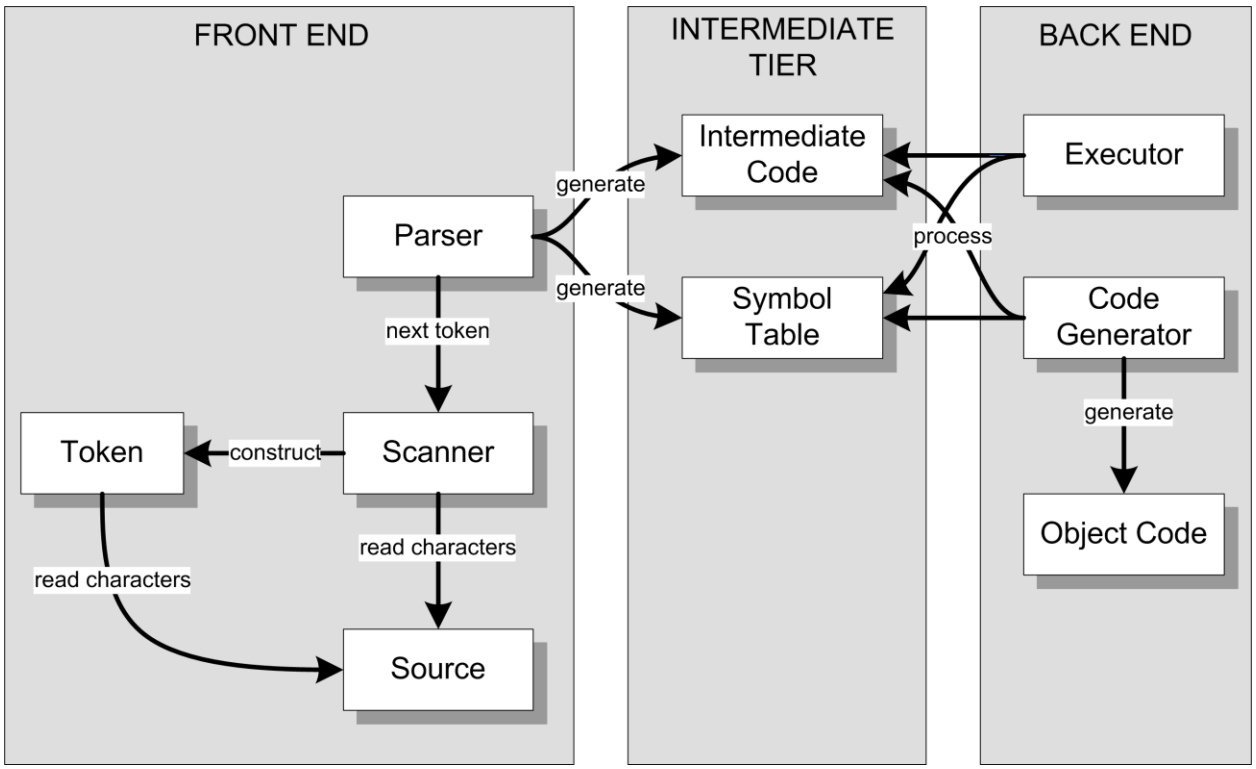


Figure 23. General Framework of a Typical Compiler [5]

3.3.2. The Front-End

The front-end is source language-specific. It must know the programming language grammar, which is a set of rules that specifies the syntax for the language. The source code of a programming language is composed of a variety of language elements called tokens. A token is simply a string of one or more characters that contributes to the language symbol, which can be categorized as an identifier, a number, a character, or any reserved word, and so on. Table 10 gives an example of tokens encountered in the C language statement:

```
int x = 5;
```

Token	Token Type
int	reserved word
x	identifier
=	assignment operator
5	number
;	end of statement

Table 10. Some C Language Tokens

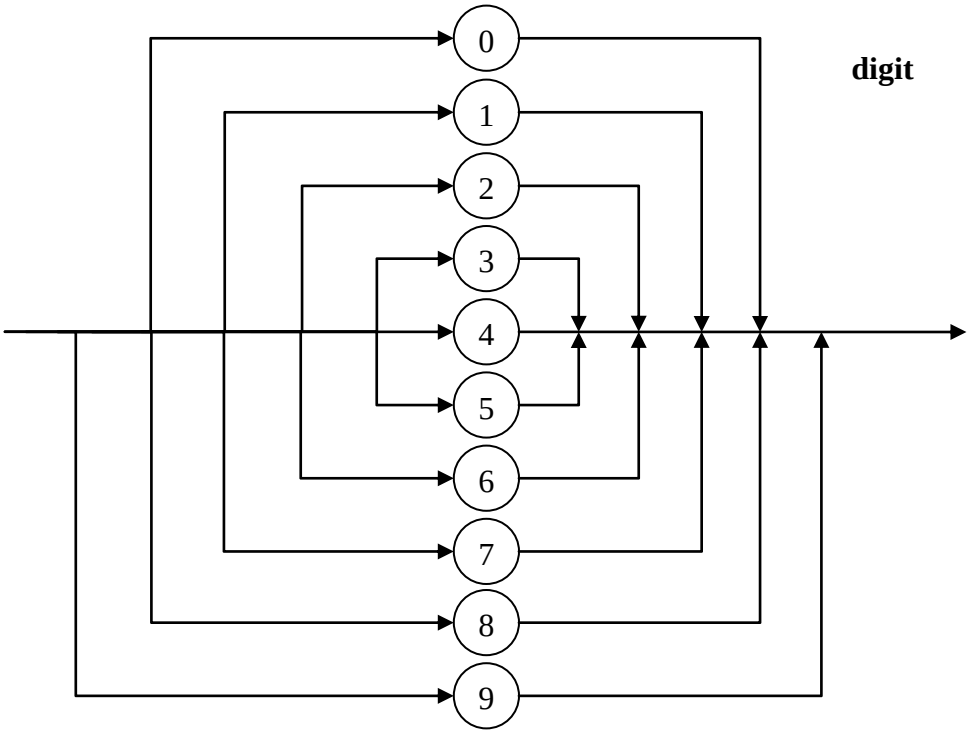
Two major components of the front-end are scanner and parser. The job of a **scanner** is to sequentially process the source code one character at a time, according to the language lexical specification, in order to construct tokens for a corresponding parser [5, 35]. The scanner is also called a lexical analyzer [5] or a tokenizer [38].

According to [5], the scanning algorithm of a scanner can be expressed verbally as follows: The scanner skips any blank characters until it encounters the first character that is non-blank. This current character signals what the next token should be, and it

becomes the first character of that token. The scanner then constructs or extracts the token by consuming all the characters that belong to that token. The next current character is then determined and the scanning process repeats again. Thus, a result of a scanning process is a sequence of tokens that are ready to be processed by a corresponding parser.

The second important component of the front-end is the parser. The **parser** understands the grammar of the source language. It receives sequences of tokens from the scanner and makes sure that the tokens are constructed correctly according to the language syntactic specification; hence the parser is also called syntactic analyzer. In that sense, the parser parses the source program according to the grammar and translates the results into intermediate code along with associated symbol tables. Intermediate code and symbol table are the two components representing the next phase in compiling process, the intermediate tier.

The grammar of a programming language can be specified using graphical syntax diagrams or it can be represented using Backus-Naur Form (BNF). Figure 24 shows an example of a tiny grammar of a digit expressed both as a syntax diagram and in BNF.



<digit> **::=** "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Figure 24. Syntax Diagram and BNF of Digit Grammar

In general, compiler designers do not write a compiler's scanner and parser from scratch [5, 38]. They use a special tool called a compiler-compiler to write compilers. One of the most popular compiler-compilers is the JavaCC, which is also the tool chosen for this project. It is presented in the next section in greater details.

3.3.3. The Intermediate Tier

The symbol table and intermediate code are the two components of this phase, the intermediate tier. Intermediate code is also called intermediate representation, the result

generated from the front-end phase. Both the intermediate representation and the symbol table are portable and source language-independent [5]. Together they encapsulate all the source program information needed by the back-end for various processing purposes.

The **symbol table** is a storage structure used to store any useful information found about the source program elements during the parsing process. It is commonly implemented as a stack data structure to allow the compiler to access the information at different scope levels. One example is to use the symbol table to store information about an identifier [5], including, but not limited to, name of the identifier, type of the identifier, location in the source code that it is declared and defined, and so on. Figure 25 is directly extracted from [5] to show structure of the symbol table. Collectively, we can have multiple symbol tables pushed onto a symbol table stack to provide different access-scope levels.

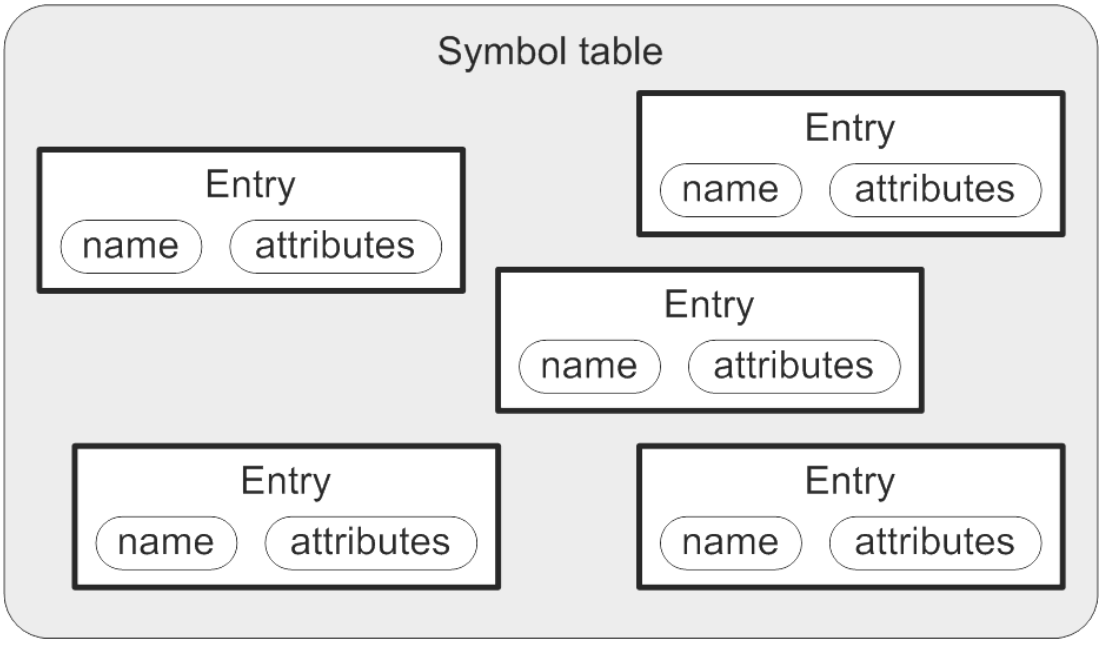


Figure 25. The Symbol Table Structure [5]

The **intermediate code** represents a new form of the source program which is portable and source language-independent. This new form is usually a tree structure, commonly known as parse tree or abstract syntax tree (AST) [5, 38]. Back-end components, covered in the next subsection, freely process the AST for many different purposes, including, but not limited to, program execution, program translation, features extraction, and so on. Figure 26 shows an example of a simple assignment expression statement and its corresponding parse tree. By walking this tree, a back-end component is able to perform necessary calculations for the expression.

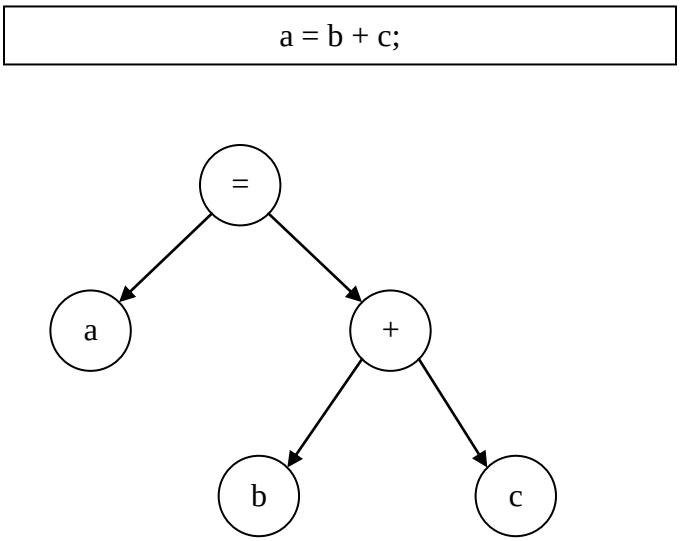


Figure 26. An Expression and Its Corresponding Parse Tree (AST)

3.3.4. The Back-End

A major component of the back-end can be either an executor or a code generator for an interpreter or a compiler, respectively [5]. The executor processes the symbol tables and the parse tree to carry out the program execution. The code generator processes the

symbol tables and the parse tree in order to translate the source program into a target program which can execute or run on a target machine.

Importantly, the back-end does not limit itself as an executor or a code generator. It can be any processing engine to carry out all kinds of specialized computing tasks. The resulting AST can be used for many different purposes. This project uses the parse tree to extract *for* loops features to perform OpenMP auto-parallelization.

4. System Design and Implementation

This major section describes the design and implementation of the proposed system, which includes various phases. First, the scope and design goals of the system are described. Then, the new approach to design and implement this auto-parallelization system is discussed. Next, development tools and software architecture are explained. Finally, implementation of packages and classes are presented in details.

4.1. Purpose of the System and Design Goals

Due to the fact that software programmers have to explicitly produce parallel programs in order to fully take advantage of the potential computing power of the underlying multi-core platform [6], this writing project presents a new way to help programmers and/or compiler designers to automatically parallelize serial C programs at the source code level using the OpenMP APIs. Thus, the main purpose of the system is to prove that the chosen approach, presented in section 4.2, actually works.

Multi-core programming using OpenMP provides a wide range of OpenMP compiler directives to apply to the shared-memory parallel programming model [1, 2]. Furthermore, loops generally consume all or most of the execution time of a running program. Therefore, the system is reasonably and practically designed and implemented focusing only on *for* loop constructs while successfully meeting the design goals set forth below.

The system should be portable, scalable, extensible, reliable, and usable [43]. The system is implemented in Java and uses OpenMP as a parallel mechanism. Therefore, it directly and mutually inherits **portable** and **scalable** features of Java and OpenMP. The

system can run on multiple platforms with Java Virtual Machine (JVM) installed. At the same time, resulted OpenMP programs can be compiled and executed on different multi-core machines.

The system is also **scalable**, **extensible**, and **reliable** because it is implemented via machine learning and compiler-based approach. The system can be incrementally trained with new training instances over time via machine learning. Specifically, the chosen instance-based technique allows the system to scale with small or large number of training records. At the same time, the system processes entire C source files using a C compiler design technique, which results in extensibility and reliability. It captures all C language elements and builds a corresponding AST which is portable and language-independent [5] that can be used to extract many other program features, besides *for* loops, for automatic parallelization purposes. As a result, one can easily extend the system capability to automatically parallelize serial C programs beyond loop constructs from the already-in-place AST.

Last but not least, the system is also **usable** simply because it is implemented beyond original goal of research purpose on the way to becoming an actual product. The resulting system provides a user-friendly Graphical User Interface (GUI) and is effectively able to apply auto-parallelism to actual serial C programs.

4.2. The Machine Learning and Compiler-based Approach

The goal of this new approach is to automatically apply the OpenMP "*for*" directive to *for* loops found in serial C source files to achieve parallelism. Recall from previous Background section that the OpenMP "*for*" directive has various clauses to properly control and parallelize *for* loops.

4.2.1. Machine Learning Approach

The objective of this approach is to predict the OpenMP schedule type *static*, *dynamic*, *guided*, *runtime*, or *auto* for the *for* loop construct (review Table 5). A schedule type cannot be automatically determined from a mathematical formula. Instead, it should be "learned" from those previously-used in OpenMP C source files by a machine learning model. It will apply the learned "knowledge" to a new *for* loop to determine the best possible schedule type for that loop.

Specifically, an instance-based, also referred to as case-based reasoning, machine learning classification technique, implemented using k-nearest neighbors algorithm, has been selected to build a corresponding classifier for this OpenMP schedule type classification problem. The k-nearest neighbors algorithm is an obvious choice because it is well-known to be used with the instance-based technique. From the general end-to-end data mining process presented in section 3, an in-depth process of implementing this project-specific instance-based classifier is specialized as illustrated in Figure 27.

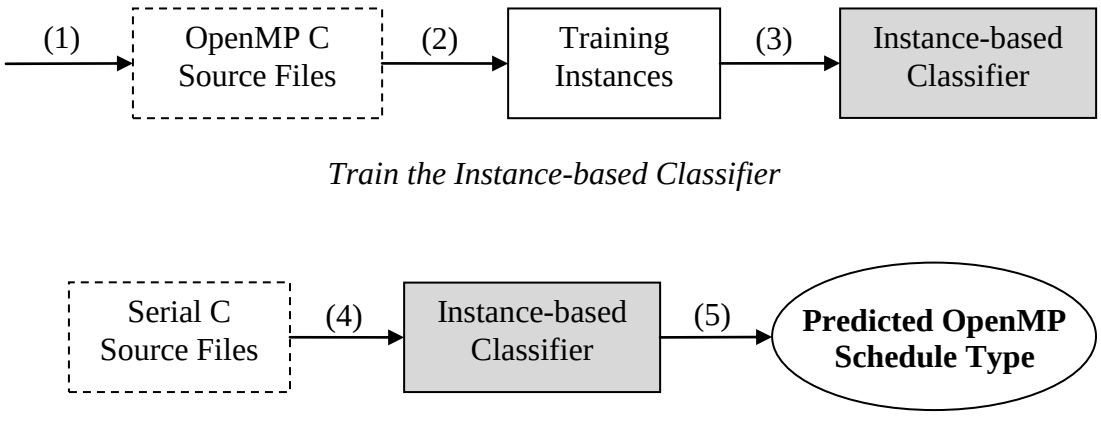


Figure 27. End-to-End Process of the Machine Learning Approach

(1) Data collection:

Pre-parallelized C source files using OpenMP are systematically collected in order to ensure having good training samples. They are directly collected from real examples provided in the textbooks or on Web site of a company, such as Intel. They are also methodically constructed from knowledge learned from [1, 2, 32]. These OpenMP files are collectively composed into four files, namely *omp_training1.c*, *omp_training2.c*, *omp_training3.c*, and *omp_training4.c*. These files can be found in Appendix B. The benefit of implementing instance-based classifier is that it can start with a small set of training records [3, 4, 28]. These four training files contribute a total of nine training instances, *for* loops with various features that are pre-parallelized using OpenMP with appropriate schedule types. These training records are different from each other to represent a wide set of training features. This significantly helps the classifier make better and higher quality decisions.

(2) Features extraction:

This training preprocessing step is to extract representative features of each *for* loop in order to properly build a relevant set of training instances. Below is the list of loop features that are interested to the application:

- Nested level of the loop
- How many nesting levels the loop has
- Number of iterations of the loop
- Number of operations carried out in the loop
- How many children the loop has

- How many threads are sharing the work in the loop
- Number of arrays being used in the loop
- Total number of variables being used in the loop
- What is the state of each variable state, i.e., read-only or modified

Figure 28 is an example output of features that are systematically extracted from a *for* loop. By using compiler design technique, which results in a resourceful AST, features of a *for* loop can be easily and efficiently extracted with the visitor design pattern. The design pattern is described in later section. Each time the system encounters an operation-typed node in a *for* loop, it increments the operation count for that loop. Number of iterations is determined at compile time via variable declarations and definitions. Other features are similarly determined from the tree structure. A different set of feature results in a different data record. The system does not process a method call inside the loop due to the C language library complexity. Method call processing and dynamic calculation of number of iterations are dedicated to future system enhancement.

```
FOR loop at line 28 column 5:
OpenMP directive: #pragma omp parallel for
num_threads(4) schedule(static) default(none)
private(j) shared(n, b, a, c)
    Nested level = 0
    Nesting levels = 0
    Number of iterations = 100000
    Number of operations = 6
    Number of children = 0
    Number of threads = 8
    Number of arrays = 3
    Total number of variables = 5
```

```
    Variable [name=j] is being updated
    Variable [name=n] is accessed only
    Variable [name=b] is accessed only
    Variable [name=a] is accessed only
    Variable [name=c] is accessed only
```

Figure 28. Sample Features Extracted from a *for* Loop

The results of this training preprocessing step are meaningful training instances that are ready to train the machine learning model, the instance-based classifier. Table 11 shows the results of these training instances, where loop complexity is computed based on the number of iterations, the number of operations being conducted inside the loop, and the imbalanced workload condition of the loop. The imbalanced workload condition of an inner loop is when its number of iterations depends on the iteration control parameter of an enclosing outer loop. Figure 29 is an example of an imbalanced workload condition found in a *for* loop.

Threads	Iterations	Nesting-levels	Complexity	Schedule Type
4	100000	0	500000	**static**
4	100000	0	600000	**static**
4	600	1	4320000	**static**
4	1200	1	23040000	**static**
4	5	2	86400000	**dynamic**
4	5	3	414720000000	**guided**
4	100	1	240000	**static**
4	100	0	800	**static**
4	10000	2	80000000000	**dynamic**

Table 11. The Nine OpenMP Schedule Type Training Instances

```
for ( i = 2; i < N; i++ )
    for ( j = 2; j <= i; j++ )
        for ( k = 1; k <= M; k++ )
            b[i][j] += a[i-1][j]/k + a[i+1][j]/k;
```

Figure 29. An Example of Loop Imbalanced Workload Condition

(3) Model training:

This model training step is very straightforward. The scope of this project is not to design a new machine learning algorithm. It makes use of practical machine learning tools introduced from [4], one of which is called Weka. Weka is presented in section 4.3, Development Environment and Tools, with more details. This tool provides a rich set of Java classes and packages to implement various machine learning techniques. Specifically, the system implements its instance-based classifier using the provided class called *weka.classifiers.lazy.IBk*. The available schedule type training instances from the previous step are used train and build the classifier, which is then used to predict an OpenMP schedule type of a new unknown record. To be portable, the resulting training instances are also saved in ARFF format, as presented in Appendix A.

Each training instance in the training set extracted in phase (2) is fed to the model for model training purpose. Figure 30 shows part of the training process. For more details, look at Appendix G: The Project Source Code.

```
/**
 * Adds a new training instance to the data set.
 * @param attributeValues String array of attribute
          values
```

```
 * @param classLabel the class label of the training
          instance
 */
public void addTrainingInstance(String[]
attributeValues, String classLabel) {

    // Attribute set
    Attribute numberOfThreads =
trainingDataSet.attribute("num_threads");
    Attribute numberOfIterations =
trainingDataSet.attribute("num_iterations");
    Attribute nestingLevels =
trainingDataSet.attribute("nesting_levels");
    Attribute complexityUnits =
trainingDataSet.attribute("complexity_units");

    // Instance to be added
    Instance instance = new Instance(5);
    instance.setValue(numberOfThreads,
Integer.parseInt(attributeValues[0]));
    instance.setValue(numberOfIterations,
Integer.parseInt(attributeValues[1]));
    instance.setValue(nestingLevels,
Integer.parseInt(attributeValues[2]));
    instance.setValue(complexityUnits,
Double.parseDouble(attributeValues[3]));
    instance.setDataset(trainingDataSet);
    instance.setClassValue(classLabel);

    // Add the training instance to the data set
    addTrainingInstance(instance);
}
```

Figure 30. Partial Code for Model Training

(4) Model applying:

Similar to model training in phase (3), model applying is very straightforward. For each new unknown record, loop features are extracted as in phase (2) except for the missing class label, which is the schedule type that the model needs to predict. Figure 31 shows code that implements part of the model applying process. For more details, look at Appendix G: The Project Source Code.

```
/**
 * Classifies a new instance.
 * @param instance a new instance to be classified
 * @return predicted class label of this instance
 * @throws Exception an exception that is propagated
           to the caller
 */
public String classifyNewInstance(Instance instance)
throws Exception {

    // The classifier has not been trained yet
    if ( trainingDataSet.numInstances() == 0 ) {
        throw new Exception("Training data set is
empty; no classifier has been built.");
    }

    // Make sure the classifier model is up-to-date
    if ( !modelUpToDate ) {

instanceBasedClassifier.buildClassifier(trainingDataSe
t);
    }
```

```
    // Perform classification, resulted in predicted
label for this instance
    double predictedClass =
instanceBasedClassifier.classifyInstance(instance);
    Attribute classAttribute =
trainingDataSet.classAttribute();
    String classLabel =
classAttribute.value((int)predictedClass);

    // The predicted class label of this instance
    return classLabel;
}
```

Figure 31. Partial Code for Model Applying

(5) Result interpretation:

Recall from the Background section that the result obtained from applying a machine learning model can be used for various technological or scientific decisions. The result sometimes can be used directly. In some other cases, it must be processed or interpreted in order to become useful information to an intended application.

In this project, the result, which is a predicted schedule type, obtained from applying the instance-based classifier in phase (4) can be used directly by our auto-parallel system to automatically generate corresponding OpenMP directive with the schedule type embedded for the target *for* loop construct. Importantly, the system is able to learn from the programs it parallelizes because the auto-generated result also becomes a new training record to train back the classifier. As a result, the system gets smarter the more programs it parallelizes.

Below is a partial code showing part of "*for*" directive construction process. For more details, look at Appendix G: The Project Source Code.

```
// OpenMP directive for the most outer loop
String pragma = "#pragma omp parallel for";
String numThreads = "num_threads(" + numberOfThreads +
")";
String scheduleClause = "schedule(";
String defaultClause = "default(none)";
String privateClause = "private(";
String sharedClause = "shared(";
StringBuffer reductionClauses = new StringBuffer("");

// features/attributes of this nested loop (unknown-
schedule-type instance)
// (number of threads is input from the user or
default value is used)
int numberOfIterations =
mostOuterForLoop.getNumberOfIterations();
int nestingLevels =
mostOuterForLoop.getNestingLevels();
double complexityUnits =
computeComplexity(forLoopTree);
String[] attributeValues = {
String.valueOf(numberOfThreads),

String.valueOf(numberOfIterations),

String.valueOf(nestingLevels),

String.valueOf(complexityUnits) };

// Predict schedule type for this nested loop (unknown
instance)
String scheduleType =
theClassifier.classifyNewInstance(attributeValues);
```

```
// build schedule clause
scheduleClause += scheduleType + ")";

...

//
// compose all clauses into a complete pragma
//
pragma += " " + numThreads;
pragma += " " + scheduleClause;
pragma += " " + defaultClause;
...
pragma += " " + reductionClauses.toString();
```

Figure 32. Partial Code to Construct the "*for*" Directive

4.2.2. Compiler-based Design Approach

This compiler-based design approach is naturally embedded into this project's end-to-end design and implementation. Recall in the general Background section that there are three phases in a compiler design framework; therefore, below are the three corresponding phases of compiler design technique applied to the project.

(1) Front-end:

The project front-end is designed using the compiler-compiler tool called JavaCC. This tool helps to generate a scanner and a parser in the Java language for a given language grammar that is the best fit into the design and implementation of this project. This leaves the project the only, but very important, task to design the grammar for the C language. With information provided from [39, 44], the project C grammar for JavaCC

has been successfully implemented. Appendices E and F contain the C grammar written in BNF and the C grammar written for JavaCC, respectively.

Besides the obvious goal of the grammar to parse C source programs, the project makes use of this compiler design technique is to analyze and extract *for* loops features for OpenMP auto-parallelization. Therefore, the JavaCC C grammar for this project goes beyond the basics It is designed and customized to enable the system to extract all information about *for* loops and their associated variables from the parse tree to properly construct corresponding OpenMP "*for*" directive. Figure 33 shows a partial example of the grammar to allow an extraction of *for* loop related information. Appendix F contains the full set of grammar structures.

```
void forStatement() :
{
    Token t;
    int beginLine = 0;
    int beginColumn = 0;
}
{
    ((t = <FOR>)
    {
        beginLine = token.beginLine;
        beginColumn = token.beginColumn;
    }
    <LEFT_PAREN> [[typeSpecifier()]
expression(NodeType.FOR_STMT_NODE)] <SEMICOLON>
[expression(NodeType.FOR_STMT_NODE)] <SEMICOLON>
[expression(NodeType.FOR_STMT_NODE)] <RIGHT_PAREN>
    statement())
    {
```

```
        jjtThis.setAttribute(NodeKey.BEGIN_LINE, new
Integer(beginLine));
        jjtThis.setAttribute(NodeKey.BEGIN_COLUMN, new
Integer(beginColumn));
        if ( t.specialToken != null ) {

jjtThis.setAttribute(NodeKey.SPECIAL_TOKEN,
t.specialToken.image);
        }
    }
}
...
...
        jjtThis.setAttribute(NodeKey.REDUCTION,
reduction);
        jjtThis.setAttribute(NodeKey.REDUCTION_OP,
reductionOp);
        return isIdentifier;
    }
}
```

Figure 33. Partial Project C Grammar for JavaCC

(2) Intermediate tier:

The symbol table and the symbol table stack in this project are designed to store information about declarations and definitions of functions and variables and more during the parsing process in phase (1). Figure 34 illustrates part of the design of symbol table of the project. Symbol tables are used to:

- Retrieve the numeric value of a variable.
- Filter out and retrieve the set of variables that belong to a particular *for* loop.

- Separate or distinguish an identifier encountered in a *for* loop, whether it is a function name or a variable.

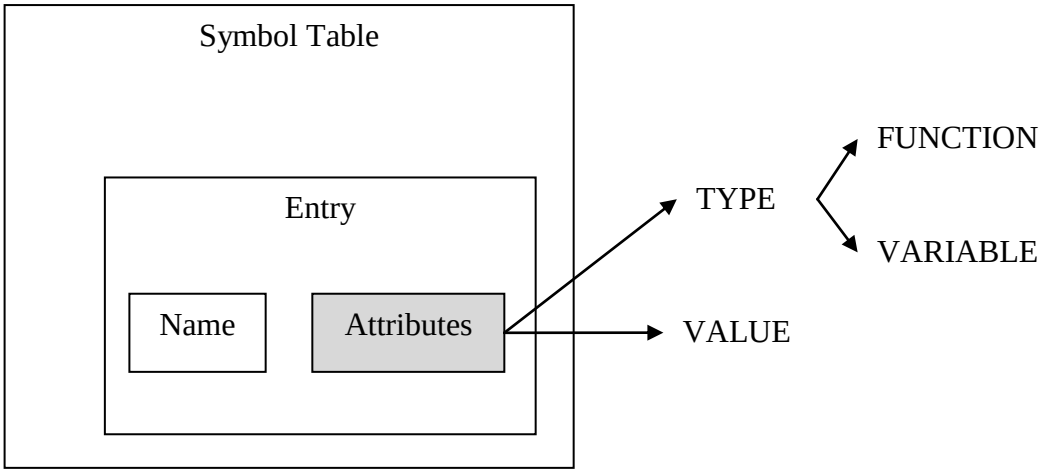


Figure 34. The Symbol Table

JavaCC has a utility called JJTree. It allows the front-end to generate a parse tree for the source program according to the project-specified grammar constructs. Importantly, the resulting parse tree has a built-in feature of the visitor design pattern, which allows the back-end to easily and efficiently process the intermediate information for OpenMP code generation purposes.

(3) Back-end:

The major back-end component of the project design is the **extractor**, called **ForLoopExtractorVisitor**. Its main task is similar to an executor or generator in a sense that it walks the resulting parse tree to obtain the source program information. However, as the name implies, it does not perform any execution or translation, but it rather extract

a variety information about *for* loops and associated variables. The information is used for various logic decisions about "*for*" directive clauses, as indicated below.

The extractor extracts the information from the parse tree and provides the results, which is another tree structure of the extracted *for* loops, to the main component of the program, called the **OpenMPAutoParallelizer**. This auto-parallelizer simply performs a breadth-first tree traversal to carry out its major tasks as follows:

- Train the OpenMP schedule type instance-based classifier via extracted *for* loops features.
- Derive an OpenMP schedule type for the "*for*" directive via the machine learning method.
- Derive other clauses for the "*for*" directive, such as the "*private*" clause, the "*shared*" clause, the "*reduction*" clause, and so on.
- Perform loop dependence analysis so that those loops that are not parallelizable are practically rejected. A common form of a loop-carried dependence is illustrated in the figure below, where a computation of an element in the array "a" at a particular iteration *i* depends on the result of previous *i*-1 iteration. This kind of dependency prevents the loop iterations to be executed in parallel, and it should be rejected by the system.

```
for ( i = 0; i < n; i++ )
{
    a[i] = a[i-1] + b[i];  // a[i] depends on a[i-1]
}
```

Figure 35. An Example of Loop-carried Dependence

Figure 36 is an example showing a partial AST generated from the serial C source file called *serial_matrix_product_1.c*. The tree is composed of various nodes representing structure of the source program. We are interested only in the "forStatement" node. The Figure also shows the features extracted from that node.

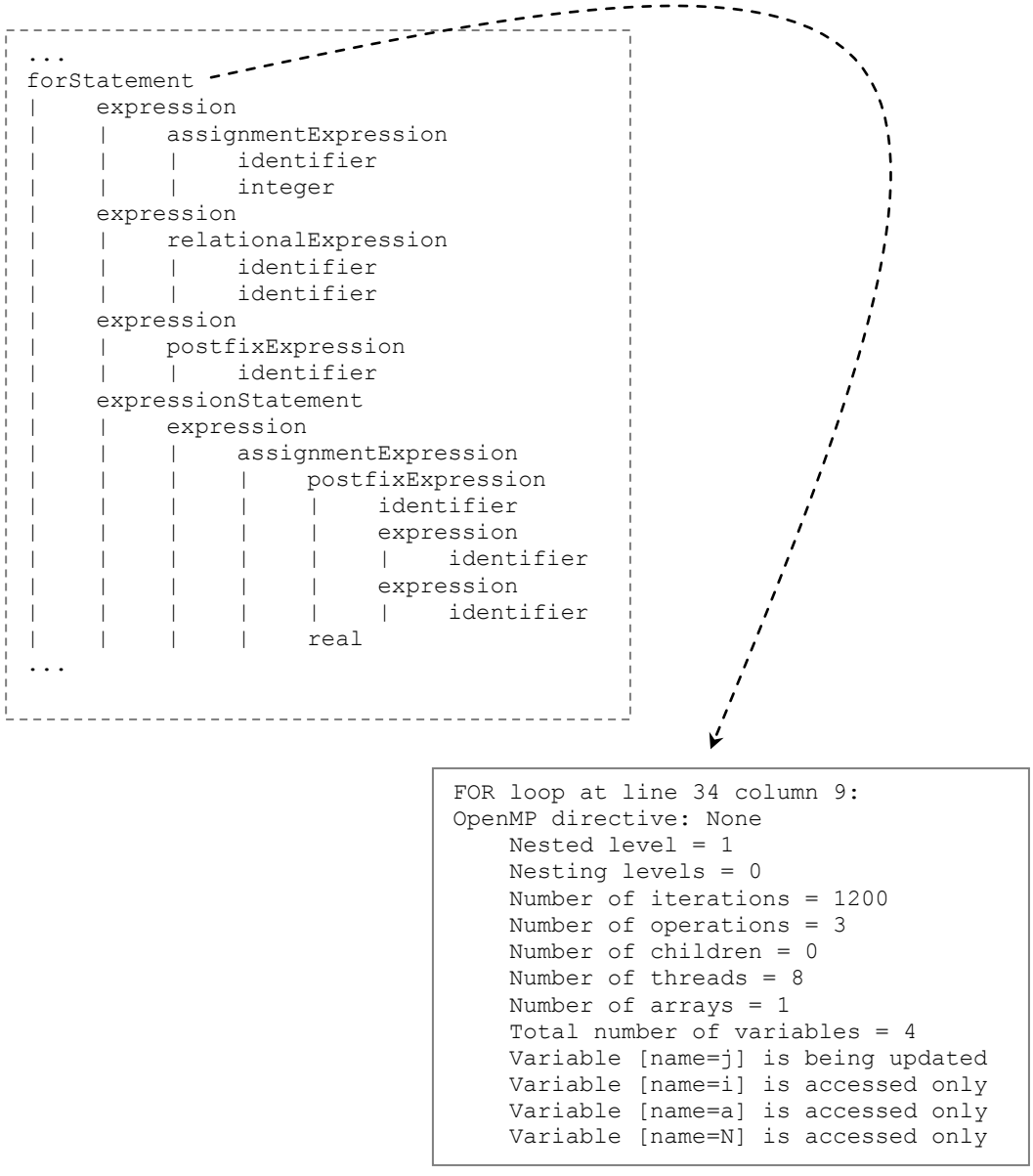


Figure 36. An Example of A Partial AST Generated from *serial_matrix_product_1.c*

4.3. Development Environment and Tools

The project has been developed and tested under both Windows and Linux platforms. Java development environment is the JDK version 1.7.0. Eclipse Java EE IDE has been chosen as the main designing, coding, and debugging tools with various plug-ins installed to serve different design purposes. UML modeling tools include ObjectAid and UMLet. Two major tools that contribute to the heart of the project are the JavaCC [5, 38. 39] and the Weka [41].

The **JavaCC** utility is a compiler-compiler tool that allows compiler designers to design and develop compilers without implement scanners and parsers from scratch. This tool automatically generates corresponding scanner and parser for a given grammar of a programming language. This tool also includes other utilities, such as JJTree and JJDoc. JJTree helps to generate a parse tree, and JJDoc can be used to generate the grammar in BNF documentation.

The **Weka** library provides a rich set of APIs to help software programmers to develop efficient machine learning applications using a wide range of choices in terms of machine learning techniques and algorithms. The programmers are also able to implement various machine learning techniques for their own designs from the available Weka packages and classes. For instance, this project implements the instance-based classifier from the Weka class called *weka.classifiers.lazy.IBk*. The classifier can be customized further with the following options as specified in the Weka online documentation [41]:

- Weight the neighbors by the inverse of their distance.
- Weight the neighbors by 1 – their distance.
- Specify number of nearest neighbors (k value) for the classification.
- Specify a nearest search algorithm to use.

The two core methods of the package used to build the classifier and to classify a new unknown instance are *buildClassifier()* and *classifyInstance()*, respectively. For details of all available configurations and methods of the classification classes, see the Weka Java documentation online [41].

4.4. Software Architecture and Design Patterns

Software architecture [43] of the system comes directly from the indicated purpose of the system and design goals. That is, the system is decomposed into two main components, machine learning (ML) and compiler design (CD). Under CD, the subsystems are decomposed according to the compiler design framework. The front-end, the intermediate tier, and the back-end components. The software architecture of this project is shown in Figure 37.

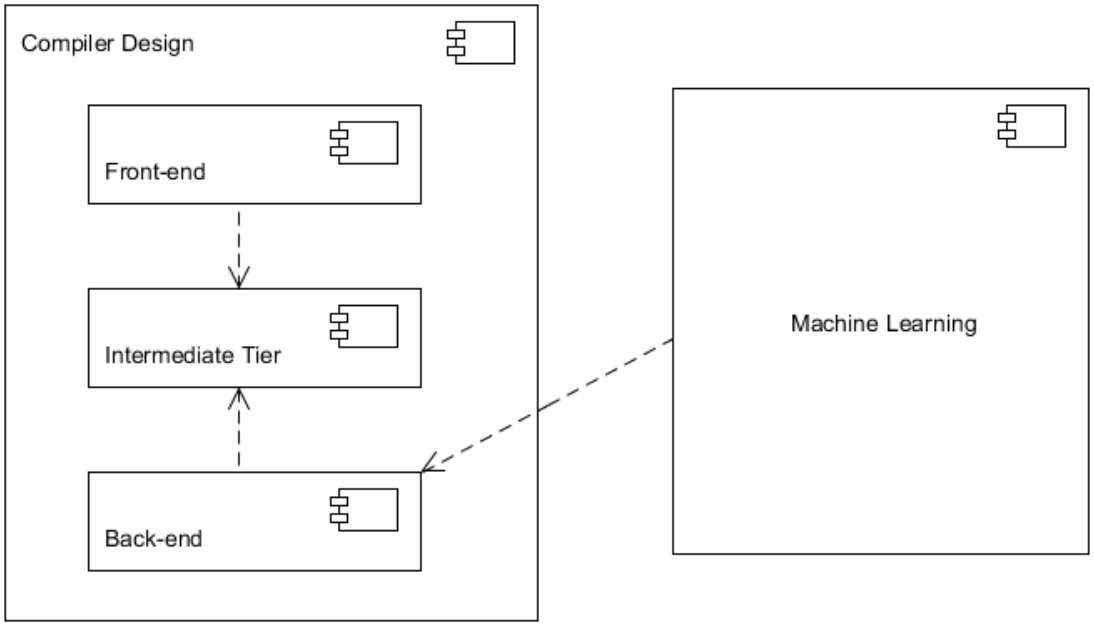


Figure 37. The Project System Software Architecture

Selected **design patterns** include two main design patterns, which are the abstract factory and the visitor design patterns. The abstract factory provides a mechanism to create and manage symbol tables and symbol table entries, as shown in Figure 38. The visitor design pattern provides an efficient way to traverse the abstract syntax tree in order to extract *for* loops features for automatic OpenMP parallelization, as shown in Figures 39.

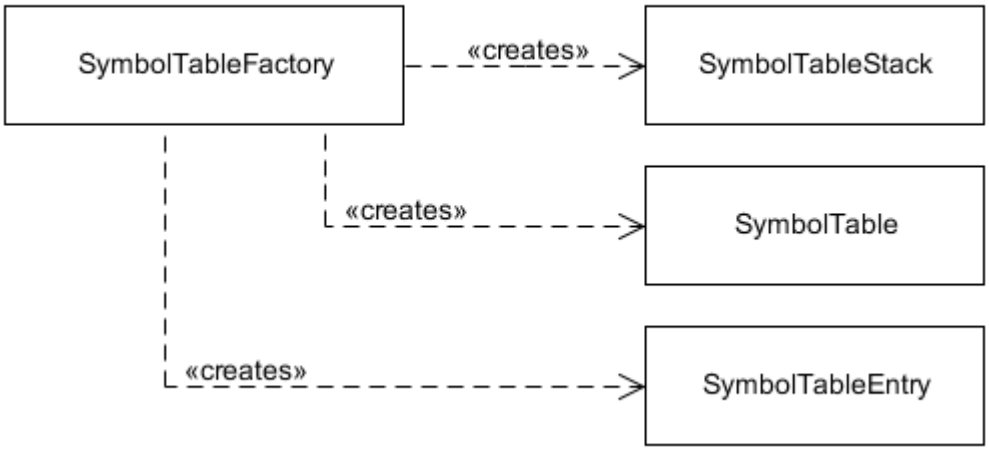


Figure 38. The Abstract Factory Design Pattern

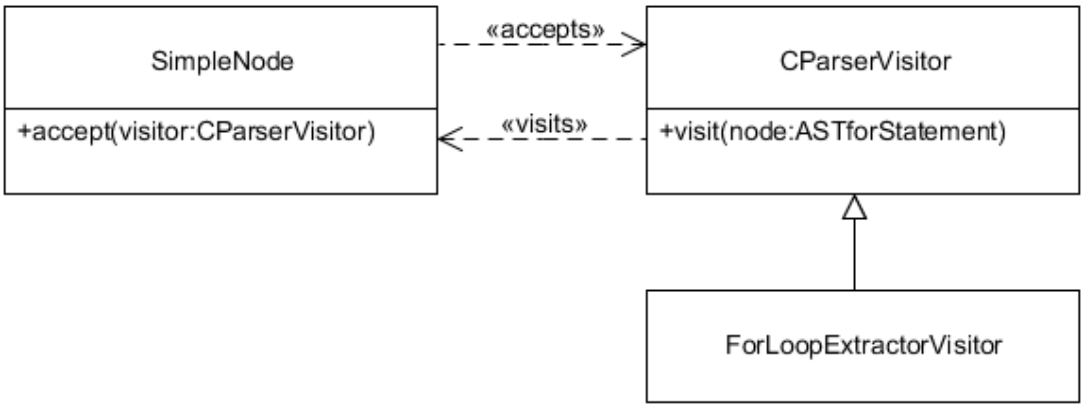


Figure 39. The Visitor Design Pattern

4.5. Implementation: Packages and Classes

The project implementation phase is practically and incrementally implemented one feature at a time, exercising the concept “build on something that works” [5] and Scrum

project backlog [36]. The list in Figure 40 shows the backlog items to implement the system.

```
[ X ]   Lexical analysis

    _x_ scan single-line comments
    _x_ scan multi-line comments
    _x_ scan single-line pre-processors
    _x_ scan multi-line spanning pre-processors
    _x_ scan key words
    _x_ scan primitive data types
    _x_ scan operators

[ X ]   Syntactic analysis

    _x_ parse constants
    _x_ parse identifiers
    _x_ parse primary expressions
    _x_ parse postfix expressions
    _x_ parse unary expressions
    _x_ parse cast expressions
    _x_ parse multiplicative expressions
    _x_ parse additive expressions
    _x_ parse shift expressions
    _x_ parse relational expressions
    _x_ parse equality expressions
    _x_ parse bitwise expressions
    _x_ parse logical expressions
    _x_ parse conditional expressions
    _x_ parse constant expressions
    _x_ parse assignment expressions
    _x_ parse if statements
    _x_ parse switch statements
```

```
    _x_ parse for statements
    _x_ parse while statements
    _x_ parse do/while statements
    _x_ parse labeled statements
    _x_ parse jump statements
    _x_ parse expression statements
    _x_ parse compound statements
    _x_ parse pointers
    _x_ parse specifiers
    _x_ parse qualifiers
    _x_ parse declarations
    _x_ parse function definitions

[ X ]   Construct a parse tree (AST) from a C source
        file

[ X ]   Extract details of each FOR loop node in the
        AST

    _x_ extract OpenMP directives/clauses applied to
        the loop
    _x_ extract current source line/column numbers of
        the loop
    _x_ extract all variables in the loop
    _x_ extract types of each variable
    _x_ extract access modes of each variable

[ X ]   Build features of each FOR loop node in the
        AST

    _x_ Nested Levels                numeric
    _x_ Nesting Levels               numeric
    _x_ Loop complexity              numeric
    _x_ Number of Threads            numeric
```

```
    _x_ Number of Iterations          numeric
    _x_ Number of Operations          numeric
    _x_ Number of Children            numeric
    _x_ Number of Arrays              numeric
    _x_ Number of Variables           numeric
    _x_ Data Dependency               YES/NO

[ X ]   Able to be trained to classify simple sample
        records using the instance-based classifier

[ X ]   Able to be trained to classify a FOR loop node
        into an OpenMP schedule type based on its
        extracted features

[ X ]   Build OpenMP num_threads clause for a FOR loop
        node based on its extracted information

[ X ]   Build OpenMP schedule clause for a FOR loop
        node based on its extracted information

[ X ]   Build OpenMP default clause for a FOR loop
        node based on its extracted information

[ X ]   Build OpenMP private clause for a FOR loop
        node based on its extracted information

[ X ]   Build OpenMP shared clause for a FOR loop node
        based on its extracted information

[ X ]   Build OpenMP reduction clause for a FOR loop
        node based on its extracted information

[ X ]   Output complete OpenMP #pragma... for loop
        construct for each encounter FOR loop node
```

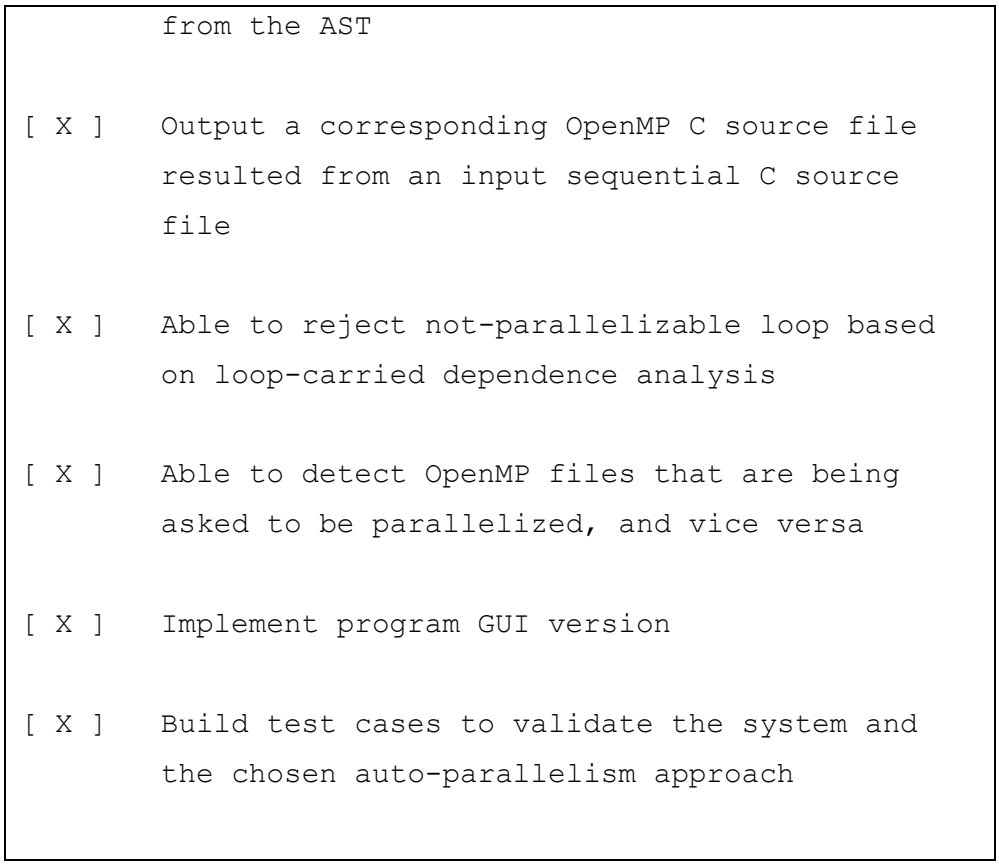


Figure 40. The Simple Scrum Backlog of the Project

For the chosen software architecture, the corresponding packages and major classes are designed and implemented and shown in the UML package and class diagrams in Figures 41-44. For readability, the diagrams show only names of the packages and classes. Refer to Appendix G: The Project Source Code for more details.

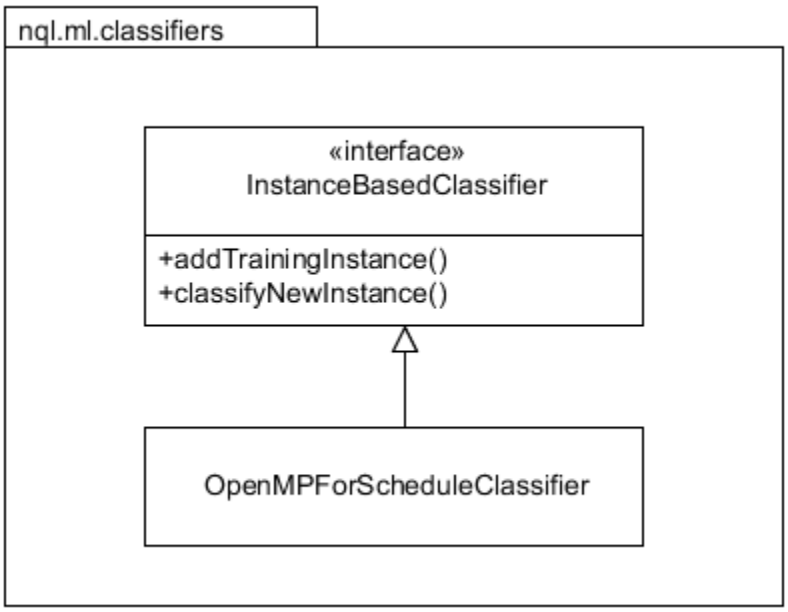


Figure 41. Classes Under *nql.ml.classifiers* Package

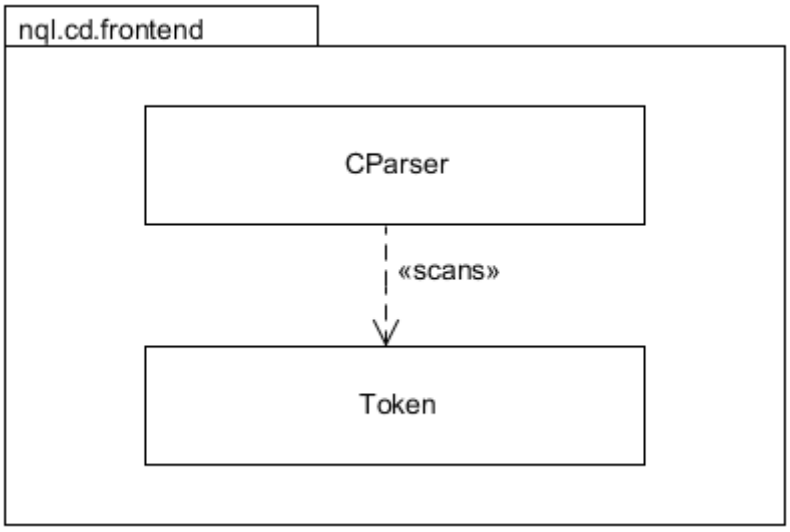


Figure 42. Classes Under *nql.cd.frontend* Package

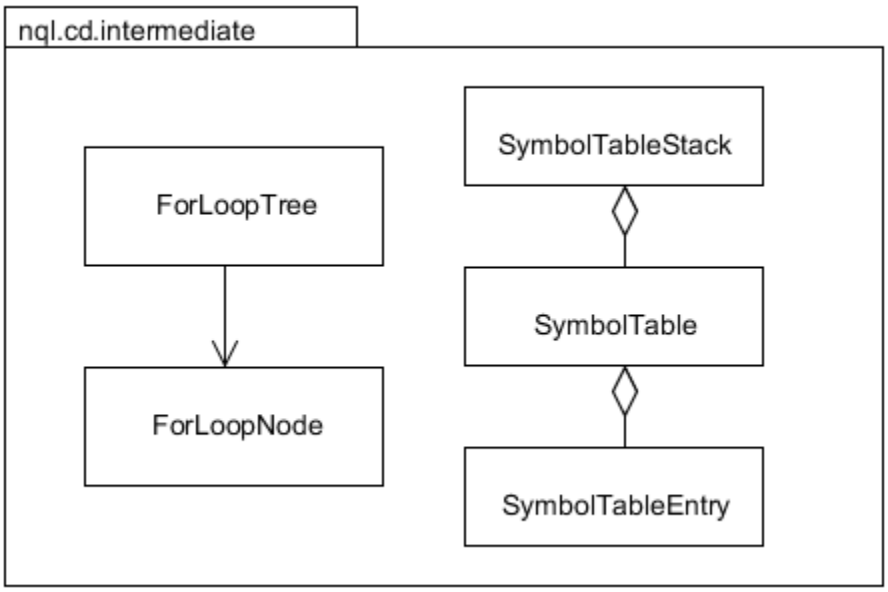


Figure 43. Classes Under *nql.cd.intermediate* Package

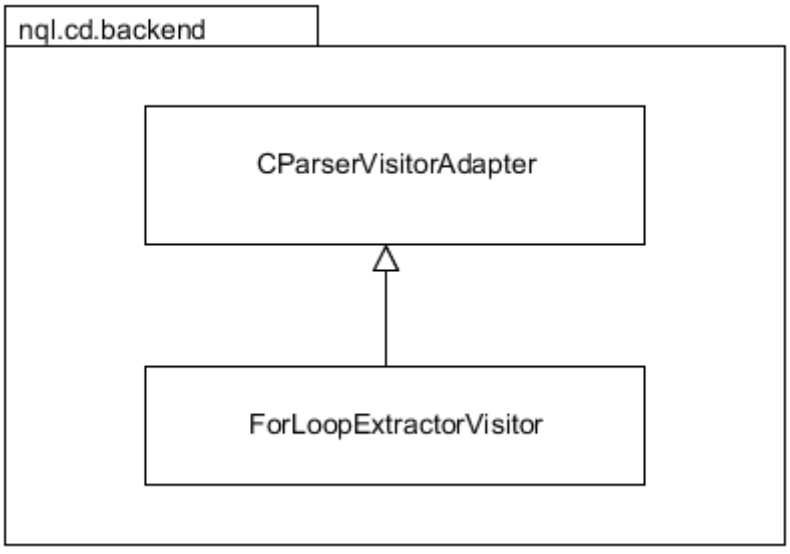


Figure 44. Classes Under *nql.cd.backend* Package

The system program can be run in two different ways: command line version or graphical user interface (GUI) version. Figures 45 and 46 show how to run the system with the command line and a snapshot of the GUI version, respectively.

```
USAGE: program-name (w/o input, GUI version)
       program-name -n num-threads -t input-file-name.c
       program-name -n num-threads -p input-file-name.c [output-file-name.c]
       -n: specifies number of threads to run
       -t: trains the classifier with the input file
       -p: parallelizes the input file and sends the result to the
           output file if it is present; standard output otherwise
```

Figure 45. Program Usage for Command Line Version

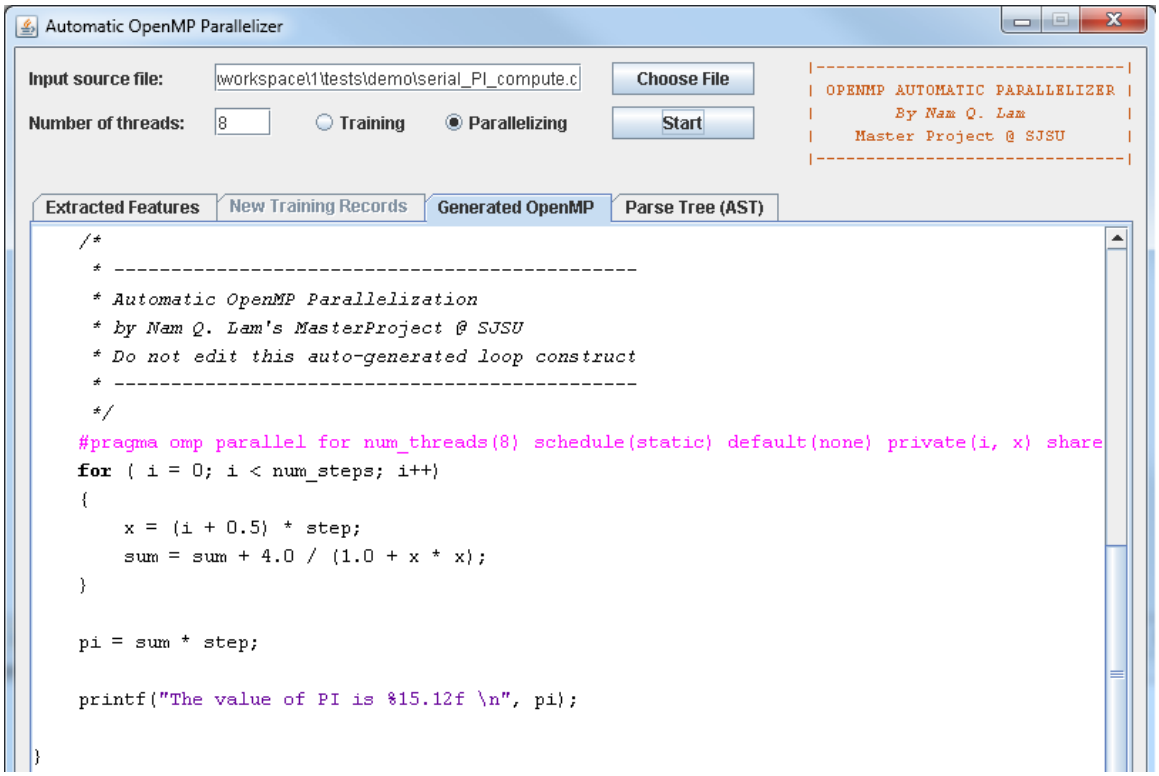


Figure 46. The GUI Version of the Program

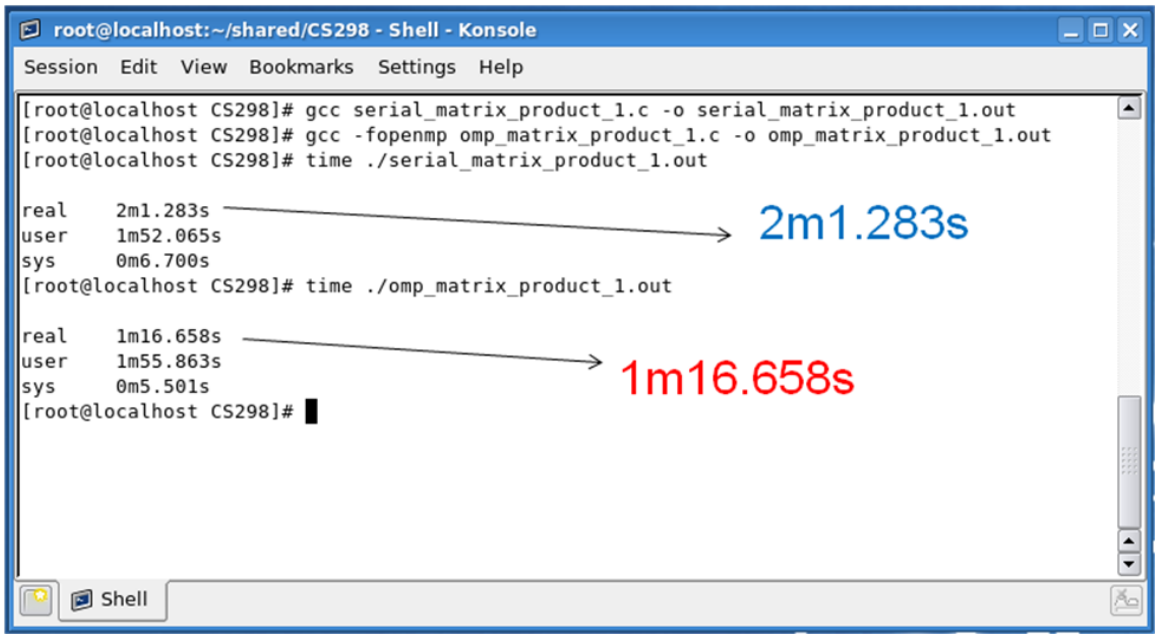
5. Experimental Results

OpenMP and auto-parallel performance are measured using the Linux operating system command called “time.” This performance measuring method is simple, but practical, as recommended by [2]. The command shows three different time measurements called “real,” “user,” and “system” for an execution of a program. Among the three, “real” time, also called wall clock time [6], is the most important measurement because it encapsulates the entire time from start to finish of an execution call. This time measurement is used to analyze this project test results.

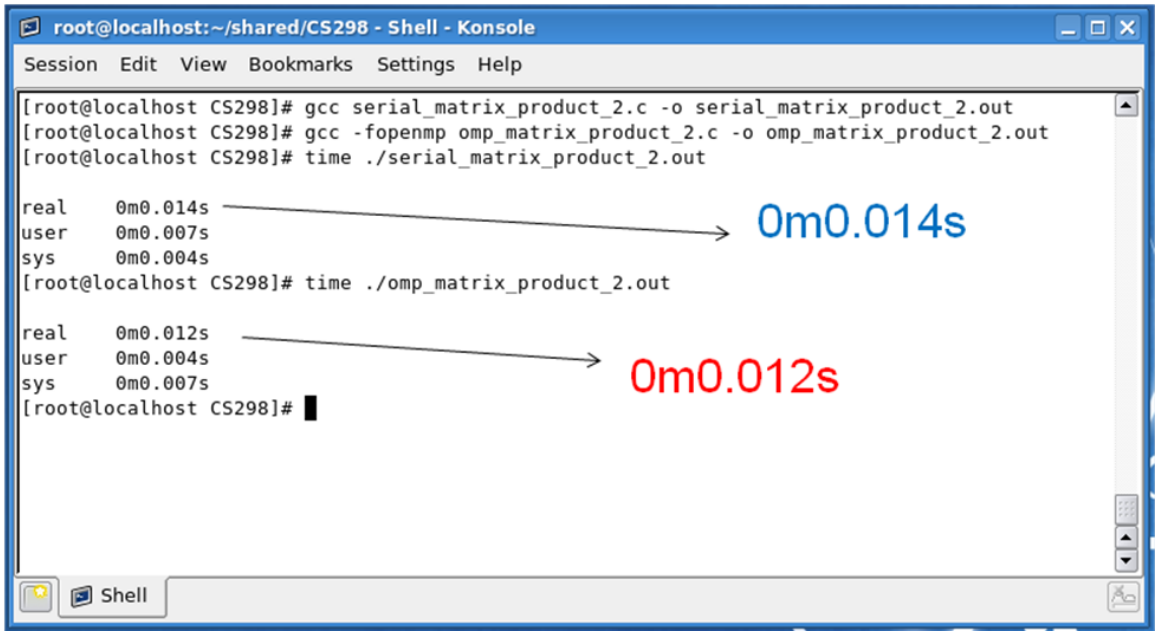
The objective of this section is to present the experimental results in order to prove the system works and to show the benefits of the machine learning and compiler-based approach chosen for this project. Below are snapshots of various test cases conducted on real samples of test files. Refer to Appendix C for the actual serial C program test files. Some cases show significant speed-up via OpenMP parallelization; while some others show not much of differences. This is understandable and acceptable [2] due to the simplicity of some the test programs and also due to the limitation of the system under test, which is a personal computer equipped with an Intel Core 2 Duo for a total of only four cores. An important concept here is that these resulting OpenMP programs are *automatically* parallelized.

Figures 47 and 48 contain charts comparing the real (wall clock) execution times between the programs to effectively visualize the test results. Refer to Appendix D for resulting auto-parallelized OpenMP files. Due to significant differences between the real time values of the test cases, two different charts are given for two different groups of test cases.

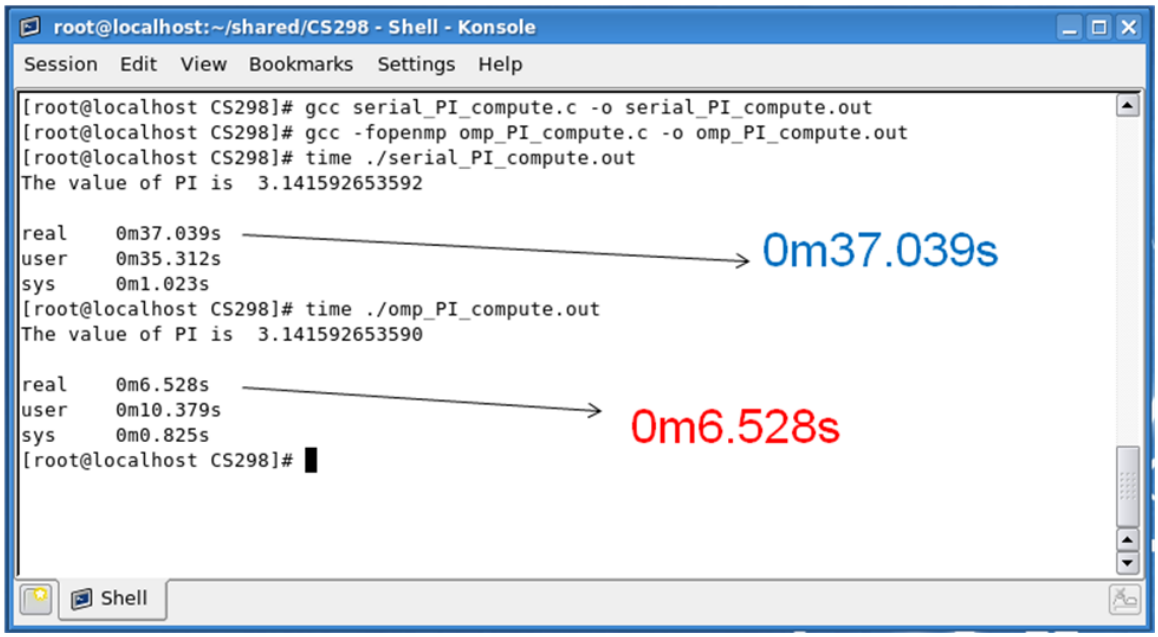
Test Case 1



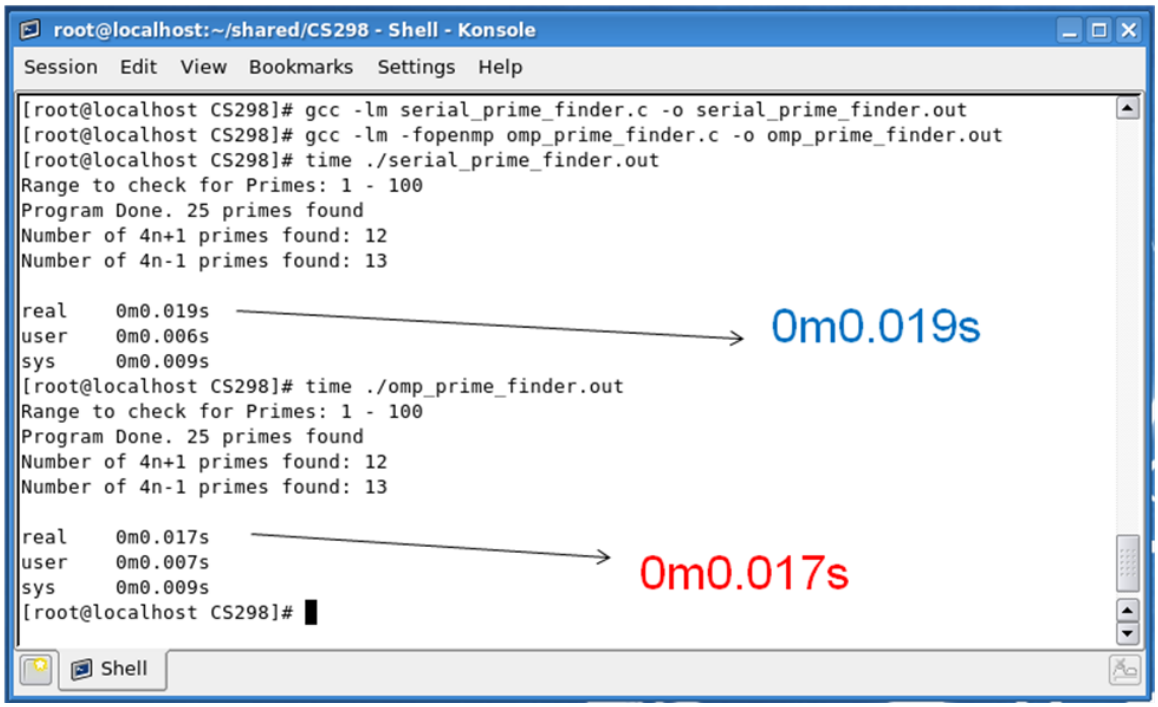
Test Case 2



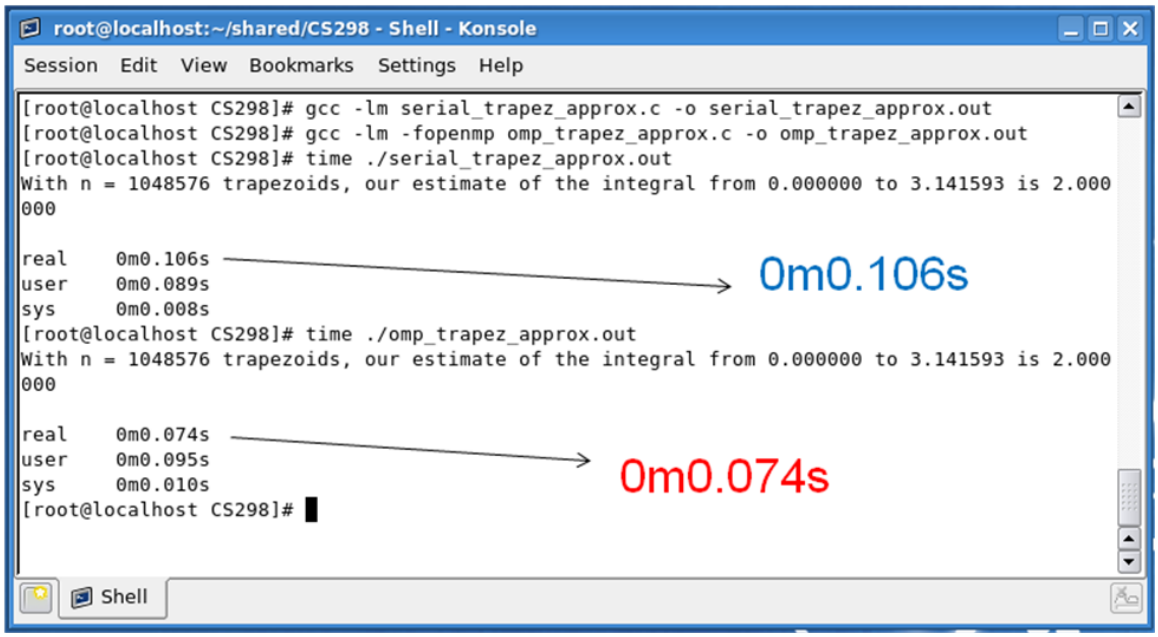
Test Case 3



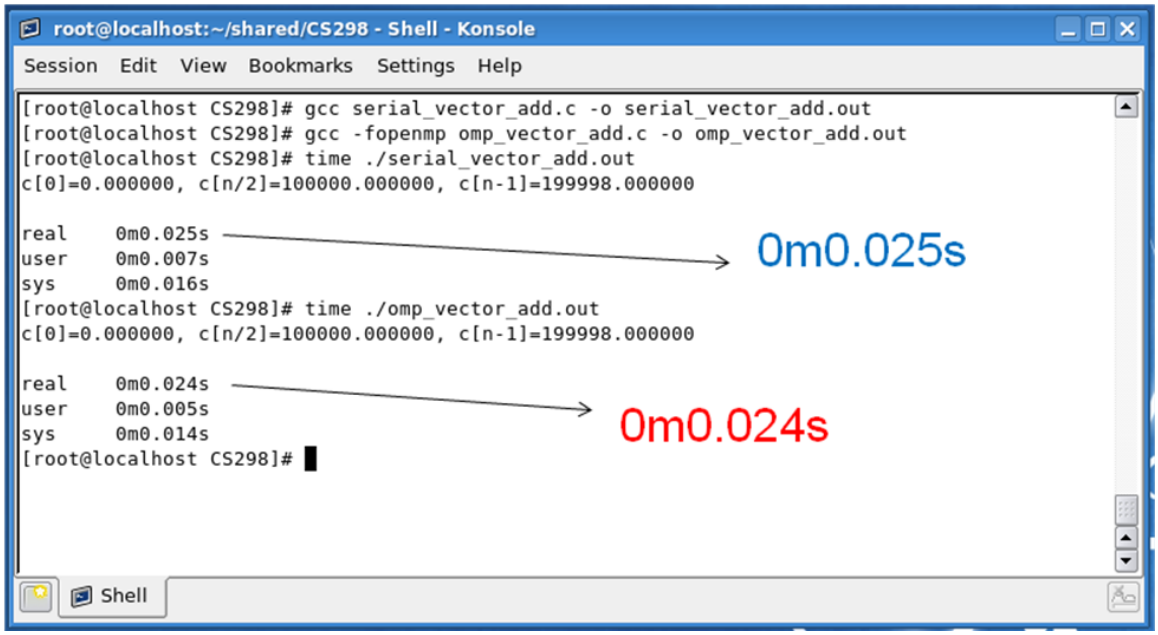
Test Case 4



Test Case 5



Test Case 6



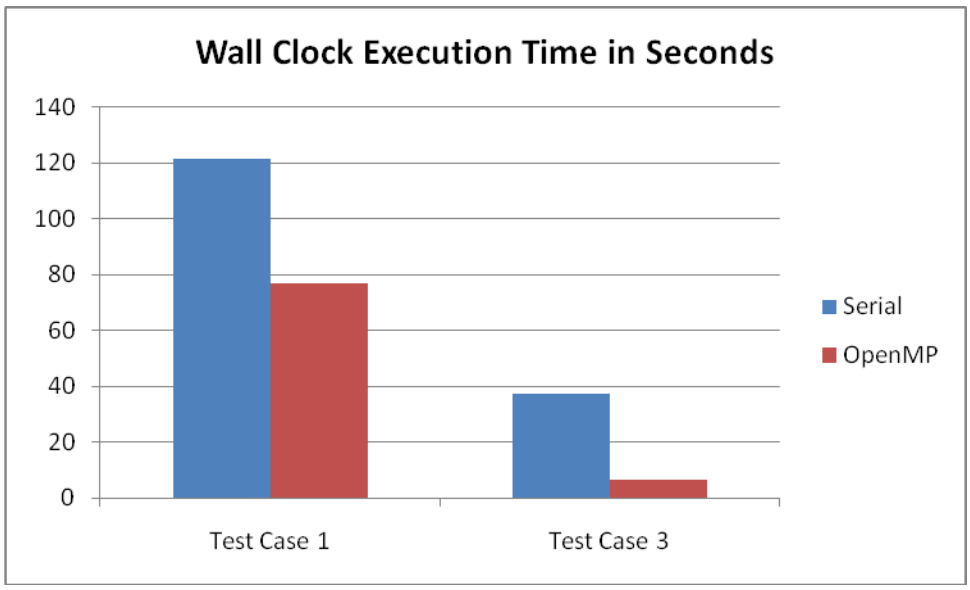


Figure 47. Real Time Chart of Test Cases 1 & 3

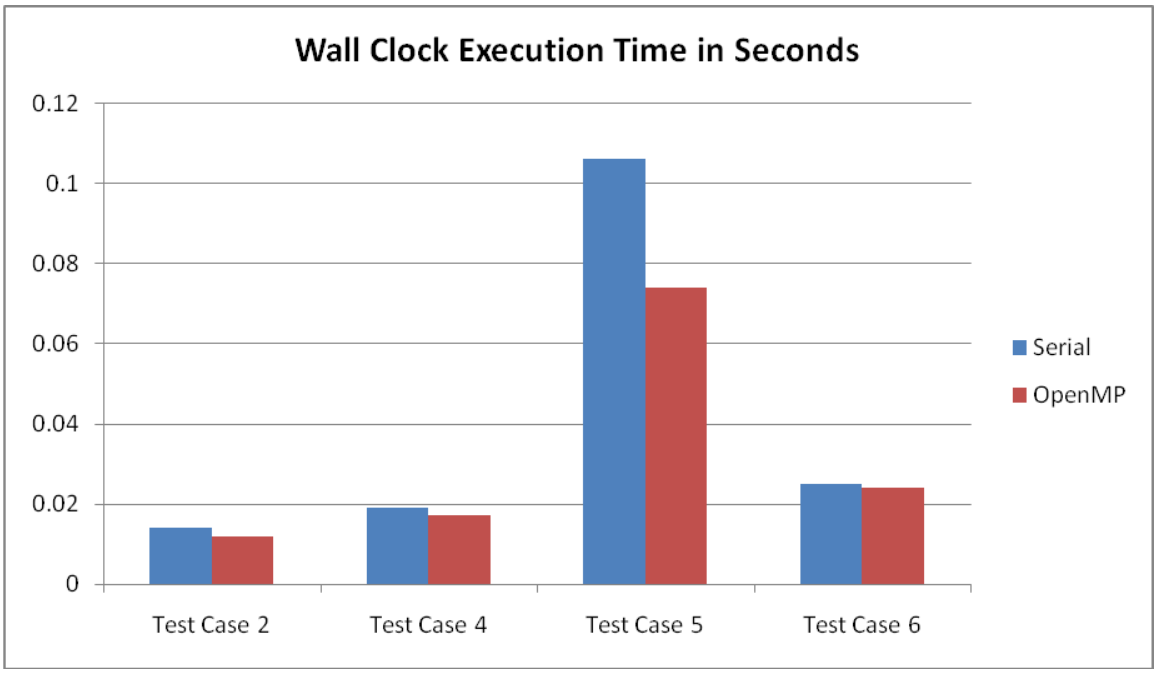


Figure 48. Real Time Chart of Test Cases 2, 4, 5, & 6

Based on the results of the test cases and the resulting chart above, we could see that the system has effectively parallelized the serial C programs into OpenMP C programs. Of course, performance of the OpenMP files inherits from performance of OpenMP itself. The project has achieved its goal, that is, the entire parallelizing process has been done *automatically*. This is the main objective of this writing project.

6. Conclusion and Future Work

In conclusion, this report successfully describes, presents, and proves an advance and practical approach, machine learning and compiler-based design, to implement a novel system to automatically parallelize serial C programs using OpenMP parallel programming model.

This is my original work. It effectively provides a foundation for automatic parallel programming on multi-core architectures at source code level. The resulting system significantly helps software programmers in many ways. The programmers can use the system as a tool to first automatically parallelize a legacy serial C program, and then from the results, the programmers have the opportunity to fine-tune the program to achieve maximum parallel performance. Furthermore, compiler designers gain different benefits from this tool as well; they can integrate it into a modern compiler to achieve automatic parallelism.

Due to achievements in design goals of the project, specially in scalability and extensibility, in the future, the system can be easily extended to fully achieve parallelism beyond *for* loops by implementing other OpenMP work-sharing constructs via the already-in-place compiler-based design framework and the machine learning model.

7. References

[1] S. Akhter and J. Roberts. *Multi-core Programming: Increasing Performance through Software Multi-threading*. Intel Press, 2006.

[2] B. Chapman, G. Jost, and R. Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, 2008.

[3] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2005.

[4] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[5] R. Mak. *Writing Compilers and Interpreters: A Software Engineering Approach*. Wiley, 2009.

[6] R. Chun. *CS 159: Parallel Processing*. San Jose State University, http://www.cs.sjsu.edu/~rchun/, 2010.

[7] J. Abbate. "Getting Small: A Short History of the Personal Computer." Proceedings of the IEEE, Vol. 87, No. 9, pp. 1695-1698, Sep 1999.

[8] A. Apon, R. Buyya, H. Jin, and J. Mache. "Cluster Computing in the Classroom: Topics, Guidelines, and Experiences." First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 476-483, May 2001.

[9] P. Werstein, H. Situ, and Z. Huang. "Load Balancing in a Cluster Computer." Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06), pp. 569-577, Jan 2006.

[10] C. Yang and K. Li. "The Anatomy of a Course in Cluster and Grid Computing." 2005 International Conference on Information Technology: Research and Education, pp. 403-407, Jun 2005.

[11] J. Mache and A. Apon. "Teaching Grid Computing: Topics, Exercises, and Experiences." Proceedings of the IEEE, Vol. 50, No. 1, pp. 3-9, Feb 2007.

[12] Shuai Zhang, Shufen Zhang, X. Chen, and X. Huo. "The Comparison Between Cloud Computing and Grid Computing." 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), pp. V11-72 - V11-75, Oct 2010.

[13] R. Allan. *The History of the Personal Computer: The People and the Technology*. Allan Publishing, 2001.

[14] Shufen Zhang, Shuai Zhang, X. Chen, and S. Wu. "Analysis and Research of Cloud Computing System Instance." Second International Conference on Future Networks, pp. 88-92, Jan 2010.

[15] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li. "Comparison of Several Cloud Computing Platforms." Second International Symposium on Information Science and Engineering, pp. 23-27, Dec 2009.

[16] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong. "The Characteristics of Cloud Computing." 39th International Conference on Parallel Processing Workshops, pp. 275-279, Sep 2010.

[17] A. Velte, T. Velte, and R. Elsenpeter. *Cloud Computing: A Practical Approach*. McGraw-Hill, 2010.

[18] F. Travostino, J. Mambretti, and G. Edwards. *Grid Networks: Enabling Grids with Advanced Communication Technology*. John Wiley & Sons, 2006.

[19] B. Brey. *Programming the 80286, 80386, 80486, and Pentium Based Personal Computer*. Prentice Hall, 1996.

[20] X. Chen and S. Long. "Adaptive Multi-versioning for OpenMP Parallelization via Machine Learning." 15th International Conference on Parallel and Distributed Systems, pp. 907-912, Dec 2009.

[21] S. Mitra and T. Acharya. *Data Mining: Multimedia, Soft Computing, and Bioinformatics*. Wiley, 2003.

[22] P. Gasper, C. Herbst, J. McGough, C. Rickeet, and G. Stubbendieck. "Automatic Parallelization of Sequential C Code." South Dakota School of Mines and Technology, http://www.mcs.sdsmt.edu/mcgough/research/docs/gasper.pdf.

[23] D. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2010.

[24] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[25] M. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2004.

[26] Z. Wang and M. O'Boyle. "Mapping Parallelism to Multi-cores: A Machine Learning Based Approach." 14th ACM SIGPLAN Symposium on Principles and

Practice of Parallel Programming, pp. 75-84, 2009.

[27] Z. Yan and L. Liu. “The Algorithm of Parallel Recognition Based on Data Dependence.” 2009 International Conference on Information Technology and Computer Science, pp. 636-639, Jul 2009.

[28] M. Hornick, E. Marcade, and S. Venkayala. *Java Data Mining: Strategy, Standard, and Practice*. MK Publishers, 2007.

[29] C. Walinsky and D. Banerjee. “A Functional Programming Language Compiler for Massively Parallel Computers.” 1990 ACM Conference on LISP and Functional Programming, pp. 131-138, 1990.

[30] T. Gross, A. Zobel, and M. Zolg. “Parallel Compilation for a Parallel Machine.” ACM SIGPLAN 1989 Conference on Programming Language Design and Implementation, pp. 91-100, 1989.

[31] M. Areanaz, J. Tourino, and R. Doallo. “A GSA-based Compiler Infrastructure to Extract Parallelism from Complex Loops.” 17th Annual International Conference on Supercomputing, pp. 193-204, 2003.

[32] “Intel Guide for Developing Multithreaded Applications,” http://software.intel.com/en-us/articles/intel-guide-for-developing-multithreaded-applications/.

[33] C. Wang, Y. Wu, E. Borin, S. Hu, W. Liu, D. Sager, T. Ngai, and J. Fang. “Dynamic Parallelization of Single-Threaded Binary Programs using Speculative Slicing.” 23rd International Conference on Supercomputing, pp. 158-168, 2009.

[34] “Introduction to Parallel Computing,” https://computing.llnl.gov/tutorials/parallel_comp/.

[35] A. Aho, M. Lam, R. Sethi, and J. Ullman.. *Compilers: Principles, Techniques, & Tools.* Pearson/Addison Wesley, 2007.

[36] M. Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison Wesley, 2010.

[37] P. Pacheco. *Parallel Programming with MPI*. MK Publishers, 1997.

[38] T. Copeland. *Generating Parsers with JavaCC*. Centennial Books, 2007.

[39] “JavaCC is a parser/scanner generator for Java,” http://java.net/projects/javacc/.

[40] “ANTLR Parser Generator,” http://www.antlr.org/.

[41] “Weka 3: Data Mining Software in Java,” http://www.cs.waikato.ac.nz/ml/weka/.

[42] “The OpenMP API Specification for Parallel Programming,” http://openmp.org/wp/.

[43] B. Bruegge and A. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall, 2010.

[44] “C Language Reference,” http://msdn.microsoft.com/en-us/library/fw5abdx6(v=VS.80).aspx.

8. Appendix

8.1. Appendix A: Training Instances in ARFF Format

```
@relation OpenMPForLoopConstructTrainingSet

@attribute num_threads numeric
@attribute num_iterations numeric
@attribute nesting_levels numeric
@attribute complexity_units numeric
@attribute schedule_type
{static,dynamic,guided,runtime,auto}

@data
4,100000,0,500000,static
4,100000,0,600000,static
4,600,1,4320000,static
4,1200,1,23040000,static
4,5,2,86400000,dynamic
4,5,3,414720000000,guided
4,100,1,240000,static
4,100,0,800,static
4,10000,2,80000000000,dynamic
```

8.2. Appendix B: OpenMP Training Source Files

omp_training1.c

```
/*
 * OpenMP training program to evaluate schedule types
 * of FOR loop constructs.
 *
 */


#include <omp.h>


int main()
{
```

```
    int i;
    int j;
    int n = 100000;

    float a[n];
    float b[n];
    float c[n];

    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i)
shared(n, b, a)
    for ( i = 0; i < n; i++ )
    {
        a[i] = b[i] = i * 1.0;
    }

    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(j)
shared(n, b, a, c)
    for ( j = 0; j < n; j++ )
    {
        c[j] = a[j] + b[j];
    }

    return 0;
}
```

omp_training2.c

```
/*
 * OpenMP training program to evaluate schedule types
 * of FOR loop constructs.
 *
 * Ref: http://www.intel.com
 * Lic: http://www.intel.com
 *
 */


#include <stdio.h>
#include <math.h>
#include <omp.h>


int main()
{
    int NTIMES = 5;  // product matrix calculations
    int SIZE = 4800; // must be a multiple of 8
    int M = 600;     // SIZE / 8
    int N = 1200;    // SIZE / 4
    int P = 2400;    // SIZE / 2

    int i, j, k, l;

    double a[M][N];
    double b[N][P];
    double c[M][P];

    // a is identity matrix
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i, j)
shared(M, a, N)
    for ( i = 0; i < M; i++ )
        for ( j = 0; j < N; j++ )
            a[i][j] = 1.0;

    // each column of b is the sequence 1,2,...,N
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i, j)
shared(N, b, P)
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < P; j++ )
```

```
            b[i][j] = i + 1.;

    // initialize product matrix
    #pragma omp parallel for num_threads(4) schedule(dynamic) default(none) private(l, i,
j) shared(NTIMES, M, P, c)
    for ( l = 0; l < NTIMES; l++ )
        for ( i = 0; i < M; i++ )
            for ( j = 0; j < P; j++ )
                c[i][j] = 0.0;

    // this for loop performs intensive calculations
    #pragma omp parallel for num_threads(4) schedule(guided) default(none) private(l, i,
k, j) shared(NTIMES, M, N, b, a, P, c)
    for ( l = 0; l < NTIMES; l++ )
    {
        // product calculation
        for ( i = 0; i < M; i++ )
        {
            for ( k = 0; k < N; k++ )
            {
                // each element of the product is just the sum 1+2+...+n
                for ( j = 0; j < P; j++ )
                {
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
        }
    }

    return 0;
}
```

omp_training3.c

```
/*
 * OpenMP training program to evaluate schedule types
 * of FOR loop constructs.
 *
 * Ref: http://www.intel.com
 * Lic: http://www.intel.com
 *
 */


#include <stdio.h>
#include <omp.h>


int main()
{
    int i, j, num;
    int ARR_SIZE = 100;

    int arr_a[ARR_SIZE];
    int arr_b[ARR_SIZE];
    int arr_c[ARR_SIZE];
    int arr_x[ARR_SIZE];

    int matrix_a[ARR_SIZE][ARR_SIZE];
    int matrix_b[ARR_SIZE][ARR_SIZE];

    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i, j)
shared(arr_b, arr_a, arr_c, ARR_SIZE, matrix_a, matrix_b)
    for ( i = 0; i < ARR_SIZE; i++ )
    {
        arr_a[i] = i;
        arr_b[i] = i;
        arr_c[i] = ARR_SIZE - i;
```

```
        for ( j = 0; j < ARR_SIZE; j++ )
        {
           matrix_a[i][j] = j;
           matrix_b[i][j] = i;
        }
    }

    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i,
num) shared(matrix_a, matrix_b, arr_b, arr_a, arr_c, arr_x, ARR_SIZE)
    for ( i = 0; i < ARR_SIZE; i++ )
    {
        num = matrix_a[arr_b[i]][arr_c[i]];
        arr_x[i] = matrix_b[arr_a[i]][num];
    }

    return 0;
}
```

omp_training4.c

```
/*
 * OpenMP training program to evaluate schedule types
 * of FOR loop constructs.
 *
 * Ref: http://www.openmp.org
 * Lic: http://www.openmp.org
 *
 */


#include <stdio.h>
#include <omp.h>


#define N 10000
#define M 20000


int main()
{
    int i;
    int j;
    int k;

    float a[N][M];
    float b[N][M];

    //
    // This is an imbalanced workload nested loop because
    // number of iterations of the inner loop depends on
    // iteration control of its outer loop.
    #pragma omp parallel for num_threads(4) schedule(dynamic) default(none) private(i, j,
k) shared(b, a)
    for ( i = 2; i < N; i++ )
        for ( j = 2; j <= i; j++ )
            for ( k = 1; k <= M; k++ )
                b[i][j] += a[i-1][j]/k + a[i+1][j]/k;

    return 0;
}
```

8.3. Appendix C: Serial Program Test Files

serial_matrix_product_1.c

```
/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Matrices multiplication
 * -- Ref: http://www.intel.com
 * -- Lic: http://www.intel.com
 *
 * To avoid "Segmentation fault":
 * Linux prompt# ulimit -s unlimited
 *
 */


#include <stdio.h>
#include <math.h>


int main()
{
    int NTIMES = 5;  // product matrix calculations
    int SIZE = 4800; // must be a multiple of 8
    int M = 600;     // SIZE / 8
    int N = 1200;    // SIZE / 4
    int P = 2400;    // SIZE / 2

    int i, j, k, l;

    double a[M][N];
    double b[N][P];
    double c[M][P];

    // a is identity matrix
    for ( i = 0; i < M; i++ )
        for ( j = 0; j < N; j++ )
            a[i][j] = 1.0;

    // each column of b is the sequence 1,2,...,N
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < P; j++ )
            b[i][j] = i + 1.;

    // initialize product matrix
    for ( l = 0; l < NTIMES; l++ )
        for ( i = 0; i < M; i++ )
            for ( j = 0; j < P; j++ )
                c[i][j] = 0.0;

    // this for loop performs intensive calculations
    for ( l = 0; l < NTIMES; l++ )
    {
        // product calculation
        for ( i = 0; i < M; i++ )
        {
            for ( k = 0; k < N; k++ )
            {
                // each element of the product is just the sum 1+2+...+n
                for ( j = 0; j < P; j++ )
                {
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
        }
    }

    return 0;
}
```

serial_matrix_product_2.c

```
/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Matrices multiplication
 * -- Ref: http://www.intel.com
 * -- Lic: http://www.intel.com
 *
 */


#include <stdio.h>


int main()
{
    int i, j, num;
    int ARR_SIZE = 100;

    int arr_a[ARR_SIZE];
    int arr_b[ARR_SIZE];
    int arr_c[ARR_SIZE];
    int arr_x[ARR_SIZE];

    int matrix_a[ARR_SIZE][ARR_SIZE];
    int matrix_b[ARR_SIZE][ARR_SIZE];

    for ( i = 0; i < ARR_SIZE; i++ )
    {
        arr_a[i] = i;
        arr_b[i] = i;
        arr_c[i] = ARR_SIZE - i;

        for ( j = 0; j < ARR_SIZE; j++ )
        {
           matrix_a[i][j] = j;
           matrix_b[i][j] = i;
        }
    }

    for ( i = 0; i < ARR_SIZE; i++ )
    {
        num = matrix_a[arr_b[i]][arr_c[i]];
        arr_x[i] = matrix_b[arr_a[i]][num];
    }

    return 0;
}
```

serial_PI_compute.c

```
/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Computes value of PI
 * -- Ref: SJSU CS 159, Dr. Chun
 *
 */


#include <stdio.h>
#include <time.h>


long long num_steps = 1000000000;
double step;
```

```
int main(int argc, char* argv[])
{
    double x, pi, sum = 0.0;
    int i;
    step = 1.0 / (double)num_steps;
    for ( i = 0; i < num_steps; i++)
    {
        x = (i + 0.5) * step;
        sum = sum + 4.0 / (1.0 + x * x);
    }

    pi = sum * step;

    printf("The value of PI is %15.12f \n", pi);

}
```

serial_prime_finder.c

```
/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Primes finder
 * -- Ref: http://www.intel.com
 * -- Lic: http://www.intel.com
 *
 */


#include <stdio.h>
#include <math.h>


int start = 1;
int end = 100;


int main(int argc, char *argv[])
{
    int i, j, limit;
    int number_of_primes = 0;   /* number of primes found */
    int number_of_41primes = 0; /* number of 4n+1 primes found */
    int number_of_43primes = 0; /* number of 4n-1 primes found */
    int prime;                  /* is the number prime? */
    int print_primes = 0;       /* should each prime be printed? */

    if ( !(start % 2) ) start++;
    printf("Range to check for Primes: %d - %d\n", start, end);

    for ( i = start; i <= end; i += 2 )
    {
        limit = (int) sqrt((float)i) + 1;
        prime = 1; /* assume number is prime */
        j = 3;

        while ( prime && (j <= limit) )
        {
            if ( i % j == 0 ) prime = 0;
            j += 2;
        }

        if ( prime )
        {
            if ( print_primes ) printf("%5d is prime\n", i);
            number_of_primes++;
```

```
            if ( i % 4 == 1 ) number_of_41primes++;
            if ( i % 4 == 3 ) number_of_43primes++;
        }
    }

    printf("Program Done. %d primes found\n", number_of_primes);
    printf("Number of 4n+1 primes found: %d\n", number_of_41primes);
    printf("Number of 4n-1 primes found: %d\n", number_of_43primes);

    return 0;
}
```

serial_trapez_approx.c

```
/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Trapezoidal approximation
 * -- Ref: http://www.cs.stolaf.edu/mtl/
 *
 */


#include <stdio.h>
#include <math.h>


const double PI = 3.14159265358979323846264338307 9;


double f(double);


int main(int argc, char **argv)
{
    /* number of subdivisions = 2^20 */
    int n = 1048576;

    /* limits of integration */
    double a = 0.0, b = PI;

    /* width of subdivision */
    double h = (b - a) / n;

    int i;
    double integral;
    integral = (f(a) + f(b)) / 2.0;

    // This for loop adds over 1 million values
    for ( i = 1; i < n; i++ )
    {
        integral += f(a + i * h);
    }

    // Display the estimated result
    integral = integral * h;
    printf("With n = %d trapezoids, our estimate of the integral from %f to %f is %f\n",
n, a, b, integral);

    return 0;
}


double f(double x)
{
    return sin(x);
}
```

serial_vector_add.c

```
/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Simple vectors addition
 * -- Ref: http://www.openmp.org
 *
 */


#include <stdio.h>


int main()
{
    int i;
    int j;
    int n = 100000;

    float a[n];
    float b[n];
    float c[n];

    // Initialize the vectors
    for ( i = 0; i < n; i++ )
    {
        a[i] = b[i] = i * 1.0;
    }

    // Add the vectors
    for ( j = 0; j < n; j++ )
    {
        c[j] = a[j] + b[j];
    }

    // Display the results
    printf("c[0]=%f, c[n/2]=%f, c[n-1]=%f\n", c[0], c[n/2], c[n-1]);

    return 0;
}
```

8.4. Appendix D: Auto-Parallel Result Files

omp_matrix_product_1.c

```
#include <omp.h>

/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Matrices multiplication
 * -- Ref: http://www.intel.com
 * -- Lic: http://www.intel.com
 *
 * To avoid "Segmentation fault":
 * Linux prompt# ulimit -s unlimited
 *
 */
```

```
#include <stdio.h>
#include <math.h>


int main()
{
    int NTIMES = 5;  // product matrix calculations
    int SIZE = 4800; // must be a multiple of 8
    int M = 600;     // SIZE / 8
    int N = 1200;    // SIZE / 4
    int P = 2400;    // SIZE / 2

    int i, j, k, l;

    double a[M][N];
    double b[N][P];
    double c[M][P];

    // a is identity matrix
    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i, j)
shared(M, a, N)
    for ( i = 0; i < M; i++ )
        for ( j = 0; j < N; j++ )
            a[i][j] = 1.0;

    // each column of b is the sequence 1,2,...,N
    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i, j)
shared(N, b, P)
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < P; j++ )
            b[i][j] = i + 1.;

    // initialize product matrix
    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(dynamic) default(none) private(l, i,
j) shared(NTIMES, M, P, c)
    for ( l = 0; l < NTIMES; l++ )
        for ( i = 0; i < M; i++ )
            for ( j = 0; j < P; j++ )
                c[i][j] = 0.0;

    // this for loop performs intensive calculations
    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(guided) default(none) private(l, i,
k, j) shared(NTIMES, M, N, b, a, P, c)
```

```
    for ( l = 0; l < NTIMES; l++ )
    {
        // product calculation
        for ( i = 0; i < M; i++ )
        {
            for ( k = 0; k < N; k++ )
            {
                // each element of the product is just the sum 1+2+...+n
                for ( j = 0; j < P; j++ )
                {
                    c[i][j] += a[i][k] * b[k][j];
                }
            }
        }
    }

    return 0;
}
```

omp_matrix_product_2.c

```
#include <omp.h>

/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Matrices multiplication
 * -- Ref: http://www.intel.com
 * -- Lic: http://www.intel.com
 *
 */


#include <stdio.h>


int main()
{
    int i, j, num;
    int ARR_SIZE = 100;

    int arr_a[ARR_SIZE];
    int arr_b[ARR_SIZE];
    int arr_c[ARR_SIZE];
    int arr_x[ARR_SIZE];

    int matrix_a[ARR_SIZE][ARR_SIZE];
    int matrix_b[ARR_SIZE][ARR_SIZE];

    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i, j)
shared(arr_b, arr_a, arr_c, ARR_SIZE, matrix_a, matrix_b)
    for ( i = 0; i < ARR_SIZE; i++ )
    {
        arr_a[i] = i;
        arr_b[i] = i;
        arr_c[i] = ARR_SIZE - i;

        for ( j = 0; j < ARR_SIZE; j++ )
        {
           matrix_a[i][j] = j;
           matrix_b[i][j] = i;
```

```
        }
    }

    /*
     * ---------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ---------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i,
num) shared(matrix_a, matrix_b, arr_b, arr_a, arr_c, arr_x, ARR_SIZE)
    for ( i = 0; i < ARR_SIZE; i++ )
    {
        num = matrix_a[arr_b[i]][arr_c[i]];
        arr_x[i] = matrix_b[arr_a[i]][num];
    }

    return 0;
}
```

omp_PI_compute.c

```
#include <omp.h>

/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Computes value of PI
 * -- Ref: SJSU CS 159, Dr. Chun
 *
 */


#include <stdio.h>
#include <time.h>


long long num_steps = 1000000000;
double step;


int main(int argc, char* argv[])
{
    double x, pi, sum = 0.0;
    int i;
    step = 1.0 / (double)num_steps;
    /*
     * ---------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ---------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i, x)
shared(num_steps, step) reduction(+: sum)
    for ( i = 0; i < num_steps; i++)
    {
        x = (i + 0.5) * step;
        sum = sum + 4.0 / (1.0 + x * x);
    }

    pi = sum * step;

    printf("The value of PI is %15.12f \n", pi);

}
```

omp_prime_finder.c

```
#include <omp.h>

/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Primes finder
 * -- Ref: http://www.intel.com
 * -- Lic: http://www.intel.com
 *
 */


#include <stdio.h>
#include <math.h>


int start = 1;
int end = 100;


int main(int argc, char *argv[])
{
    int i, j, limit;
    int number_of_primes = 0;   /* number of primes found */
    int number_of_41primes = 0; /* number of 4n+1 primes found */
    int number_of_43primes = 0; /* number of 4n-1 primes found */
    int prime;                  /* is the number prime? */
    int print_primes = 0;       /* should each prime be printed? */

    if ( !(start % 2) ) start++;
    printf("Range to check for Primes: %d - %d\n", start, end);

    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i,
limit, prime) shared(start, end, print_primes) reduction(+: j) reduction(+:
number_of_43primes) reduction(+: number_of_41primes) reduction(+: number_of_primes)
    for ( i = start; i <= end; i += 2 )
    {
        limit = (int) sqrt((float)i) + 1;
        prime = 1; /* assume number is prime */
        j = 3;

        while ( prime && (j <= limit) )
        {
            if ( i % j == 0 ) prime = 0;
            j += 2;
        }

        if ( prime )
        {
            if ( print_primes ) printf("%5d is prime\n", i);
            number_of_primes++;

            if ( i % 4 == 1 ) number_of_41primes++;
            if ( i % 4 == 3 ) number_of_43primes++;
        }
    }

    printf("Program Done. %d primes found\n", number_of_primes);
```

```
    printf("Number of 4n+1 primes found: %d\n", number_of_41primes);
    printf("Number of 4n-1 primes found: %d\n", number_of_43primes);

    return 0;
}
```

omp_trapez_approx.c

```
#include <omp.h>

/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Trapezoidal approximation
 * -- Ref: http://www.cs.stolaf.edu/mtl/
 *
 */


#include <stdio.h>
#include <math.h>


const double PI = 3.14159265358979323846264338307 9;


double f(double);


int main(int argc, char **argv)
{
    /* number of subdivisions = 2^20 */
    int n = 1048576;

    /* limits of integration */
    double a = 0.0, b = PI;

    /* width of subdivision */
    double h = (b - a) / n;

    int i;
    double integral;
    integral = (f(a) + f(b)) / 2.0;

    // This for loop adds over 1 million values
    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i)
shared(n, a, h) reduction(+: integral)
    for ( i = 1; i < n; i++ )
    {
        integral += f(a + i * h);
    }

    // Display the estimated result
    integral = integral * h;
    printf("With n = %d trapezoids, our estimate of the integral from %f to %f is %f\n",
n, a, b, integral);

    return 0;
}
```

```
double f(double x)
{
    return sin(x);
}
```

omp_vector_add.c

```
#include <omp.h>

/*
 * Serial test program for OpenMP auto-parallelization
 *
 * -- Simple vectors addition
 * -- Ref: http://www.openmp.org
 *
 */


#include <stdio.h>


int main()
{
    int i;
    int j;
    int n = 100000;

    float a[n];
    float b[n];
    float c[n];

    // Initialize the vectors
    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(i)
shared(n, b, a)
    for ( i = 0; i < n; i++ )
    {
        a[i] = b[i] = i * 1.0;
    }

    // Add the vectors
    /*
     * ------------------------------------------------
     * Automatic OpenMP Parallelization
     * by Nam Q. Lam's MasterProject @ SJSU
     * Do not edit this auto-generated loop construct
     * ------------------------------------------------
     */
    #pragma omp parallel for num_threads(4) schedule(static) default(none) private(j)
shared(n, b, a, c)
    for ( j = 0; j < n; j++ )
    {
        c[j] = a[j] + b[j];
    }

    // Display the results
    printf("c[0]=%f, c[n/2]=%f, c[n-1]=%f\n", c[0], c[n/2], c[n-1]);

    return 0;
}
```

8.5. Appendix E: C Grammar in BNF Format

```
/*
 * Lexical Specification -- TOKENS
 *
 */

<DEFAULT> MORE : {
<PREPROCESSOR_START: "#"> : INSIDE_PREPROCESSOR
}

<INSIDE_PREPROCESSOR> MORE : {
<PREPROCESSOR_DATA: "\\" ("\r" | "\n" | "\r\n") | <ANYCHAR>>
}

<INSIDE_PREPROCESSOR> SPECIAL : {
<PREPROCESSOR_END: "\r" | "\n" | "\r\n"> : DEFAULT
}

<DEFAULT> MORE : {
<BLOCK_START: <BLOCK_COMMENT_START>> : INSIDE_BLOCK_COMMENT
}

<INSIDE_BLOCK_COMMENT> MORE : {
<INSIDE_COMMENTS: <ANYCHAR>>
}

<INSIDE_BLOCK_COMMENT> SPECIAL : {
<BLOCK_END: <BLOCK_COMMENT_END>> : DEFAULT
}

<DEFAULT> SPECIAL : {
<LINE_COMMENT: <DOUBLE_SLASHES> (<ANYCHAR_NOT_EOL>)*>
}

<DEFAULT> SKIP : {
<SPACE: " " | "\t">
| <EOL: "\r" | "\n" | "\r\n">
}

<DEFAULT> TOKEN : {
<INTEGER: <OCTAL> (<INTEGER_SUFFIX>)? | <DEC> (<INTEGER_SUFFIX>)? | <HEX>
(<INTEGER_SUFFIX>)?>
| <REAL: <WHOLE> <DPOINT> (<FRACTION>)? (<EXPONENT>)? (<REAL_SUFFIX>)? | <DPOINT>
<FRACTION> (<EXPONENT>)? (<REAL_SUFFIX>)? | <FRACTION> <EXPONENT> (<REAL_SUFFIX>)? |
<WHOLE> (<EXPONENT>)? (<REAL_SUFFIX>)?>
| <CHARACTER: "\'" (<ANYCHAR_NOT_ESCAPE> | <ESCAPE_SEQUENCE>)? "\'">
| <STRING: "\"" (<ANYCHAR_NOT_ESCAPE> | <ESCAPE_SEQUENCE> | "\\" ("\r" | "\n" | "\r\n"))*
"\"">
| <AUTO: "auto">
| <REGISTER: "register">
| <EXTERN: "extern">
| <STRUCT: "struct">
| <UNION: "union">
| <STATIC: "static">
| <TYPEDEF: "typedef">
| <ENUM: "enum">
| <CONST: "const">
| <VOLATILE: "volatile">
| <VOID: "void">
| <CHAR: "char">
| <SHORT: "short">
| <INT: "int">
| <LONG: "long">
| <FLOAT: "float">
| <DOUBLE: "double">
| <SIGNED: "signed">
| <UNSIGNED: "unsigned">
| <IF: "if">
```

```
| <ELSE: "else">
| <SWITCH: "switch">
| <CASE: "case">
| <FOR: "for">
| <DO: "do">
| <WHILE: "while">
| <BREAK: "break">
| <CONTINUE: "continue">
| <CDEFAULT: "default">
| <GOTO: "goto">
| <SIZEOF: "sizeof">
| <RETURN: "return">
| <PLUS_PLUS: "++">
| <MINUS_MINUS: "--">
| <POINTER_SELECT: "->">
| <DOT: <PERIOD>>
| <LEFT_PAREN: "(">
| <RIGHT_PAREN: ")">
| <LEFT_BRACKET: "[">
| <RIGHT_BRACKET: "]">
| <LEFT_BRACE: "{">
| <RIGHT_BRACE: "}">
| <AMPERSAND: "&">
| <STAR: "*">
| <PLUS: "+">
| <MINUS: "-">
| <BITWISE_NOT: "~">
| <LOGICAL_NOT: "!">
| <SLASH: "/">
| <PERCENT: "%">
| <SHIFT_LEFT: "<<">
| <SHIFT_RIGHT: ">>">
| <LESS_THAN: "<">
| <GREATER_THAN: ">">
| <LESS_EQUALS: "<=">
| <GREATER_EQUALS: ">=">
| <EQUALS_EQUALS: "==">
| <NOT_EQUALS: "!=">
| <BITWISE_XOR: "^">
| <BITWISE_OR: "|">
| <LOGICAL_AND: "&&">
| <LOGICAL_OR: "||">
| <QUESTION_MARK: "?">
| <COLON: ":">
| <EQUALS: "=">
| <STAR_EQUALS: "*=">
| <SLASH_EQUALS: "/=">
| <PERCENT_EQUALS: "%=">
| <PLUS_EQUALS: "+=">
| <MINUS_EQUALS: "-=">
| <LESS_LESS_EQUALS: "<<=">
| <GREATER_GREATER_EQUALS: ">>=">
| <BITWISE_OR_EQUALS: "|=">
| <AMP_EQUALS: "&=">
| <BITWISE_XOR_EQUALS: "^=">
| <SEMICOLON: ";">
| <COMMA: ",">
| <SINGLE_QUOTE: "\'">
| <DOUBLE_QUOTE: "\"">
| <AT_SIGN: "@">
| <THREE_DOTS: "...">
| <SLASH_SLASH: <DOUBLE_SLASHES>>
| <SLASH_STAR: <BLOCK_COMMENT_START>>
| <STAR_SLASH: <BLOCK_COMMENT_END>>
| <IDENTIFIER: ("_" | <LETTER>) ("_" | <LETTER> | <DIGIT>)*>
| <#PERIOD: ".">
| <#ANYCHAR: ~[]>
| <#ANYCHAR_NOT_EOL: ~["\n","\r"]>
| <#ANYCHAR_NOT_ESCAPE: ~["\n","\r","\'","\"","\\"]>
| <#ESCAPE_SEQUENCE: "\\" (["a","b","f","n","r","t","v","\'","\"","?","\\","0"] | (["0"-
"7"]){1,3} | ["x","X"] (["0"-"9","a"-"f","A"-"F"]){1,4})>
```

```
| <#LETTER: ["A"-"Z"] | ["a"-"z"]>
| <#DIGIT: ["0"-"9"]>
| <#DIGIT_NOT_ZERO: ["1"-"9"]>
| <#OCTAL: "0" (["0"-"7"])*>
| <#DEC: <DIGIT_NOT_ZERO> (<DIGIT>)*>
| <#HEX: "0" ["x","X"] (["0"-"9","a"-"f","A"-"F"])+>
| <#INTEGER_SUFFIX: ["L","l"] (["U","u"])? | ["U","u"] (["L","l"])?>
| <#REAL_SUFFIX: ["L","l","F","f"]>
| <#WHOLE: (<DIGIT>)+>
| <#DPOINT: <PERIOD>>
| <#FRACTION: (<DIGIT>)+>
| <#EXPONENT: ["e","E"] (["+","-"])? (<DIGIT>)+>
| <#BLOCK_COMMENT_START: "/*">
| <#BLOCK_COMMENT_END: "*/">
| <#DOUBLE_SLASHES: "//">
}


/*
 * Syntactic Specification -- NON-TERMINALS
 *
 */

parse                        :=    ( externalDeclaration )+
externalDeclaration          :=    ( functionDefinition | declaration )
functionDefinition           :=    ( declarationSpecifiers )? declarator ( declarationList
)? compoundStatement
functionDefinitionLookahead :=    ( declarationSpecifiers )? declarator ( declarationList
)? <LEFT_BRACE>
declarationList              :=    ( declaration )+
declaration                  :=    declarationSpecifiers ( initDeclaratorList )?
<SEMICOLON>
declarationSpecifiers        :=    ( storageClassSpecifier ( declarationSpecifiers )? |
typeSpecifier ( declarationSpecifiers )? | typeQualifier ( declarationSpecifiers )? )
initDeclaratorList           :=    initDeclarator ( <COMMA> initDeclarator )*
initDeclarator               :=    declarator ( <EQUALS> initializer )?
initializer                  :=    ( assignmentExpression | <LEFT_BRACE> initializerList (
<COMMA> )? <RIGHT_BRACE> )
initializerList              :=    initializer ( <COMMA> initializer )*
typeName                     :=    specifierQualifierList ( abstractDeclarator )?
storageClassSpecifier        :=    ( <AUTO> | <REGISTER> | <STATIC> | <EXTERN> | <TYPEDEF>
)
typeSpecifier                :=    ( <VOID> | <CHAR> | <SHORT> | <INT> | <LONG> | <FLOAT>
| <DOUBLE> | <SIGNED> | <UNSIGNED> | structOrUnionSpecifier | enumSpecifier | typedefName
)
typedefName                  :=    identifier
typeQualifier                :=    ( <CONST> | <VOLATILE> )
structOrUnionSpecifier       :=    structOrUnion ( ( identifier )? <LEFT_BRACE>
structDeclarationList <RIGHT_BRACE> | identifier )
structOrUnion                :=    ( <STRUCT> | <UNION> )
structDeclarationList        :=    ( structDeclaration )+
structDeclaration            :=    specifierQualifierList structDeclaratorList <SEMICOLON>
specifierQualifierList       :=    typeSpecifier ( specifierQualifierList )? |
typeQualifier ( specifierQualifierList )?
structDeclaratorList         :=    structDeclarator ( <COMMA> structDeclarator )*
structDeclarator             :=    ( declarator | typeSpecifier ( declarator )? <COLON>
constantExpression )
enumSpecifier                :=    <ENUM> ( ( identifier )? <LEFT_BRACE> enumeratorList
<RIGHT_BRACE> | identifier )
enumeratorList               :=    enumerator ( <COMMA> enumerator )*
enumerator                   :=    identifier ( <EQUALS> constantExpression )?
declarator                   :=    ( pointer )? directDeclarator
pointer                      :=    <STAR> ( typeQualifierList )? ( pointer )?
typeQualifierList            :=    ( typeQualifier )+
directDeclarator             :=    ( ( identifier | <LEFT_PAREN> declarator <RIGHT_PAREN>
) ( <LEFT_BRACKET> ( constantExpression )? <RIGHT_BRACKET> | <LEFT_PAREN>
parameterTypeList <RIGHT_PAREN> | <LEFT_PAREN> ( identifierList )? <RIGHT_PAREN> )* )
parameterTypeList            :=    parameterList ( <COMMA> <THREE_DOTS> )?
parameterList                :=    parameterDeclaration ( <COMMA> parameterDeclaration )*
parameterDeclaration         :=    declarationSpecifiers ( declarator | (
abstractDeclarator )? )
```

```
abstractDeclarator          :=    ( pointer | ( pointer )? directAbstractDeclarator )
directAbstractDeclarator    :=    ( <LEFT_PAREN> abstractDeclarator <RIGHT_PAREN> |
<LEFT_BRACKET> ( constantExpression )? <RIGHT_BRACKET> | <LEFT_PAREN> ( parameterTypeList
)? <RIGHT_PAREN> ) ( <LEFT_BRACKET> ( constantExpression )? <RIGHT_BRACKET> |
<LEFT_PAREN> ( parameterTypeList )? <RIGHT_PAREN> )*
identifierList              :=    identifier ( <COMMA> identifier )*
statementList               :=    ( statement )+
statement                   :=    ( compoundStatement | labeledStatement | jumpStatement
| expressionStatement | ifStatement | switchStatement | forStatement | whileStatement |
doWhileStatement )
compoundStatement           :=    <LEFT_BRACE> ( declarationList )? ( statementList )?
<RIGHT_BRACE>
labeledStatement            :=    ( <CASE> constantExpression <COLON> statement |
<CDEFAULT> <COLON> statement | identifier <COLON> statement )
jumpStatement               :=    ( <GOTO> identifier <SEMICOLON> | <BREAK> <SEMICOLON> |
<CONTINUE> <SEMICOLON> | <RETURN> ( expression )? <SEMICOLON> )
expressionStatement         :=    ( expression )? <SEMICOLON>
ifStatement                 :=    <IF> <LEFT_PAREN> expression <RIGHT_PAREN> statement (
<ELSE> statement )?
switchStatement             :=    <SWITCH> <LEFT_PAREN> expression <RIGHT_PAREN>
statement
forStatement                :=    ( ( <FOR> ) <LEFT_PAREN> ( ( typeSpecifier )?
expression )? <SEMICOLON> ( expression )? <SEMICOLON> ( expression )? <RIGHT_PAREN>
statement )
whileStatement              :=    <WHILE> <LEFT_PAREN> expression <RIGHT_PAREN> statement
doWhileStatement            :=    <DO> statement <WHILE> <LEFT_PAREN> expression
<RIGHT_PAREN> <SEMICOLON>
expression                  :=    ( assignmentExpression ( <COMMA> assignmentExpression
)* )
assignmentExpression        :=    ( unaryExpression ( <EQUALS> | <STAR_EQUALS> |
<SLASH_EQUALS> | <PERCENT_EQUALS> | <PLUS_EQUALS> | <MINUS_EQUALS> | <LESS_LESS_EQUALS> |
<GREATER_GREATER_EQUALS> | <BITWISE_OR_EQUALS> | <AMP_EQUALS> | <BITWISE_XOR_EQUALS> )
assignmentExpression | conditionalExpression )
constantExpression          :=    conditionalExpression
conditionalExpression       :=    logicalORExpression ( <QUESTION_MARK> expression
<COLON> conditionalExpression )?
logicalORExpression         :=    logicalANDExpression ( <LOGICAL_OR> logicalORExpression
)?
logicalANDExpression        :=    bitwiseORExpression ( <LOGICAL_AND>
logicalANDExpression )?
bitwiseORExpression         :=    bitwiseXORExpression ( <BITWISE_OR> bitwiseORExpression
)?
bitwiseXORExpression        :=    bitwiseANDExpression ( <BITWISE_XOR>
bitwiseXORExpression )?
bitwiseANDExpression        :=    equalityExpression ( <AMPERSAND> bitwiseANDExpression
)?
equalityExpression          :=    relationalExpression ( ( <EQUALS_EQUALS> | <NOT_EQUALS>
) equalityExpression )?
relationalExpression        :=    shiftExpression ( ( <LESS_THAN> | <LESS_EQUALS> |
<GREATER_THAN> | <GREATER_EQUALS> ) relationalExpression )?
shiftExpression             :=    additiveExpression ( ( <SHIFT_LEFT> | <SHIFT_RIGHT> )
shiftExpression )?
additiveExpression          :=    multiplicativeExpression ( ( <PLUS> | <MINUS> )
additiveExpression )?
multiplicativeExpression    :=    castExpression ( ( <STAR> | <SLASH> | <PERCENT> )
multiplicativeExpression )?
castExpression              :=    ( <LEFT_PAREN> typeName <RIGHT_PAREN> castExpression |
unaryExpression )
unaryExpression             :=    ( ( postfixExpression ) | ( <PLUS_PLUS> | <MINUS_MINUS>
) ( unaryExpression ) | ( <AMPERSAND> | <STAR> | <PLUS> | <MINUS> | <BITWISE_NOT> |
<LOGICAL_NOT> ) castExpression | <SIZEOF> ( unaryExpression | <LEFT_PAREN> typeName
<RIGHT_PAREN> ) )
postfixExpression           :=    ( ( primaryExpression ) ( ( <LEFT_BRACKET> expression
<RIGHT_BRACKET> | <LEFT_PAREN> ( expression )? <RIGHT_PAREN> | <DOT> identifier |
<POINTER_SELECT> identifier | <PLUS_PLUS> | <MINUS_MINUS> ) )* )
primaryExpression           :=    ( identifier | constant | <LEFT_PAREN> expression
<RIGHT_PAREN> )
identifier                  :=    ( <IDENTIFIER> )
constant                    :=    ( integer | real | string | character )
integer                     :=    ( <INTEGER> )
real                        :=    ( <REAL> )
```

```
string                      :=    ( <STRING> )
character                   :=    ( <CHARACTER> )
```

8.6. Appendix F: C Grammar for JavaCC

```
/*
 * C.jjt
 *
 * Created on 01/25/2011
 *
 * by Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 * Purpose:
 * - To build the C language parser using JavaCC,
 *   in which the AST is constructed where each
 *   JJTree node is specifically implemented to
 *   to extract appropriate information for
 *   parallelism analysis of C programs.
 *
 * References:
 * - C Language Reference on MSDN Library site, http://msdn.microsoft.com/en-
us/library/fw5abdx6.aspx
 * - Writing Compilers and Interpreters: A Software Engineering Approach by Ronald Mak
 * - Generating Parsers with JavaCC by Tom Compeland
 * - JavaCC grammars repository on java.net
 * - http://www.openmp.org
 *
 */


options {
  STATIC = false;
  MULTI = true;
  VISITOR = true;
  JDK_VERSION = "1.5";
  JJTREE_OUTPUT_DIRECTORY = "frontend";
  NODE_EXTENDS = "nql.cd.intermediate.NotSimpleNode";

  //// DEBUG_TOKEN_MANAGER = true;
  //// DEBUG_PARSER = true;
}


PARSER_BEGIN(CParser)
package nql.cd.frontend;
import nql.cd.intermediate.*;
import java.util.*;
import java.io.*;
public class CParser {

    /* Handle typedef types. */
    private Set<String> types = new HashSet<String>();
    private Stack<Boolean> typedefParsingStack = new Stack<Boolean>();
    private boolean isType(String type) {
           return types.contains(type);
    }
    private void addType(String type) {
           types.add(type);
    }

    /* This main entry is a test driver for the generated parser. */
    public static void main(String [] args) {
        try {
            StringBuffer sb = new StringBuffer();
            Reader reader = new FileReader(args[0]);
```

```
            CParser parser = new CParser(reader);
            SimpleNode rootNode = parser.parse();
            rootNode.dump(">", sb);
            System.out.println(sb.toString());
        }
        catch ( Exception ex ) {
            ex.printStackTrace();
        }
    }
}
PARSER_END(CParser)


TOKEN_MGR_DECLS :
{
    /* This main entry is a test driver for token specifications. */
    public static void main(String [] args) {
        try {
            java.io.BufferedReader cSourceFile = new java.io.BufferedReader(new
java.io.FileReader(args[0]));
            SimpleCharStream scs = new SimpleCharStream(cSourceFile);
            CParserTokenManager mgr = new CParserTokenManager(scs);
            java.lang.reflect.Field[] tokens =
CParserConstants.class.getDeclaredFields();
            Token t = mgr.getNextToken();
            while ( t.kind != EOF ) {
                String tokenName = tokens[t.kind].getName();

                // STATIC = true
                // debugStream.println(tokenName + ": " + t.image);

                // STATIC = false
                System.out.println(tokenName + ": " + t.image);

                t = mgr.getNextToken();
            }
        }
        catch ( Exception ex ) {
            ex.printStackTrace();
        }
    }
}


/*
 * Lexical Specification
 *
 */


MORE : {
  <PREPROCESSOR_START : "#"> : INSIDE_PREPROCESSOR
}


<INSIDE_PREPROCESSOR>
MORE : {
  <PREPROCESSOR_DATA : (("\\" ("\r" | "\n" | "\r\n")) | <ANYCHAR>)>
}


<INSIDE_PREPROCESSOR>
SPECIAL_TOKEN : {
  <PREPROCESSOR_END : ("\r" | "\n" | "\r\n")> : DEFAULT
}


MORE : {
  <BLOCK_START : <BLOCK_COMMENT_START>> : INSIDE_BLOCK_COMMENT
}
```

```
<INSIDE_BLOCK_COMMENT>
MORE : {
  <INSIDE_COMMENTS : <ANYCHAR>>
}


<INSIDE_BLOCK_COMMENT>
SPECIAL_TOKEN : {
  <BLOCK_END : <BLOCK_COMMENT_END>> : DEFAULT
}


SPECIAL_TOKEN : {
  <LINE_COMMENT : <DOUBLE_SLASHES> (<ANYCHAR_NOT_EOL>)*>
}


SKIP : {
  <SPACE : (" " | "\t")>
  | <EOL : ("\r" | "\n" | "\r\n")>
}


TOKEN : {
  <INTEGER: <OCTAL>(<INTEGER_SUFFIX>)? | <DEC>(<INTEGER_SUFFIX>)? |
<HEX>(<INTEGER_SUFFIX>)?>
  | <REAL : <WHOLE><DPOINT>(<FRACTION>)?(<EXPONENT>)?(<REAL_SUFFIX>)?
  | <DPOINT><FRACTION>(<EXPONENT>)?(<REAL_SUFFIX>)?
  | <FRACTION><EXPONENT>(<REAL_SUFFIX>)?
  | <WHOLE>(<EXPONENT>)?(<REAL_SUFFIX>)?>
  | <CHARACTER : "'"(<ANYCHAR_NOT_ESCAPE> | <ESCAPE_SEQUENCE>)?"'">
  | <STRING : "\""(<ANYCHAR_NOT_ESCAPE> | <ESCAPE_SEQUENCE> | ("\\" ("\r" | "\n" |
"\r\n")))*"\"">
  | <AUTO : "auto">
  | <REGISTER : "register">
  | <EXTERN : "extern">
  | <STRUCT: "struct">
  | <UNION : "union">
  | <STATIC : "static">
  | <TYPEDEF : "typedef">
  | <ENUM : "enum">
  | <CONST : "const">
  | <VOLATILE : "volatile">
  | <VOID : "void">
  | <CHAR : "char">
  | <SHORT : "short">
  | <INT : "int">
  | <LONG : "long">
  | <FLOAT : "float">
  | <DOUBLE : "double">
  | <SIGNED : "signed">
  | <UNSIGNED : "unsigned">
  | <IF : "if">
  | <ELSE : "else">
  | <SWITCH : "switch">
  | <CASE : "case">
  | <FOR : "for">
  | <DO : "do">
  | <WHILE : "while">
  | <BREAK : "break">
  | <CONTINUE : "continue">
  | <CDEFAULT : "default">
  | <GOTO : "goto">
  | <SIZEOF : "sizeof">
  | <RETURN : "return">
  | <PLUS_PLUS : "++">
  | <MINUS_MINUS : "--">
  | <POINTER_SELECT : "->">
  | <DOT : <PERIOD>>
  | <LEFT_PAREN : "(">
```

```
  | <RIGHT_PAREN : ")">
  | <LEFT_BRACKET : "[">
  | <RIGHT_BRACKET : "]">
  | <LEFT_BRACE : "{">
  | <RIGHT_BRACE : "}">
  | <AMPERSAND : "&">
  | <STAR : "*">
  | <PLUS : "+">
  | <MINUS : "-">
  | <BITWISE_NOT : "~">
  | <LOGICAL_NOT : "!">
  | <SLASH : "/">
  | <PERCENT : "%">
  | <SHIFT_LEFT : "<<">
  | <SHIFT_RIGHT : ">>">
  | <LESS_THAN : "<">
  | <GREATER_THAN : ">">
  | <LESS_EQUALS : "<=">
  | <GREATER_EQUALS : ">=">
  | <EQUALS_EQUALS : "==">
  | <NOT_EQUALS : "!=">
  | <BITWISE_XOR : "^">
  | <BITWISE_OR : "|">
  | <LOGICAL_AND : "&&">
  | <LOGICAL_OR : "||">
  | <QUESTION_MARK : "?">
  | <COLON : ":">
  | <EQUALS : "=">
  | <STAR_EQUALS : "*=">
  | <SLASH_EQUALS : "/=">
  | <PERCENT_EQUALS : "%=">
  | <PLUS_EQUALS : "+=">
  | <MINUS_EQUALS : "-=">
  | <LESS_LESS_EQUALS : "<<=">
  | <GREATER_GREATER_EQUALS : ">>=">
  | <BITWISE_OR_EQUALS : "|=">
  | <AMP_EQUALS : "&=">
  | <BITWISE_XOR_EQUALS : "^=">
  | <SEMICOLON : ";">
  | <COMMA : ",">
  | <SINGLE_QUOTE : "'">
  | <DOUBLE_QUOTE : "\"">
  | <AT_SIGN : "@">
  | <THREE_DOTS : "...">
  | <SLASH_SLASH : <DOUBLE_SLASHES>>
  | <SLASH_STAR : <BLOCK_COMMENT_START>>
  | <STAR_SLASH : <BLOCK_COMMENT_END>>
  | <IDENTIFIER : ("_" | <LETTER>) ("_" | <LETTER> | <DIGIT>)*>
  | <#PERIOD : ".">
  | <#ANYCHAR : ~[]>
  | <#ANYCHAR_NOT_EOL : ~["\n","\r"]>
  | <#ANYCHAR_NOT_ESCAPE : ~["\n","\r","'","\"","\\"]>
  | <#ESCAPE_SEQUENCE : "\\"(["a","b","f","n","r","t","v","'","\"","?","\\","0"] | (["0"-
"7"]){1,3} | ["x","X"](["0"-"9","a"-"f","A"-"F"]){1,4})>
  | <#LETTER : ["A"-"Z"] | ["a"-"z"]>
  | <#DIGIT : ["0"-"9"]>
  | <#DIGIT_NOT_ZERO : ["1"-"9"]>
  | <#OCTAL : "0"(["0"-"7"])*>
  | <#DEC : <DIGIT_NOT_ZERO>(<DIGIT>)*>
  | <#HEX : "0"["x","X"](["0"-"9","a"-"f","A"-"F"])+>
  | <#INTEGER_SUFFIX : (["L","l"](["U","u"])?) | (["U","u"](["L","l"])?)>
  | <#REAL_SUFFIX : ["L","l","F","f"]>
  | <#WHOLE : (<DIGIT>)+>
  | <#DPOINT : <PERIOD>>
  | <#FRACTION : (<DIGIT>)+>
  | <#EXPONENT : ["e","E"](["+","-"])?(<DIGIT>)+>
  | <#BLOCK_COMMENT_START : "/*">
  | <#BLOCK_COMMENT_END : "*/">
  | <#DOUBLE_SLASHES : "//">
}
```

```
/*
 * Syntactic Specification
 *
 */


SimpleNode parse() : {}
{
    (externalDeclaration())+
    { return jjtThis; }
}


void externalDeclaration() : {}
{
    (LOOKAHEAD(functionDefinitionLookahead()) functionDefinition() | declaration())
}


void functionDefinition() : {}
{
    [LOOKAHEAD(declarationSpecifiers()) declarationSpecifiers()] declarator()
[declarationList()]
    compoundStatement()
}


void functionDefinitionLookahead() : {}
{
    [LOOKAHEAD(declarationSpecifiers()) declarationSpecifiers()] declarator()
[declarationList()] <LEFT_BRACE>
}


void declarationList() : {}
{
    (LOOKAHEAD(declaration()) declaration())+
}


void declaration() : {}
{
    declarationSpecifiers() [initDeclaratorList()] <SEMICOLON>
}


void declarationSpecifiers() : {}
{
    storageClassSpecifier() [LOOKAHEAD(declarationSpecifiers()) declarationSpecifiers()]
    | typeSpecifier() [LOOKAHEAD(declarationSpecifiers()) declarationSpecifiers()]
    | typeQualifier() [LOOKAHEAD(declarationSpecifiers()) declarationSpecifiers()]
}


void initDeclaratorList() : {}
{
    initDeclarator() (<COMMA> initDeclarator())*
    {
        // Finished with a typedefDeclaration??
        if ( !(typedefParsingStack.empty()) &&
((Boolean)typedefParsingStack.peek()).booleanValue() ) {
            typedefParsingStack.pop();
        }
    }
}


void initDeclarator() : {}
{
    declarator() [<EQUALS> initializer()]
```

```
}


void initializer() : {}
{
    (assignmentExpression()
    | <LEFT_BRACE> initializerList() [<COMMA>] <RIGHT_BRACE>)
}


void initializerList() : {}
{
    initializer() (LOOKAHEAD(2) <COMMA> initializer())*
}


void typeName() : {}
{
    specifierQualifierList() [abstractDeclarator()]
}


void storageClassSpecifier() : {}
{
    (<AUTO> | <REGISTER> | <STATIC> | <EXTERN> | <TYPEDEF>
    { typedefParsingStack.push(Boolean.TRUE); })
}


void typeSpecifier() : {}
{
    (<VOID> | <CHAR> | <SHORT> | <INT> | <LONG> | <FLOAT> | <DOUBLE> | <SIGNED> |
<UNSIGNED>
    | structOrUnionSpecifier()
    | enumSpecifier()
    | LOOKAHEAD({ isType(getToken(1).image) }) typedefName())
}


void typedefName() : {}
{
    identifier(false)
}


void typeQualifier() : {}
{
    (<CONST> | <VOLATILE>)
}


void structOrUnionSpecifier() : {}
{
    { typedefParsingStack.push(Boolean.FALSE); }
    structOrUnion() (LOOKAHEAD(3) [identifier(false)] <LEFT_BRACE>
structDeclarationList() <RIGHT_BRACE> | identifier(false))
    { typedefParsingStack.pop(); }
}


void structOrUnion() : {}
{
    (<STRUCT> | <UNION>)
}


void structDeclarationList() : {}
{
    (structDeclaration())+
}
```

```
void structDeclaration() : {}
{
    specifierQualifierList() structDeclaratorList() <SEMICOLON>
}


void specifierQualifierList() #void : {}
{
    typeSpecifier() [LOOKAHEAD(specifierQualifierList()) specifierQualifierList()]
    | typeQualifier() [LOOKAHEAD(specifierQualifierList()) specifierQualifierList()]
}


void structDeclaratorList() : {}
{
    structDeclarator() (<COMMA> structDeclarator())*
}


void structDeclarator() : {}
{
    (LOOKAHEAD(3) declarator()
    | typeSpecifier() [declarator()] <COLON> constantExpression())
}


void enumSpecifier() : {}
{
    <ENUM> (LOOKAHEAD(3) [identifier(false)] <LEFT_BRACE> enumeratorList() <RIGHT_BRACE>
| identifier(false))
}


void enumeratorList() : {}
{
    enumerator() (<COMMA> enumerator())*
}


void enumerator() : {}
{
    identifier(false) [<EQUALS> constantExpression()]
}


void declarator() : {}
{
    [pointer()] directDeclarator()
}


void pointer() : {}
{
    <STAR> [typeQualifierList()] [pointer()]
}


void typeQualifierList() #void : {}
{
    (typeQualifier())+
}


void directDeclarator() :
{
    Token t;
    boolean isFunction = false;
}
{
    ((t = identifier(false)
```

```
    {
        if ( !(typedefParsingStack.empty()) &&
((Boolean)typedefParsingStack.peek()).booleanValue() ) {
            addType(t.image);
        }
    }
    | <LEFT_PAREN> declarator() <RIGHT_PAREN>)
    {
        typedefParsingStack.push(Boolean.FALSE);
    }
    (<LEFT_BRACKET> [constantExpression()] <RIGHT_BRACKET>
    | LOOKAHEAD(3)
      <LEFT_PAREN> { isFunction = true; } parameterTypeList() <RIGHT_PAREN>
    | <LEFT_PAREN> { isFunction = true; } [identifierList()] <RIGHT_PAREN>)*
    {
        typedefParsingStack.pop();
    })
    {
        jjtThis.setAttribute(NodeKey.FUNCTION_DEC_DEF, isFunction);
    }
}


void parameterTypeList() : {}
{
    parameterList() [<COMMA> <THREE_DOTS>]
}


void parameterList() : {}
{
    parameterDeclaration() (LOOKAHEAD(2) <COMMA> parameterDeclaration())*
}


void parameterDeclaration() : {}
{
    declarationSpecifiers() (LOOKAHEAD(declarator()) declarator() |
[abstractDeclarator()])
}


void abstractDeclarator() : {}
{
    (LOOKAHEAD(3) pointer()
    | [pointer()] directAbstractDeclarator())
}


void directAbstractDeclarator() : {}
{
    (LOOKAHEAD(2) <LEFT_PAREN> abstractDeclarator() <RIGHT_PAREN>
    | <LEFT_BRACKET> [constantExpression()] <RIGHT_BRACKET>
    | <LEFT_PAREN> [parameterTypeList()] <RIGHT_PAREN>)
    (<LEFT_BRACKET> [constantExpression()] <RIGHT_BRACKET>
    | <LEFT_PAREN> [parameterTypeList()] <RIGHT_PAREN>)*
}


void identifierList() #void : {}
{
    identifier(false) (<COMMA> identifier(false))*
}


void statementList() #void : {}
{
    (statement())+
}
```

```
void statement() #void : {}
{
    (compoundStatement()
    | LOOKAHEAD(2) labeledStatement()
    | jumpStatement()
    | expressionStatement()
    | ifStatement()
    | switchStatement()
    | forStatement()
    | whileStatement()
    | doWhileStatement())
}


void compoundStatement() : {}
{
    <LEFT_BRACE> [LOOKAHEAD(declarationList()) declarationList()] [statementList()]
<RIGHT_BRACE>
}


void labeledStatement() : {}
{
    (<CASE> constantExpression() <COLON> statement()
    | <CDEFAULT> <COLON> statement()
    | identifier(false) <COLON> statement())
}


void jumpStatement() : {}
{
    (<GOTO> identifier(false) <SEMICOLON>
    | <BREAK> <SEMICOLON>
    | <CONTINUE> <SEMICOLON>
    | <RETURN> [expression(NodeType.JUMP_STMT_NODE)] <SEMICOLON>)
}


void expressionStatement() : {}
{
    [expression(NodeType.EXPRESSION_STMT_NODE)] <SEMICOLON>
}


void ifStatement() : {}
{
    <IF> <LEFT_PAREN> expression(NodeType.IF_STMT_NODE) <RIGHT_PAREN>
    statement()
    [LOOKAHEAD(2) <ELSE> statement()]
}


void switchStatement() : {}
{
    <SWITCH> <LEFT_PAREN> expression(NodeType.SWITCH_STMT_NODE) <RIGHT_PAREN>
    statement()
}


void forStatement() :
{
    Token t;
    int beginLine = 0;
    int beginColumn = 0;
}
{
    ((t = <FOR>)
    {
        beginLine = token.beginLine;
        beginColumn = token.beginColumn;
    }
```

```
    <LEFT_PAREN> [[typeSpecifier()] expression(NodeType.FOR_STMT_NODE)] <SEMICOLON>
[expression(NodeType.FOR_STMT_NODE)] <SEMICOLON> [expression(NodeType.FOR_STMT_NODE)]
<RIGHT_PAREN>
    statement())
    {
        jjtThis.setAttribute(NodeKey.BEGIN_LINE, new Integer(beginLine));
        jjtThis.setAttribute(NodeKey.BEGIN_COLUMN, new Integer(beginColumn));
        if ( t.specialToken != null ) {
            jjtThis.setAttribute(NodeKey.SPECIAL_TOKEN, t.specialToken.image);
        }
    }
}


void whileStatement() : {}
{
    <WHILE> <LEFT_PAREN> expression(NodeType.WHILE_STMT_NODE) <RIGHT_PAREN>
    statement()
}


void doWhileStatement() : {}
{
    <DO> statement()
    <WHILE> <LEFT_PAREN> expression(NodeType.DO_WHILE_STMT_NODE) <RIGHT_PAREN>
<SEMICOLON>
}


void expression(NodeType type) : {}
{
    (assignmentExpression() (<COMMA> assignmentExpression())*)
    {
        jjtThis.setAttribute(NodeKey.CALLED_FROM, type);
    }
}


void assignmentExpression() #assignmentExpression(>1) :
{
    boolean reduction = false;
    String reductionOp = null;
}
{
    (LOOKAHEAD(unaryExpression(true) (<EQUALS> | <STAR_EQUALS> | <SLASH_EQUALS> |
<PERCENT_EQUALS>
              | <PLUS_EQUALS> | <MINUS_EQUALS> | <LESS_LESS_EQUALS> |
<GREATER_GREATER_EQUALS>
              | <BITWISE_OR_EQUALS> | <AMP_EQUALS> | <BITWISE_XOR_EQUALS>))

                unaryExpression(true)
                (
                  <EQUALS>
                | <STAR_EQUALS> { reduction = true; reductionOp = "*"; }
                | <SLASH_EQUALS> { reduction = true; reductionOp = "/"; }
                | <PERCENT_EQUALS>
                | <PLUS_EQUALS> { reduction = true; reductionOp = "+"; }
                | <MINUS_EQUALS> { reduction = true; reductionOp = "-"; }
                | <LESS_LESS_EQUALS>
                | <GREATER_GREATER_EQUALS>
                | <BITWISE_OR_EQUALS> { reduction = true; reductionOp = "^"; }
                | <AMP_EQUALS> { reduction = true; reductionOp = "&"; }
                | <BITWISE_XOR_EQUALS> { reduction = true; reductionOp = "|"; }
                )
                assignmentExpression()
    | LOOKAHEAD(3) conditionalExpression())
    {
        jjtThis.setAttribute(NodeKey.REDUCTION, reduction);
        jjtThis.setAttribute(NodeKey.REDUCTION_OP, reductionOp);
    }
}
```

```
void constantExpression() #void : {}
{
    conditionalExpression()
}


void conditionalExpression() #conditionalExpression(>2) : {}
{
    logicalORExpression() [<QUESTION_MARK>
expression(NodeType.CONDITIONAL_EXPRESSION_NODE) <COLON> conditionalExpression()]
}


void logicalORExpression() #logicalORExpression(>1) : {}
{
    logicalANDExpression() [<LOGICAL_OR> logicalORExpression()]
}


void logicalANDExpression() #logicalANDExpression(>1) : {}
{
    bitwiseORExpression() [<LOGICAL_AND> logicalANDExpression()]
}


void bitwiseORExpression() #bitwiseORExpression(>1) : {}
{
    bitwiseXORExpression() [<BITWISE_OR> bitwiseORExpression()]
}


void bitwiseXORExpression() #bitwiseXORExpression(>1) : {}
{
    bitwiseANDExpression() [<BITWISE_XOR> bitwiseXORExpression()]
}


void bitwiseANDExpression() #bitwiseANDExpression(>1) : {}
{
    equalityExpression() [<AMPERSAND> bitwiseANDExpression()]
}


void equalityExpression() #equalityExpression(>1) : {}
{
    relationalExpression()
    [(<EQUALS_EQUALS> | <NOT_EQUALS>) equalityExpression()]
}


void relationalExpression() #relationalExpression(>1) : {}
{
    shiftExpression()
    [(<LESS_THAN> | <LESS_EQUALS> | <GREATER_THAN> | <GREATER_EQUALS>)
relationalExpression()]
}


void shiftExpression() #shiftExpression(>1) : {}
{
    additiveExpression()
    [(<SHIFT_LEFT> | <SHIFT_RIGHT>) shiftExpression()]
}


void additiveExpression() #additiveExpression(>1) : {}
{
    multiplicativeExpression()
    [(<PLUS> { jjtThis.setAttribute(NodeKey.OPERATOR, "+"); }
```

```
    | <MINUS> { jjtThis.setAttribute(NodeKey.OPERATOR, "-"); }
    ) additiveExpression()]
}


void multiplicativeExpression() #multiplicativeExpression(>1) : {}
{
    castExpression()
    [(<STAR> { jjtThis.setAttribute(NodeKey.OPERATOR, "*"); }
    | <SLASH> { jjtThis.setAttribute(NodeKey.OPERATOR, "/"); }
    | <PERCENT> { jjtThis.setAttribute(NodeKey.OPERATOR, "%"); }
    ) multiplicativeExpression()]
}


void castExpression() #castExpression(isCasted) :
{
    boolean isCasted = false;
}
{
    (LOOKAHEAD(<LEFT_PAREN> typeName() <RIGHT_PAREN> castExpression()) <LEFT_PAREN>
typeName() <RIGHT_PAREN> { isCasted = true; } castExpression()
    | unaryExpression(false))
}


boolean unaryExpression(boolean onLeftSideOfAssignment) #unaryExpression(isUnaried) :
{
    boolean isUnaried = false;
    boolean isIdentifier = false;
    boolean identifierBeingModified = false;
}
{
    ((isIdentifier = postfixExpression(onLeftSideOfAssignment))
    | (<PLUS_PLUS> | <MINUS_MINUS>)
    {
        isUnaried = true;
    }
    (isIdentifier = unaryExpression(onLeftSideOfAssignment))
    {
        identifierBeingModified = isIdentifier;
    }
    | (<AMPERSAND> | <STAR> | <PLUS> | <MINUS> | <BITWISE_NOT> | <LOGICAL_NOT>) {
isUnaried = true; } castExpression()
    | <SIZEOF> { isUnaried = true; } (LOOKAHEAD(unaryExpression(onLeftSideOfAssignment))
unaryExpression(onLeftSideOfAssignment) | <LEFT_PAREN> typeName() <RIGHT_PAREN>))
    {
        jjtThis.setAttribute(NodeKey.MODIFYING_IDENTIFIER, onLeftSideOfAssignment ||
identifierBeingModified);
        return isIdentifier;
    }
}


boolean postfixExpression(boolean onLeftSideOfAssignment) #postfixExpression(isPostfixed)
:
{
    boolean isPostfixed = false;
    boolean isIdentifier = false;
    boolean isArray = false;
    boolean identifierBeingModified = false;
    boolean reduction = false;
    String reductionOp = null;
}
{
    ((isIdentifier = primaryExpression(onLeftSideOfAssignment))
    ((<LEFT_BRACKET> { isArray = true; } expression(NodeType.POSTFIX_EXPRESSION_NODE)
<RIGHT_BRACKET>
    | <LEFT_PAREN> [expression(NodeType.POSTFIX_EXPRESSION_NODE)] <RIGHT_PAREN>
    | <DOT> identifier(true)
    | <POINTER_SELECT> identifier(true)
```

```
    | <PLUS_PLUS>
    {
        identifierBeingModified = isIdentifier;
        reduction = true;
        reductionOp = "+";
    }
    | <MINUS_MINUS>
    {
        identifierBeingModified = isIdentifier;
        reduction = true;
        reductionOp = "+";
    }
    )
    {
        isPostfixed = true;
    }
    )*)
    {
        jjtThis.setAttribute(NodeKey.MODIFYING_IDENTIFIER, (onLeftSideOfAssignment &&
(!isArray)) || identifierBeingModified);
        jjtThis.setAttribute(NodeKey.ARRAY_POSTFIXED, isArray);
        jjtThis.setAttribute(NodeKey.REDUCTION, reduction);
        jjtThis.setAttribute(NodeKey.REDUCTION_OP, reductionOp);
        return isIdentifier;
    }
}


boolean primaryExpression(boolean onLeftSideOfAssignment) #void :
{
    boolean isIdentifier = false;
}
{
    (identifier(onLeftSideOfAssignment) { isIdentifier = true; }
    | constant()
    | <LEFT_PAREN> expression(NodeType.PRIMARY_EXPRESSION_NODE) <RIGHT_PAREN>)
    {
        return isIdentifier;
    }
}


Token identifier(Boolean onLeftSideOfAssignment) :
{
    Token t;
}
{
    (t = <IDENTIFIER>)
    {
        jjtThis.setAttribute(NodeKey.NAME, t.image);
        jjtThis.setAttribute(NodeKey.MODIFYING_IDENTIFIER, onLeftSideOfAssignment);
        return t;
    }
}


void constant() #void : {}
{
    (integer() | real() | string() | character())
}


void integer() :
{
    Token t;
}
{
    (t = <INTEGER>)
    {
        jjtThis.setAttribute(NodeKey.VALUE, t.image);
    }
```

```
}


void real() :
{
    Token t;
}
{
    (t = <REAL>)
    {
        jjtThis.setAttribute(NodeKey.VALUE, t.image);
    }
}


void string() :
{
    Token t;
}
{
    (t = <STRING>)
    {
        jjtThis.setAttribute(NodeKey.VALUE, t.image);
    }
}


void character() :
{
    Token t;
}
{
    (t = <CHARACTER>)
    {
        jjtThis.setAttribute(NodeKey.VALUE, t.image);
    }
}
```

8.7. Appendix G: The Project Source Code

```
/*
 * Main.java
 *
 * Created on January 25, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql;


import java.io.File;
import java.io.FileNotFoundException;

import nql.cd.frontend.ParseException;


/**
 * The program main entry.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
```

```
public class Main {

    /**
     * The main entry to run the program
     * @param args command line arguments
     */
    public static void main(String[] args) {

        int numThreads = 1;            // default number of threads to run
        String flag = null;            // command line flag for different operations
        String inputFileName = null;   // input file name of a C language source file
        String outputFileName = null;  // output file name for OpenMP C source file

        // Validate user command usage
        if ( !isUsageValid(args) ) {
            displayUsage();
            return;
        }

        // Start GUI version
        if ( args.length == 0 ) {
            MainFrame.main(args);
        }

        // Start command line version
        else {

            // Parse user input parameters
            try {
                numThreads = Integer.parseInt(args[1]);
                if ( numThreads < Runtime.getRuntime().availableProcessors() ) {
                    numThreads = Runtime.getRuntime().availableProcessors();
                }
            }
            catch ( NumberFormatException e ) {
                displayUsage();
                return;
            }
            flag = args[2];
            inputFileName = args[3];
            if ( args.length == 5 ) {
                outputFileName = args[4];
            }

            // Carry out program tasks
            try {
                // Make sure the input source file exists
                if ( !(new File(inputFileName).exists()) ) {
                    throw new FileNotFoundException();
                }

                // Create and initialize the program auto parallelizer
                OpenMPAutoParallelizer ompParallelizer = new
OpenMPAutoParallelizer(numThreads);

                // Train the classifier with the input source file
                if ( flag.equalsIgnoreCase("-t") ) {
                    ompParallelizer.doTraining(inputFileName, null);
                }

                // Parallelize the sequential input source file
                else if ( flag.equalsIgnoreCase("-p") ) {
                    ompParallelizer.doParallelizing(inputFileName, outputFileName, null);
                }

                // Should not be here...
                else {
                    System.out.println("\n*** Internal system error, unknown program
task.\n");
                }
            }
```

```
            catch ( FileNotFoundException e ) {
                System.out.println("\n*** Input source file not found, please try
again.\n");
            }
            catch ( ParseException e ) {
                System.out.println("\n*** Failed to process input source file.");
                System.out.println("    Please verify syntax for correctness.\n");
            }
            catch ( Exception e ) {
                System.out.println("\n*** Internal System Error ***\n");
                System.out.println(e.toString());
            }
        }
    }

    /**
     * Validates program usage of command line arguments.
     * @param args the command line arguments
     * @return true if command line arguments are valid, false otherwise
     */
    private static boolean isUsageValid(String[] args) {

        // GUI version
        if ( args.length == 0 ) {
            return true;
        }

        // Command line version
        if ( args.length < 4 || args.length > 5 ) {
            return false;
        }
        boolean invalidFlag = !(args[0].equals("-n") && (args[2].equals("-t") ||
args[2].equals("-p")));
        boolean invalidOption = args[2].equals("-t") && args.length != 4;
        return !(invalidFlag || invalidOption);
    }

    /**
     * Displays program usage.
     */
    private static void displayUsage() {
        System.out.println("USAGE: program-name (w/o input, GUI version)");
        System.out.println("       program-name -n num-threads -t input-file-name.c");
        System.out.println("       program-name -n num-threads -p input-file-name.c
[output-file-name.c]");
        System.out.println("       -n: specifies number of threads to run");
        System.out.println("       -t: trains the classifier with the input file");
        System.out.println("       -p: parallelizes the input file and sends the result
to the");
        System.out.println("           output file if it is present; standard output
otherwise");
    }
}
```

```
/*
 * MainFrame.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql;
```

```
import java.awt.EventQueue;
import java.awt.Font;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.File;
import java.io.FileNotFoundException;
import java.awt.Rectangle;
import javax.swing.JTabbedPane;
import javax.swing.ScrollPaneConstants;
import javax.swing.SwingConstants;

import nql.cd.frontend.ParseException;

import org.syntax.jedit.JEditTextArea;
import org.syntax.jedit.tokenmarker.CTokenMarker;


/**
 * The GUI version of the program.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class MainFrame extends JFrame implements ActionListener {

    // Default number of threads to run
    private int numThreads = 1;

    // C or OpenMP input source file
    private String inputSourceFile;

    // The program auto parallelizer
    private OpenMPAutoParallelizer ompParallelizer;

    // Swing components
    private JPanel contentPane;
    private JPanel inputPane;
    private JTabbedPane tabbedPane;
    private JTextField txtInputSourceFile;
    private JTextField txtNumberOfThreads;
    private JRadioButton rdbtnTraining;
    private JRadioButton rdbtnParallelizing;
    private JTextArea[] txtAreaProgramOutputs = new JTextArea[4];
    private JEditTextArea txtAreaOpenMPFile = new JEditTextArea();
    private JFileChooser fileChooser = new JFileChooser();
    private final ButtonGroup buttonGroup = new ButtonGroup();

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MainFrame frame = new MainFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the frame.
 */
public MainFrame() {
    setTitle("Automatic OpenMP Parallelizer");
    setResizable(false);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 800, 600);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(null);
    setContentPane(contentPane);

    inputPane = new JPanel();
    inputPane.setLocation(0, 0);
    inputPane.setSize(800, 90);
    inputPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    inputPane.setLayout(null);
    contentPane.add(inputPane);

    JLabel lblInputSourceFile = new JLabel("Input source file:");
    lblInputSourceFile.setBounds(10, 11, 120, 20);
    inputPane.add(lblInputSourceFile);

    JLabel lblNumberOfThreads = new JLabel("Number of threads:");
    lblNumberOfThreads.setBounds(10, 42, 120, 20);
    inputPane.add(lblNumberOfThreads);

    String logoText =
        "<html><pre><font color=\"#C85A17\">"          +
        "|-------------------------------|<br>"        +
        "| OPENMP AUTOMATIC PARALLELIZER |<br>"        +
        "|        <i>By Nam Q. Lam</i>          |<br>" +
        "|    Master Project @ SJSU      |<br>"        +
        "|-------------------------------|"            +
        "</font></pre></html>";

    JLabel lblLogo = new JLabel(logoText);
    lblLogo.setFont(new Font("Monospaced", Font.PLAIN, 11));
    lblLogo.setHorizontalAlignment(SwingConstants.CENTER);
    lblLogo.setBounds(550, 5, 240, 80);
    inputPane.add(lblLogo);

    txtInputSourceFile = new JTextField();
    txtInputSourceFile.setBounds(140, 11, 258, 20);
    inputPane.add(txtInputSourceFile);
    txtInputSourceFile.setColumns(10);

    txtNumberOfThreads = new JTextField();
    txtNumberOfThreads.setBounds(140, 42, 40, 20);
    inputPane.add(txtNumberOfThreads);
    txtNumberOfThreads.setColumns(10);

    rdbtnTraining = new JRadioButton("Training");
    rdbtnTraining.addActionListener(this);
    buttonGroup.add(rdbtnTraining);
    rdbtnTraining.setSelected(true);
    rdbtnTraining.setBounds(207, 41, 89, 23);
    inputPane.add(rdbtnTraining);

    rdbtnParallelizing = new JRadioButton("Parallelizing");
    rdbtnParallelizing.addActionListener(this);
    buttonGroup.add(rdbtnParallelizing);
    rdbtnParallelizing.setBounds(298, 41, 100, 23);
    inputPane.add(rdbtnParallelizing);
```

```
        JButton btnChooseInputSourceFile = new JButton("Choose File");
        btnChooseInputSourceFile.addActionListener(this);
        btnChooseInputSourceFile.setBounds(419, 10, 100, 23);
        inputPane.add(btnChooseInputSourceFile);

        JButton btnStart = new JButton("Start");
        btnStart.addActionListener(this);
        btnStart.setBounds(419, 41, 100, 23);
        inputPane.add(btnStart);

        tabbedPane = new JTabbedPane(JTabbedPane.TOP);
        tabbedPane.setBounds(10, 100, 775, 465);
        contentPane.add(tabbedPane);

        txtAreaProgramOutputs[0] = new JTextArea();
        txtAreaProgramOutputs[1] = new JTextArea();
        txtAreaProgramOutputs[2] = new JTextArea();
        txtAreaProgramOutputs[3] = new JTextArea();

        txtAreaProgramOutputs[0].setEditable(false);
        txtAreaProgramOutputs[1].setEditable(false);
        txtAreaProgramOutputs[2].setEditable(false);
        txtAreaProgramOutputs[3].setEditable(false);

        txtAreaProgramOutputs[0].setLineWrap(false);
        txtAreaProgramOutputs[1].setLineWrap(false);
        txtAreaProgramOutputs[2].setLineWrap(false);
        txtAreaProgramOutputs[3].setLineWrap(false);

        txtAreaProgramOutputs[0].setFont(new Font("Monospaced", Font.PLAIN, 12));
        txtAreaProgramOutputs[1].setFont(new Font("Monospaced", Font.PLAIN, 12));
        txtAreaProgramOutputs[2].setFont(new Font("Monospaced", Font.PLAIN, 12));
        txtAreaProgramOutputs[3].setFont(new Font("Monospaced", Font.PLAIN, 12));

        JScrollPane pScroll0 = new JScrollPane(txtAreaProgramOutputs[0],
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        JScrollPane pScroll1 = new JScrollPane(txtAreaProgramOutputs[1],
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        JScrollPane pScroll2 = new JScrollPane(txtAreaProgramOutputs[2],
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        JScrollPane pScroll3 = new JScrollPane(txtAreaProgramOutputs[3],
ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);

        // Enable C syntax highlight for the auto-generated OpenMP file;
        // otherwise, pScroll2 can be used instead in the tabbedPane.add() below
        txtAreaOpenMPFile.setEditable(false);
        txtAreaOpenMPFile.setCaretBlinkEnabled(false);
        txtAreaOpenMPFile.setTokenMarker(new CTokenMarker());
        txtAreaOpenMPFile.setText("");

        tabbedPane.add("Extracted Features", pScroll0);
        tabbedPane.add("New Training Records", pScroll1);
        tabbedPane.add("Generated OpenMP", txtAreaOpenMPFile);
        tabbedPane.add("Parse Tree (AST)", pScroll3);
        tabbedPane.setEnabledAt(2, false);
    }

    public void actionPerformed(ActionEvent event) {
        String cmd = event.getActionCommand();

        // Disable appropriate tabs if Training option is selected
        if ( cmd.equals("Training") ) {
            tabbedPane.setEnabledAt(1, true);
            tabbedPane.setEnabledAt(2, false);
        }
```

```
        // Disable appropriate tabs if Parallelizing option is selected
        else if ( cmd.equals("Parallelizing") ) {
            tabbedPane.setEnabledAt(2, true);
            tabbedPane.setEnabledAt(1, false);
        }

        // Retrieve C or OpenMP input source file
        else if ( cmd.equals("Choose File") ) {
            fileChooser.setFileFilter(new SimpleFileFilter("C source file (*.c)", "c"));
            int status = fileChooser.showOpenDialog(MainFrame.this);
            if ( status == JFileChooser.APPROVE_OPTION ) {
                File selectedFile = fileChooser.getSelectedFile();
                txtInputSourceFile.setText(selectedFile.getAbsolutePath());
            }
        }

        // Carry out program tasks
        else if ( cmd.equals("Start") ) {
            try {
                // Input source file
                inputSourceFile = txtInputSourceFile.getText();
                if ( !(new File(inputSourceFile).exists()) ) {
                    throw new FileNotFoundException();
                }

                // Number of threads
                numThreads = Integer.parseInt(txtNumberOfThreads.getText());
                if ( numThreads < Runtime.getRuntime().availableProcessors() ) {
                    numThreads = Runtime.getRuntime().availableProcessors();
                }

                // Create and initialize the program auto parallelizer
                ompParallelizer = new OpenMPAutoParallelizer(numThreads);

                // Train the classifier with the input source file
                if ( rdbtnTraining.isSelected() ) {
                    ompParallelizer.doTraining(inputSourceFile, txtAreaProgramOutputs);
                }

                // Parallelize the sequential input source file
                else if ( rdbtnParallelizing.isSelected() ) {
                    String ompFile = ompParallelizer.doParallelizing(inputSourceFile,
null, txtAreaProgramOutputs);
                    txtAreaOpenMPFile.setText(ompFile);
                }

                // Should not be here...
                else {
                    JOptionPane.showMessageDialog(
                            null,
                            "Internal system error, unknown program task.",
                            "Internal System Error",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
            catch ( FileNotFoundException e ) {
                JOptionPane.showMessageDialog(
                        null,
                        "Input source file not found, please try again.",
                        "File Not Found Error",
                        JOptionPane.ERROR_MESSAGE);
            }
            catch ( NumberFormatException e ) {
                JOptionPane.showMessageDialog(
                        null,
                        "Please use digit(s) for number of threads.",
                        "Number of Threads Error",
                        JOptionPane.ERROR_MESSAGE);
            }
            catch ( ParseException e ) {
                JOptionPane.showMessageDialog(
```

```
                        null,
                        "Failed to process input source file.\nPlease verify syntax for
correctness.",
                        "Syntax Error",
                        JOptionPane.ERROR_MESSAGE);
            }
            catch ( Exception e ) {
                JOptionPane.showMessageDialog(
                        null,
                        e.toString(),
                        "Internal Error",
                        JOptionPane.ERROR_MESSAGE);
            }
        }

        // Should not be here...
        else {
            JOptionPane.showMessageDialog(
                    null,
                    "Internal system error, unknown action command.",
                    "Internal System Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

```
/*
 * OpenMPAutoParallelizer.java
 *
 * Created on March 11, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql;


import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.swing.JTextArea;

import nql.cd.backend.ForLoopExtractorVisitor;
import nql.cd.frontend.CParser;
import nql.cd.frontend.SimpleNode;
import nql.cd.intermediate.ForLoopNode;
import nql.cd.intermediate.ForLoopTree;
import nql.cd.intermediate.Variable;
import nql.exceptions.LoopCarriedDependenceException;
import nql.ml.classifiers.OpenMPForScheduleClassifier;
import nql.utils.ObjectSerializer;
import nql.utils.SourceFile;
```

```
/**
 * The OpenMP automatic parallelizer.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class OpenMPAutoParallelizer {

    /** Default number of threads for the program. */
    private int numberOfThreads = Runtime.getRuntime().availableProcessors();

    /** The classifier used to predict schedule type. */
    private OpenMPForScheduleClassifier theClassifier = null;

    /** Name of the file used to serialize the classifier. */
    private String classifierObjectFile = "obj\\IBkScheduleClassifier.obj";

    /** Is GUI version selected? */
    private boolean isGUIVersion = false;

    /** String buffers to hold program outputs. */
    private StringBuffer[] programOutputs = new StringBuffer[4];

    /**
     * Constructor.
     */
    public OpenMPAutoParallelizer(int numThreads) throws Exception {

        // Number of threads on target system
        this.numberOfThreads = numThreads;

        // Make sure the object file exists, create one otherwise
        File objectFile = new File(classifierObjectFile);
        if ( !objectFile.exists() ) {
            objectFile.createNewFile();
        }

        // Retrieve the instance based classifier from the object file
        theClassifier =
(OpenMPForScheduleClassifier)ObjectSerializer.readObject(classifierObjectFile);

        // Technically, this should happen ONCE when the object file
        // does not exist and needs to be created.
        if ( theClassifier == null ) {
            theClassifier = new OpenMPForScheduleClassifier();
            ObjectSerializer.writeObject(theClassifier, classifierObjectFile);
        }

        // String buffers to hold program outputs for GUI version
        programOutputs[0] = new StringBuffer(); // extracted features
        programOutputs[1] = new StringBuffer(); // new training records
        programOutputs[2] = new StringBuffer(); // generated OpenMP
        programOutputs[3] = new StringBuffer(); // parse tree (AST)
    }

    /**
     * Trains the classifier with an OpenMP file (a new instance/record).
     * @param inputFileName name of OpenMP file used to train the classifier
     * @param guiOutputs text ares to hold program outputs for GUI version
     * @throws Exception an exception that is propagated to the caller
     */
    public void doTraining(String inputFileName, JTextArea[] guiOutputs) throws Exception
{

        // Is this being called from GUI version?
        isGUIVersion = (guiOutputs != null) ? true : false;

        // Print the program header
        if ( !isGUIVersion ) {
            printProgramHeader(0);
            System.out.println("\nOpenMP source file: " + inputFileName + "\n");
```

```
        }

        // Extract all FOR loops from the input file
        ArrayList<ForLoopTree> forLoopTreeList = extractForLoops(inputFileName);

        // Display information of each extracted FOR loop
        printForLoopInfo(forLoopTreeList);

        // Use extracted information/features to train the classifier
        trainTheClassifier(forLoopTreeList);

        // If called from GUI version, send outputs to text areas
        if ( isGUIVersion ) {
            guiOutputs[0].setText(programOutputs[0].toString());
            guiOutputs[1].setText(programOutputs[1].toString());
            guiOutputs[3].setText(programOutputs[3].toString());
        }
        else {
            System.out.print(programOutputs[0].toString());
            System.out.print(programOutputs[1].toString());
            System.out.print(programOutputs[3].toString());
        }
    }

    /**
     * Performs OpenMP parallelization on the input file and the result
     * is saved to the output file; standard output if output file is null.
     * @param inputFileName name of input file to be parallelized
     * @param outputFileName name of the OpenMP output file
     * @param guiOutputs text ares to hold program outputs for GUI version
     * @throws Exception an exception that is propagated to the caller
     * @return auto-generated OpenMP file as a String
     */
    public String doParallelizing(String inputFileName, String outputFileName,
JTextArea[] guiOutputs) throws Exception {

        // Is this being called from GUI version?
        isGUIVersion = (guiOutputs != null) ? true : false;

        // Print the program header
        if ( !isGUIVersion ) {
            printProgramHeader(1);
            System.out.println("\nOpenMP source file: " + inputFileName + "\n");
        }

        // Extract all FOR loops from the input file
        ArrayList<ForLoopTree> forLoopTreeList = extractForLoops(inputFileName);

        // Display information of each extracted FOR loop
        printForLoopInfo(forLoopTreeList);

        // Use extracted information/features to generate OpenMP file
        SourceFile ompFile = generateOpenMPFile(forLoopTreeList, inputFileName);

        // Command line version, output parallelized OpenMP program to file;
        // otherwise, send outputs to either console or GUI text areas
        programOutputs[2].append("\nResulted OpenMP C program\n");
        programOutputs[2].append("-----------------------------------------------------
--------------------------\n\n");
        if ( outputFileName != null ) {
            assert(guiOutputs == null);
            programOutputs[2].append("File: " + outputFileName + "\n\n");
            ompFile.printTo(outputFileName);
        }
        else {
            ompFile.printTo(programOutputs[2]);
        }

        // If called from GUI version, send outputs to text areas;
        // otherwise, send outputs to console
        if ( isGUIVersion ) {
```

```
            guiOutputs[0].setText(programOutputs[0].toString());
            guiOutputs[2].setText(programOutputs[2].toString());
            guiOutputs[3].setText(programOutputs[3].toString());
        }
        else {
            System.out.print(programOutputs[0].toString());
            System.out.print(programOutputs[2].toString());
            System.out.print(programOutputs[3].toString());
        }

        // Return the auto-generated OpenMP file as a String
        return ompFile.toString();
    }

    /**
     * Extracts all FOR loops from the input file.
     * @param inputFileName name of the input C source file
     * @return list of extracted ForLoopTree(s)
     * @throws Exception an exception that is propagated to the caller
     */
    private ArrayList<ForLoopTree> extractForLoops(String inputFileName) throws Exception
{

        BufferedReader reader = new BufferedReader(new FileReader(inputFileName));

        // Parse the input C source file
        // and build the corresponding parse tree
        CParser parser = new CParser(reader);
        SimpleNode rootNode = parser.parse();

        // Print out the parse tree
        printParseTree(rootNode);

        // Not to disturbing the C grammar of #define, definitions are extracted
        // here because only interested in this form: #define <IDENTIFIER> <INTEGER>
        HashMap<String, Integer> poundDefs = extractPoundDefs(inputFileName);

        // Traversing the parse tree
        ForLoopExtractorVisitor forLoopVisitor = new ForLoopExtractorVisitor(poundDefs);
        rootNode.jjtAccept(forLoopVisitor, null);

        // Retrieve and return list of extracted ForLoopTree(s) from the visitor
        ArrayList<ForLoopTree> forLoopTreeList = forLoopVisitor.getForLoopTreeList();
        return forLoopTreeList;
    }

    /**
     * Prints the parse tree.
     * @param rootNode root node of the tree
     */
    private void printParseTree(SimpleNode rootNode) {
        programOutputs[3].append("\nThe parse tree (AST)\n");
        programOutputs[3].append("------------------------------------------------------------
-------------------------\n\n");
        rootNode.dump("", programOutputs[3]);
        programOutputs[3].append("\n");
    }

    /**
     * Prints extracted information/features of each FOR loop.
     * @param forLoopTreeList the list of extracted ForLoopTree(s)
     */
    private void printForLoopInfo(ArrayList<ForLoopTree> forLoopTreeList) {

        programOutputs[0].append("\nFeatures found in each FOR loop\n");
        programOutputs[0].append("------------------------------------------------------------
-------------------------\n\n");

        for ( int i = 0; i < forLoopTreeList.size(); i++ ) {
            ForLoopTree forLoopTree = (ForLoopTree)forLoopTreeList.get(i);
            Enumeration e = forLoopTree.breadthFirstTraversal();
```

```
            while ( e.hasMoreElements() ) {
                ForLoopNode forLoopNode = (ForLoopNode)e.nextElement();
                String pragma = forLoopNode.getOpenMPPragma();
                programOutputs[0].append("FOR loop at line " + forLoopNode.getBeginLine()
+
                                " column " + forLoopNode.getBeginColumn() + ":" +
                                "\nOpenMP directive: " + (pragma == null ? "None" :
pragma) +
                                "\n    Nested level = " + forLoopNode.getNestedLevel() +
                                "\n    Nesting levels = " + forLoopNode.getNestingLevels()
+
                                "\n    Number of iterations = " +
forLoopNode.getNumberOfIterations() +
                                "\n    Number of operations = " +
forLoopNode.getOperationsCount() +
                                "\n    Number of children = " +
forLoopNode.getNumberOfChildren() +
                                "\n    Number of threads = " + numberOfThreads +
                                "\n    Number of arrays = " +
forLoopNode.getNumberOfArrayTypedVariables() +
                                "\n    Total number of variables = " +
forLoopNode.getNumberOfVariables());

                for ( Variable v : forLoopNode.getSetOfVariables() ) {
                    programOutputs[0].append("\n    " + v + " is " + (v.isReadOnly() ?
"accessed only" : "being updated"));
                }
                programOutputs[0].append("\n\n");
            }
        }
    }

    /**
     * Generate parallel OpenMP file from the input file
     * using extracted information/features.
     * @param forLoopTreeList the list of extracted ForLoopTree(s)
     * @param inputFileName name of input file to be parallelized
     * @return SourceFile object representing contents of the result OpenMP file
     */
    private SourceFile generateOpenMPFile(ArrayList<ForLoopTree> forLoopTreeList, String
inputFileName)
        throws Exception {

        String metaData1  = "/*";
        String metaData2  = " * ---------------------------------------------";
        String metaData3  = " * Automatic OpenMP Parallelization              ";
        String metaData4  = " * by Nam Q. Lam's MasterProject @ SJSU          ";
        String metaData5a = " * => This loop has loop-carried dependence      ";
        String metaData5b = " * => It cannot be auto-parallelized             ";
        String metaData5c = " * => This loop has been previously parallelized ";
        String metaData5d = " * => No change has been made to this loop       ";
        String metaData5e = " * Do not edit this auto-generated loop construct";
        String metaData6  = " * ---------------------------------------------";
        String metaData7  = " */";

        SourceFile src = new SourceFile(inputFileName);
        int linesAdded = 0;

        // For each nested loop, build OpenMP pragma for
        // the most outer loop. See notes in generateOpenMPPragma(...)
        for ( int i = 0; i < forLoopTreeList.size(); i++ ) {

            ForLoopTree forLoopTree = (ForLoopTree)forLoopTreeList.get(i);
            boolean loopCarriedDependence = false;
            String ompPragma = null;

            // Retrieve OpenMP #pragma
            try { ompPragma = generateOpenMPPragma(forLoopTree); }
            catch ( LoopCarriedDependenceException e ) {
                loopCarriedDependence = true;
            }
```

```
    ForLoopNode mostOuterForLoop = (ForLoopNode)forLoopTree.getRootNode();
    int row = mostOuterForLoop.getBeginLine();
    int col = mostOuterForLoop.getBeginColumn();

    // Loop-carried dependence found? If so,
    // print appropriate information to indicate so
    if ( loopCarriedDependence == true ) {
        src.addLine(linesAdded + row - 1, col - 1, metaData1);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData2);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData3);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData4);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData5a);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData5b);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData6);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData7);
        linesAdded++;
    }
    // Already been previously parallelized using OpenMP,
    // print appropriate information to indicate so
    else if ( ompPragma == null ) {
        src.addLine(linesAdded + row - 2, col - 1, metaData1);
        linesAdded++;
        src.addLine(linesAdded + row - 2, col - 1, metaData2);
        linesAdded++;
        src.addLine(linesAdded + row - 2, col - 1, metaData3);
        linesAdded++;
        src.addLine(linesAdded + row - 2, col - 1, metaData4);
        linesAdded++;
        src.addLine(linesAdded + row - 2, col - 1, metaData5c);
        linesAdded++;
        src.addLine(linesAdded + row - 2, col - 1, metaData5d);
        linesAdded++;
        src.addLine(linesAdded + row - 2, col - 1, metaData6);
        linesAdded++;
        src.addLine(linesAdded + row - 2, col - 1, metaData7);
        linesAdded++;
    }
    // Just been parallelized via this program auto-parallelizer,
    // print resulting information
    else {
        src.addLine(linesAdded + row - 1, col - 1, metaData1);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData2);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData3);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData4);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData5e);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData6);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, metaData7);
        linesAdded++;
        src.addLine(linesAdded + row - 1, col - 1, ompPragma);
        linesAdded++;
    }
}

// insert OpenMP header omp.h
insertOpenMPHeaderFile(src);

// return result OpenMP parallelized source file
```

```
        return src;
    }

    /**
     * Trains the classifier using information/features from the list
     * of extracted ForLoopTree(s).
     * @param forLoopTreeList the list of extracted ForLoopTree(s)
     */
    private void trainTheClassifier(ArrayList<ForLoopTree> forLoopTreeList) throws
Exception {
        boolean foundAtLeastOneTrainingRecord = false;

        // Header for the training records/statistics
        programOutputs[1].append("\nTraining records added to the model\n");
        programOutputs[1].append("-----------------------------------------------------------
---------------------------\n\n");
        programOutputs[1].append("Row  Col  Threads  Iterations  Nesting-levels
Complexity   Class (Schedule-type)\n");

        // For each nested loop, use only OpenMP pragma of
        // the most outer loop to train the classifier
        for ( int i = 0; i < forLoopTreeList.size(); i++ ) {

            ForLoopTree forLoopTree = forLoopTreeList.get(i);
            ForLoopNode mostOuterForLoop = (ForLoopNode)forLoopTree.getRootNode();

            // No OpenMP pragma specified for this outer loop
            if ( mostOuterForLoop.getOpenMPPragma() == null ) {
                continue;
            }
            foundAtLeastOneTrainingRecord = true;

            // features/attributes of this loop (training instance)
            // (number of threads is input from the user or default value is used)
            int numberOfIterations = mostOuterForLoop.getNumberOfIterations();
            int nestingLevels = mostOuterForLoop.getNestingLevels();
            double complexityUnits = computeComplexity(forLoopTree);
            String classLabel = retrieveScheduleType(mostOuterForLoop);
            String[] attributeValues = { String.valueOf(numberOfThreads),
                                         String.valueOf(numberOfIterations),
                                         String.valueOf(nestingLevels),
                                         String.valueOf(complexityUnits) };

            // add new record for training
            theClassifier.addTrainingInstance(attributeValues, classLabel);
            theClassifier.saveTrainingDataSetToArffFile();

            //
            // Display training records/statistics
            //
            int row = mostOuterForLoop.getBeginLine();
            int col = mostOuterForLoop.getBeginColumn();
            programOutputs[1].append(String.format("%3d  %3d  %7d  %10d  %14d  %12.0f
%s\n",
                    row, col, numberOfThreads, numberOfIterations,
                    nestingLevels, complexityUnits, classLabel));
        }

        // Not at least one training record found
        if ( foundAtLeastOneTrainingRecord == false ) {
            programOutputs[1].append("\n===> No training record has been found.\n");
        }

        // MUST serialize/save the classifier!
        ObjectSerializer.writeObject(theClassifier, classifierObjectFile);
    }

    /**
     * Generates an OpenMP pragma for the most outer FOR loop.
     * @param forLoopTree a tree of extracted FOR loops (a nested loop)
     * @return the complete OpenMP pragma
```

```
     */
    private String generateOpenMPPragma(ForLoopTree forLoopTree) throws Exception {
        //
        // Generates OpenMP pragma for the most outer loop
        //
        ForLoopNode mostOuterForLoop = (ForLoopNode)forLoopTree.getRootNode();

        // Return null to indicate that this FOR loop has
        // already been previously parallelized using OpenMP
        if ( mostOuterForLoop.getOpenMPPragma() != null ) {
            return null;
        }
        // Throw the exception to indicate that this FOR loop contains
        // a loop-carried dependence, cannot be parallelized
        else if ( mostOuterForLoop.hasLoopCarriedDependence() == true ) {
            throw new LoopCarriedDependenceException("Loop-carried dependence found.");
        }
        // To be conservative, inner loops are also checked
        // for loop-carried dependence
        else {
            Enumeration e = forLoopTree.breadthFirstTraversal();
            while ( e.hasMoreElements() ) {
                ForLoopNode forLoopNode = (ForLoopNode)e.nextElement();
                if ( forLoopNode.hasLoopCarriedDependence() == true ) {
                    throw new LoopCarriedDependenceException("Loop-carried dependence
found.");
                }
            }
        }

        // OpenMP directive for the most outer loop
        String pragma = "#pragma omp parallel for";
        String numThreads = "num_threads(" + numberOfThreads + ")";
        String scheduleClause = "schedule(";
        String defaultClause = "default(none)";
        String privateClause = "private(";
        String sharedClause = "shared(";
        StringBuffer reductionClauses = new StringBuffer("");

        // features/attributes of this nested loop (unknown-schedule-type instance)
        // (number of threads is input from the user or default value is used)
        int numberOfIterations = mostOuterForLoop.getNumberOfIterations();
        int nestingLevels = mostOuterForLoop.getNestingLevels();
        double complexityUnits = computeComplexity(forLoopTree);
        String[] attributeValues = { String.valueOf(numberOfThreads),
                                     String.valueOf(numberOfIterations),
                                     String.valueOf(nestingLevels),
                                     String.valueOf(complexityUnits) };

        // Predict schedule type for this nested loop (unknown instance)
        String scheduleType = theClassifier.classifyNewInstance(attributeValues);

        // The classifier learns from the programs it parallelizes,
        // so it is getting smarter the more program it parallelizes.
        theClassifier.addTrainingInstance(attributeValues, scheduleType);
        theClassifier.saveTrainingDataSetToArffFile();
        ObjectSerializer.writeObject(theClassifier, classifierObjectFile);

        // build schedule clause
        scheduleClause += scheduleType + ")";

        // Build lists of private/shared/reduction variables for
        // the most outer loop, which include those from inner loops
        ArrayList<Variable> allPrivateVariables = new ArrayList<Variable>();
        ArrayList<Variable> allSharedVariables = new ArrayList<Variable>();
        ArrayList<Variable> allReductionVariables = new ArrayList<Variable>();
        Enumeration e = forLoopTree.breadthFirstTraversal();
        while ( e.hasMoreElements() ) {
            ForLoopNode forLoopNode = (ForLoopNode)e.nextElement();
            ArrayList<Variable> privateVariables = forLoopNode.getPrivateVariables();
            ArrayList<Variable> sharedVariables = forLoopNode.getSharedVariables();
```

```
            ArrayList<Variable> reductionVariables = forLoopNode.getReductionVariables();

            // build list of reduction variables
            for ( int i = 0; i < reductionVariables.size(); i++ ) {
                Variable v = reductionVariables.get(i);

                // add v to the list if it is not in the list
                if ( !allReductionVariables.contains(v) ) {
                    allReductionVariables.add(v);
                }
            }

            // build list of private variables
            for ( int j = 0; j < privateVariables.size(); j++ ) {
                Variable v = privateVariables.get(j);

                // add v to the list if it is not in the lists
                if ( !allPrivateVariables.contains(v) &&
!allReductionVariables.contains(v) ) {
                    allPrivateVariables.add(v);
                }
            }

            // build list of shared variables
            for ( int k = 0; k < sharedVariables.size(); k++ ) {
                Variable v = sharedVariables.get(k);

                // add v to the list if it is not in the lists
                if ( !allSharedVariables.contains(v) && !allPrivateVariables.contains(v)
&& !allReductionVariables.contains(v) ) {
                    allSharedVariables.add(v);
                }
            }
        }

        // Reduction variables take over shared variables, i.e.,
        // remove any shared variable if it is already in the reduction list
        for ( int i = 0; i < allSharedVariables.size(); i++ ) {
            Variable v = allSharedVariables.get(i);
            if ( allReductionVariables.contains(v) ) {
                allSharedVariables.remove(i);
            }
        }

        // Reduction variables also take over private variables, i.e.,
        // remove any private variable if it is already in the reduction list
        for ( int i = 0; i < allPrivateVariables.size(); i++ ) {
            Variable v = allPrivateVariables.get(i);
            if ( allReductionVariables.contains(v) ) {
                allPrivateVariables.remove(i);
            }
        }

        // Private variables take over shared variables, i.e.,
        // remove any shared variable if it is already in the private list
        for ( int i = 0; i < allSharedVariables.size(); i++ ) {
            Variable v = allSharedVariables.get(i);
            if ( allPrivateVariables.contains(v) ) {
                allSharedVariables.remove(i);
            }
        }

        // build private clause
        for ( int i = 0; i < allPrivateVariables.size(); i++ ) {
            if ( i != 0 ) privateClause += ", ";
            privateClause += allPrivateVariables.get(i).getName();
        }
        privateClause += ")";

        // build shared clause
        for ( int i = 0; i < allSharedVariables.size(); i++ ) {
```

```
            if ( i != 0 ) sharedClause += ", ";
            sharedClause += allSharedVariables.get(i).getName();
        }
        sharedClause += ")";

        // build reduction clause
        for ( int i = 0; i < allReductionVariables.size(); i++ ) {
            Variable var = allReductionVariables.get(i);
            assert(var.getReductionOp() != null);
            reductionClauses.append("reduction(" + var.getReductionOp() + ": " +
var.getName() + ") ");
        }

        //
        // compose all clauses into a complete pragma
        //
        pragma += " " + numThreads;
        pragma += " " + scheduleClause;
        pragma += " " + defaultClause;
        if ( allPrivateVariables.size() > 0 ) {
            pragma += " " + privateClause;
        }
        if ( allSharedVariables.size() > 0 ) {
            pragma += " " + sharedClause;
        }
        pragma += " " + reductionClauses.toString();

        // return the complete pragma for the most outer loop
        return pragma;
    }

    /**
     * Computes loop complexity.
     * @param forLoopTree a tree of extracted FOR loops (a nested loop)
     * @return the loop complexity
     */
    private double computeComplexity(ForLoopTree forLoopTree) {

        // Current computing model for this complexity
        // is based on loop iterations, units/steps of operations,
        // and whether the loops has imbalanced workload or not

        ForLoopNode mostOuterForLoop = (ForLoopNode)forLoopTree.getRootNode();
        String iterationVariableName = mostOuterForLoop.getIterationVariableName();

        boolean foundImbalancedLoop = false;
        double complexityUnits = 1;
        Enumeration e = forLoopTree.breadthFirstTraversal();
        while ( e.hasMoreElements() ) {
            ForLoopNode forLoopNode = (ForLoopNode)e.nextElement();
            int iterations = forLoopNode.getNumberOfIterations();
            int operations = forLoopNode.getOperationsCount();
            if ( iterations != 0 && operations != 0 ) {
                complexityUnits *= (iterations * operations);
            }

            String numberIterationsVariableName =
forLoopNode.getNumberIterationsVariableName();
            if ( iterationVariableName != null && numberIterationsVariableName != null )
{
                if ( iterationVariableName.equals(numberIterationsVariableName) ) {
                    foundImbalancedLoop = true;
                }
            }

            if ( foundImbalancedLoop ) {
                if ( operations != 0 ) {
                    complexityUnits *= operations;
                }
            }
        }
```

```
        return complexityUnits;
    }

    /**
     * Retrieves the loop construct schedule type
     * @param loop ForLoopNode object of an OpenMP FOR loop construct
     * @return the loop construct schedule type
     */
    private String retrieveScheduleType(ForLoopNode forLoopNode) throws Exception {
        String ompPragma = forLoopNode.getOpenMPPragma();
        String defaultScheduleType = "auto";

        // Is the #pragma specified for this loop?
        if ( ompPragma == null ) {
            throw new Exception("Missing OpenMP #pragma, cannot retrieve schedule
clause.");
        }

        // Is schedule clause included? If so, get the type
        if ( ompPragma.indexOf("schedule") > 0 ) {
            if ( ompPragma.indexOf("static") > 0 ) return "static";
            if ( ompPragma.indexOf("dynamic") > 0 ) return "dynamic";
            if ( ompPragma.indexOf("guided") > 0 ) return "guided";
            if ( ompPragma.indexOf("runtime") > 0 ) return "runtime";
        }

        // default is "auto"
        return defaultScheduleType;
    }

    /**
     * Inserts the OpenMP header omp.h into the input SourceFile
     * @param src the input SourceFile
     */
    private void insertOpenMPHeaderFile(SourceFile src) {
        boolean ompHeaderExists = false;
        ArrayList<String> lines = src.getLines();
        for ( int i = 0; i < lines.size(); i++ ) {
            String line = lines.get(i);
            if ( line.indexOf("#include") >= 0 && line.indexOf("<omp.h>") >= 0 ) {
                ompHeaderExists = true;
                break;
            }
        }

        // insert omp.h only if it is not included yet
        if ( !ompHeaderExists ) {
            src.addLine(0, 0, "#include <omp.h>");
            src.addLine(1, 0, "");
        }
    }

    /**
     * Prints the program header.
     * @param op operation type, 0 = training, 1 = parallelizing
     */
    private void printProgramHeader(int op) {
        System.out.println("-----------------------------------------------");
        System.out.println("|                                             |");
        System.out.println("|        OPENMP AUTOMATIC PARALLELIZER        |");
        System.out.println("|                                             |");
        System.out.println("|               By Nam Q. Lam                 |");
        System.out.println("|                                             |");
        System.out.println("|  Master Project @ San Jose State University  |");
        System.out.println("|                                             |");
        System.out.println("-----------------------------------------------");
        System.out.println();
        if ( op == 0 ) {
            System.out.println("============  Train the Classifier  ============");
        }
```

```
        else {
            System.out.println("============  Parallelize  Program  ============");
        }
    }

    /**
     * Extracts C #define, interested only in the form: #define <IDENTIFIER> <INTEGER>
     * @param inputFileName input C source file
     * @return a HashMap containing <name, value> pairs of extracted defs
     */
    private HashMap<String, Integer> extractPoundDefs(String inputFileName) {
        HashMap<String, Integer> poundDefs = new HashMap<String, Integer>();
        SourceFile src = new SourceFile(inputFileName);
        ArrayList<String> lines = src.getLines();
        for ( int i = 0; i < lines.size(); i++ ) {
            String line = lines.get(i);
            if ( line.indexOf("#define") >= 0 ) {
                String[] tokens = line.split("\\s+");
                if ( tokens.length == 3 ) {
                    if ( isCIdentifier(tokens[1]) && isInteger(tokens[2]) ) {
                        poundDefs.put(tokens[1], Integer.parseInt(tokens[2]));
                    }
                }
            }
        }
        return poundDefs;
    }

    /**
     * Validates a C language identifier.
     * @param str identifier string to be validated
     * @return true if str is a valid identifier, false otherwise
     */
    private boolean isCIdentifier(String str) {
        String identifierRegex = "^(_|[A-Za-z])(_|[A-Za-z]|[0-9])*$";
        Pattern pattern = Pattern.compile(identifierRegex);
        Matcher matcher = pattern.matcher(str);
        return matcher.find();
    }

    /**
     * Validates an integer represented by a string.
     * @param str integer string to be validated
     * @return true if str is representing an integer, false otherwise
     */
    private boolean isInteger(String str) {
        try { Integer.parseInt(str); }
        catch ( NumberFormatException e ) {    return false; }
        return true;
    }

}
```

```
/*
 * SimpleFileFilter.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql;
```

```
import java.io.File;
import javax.swing.filechooser.FileFilter;


/**
 * A simple file extension filter.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class SimpleFileFilter extends FileFilter  {

    /** Description for the extensions. */
    private String description;

    /** Interested extensions. */
    private String[] extensions;

    /**
     * Constructor.
     * @param desc description for the extension
     * @param ext a single interested extension
     */
    public SimpleFileFilter(String desc, String ext) {
        this(desc, new String[] { ext });
    }

    /**
     * Constructor.
     * @param desc description for the extensions
     * @param exts list of interested extensions
     */
    public SimpleFileFilter(String desc, String[] exts) {
        description = desc;
        extensions = new String[exts.length];
        for ( int i = 0; i < exts.length; i++ ) {
            extensions[i] = exts[i].toLowerCase();
        }
    }

    /* (non-Javadoc)
     * @see javax.swing.filechooser.FileFilter#accept(java.io.File)
     */
    public boolean accept(File f) {

        // Allow all directories/folders
        if ( f.isDirectory() ) {
            return true;
        }

        // Accept only those interested extensions
        String name = f.getName().toLowerCase();
        for ( int i = 0; i < extensions.length; i++ ) {
            if ( name.endsWith(extensions[i]) ) {
                return true;
            }
        }

        // The rest are ignored
        return false;
    }

    /* (non-Javadoc)
     * @see javax.swing.filechooser.FileFilter#getDescription()
     */
    public String getDescription() {
        return description;
    }

}
```

```
/*
 * CParserVisitorAdapter.java
 *
 * Created on March 16, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.backend;


import nql.cd.frontend.ASTabstractDeclarator;
import nql.cd.frontend.ASTadditiveExpression;
import nql.cd.frontend.ASTassignmentExpression;
import nql.cd.frontend.ASTbitwiseANDExpression;
import nql.cd.frontend.ASTbitwiseORExpression;
import nql.cd.frontend.ASTbitwiseXORExpression;
import nql.cd.frontend.ASTcastExpression;
import nql.cd.frontend.ASTcharacter;
import nql.cd.frontend.ASTcompoundStatement;
import nql.cd.frontend.ASTconditionalExpression;
import nql.cd.frontend.ASTdeclaration;
import nql.cd.frontend.ASTdeclarationList;
import nql.cd.frontend.ASTdeclarationSpecifiers;
import nql.cd.frontend.ASTdeclarator;
import nql.cd.frontend.ASTdirectAbstractDeclarator;
import nql.cd.frontend.ASTdirectDeclarator;
import nql.cd.frontend.ASTdoWhileStatement;
import nql.cd.frontend.ASTenumSpecifier;
import nql.cd.frontend.ASTenumerator;
import nql.cd.frontend.ASTenumeratorList;
import nql.cd.frontend.ASTequalityExpression;
import nql.cd.frontend.ASTexpression;
import nql.cd.frontend.ASTexpressionStatement;
import nql.cd.frontend.ASTexternalDeclaration;
import nql.cd.frontend.ASTforStatement;
import nql.cd.frontend.ASTfunctionDefinition;
import nql.cd.frontend.ASTfunctionDefinitionLookahead;
import nql.cd.frontend.ASTidentifier;
import nql.cd.frontend.ASTifStatement;
import nql.cd.frontend.ASTinitDeclarator;
import nql.cd.frontend.ASTinitDeclaratorList;
import nql.cd.frontend.ASTinitializer;
import nql.cd.frontend.ASTinitializerList;
import nql.cd.frontend.ASTinteger;
import nql.cd.frontend.ASTjumpStatement;
import nql.cd.frontend.ASTlabeledStatement;
import nql.cd.frontend.ASTlogicalANDExpression;
import nql.cd.frontend.ASTlogicalORExpression;
import nql.cd.frontend.ASTmultiplicativeExpression;
import nql.cd.frontend.ASTparameterDeclaration;
import nql.cd.frontend.ASTparameterList;
import nql.cd.frontend.ASTparameterTypeList;
import nql.cd.frontend.ASTparse;
import nql.cd.frontend.ASTpointer;
import nql.cd.frontend.ASTpostfixExpression;
import nql.cd.frontend.ASTreal;
import nql.cd.frontend.ASTrelationalExpression;
import nql.cd.frontend.ASTshiftExpression;
import nql.cd.frontend.ASTstorageClassSpecifier;
import nql.cd.frontend.ASTstring;
import nql.cd.frontend.ASTstructDeclaration;
import nql.cd.frontend.ASTstructDeclarationList;
import nql.cd.frontend.ASTstructDeclarator;
```

```
import nql.cd.frontend.ASTstructDeclaratorList;
import nql.cd.frontend.ASTstructOrUnion;
import nql.cd.frontend.ASTstructOrUnionSpecifier;
import nql.cd.frontend.ASTswitchStatement;
import nql.cd.frontend.ASTtypeName;
import nql.cd.frontend.ASTtypeQualifier;
import nql.cd.frontend.ASTtypeSpecifier;
import nql.cd.frontend.ASTtypedefName;
import nql.cd.frontend.ASTunaryExpression;
import nql.cd.frontend.ASTwhileStatement;
import nql.cd.frontend.CParserVisitor;
import nql.cd.frontend.SimpleNode;


/**
 * Adapter to the main parser visitor to provide
 * ease of implementations of task-specific visitors.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class CParserVisitorAdapter implements CParserVisitor {

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.SimpleNode, java.lang.Object)
     */
    public Object visit(SimpleNode node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTparse, java.lang.Object)
     */
    public Object visit(ASTparse node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTexternalDeclaration,
java.lang.Object)
     */
    public Object visit(ASTexternalDeclaration node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTfunctionDefinition,
java.lang.Object)
     */
    public Object visit(ASTfunctionDefinition node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTfunctionDefinitionLookahead,
java.lang.Object)
     */
    public Object visit(ASTfunctionDefinitionLookahead node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTdeclarationList,
java.lang.Object)
     */
    public Object visit(ASTdeclarationList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
```

```
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTdeclaration,
java.lang.Object)
     */
    public Object visit(ASTdeclaration node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTdeclarationSpecifiers,
java.lang.Object)
     */
    public Object visit(ASTdeclarationSpecifiers node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTinitDeclaratorList,
java.lang.Object)
     */
    public Object visit(ASTinitDeclaratorList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTinitDeclarator,
java.lang.Object)
     */
    public Object visit(ASTinitDeclarator node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTinitializer,
java.lang.Object)
     */
    public Object visit(ASTinitializer node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTinitializerList,
java.lang.Object)
     */
    public Object visit(ASTinitializerList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTtypeName, java.lang.Object)
     */
    public Object visit(ASTtypeName node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstorageClassSpecifier,
java.lang.Object)
     */
    public Object visit(ASTstorageClassSpecifier node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTtypeSpecifier,
java.lang.Object)
     */
    public Object visit(ASTtypeSpecifier node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
```

```
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTtypedefName,
java.lang.Object)
     */
    public Object visit(ASTtypedefName node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTtypeQualifier,
java.lang.Object)
     */
    public Object visit(ASTtypeQualifier node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstructOrUnionSpecifier,
java.lang.Object)
     */
    public Object visit(ASTstructOrUnionSpecifier node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstructOrUnion,
java.lang.Object)
     */
    public Object visit(ASTstructOrUnion node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstructDeclarationList,
java.lang.Object)
     */
    public Object visit(ASTstructDeclarationList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstructDeclaration,
java.lang.Object)
     */
    public Object visit(ASTstructDeclaration node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstructDeclaratorList,
java.lang.Object)
     */
    public Object visit(ASTstructDeclaratorList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstructDeclarator,
java.lang.Object)
     */
    public Object visit(ASTstructDeclarator node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTenumSpecifier,
java.lang.Object)
     */
    public Object visit(ASTenumSpecifier node, Object data) {
        return node.childrenAccept(this, data);
    }
```

```
    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTenumeratorList,
java.lang.Object)
     */
    public Object visit(ASTenumeratorList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTenumerator, java.lang.Object)
     */
    public Object visit(ASTenumerator node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTdeclarator, java.lang.Object)
     */
    public Object visit(ASTdeclarator node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTpointer, java.lang.Object)
     */
    public Object visit(ASTpointer node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTdirectDeclarator,
java.lang.Object)
     */
    public Object visit(ASTdirectDeclarator node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTparameterTypeList,
java.lang.Object)
     */
    public Object visit(ASTparameterTypeList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTparameterList,
java.lang.Object)
     */
    public Object visit(ASTparameterList node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTparameterDeclaration,
java.lang.Object)
     */
    public Object visit(ASTparameterDeclaration node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTabstractDeclarator,
java.lang.Object)
     */
    public Object visit(ASTabstractDeclarator node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
```

```
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTdirectAbstractDeclarator,
java.lang.Object)
     */
    public Object visit(ASTdirectAbstractDeclarator node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTcompoundStatement,
java.lang.Object)
     */
    public Object visit(ASTcompoundStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTlabeledStatement,
java.lang.Object)
     */
    public Object visit(ASTlabeledStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTjumpStatement,
java.lang.Object)
     */
    public Object visit(ASTjumpStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTexpressionStatement,
java.lang.Object)
     */
    public Object visit(ASTexpressionStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTifStatement,
java.lang.Object)
     */
    public Object visit(ASTifStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTswitchStatement,
java.lang.Object)
     */
    public Object visit(ASTswitchStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTforStatement,
java.lang.Object)
     */
    public Object visit(ASTforStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTwhileStatement,
java.lang.Object)
     */
    public Object visit(ASTwhileStatement node, Object data) {
        return node.childrenAccept(this, data);
    }
```

```
    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTdoWhileStatement,
java.lang.Object)
     */
    public Object visit(ASTdoWhileStatement node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTexpression, java.lang.Object)
     */
    public Object visit(ASTexpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTassignmentExpression,
java.lang.Object)
     */
    public Object visit(ASTassignmentExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTconditionalExpression,
java.lang.Object)
     */
    public Object visit(ASTconditionalExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTlogicalORExpression,
java.lang.Object)
     */
    public Object visit(ASTlogicalORExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTlogicalANDExpression,
java.lang.Object)
     */
    public Object visit(ASTlogicalANDExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTbitwiseORExpression,
java.lang.Object)
     */
    public Object visit(ASTbitwiseORExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTbitwiseXORExpression,
java.lang.Object)
     */
    public Object visit(ASTbitwiseXORExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTbitwiseANDExpression,
java.lang.Object)
     */
    public Object visit(ASTbitwiseANDExpression node, Object data) {
        return node.childrenAccept(this, data);
    }
```

```
    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTequalityExpression,
java.lang.Object)
     */
    public Object visit(ASTequalityExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTrelationalExpression,
java.lang.Object)
     */
    public Object visit(ASTrelationalExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTshiftExpression,
java.lang.Object)
     */
    public Object visit(ASTshiftExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTadditiveExpression,
java.lang.Object)
     */
    public Object visit(ASTadditiveExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTmultiplicativeExpression,
java.lang.Object)
     */
    public Object visit(ASTmultiplicativeExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTcastExpression,
java.lang.Object)
     */
    public Object visit(ASTcastExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTunaryExpression,
java.lang.Object)
     */
    public Object visit(ASTunaryExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTpostfixExpression,
java.lang.Object)
     */
    public Object visit(ASTpostfixExpression node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTidentifier, java.lang.Object)
     */
    public Object visit(ASTidentifier node, Object data) {
        return node.childrenAccept(this, data);
    }
```

```
    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTinteger, java.lang.Object)
     */
    public Object visit(ASTinteger node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTreal, java.lang.Object)
     */
    public Object visit(ASTreal node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTstring, java.lang.Object)
     */
    public Object visit(ASTstring node, Object data) {
        return node.childrenAccept(this, data);
    }

    /* (non-Javadoc)
     * @see nql.parsers.CParserVisitor#visit(nql.parsers.ASTcharacter, java.lang.Object)
     */
    public Object visit(ASTcharacter node, Object data) {
        return node.childrenAccept(this, data);
    }
}
```

```
/*
 * ForLoopExtractorVisitor.java
 *
 * Created on March 16, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.backend;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import nql.cd.frontend.ASTadditiveExpression;
import nql.cd.frontend.ASTassignmentExpression;
import nql.cd.frontend.ASTbitwiseANDExpression;
import nql.cd.frontend.ASTbitwiseORExpression;
import nql.cd.frontend.ASTbitwiseXORExpression;
import nql.cd.frontend.ASTcastExpression;
import nql.cd.frontend.ASTconditionalExpression;
import nql.cd.frontend.ASTdirectDeclarator;
import nql.cd.frontend.ASTequalityExpression;
import nql.cd.frontend.ASTexpression;
import nql.cd.frontend.ASTforStatement;
import nql.cd.frontend.ASTfunctionDefinition;
import nql.cd.frontend.ASTidentifier;
import nql.cd.frontend.ASTinitDeclarator;
import nql.cd.frontend.ASTinteger;
import nql.cd.frontend.ASTlogicalANDExpression;
import nql.cd.frontend.ASTlogicalORExpression;
import nql.cd.frontend.ASTmultiplicativeExpression;
```

```
import nql.cd.frontend.ASTpostfixExpression;
import nql.cd.frontend.ASTrelationalExpression;
import nql.cd.frontend.ASTshiftExpression;
import nql.cd.frontend.ASTunaryExpression;
import nql.cd.frontend.CParserTreeConstants;
import nql.cd.frontend.SimpleNode;
import nql.cd.intermediate.ForLoopNode;
import nql.cd.intermediate.ForLoopTree;
import nql.cd.intermediate.NodeKey;
import nql.cd.intermediate.NodeType;
import nql.cd.intermediate.SymbolTableEntry;
import nql.cd.intermediate.SymbolTableFactory;
import nql.cd.intermediate.SymbolTableKey;
import nql.cd.intermediate.SymbolTableStack;
import nql.cd.intermediate.SymbolTableType;
import nql.cd.intermediate.Variable;
import nql.cd.intermediate.VariableType;
import nql.exceptions.AutoParallelizerRestrictionException;


/**
 * Visitor to extract information of all FOR loop nodes in the AST.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class ForLoopExtractorVisitor extends CParserVisitorAdapter {

    /** Nested level of a current FOR loop node. */
    private int currentNestedLevel;

    /** Nesting level of a current FOR loop node. */
    private int currentNestingLevel;

    /** Is the traversal currently inside a variable declaration? */
    private boolean insideVariableDeclaration;

    /** Is the traversal currently inside a function declaration/definition? */
    private boolean insideFunctionDecOrDef;

    /** Is the traversal currently inside a function definition? */
    private boolean insideFunctionDefinition;

    /** Is the traversal currently inside a top-level FOR loop node? */
    private boolean insideTopLevelForLoopNode;

    /**
     * Is the traversal currently inside an assignment expression
     * that contributes to OpenMP deduction?
     */
    private boolean insideAssignmentExpressionAndReduction;

    /**
     * Is the traversal currently inside a postfix expression
     * that contributes to OpenMP deduction?
     */
    private boolean insidePostfixExpressionAndReduction;

    /** Is the traversal currently inside a post increment/decrement expression? */
    private boolean insidePostIncrementDecrementExpression;

    /** Is the traversal currently inside a pre increment/decrement expression? */
    private boolean insidePreIncrementDecrementExpression;

    /** Is the traversal currently inside a FOR loop control parameters? */
    private boolean expInsideForLoopControlParameters;

    /** Bi-operator of an assignment expression inside a FOR loop. */
    private String assignmentExpressionOp;

    /** Operator of a postfix expression inside a FOR loop. */
```

```
    private String postfixExpressionOp;

    /**
     * List of array-typed postfixExpression nodes that are
     * on the left of an assignment expression inside a FOR loop.
     */
    private ArrayList<SimpleNode> leftArrayPostfixExpressionList;

    /**
     * List of array-typed postfixExpression nodes that are
     * on the right of an assignment expression inside a FOR loop.
     */
    private ArrayList<SimpleNode> rightArrayPostfixExpressionList;

    /** Tree of FOR loops (a nested loop). */
    private ForLoopTree forLoopTree;

    /** List of extracted ForLoopTree(s). */
    private ArrayList<ForLoopTree> forLoopTreeList;

    /** Symbol table stack to store identifier information. */
    private SymbolTableStack symbolTableStack;

    /**
     * Constructor.
     */
    public ForLoopExtractorVisitor(HashMap<String, Integer> poundDefs) {
        currentNestedLevel = -1;
        currentNestingLevel = -1;
        insideVariableDeclaration = false;
        insideFunctionDecOrDef = false;
        insideFunctionDefinition = false;
        insideTopLevelForLoopNode = false;
        insideAssignmentExpressionAndReduction = false;
        assignmentExpressionOp = null;
        insidePostfixExpressionAndReduction = false;
        postfixExpressionOp = null;
        expInsideForLoopControlParameters = false;
        leftArrayPostfixExpressionList = new ArrayList<SimpleNode>();
        rightArrayPostfixExpressionList = new ArrayList<SimpleNode>();
        forLoopTree = new ForLoopTree();
        forLoopTreeList = new ArrayList<ForLoopTree>();
        symbolTableStack = SymbolTableFactory.createSymbolTableStack();

        // Process #define definitions
        if ( !poundDefs.isEmpty() ) {
            Set<String> keySet = poundDefs.keySet();
            Iterator<String> it = keySet.iterator();
            while ( it.hasNext() ) {
                String name = it.next();
                Integer value = (Integer)poundDefs.get(name);
                SymbolTableEntry definedEntry =    symbolTableStack.enterGlobal(name);
                definedEntry.setAttribute(SymbolTableKey.TYPE, SymbolTableType.DEFINE);
                definedEntry.setAttribute(SymbolTableKey.VALUE, value);
            }
        }

        // Populate pre-defined library function names
        symbolTableStack.enterGlobal("scanf").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
        symbolTableStack.enterGlobal("printf").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
        symbolTableStack.enterGlobal("sin").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
        symbolTableStack.enterGlobal("cos").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
        symbolTableStack.enterGlobal("tan").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
        symbolTableStack.enterGlobal("sqrt").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
```

```
        symbolTableStack.enterGlobal("pow").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
        symbolTableStack.enterGlobal("log").setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
    }

    /**
     * Retrieves the extracted FOR loops as a list of ForLoopTree(s).
     * @return list of extracted ForLoopTree(s)
     */
    public ArrayList<ForLoopTree> getForLoopTreeList() {
        return forLoopTreeList;
    }

    /* (non-Javadoc)
     * @see
nql.cd.intermediate.CParserVisitorAdapter#visit(nql.cd.frontend.ASTfunctionDefinition,
java.lang.Object)
     */
    public Object visit(ASTfunctionDefinition node, Object data) {

        // Enter a function scope
        symbolTableStack.push();

        // Set the flag
        insideFunctionDefinition = true;

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        // Clear the flag
        insideFunctionDefinition = false;

        // Leave the function scope
        symbolTableStack.pop();

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTinitDeclarator, java.lang.Object)
     */
    public Object visit(ASTinitDeclarator node, Object data) {

        // Retrieve variable and its initial value
        String name = null;
        Integer value = null;
        SymbolTableEntry varEntry = null;
        if ( node.jjtGetNumChildren() > 1 ) {
            // Interested in initialization declaration of the form <identifier> =
<integer>,
            // which will be used to compute number of iterations of a FOR loop
            SimpleNode identifierNode =
(SimpleNode)node.jjtGetChild(0).jjtGetChild(0).jjtGetChild(0);
            SimpleNode integerNode = (SimpleNode)node.jjtGetChild(1).jjtGetChild(0);
            if ( identifierNode instanceof ASTidentifier && integerNode instanceof
ASTinteger ) {
                name = (String)identifierNode.getAttribute(NodeKey.NAME);
                value = new Integer((String)integerNode.getAttribute(NodeKey.VALUE));
            }
        }

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        // Set init value for the variable
        if ( name != null && value != null ) {
```

```
            if ( (varEntry = symbolTableStack.lookupLocal(name)) != null ) {
                varEntry.setAttribute(SymbolTableKey.VALUE, value);
            }
        }

        return obj;
    }

    /* (non-Javadoc)
     * @see
nql.cd.intermediate.CParserVisitorAdapter#visit(nql.cd.frontend.ASTdirectDeclarator,
java.lang.Object)
     */
    public Object visit(ASTdirectDeclarator node, Object data) {

        // Set the flags
        boolean isFunctionDecOrDef =
((Boolean)node.getAttribute(NodeKey.FUNCTION_DEC_DEF)).booleanValue();
        if ( isFunctionDecOrDef ) {
            insideFunctionDecOrDef = true;
        }
        else {
            insideVariableDeclaration = true;
        }

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        // Clear the flags
        insideFunctionDecOrDef = false;
        insideVariableDeclaration = false;

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTforStatement, java.lang.Object)
     */
    public Object visit(ASTforStatement node, Object data) {

        // Enter a for statement scope
        symbolTableStack.push();

        // Set the flag
        // We're [still] inside a top-level FOR loop now
        insideTopLevelForLoopNode = true;

        // The ForLoopNode for the parent
        ForLoopNode parentForLoopNode = null;

        // Retrieve the parent FOR loop
        SimpleNode pNode = getParentForLoop(node);

        // Update number of parents children, except the root
        int children = 0;
        if ( pNode != null && pNode instanceof ASTforStatement ) {
            children = ((Integer)pNode.getAttribute(NodeKey.NUM_CHILDREN)).intValue();
            pNode.setAttribute(NodeKey.NUM_CHILDREN, new Integer(children + 1));
        }

        // Parents children index for a new ForLoopNode child
        int nextParentsChildrenIndex = children;

        // If this FOR loop has a sibling
        if ( currentNestingLevel != currentNestedLevel ) {

            // Conditionally save currentNestingLevels to the parent
            // and retrieve the ForLoopNode
            int pLevel = 0;
```

```
            assert(pNode != null);
            if ( pNode instanceof ASTforStatement ) {

                pLevel =
((Integer)pNode.getAttribute(NodeKey.CURRENT_NESTING_LEVELS)).intValue();
                if ( pLevel < currentNestingLevel ) {
                    pNode.setAttribute(NodeKey.CURRENT_NESTING_LEVELS, new
Integer(currentNestingLevel));
                }
                parentForLoopNode =
(ForLoopNode)pNode.getAttribute(NodeKey.FOR_LOOP_NODE);
            }

            // Continue with new levels
            currentNestedLevel++;
            currentNestingLevel = currentNestedLevel;
        }
        // If not...
        else {
            currentNestingLevel++;
            currentNestedLevel++;
        }

        // Current attribute of this FOR loop, which might be
        // updated down the path by the its children
        node.setAttribute(NodeKey.CURRENT_NESTING_LEVELS, new
Integer(currentNestingLevel));
        node.setAttribute(NodeKey.NUM_CHILDREN, new Integer(0));

        // Retrieve OpenMP pragma specified for this FOR loop, if any
        String ompPragma = (String)node.getAttribute(NodeKey.SPECIAL_TOKEN);
        if ( ompPragma != null ) {
            if ( (ompPragma.length() > 7) && (ompPragma.substring(0,
7).equals("#pragma")) ) {

                // Remove end-line terminator of the ompPragma
                ompPragma = ompPragma.replaceAll("\\n|\\r", "");
            }
            else {
                ompPragma = null;
            }
        }

        // Retrieve source-code positions of this FOR loop
        int beginLine = ((Integer)node.getAttribute(NodeKey.BEGIN_LINE)).intValue();
        int beginColumn = ((Integer)node.getAttribute(NodeKey.BEGIN_COLUMN)).intValue();

        //
        // Navigate to the relational/equality expression of this FOR loop,
        // and extract the integer node on the right of the expression, where
        // expCount = 0, 1, or 2 due to the three C source code cases below.
        //
        // for (                                          ; <relational/equality
expression>; <...>)
        // for (                <assignment expression>; <relational/equality
expression>; <...>)
        // for (<type specifier> <assignment expression>; <relational/equality
expression>; <...>)
        //
        SimpleNode numberIterationsNode = null;
        for ( int expCount = 0; expCount < 3; expCount++ ) {

            // Skip the type specifier, if any, in the first loop control expression
            if ( ((SimpleNode)node.jjtGetChild(expCount)).jjtGetNumChildren() > 0 ) {

                numberIterationsNode =
(SimpleNode)node.jjtGetChild(expCount).jjtGetChild(0);
                if ( (numberIterationsNode instanceof ASTrelationalExpression) ||
                     (numberIterationsNode instanceof ASTequalityExpression) ) {
                       numberIterationsNode =
(SimpleNode)numberIterationsNode.jjtGetChild(1);
```

```
                    break;
                }
            }
        }

        // Retrieve number of loop iterations via the numberIterationsNode extracted
above;
        // if it is an identifier, retrieve its name
        int numberOfIterations = 0;
        String numberIterationsVariableName = null;
        if ( numberIterationsNode instanceof ASTinteger ) {
            numberOfIterations =
Integer.parseInt((String)numberIterationsNode.getAttribute(NodeKey.VALUE));
        }
        else if ( numberIterationsNode instanceof ASTidentifier ) {
            numberIterationsVariableName =
(String)numberIterationsNode.getAttribute(NodeKey.NAME);
            SymbolTableEntry varEntry =
symbolTableStack.lookup(numberIterationsVariableName);
            if ( varEntry != null ) {
                Integer intValue = (Integer)varEntry.getAttribute(SymbolTableKey.VALUE);
                if ( intValue != null ) {
                    numberOfIterations = intValue.intValue();
                }
            }
        }
        else {
            // Display restriction violation information
            System.out.println("\n*** A restriction violation has occurred. Cannot parse
number of iterations.\n");
        }

        // Extract the name of this loop's iteration variable
        String iterationVariableName = null;
        SimpleNode anExpNode = (SimpleNode)node.jjtGetChild(0);
        if ( anExpNode.jjtGetNumChildren() == 0 ) {
            // _Child(0) is a type specifier node, skip it, get _Child(1) instead
            anExpNode = (SimpleNode)node.jjtGetChild(1);
        }
        if ( anExpNode != null ) {
            SimpleNode anAssignExpNode = (SimpleNode)anExpNode.jjtGetChild(0);
            if ( anAssignExpNode instanceof ASTassignmentExpression ) {
                SimpleNode iterationVariableNode =
(SimpleNode)anAssignExpNode.jjtGetChild(0);
                if ( iterationVariableNode instanceof ASTidentifier ) {
                    iterationVariableName =
(String)iterationVariableNode.getAttribute(NodeKey.NAME);
                }
            }
        }

        // Create node for this FOR loop
        ForLoopNode forLoopNode = new ForLoopNode();
        forLoopNode.setBeginLine(beginLine);
        forLoopNode.setBeginColumn(beginColumn);
        forLoopNode.setNestedLevel(currentNestedLevel);
        forLoopNode.setNumberOfIterations(numberOfIterations);
        forLoopNode.setSetOfVariables(new HashSet<Variable>());
        forLoopNode.setOpenMPPragma(ompPragma);
        forLoopNode.setNumberIterationsVariableName(numberIterationsVariableName);
        forLoopNode.setIterationVariableName(iterationVariableName);

        // Store the node as an attribute
        node.setAttribute(NodeKey.FOR_LOOP_NODE, forLoopNode);

        // Add the node to the tree
        if ( forLoopTree.isEmpty() ) {
            forLoopTree = new ForLoopTree(forLoopNode);
        }
        else if ( nextParentsChildrenIndex == 0 ) {
            forLoopTree.insert(forLoopNode);
```

```
        }
        else if ( parentForLoopNode != null ) {
            forLoopTree.insert(forLoopNode, parentForLoopNode, nextParentsChildrenIndex);
        }

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        // Determine whether this FOR loop is parallelizable or not
        // based upon loop-carried dependence analysis
        if ( hasLoopCarriedDependence(leftArrayPostfixExpressionList,
rightArrayPostfixExpressionList) ) {
            forLoopNode.setLoopCarriedDependence(true);
        }

        // Determine and set nesting levels of this FOR loop;
        // pick the longest one...
        int nestingLevels = currentNestingLevel - currentNestedLevel;
        int aLevel =
((Integer)node.getAttribute(NodeKey.CURRENT_NESTING_LEVELS)).intValue();
        if ( aLevel > currentNestingLevel ) {
            nestingLevels = aLevel - currentNestedLevel;
        }
        forLoopNode.setNestingLevels(nestingLevels);

        // Determine and set number of children of this FOR loop
        int newNumChildren =
((Integer)node.getAttribute(NodeKey.NUM_CHILDREN)).intValue();
        forLoopNode.setNumberOfChildren(newNumChildren);

        // Done with attributes of this FOR loop,
        // reset data for a next FOR loop
        leftArrayPostfixExpressionList.clear();
        rightArrayPostfixExpressionList.clear();
        currentNestedLevel--;

        // Has completed a nested FOR loop (reached back to the first-level FOR loop),
        // add it to the two-dimensional list, then start a new empty list for another
nested FOR loop.
        if ( currentNestedLevel == -1 ) {
            currentNestingLevel = -1;
            forLoopTreeList.add(forLoopTree);
            forLoopTree = new ForLoopTree();

            // Clear the flag
            insideTopLevelForLoopNode = false;
        }

        // Leave the for statement scope
        symbolTableStack.pop();

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTexpression, java.lang.Object)
     */
    public Object visit(ASTexpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Where is this expression called from?
        NodeType type = (NodeType)node.getAttribute(NodeKey.CALLED_FROM);

        // Set the flag
```

```
        // If it is called from FOR statement control parameters,
        // set the flag so that other nodes down the tree can act accordingly
        if ( type == NodeType.FOR_STMT_NODE ) {
            expInsideForLoopControlParameters = true;
        }

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        // Clear the flag
        // Existing this expression node
        expInsideForLoopControlParameters = false;

        return obj;
    }

    public Object visit(ASTassignmentExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Skip over assignment expressions that belong
        // inside a for loop control parameters
        if ( !expInsideForLoopControlParameters ) {

            // Left and right subtrees of this node
            SimpleNode nodeLeft = (SimpleNode)node.jjtGetChild(0);
            SimpleNode nodeRight = (SimpleNode)node.jjtGetChild(1);

            // Extract array-typed postfixExpression nodes, if any,
            // inside this assignment expression
            leftArrayPostfixExpressionList.addAll(extractArrayPostfixNodes(nodeLeft));
            rightArrayPostfixExpressionList.addAll(extractArrayPostfixNodes(nodeRight));

            // Set the flag
            // Does this node contribute to a reduction?
            boolean reduction =
((Boolean)node.getAttribute(NodeKey.REDUCTION)).booleanValue();
            String op = (String)node.getAttribute(NodeKey.REDUCTION_OP);
            if ( reduction ) {
                // Interested only in this form of reduction:
                // <identifier> [+*-/&^|]= <expression>
                assert(op != null);
                insideAssignmentExpressionAndReduction = true;
                assignmentExpressionOp = op;
            }
            else {
                // Interested only in this form of reduction:
                // <identifier> = <identifier> [+-*/] <expression>
                if ( (nodeLeft instanceof ASTidentifier) &&
                     ((nodeRight instanceof ASTadditiveExpression) ||
                      (nodeRight instanceof ASTmultiplicativeExpression)) ) {
                    op = (String)nodeRight.getAttribute(NodeKey.OPERATOR);
                    SimpleNode node2 = (SimpleNode)nodeRight.jjtGetChild(0);
                    if ( (node2 instanceof ASTidentifier) && (!op.equals("%")) ) {
                        String leftName = (String)nodeLeft.getAttribute(NodeKey.NAME);
                        String rightName = (String)node2.getAttribute(NodeKey.NAME);
                        if ( leftName.equals(rightName) ) {
                            insideAssignmentExpressionAndReduction = true;
                            assignmentExpressionOp = op;
                        }
                    }
                }
            }
        }
```

```
        // ----------------------------------
        // Traversing down the tree...
        // ----------------------------------
        Object obj = super.visit(node, data);

        // Clear the flag
        // Existing this assignment expression node
        insideAssignmentExpressionAndReduction = false;
        assignmentExpressionOp = null;

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTconditionalExpression,
java.lang.Object)
     */
    public Object visit(ASTconditionalExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ----------------------------------
        // Traversing down the tree...
        // ----------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTlogicalORExpression,
java.lang.Object)
     */
    public Object visit(ASTlogicalORExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ----------------------------------
        // Traversing down the tree...
        // ----------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTlogicalANDExpression,
java.lang.Object)
     */
    public Object visit(ASTlogicalANDExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
```

```
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ----------------------------------
        // Traversing down the tree...
        // ----------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTbitwiseORExpression,
java.lang.Object)
     */
    public Object visit(ASTbitwiseORExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ----------------------------------
        // Traversing down the tree...
        // ----------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTbitwiseXORExpression,
java.lang.Object)
     */
    public Object visit(ASTbitwiseXORExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ----------------------------------
        // Traversing down the tree...
        // ----------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTbitwiseANDExpression,
java.lang.Object)
     */
```

```
    public Object visit(ASTbitwiseANDExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTequalityExpression,
java.lang.Object)
     */
    public Object visit(ASTequalityExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTrelationalExpression,
java.lang.Object)
     */
    public Object visit(ASTrelationalExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }
```

```
    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTshiftExpression, java.lang.Object)
     */
    public Object visit(ASTshiftExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTadditiveExpression,
java.lang.Object)
     */
    public Object visit(ASTadditiveExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTmultiplicativeExpression,
java.lang.Object)
     */
    public Object visit(ASTmultiplicativeExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);
```

```
        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTcastExpression, java.lang.Object)
     */
    public Object visit(ASTcastExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTunaryExpression, java.lang.Object)
     */
    public Object visit(ASTunaryExpression node, Object data) {

        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // State of the identifier inside this unary expression
        boolean isCurrentIdentifierBeingModified =
((Boolean)node.getAttribute(NodeKey.MODIFYING_IDENTIFIER)).booleanValue();

        // Set the flag
        // If the identifier is being modified or updated,
        // set the flag so that other nodes down the tree can act accordingly
        if ( isCurrentIdentifierBeingModified ) {
            insidePreIncrementDecrementExpression = true;
        }

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        // Clear the flag
        // Existing this unary expression node
        insidePreIncrementDecrementExpression = false;

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTpostfixExpression,
java.lang.Object)
     */
    public Object visit(ASTpostfixExpression node, Object data) {
```

```
        // Process this node only if it is currently inside a FOR loop node;
        // otherwise, simply continue with the traversal.
        if ( !insideTopLevelForLoopNode ) {
            return super.visit(node, data);
        }

        // Each time this node is encountered, increment operations count
        // for its current enclosing FOR loop; operations count in a
        // FOR loop can be used to compute its complexity, etc.
        incrementOperationsCount(node);

        // State of the identifier inside this postfix expression
        boolean isCurrentIdentifierBeingModified =
((Boolean)node.getAttribute(NodeKey.MODIFYING_IDENTIFIER)).booleanValue();

        // Set the flags
        // If the identifier is being modified or updated,
        // set the flag so that other nodes down the tree can act accordingly
        if ( isCurrentIdentifierBeingModified ) {
            insidePostIncrementDecrementExpression = true;

            // Does this node contribute to a reduction?
            if ( !expInsideForLoopControlParameters ) {
                boolean reduction =
((Boolean)node.getAttribute(NodeKey.REDUCTION)).booleanValue();
                String op = (String)node.getAttribute(NodeKey.REDUCTION_OP);
                if ( reduction ) {
                    assert(op != null);
                    insidePostfixExpressionAndReduction = true;
                    postfixExpressionOp = op;
                }
            }
        }

        // Is this node an array postfix? If so, retrieve the array name.
        String arrayName = null;
        boolean isArrayPostfixed =
((Boolean)node.getAttribute(NodeKey.ARRAY_POSTFIXED)).booleanValue();
        if ( isArrayPostfixed ) {
            SimpleNode aNode = (SimpleNode)node.jjtGetChild(0);
            if ( aNode instanceof ASTidentifier ) {
                arrayName = (String)aNode.getAttribute(NodeKey.NAME);
            }
        }

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        // Clear the flags
        // Existing this postfix expression
        insidePostIncrementDecrementExpression = false;
        insidePostfixExpressionAndReduction = false;
        postfixExpressionOp = null;

        // Update the identifier state if this node is an array postfix, i.e.,
        // although the identifier (array name) happens to be on the left side
        // of an assignment, but array name itself is not updated or modified,
        // only its elements, if any.
        if ( isArrayPostfixed && (arrayName != null) ) {

            Variable oldVariable = new Variable(arrayName, false);
            Variable newVariable = new Variable(arrayName, true);
            newVariable.setType(VariableType.ARRAY);

            SimpleNode pNode = getParentForLoop(node);
            assert(pNode != null);
            if ( pNode instanceof ASTforStatement ) {
```

```
                ForLoopNode forLoopNode =
(ForLoopNode)pNode.getAttribute(NodeKey.FOR_LOOP_NODE);
                HashSet<Variable> setOfVariables = forLoopNode.getSetOfVariables();
                setOfVariables.remove(oldVariable);
                setOfVariables.add(newVariable);
                forLoopNode.setSetOfVariables(setOfVariables);
            }
        }

        return obj;
    }

    /* (non-Javadoc)
     * @see CParserVisitorAdapter#visit(nql.parsers.ASTidentifier, java.lang.Object)
     */
    public Object visit(ASTidentifier node, Object data) {

        // Retrieve name of the identifier, which can be a variable or a function name
        String identifierName = (String)node.getAttribute(NodeKey.NAME);

        // If the identifier is a function, known from function declaration/definition,
        // add it to the global symbol table (bottom of the stack); otherwise, it is
        // a variable and is added to a local symbol table (top of the stack).
        // Note: when encounter a function declaration/definition, first identifier must
be the function name
        if ( insideFunctionDecOrDef ) {

symbolTableStack.enterGlobal(identifierName).setAttribute(SymbolTableKey.TYPE,
SymbolTableType.FUNCTION);
            insideFunctionDecOrDef = false;
        }
        else if ( insideVariableDeclaration ) {
            symbolTableStack.enterLocal(identifierName).setAttribute(SymbolTableKey.TYPE,
SymbolTableType.VARIABLE);
            insideVariableDeclaration = false;
        }

        // Extract only variables inside their current belong FOR loop
        if ( insideTopLevelForLoopNode ) {

            // Retrieve identifier information
            SymbolTableEntry tableEntry = null;
            SymbolTableType entryType = null;
            if ( (tableEntry = symbolTableStack.lookup(identifierName)) != null ) {
                entryType =
(SymbolTableType)tableEntry.getAttribute(SymbolTableKey.TYPE);
            }

            // Function calls and #define definitions MUST NOT be added to the variable
set.
            if ( (entryType == null) ||
                 (entryType != null && entryType != SymbolTableType.DEFINE && entryType
!= SymbolTableType.FUNCTION) ) {

                boolean onLeftSideOfAssignment =
((Boolean)node.getAttribute(NodeKey.MODIFYING_IDENTIFIER)).booleanValue();
                boolean isReadOnly = true;
                boolean isReduction = false;
                String reductionOp = null;

                // Determine if this currently being visited identifier (variable)
                // is being updated or modified.
                if ( onLeftSideOfAssignment
                        || insidePostIncrementDecrementExpression
                        || insidePreIncrementDecrementExpression ) {
                    isReadOnly = false;
                }

                // Determine if this currently being visited identifier (variable)
                // is part of reduction clause.
                if ( insideAssignmentExpressionAndReduction ) {
```

```
                    isReduction = true;
                    reductionOp = assignmentExpressionOp;
                    insideAssignmentExpressionAndReduction = false;
                    assignmentExpressionOp = null;
                }
                else if ( insidePostfixExpressionAndReduction ) {
                    isReduction = true;
                    reductionOp = postfixExpressionOp;
                    insidePostfixExpressionAndReduction = false;
                    postfixExpressionOp = null;
                }

                // Add current encountered variable to the set if it does not exist,
                // otherwise, update it if necessary.
                Variable var = new Variable(identifierName, isReadOnly);
                var.setReduction(isReduction);
                var.setReductionOp(reductionOp);
                SimpleNode pNode = getParentForLoop(node);
                assert(pNode != null);
                if ( pNode instanceof ASTforStatement ) {
                    ForLoopNode forLoopNode =
(ForLoopNode)pNode.getAttribute(NodeKey.FOR_LOOP_NODE);
                    HashSet<Variable> setOfVariables = forLoopNode.getSetOfVariables();
                    if ( setOfVariables.contains(var) ) {
                        if ( (isReduction) || (!isReadOnly) ) {
                            setOfVariables.remove(var);
                        }
                    }
                    setOfVariables.add(var);
                    forLoopNode.setSetOfVariables(setOfVariables);
                }
            }
        }

        // ---------------------------------
        // Traversing down the tree...
        // ---------------------------------
        Object obj = super.visit(node, data);

        return obj;
    }

    /**
     * Increments operations count inside the current FOR loop.
     * @param currentNode a current operator node
     */
    private void incrementOperationsCount(SimpleNode currentNode) {
        SimpleNode pNode = getParentForLoop(currentNode);
        assert(pNode != null);
        if ( pNode instanceof ASTforStatement ) {
            ForLoopNode forLoopNode =
(ForLoopNode)pNode.getAttribute(NodeKey.FOR_LOOP_NODE);
            forLoopNode.incrementOperationsCountBy(1);
        }
    }

    /**
     * Retrieves parent node that is a FOR loop.
     * @param node current node
     * @return parent node that is a FOR loop, null otherwise
     */
    private SimpleNode getParentForLoop(SimpleNode node) {
        SimpleNode pNode = null;
        pNode = (SimpleNode)node.jjtGetParent();
        while ( !(pNode instanceof ASTforStatement) && pNode != null ) {
            pNode = (SimpleNode)pNode.jjtGetParent();
        }
        return pNode;
    }

    /**
```

```
     * Extracts array-typed postfixExpression nodes,
     * if any, that belong inside a subtree.
     * @param node root node of the subtree
     * @return list of array-typed postfixExpression nodes
     */
    private ArrayList<SimpleNode> extractArrayPostfixNodes(SimpleNode node) {
        ArrayList<SimpleNode> nodeList = new ArrayList<SimpleNode>();
        extractArrayPostfixNodesHelper(node, nodeList);
        return nodeList;
    }

    /**
     * Recursively walks the tree in pre-order breadth-first traversal
     * to extract array-typed postfixExpression nodes.
     * @param node root node of the tree
     * @param nodeList list to store extracted nodes
     */
    private void extractArrayPostfixNodesHelper(SimpleNode node, ArrayList<SimpleNode>
nodeList) {

        // Base case where this is a leaf node
        if ( node.jjtGetNumChildren() == 0 ) {
            return;
        }

        // Base case where only interested in array-postfix expression node,
        // skip over all other postfixExpression nodes
        else if ( node instanceof ASTpostfixExpression ) {
            boolean isArrayPostfixed =
((Boolean)node.getAttribute(NodeKey.ARRAY_POSTFIXED)).booleanValue();
            if ( isArrayPostfixed == true ) {
                nodeList.add(node);
            }
            return;
        }

        // Recursively traverse the tree
        else {
            for ( int i = 0; i < node.jjtGetNumChildren(); i++ ) {
                SimpleNode aNode = (SimpleNode)node.jjtGetChild(i);
                extractArrayPostfixNodesHelper(aNode, nodeList);
            }
        }
    }

    /**
     * Determines whether loop-carried dependence exists in a FOR loop.
     * @param leftArrays list of array-typed postfix expression nodes on left of an
assignment inside this FOR loop
     * @param rightArrays list of array-typed postfix expression nodes on right of an
assignment inside this FOR loop
     * @return true if this FOR loop has loop-carred dependence, false otherwise
     */
    private boolean hasLoopCarriedDependence(ArrayList<SimpleNode> leftArrays,
ArrayList<SimpleNode> rightArrays) {

        /* Interested only in this form of loop-carried dependence:
         *
         *    a[i] = a[exp_of_i]..., where:
         *    a is an array
         *    i is identifier index
         *    exp_of_i is an expression containing i
         *
         */

        for ( int i = 0; i < leftArrays.size(); i++ ) {
            SimpleNode leftArrayNode = leftArrays.get(i);
            SimpleNode leftArrayNameNode = (SimpleNode)leftArrayNode.jjtGetChild(0);
            SimpleNode leftArrayIndexNode =
(SimpleNode)leftArrayNode.jjtGetChild(1).jjtGetChild(0);
```

```
            for ( int j = 0; j < rightArrays.size(); j++ ) {
                SimpleNode rightArrayNode = rightArrays.get(j);
                SimpleNode rightArrayNameNode =
(SimpleNode)rightArrayNode.jjtGetChild(0);
                SimpleNode rightArrayIndexNode =
(SimpleNode)rightArrayNode.jjtGetChild(1).jjtGetChild(0);

                String leftArrayName =
(String)leftArrayNameNode.getAttribute(NodeKey.NAME);
                String rightArrayName =
(String)rightArrayNameNode.getAttribute(NodeKey.NAME);

                // Same array name?
                if ( leftArrayName.equals(rightArrayName) ) {

                    // Left array index should be an identifier and the identifier
                    // should appear inside the rightArrayIndexNode
                    if ( (leftArrayIndexNode instanceof ASTidentifier) &&
                         (doesExpressionContainIdentifier(rightArrayIndexNode,
leftArrayIndexNode) == true) ) {

                        // Different index distance?
                        if (
!(leftArrayIndexNode.getClass().equals(rightArrayIndexNode.getClass())) ) {

                            // Same C array but different index distance;
                            // therefore, a loop-carried dependence exist in this FOR
loop
                            return true;
                        }
                    }
                }
            }
        }

        // This FOR loop has no loop-carried dependence
        return false;
    }

    /**
     * Determine whether an expression containing an identifier
     * @param expression the target expression
     * @param identifier an interested identifier
     * @return true if the expression containing the identifier, false otherwise
     */
    private boolean doesExpressionContainIdentifier(SimpleNode expression, SimpleNode
identifier) {
        boolean found = false;

        // Base case where expression is identifier node
        if ( expression instanceof ASTidentifier ) {
            String name1 = (String)identifier.getAttribute(NodeKey.NAME);
            String name2 = (String)expression.getAttribute(NodeKey.NAME);
            if ( name1.equals(name2) ) {
                return true;
            }
        }

        // Base case where expression is a leaf node that is not an identifier
        else if ( expression.jjtGetNumChildren() == 0 ) {
            return false;
        }

        // Recursively traverse the tree
        else {
            for ( int i = 0; i < expression.jjtGetNumChildren(); i++ ) {
                SimpleNode aNode = (SimpleNode)expression.jjtGetChild(i);
                found = doesExpressionContainIdentifier(aNode, identifier);
                if ( found == true ) break;
            }
        }
```

```
        return found;
    }

    /**
     * Throws AutoParallelizerRestrictionException to indicate a program restriction
violation.
     * @param message a message about the exception
     * @throws Exception the AutoParallelizerRestrictionException
     */
    private void throwAutoParallelizerRestrictionException(String message) throws
Exception {
        throw new AutoParallelizerRestrictionException(message);
    }

}
```

```
/*
 * ForLoopNode.java
 *
 * Created on March 16, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


import java.util.ArrayList;
import java.util.HashSet;

import javax.swing.tree.DefaultMutableTreeNode;



/**
 * Node to represent a FOR loop structure
 * found in a C language source file.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class ForLoopNode extends DefaultMutableTreeNode {

    /** Begin line number of this FOR loop. */
    private int beginLine;

    /** Begin column number of this FOR loop. */
    private int beginColumn;

    /** Nested level of this FOR loop. */
    private int nestedLevel;

    /** Nesting levels of this FOR loop. */
    private int nestingLevels;

    /** Number of children that this FOR loop has. */
    private int numberOfChildren;

    /** Number of iterations of this FOR loop. */
    private int numberOfIterations;

    /** Count number of operations inside this FOR loop. */
    private int operationsCount;
```

```
/** OpenMP pragma specified for this FOR loop, if any. */
private String ompPragma;

/** Set of variables inside this FOR loop. */
private HashSet<Variable> setOfVariables = new HashSet<Variable>();

/** Loop-carried dependence status of this FOR loop. */
private boolean loopCarriedDependence;

/** Name of iteration variable. */
private String iterationVariableName;

/** Name of number-of-iteration variable. */
private String numberIterationsVariableName;

/**
 * Constructor.
 */
public ForLoopNode() {
    beginLine = 0;
    beginColumn = 0;
    nestedLevel = 0;
    nestingLevels = 0;
    numberOfIterations = 0;
    operationsCount = 0;
    ompPragma = null;
    loopCarriedDependence = false;
    iterationVariableName = null;
    numberIterationsVariableName = null;
}

/**
 * Constructor.
 * @param line begin line number
 * @param column begin column number
 * @param nestedLevel nested level of the loop
 * @param nestingLevel nesting levels of the loop
 */
public ForLoopNode(
        int line, int column, int nestedLevel, int nestingLevels,
        int numIterations, int opsCount, String ompPragma,
        boolean carriedDependence, String itVarName, String numItVarName) {
    setBeginLine(line);
    setBeginColumn(column);
    setNestedLevel(nestedLevel);
    setNestingLevels(nestingLevels);
    setNumberOfIterations(numIterations);
    setOperationsCount(opsCount);
    setOpenMPPragma(ompPragma);
    setLoopCarriedDependence(carriedDependence);
    setIterationVariableName(itVarName);
    setNumberIterationsVariableName(numItVarName);
}

/**
 * Sets the begin line number.
 * @param line begin line number
 */
public void setBeginLine(int line) {
    beginLine = line;
}

/**
 * Sets the begin column number.
 * @param column begin column number
 */
public void setBeginColumn(int column) {
    beginColumn = column;
}

/**
```

```
 * Sets nested level of the loop.
 * @param level nested level of the loop
 */
public void setNestedLevel(int level) {
    nestedLevel = level;
}

/**
 * Sets nesting levels of the loop.
 * @param levels nesting levels of the loop
 */
public void setNestingLevels(int levels) {
    nestingLevels = levels;
}

/**
 * Sets number of children that the loop has.
 * @param children children count of the loop
 */
public void setNumberOfChildren(int children) {
    numberOfChildren = children;
}

/**
 * Sets number of iterations of the loop.
 * @param numIterations number of iterations of the loop
 */
public void setNumberOfIterations(int numIterations) {
    numberOfIterations = numIterations;
}

/**
 * Sets number of operations inside the loop.
 * @param opsCount number of operations inside the loop
 */
public void setOperationsCount(int opsCount) {
    operationsCount = opsCount;
}

/**
 * Sets OpenMP pragma for this FOR loop, if any.
 * @param ompPragma the OpenMP pragma specified for this FOR loop
 */
public void setOpenMPPragma(String ompPragma) {
    this.ompPragma = ompPragma;
}

/**
 * Give the loop a set of variables found inside it.
 * @param variables set of variables
 */
public void setSetOfVariables(HashSet<Variable> variables) {
    setOfVariables = variables;
}

/**
 * Sets loop-carried dependence status of this FOR loop.
 * @param hasDependence true/false dependency status
 */
public void setLoopCarriedDependence(boolean hasDependence) {
    loopCarriedDependence = hasDependence;
}

/**
 * Set name of iteration variable of this FOR loop.
 * @param itVarName the name of the variable
 */
public void setIterationVariableName(String itVarName) {
    iterationVariableName = itVarName;
}
```

```
/**
 * Set name of number-of-iteration variable of this FOR loop.
 * @param numItVarName the name of the variable
 */
public void setNumberIterationsVariableName(String numItVarName) {
    numberIterationsVariableName = numItVarName;
}

/**
 * Gets the begin line number.
 * @return the begin line number
 */
public int getBeginLine() {
    return beginLine;
}

/**
 * Gets the begin column number.
 * @return the begin column number
 */
public int getBeginColumn() {
    return beginColumn;
}

/**
 * Gets the nested level of the loop.
 * @return nested level of the loop
 */
public int getNestedLevel() {
    return nestedLevel;
}

/**
 * Gets the nesting levels of the loop.
 * @return nesting levels of the loop
 */
public int getNestingLevels() {
    return nestingLevels;
}

/**
 * Gets the number of children that the loop has.
 * @return number of children of the loop
 */
public int getNumberOfChildren() {
    return numberOfChildren;
}

/**
 * Gets the number of iterations of the loop.
 * @return number of iterations of the loop
 */
public int getNumberOfIterations() {
    return numberOfIterations;
}

/**
 * Gets number of operations inside the loop.
 * @return number of operations inside the loop
 */
public int getOperationsCount() {
    return operationsCount;
}

/**
 * Gets the OpenMP pragma specified for this FOR loop.
 * @return the OpenMP pragma specified for this FOR loop, null otherwise
 */
public String getOpenMPPragma() {
    return ompPragma;
}
```

```
/**
 * Gets the set of variables inside this loop.
 * @return the set of variables inside this loop
 */
public HashSet<Variable> getSetOfVariables() {
    return setOfVariables;
}

/**
 * Gets loop-carried dependence status of this FOR loop;
 * @return true if the loop has the dependence, false otherwise
 */
public boolean hasLoopCarriedDependence() {
    return loopCarriedDependence;
}

/**
 * Gets the name of iteration variable.
 * @return the name of the variable
 */
public String getIterationVariableName() {
    return iterationVariableName;
}

/**
 * Gets the name of number-of-iteration variable.
 * @return the name of the variable
 */
public String getNumberIterationsVariableName() {
    return numberIterationsVariableName;
}

/**
 * Retrieves list of private variables inside the loop.
 * Private variables are those being modified and
 * should contribute to the list for OpenMP private clause.
 * @return list of variables being modified
 */
public ArrayList<Variable> getPrivateVariables() {
    ArrayList<Variable> vars = new ArrayList<Variable>();
    for ( Variable v : setOfVariables ) {
        if ( !v.isReadOnly() ) {
            vars.add(v);
        }
    }
    return vars;
}

/**
 * Retrieves names of private variables inside the loop.
 * Private variables are those being modified and
 * should contribute to the list for OpenMP private clause.
 * @return names of variables being modified (for OpenMP private clause)
 */
public ArrayList<String> getPrivateVariableNames() {
    ArrayList<Variable> privateVars = getPrivateVariables();
    ArrayList<String> varNames = new ArrayList<String>();
    for ( Variable v : privateVars ) {
        varNames.add(v.getName());
    }
    return varNames;
}

/**
 * Retrieves list of shared variables inside the loop.
 * Shared variables are those being read only and
 * should contribute to the list for OpenMP shared clause.
 * @return list of variables being read only
 */
public ArrayList<Variable> getSharedVariables() {
```

```
    ArrayList<Variable> vars = new ArrayList<Variable>();
    for ( Variable v : setOfVariables ) {
        if ( v.isReadOnly() ) {
            vars.add(v);
        }
    }
    return vars;
}

/**
 * Retrieves names of shared variables inside the loop.
 * Shared variables are those being read only and
 * should contribute to the list for OpenMP shared clause.
 * @return names of variables being read only (for OpenMP shared clause)
 */
public ArrayList<String> getSharedVariableNames() {
    ArrayList<Variable> sharedVariables = getSharedVariables();
    ArrayList<String> varNames = new ArrayList<String>();
    for ( Variable v : sharedVariables ) {
        varNames.add(v.getName());
    }
    return varNames;
}

/**
 * Retrieves list of reduction variables inside the loop.
 * @return list of variables to be in OpenMP reduction clause
 */
public ArrayList<Variable> getReductionVariables() {
    ArrayList<Variable> vars = new ArrayList<Variable>();
    for ( Variable v : setOfVariables ) {
        if ( v.isReduction() ) {
            vars.add(v);
        }
    }
    return vars;
}

/**
 * Retrieves names of reduction variables inside the loop.
 * @return names of variables to be in OpenMP reduction clause
 */
public ArrayList<String> getReductionVariableNames() {
    ArrayList<Variable> reductionVariables = getReductionVariables();
    ArrayList<String> varNames = new ArrayList<String>();
    for ( Variable v : reductionVariables ) {
        varNames.add(v.getName());
    }
    return varNames;
}

/**
 * Retrieves list of array typed variables inside the loop.
 * @return list of array typed variables inside the loop
 */
public ArrayList<Variable> getArrayTypedVariables() {
    ArrayList<Variable> vars = new ArrayList<Variable>();
    for ( Variable v : setOfVariables ) {
        if ( v.getType() == VariableType.ARRAY ) {
            vars.add(v);
        }
    }
    return vars;
}

/**
 * Retrieves names of arrays inside the loop.
 * @return names of arrays inside the loop
 */
public ArrayList<String> getArrayTypedVariableNames() {
    ArrayList<Variable> arrayTypedVariables = getArrayTypedVariables();
```

```
        ArrayList<String> varNames = new ArrayList<String>();
        for ( Variable v : arrayTypedVariables ) {
            varNames.add(v.getName());
        }
        return varNames;
    }

    /**
     * Retrieves number of variables inside the loop.
     * @return number of variables inside the loop
     */
    public int getNumberOfVariables() {
        return getSetOfVariables().size();
    }

    /**
     * Retrieves number of private variables inside the loop.
     * @return number of private variables inside the loop
     */
    public int getNumberOfPrivateVariables() {
        return getPrivateVariables().size();
    }

    /**
     * Retrieves number of shared variables inside the loop.
     * @return number of shared variables inside the loop
     */
    public int getNumberOfSharedVariables() {
        return getSharedVariables().size();
    }

    /**
     * Retrieves number of arrays inside the loop.
     * @return number of arrays inside the loop
     */
    public int getNumberOfArrayTypedVariables() {
        return getArrayTypedVariables().size();
    }

    /**
     * Increments operations count by a specific amount
     * @param count amount to add to current operations count
     */
    public void incrementOperationsCountBy(int count) {
        operationsCount += count;
    }

    @Override
    public String toString() {
        return "ForLoopNode [beginLine=" + beginLine + ", beginColumn="
                + beginColumn + "]";
    }
}
```

```
/*
 * ForLoopTree.java
 *
 * Created on March 11, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;
```

```
import java.util.Enumeration;

import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;


/**
 * Tree structure of ForLoopNode(s)
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class ForLoopTree {

    /** The tree structure. */
    private DefaultTreeModel forLoopTree;

    /** Most recent inserted node. */
    private ForLoopNode currentNode;

    /**
     * Constructor.
     */
    public ForLoopTree() {
        forLoopTree = null;
        currentNode = null;
    }

    /**
     * Constructor.
     * @param rootNode root node of the tree
     */
    public ForLoopTree(ForLoopNode rootNode) {
        forLoopTree = new DefaultTreeModel(rootNode);
        currentNode = rootNode;
    }

    /**
     * Inserts a new node.
     * @param node the node to be inserted
     */
    public void insert(ForLoopNode node) {
        assert(forLoopTree != null);
        forLoopTree.insertNodeInto(node, currentNode, 0);
        currentNode = node;
    }

    /**
     * Inserts a new node.
     * @param node the node to be inserted
     * @param parent parent of the node
     * @param index index of parents children
     */
    public void insert(ForLoopNode node, ForLoopNode parent, int index) {
        assert(forLoopTree != null);
        forLoopTree.insertNodeInto(node, parent, index);
        currentNode = node;
    }

    /**
     * Sets a new root node for the tree.
     * @param node a new root node
     */
    public void setRootNode(ForLoopNode node) {
        assert(forLoopTree != null);
        forLoopTree.setRoot(node);
        currentNode = node;
    }

    /**
```

```
 * Gets the tree root node.
 * @return the tree root node.
 */
public Object getRootNode() {
    return forLoopTree.getRoot();
}

/**
 * Gets a child node.
 * @param parent parent node whose child being retrieved
 * @param index the parents children index
 * @return a child node
 */
public Object getChild(Object parent, int index) {
    return forLoopTree.getChild(parent, index);
}

/**
 * Gets index of the child.
 * @param parent parent of the child
 * @param child the child whose index being retrieved
 * @return index of the child
 */
public int getChildIndex(Object parent, Object child) {
    return forLoopTree.getIndexOfChild(parent, child);
}

/**
 * Gets number of children that the parent has
 * @param parent the parent node
 * @return number of children
 */
public int getNumberOfChildren(Object parent) {
    return forLoopTree.getChildCount(parent);
}

/**
 * Determines whether the node is a leaf node.
 * @param node node to be determined
 * @return true of node is a leaf node, false otherwise
 */
public boolean isLeafNode(Object node) {
    return forLoopTree.isLeaf(node);
}

/**
 * Is this tree empty?
 * @return true if the tree is empty, false otherwise
 */
public boolean isEmpty() {
    return (forLoopTree == null);
}

/**
 * Constructs all nodes of the tree in depth-first order fashion.
 * @return an enumeration of the nodes
 */
public Enumeration depthFirstTraversal() {
    ForLoopNode root = (ForLoopNode)forLoopTree.getRoot();
    return root.depthFirstEnumeration();
}

/**
 * Constructs all nodes of the tree in breadth-first order fashion.
 * @return an enumeration of the nodes
 */
public Enumeration breadthFirstTraversal() {
    ForLoopNode root = (ForLoopNode)forLoopTree.getRoot();
    return root.breadthFirstEnumeration();
}
```

```
    /**
     * Prints all nodes of the tree in breadth-first order.
     */
    public void printTree() {
        ForLoopNode root = (ForLoopNode)forLoopTree.getRoot();
        Enumeration e = root.breadthFirstEnumeration();

        while ( e.hasMoreElements() ) {
            System.out.println((ForLoopNode)e.nextElement());
        }
        System.out.println();
    }

}
```

```
/*
 * NodeKey.java
 *
 * Created on March 16, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


/**
 * Attribute of an AST node.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public enum NodeKey {

    NAME, VALUE,

    BEGIN_LINE, BEGIN_COLUMN,

    SPECIAL_TOKEN,

    MODIFYING_IDENTIFIER,

    ARRAY_POSTFIXED,

    CALLED_FROM,

    NUM_CHILDREN,

    FOR_LOOP_NODE,

    OPERATOR,

    REDUCTION,

    REDUCTION_OP,

    FUNCTION_DEC_DEF,

    CURRENT_NESTING_LEVELS,
}
```

```
/*
```

```
 * NodeType.java
 *
 * Created on March 19, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


/**
 * Type of an AST node.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public enum NodeType {

    JUMP_STMT_NODE,

    EXPRESSION_STMT_NODE,

    IF_STMT_NODE,

    SWITCH_STMT_NODE,

    FOR_STMT_NODE,

    WHILE_STMT_NODE,

    DO_WHILE_STMT_NODE,

    CONDITIONAL_EXPRESSION_NODE,

    POSTFIX_EXPRESSION_NODE,

    PRIMARY_EXPRESSION_NODE,

    NULL,
}
```

```
/*
 * NotSimpleNode.java
 *
 * Created on March 16, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


import java.util.HashMap;


/**
 * This class is to be extended by SimpleNode generated by JavaCC
 * in order to enhance node features and capabilities of the AST.
 *
```

```
 * @author Nam Q. Lam
 * @version 1.0
 */
public class NotSimpleNode {

	/** The version identifier for this Serializable class. */
	private static final long serialVersionUID = 1351059215137481754L;

	/** Attributes of this node are store in this HashMap. */
	private HashMap<NodeKey, Object> nodeAttributes;

	/**
	 * Constructor.
	 */
	public NotSimpleNode() {
		nodeAttributes = new HashMap<NodeKey, Object>();
	}

	/**
	 * Sets attributes for this node.
	 * @param key key of an attribute being set
	 * @param value corresponding value of the attribute key
	 */
	public void setAttribute(NodeKey key, Object value) {
		nodeAttributes.put(key, value);
	}

	/**
	 * Gets an attribute from this node.
	 * @param key the attribute key
	 * @return the corresponding value of the attribute key
	 */
	public Object getAttribute(NodeKey key) {
		return nodeAttributes.get(key);
	}
}
```

```
/*
 * SymbolTable.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


import java.util.*;


/**
 * Symbol table used to store symbol table entries
 * at a particular nesting level or scope.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class SymbolTable {

    /** Nesting level of this symbol table. */
    private int nestingLevel;

    /** Data structure for symbol table entries. */
```

```
    private HashMap<String, SymbolTableEntry> entries;

    /**
     * Constructor.
     * @param nestingLevel nesting level of this symbol table
     */
    public SymbolTable(int nestingLevel) {
        this.nestingLevel = nestingLevel;
        entries = new HashMap<String, SymbolTableEntry>();
    }

    /**
     * Sets symbol table nesting level.
     * @param nestingLevel nesting level of this symbol table
     */
    public void setNestingLevel(int nestingLevel) {
        this.nestingLevel = nestingLevel;
    }

    /**
     * Gets symbol table nesting level.
     * @return nesting level of this symbol table
     */
    public int getNestingLevel() {
        return nestingLevel;
    }

    /**
     * Enters a new entry into this symbol table.
     * @param name name of the new entry
     * @return a newly create symbol table entry
     */
    public SymbolTableEntry enter(String name) {
        SymbolTableEntry newEntry = SymbolTableFactory.createSymbolTableEntry(name,
this);
        entries.put(name, newEntry);
        return newEntry;
    }

    /**
     * Looks up for an existing symbol table entry.
     * @param name name of an interested symbol table entry
     * @return found existing symbol table entry, null otherwise
     */
    public SymbolTableEntry lookup(String name) {
        return entries.get(name);
    }

}
```

```
/*
 * SymbolTableEntry.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


import java.util.*;
```

```
/**
 * Symbol table entry used to store any useful information
 * about parsing identifiers, variables, and functions.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class SymbolTableEntry {

    /** Entry name. */
    private String name;

    /** Symbol table that this entry belongs to. */
    private SymbolTable symbolTable;

    /** Data structure for entry attributes. */
    private HashMap<SymbolTableKey, Object> attributes;

    /**
     * Constructor.
     * @param name name of a new entry
     * @param symbolTable symbol table that the entry belongs to
     */
    public SymbolTableEntry(String name, SymbolTable symbolTable) {
        this.name = name;
        this.symbolTable = symbolTable;
        attributes = new HashMap<SymbolTableKey, Object>();
    }

    /**
     * Sets a new name of the entry.
     * @param name a new name of the entry
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Sets a new symbol table containing this entry.
     * @param symbolTable a new symbol table containing this entry
     */
    public void setSymbolTable(SymbolTable symbolTable) {
        this.symbolTable = symbolTable;
    }

    /**
     * Sets entry attribute.
     * @param key attribute key
     * @param value attribute value
     */
    public void setAttribute(SymbolTableKey key, Object value) {
        attributes.put(key, value);
    }

    /**
     * Gets the entry name.
     * @return name of the entry
     */
    public String getName() {
        return name;
    }

    /**
     * Gets this entry's symbol table.
     * @return symbol table of the entry
     */
    public SymbolTable getSymbolTable() {
        return symbolTable;
    }

    /**
```

```
     * Gets entry attribute.
     * @param key attribute key
     * @return found attribute value, null otherwise
     */
    public Object getAttribute(SymbolTableKey key) {
        return attributes.get(key);
    }

}
```

```
/*
 * SymbolTableFactory.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


/**
 * Factory to create symbol table related items.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class SymbolTableFactory {

    /**
     * Creates a new SymbolTableStack.
     * @return newly created SymbolTableStack
     */
    public static SymbolTableStack createSymbolTableStack() {
        return new SymbolTableStack();
    }

    /**
     * Creates a new SymbolTable.
     * @param nestingLevel interested nesting level
     * @return newly created SymbolTable
     */
    public static SymbolTable createSymbolTable(int nestingLevel) {
        return new SymbolTable(nestingLevel);
    }

    /**
     * Creates a new SymbolTableEntry.
     * @param name name of a new entry
     * @param symbolTable symbol table containing the entry
     * @return newly created SymbolTableEntry
     */
    public static SymbolTableEntry createSymbolTableEntry(String name, SymbolTable
symbolTable) {
        return new SymbolTableEntry(name, symbolTable);
    }

}
```

```
/*
 * SymbolTableKey.java
 *
```

```
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


/**
 * Attributes of a symbol table entry.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public enum SymbolTableKey {

    TYPE,

    VALUE,

}
```

```
/*
 * SymbolTableStack.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


import java.util.ArrayList;


/**
 * A stack structure of symbol tables.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class SymbolTableStack {

    /** Current nesting level/scope or top of stack. */
    private int currentNestingLevel;

    /** Data structure for symbol tables. */
    private ArrayList<SymbolTable> symbolTables;

    /**
     * Constructor.
     */
    public SymbolTableStack() {
        currentNestingLevel = 0;
        symbolTables = new ArrayList<SymbolTable>();
        symbolTables.add(SymbolTableFactory.createSymbolTable(currentNestingLevel));
    }
```

```
    /**
     * Sets current nesting level of the stack.
     * @param currentNestingLevel current nesting level
     */
    public void setCurrentNestingLevel(int currentNestingLevel) {
        this.currentNestingLevel = currentNestingLevel;
    }

    /**
     * Gets current nesting level of the stack.
     * @return current nesting level
     */
    public int getCurrentNestingLevel() {
        return currentNestingLevel;
    }

    /**
     * Inserts a new symbol table on top of stack.
     * @return the newly created symbol table
     */
    public SymbolTable push() {
        currentNestingLevel++;
        SymbolTable symbolTable =
SymbolTableFactory.createSymbolTable(currentNestingLevel);
        symbolTables.add(symbolTable);
        return symbolTable;
    }

    /**
     * Inserts a symbol table on top of stack.
     * @param symbolTable symbol table to be inserted
     * @return the inserted symbol table
     */
    public SymbolTable push(SymbolTable symbolTable) {
        currentNestingLevel++;
        symbolTable.setNestingLevel(currentNestingLevel);
        symbolTables.add(symbolTable);
        return symbolTable;
    }

    /**
     * Removes the symbol table on top of stack.
     * @return the top-of-stack symbol table
     */
    public SymbolTable pop() {
        SymbolTable symbolTable = symbolTables.get(currentNestingLevel);
        symbolTables.remove(currentNestingLevel);
        currentNestingLevel--;
        return symbolTable;
    }

    /**
     * Enters a new symbol table entry into top-of-stack symbol table.
     * @param name name of the new symbol table entry
     * @return the newly created symbol table entry
     */
    public SymbolTableEntry enterLocal(String name) {
        return symbolTables.get(currentNestingLevel).enter(name);
    }

    /**
     * Enters a new symbol table entry into bottom-of-stack symbol table.
     * @param name name of the new symbol table entry
     * @return the newly created symbol table entry
     */
    public SymbolTableEntry enterGlobal(String name) {
        return symbolTables.get(0).enter(name);
    }

    /**
     * Looks up for an existing symbol table entry
```

```
     * in symbol table on top of stack.
     * @param name name of an interested symbol table entry
     * @return found existing symbol table entry, null otherwise
     */
    public SymbolTableEntry lookupLocal(String name) {
        return symbolTables.get(currentNestingLevel).lookup(name);
    }

    /**
     * Looks up for an existing symbol table entry
     * in entire stack, starting from top of stack.
     * @param name name of an interested symbol table entry
     * @return first symbol table entry found, null otherwise
     */
    public SymbolTableEntry lookup(String name) {
        SymbolTableEntry foundEntry = null;
        for ( int i = currentNestingLevel; i >= 0; i-- ) {
            foundEntry = symbolTables.get(i).lookup(name);
            if ( foundEntry != null ) break;
        }
        return foundEntry;
    }

}
```

```
/*
 * SymbolTableType.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


/**
 * Type of a symbol table entry.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
package nql.cd.intermediate;


public enum SymbolTableType {

    DEFINE,

    FUNCTION,

    VARIABLE,

}
```

```
/*
 * Variable.java
 *
 * Created on March 17, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
```

```
 * ========================================
 * NOTES:
 * - Purely for purpose of this project,
 *   Variables are simply compared only
 *   base on its name.
 * ========================================
 *
 */


package nql.cd.intermediate;


/**
 * This is a variable in a C source file.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class Variable {

    /** A variable has a name. */
    private String name;

    /** Type of this variable. */
    private VariableType type;

    /** Value of this variable. */
    private Object value;

    /** Is this variable being modified or not? */
    private boolean readOnly;

    /** Should this variable be in OpenMP reduction clause? */
    private boolean reduction;

    /** Corresponding operator if this variable is reductible. */
    private String reductionOp;

    /**
     * Constructor.
     */
    public Variable() {}

    /**
     * Constructor.
     * @param variableName name of the variable
     * @param isReadOnly whether this variable being modified or not
     */
    public Variable(String variableName, boolean isReadOnly) {
        setName(variableName);
        setReadOnly(isReadOnly);
        setReduction(false);
        setReductionOp(null);
        setType(VariableType.NULL);
        setValue(null);
    }

    /**
     * Sets name of this variable.
     * @param variableName name of the variable
     */
    public void setName(String variableName) {
        name = variableName;
    }

    /**
     * Sets type of this variable.
     * @param variableType type of the variable
     */
    public void setType(VariableType variableType) {
```

```
    type = variableType;
}

/**
 * Sets value of this variable.
 * @param value value of the variable
 */
public void setValue(Object value) {
    this.value = value;
}

/**
 * Sets read only status of this variable.
 * @param isReadOnly true if being read only, false otherwise
 */
public void setReadOnly(boolean isReadOnly) {
    readOnly = isReadOnly;
}

/**
 * Sets reduction status of this variable.
 * @param isReduction true if reduction, false otherwise
 */
public void setReduction(boolean isReduction) {
    reduction = isReduction;
}

/**
 * Sets reduction operator.
 * @param op reduction operator
 */
public void setReductionOp(String op) {
    reductionOp = op;
}

/**
 * Retrieves the name of this variable.
 * @return name of this variable
 */
public String getName() {
    return name;
}

/**
 * Retrieves the type of this variable.
 * @return type of this variable
 */
public VariableType getType() {
    return type;
}

/**
 * Retrieves current value of this variable.
 * @return current value of this variable
 */
public Object getValue() {
    return value;
}

/**
 * Retrieve read only status of this variable.
 * @return true if being read only, false otherwise
 */
public boolean isReadOnly() {
    return readOnly;
}

/**
 * Retrieve reduction status of this variable.
 * @return true if reduction, false otherwise
 */
```

```
    public boolean isReduction() {
        return reduction;
    }

    /**
     * Gets reduction operator of this reductible variable.
     * @return reduction operator
     */
    public String getReductionOp() {
        return reductionOp;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Variable other = (Variable) obj;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "Variable [name=" + name + "]";
    }
}
```

```
/*
 * VariableType.java
 *
 * Created on March 19, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.cd.intermediate;


/**
 * Type of a Variable (identifier node in the AST).
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public enum VariableType {

    ARRAY,
```

```
    INTEGER,

    REAL,

    STRING,

    CHARACTER,

    NULL,
}
```

```
/*
 * AutoParallelizerRestrictionException.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.exceptions;


/**
 * An exception to indicate a violation of program restrictions.
 * @author Nam Q. Lam
 * @version 1.0
 */
public class AutoParallelizerRestrictionException extends Exception {

    /** Exception message of this violation. */
    private String message;

    /**
     * Constructor.
     */
    public AutoParallelizerRestrictionException() {
        super();
        message = "A program restriction violation has occurred.";
    }

    /**
     * Constructor.
     * @param errMsg exception message of this violation
     */
    public AutoParallelizerRestrictionException(String errMsg) {
        super(errMsg);
        message = errMsg;
    }

}
```

```
/*
 * LoopCarriedDependenceException.java
 *
 * Created on September 1, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */
```

```
package nql.exceptions;


/**
 * An exception to indicate a loop-carried dependence.
 * @author Nam Q. Lam
 * @version 1.0
 */
public class LoopCarriedDependenceException extends Exception {

    /** Exception message of this dependency. */
    private String message;

    /**
     * Constructor.
     */
    public LoopCarriedDependenceException() {
        super();
        message = "Loop-carried dependence found.";
    }

    /**
     * Constructor.
     * @param errMsg exception message of this dependency
     */
    public LoopCarriedDependenceException(String errMsg) {
        super(errMsg);
        message = errMsg;
    }

}
```

```
/*
 * InstanceBasedClassifier.java
 *
 * Created on February 14, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.ml.classifiers;


import java.io.File;
import java.io.Serializable;

import weka.classifiers.Classifier;
import weka.classifiers.lazy.IBk;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.core.converters.ConverterUtils.DataSource;


/**
 * The instance-based classifier interface.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
```

```
public interface InstanceBasedClassifier {

    /**
     * Adds a new training instance to the data set.
     * @param instance new training instance to be added to the data set
     */
    public void addTrainingInstance(Instance instance);

    /**
     * Adds a new training instance to the data set.
     * @param attributeValues String array of attribute values
     * @param classLabel the class label of the training instance
     */
    public void addTrainingInstance(String[] attributeValues, String classLabel);

    /**
     * Classifies a new instance.
     * @param instance a new instance to be classified
     * @return predicted class label of this instance
     * @throws Exception an exception that is propagated to the caller
     */
    public String classifyNewInstance(Instance instance) throws Exception;

    /**
     * Classifies a new instance.
     * @param attributeValues String array of attribute values
     * @return predicted class label of this instance
     * @throws Exception an exception that is propagated to the caller
     */
    public String classifyNewInstance(String[] attributeValues) throws Exception;

    /**
     * Saves the training data set to ARFF file format.
     */
    public void saveTrainingDataSetToArffFile() throws Exception;

    /**
     * Resets and re-initializes the classifier.
     * @throws Exception an exception that is propagated to the caller
     */
    public void reset() throws Exception;
}
```

```
/*
 * OpenMPForScheduleClassifier.java
 *
 * Created on February 14, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.ml.classifiers;


import java.io.File;
import java.io.Serializable;

import weka.classifiers.Classifier;
import weka.classifiers.lazy.IBk;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ArffSaver;
```

```
import weka.core.converters.ConverterUtils.DataSource;


/**
 * The instance-based classifier for OpenMP FOR loop constructs.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class OpenMPForScheduleClassifier implements InstanceBasedClassifier, Serializable
{

    /** The version identifier for this Serializable class. */
    private static final long serialVersionUID = 3293608295073600116L;

    /** Name of the OpenMP FOR loop construct training set. */
    private String nameOfDataSet = "OpenMPForLoopConstructTrainingSet";

    /** The OpenMP FOR loop construct training set. */
    private Instances trainingDataSet = null;

    /** ARFF file used to save the training data set. */
    private String trainingDataSetInArff = "obj\\TrainingDataSet.arff";

    /** The instance-based classifier to be trained and to perform classifying
operations. */
    private Classifier instanceBasedClassifier = new IBk();

    /** Is the classifier model up-to-date? */
    private boolean modelUpToDate = false;

    /**
     * Constructor.
     * @throws Exception an exception that is propagated to the caller
     */
    public OpenMPForScheduleClassifier() throws Exception {
        buildInMemoryDataset();
    }

    /**
     * Constructor.
     * @param trainingDataSet an initial training set
     * @throws Exception an exception that is propagated to the caller
     */
    public OpenMPForScheduleClassifier(Instances trainingDataSet) throws Exception {
        this.trainingDataSet = trainingDataSet;
        trainingDataSet.setClassIndex(trainingDataSet.numAttributes() - 1);
    }

    /**
     * Constructor.
     * @param trainingDataFileArff an initial training set in ARFF format
     * @throws Exception an exception that is propagated to the caller
     */
    public OpenMPForScheduleClassifier(String trainingDataFileArff) throws Exception {
        trainingDataSet = DataSource.read(trainingDataFileArff);
        trainingDataSet.setClassIndex(trainingDataSet.numAttributes() - 1);
    }

    /**
     * Adds a new training instance to the data set.
     * @param instance new training instance to be added to the data set
     */
    public void addTrainingInstance(Instance instance) {

        // Add the training instance to the data set
        trainingDataSet.add(instance);
        modelUpToDate = false;
    }

    /**
```

```
     * Adds a new training instance to the data set.
     * @param attributeValues String array of attribute values
     * @param classLabel the class label of the training instance
     */
    public void addTrainingInstance(String[] attributeValues, String classLabel) {

        // Attribute set
        Attribute numberOfThreads = trainingDataSet.attribute("num_threads");
        Attribute numberOfIterations = trainingDataSet.attribute("num_iterations");
        Attribute nestingLevels = trainingDataSet.attribute("nesting_levels");
        Attribute complexityUnits = trainingDataSet.attribute("complexity_units");

        // Instance to be added
        Instance instance = new Instance(5);
        instance.setValue(numberOfThreads, Integer.parseInt(attributeValues[0]));
        instance.setValue(numberOfIterations, Integer.parseInt(attributeValues[1]));
        instance.setValue(nestingLevels, Integer.parseInt(attributeValues[2]));
        instance.setValue(complexityUnits, Double.parseDouble(attributeValues[3]));
        instance.setDataset(trainingDataSet);
        instance.setClassValue(classLabel);

        // Add the training instance to the data set
        addTrainingInstance(instance);
    }

    /**
     * Classifies a new instance.
     * @param instance a new instance to be classified
     * @return predicted class label of this instance
     * @throws Exception an exception that is propagated to the caller
     */
    public String classifyNewInstance(Instance instance) throws Exception {

        // The classifier has not been trained yet
        if ( trainingDataSet.numInstances() == 0 ) {
            throw new Exception("Training data set is empty; no classifier has been
built.");
        }

        // Make sure the classifier model is up-to-date
        if ( !modelUpToDate ) {
            instanceBasedClassifier.buildClassifier(trainingDataSet);
        }

        // Perform classification, resulted in predicted label for this instance
        double predictedClass = instanceBasedClassifier.classifyInstance(instance);
        Attribute classAttribute = trainingDataSet.classAttribute();
        String classLabel = classAttribute.value((int)predictedClass);

        // The predicted class label of this instance
        return classLabel;
    }

    /**
     * Classifies a new instance.
     * @param attributeValues String array of attribute values
     * @return predicted class label of this instance
     * @throws Exception an exception that is propagated to the caller
     */
    public String classifyNewInstance(String[] attributeValues) throws Exception {

        // Attribute set
        Instances temp = new Instances(trainingDataSet);
        Attribute numberOfThreads = temp.attribute("num_threads");
        Attribute numberOfIterations = temp.attribute("num_iterations");
        Attribute nestingLevels = temp.attribute("nesting_levels");
        Attribute complexityUnits = temp.attribute("complexity_units");

        // Instance to be classified
        Instance instance = new Instance(5);
        instance.setValue(numberOfThreads, Integer.parseInt(attributeValues[0]));
```

```
        instance.setValue(numberOfIterations, Integer.parseInt(attributeValues[1]));
        instance.setValue(nestingLevels, Integer.parseInt(attributeValues[2]));
        instance.setValue(complexityUnits, Double.parseDouble(attributeValues[3]));
        instance.setDataset(temp);

        // Classify the instance and return predicted class label
        return classifyNewInstance(instance);
    }

    /**
     * Saves the training data set to ARFF file format.
     */
    public void saveTrainingDataSetToArffFile() throws Exception {
        ArffSaver saver = new ArffSaver();
        saver.setInstances(trainingDataSet);
        saver.setFile(new File(trainingDataSetInArff));
        saver.writeBatch();
    }

    /**
     * Resets and re-initializes the classifier.
     * @throws Exception an exception that is propagated to the caller
     */
    public void reset() throws Exception {
        trainingDataSet = null;
        instanceBasedClassifier = new IBk();
        modelUpToDate = false;
        buildInMemoryDataset();
    }

    /**
     * Builds an in-memory empty training data set.
     * @throws Exception an exception that is propagated to the caller
     */
    private void buildInMemoryDataset() throws Exception {

        // nominal values of class labels
        FastVector classValues = new FastVector();
        classValues.addElement("static");
        classValues.addElement("dynamic");
        classValues.addElement("guided");
        classValues.addElement("runtime");
        classValues.addElement("auto");

        // attributes
        FastVector attributes = new FastVector();
        attributes.addElement(new Attribute("num_threads"));
        attributes.addElement(new Attribute("num_iterations"));
        attributes.addElement(new Attribute("nesting_levels"));
        attributes.addElement(new Attribute("complexity_units"));
        attributes.addElement(new Attribute("schedule_type", classValues));

        // initial empty data set
        trainingDataSet = new Instances(nameOfDataSet, attributes, 0);
        trainingDataSet.setClassIndex(trainingDataSet.numAttributes() - 1);
    }
}
```

```
/*
 * ObjectSerializer.java
 *
 * Created on April 21, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */
```

```
package nql.utils;


import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;


/**
 * A utility class to read/write objects from/to files.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public final class ObjectSerializer {

    /**
     * Private constructor.
     */
    private ObjectSerializer() {
    }

    /**
     * Saves object to file.
     * @param outObj an object to be saved
     * @param fileName name of file to hold object's data
     */
    public static synchronized void writeObject(Object outObj, String fileName) {
        try {
            ObjectOutputStream output =
                new ObjectOutputStream(
                    new FileOutputStream(fileName));

            if (outObj != null) {
                output.writeObject(outObj);
            }

            if (output != null) {
                output.close();
            }

        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }

    /**
     * Reads object from file.
     * @param fileName name of file holding the object's data
     * @return the object previously saved, null other wise
     */
    public static synchronized Object readObject(String fileName) {
        Object inObj = null;

        try {
            ObjectInputStream input =
                new ObjectInputStream(
                    new FileInputStream(fileName));
            try {
                inObj = input.readObject();
            }
            catch ( EOFException e ) {
            } // OK!

            if (input != null) {
                input.close();
```

```
            }
        }
        catch ( EOFException e ) {
        } // OK!
        catch ( Exception e ) {
            e.printStackTrace();
        }

        return inObj;
    }
}
```

```
/*
 * SourceFile.java
 *
 * Created on April 20, 2011
 *
 * By Nam Q. Lam
 *
 * Computer Science Master Project
 * San Jose State University
 *
 */


package nql.utils;


import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;


/**
 * This class is to represent, hold, and manipulate
 * contents (simply are text lines) of a text file.
 *
 * @author Nam Q. Lam
 * @version 1.0
 */
public class SourceFile {

    /** Lines in the file. */
    private ArrayList<String> linesInFile = new ArrayList<String>();

    /**
     * Constructor.
     * @param fileName name of the file
     */
    public SourceFile(String fileName) {
        linesInFile = readLines(fileName);
    }

    /**
     * Adds a new line to file at specific location.
     * @param row line number
     * @param col column number
     * @param line the text line to be added
     */
    public void addLine(int row, int col, String line) {
        String newLine = "";
        for ( int i = 0; i < col; i++ ) {
            newLine += " ";
        }
        newLine += line;
```

```
        linesInFile.add(row, newLine);
    }

    /**
     * Retrieves a list of all lines in file.
     * @return a list of all lines in file
     */
    public ArrayList<String> getLines() {
        return linesInFile;
    }

    /**
     * Prints/saves the contents to file.
     * @param outputFileName name of the file to save to
     * @throws Exception file related exceptions to be handled by a caller
     */
    public void printTo(String outputFileName) throws Exception {

        PrintWriter writer = new PrintWriter(new BufferedWriter(new
FileWriter(outputFileName)), true);

        for ( int i = 0; i < linesInFile.size(); i++ ) {
            writer.println(linesInFile.get(i));
        }

        if ( writer != null ) {
            writer.close();
        }
    }

    /**
     * Prints/saves the contents to a string buffer.
     * @param sb the string buffer to save to
     */
    public void printTo(StringBuffer sb) {
        for ( int i = 0; i < linesInFile.size(); i++ ) {
            sb.append(linesInFile.get(i) + "\n");
        }
        sb.append("\n");
    }

    /**
     * Retrieves the contents/lines of the input file.
     * @param fileName name of input file
     * @return list of lines in the file
     */
    private ArrayList<String> readLines(String fileName) {
        ArrayList<String> lines = new ArrayList<String>();

        try {
            BufferedReader reader = new BufferedReader(new FileReader(fileName));
            String line = "";
            while ( (line = reader.readLine()) != null ) {
                lines.add(line);
            }
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }

        return lines;
    }

    /* (non-Javadoc)
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        StringBuffer sb = new StringBuffer("");
        for ( int i = 0; i < linesInFile.size(); i++ ) {
            sb.append(linesInFile.get(i) + "\n");
```

```
        }
        sb.append("\n");
        return sb.toString();
    }
}
```

```
/* Generated By:JJTree&JavaCC: Do not edit this line. CParser.java */
package nql.cd.frontend;
import nql.cd.intermediate.*;
import java.util.*;
import java.io.*;
public class CParser/*@bgen(jjtree)*/implements CParserTreeConstants, CParserConstants
{/*@bgen(jjtree)*/
  protected JJTCParserState jjtree = new JJTCParserState();
    /* Handle typedef types. */
    private Set<String> types = new HashSet<String>();
    private Stack<Boolean> typedefParsingStack = new Stack<Boolean>();
    private boolean isType(String type) {
            return types.contains(type);
    }
    private void addType(String type) {
            types.add(type);
    }

    /* This main entry is a test driver for the generated parser. */
    public static void main(String [] args) {
        try {
            StringBuffer sb = new StringBuffer();
            Reader reader = new FileReader(args[0]);
            CParser parser = new CParser(reader);
            SimpleNode rootNode = parser.parse();
            rootNode.dump(">", sb);
            System.out.println(sb.toString());
        }
        catch ( Exception ex ) {
            ex.printStackTrace();
        }
    }

/*
 * Syntactic Specification
 *
 */
  final public SimpleNode parse() throws ParseException {
                      /*@bgen(jjtree) parse */
  ASTparse jjtn000 = new ASTparse(JJTPARSE);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      label_1:
      while (true) {
        externalDeclaration();
        if (jj_2_1(1)) {
          ;
        } else {
          break label_1;
        }
      }
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
      {if (true) return jjtn000;}
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
```

```
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
  throw new Error("Missing return statement in function");
}

final public void externalDeclaration() throws ParseException {
                            /*@bgen(jjtree) externalDeclaration */
ASTexternalDeclaration jjtn000 = new ASTexternalDeclaration(JJTEXTERNALDECLARATION);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    if (jj_2_2(2147483647)) {
      functionDefinition();
    } else if (jj_2_3(1)) {
      declaration();
    } else {
      jj_consume_token(-1);
      throw new ParseException();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void functionDefinition() throws ParseException {
                           /*@bgen(jjtree) functionDefinition */
ASTfunctionDefinition jjtn000 = new ASTfunctionDefinition(JJTFUNCTIONDEFINITION);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    if (jj_2_4(2147483647)) {
      declarationSpecifiers();
    } else {
      ;
    }
    declarator();
    if (jj_2_5(1)) {
      declarationList();
    } else {
      ;
    }
    compoundStatement();
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
```

```
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void functionDefinitionLookahead() throws ParseException {
                                      /*@bgen(jjtree) functionDefinitionLookahead */
  ASTfunctionDefinitionLookahead jjtn000 = new
ASTfunctionDefinitionLookahead(JJTFUNCTIONDEFINITIONLOOKAHEAD);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      if (jj_2_6(2147483647)) {
        declarationSpecifiers();
      } else {
        ;
      }
      declarator();
      if (jj_2_7(1)) {
        declarationList();
      } else {
        ;
      }
      jj_consume_token(LEFT_BRACE);
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void declarationList() throws ParseException {
                           /*@bgen(jjtree) declarationList */
  ASTdeclarationList jjtn000 = new ASTdeclarationList(JJTDECLARATIONLIST);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      label_2:
      while (true) {
        declaration();
        if (jj_2_8(2147483647)) {
          ;
        } else {
          break label_2;
        }
```

```
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void declaration() throws ParseException {
                      /*@bgen(jjtree) declaration */
  ASTdeclaration jjtn000 = new ASTdeclaration(JJTDECLARATION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      declarationSpecifiers();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LEFT_PAREN:
      case STAR:
      case IDENTIFIER:
        initDeclaratorList();
        break;
      default:
        jj_la1[0] = jj_gen;
        ;
      }
      jj_consume_token(SEMICOLON);
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void declarationSpecifiers() throws ParseException {
                                /*@bgen(jjtree) declarationSpecifiers */
  ASTdeclarationSpecifiers jjtn000 = new
ASTdeclarationSpecifiers(JJTDECLARATIONSPECIFIERS);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case AUTO:
      case REGISTER:
```

```
    case EXTERN:
    case STATIC:
    case TYPEDEF:
      storageClassSpecifier();
      if (jj_2_9(2147483647)) {
        declarationSpecifiers();
      } else {
        ;
      }
      break;
    default:
      jj_la1[1] = jj_gen;
      if (jj_2_12(1)) {
        typeSpecifier();
        if (jj_2_10(2147483647)) {
          declarationSpecifiers();
        } else {
          ;
        }
      } else {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case CONST:
        case VOLATILE:
          typeQualifier();
          if (jj_2_11(2147483647)) {
            declarationSpecifiers();
          } else {
            ;
          }
          break;
        default:
          jj_la1[2] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
      }
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void initDeclaratorList() throws ParseException {
                           /*@bgen(jjtree) initDeclaratorList */
ASTinitDeclaratorList jjtn000 = new ASTinitDeclaratorList(JJTINITDECLARATORLIST);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    initDeclarator();
    label_3:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case COMMA:
        ;
        break;
```

```
        default:
          jj_la1[3] = jj_gen;
          break label_3;
        }
        jj_consume_token(COMMA);
        initDeclarator();
      }
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
        // Finished with a typedefDeclaration??
        if ( !(typedefParsingStack.empty()) &&
((Boolean)typedefParsingStack.peek()).booleanValue() ) {
            typedefParsingStack.pop();
        }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void initDeclarator() throws ParseException {
                          /*@bgen(jjtree) initDeclarator */
  ASTinitDeclarator jjtn000 = new ASTinitDeclarator(JJTINITDECLARATOR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      declarator();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case EQUALS:
        jj_consume_token(EQUALS);
        initializer();
        break;
      default:
        jj_la1[4] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }
```

```
final public void initializer() throws ParseException {
                    /*@bgen(jjtree) initializer */
ASTinitializer jjtn000 = new ASTinitializer(JJTINITIALIZER);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case INTEGER:
    case REAL:
    case CHARACTER:
    case STRING:
    case SIZEOF:
    case PLUS_PLUS:
    case MINUS_MINUS:
    case LEFT_PAREN:
    case AMPERSAND:
    case STAR:
    case PLUS:
    case MINUS:
    case BITWISE_NOT:
    case LOGICAL_NOT:
    case IDENTIFIER:
      assignmentExpression();
      break;
    case LEFT_BRACE:
      jj_consume_token(LEFT_BRACE);
      initializerList();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case COMMA:
        jj_consume_token(COMMA);
        break;
      default:
        jj_la1[5] = jj_gen;
        ;
      }
      jj_consume_token(RIGHT_BRACE);
      break;
    default:
      jj_la1[6] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void initializerList() throws ParseException {
                        /*@bgen(jjtree) initializerList */
ASTinitializerList jjtn000 = new ASTinitializerList(JJTINITIALIZERLIST);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    initializer();
```

```
    label_4:
    while (true) {
      if (jj_2_13(2)) {
        ;
      } else {
        break label_4;
      }
      jj_consume_token(COMMA);
      initializer();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void typeName() throws ParseException {
                 /*@bgen(jjtree) typeName */
ASTtypeName jjtn000 = new ASTtypeName(JJTTYPENAME);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    specifierQualifierList();
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case LEFT_PAREN:
    case LEFT_BRACKET:
    case STAR:
      abstractDeclarator();
      break;
    default:
      jj_la1[7] = jj_gen;
      ;
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void storageClassSpecifier() throws ParseException {
                              /*@bgen(jjtree) storageClassSpecifier */
```

```
  ASTstorageClassSpecifier jjtn000 = new
ASTstorageClassSpecifier(JJTSTORAGECLASSSPECIFIER);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case AUTO:
        jj_consume_token(AUTO);
        break;
      case REGISTER:
        jj_consume_token(REGISTER);
        break;
      case STATIC:
        jj_consume_token(STATIC);
        break;
      case EXTERN:
        jj_consume_token(EXTERN);
        break;
      case TYPEDEF:
        jj_consume_token(TYPEDEF);
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
      typedefParsingStack.push(Boolean.TRUE);
        break;
      default:
        jj_la1[8] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void typeSpecifier() throws ParseException {
                        /*@bgen(jjtree) typeSpecifier */
  ASTtypeSpecifier jjtn000 = new ASTtypeSpecifier(JJTTYPESPECIFIER);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case VOID:
        jj_consume_token(VOID);
        break;
      case CHAR:
        jj_consume_token(CHAR);
        break;
      case SHORT:
        jj_consume_token(SHORT);
        break;
      case INT:
        jj_consume_token(INT);
        break;
      case LONG:
        jj_consume_token(LONG);
        break;
      case FLOAT:
        jj_consume_token(FLOAT);
        break;
      case DOUBLE:
        jj_consume_token(DOUBLE);
        break;
      case SIGNED:
        jj_consume_token(SIGNED);
        break;
      case UNSIGNED:
        jj_consume_token(UNSIGNED);
        break;
      case STRUCT:
```

```
    case UNION:
      structOrUnionSpecifier();
      break;
    case ENUM:
      enumSpecifier();
      break;
    default:
      jj_la1[9] = jj_gen;
      if (isType(getToken(1).image)) {
        typedefName();
      } else {
        jj_consume_token(-1);
        throw new ParseException();
      }
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void typedefName() throws ParseException {
                    /*@bgen(jjtree) typedefName */
ASTtypedefName jjtn000 = new ASTtypedefName(JJTTYPEDEFNAME);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    identifier(false);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void typeQualifier() throws ParseException {
                      /*@bgen(jjtree) typeQualifier */
ASTtypeQualifier jjtn000 = new ASTtypeQualifier(JJTTYPEQUALIFIER);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
```

```
      case CONST:
        jj_consume_token(CONST);
        break;
      case VOLATILE:
        jj_consume_token(VOLATILE);
        break;
      default:
        jj_la1[10] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void structOrUnionSpecifier() throws ParseException {
                                  /*@bgen(jjtree) structOrUnionSpecifier */
  ASTstructOrUnionSpecifier jjtn000 = new
ASTstructOrUnionSpecifier(JJTSTRUCTORUNIONSPECIFIER);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      typedefParsingStack.push(Boolean.FALSE);
      structOrUnion();
      if (jj_2_14(3)) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case IDENTIFIER:
          identifier(false);
          break;
        default:
          jj_la1[11] = jj_gen;
          ;
        }
        jj_consume_token(LEFT_BRACE);
        structDeclarationList();
        jj_consume_token(RIGHT_BRACE);
      } else {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case IDENTIFIER:
          identifier(false);
          break;
        default:
          jj_la1[12] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
      }
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
      typedefParsingStack.pop();
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
```

```
    }
  }

  final public void structOrUnion() throws ParseException {
                        /*@bgen(jjtree) structOrUnion */
  ASTstructOrUnion jjtn000 = new ASTstructOrUnion(JJTSTRUCTORUNION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case STRUCT:
        jj_consume_token(STRUCT);
        break;
      case UNION:
        jj_consume_token(UNION);
        break;
      default:
        jj_la1[13] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void structDeclarationList() throws ParseException {
                                /*@bgen(jjtree) structDeclarationList */
  ASTstructDeclarationList jjtn000 = new
ASTstructDeclarationList(JJTSTRUCTDECLARATIONLIST);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      label_5:
      while (true) {
        structDeclaration();
        if (jj_2_15(1)) {
          ;
        } else {
          break label_5;
        }
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void structDeclaration() throws ParseException {
                            /*@bgen(jjtree) structDeclaration */
  ASTstructDeclaration jjtn000 = new ASTstructDeclaration(JJTSTRUCTDECLARATION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
```

```
    specifierQualifierList();
    structDeclaratorList();
    jj_consume_token(SEMICOLON);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void specifierQualifierList() throws ParseException {
  if (jj_2_18(1)) {
    typeSpecifier();
    if (jj_2_16(2147483647)) {
      specifierQualifierList();
    } else {
      ;
    }
  } else {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case CONST:
    case VOLATILE:
      typeQualifier();
      if (jj_2_17(2147483647)) {
        specifierQualifierList();
      } else {
        ;
      }
      break;
    default:
      jj_la1[14] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
  }
}

final public void structDeclaratorList() throws ParseException {
                              /*@bgen(jjtree) structDeclaratorList */
ASTstructDeclaratorList jjtn000 = new ASTstructDeclaratorList(JJTSTRUCTDECLARATORLIST);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    structDeclarator();
    label_6:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case COMMA:
        ;
        break;
      default:
        jj_la1[15] = jj_gen;
        break label_6;
      }
      jj_consume_token(COMMA);
      structDeclarator();
    }
```

```
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void structDeclarator() throws ParseException {
                          /*@bgen(jjtree) structDeclarator */
ASTstructDeclarator jjtn000 = new ASTstructDeclarator(JJTSTRUCTDECLARATOR);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    if (jj_2_19(3)) {
      declarator();
    } else if (jj_2_20(1)) {
      typeSpecifier();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LEFT_PAREN:
      case STAR:
      case IDENTIFIER:
        declarator();
        break;
      default:
        jj_la1[16] = jj_gen;
        ;
      }
      jj_consume_token(COLON);
      constantExpression();
    } else {
      jj_consume_token(-1);
      throw new ParseException();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void enumSpecifier() throws ParseException {
                       /*@bgen(jjtree) enumSpecifier */
ASTenumSpecifier jjtn000 = new ASTenumSpecifier(JJTENUMSPECIFIER);
```

```
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(ENUM);
    if (jj_2_21(3)) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case IDENTIFIER:
        identifier(false);
        break;
      default:
        jj_la1[17] = jj_gen;
        ;
      }
      jj_consume_token(LEFT_BRACE);
      enumeratorList();
      jj_consume_token(RIGHT_BRACE);
    } else {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case IDENTIFIER:
        identifier(false);
        break;
      default:
        jj_la1[18] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void enumeratorList() throws ParseException {
                        /*@bgen(jjtree) enumeratorList */
ASTenumeratorList jjtn000 = new ASTenumeratorList(JJTENUMERATORLIST);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    enumerator();
    label_7:
    while (true) {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case COMMA:
        ;
        break;
      default:
        jj_la1[19] = jj_gen;
        break label_7;
      }
      jj_consume_token(COMMA);
      enumerator();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
```

```
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void enumerator() throws ParseException {
                  /*@bgen(jjtree) enumerator */
ASTenumerator jjtn000 = new ASTenumerator(JJTENUMERATOR);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    identifier(false);
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case EQUALS:
      jj_consume_token(EQUALS);
      constantExpression();
      break;
    default:
      jj_la1[20] = jj_gen;
      ;
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void declarator() throws ParseException {
                  /*@bgen(jjtree) declarator */
ASTdeclarator jjtn000 = new ASTdeclarator(JJTDECLARATOR);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case STAR:
      pointer();
      break;
    default:
      jj_la1[21] = jj_gen;
      ;
    }
    directDeclarator();
  } catch (Throwable jjte000) {
```

```
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void pointer() throws ParseException {
                /*@bgen(jjtree) pointer */
ASTpointer jjtn000 = new ASTpointer(JJTPOINTER);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(STAR);
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case CONST:
    case VOLATILE:
      typeQualifierList();
      break;
    default:
      jj_la1[22] = jj_gen;
      ;
    }
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case STAR:
      pointer();
      break;
    default:
      jj_la1[23] = jj_gen;
      ;
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void typeQualifierList() throws ParseException {
  label_8:
  while (true) {
    typeQualifier();
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case CONST:
```

```
      case VOLATILE:
        ;
        break;
      default:
        jj_la1[24] = jj_gen;
        break label_8;
      }
    }
  }

  final public void directDeclarator() throws ParseException {
 /*@bgen(jjtree) directDeclarator */
    ASTdirectDeclarator jjtn000 = new ASTdirectDeclarator(JJTDIRECTDECLARATOR);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);Token t;
    boolean isFunction = false;
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case IDENTIFIER:
        t = identifier(false);
        if ( !(typedefParsingStack.empty()) &&
((Boolean)typedefParsingStack.peek()).booleanValue() ) {
            addType(t.image);
        }
        break;
      case LEFT_PAREN:
        jj_consume_token(LEFT_PAREN);
        declarator();
        jj_consume_token(RIGHT_PAREN);
        break;
      default:
        jj_la1[25] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
        typedefParsingStack.push(Boolean.FALSE);
      label_9:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case LEFT_PAREN:
        case LEFT_BRACKET:
          ;
          break;
        default:
          jj_la1[26] = jj_gen;
          break label_9;
        }
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case LEFT_BRACKET:
          jj_consume_token(LEFT_BRACKET);
          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
          case INTEGER:
          case REAL:
          case CHARACTER:
          case STRING:
          case SIZEOF:
          case PLUS_PLUS:
          case MINUS_MINUS:
          case LEFT_PAREN:
          case AMPERSAND:
          case STAR:
          case PLUS:
          case MINUS:
          case BITWISE_NOT:
          case LOGICAL_NOT:
          case IDENTIFIER:
            constantExpression();
            break;
          default:
            jj_la1[27] = jj_gen;
            ;
```

```
        }
        jj_consume_token(RIGHT_BRACKET);
        break;
      default:
        jj_la1[29] = jj_gen;
        if (jj_2_22(3)) {
          jj_consume_token(LEFT_PAREN);
                   isFunction = true;
          parameterTypeList();
          jj_consume_token(RIGHT_PAREN);
        } else {
          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
          case LEFT_PAREN:
            jj_consume_token(LEFT_PAREN);
                   isFunction = true;
            switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
            case IDENTIFIER:
              identifierList();
              break;
            default:
              jj_la1[28] = jj_gen;
              ;
            }
            jj_consume_token(RIGHT_PAREN);
            break;
          default:
            jj_la1[30] = jj_gen;
            jj_consume_token(-1);
            throw new ParseException();
          }
        }
      }
    }
      typedefParsingStack.pop();
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
      jjtn000.setAttribute(NodeKey.FUNCTION_DEC_DEF, isFunction);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void parameterTypeList() throws ParseException {
                          /*@bgen(jjtree) parameterTypeList */
ASTparameterTypeList jjtn000 = new ASTparameterTypeList(JJTPARAMETERTYPELIST);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    parameterList();
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case COMMA:
      jj_consume_token(COMMA);
      jj_consume_token(THREE_DOTS);
      break;
    default:
```

```
      jj_la1[31] = jj_gen;
      ;
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void parameterList() throws ParseException {
                      /*@bgen(jjtree) parameterList */
ASTparameterList jjtn000 = new ASTparameterList(JJTPARAMETERLIST);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    parameterDeclaration();
    label_10:
    while (true) {
      if (jj_2_23(2)) {
        ;
      } else {
        break label_10;
      }
      jj_consume_token(COMMA);
      parameterDeclaration();
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void parameterDeclaration() throws ParseException {
                             /*@bgen(jjtree) parameterDeclaration */
ASTparameterDeclaration jjtn000 = new ASTparameterDeclaration(JJTPARAMETERDECLARATION);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    declarationSpecifiers();
    if (jj_2_24(2147483647)) {
      declarator();
```

```
    } else {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LEFT_PAREN:
      case LEFT_BRACKET:
      case STAR:
        abstractDeclarator();
        break;
      default:
        jj_la1[32] = jj_gen;
        ;
      }
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void abstractDeclarator() throws ParseException {
                            /*@bgen(jjtree) abstractDeclarator */
ASTabstractDeclarator jjtn000 = new ASTabstractDeclarator(JJTABSTRACTDECLARATOR);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    if (jj_2_25(3)) {
      pointer();
    } else {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LEFT_PAREN:
      case LEFT_BRACKET:
      case STAR:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case STAR:
          pointer();
          break;
        default:
          jj_la1[33] = jj_gen;
          ;
        }
        directAbstractDeclarator();
        break;
      default:
        jj_la1[34] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
```

```
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void directAbstractDeclarator() throws ParseException {
                                    /*@bgen(jjtree) directAbstractDeclarator */
  ASTdirectAbstractDeclarator jjtn000 = new
ASTdirectAbstractDeclarator(JJTDIRECTABSTRACTDECLARATOR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      if (jj_2_27(2)) {
        jj_consume_token(LEFT_PAREN);
        abstractDeclarator();
        jj_consume_token(RIGHT_PAREN);
      } else {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case LEFT_BRACKET:
          jj_consume_token(LEFT_BRACKET);
          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
          case INTEGER:
          case REAL:
          case CHARACTER:
          case STRING:
          case SIZEOF:
          case PLUS_PLUS:
          case MINUS_MINUS:
          case LEFT_PAREN:
          case AMPERSAND:
          case STAR:
          case PLUS:
          case MINUS:
          case BITWISE_NOT:
          case LOGICAL_NOT:
          case IDENTIFIER:
            constantExpression();
            break;
          default:
            jj_la1[35] = jj_gen;
            ;
          }
          jj_consume_token(RIGHT_BRACKET);
          break;
        case LEFT_PAREN:
          jj_consume_token(LEFT_PAREN);
          if (jj_2_26(1)) {
            parameterTypeList();
          } else {
            ;
          }
          jj_consume_token(RIGHT_PAREN);
          break;
        default:
          jj_la1[36] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
      }
      label_11:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case LEFT_PAREN:
        case LEFT_BRACKET:
```

```
        ;
        break;
      default:
        jj_la1[37] = jj_gen;
        break label_11;
      }
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LEFT_BRACKET:
        jj_consume_token(LEFT_BRACKET);
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case INTEGER:
        case REAL:
        case CHARACTER:
        case STRING:
        case SIZEOF:
        case PLUS_PLUS:
        case MINUS_MINUS:
        case LEFT_PAREN:
        case AMPERSAND:
        case STAR:
        case PLUS:
        case MINUS:
        case BITWISE_NOT:
        case LOGICAL_NOT:
        case IDENTIFIER:
          constantExpression();
          break;
        default:
          jj_la1[38] = jj_gen;
          ;
        }
        jj_consume_token(RIGHT_BRACKET);
        break;
      case LEFT_PAREN:
        jj_consume_token(LEFT_PAREN);
        if (jj_2_28(1)) {
          parameterTypeList();
        } else {
          ;
        }
        jj_consume_token(RIGHT_PAREN);
        break;
      default:
        jj_la1[39] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void identifierList() throws ParseException {
  identifier(false);
```

```
  label_12:
  while (true) {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case COMMA:
      ;
      break;
    default:
      jj_la1[40] = jj_gen;
      break label_12;
    }
    jj_consume_token(COMMA);
    identifier(false);
  }
}

final public void statementList() throws ParseException {
  label_13:
  while (true) {
    statement();
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case INTEGER:
    case REAL:
    case CHARACTER:
    case STRING:
    case IF:
    case SWITCH:
    case CASE:
    case FOR:
    case DO:
    case WHILE:
    case BREAK:
    case CONTINUE:
    case CDEFAULT:
    case GOTO:
    case SIZEOF:
    case RETURN:
    case PLUS_PLUS:
    case MINUS_MINUS:
    case LEFT_PAREN:
    case LEFT_BRACE:
    case AMPERSAND:
    case STAR:
    case PLUS:
    case MINUS:
    case BITWISE_NOT:
    case LOGICAL_NOT:
    case SEMICOLON:
    case IDENTIFIER:
      ;
      break;
    default:
      jj_la1[41] = jj_gen;
      break label_13;
    }
  }
}

final public void statement() throws ParseException {
  switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
  case LEFT_BRACE:
    compoundStatement();
    break;
  default:
    jj_la1[42] = jj_gen;
    if (jj_2_29(2)) {
      labeledStatement();
    } else {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case BREAK:
      case CONTINUE:
      case GOTO:
```

```
      case RETURN:
        jumpStatement();
        break;
      case INTEGER:
      case REAL:
      case CHARACTER:
      case STRING:
      case SIZEOF:
      case PLUS_PLUS:
      case MINUS_MINUS:
      case LEFT_PAREN:
      case AMPERSAND:
      case STAR:
      case PLUS:
      case MINUS:
      case BITWISE_NOT:
      case LOGICAL_NOT:
      case SEMICOLON:
      case IDENTIFIER:
        expressionStatement();
        break;
      case IF:
        ifStatement();
        break;
      case SWITCH:
        switchStatement();
        break;
      case FOR:
        forStatement();
        break;
      case WHILE:
        whileStatement();
        break;
      case DO:
        doWhileStatement();
        break;
      default:
        jj_la1[43] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
    }
  }
}

final public void compoundStatement() throws ParseException {
                          /*@bgen(jjtree) compoundStatement */
ASTcompoundStatement jjtn000 = new ASTcompoundStatement(JJTCOMPOUNDSTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(LEFT_BRACE);
    if (jj_2_30(2147483647)) {
      declarationList();
    } else {
      ;
    }
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case INTEGER:
    case REAL:
    case CHARACTER:
    case STRING:
    case IF:
    case SWITCH:
    case CASE:
    case FOR:
    case DO:
    case WHILE:
    case BREAK:
    case CONTINUE:
    case CDEFAULT:
```

```
    case GOTO:
    case SIZEOF:
    case RETURN:
    case PLUS_PLUS:
    case MINUS_MINUS:
    case LEFT_PAREN:
    case LEFT_BRACE:
    case AMPERSAND:
    case STAR:
    case PLUS:
    case MINUS:
    case BITWISE_NOT:
    case LOGICAL_NOT:
    case SEMICOLON:
    case IDENTIFIER:
      statementList();
      break;
    default:
      jj_la1[44] = jj_gen;
      ;
    }
    jj_consume_token(RIGHT_BRACE);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void labeledStatement() throws ParseException {
                         /*@bgen(jjtree) labeledStatement */
ASTlabeledStatement jjtn000 = new ASTlabeledStatement(JJTLABELEDSTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case CASE:
      jj_consume_token(CASE);
      constantExpression();
      jj_consume_token(COLON);
      statement();
      break;
    case CDEFAULT:
      jj_consume_token(CDEFAULT);
      jj_consume_token(COLON);
      statement();
      break;
    case IDENTIFIER:
      identifier(false);
      jj_consume_token(COLON);
      statement();
      break;
    default:
      jj_la1[45] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
```

```
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void jumpStatement() throws ParseException {
                      /*@bgen(jjtree) jumpStatement */
ASTjumpStatement jjtn000 = new ASTjumpStatement(JJTJUMPSTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case GOTO:
      jj_consume_token(GOTO);
      identifier(false);
      jj_consume_token(SEMICOLON);
      break;
    case BREAK:
      jj_consume_token(BREAK);
      jj_consume_token(SEMICOLON);
      break;
    case CONTINUE:
      jj_consume_token(CONTINUE);
      jj_consume_token(SEMICOLON);
      break;
    case RETURN:
      jj_consume_token(RETURN);
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case INTEGER:
      case REAL:
      case CHARACTER:
      case STRING:
      case SIZEOF:
      case PLUS_PLUS:
      case MINUS_MINUS:
      case LEFT_PAREN:
      case AMPERSAND:
      case STAR:
      case PLUS:
      case MINUS:
      case BITWISE_NOT:
      case LOGICAL_NOT:
      case IDENTIFIER:
        expression(NodeType.JUMP_STMT_NODE);
        break;
      default:
        jj_la1[46] = jj_gen;
        ;
      }
      jj_consume_token(SEMICOLON);
      break;
    default:
      jj_la1[47] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
```

```
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void expressionStatement() throws ParseException {
                            /*@bgen(jjtree) expressionStatement */
ASTexpressionStatement jjtn000 = new ASTexpressionStatement(JJTEXPRESSIONSTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case INTEGER:
    case REAL:
    case CHARACTER:
    case STRING:
    case SIZEOF:
    case PLUS_PLUS:
    case MINUS_MINUS:
    case LEFT_PAREN:
    case AMPERSAND:
    case STAR:
    case PLUS:
    case MINUS:
    case BITWISE_NOT:
    case LOGICAL_NOT:
    case IDENTIFIER:
      expression(NodeType.EXPRESSION_STMT_NODE);
      break;
    default:
      jj_la1[48] = jj_gen;
      ;
    }
    jj_consume_token(SEMICOLON);
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}
```

```
final public void ifStatement() throws ParseException {
                    /*@bgen(jjtree) ifStatement */
ASTifStatement jjtn000 = new ASTifStatement(JJTIFSTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(IF);
    jj_consume_token(LEFT_PAREN);
    expression(NodeType.IF_STMT_NODE);
    jj_consume_token(RIGHT_PAREN);
    statement();
    if (jj_2_31(2)) {
      jj_consume_token(ELSE);
      statement();
    } else {
      ;
    }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void switchStatement() throws ParseException {
                        /*@bgen(jjtree) switchStatement */
ASTswitchStatement jjtn000 = new ASTswitchStatement(JJTSWITCHSTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(SWITCH);
    jj_consume_token(LEFT_PAREN);
    expression(NodeType.SWITCH_STMT_NODE);
    jj_consume_token(RIGHT_PAREN);
    statement();
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}
```

```
 final public void forStatement() throws ParseException {
/*@bgen(jjtree) forStatement */
   ASTforStatement jjtn000 = new ASTforStatement(JJTFORSTATEMENT);
   boolean jjtc000 = true;
   jjtree.openNodeScope(jjtn000);Token t;
   int beginLine = 0;
   int beginColumn = 0;
   try {
     t = jj_consume_token(FOR);
       beginLine = token.beginLine;
       beginColumn = token.beginColumn;
     jj_consume_token(LEFT_PAREN);
     if (jj_2_33(1)) {
       if (jj_2_32(1)) {
         typeSpecifier();
       } else {
         ;
       }
       expression(NodeType.FOR_STMT_NODE);
     } else {
       ;
     }
     jj_consume_token(SEMICOLON);
     switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
     case INTEGER:
     case REAL:
     case CHARACTER:
     case STRING:
     case SIZEOF:
     case PLUS_PLUS:
     case MINUS_MINUS:
     case LEFT_PAREN:
     case AMPERSAND:
     case STAR:
     case PLUS:
     case MINUS:
     case BITWISE_NOT:
     case LOGICAL_NOT:
     case IDENTIFIER:
       expression(NodeType.FOR_STMT_NODE);
       break;
     default:
       jj_la1[49] = jj_gen;
       ;
     }
     jj_consume_token(SEMICOLON);
     switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
     case INTEGER:
     case REAL:
     case CHARACTER:
     case STRING:
     case SIZEOF:
     case PLUS_PLUS:
     case MINUS_MINUS:
     case LEFT_PAREN:
     case AMPERSAND:
     case STAR:
     case PLUS:
     case MINUS:
     case BITWISE_NOT:
     case LOGICAL_NOT:
     case IDENTIFIER:
       expression(NodeType.FOR_STMT_NODE);
       break;
     default:
       jj_la1[50] = jj_gen;
       ;
     }
     jj_consume_token(RIGHT_PAREN);
     statement();
     jjtree.closeNodeScope(jjtn000, true);
```

```
    jjtc000 = false;
      jjtn000.setAttribute(NodeKey.BEGIN_LINE, new Integer(beginLine));
      jjtn000.setAttribute(NodeKey.BEGIN_COLUMN, new Integer(beginColumn));
      if ( t.specialToken != null ) {
          jjtn000.setAttribute(NodeKey.SPECIAL_TOKEN, t.specialToken.image);
      }
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void whileStatement() throws ParseException {
                      /*@bgen(jjtree) whileStatement */
ASTwhileStatement jjtn000 = new ASTwhileStatement(JJTWHILESTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(WHILE);
    jj_consume_token(LEFT_PAREN);
    expression(NodeType.WHILE_STMT_NODE);
    jj_consume_token(RIGHT_PAREN);
    statement();
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      {if (true) throw (RuntimeException)jjte000;}
    }
    if (jjte000 instanceof ParseException) {
      {if (true) throw (ParseException)jjte000;}
    }
    {if (true) throw (Error)jjte000;}
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
}

final public void doWhileStatement() throws ParseException {
                        /*@bgen(jjtree) doWhileStatement */
ASTdoWhileStatement jjtn000 = new ASTdoWhileStatement(JJTDOWHILESTATEMENT);
boolean jjtc000 = true;
jjtree.openNodeScope(jjtn000);
  try {
    jj_consume_token(DO);
    statement();
    jj_consume_token(WHILE);
    jj_consume_token(LEFT_PAREN);
    expression(NodeType.DO_WHILE_STMT_NODE);
    jj_consume_token(RIGHT_PAREN);
```

```
      jj_consume_token(SEMICOLON);
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void expression(NodeType type) throws ParseException {
                                    /*@bgen(jjtree) expression */
  ASTexpression jjtn000 = new ASTexpression(JJTEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      assignmentExpression();
      label_14:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case COMMA:
          ;
          break;
        default:
          jj_la1[51] = jj_gen;
          break label_14;
        }
        jj_consume_token(COMMA);
        assignmentExpression();
      }
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
        jjtn000.setAttribute(NodeKey.CALLED_FROM, type);
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void assignmentExpression() throws ParseException {
 /*@bgen(jjtree) #assignmentExpression(> 1) */
    ASTassignmentExpression jjtn000 = new
ASTassignmentExpression(JJTASSIGNMENTEXPRESSION);
    boolean jjtc000 = true;
```

```
jjtree.openNodeScope(jjtn000);boolean reduction = false;
String reductionOp = null;
try {
  if (jj_2_34(2147483647)) {
    unaryExpression(true);
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case EQUALS:
      jj_consume_token(EQUALS);
      break;
    case STAR_EQUALS:
      jj_consume_token(STAR_EQUALS);
                              reduction = true; reductionOp = "*";
      break;
    case SLASH_EQUALS:
      jj_consume_token(SLASH_EQUALS);
                               reduction = true; reductionOp = "/";
      break;
    case PERCENT_EQUALS:
      jj_consume_token(PERCENT_EQUALS);
      break;
    case PLUS_EQUALS:
      jj_consume_token(PLUS_EQUALS);
                              reduction = true; reductionOp = "+";
      break;
    case MINUS_EQUALS:
      jj_consume_token(MINUS_EQUALS);
                               reduction = true; reductionOp = "-";
      break;
    case LESS_LESS_EQUALS:
      jj_consume_token(LESS_LESS_EQUALS);
      break;
    case GREATER_GREATER_EQUALS:
      jj_consume_token(GREATER_GREATER_EQUALS);
      break;
    case BITWISE_OR_EQUALS:
      jj_consume_token(BITWISE_OR_EQUALS);
                                    reduction = true; reductionOp = "^";
      break;
    case AMP_EQUALS:
      jj_consume_token(AMP_EQUALS);
                             reduction = true; reductionOp = "&";
      break;
    case BITWISE_XOR_EQUALS:
      jj_consume_token(BITWISE_XOR_EQUALS);
                                     reduction = true; reductionOp = "|";
      break;
    default:
      jj_la1[52] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
    assignmentExpression();
  } else if (jj_2_35(3)) {
    conditionalExpression();
  } else {
    jj_consume_token(-1);
    throw new ParseException();
  }
  jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
  jjtc000 = false;
    jjtn000.setAttribute(NodeKey.REDUCTION, reduction);
    jjtn000.setAttribute(NodeKey.REDUCTION_OP, reductionOp);
} catch (Throwable jjte000) {
  if (jjtc000) {
    jjtree.clearNodeScope(jjtn000);
    jjtc000 = false;
  } else {
    jjtree.popNode();
  }
  if (jjte000 instanceof RuntimeException) {
    {if (true) throw (RuntimeException)jjte000;}
```

```
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void constantExpression() throws ParseException {
    conditionalExpression();
  }

  final public void conditionalExpression() throws ParseException {
                                                                  /*@bgen(jjtree)
#conditionalExpression(> 2) */
  ASTconditionalExpression jjtn000 = new
ASTconditionalExpression(JJTCONDITIONALEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      logicalORExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case QUESTION_MARK:
        jj_consume_token(QUESTION_MARK);
        expression(NodeType.CONDITIONAL_EXPRESSION_NODE);
        jj_consume_token(COLON);
        conditionalExpression();
        break;
      default:
        jj_la1[53] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 2);
      }
    }
  }

  final public void logicalORExpression() throws ParseException {
                                                        /*@bgen(jjtree)
#logicalORExpression(> 1) */
  ASTlogicalORExpression jjtn000 = new ASTlogicalORExpression(JJTLOGICALOREXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      logicalANDExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LOGICAL_OR:
        jj_consume_token(LOGICAL_OR);
        logicalORExpression();
        break;
      default:
```

```
        jj_la1[54] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void logicalANDExpression() throws ParseException {
                                                          /*@bgen(jjtree)
#logicalANDExpression(> 1) */
  ASTlogicalANDExpression jjtn000 = new ASTlogicalANDExpression(JJTLOGICALANDEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      bitwiseORExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LOGICAL_AND:
        jj_consume_token(LOGICAL_AND);
        logicalANDExpression();
        break;
      default:
        jj_la1[55] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void bitwiseORExpression() throws ParseException {
                                                        /*@bgen(jjtree)
#bitwiseORExpression(> 1) */
  ASTbitwiseORExpression jjtn000 = new ASTbitwiseORExpression(JJTBITWISEOREXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      bitwiseXORExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
```

```
      case BITWISE_OR:
        jj_consume_token(BITWISE_OR);
        bitwiseORExpression();
        break;
      default:
        jj_la1[56] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void bitwiseXORExpression() throws ParseException {
                                                                  /*@bgen(jjtree)
#bitwiseXORExpression(> 1) */
  ASTbitwiseXORExpression jjtn000 = new ASTbitwiseXORExpression(JJTBITWISEXOREXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      bitwiseANDExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case BITWISE_XOR:
        jj_consume_token(BITWISE_XOR);
        bitwiseXORExpression();
        break;
      default:
        jj_la1[57] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void bitwiseANDExpression() throws ParseException {
                                                                  /*@bgen(jjtree)
#bitwiseANDExpression(> 1) */
  ASTbitwiseANDExpression jjtn000 = new ASTbitwiseANDExpression(JJTBITWISEANDEXPRESSION);
```

```
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      equalityExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case AMPERSAND:
        jj_consume_token(AMPERSAND);
        bitwiseANDExpression();
        break;
      default:
        jj_la1[58] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void equalityExpression() throws ParseException {
                                                        /*@bgen(jjtree)
#equalityExpression(> 1) */
  ASTequalityExpression jjtn000 = new ASTequalityExpression(JJTEQUALITYEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      relationalExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case EQUALS_EQUALS:
      case NOT_EQUALS:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case EQUALS_EQUALS:
          jj_consume_token(EQUALS_EQUALS);
          break;
        case NOT_EQUALS:
          jj_consume_token(NOT_EQUALS);
          break;
        default:
          jj_la1[59] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
        equalityExpression();
        break;
      default:
        jj_la1[60] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
```

```
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void relationalExpression() throws ParseException {
                                                                  /*@bgen(jjtree)
#relationalExpression(> 1) */
  ASTrelationalExpression jjtn000 = new ASTrelationalExpression(JJTRELATIONALEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      shiftExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case LESS_THAN:
      case GREATER_THAN:
      case LESS_EQUALS:
      case GREATER_EQUALS:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case LESS_THAN:
          jj_consume_token(LESS_THAN);
          break;
        case LESS_EQUALS:
          jj_consume_token(LESS_EQUALS);
          break;
        case GREATER_THAN:
          jj_consume_token(GREATER_THAN);
          break;
        case GREATER_EQUALS:
          jj_consume_token(GREATER_EQUALS);
          break;
        default:
          jj_la1[61] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
        relationalExpression();
        break;
      default:
        jj_la1[62] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }
```

```
  final public void shiftExpression() throws ParseException {
                                                /*@bgen(jjtree) #shiftExpression(> 1) */
  ASTshiftExpression jjtn000 = new ASTshiftExpression(JJTSHIFTEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      additiveExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case SHIFT_LEFT:
      case SHIFT_RIGHT:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case SHIFT_LEFT:
          jj_consume_token(SHIFT_LEFT);
          break;
        case SHIFT_RIGHT:
          jj_consume_token(SHIFT_RIGHT);
          break;
        default:
          jj_la1[63] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
        shiftExpression();
        break;
      default:
        jj_la1[64] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void additiveExpression() throws ParseException {
                                                        /*@bgen(jjtree)
#additiveExpression(> 1) */
  ASTadditiveExpression jjtn000 = new ASTadditiveExpression(JJTADDITIVEEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      multiplicativeExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case PLUS:
      case MINUS:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case PLUS:
          jj_consume_token(PLUS);
               jjtn000.setAttribute(NodeKey.OPERATOR, "+");
          break;
        case MINUS:
          jj_consume_token(MINUS);
                jjtn000.setAttribute(NodeKey.OPERATOR, "-");
          break;
        default:
          jj_la1[65] = jj_gen;
```

```
          jj_consume_token(-1);
          throw new ParseException();
        }
        additiveExpression();
        break;
      default:
        jj_la1[66] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void multiplicativeExpression() throws ParseException {
                                                                    /*@bgen(jjtree)
#multiplicativeExpression(> 1) */
  ASTmultiplicativeExpression jjtn000 = new
ASTmultiplicativeExpression(JJTMULTIPLICATIVEEXPRESSION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
    try {
      castExpression();
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case STAR:
      case SLASH:
      case PERCENT:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case STAR:
          jj_consume_token(STAR);
               jjtn000.setAttribute(NodeKey.OPERATOR, "*");
          break;
        case SLASH:
          jj_consume_token(SLASH);
                jjtn000.setAttribute(NodeKey.OPERATOR, "/");
          break;
        case PERCENT:
          jj_consume_token(PERCENT);
                  jjtn000.setAttribute(NodeKey.OPERATOR, "%");
          break;
        default:
          jj_la1[67] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
        multiplicativeExpression();
        break;
      default:
        jj_la1[68] = jj_gen;
        ;
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
```

```
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, jjtree.nodeArity() > 1);
      }
    }
  }

  final public void castExpression() throws ParseException {
 /*@bgen(jjtree) #castExpression( isCasted) */
    ASTcastExpression jjtn000 = new ASTcastExpression(JJTCASTEXPRESSION);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);boolean isCasted = false;
    try {
      if (jj_2_36(2147483647)) {
        jj_consume_token(LEFT_PAREN);
        typeName();
        jj_consume_token(RIGHT_PAREN);

isCasted = true;
        castExpression();
      } else {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case INTEGER:
        case REAL:
        case CHARACTER:
        case STRING:
        case SIZEOF:
        case PLUS_PLUS:
        case MINUS_MINUS:
        case LEFT_PAREN:
        case AMPERSAND:
        case STAR:
        case PLUS:
        case MINUS:
        case BITWISE_NOT:
        case LOGICAL_NOT:
        case IDENTIFIER:
          unaryExpression(false);
          break;
        default:
          jj_la1[69] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
      }
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
```

```
        jjtree.closeNodeScope(jjtn000,  isCasted);
      }
    }
  }

  final public boolean unaryExpression(boolean onLeftSideOfAssignment) throws
ParseException {
 /*@bgen(jjtree) #unaryExpression( isUnaried) */
    ASTunaryExpression jjtn000 = new ASTunaryExpression(JJTUNARYEXPRESSION);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);boolean isUnaried = false;
    boolean isIdentifier = false;
    boolean identifierBeingModified = false;
    try {
      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
      case INTEGER:
      case REAL:
      case CHARACTER:
      case STRING:
      case LEFT_PAREN:
      case IDENTIFIER:
        isIdentifier = postfixExpression(onLeftSideOfAssignment);
        break;
      case PLUS_PLUS:
      case MINUS_MINUS:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case PLUS_PLUS:
          jj_consume_token(PLUS_PLUS);
          break;
        case MINUS_MINUS:
          jj_consume_token(MINUS_MINUS);
          break;
        default:
          jj_la1[70] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
        isUnaried = true;
        isIdentifier = unaryExpression(onLeftSideOfAssignment);
        identifierBeingModified = isIdentifier;
        break;
      case AMPERSAND:
      case STAR:
      case PLUS:
      case MINUS:
      case BITWISE_NOT:
      case LOGICAL_NOT:
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
        case AMPERSAND:
          jj_consume_token(AMPERSAND);
          break;
        case STAR:
          jj_consume_token(STAR);
          break;
        case PLUS:
          jj_consume_token(PLUS);
          break;
        case MINUS:
          jj_consume_token(MINUS);
          break;
        case BITWISE_NOT:
          jj_consume_token(BITWISE_NOT);
          break;
        case LOGICAL_NOT:
          jj_consume_token(LOGICAL_NOT);
          break;
        default:
          jj_la1[71] = jj_gen;
          jj_consume_token(-1);
          throw new ParseException();
        }
```

```
isUnaried = true;
        castExpression();
        break;
      case SIZEOF:
        jj_consume_token(SIZEOF);
                isUnaried = true;
        if (jj_2_37(2147483647)) {
          unaryExpression(onLeftSideOfAssignment);
        } else {
          switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
          case LEFT_PAREN:
            jj_consume_token(LEFT_PAREN);
            typeName();
            jj_consume_token(RIGHT_PAREN);
            break;
          default:
            jj_la1[72] = jj_gen;
            jj_consume_token(-1);
            throw new ParseException();
          }
        }
        break;
      default:
        jj_la1[73] = jj_gen;
        jj_consume_token(-1);
        throw new ParseException();
      }
      jjtree.closeNodeScope(jjtn000,  isUnaried);
      jjtc000 = false;
        jjtn000.setAttribute(NodeKey.MODIFYING_IDENTIFIER, onLeftSideOfAssignment ||
identifierBeingModified);
        {if (true) return isIdentifier;}
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000,  isUnaried);
      }
    }
    throw new Error("Missing return statement in function");
  }

  final public boolean postfixExpression(boolean onLeftSideOfAssignment) throws
ParseException {
 /*@bgen(jjtree) #postfixExpression( isPostfixed) */
    ASTpostfixExpression jjtn000 = new ASTpostfixExpression(JJTPOSTFIXEXPRESSION);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);boolean isPostfixed = false;
    boolean isIdentifier = false;
    boolean isArray = false;
    boolean identifierBeingModified = false;
    boolean reduction = false;
    String reductionOp = null;
    try {
      isIdentifier = primaryExpression(onLeftSideOfAssignment);
      label_15:
      while (true) {
        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
```

```
case PLUS_PLUS:
case MINUS_MINUS:
case POINTER_SELECT:
case DOT:
case LEFT_PAREN:
case LEFT_BRACKET:
  ;
  break;
default:
  jj_la1[74] = jj_gen;
  break label_15;
}
switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
case LEFT_BRACKET:
  jj_consume_token(LEFT_BRACKET);
               isArray = true;
  expression(NodeType.POSTFIX_EXPRESSION_NODE);
  jj_consume_token(RIGHT_BRACKET);
  break;
case LEFT_PAREN:
  jj_consume_token(LEFT_PAREN);
  switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
  case INTEGER:
  case REAL:
  case CHARACTER:
  case STRING:
  case SIZEOF:
  case PLUS_PLUS:
  case MINUS_MINUS:
  case LEFT_PAREN:
  case AMPERSAND:
  case STAR:
  case PLUS:
  case MINUS:
  case BITWISE_NOT:
  case LOGICAL_NOT:
  case IDENTIFIER:
    expression(NodeType.POSTFIX_EXPRESSION_NODE);
    break;
  default:
    jj_la1[75] = jj_gen;
    ;
  }
  jj_consume_token(RIGHT_PAREN);
  break;
case DOT:
  jj_consume_token(DOT);
  identifier(true);
  break;
case POINTER_SELECT:
  jj_consume_token(POINTER_SELECT);
  identifier(true);
  break;
case PLUS_PLUS:
  jj_consume_token(PLUS_PLUS);
identifierBeingModified = isIdentifier;
reduction = true;
reductionOp = "+";
  break;
case MINUS_MINUS:
  jj_consume_token(MINUS_MINUS);
identifierBeingModified = isIdentifier;
reduction = true;
reductionOp = "+";
  break;
default:
  jj_la1[76] = jj_gen;
  jj_consume_token(-1);
  throw new ParseException();
}
isPostfixed = true;
```

```
      }
      jjtree.closeNodeScope(jjtn000,  isPostfixed);
      jjtc000 = false;
        jjtn000.setAttribute(NodeKey.MODIFYING_IDENTIFIER, (onLeftSideOfAssignment &&
(!isArray)) || identifierBeingModified);
        jjtn000.setAttribute(NodeKey.ARRAY_POSTFIXED, isArray);
        jjtn000.setAttribute(NodeKey.REDUCTION, reduction);
        jjtn000.setAttribute(NodeKey.REDUCTION_OP, reductionOp);
        {if (true) return isIdentifier;}
    } catch (Throwable jjte000) {
      if (jjtc000) {
        jjtree.clearNodeScope(jjtn000);
        jjtc000 = false;
      } else {
        jjtree.popNode();
      }
      if (jjte000 instanceof RuntimeException) {
        {if (true) throw (RuntimeException)jjte000;}
      }
      if (jjte000 instanceof ParseException) {
        {if (true) throw (ParseException)jjte000;}
      }
      {if (true) throw (Error)jjte000;}
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000,  isPostfixed);
      }
    }
    throw new Error("Missing return statement in function");
  }

  final public boolean primaryExpression(boolean onLeftSideOfAssignment) throws
ParseException {
    boolean isIdentifier = false;
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case IDENTIFIER:
      identifier(onLeftSideOfAssignment);
                                          isIdentifier = true;
      break;
    case INTEGER:
    case REAL:
    case CHARACTER:
    case STRING:
      constant();
      break;
    case LEFT_PAREN:
      jj_consume_token(LEFT_PAREN);
      expression(NodeType.PRIMARY_EXPRESSION_NODE);
      jj_consume_token(RIGHT_PAREN);
      break;
    default:
      jj_la1[77] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
        {if (true) return isIdentifier;}
    throw new Error("Missing return statement in function");
  }

  final public Token identifier(Boolean onLeftSideOfAssignment) throws ParseException {
 /*@bgen(jjtree) identifier */
    ASTidentifier jjtn000 = new ASTidentifier(JJTIDENTIFIER);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);Token t;
    try {
      t = jj_consume_token(IDENTIFIER);
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
        jjtn000.setAttribute(NodeKey.NAME, t.image);
        jjtn000.setAttribute(NodeKey.MODIFYING_IDENTIFIER, onLeftSideOfAssignment);
        {if (true) return t;}
```

```
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
    throw new Error("Missing return statement in function");
  }

  final public void constant() throws ParseException {
    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
    case INTEGER:
      integer();
      break;
    case REAL:
      real();
      break;
    case STRING:
      string();
      break;
    case CHARACTER:
      character();
      break;
    default:
      jj_la1[78] = jj_gen;
      jj_consume_token(-1);
      throw new ParseException();
    }
  }

  final public void integer() throws ParseException {
/*@bgen(jjtree) integer */
    ASTinteger jjtn000 = new ASTinteger(JJTINTEGER);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);Token t;
    try {
      t = jj_consume_token(INTEGER);
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
        jjtn000.setAttribute(NodeKey.VALUE, t.image);
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void real() throws ParseException {
/*@bgen(jjtree) real */
    ASTreal jjtn000 = new ASTreal(JJTREAL);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);Token t;
    try {
      t = jj_consume_token(REAL);
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
        jjtn000.setAttribute(NodeKey.VALUE, t.image);
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void string() throws ParseException {
/*@bgen(jjtree) string */
    ASTstring jjtn000 = new ASTstring(JJTSTRING);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);Token t;
    try {
      t = jj_consume_token(STRING);
      jjtree.closeNodeScope(jjtn000, true);
```

```
    jjtc000 = false;
      jjtn000.setAttribute(NodeKey.VALUE, t.image);
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  final public void character() throws ParseException {
 /*@bgen(jjtree) character */
    ASTcharacter jjtn000 = new ASTcharacter(JJTCHARACTER);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);Token t;
    try {
      t = jj_consume_token(CHARACTER);
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
        jjtn000.setAttribute(NodeKey.VALUE, t.image);
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  }

  private boolean jj_2_1(int xla) {
    jj_la = xla; jj_lastpos = jj_scanpos = token;
    try { return !jj_3_1(); }
    catch(LookaheadSuccess ls) { return true; }
    finally { jj_save(0, xla); }
  }

  private boolean jj_2_2(int xla) {
    jj_la = xla; jj_lastpos = jj_scanpos = token;
    try { return !jj_3_2(); }
    catch(LookaheadSuccess ls) { return true; }
    finally { jj_save(1, xla); }
  }

  private boolean jj_2_3(int xla) {
    jj_la = xla; jj_lastpos = jj_scanpos = token;
    try { return !jj_3_3(); }
    catch(LookaheadSuccess ls) { return true; }
    finally { jj_save(2, xla); }
  }

  private boolean jj_2_4(int xla) {
    jj_la = xla; jj_lastpos = jj_scanpos = token;
    try { return !jj_3_4(); }
    catch(LookaheadSuccess ls) { return true; }
    finally { jj_save(3, xla); }
  }

  private boolean jj_2_5(int xla) {
    jj_la = xla; jj_lastpos = jj_scanpos = token;
    try { return !jj_3_5(); }
    catch(LookaheadSuccess ls) { return true; }
    finally { jj_save(4, xla); }
  }

  private boolean jj_2_6(int xla) {
    jj_la = xla; jj_lastpos = jj_scanpos = token;
    try { return !jj_3_6(); }
    catch(LookaheadSuccess ls) { return true; }
    finally { jj_save(5, xla); }
  }

  private boolean jj_2_7(int xla) {
    jj_la = xla; jj_lastpos = jj_scanpos = token;
    try { return !jj_3_7(); }
```

```
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(6, xla); }
}

private boolean jj_2_8(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_8(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(7, xla); }
}

private boolean jj_2_9(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_9(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(8, xla); }
}

private boolean jj_2_10(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_10(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(9, xla); }
}

private boolean jj_2_11(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_11(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(10, xla); }
}

private boolean jj_2_12(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_12(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(11, xla); }
}

private boolean jj_2_13(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_13(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(12, xla); }
}

private boolean jj_2_14(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_14(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(13, xla); }
}

private boolean jj_2_15(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_15(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(14, xla); }
}

private boolean jj_2_16(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_16(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(15, xla); }
}

private boolean jj_2_17(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_17(); }
  catch(LookaheadSuccess ls) { return true; }
```

```
  finally { jj_save(16, xla); }
}

private boolean jj_2_18(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_18(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(17, xla); }
}

private boolean jj_2_19(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_19(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(18, xla); }
}

private boolean jj_2_20(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_20(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(19, xla); }
}

private boolean jj_2_21(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_21(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(20, xla); }
}

private boolean jj_2_22(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_22(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(21, xla); }
}

private boolean jj_2_23(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_23(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(22, xla); }
}

private boolean jj_2_24(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_24(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(23, xla); }
}

private boolean jj_2_25(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_25(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(24, xla); }
}

private boolean jj_2_26(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_26(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(25, xla); }
}

private boolean jj_2_27(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_27(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(26, xla); }
```

```
}

private boolean jj_2_28(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_28(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(27, xla); }
}

private boolean jj_2_29(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_29(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(28, xla); }
}

private boolean jj_2_30(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_30(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(29, xla); }
}

private boolean jj_2_31(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_31(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(30, xla); }
}

private boolean jj_2_32(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_32(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(31, xla); }
}

private boolean jj_2_33(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_33(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(32, xla); }
}

private boolean jj_2_34(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_34(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(33, xla); }
}

private boolean jj_2_35(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_35(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(34, xla); }
}

private boolean jj_2_36(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_36(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(35, xla); }
}

private boolean jj_2_37(int xla) {
  jj_la = xla; jj_lastpos = jj_scanpos = token;
  try { return !jj_3_37(); }
  catch(LookaheadSuccess ls) { return true; }
  finally { jj_save(36, xla); }
}
```

```
private boolean jj_3R_195() {
  if (jj_3R_40()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_197()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_49() {
  if (jj_3R_92()) return true;
  return false;
}

private boolean jj_3_7() {
  if (jj_3R_20()) return true;
  return false;
}

private boolean jj_3R_48() {
  if (jj_3R_91()) return true;
  return false;
}

private boolean jj_3R_47() {
  if (jj_3R_90()) return true;
  return false;
}

private boolean jj_3R_199() {
  if (jj_scan_token(MINUS)) return true;
  return false;
}

private boolean jj_3R_198() {
  if (jj_scan_token(PLUS)) return true;
  return false;
}

private boolean jj_3R_21() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(24)) {
  jj_scanpos = xsp;
  if (jj_scan_token(25)) {
  jj_scanpos = xsp;
  if (jj_scan_token(26)) {
  jj_scanpos = xsp;
  if (jj_scan_token(27)) {
  jj_scanpos = xsp;
  if (jj_scan_token(28)) {
  jj_scanpos = xsp;
  if (jj_scan_token(29)) {
  jj_scanpos = xsp;
  if (jj_scan_token(30)) {
  jj_scanpos = xsp;
  if (jj_scan_token(31)) {
  jj_scanpos = xsp;
  if (jj_scan_token(32)) {
  jj_scanpos = xsp;
  if (jj_3R_47()) {
  jj_scanpos = xsp;
  if (jj_3R_48()) {
  jj_scanpos = xsp;
  jj_lookingAhead = true;
  jj_semLA = isType(getToken(1).image);
  jj_lookingAhead = false;
  if (!jj_semLA || jj_3R_49()) return true;
  }
  }
  }
```

```
    }
    }
    }
    }
    }
    }
    }
    }
    return false;
  }

  private boolean jj_3R_196() {
    Token xsp;
    xsp = jj_scanpos;
    if (jj_3R_198()) {
    jj_scanpos = xsp;
    if (jj_3R_199()) return true;
    }
    if (jj_3R_192()) return true;
    return false;
  }

  private boolean jj_3_5() {
    if (jj_3R_20()) return true;
    return false;
  }

  private boolean jj_3R_192() {
    if (jj_3R_195()) return true;
    Token xsp;
    xsp = jj_scanpos;
    if (jj_3R_196()) jj_scanpos = xsp;
    return false;
  }

  private boolean jj_3_13() {
    if (jj_scan_token(COMMA)) return true;
    if (jj_3R_22()) return true;
    return false;
  }

  private boolean jj_3R_84() {
    Token xsp;
    xsp = jj_scanpos;
    if (jj_scan_token(14)) {
    jj_scanpos = xsp;
    if (jj_scan_token(15)) {
    jj_scanpos = xsp;
    if (jj_scan_token(19)) {
    jj_scanpos = xsp;
    if (jj_scan_token(16)) {
    jj_scanpos = xsp;
    if (jj_3R_118()) return true;
    }
    }
    }
    }
    return false;
  }

  private boolean jj_3R_193() {
    Token xsp;
    xsp = jj_scanpos;
    if (jj_scan_token(64)) {
    jj_scanpos = xsp;
    if (jj_scan_token(65)) return true;
    }
    if (jj_3R_190()) return true;
    return false;
  }
```

```
private boolean jj_3R_171() {
  if (jj_scan_token(AMPERSAND)) return true;
  if (jj_3R_161()) return true;
  return false;
}

private boolean jj_3R_190() {
  if (jj_3R_192()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_193()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_162() {
  if (jj_scan_token(BITWISE_XOR)) return true;
  if (jj_3R_153()) return true;
  return false;
}

private boolean jj_3R_39() {
  if (jj_3R_26()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_80()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_191() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(66)) {
  jj_scanpos = xsp;
  if (jj_scan_token(68)) {
  jj_scanpos = xsp;
  if (jj_scan_token(67)) {
  jj_scanpos = xsp;
  if (jj_scan_token(69)) return true;
  }
  }
  }
  if (jj_3R_180()) return true;
  return false;
}

private boolean jj_3R_180() {
  if (jj_3R_190()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_191()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3_9() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_154() {
  if (jj_scan_token(BITWISE_OR)) return true;
  if (jj_3R_137()) return true;
  return false;
}

private boolean jj_3R_139() {
  if (jj_scan_token(EQUALS)) return true;
  if (jj_3R_22()) return true;
  return false;
}

private boolean jj_3_3() {
```

```
  if (jj_3R_18()) return true;
  return false;
}

private boolean jj_3R_175() {
  if (jj_3R_22()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3_13()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3_11() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_181() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(70)) {
  jj_scanpos = xsp;
  if (jj_scan_token(71)) return true;
  }
  if (jj_3R_170()) return true;
  return false;
}

private boolean jj_3_10() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_51() {
  if (jj_scan_token(LEFT_BRACE)) return true;
  if (jj_3R_175()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(90)) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_BRACE)) return true;
  return false;
}

private boolean jj_3R_170() {
  if (jj_3R_180()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_181()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_138() {
  if (jj_scan_token(LOGICAL_AND)) return true;
  if (jj_3R_115()) return true;
  return false;
}

private boolean jj_3R_50() {
  if (jj_3R_73()) return true;
  return false;
}

private boolean jj_3R_120() {
  if (jj_scan_token(COMMA)) return true;
  if (jj_3R_119()) return true;
  return false;
}

private boolean jj_3R_22() {
```

```
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_50()) {
  jj_scanpos = xsp;
  if (jj_3R_51()) return true;
  }
  return false;
}

private boolean jj_3R_85() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_161() {
  if (jj_3R_170()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_171()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_116() {
  if (jj_scan_token(LOGICAL_OR)) return true;
  if (jj_3R_78()) return true;
  return false;
}

private boolean jj_3R_88() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_119() {
  if (jj_3R_27()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_139()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_86() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_46() {
  if (jj_3R_89()) return true;
  return false;
}

private boolean jj_3R_153() {
  if (jj_3R_161()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_162()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_79() {
  if (jj_scan_token(QUESTION_MARK)) return true;
  if (jj_3R_36()) return true;
  if (jj_scan_token(COLON)) return true;
  if (jj_3R_38()) return true;
  return false;
}

private boolean jj_3R_137() {
  if (jj_3R_153()) return true;
  Token xsp;
  xsp = jj_scanpos;
```

```
  if (jj_3R_154()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_89() {
  if (jj_3R_119()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_120()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_44() {
  if (jj_3R_87()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_88()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_115() {
  if (jj_3R_137()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_138()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3_12() {
  if (jj_3R_21()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_86()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_43() {
  if (jj_3R_84()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_85()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_19() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_43()) {
  jj_scanpos = xsp;
  if (jj_3_12()) {
  jj_scanpos = xsp;
  if (jj_3R_44()) return true;
  }
  }
  return false;
}

private boolean jj_3_8() {
  if (jj_3R_18()) return true;
  return false;
}

private boolean jj_3R_78() {
  if (jj_3R_115()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_116()) jj_scanpos = xsp;
  return false;
}
```

```
private boolean jj_3R_18() {
  if (jj_3R_19()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_46()) jj_scanpos = xsp;
  if (jj_scan_token(SEMICOLON)) return true;
  return false;
}

private boolean jj_3_6() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_38() {
  if (jj_3R_78()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_79()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_45() {
  if (jj_3R_18()) return true;
  return false;
}

private boolean jj_3R_20() {
  Token xsp;
  if (jj_3R_45()) return true;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_45()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_189() {
  if (jj_scan_token(BITWISE_XOR_EQUALS)) return true;
  return false;
}

private boolean jj_3R_188() {
  if (jj_scan_token(AMP_EQUALS)) return true;
  return false;
}

private boolean jj_3_4() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_102() {
  if (jj_3R_38()) return true;
  return false;
}

private boolean jj_3R_187() {
  if (jj_scan_token(BITWISE_OR_EQUALS)) return true;
  return false;
}

private boolean jj_3R_42() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_186() {
  if (jj_scan_token(MINUS_EQUALS)) return true;
  return false;
```

```
}

private boolean jj_3R_17() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_42()) jj_scanpos = xsp;
  if (jj_3R_27()) return true;
  xsp = jj_scanpos;
  if (jj_3_7()) jj_scanpos = xsp;
  if (jj_scan_token(LEFT_BRACE)) return true;
  return false;
}

private boolean jj_3R_185() {
  if (jj_scan_token(PLUS_EQUALS)) return true;
  return false;
}

private boolean jj_3_2() {
  if (jj_3R_17()) return true;
  return false;
}

private boolean jj_3R_184() {
  if (jj_scan_token(SLASH_EQUALS)) return true;
  return false;
}

private boolean jj_3R_183() {
  if (jj_scan_token(STAR_EQUALS)) return true;
  return false;
}

private boolean jj_3_35() {
  if (jj_3R_38()) return true;
  return false;
}

private boolean jj_3R_117() {
  if (jj_3R_19()) return true;
  return false;
}

private boolean jj_3R_83() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_117()) jj_scanpos = xsp;
  if (jj_3R_27()) return true;
  return false;
}

private boolean jj_3R_41() {
  if (jj_3R_83()) return true;
  return false;
}

private boolean jj_3R_169() {
  if (jj_scan_token(COMMA)) return true;
  if (jj_3R_73()) return true;
  return false;
}

private boolean jj_3R_16() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_41()) {
  jj_scanpos = xsp;
  if (jj_3_3()) return true;
  }
  return false;
}
```

```
private boolean jj_3_34() {
  if (jj_3R_37()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(78)) {
  jj_scanpos = xsp;
  if (jj_scan_token(79)) {
  jj_scanpos = xsp;
  if (jj_scan_token(80)) {
  jj_scanpos = xsp;
  if (jj_scan_token(81)) {
  jj_scanpos = xsp;
  if (jj_scan_token(82)) {
  jj_scanpos = xsp;
  if (jj_scan_token(83)) {
  jj_scanpos = xsp;
  if (jj_scan_token(84)) {
  jj_scanpos = xsp;
  if (jj_scan_token(85)) {
  jj_scanpos = xsp;
  if (jj_scan_token(86)) {
  jj_scanpos = xsp;
  if (jj_scan_token(87)) {
  jj_scanpos = xsp;
  if (jj_scan_token(88)) return true;
  }
  }
  }
  }
  }
  }
  }
  }
  }
  return false;
}

private boolean jj_3_1() {
  if (jj_3R_16()) return true;
  return false;
}

private boolean jj_3R_111() {
  if (jj_3R_37()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(78)) {
  jj_scanpos = xsp;
  if (jj_3R_183()) {
  jj_scanpos = xsp;
  if (jj_3R_184()) {
  jj_scanpos = xsp;
  if (jj_scan_token(81)) {
  jj_scanpos = xsp;
  if (jj_3R_185()) {
  jj_scanpos = xsp;
  if (jj_3R_186()) {
  jj_scanpos = xsp;
  if (jj_scan_token(84)) {
  jj_scanpos = xsp;
  if (jj_scan_token(85)) {
  jj_scanpos = xsp;
  if (jj_3R_187()) {
  jj_scanpos = xsp;
  if (jj_3R_188()) {
  jj_scanpos = xsp;
  if (jj_3R_189()) return true;
  }
  }
```

```
    }
    }
    }
    }
    }
    }
    }
    }
    if (jj_3R_73()) return true;
    return false;
  }

  private boolean jj_3R_73() {
    Token xsp;
    xsp = jj_scanpos;
    if (jj_3R_111()) {
    jj_scanpos = xsp;
    if (jj_3_35()) return true;
    }
    return false;
  }

  private boolean jj_3R_36() {
    if (jj_3R_73()) return true;
    Token xsp;
    while (true) {
      xsp = jj_scanpos;
      if (jj_3R_169()) { jj_scanpos = xsp; break; }
    }
    return false;
  }

  private boolean jj_3R_110() {
    if (jj_scan_token(DO)) return true;
    return false;
  }

  private boolean jj_3_32() {
    if (jj_3R_21()) return true;
    return false;
  }

  private boolean jj_3_33() {
    Token xsp;
    xsp = jj_scanpos;
    if (jj_3_32()) jj_scanpos = xsp;
    if (jj_3R_36()) return true;
    return false;
  }

  private boolean jj_3R_109() {
    if (jj_scan_token(WHILE)) return true;
    return false;
  }

  private boolean jj_3R_108() {
    if (jj_scan_token(FOR)) return true;
    return false;
  }

  private boolean jj_3R_107() {
    if (jj_scan_token(SWITCH)) return true;
    return false;
  }

  private boolean jj_3_31() {
    if (jj_scan_token(ELSE)) return true;
    if (jj_3R_35()) return true;
    return false;
  }
```

```
private boolean jj_3R_106() {
  if (jj_scan_token(IF)) return true;
  return false;
}

private boolean jj_3R_134() {
  if (jj_3R_36()) return true;
  return false;
}

private boolean jj_3_30() {
  if (jj_3R_20()) return true;
  return false;
}

private boolean jj_3R_105() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_134()) jj_scanpos = xsp;
  if (jj_scan_token(SEMICOLON)) return true;
  return false;
}

private boolean jj_3R_133() {
  if (jj_scan_token(RETURN)) return true;
  return false;
}

private boolean jj_3R_132() {
  if (jj_scan_token(CONTINUE)) return true;
  return false;
}

private boolean jj_3R_131() {
  if (jj_scan_token(BREAK)) return true;
  return false;
}

private boolean jj_3R_143() {
  if (jj_3R_33()) return true;
  return false;
}

private boolean jj_3R_130() {
  if (jj_scan_token(GOTO)) return true;
  return false;
}

private boolean jj_3R_126() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_143()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_104() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_130()) {
  jj_scanpos = xsp;
  if (jj_3R_131()) {
  jj_scanpos = xsp;
  if (jj_3R_132()) {
  jj_scanpos = xsp;
  if (jj_3R_133()) return true;
  }
  }
  }
  return false;
}
```

```
private boolean jj_3R_64() {
  if (jj_3R_52()) return true;
  if (jj_scan_token(COLON)) return true;
  return false;
}

private boolean jj_3R_63() {
  if (jj_scan_token(CDEFAULT)) return true;
  if (jj_scan_token(COLON)) return true;
  return false;
}

private boolean jj_3R_62() {
  if (jj_scan_token(CASE)) return true;
  if (jj_3R_102()) return true;
  return false;
}

private boolean jj_3R_34() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_62()) {
  jj_scanpos = xsp;
  if (jj_3R_63()) {
  jj_scanpos = xsp;
  if (jj_3R_64()) return true;
  }
  }
  return false;
}

private boolean jj_3R_103() {
  if (jj_scan_token(LEFT_BRACE)) return true;
  return false;
}

private boolean jj_3R_72() {
  if (jj_3R_110()) return true;
  return false;
}

private boolean jj_3R_71() {
  if (jj_3R_109()) return true;
  return false;
}

private boolean jj_3R_70() {
  if (jj_3R_108()) return true;
  return false;
}

private boolean jj_3R_164() {
  if (jj_scan_token(COMMA)) return true;
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_69() {
  if (jj_3R_107()) return true;
  return false;
}

private boolean jj_3R_68() {
  if (jj_3R_106()) return true;
  return false;
}

private boolean jj_3R_67() {
  if (jj_3R_105()) return true;
  return false;
}
```

```
private boolean jj_3R_66() {
  if (jj_3R_104()) return true;
  return false;
}

private boolean jj_3_29() {
  if (jj_3R_34()) return true;
  return false;
}

private boolean jj_3R_65() {
  if (jj_3R_103()) return true;
  return false;
}

private boolean jj_3R_35() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_65()) {
  jj_scanpos = xsp;
  if (jj_3_29()) {
  jj_scanpos = xsp;
  if (jj_3R_66()) {
  jj_scanpos = xsp;
  if (jj_3R_67()) {
  jj_scanpos = xsp;
  if (jj_3R_68()) {
  jj_scanpos = xsp;
  if (jj_3R_69()) {
  jj_scanpos = xsp;
  if (jj_3R_70()) {
  jj_scanpos = xsp;
  if (jj_3R_71()) {
  jj_scanpos = xsp;
  if (jj_3R_72()) return true;
  }
  }
  }
  }
  }
  }
  }
  return false;
}

private boolean jj_3_24() {
  if (jj_3R_27()) return true;
  return false;
}

private boolean jj_3R_182() {
  if (jj_3R_102()) return true;
  return false;
}

private boolean jj_3_28() {
  if (jj_3R_30()) return true;
  return false;
}

private boolean jj_3R_172() {
  if (jj_3R_102()) return true;
  return false;
}

private boolean jj_3_26() {
  if (jj_3R_30()) return true;
  return false;
}
```

```
private boolean jj_3R_125() {
  if (jj_3R_27()) return true;
  return false;
}

private boolean jj_3R_114() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  if (jj_3R_39()) return true;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3R_157() {
  if (jj_3R_52()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_164()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_179() {
  if (jj_scan_token(CHARACTER)) return true;
  return false;
}

private boolean jj_3R_174() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_28()) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3_23() {
  if (jj_scan_token(COMMA)) return true;
  if (jj_3R_31()) return true;
  return false;
}

private boolean jj_3R_173() {
  if (jj_scan_token(LEFT_BRACKET)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_182()) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_BRACKET)) return true;
  return false;
}

private boolean jj_3R_163() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_173()) {
  jj_scanpos = xsp;
  if (jj_3R_174()) return true;
  }
  return false;
}

private boolean jj_3R_129() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_26()) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}
```

```
private boolean jj_3R_128() {
  if (jj_scan_token(LEFT_BRACKET)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_172()) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_BRACKET)) return true;
  return false;
}

private boolean jj_3R_142() {
  if (jj_3R_157()) return true;
  return false;
}

private boolean jj_3_27() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  if (jj_3R_33()) return true;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3R_101() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_27()) {
  jj_scanpos = xsp;
  if (jj_3R_128()) {
  jj_scanpos = xsp;
  if (jj_3R_129()) return true;
  }
  }
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_163()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_178() {
  if (jj_scan_token(STRING)) return true;
  return false;
}

private boolean jj_3R_168() {
  if (jj_3R_179()) return true;
  return false;
}

private boolean jj_3R_156() {
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_100() {
  if (jj_3R_32()) return true;
  return false;
}

private boolean jj_3R_61() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_100()) jj_scanpos = xsp;
  if (jj_3R_101()) return true;
  return false;
}

private boolean jj_3_25() {
  if (jj_3R_32()) return true;
  return false;
}
```

```
private boolean jj_3R_33() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_25()) {
  jj_scanpos = xsp;
  if (jj_3R_61()) return true;
  }
  return false;
}

private boolean jj_3R_158() {
  if (jj_scan_token(COMMA)) return true;
  if (jj_scan_token(THREE_DOTS)) return true;
  return false;
}

private boolean jj_3R_31() {
  if (jj_3R_19()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_125()) {
  jj_scanpos = xsp;
  if (jj_3R_126()) return true;
  }
  return false;
}

private boolean jj_3R_167() {
  if (jj_3R_178()) return true;
  return false;
}

private boolean jj_3R_177() {
  if (jj_scan_token(REAL)) return true;
  return false;
}

private boolean jj_3R_58() {
  if (jj_3R_31()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3_23()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_166() {
  if (jj_3R_177()) return true;
  return false;
}

private boolean jj_3R_141() {
  if (jj_3R_102()) return true;
  return false;
}

private boolean jj_3R_30() {
  if (jj_3R_58()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_158()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_176() {
  if (jj_scan_token(INTEGER)) return true;
  return false;
}

private boolean jj_3R_155() {
```

```
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_165() {
  if (jj_3R_176()) return true;
  return false;
}

private boolean jj_3R_124() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_142()) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3R_159() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_165()) {
  jj_scanpos = xsp;
  if (jj_3R_166()) {
  jj_scanpos = xsp;
  if (jj_3R_167()) {
  jj_scanpos = xsp;
  if (jj_3R_168()) return true;
  }
  }
  }
  return false;
}

private boolean jj_3R_60() {
  if (jj_3R_32()) return true;
  return false;
}

private boolean jj_3_22() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  if (jj_3R_30()) return true;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3R_123() {
  if (jj_scan_token(LEFT_BRACKET)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_141()) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_BRACKET)) return true;
  return false;
}

private boolean jj_3R_97() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_123()) {
  jj_scanpos = xsp;
  if (jj_3_22()) {
  jj_scanpos = xsp;
  if (jj_3R_124()) return true;
  }
  }
  return false;
}

private boolean jj_3R_96() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  if (jj_3R_27()) return true;
```

```
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3R_52() {
  if (jj_scan_token(IDENTIFIER)) return true;
  return false;
}

private boolean jj_3R_95() {
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_55() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_95()) {
  jj_scanpos = xsp;
  if (jj_3R_96()) return true;
  }
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_97()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_146() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  if (jj_3R_36()) return true;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3R_145() {
  if (jj_3R_159()) return true;
  return false;
}

private boolean jj_3R_59() {
  if (jj_3R_99()) return true;
  return false;
}

private boolean jj_3R_127() {
  if (jj_3R_87()) return true;
  return false;
}

private boolean jj_3R_98() {
  if (jj_scan_token(EQUALS)) return true;
  if (jj_3R_102()) return true;
  return false;
}

private boolean jj_3R_144() {
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_99() {
  Token xsp;
  if (jj_3R_127()) return true;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_127()) { jj_scanpos = xsp; break; }
  }
  return false;
}
```

```
private boolean jj_3R_135() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_144()) {
  jj_scanpos = xsp;
  if (jj_3R_145()) {
  jj_scanpos = xsp;
  if (jj_3R_146()) return true;
  }
  }
  return false;
}

private boolean jj_3R_32() {
  if (jj_scan_token(STAR)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_59()) jj_scanpos = xsp;
  xsp = jj_scanpos;
  if (jj_3R_60()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_28() {
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_57() {
  if (jj_scan_token(COMMA)) return true;
  if (jj_3R_56()) return true;
  return false;
}

private boolean jj_3R_54() {
  if (jj_3R_32()) return true;
  return false;
}

private boolean jj_3R_27() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_54()) jj_scanpos = xsp;
  if (jj_3R_55()) return true;
  return false;
}

private boolean jj_3_37() {
  if (jj_3R_37()) return true;
  return false;
}

private boolean jj_3R_200() {
  if (jj_3R_27()) return true;
  return false;
}

private boolean jj_3R_56() {
  if (jj_3R_52()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_98()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_160() {
  if (jj_3R_36()) return true;
  return false;
}

private boolean jj_3_17() {
```

```
  if (jj_3R_26()) return true;
  return false;
}

private boolean jj_3_21() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_28()) jj_scanpos = xsp;
  if (jj_scan_token(LEFT_BRACE)) return true;
  if (jj_3R_29()) return true;
  if (jj_scan_token(RIGHT_BRACE)) return true;
  return false;
}

private boolean jj_3R_113() {
  if (jj_3R_37()) return true;
  return false;
}

private boolean jj_3R_194() {
  if (jj_scan_token(COMMA)) return true;
  if (jj_3R_140()) return true;
  return false;
}

private boolean jj_3_16() {
  if (jj_3R_26()) return true;
  return false;
}

private boolean jj_3R_29() {
  if (jj_3R_56()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_57()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_152() {
  if (jj_scan_token(MINUS_MINUS)) return true;
  return false;
}

private boolean jj_3R_91() {
  if (jj_scan_token(ENUM)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_21()) {
  jj_scanpos = xsp;
  if (jj_3R_156()) return true;
  }
  return false;
}

private boolean jj_3R_94() {
  if (jj_3R_26()) return true;
  return false;
}

private boolean jj_3R_151() {
  if (jj_scan_token(PLUS_PLUS)) return true;
  return false;
}

private boolean jj_3R_150() {
  if (jj_scan_token(POINTER_SELECT)) return true;
  if (jj_3R_52()) return true;
  return false;
}
```

```
private boolean jj_3R_149() {
  if (jj_scan_token(DOT)) return true;
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_93() {
  if (jj_3R_26()) return true;
  return false;
}

private boolean jj_3R_148() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_160()) jj_scanpos = xsp;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  return false;
}

private boolean jj_3_20() {
  if (jj_3R_21()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_200()) jj_scanpos = xsp;
  if (jj_scan_token(COLON)) return true;
  if (jj_3R_102()) return true;
  return false;
}

private boolean jj_3R_147() {
  if (jj_scan_token(LEFT_BRACKET)) return true;
  if (jj_3R_36()) return true;
  if (jj_scan_token(RIGHT_BRACKET)) return true;
  return false;
}

private boolean jj_3R_136() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_147()) {
  jj_scanpos = xsp;
  if (jj_3R_148()) {
  jj_scanpos = xsp;
  if (jj_3R_149()) {
  jj_scanpos = xsp;
  if (jj_3R_150()) {
  jj_scanpos = xsp;
  if (jj_3R_151()) {
  jj_scanpos = xsp;
  if (jj_3R_152()) return true;
  }
  }
  }
  }
  }
  return false;
}

private boolean jj_3_19() {
  if (jj_3R_27()) return true;
  return false;
}

private boolean jj_3R_140() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_19()) {
  jj_scanpos = xsp;
  if (jj_3_20()) return true;
```

```
  }
  return false;
}

private boolean jj_3R_112() {
  if (jj_3R_135()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_136()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_122() {
  if (jj_3R_140()) return true;
  Token xsp;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3R_194()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_23() {
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_53() {
  if (jj_3R_87()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_94()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_26() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_18()) {
  jj_scanpos = xsp;
  if (jj_3R_53()) return true;
  }
  return false;
}

private boolean jj_3_18() {
  if (jj_3R_21()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_93()) jj_scanpos = xsp;
  return false;
}

private boolean jj_3R_77() {
  if (jj_scan_token(SIZEOF)) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_113()) {
  jj_scanpos = xsp;
  if (jj_3R_114()) return true;
  }
  return false;
}

private boolean jj_3R_76() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(56)) {
  jj_scanpos = xsp;
```

```
  if (jj_scan_token(57)) {
  jj_scanpos = xsp;
  if (jj_scan_token(58)) {
  jj_scanpos = xsp;
  if (jj_scan_token(59)) {
  jj_scanpos = xsp;
  if (jj_scan_token(60)) {
  jj_scanpos = xsp;
  if (jj_scan_token(61)) return true;
  }
  }
  }
  }
  }
  if (jj_3R_40()) return true;
  return false;
}

private boolean jj_3R_25() {
  if (jj_3R_26()) return true;
  if (jj_3R_122()) return true;
  if (jj_scan_token(SEMICOLON)) return true;
  return false;
}

private boolean jj_3_14() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_23()) jj_scanpos = xsp;
  if (jj_scan_token(LEFT_BRACE)) return true;
  if (jj_3R_24()) return true;
  if (jj_scan_token(RIGHT_BRACE)) return true;
  return false;
}

private boolean jj_3R_118() {
  if (jj_scan_token(TYPEDEF)) return true;
  return false;
}

private boolean jj_3_15() {
  if (jj_3R_25()) return true;
  return false;
}

private boolean jj_3R_24() {
  Token xsp;
  if (jj_3_15()) return true;
  while (true) {
    xsp = jj_scanpos;
    if (jj_3_15()) { jj_scanpos = xsp; break; }
  }
  return false;
}

private boolean jj_3R_75() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(46)) {
  jj_scanpos = xsp;
  if (jj_scan_token(47)) return true;
  }
  if (jj_3R_37()) return true;
  return false;
}

private boolean jj_3R_74() {
  if (jj_3R_112()) return true;
  return false;
}
```

```
private boolean jj_3R_37() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_74()) {
  jj_scanpos = xsp;
  if (jj_3R_75()) {
  jj_scanpos = xsp;
  if (jj_3R_76()) {
  jj_scanpos = xsp;
  if (jj_3R_77()) return true;
  }
  }
  }
  return false;
}

private boolean jj_3_36() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  if (jj_3R_39()) return true;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  if (jj_3R_40()) return true;
  return false;
}

private boolean jj_3R_121() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_scan_token(17)) {
  jj_scanpos = xsp;
  if (jj_scan_token(18)) return true;
  }
  return false;
}

private boolean jj_3R_82() {
  if (jj_3R_37()) return true;
  return false;
}

private boolean jj_3R_90() {
  if (jj_3R_121()) return true;
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3_14()) {
  jj_scanpos = xsp;
  if (jj_3R_155()) return true;
  }
  return false;
}

private boolean jj_3R_81() {
  if (jj_scan_token(LEFT_PAREN)) return true;
  if (jj_3R_39()) return true;
  if (jj_scan_token(RIGHT_PAREN)) return true;
  if (jj_3R_40()) return true;
  return false;
}

private boolean jj_3R_40() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_81()) {
  jj_scanpos = xsp;
  if (jj_3R_82()) return true;
  }
  return false;
}

private boolean jj_3R_87() {
  Token xsp;
  xsp = jj_scanpos;
```

```
  if (jj_scan_token(22)) {
  jj_scanpos = xsp;
  if (jj_scan_token(23)) return true;
  }
  return false;
}

private boolean jj_3R_80() {
  if (jj_3R_33()) return true;
  return false;
}

private boolean jj_3R_203() {
  if (jj_scan_token(PERCENT)) return true;
  return false;
}

private boolean jj_3R_202() {
  if (jj_scan_token(SLASH)) return true;
  return false;
}

private boolean jj_3R_201() {
  if (jj_scan_token(STAR)) return true;
  return false;
}

private boolean jj_3R_92() {
  if (jj_3R_52()) return true;
  return false;
}

private boolean jj_3R_197() {
  Token xsp;
  xsp = jj_scanpos;
  if (jj_3R_201()) {
  jj_scanpos = xsp;
  if (jj_3R_202()) {
  jj_scanpos = xsp;
  if (jj_3R_203()) return true;
  }
  }
  if (jj_3R_195()) return true;
  return false;
}

/** Generated Token Manager. */
public CParserTokenManager token_source;
SimpleCharStream jj_input_stream;
/** Current token. */
public Token token;
/** Next token. */
public Token jj_nt;
private int jj_ntk;
private Token jj_scanpos, jj_lastpos;
private int jj_la;
/** Whether we are looking ahead. */
private boolean jj_lookingAhead = false;
private boolean jj_semLA;
private int jj_gen;
final private int[] jj_la1 = new int[79];
static private int[] jj_la1_0;
static private int[] jj_la1_1;
static private int[] jj_la1_2;
static private int[] jj_la1_3;
static {
    jj_la1_init_0();
    jj_la1_init_1();
    jj_la1_init_2();
    jj_la1_init_3();
 }
```

```
   private static void jj_la1_init_0() {
      jj_la1_0 = new int[]
{0x0,0x19c000,0xc00000,0x0,0x0,0x0,0x3c00,0x0,0x19c000,0xff260000,0xc00000,0x0,0x0,0x6000
0,0xc00000,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0xc00000,0x0,0xc00000,0x0,0x0,0x3c00,0x0,0x0,0x0,0
x0,0x0,0x0,0x0,0x3c00,0x0,0x0,0x3c00,0x0,0x0,0x3c00,0x0,0x3c00,0x3c00,0x0,0x3c00,0x0,0x3c
00,0x3c00,0x3c00,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,
0x3c00,0x0,0x0,0x0,0x0,0x3c00,0x0,0x3c00,0x0,0x3c00,0x3c00,};
   }
   private static void jj_la1_init_1() {
      jj_la1_1 = new int[]
{0x2040000,0x0,0x0,0x0,0x0,0x0,0x3f44d000,0x2140000,0x0,0x1,0x0,0x0,0x0,0x0,0x0,0x0,0x204
0000,0x0,0x0,0x0,0x0,0x0,0x2000000,0x0,0x2000000,0x0,0x40000,0x140000,0x3f04d000,0x0,0x100000
,0x40000,0x0,0x2140000,0x2000000,0x2140000,0x3f04d000,0x140000,0x140000,0x3f04d000,0x1400
00,0x0,0x3f44fffa,0x400000,0x3f04fbea,0x3f44fffa,0x410,0x3f04d000,0x2b00,0x3f04d000,0x3f0
4d000,0x3f04d000,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x1000000,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0xc000000,
0xc000000,0xc2000000,0xc2000000,0x3f04d000,0xc000,0x3f000000,0x40000,0x3f04d000,0x17c000,
0x3f04d000,0x17c000,0x40000,0x0,};
   }
   private static void jj_la1_init_2() {
      jj_la1_2 = new int[]
{0x0,0x0,0x0,0x4000000,0x4000,0x4000000,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x4000000,0x0
,0x0,0x0,0x4000000,0x4000,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x4000000,0x0,0x0,0x0,0
x0,0x0,0x0,0x0,0x0,0x0,0x4000000,0x2000000,0x0,0x2000000,0x2000000,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
4000000,0x1ffc000,0x1000,0x800,0x400,0x200,0x100,0x0,0xc0,0xc0,0x3c,0x3c,0x3,0x3,0x0,0x0,
0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,};
   }
   private static void jj_la1_init_3() {
      jj_la1_3 = new int[]
{0x4,0x0,0x0,0x0,0x0,0x0,0x4,0x0,0x0,0x0,0x0,0x4,0x4,0x0,0x0,0x0,0x4,0x4,0x4,0x0,0x0,0x0,
0x0,0x0,0x0,0x4,0x0,0x4,0x4,0x0,0x0,0x0,0x0,0x0,0x0,0x4,0x0,0x0,0x4,0x0,0x0,0x4,0x0,0x4,0
x4,0x4,0x4,0x0,0x4,0x4,0x4,0x4,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x
0,0x0,0x0,0x4,0x0,0x0,0x0,0x4,0x0,0x4,0x0,0x4,0x0,};
   }
  final private JJCalls[] jj_2_rtns = new JJCalls[37];
  private boolean jj_rescan = false;
  private int jj_gc = 0;

  /** Constructor with InputStream. */
  public CParser(java.io.InputStream stream) {
     this(stream, null);
  }
  /** Constructor with InputStream and supplied encoding */
  public CParser(java.io.InputStream stream, String encoding) {
    try { jj_input_stream = new SimpleCharStream(stream, encoding, 1, 1); }
catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
    token_source = new CParserTokenManager(jj_input_stream);
    token = new Token();
    jj_ntk = -1;
    jj_gen = 0;
    for (int i = 0; i < 79; i++) jj_la1[i] = -1;
    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
  }

  /** Reinitialise. */
  public void ReInit(java.io.InputStream stream) {
     ReInit(stream, null);
  }
  /** Reinitialise. */
  public void ReInit(java.io.InputStream stream, String encoding) {
    try { jj_input_stream.ReInit(stream, encoding, 1, 1); }
catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
    token_source.ReInit(jj_input_stream);
    token = new Token();
    jj_ntk = -1;
    jjtree.reset();
    jj_gen = 0;
    for (int i = 0; i < 79; i++) jj_la1[i] = -1;
    for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
  }

  /** Constructor. */
```

```
public CParser(java.io.Reader stream) {
  jj_input_stream = new SimpleCharStream(stream, 1, 1);
  token_source = new CParserTokenManager(jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 79; i++) jj_la1[i] = -1;
  for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
}

/** Reinitialise. */
public void ReInit(java.io.Reader stream) {
  jj_input_stream.ReInit(stream, 1, 1);
  token_source.ReInit(jj_input_stream);
  token = new Token();
  jj_ntk = -1;
  jjtree.reset();
  jj_gen = 0;
  for (int i = 0; i < 79; i++) jj_la1[i] = -1;
  for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
}

/** Constructor with generated Token Manager. */
public CParser(CParserTokenManager tm) {
  token_source = tm;
  token = new Token();
  jj_ntk = -1;
  jj_gen = 0;
  for (int i = 0; i < 79; i++) jj_la1[i] = -1;
  for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
}

/** Reinitialise. */
public void ReInit(CParserTokenManager tm) {
  token_source = tm;
  token = new Token();
  jj_ntk = -1;
  jjtree.reset();
  jj_gen = 0;
  for (int i = 0; i < 79; i++) jj_la1[i] = -1;
  for (int i = 0; i < jj_2_rtns.length; i++) jj_2_rtns[i] = new JJCalls();
}

private Token jj_consume_token(int kind) throws ParseException {
  Token oldToken;
  if ((oldToken = token).next != null) token = token.next;
  else token = token.next = token_source.getNextToken();
  jj_ntk = -1;
  if (token.kind == kind) {
    jj_gen++;
    if (++jj_gc > 100) {
      jj_gc = 0;
      for (int i = 0; i < jj_2_rtns.length; i++) {
        JJCalls c = jj_2_rtns[i];
        while (c != null) {
          if (c.gen < jj_gen) c.first = null;
          c = c.next;
        }
      }
    }
    return token;
  }
  token = oldToken;
  jj_kind = kind;
  throw generateParseException();
}

static private final class LookaheadSuccess extends java.lang.Error { }
final private LookaheadSuccess jj_ls = new LookaheadSuccess();
private boolean jj_scan_token(int kind) {
  if (jj_scanpos == jj_lastpos) {
```

```
      jj_la--;
      if (jj_scanpos.next == null) {
        jj_lastpos = jj_scanpos = jj_scanpos.next = token_source.getNextToken();
      } else {
        jj_lastpos = jj_scanpos = jj_scanpos.next;
      }
    } else {
      jj_scanpos = jj_scanpos.next;
    }
    if (jj_rescan) {
      int i = 0; Token tok = token;
      while (tok != null && tok != jj_scanpos) { i++; tok = tok.next; }
      if (tok != null) jj_add_error_token(kind, i);
    }
    if (jj_scanpos.kind != kind) return true;
    if (jj_la == 0 && jj_scanpos == jj_lastpos) throw jj_ls;
    return false;
  }


/** Get the next Token. */
  final public Token getNextToken() {
    if (token.next != null) token = token.next;
    else token = token.next = token_source.getNextToken();
    jj_ntk = -1;
    jj_gen++;
    return token;
  }

/** Get the specific Token. */
  final public Token getToken(int index) {
    Token t = jj_lookingAhead ? jj_scanpos : token;
    for (int i = 0; i < index; i++) {
      if (t.next != null) t = t.next;
      else t = t.next = token_source.getNextToken();
    }
    return t;
  }

  private int jj_ntk() {
    if ((jj_nt=token.next) == null)
      return (jj_ntk = (token.next=token_source.getNextToken()).kind);
    else
      return (jj_ntk = jj_nt.kind);
  }

  private java.util.List<int[]> jj_expentries = new java.util.ArrayList<int[]>();
  private int[] jj_expentry;
  private int jj_kind = -1;
  private int[] jj_lasttokens = new int[100];
  private int jj_endpos;

  private void jj_add_error_token(int kind, int pos) {
    if (pos >= 100) return;
    if (pos == jj_endpos + 1) {
      jj_lasttokens[jj_endpos++] = kind;
    } else if (jj_endpos != 0) {
      jj_expentry = new int[jj_endpos];
      for (int i = 0; i < jj_endpos; i++) {
        jj_expentry[i] = jj_lasttokens[i];
      }
      jj_entries_loop: for (java.util.Iterator<?> it = jj_expentries.iterator();
it.hasNext();) {
        int[] oldentry = (int[])(it.next());
        if (oldentry.length == jj_expentry.length) {
          for (int i = 0; i < jj_expentry.length; i++) {
            if (oldentry[i] != jj_expentry[i]) {
              continue jj_entries_loop;
            }
          }
          jj_expentries.add(jj_expentry);
```

```
        break jj_entries_loop;
      }
    }
    if (pos != 0) jj_lasttokens[(jj_endpos = pos) - 1] = kind;
  }
}

/** Generate ParseException. */
public ParseException generateParseException() {
  jj_expentries.clear();
  boolean[] la1tokens = new boolean[119];
  if (jj_kind >= 0) {
    la1tokens[jj_kind] = true;
    jj_kind = -1;
  }
  for (int i = 0; i < 79; i++) {
    if (jj_la1[i] == jj_gen) {
      for (int j = 0; j < 32; j++) {
        if ((jj_la1_0[i] & (1<<j)) != 0) {
          la1tokens[j] = true;
        }
        if ((jj_la1_1[i] & (1<<j)) != 0) {
          la1tokens[32+j] = true;
        }
        if ((jj_la1_2[i] & (1<<j)) != 0) {
          la1tokens[64+j] = true;
        }
        if ((jj_la1_3[i] & (1<<j)) != 0) {
          la1tokens[96+j] = true;
        }
      }
    }
  }
  for (int i = 0; i < 119; i++) {
    if (la1tokens[i]) {
      jj_expentry = new int[1];
      jj_expentry[0] = i;
      jj_expentries.add(jj_expentry);
    }
  }
  jj_endpos = 0;
  jj_rescan_token();
  jj_add_error_token(0, 0);
  int[][] exptokseq = new int[jj_expentries.size()][];
  for (int i = 0; i < jj_expentries.size(); i++) {
    exptokseq[i] = jj_expentries.get(i);
  }
  return new ParseException(token, exptokseq, tokenImage);
}

/** Enable tracing. */
final public void enable_tracing() {
}

/** Disable tracing. */
final public void disable_tracing() {
}

private void jj_rescan_token() {
  jj_rescan = true;
  for (int i = 0; i < 37; i++) {
  try {
    JJCalls p = jj_2_rtns[i];
    do {
      if (p.gen > jj_gen) {
        jj_la = p.arg; jj_lastpos = jj_scanpos = p.first;
        switch (i) {
          case 0: jj_3_1(); break;
          case 1: jj_3_2(); break;
          case 2: jj_3_3(); break;
          case 3: jj_3_4(); break;
```

```
            case 4: jj_3_5(); break;
            case 5: jj_3_6(); break;
            case 6: jj_3_7(); break;
            case 7: jj_3_8(); break;
            case 8: jj_3_9(); break;
            case 9: jj_3_10(); break;
            case 10: jj_3_11(); break;
            case 11: jj_3_12(); break;
            case 12: jj_3_13(); break;
            case 13: jj_3_14(); break;
            case 14: jj_3_15(); break;
            case 15: jj_3_16(); break;
            case 16: jj_3_17(); break;
            case 17: jj_3_18(); break;
            case 18: jj_3_19(); break;
            case 19: jj_3_20(); break;
            case 20: jj_3_21(); break;
            case 21: jj_3_22(); break;
            case 22: jj_3_23(); break;
            case 23: jj_3_24(); break;
            case 24: jj_3_25(); break;
            case 25: jj_3_26(); break;
            case 26: jj_3_27(); break;
            case 27: jj_3_28(); break;
            case 28: jj_3_29(); break;
            case 29: jj_3_30(); break;
            case 30: jj_3_31(); break;
            case 31: jj_3_32(); break;
            case 32: jj_3_33(); break;
            case 33: jj_3_34(); break;
            case 34: jj_3_35(); break;
            case 35: jj_3_36(); break;
            case 36: jj_3_37(); break;
          }
        }
        p = p.next;
      } while (p != null);
      } catch(LookaheadSuccess ls) { }
    }
    jj_rescan = false;
  }

  private void jj_save(int index, int xla) {
    JJCalls p = jj_2_rtns[index];
    while (p.gen > jj_gen) {
      if (p.next == null) { p = p.next = new JJCalls(); break; }
      p = p.next;
    }
    p.gen = jj_gen + xla - jj_la; p.first = token; p.arg = xla;
  }

  static final class JJCalls {
    int gen;
    Token first;
    int arg;
    JJCalls next;
  }

}

/* Generated By:JJTree&JavaCC: Do not edit this line. CParserConstants.java */
package nql.cd.frontend;


/**
 * Token literal values and constants.
 * Generated by org.javacc.parser.OtherFilesGen#start()
 */
public interface CParserConstants {

  /** End of File. */
```

```
int EOF = 0;
/** RegularExpression Id. */
int PREPROCESSOR_START = 1;
/** RegularExpression Id. */
int PREPROCESSOR_DATA = 2;
/** RegularExpression Id. */
int PREPROCESSOR_END = 3;
/** RegularExpression Id. */
int BLOCK_START = 4;
/** RegularExpression Id. */
int INSIDE_COMMENTS = 5;
/** RegularExpression Id. */
int BLOCK_END = 6;
/** RegularExpression Id. */
int LINE_COMMENT = 7;
/** RegularExpression Id. */
int SPACE = 8;
/** RegularExpression Id. */
int EOL = 9;
/** RegularExpression Id. */
int INTEGER = 10;
/** RegularExpression Id. */
int REAL = 11;
/** RegularExpression Id. */
int CHARACTER = 12;
/** RegularExpression Id. */
int STRING = 13;
/** RegularExpression Id. */
int AUTO = 14;
/** RegularExpression Id. */
int REGISTER = 15;
/** RegularExpression Id. */
int EXTERN = 16;
/** RegularExpression Id. */
int STRUCT = 17;
/** RegularExpression Id. */
int UNION = 18;
/** RegularExpression Id. */
int STATIC = 19;
/** RegularExpression Id. */
int TYPEDEF = 20;
/** RegularExpression Id. */
int ENUM = 21;
/** RegularExpression Id. */
int CONST = 22;
/** RegularExpression Id. */
int VOLATILE = 23;
/** RegularExpression Id. */
int VOID = 24;
/** RegularExpression Id. */
int CHAR = 25;
/** RegularExpression Id. */
int SHORT = 26;
/** RegularExpression Id. */
int INT = 27;
/** RegularExpression Id. */
int LONG = 28;
/** RegularExpression Id. */
int FLOAT = 29;
/** RegularExpression Id. */
int DOUBLE = 30;
/** RegularExpression Id. */
int SIGNED = 31;
/** RegularExpression Id. */
int UNSIGNED = 32;
/** RegularExpression Id. */
int IF = 33;
/** RegularExpression Id. */
int ELSE = 34;
/** RegularExpression Id. */
int SWITCH = 35;
```

```
/** RegularExpression Id. */
int CASE = 36;
/** RegularExpression Id. */
int FOR = 37;
/** RegularExpression Id. */
int DO = 38;
/** RegularExpression Id. */
int WHILE = 39;
/** RegularExpression Id. */
int BREAK = 40;
/** RegularExpression Id. */
int CONTINUE = 41;
/** RegularExpression Id. */
int CDEFAULT = 42;
/** RegularExpression Id. */
int GOTO = 43;
/** RegularExpression Id. */
int SIZEOF = 44;
/** RegularExpression Id. */
int RETURN = 45;
/** RegularExpression Id. */
int PLUS_PLUS = 46;
/** RegularExpression Id. */
int MINUS_MINUS = 47;
/** RegularExpression Id. */
int POINTER_SELECT = 48;
/** RegularExpression Id. */
int DOT = 49;
/** RegularExpression Id. */
int LEFT_PAREN = 50;
/** RegularExpression Id. */
int RIGHT_PAREN = 51;
/** RegularExpression Id. */
int LEFT_BRACKET = 52;
/** RegularExpression Id. */
int RIGHT_BRACKET = 53;
/** RegularExpression Id. */
int LEFT_BRACE = 54;
/** RegularExpression Id. */
int RIGHT_BRACE = 55;
/** RegularExpression Id. */
int AMPERSAND = 56;
/** RegularExpression Id. */
int STAR = 57;
/** RegularExpression Id. */
int PLUS = 58;
/** RegularExpression Id. */
int MINUS = 59;
/** RegularExpression Id. */
int BITWISE_NOT = 60;
/** RegularExpression Id. */
int LOGICAL_NOT = 61;
/** RegularExpression Id. */
int SLASH = 62;
/** RegularExpression Id. */
int PERCENT = 63;
/** RegularExpression Id. */
int SHIFT_LEFT = 64;
/** RegularExpression Id. */
int SHIFT_RIGHT = 65;
/** RegularExpression Id. */
int LESS_THAN = 66;
/** RegularExpression Id. */
int GREATER_THAN = 67;
/** RegularExpression Id. */
int LESS_EQUALS = 68;
/** RegularExpression Id. */
int GREATER_EQUALS = 69;
/** RegularExpression Id. */
int EQUALS_EQUALS = 70;
/** RegularExpression Id. */
```

```
int NOT_EQUALS = 71;
/** RegularExpression Id. */
int BITWISE_XOR = 72;
/** RegularExpression Id. */
int BITWISE_OR = 73;
/** RegularExpression Id. */
int LOGICAL_AND = 74;
/** RegularExpression Id. */
int LOGICAL_OR = 75;
/** RegularExpression Id. */
int QUESTION_MARK = 76;
/** RegularExpression Id. */
int COLON = 77;
/** RegularExpression Id. */
int EQUALS = 78;
/** RegularExpression Id. */
int STAR_EQUALS = 79;
/** RegularExpression Id. */
int SLASH_EQUALS = 80;
/** RegularExpression Id. */
int PERCENT_EQUALS = 81;
/** RegularExpression Id. */
int PLUS_EQUALS = 82;
/** RegularExpression Id. */
int MINUS_EQUALS = 83;
/** RegularExpression Id. */
int LESS_LESS_EQUALS = 84;
/** RegularExpression Id. */
int GREATER_GREATER_EQUALS = 85;
/** RegularExpression Id. */
int BITWISE_OR_EQUALS = 86;
/** RegularExpression Id. */
int AMP_EQUALS = 87;
/** RegularExpression Id. */
int BITWISE_XOR_EQUALS = 88;
/** RegularExpression Id. */
int SEMICOLON = 89;
/** RegularExpression Id. */
int COMMA = 90;
/** RegularExpression Id. */
int SINGLE_QUOTE = 91;
/** RegularExpression Id. */
int DOUBLE_QUOTE = 92;
/** RegularExpression Id. */
int AT_SIGN = 93;
/** RegularExpression Id. */
int THREE_DOTS = 94;
/** RegularExpression Id. */
int SLASH_SLASH = 95;
/** RegularExpression Id. */
int SLASH_STAR = 96;
/** RegularExpression Id. */
int STAR_SLASH = 97;
/** RegularExpression Id. */
int IDENTIFIER = 98;
/** RegularExpression Id. */
int PERIOD = 99;
/** RegularExpression Id. */
int ANYCHAR = 100;
/** RegularExpression Id. */
int ANYCHAR_NOT_EOL = 101;
/** RegularExpression Id. */
int ANYCHAR_NOT_ESCAPE = 102;
/** RegularExpression Id. */
int ESCAPE_SEQUENCE = 103;
/** RegularExpression Id. */
int LETTER = 104;
/** RegularExpression Id. */
int DIGIT = 105;
/** RegularExpression Id. */
int DIGIT_NOT_ZERO = 106;
```

```
/** RegularExpression Id. */
int OCTAL = 107;
/** RegularExpression Id. */
int DEC = 108;
/** RegularExpression Id. */
int HEX = 109;
/** RegularExpression Id. */
int INTEGER_SUFFIX = 110;
/** RegularExpression Id. */
int REAL_SUFFIX = 111;
/** RegularExpression Id. */
int WHOLE = 112;
/** RegularExpression Id. */
int DPOINT = 113;
/** RegularExpression Id. */
int FRACTION = 114;
/** RegularExpression Id. */
int EXPONENT = 115;
/** RegularExpression Id. */
int BLOCK_COMMENT_START = 116;
/** RegularExpression Id. */
int BLOCK_COMMENT_END = 117;
/** RegularExpression Id. */
int DOUBLE_SLASHES = 118;

/** Lexical state. */
int DEFAULT = 0;
/** Lexical state. */
int INSIDE_PREPROCESSOR = 1;
/** Lexical state. */
int INSIDE_BLOCK_COMMENT = 2;

/** Literal token values. */
String[] tokenImage = {
  "<EOF>",
  "\"#\"",
  "<PREPROCESSOR_DATA>",
  "<PREPROCESSOR_END>",
  "<BLOCK_START>",
  "<INSIDE_COMMENTS>",
  "<BLOCK_END>",
  "<LINE_COMMENT>",
  "<SPACE>",
  "<EOL>",
  "<INTEGER>",
  "<REAL>",
  "<CHARACTER>",
  "<STRING>",
  "\"auto\"",
  "\"register\"",
  "\"extern\"",
  "\"struct\"",
  "\"union\"",
  "\"static\"",
  "\"typedef\"",
  "\"enum\"",
  "\"const\"",
  "\"volatile\"",
  "\"void\"",
  "\"char\"",
  "\"short\"",
  "\"int\"",
  "\"long\"",
  "\"float\"",
  "\"double\"",
  "\"signed\"",
  "\"unsigned\"",
  "\"if\"",
  "\"else\"",
  "\"switch\"",
  "\"case\"",
```

```
"\"for\"",
"\"do\"",
"\"while\"",
"\"break\"",
"\"continue\"",
"\"default\"",
"\"goto\"",
"\"sizeof\"",
"\"return\"",
"\"++\"",
"\"--\"",
"\"->\"",
"<DOT>",
"\"(\"",
"\")\"",
"\"[\"",
"\"]\"",
"\"{\"",
"\"}\"",
"\"&\"",
"\"*\"",
"\"+\"",
"\"-\"",
"\"~\"",
"\"!\"",
"\"/\"",
"\"%\"",
"\"<<\"",
"\">>\"",
"\"<\"",
"\">\"",
"\"<=\"",
"\">=\"",
"\"==\"",
"\"!=\"",
"\"^\"",
"\"|\"",
"\"&&\"",
"\"||\"",
"\"?\"",
"\":\"",
"\"=\"",
"\"*=\"",
"\"/=\"",
"\"%=\"",
"\"+=\"",
"\"-=\"",
"\"<<=\"",
"\">>=\"",
"\"|=\"",
"\"&=\"",
"\"^=\"",
"\";\"",
"\",\"",
"\"\\\'\"",
"\"\\\"\"",
"\"@\"",
"\"...\"",
"<SLASH_SLASH>",
"<SLASH_STAR>",
"<STAR_SLASH>",
"<IDENTIFIER>",
"\".\"",
"<ANYCHAR>",
"<ANYCHAR_NOT_EOL>",
"<ANYCHAR_NOT_ESCAPE>",
"<ESCAPE_SEQUENCE>",
"<LETTER>",
"<DIGIT>",
"<DIGIT_NOT_ZERO>",
"<OCTAL>",
```

```
    "<DEC>",
    "<HEX>",
    "<INTEGER_SUFFIX>",
    "<REAL_SUFFIX>",
    "<WHOLE>",
    "<DPOINT>",
    "<FRACTION>",
    "<EXPONENT>",
    "\"/*\"",
    "\"*/\"",
    "\"//\"",
  };

}

/* Generated By:JJTree&JavaCC: Do not edit this line. CParserTokenManager.java */
package nql.cd.frontend;
import nql.cd.intermediate.*;
import java.util.*;
import java.io.*;

/** Token Manager. */
public class CParserTokenManager implements CParserConstants
{
    /* This main entry is a test driver for token specifications. */
    public static void main(String [] args) {
        try {
            java.io.BufferedReader cSourceFile = new java.io.BufferedReader(new
java.io.FileReader(args[0]));
            SimpleCharStream scs = new SimpleCharStream(cSourceFile);
            CParserTokenManager mgr = new CParserTokenManager(scs);
            java.lang.reflect.Field[] tokens =
CParserConstants.class.getDeclaredFields();
            Token t = mgr.getNextToken();
            while ( t.kind != EOF ) {
                String tokenName = tokens[t.kind].getName();

                // STATIC = true
                // debugStream.println(tokenName + ": " + t.image);

                // STATIC = false
                System.out.println(tokenName + ": " + t.image);

                t = mgr.getNextToken();
            }
        }
        catch ( Exception ex ) {
            ex.printStackTrace();
        }
    }

  /** Debug output. */
  public  java.io.PrintStream debugStream = System.out;
  /** Set debug output. */
  public  void setDebugStream(java.io.PrintStream ds) { debugStream = ds; }
private final int jjStopStringLiteralDfa_0(int pos, long active0, long active1)
{
   switch (pos)
   {
      case 0:
         if ((active0 & 0x4000000000000000L) != 0L || (active1 & 0x10000L) != 0L)
            return 68;
         if ((active1 & 0x40000000L) != 0L)
         {
            jjmatchedKind = 49;
            return 74;
         }
         if ((active1 & 0x10000000L) != 0L)
            return 78;
         if ((active1 & 0x8000000L) != 0L)
            return 11;
```

```
   if ((active0 & 0x2000000000000000L) != 0L || (active1 & 0x8000L) != 0L)
      return 39;
   if ((active0 & 0x3ffffffc000L) != 0L)
   {
      jjmatchedKind = 98;
      return 42;
   }
   return -1;
case 1:
   if ((active0 & 0x3fbdbfffc000L) != 0L)
   {
      if (jjmatchedPos != 1)
      {
         jjmatchedKind = 98;
         jjmatchedPos = 1;
      }
      return 42;
   }
   if ((active1 & 0x40000000L) != 0L)
   {
      if (jjmatchedPos == 0)
      {
         jjmatchedKind = 49;
         jjmatchedPos = 0;
      }
      return -1;
   }
   if ((active0 & 0x4240000000L) != 0L)
      return 42;
   return -1;
case 2:
   if ((active0 & 0x3f9df7ffc000L) != 0L)
   {
      jjmatchedKind = 98;
      jjmatchedPos = 2;
      return 42;
   }
   if ((active1 & 0x40000000L) != 0L)
   {
      if (jjmatchedPos == 0)
      {
         jjmatchedKind = 49;
         jjmatchedPos = 0;
      }
      return -1;
   }
   if ((active0 & 0x2008000000L) != 0L)
      return 42;
   return -1;
case 3:
   if ((active0 & 0x3789e4df8000L) != 0L)
   {
      jjmatchedKind = 98;
      jjmatchedPos = 3;
      return 42;
   }
   if ((active0 & 0x81413204000L) != 0L)
      return 42;
   return -1;
case 4:
   if ((active0 & 0x3609c09b8000L) != 0L)
   {
      jjmatchedKind = 98;
      jjmatchedPos = 4;
      return 42;
   }
   if ((active0 & 0x18024440000L) != 0L)
      return 42;
   return -1;
case 5:
   if ((active0 & 0x60100908000L) != 0L)
```

```
         {
            jjmatchedKind = 98;
            jjmatchedPos = 5;
            return 42;
         }
         if ((active0 & 0x3008c00b0000L) != 0L)
            return 42;
         return -1;
      case 6:
         if ((active0 & 0x20100808000L) != 0L)
         {
            jjmatchedKind = 98;
            jjmatchedPos = 6;
            return 42;
         }
         if ((active0 & 0x40000100000L) != 0L)
            return 42;
         return -1;
      default :
         return -1;
   }
}
private final int jjStartNfa_0(int pos, long active0, long active1)
{
   return jjMoveNfa_0(jjStopStringLiteralDfa_0(pos, active0, active1), pos + 1);
}
private int jjStopAtPos(int pos, int kind)
{
   jjmatchedKind = kind;
   jjmatchedPos = pos;
   return pos + 1;
}
private int jjMoveStringLiteralDfa0_0()
{
   switch(curChar)
   {
      case 33:
         jjmatchedKind = 61;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x80L);
      case 34:
         return jjStartNfaWithStates_0(0, 92, 78);
      case 35:
         return jjStopAtPos(0, 1);
      case 37:
         jjmatchedKind = 63;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x20000L);
      case 38:
         jjmatchedKind = 56;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x800400L);
      case 39:
         return jjStartNfaWithStates_0(0, 91, 11);
      case 40:
         return jjStopAtPos(0, 50);
      case 41:
         return jjStopAtPos(0, 51);
      case 42:
         jjmatchedKind = 57;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x8000L);
      case 43:
         jjmatchedKind = 58;
         return jjMoveStringLiteralDfa1_0(0x4000000000000L, 0x40000L);
      case 44:
         return jjStopAtPos(0, 90);
      case 45:
         jjmatchedKind = 59;
         return jjMoveStringLiteralDfa1_0(0x1800000000000L, 0x80000L);
      case 46:
         return jjMoveStringLiteralDfa1_0(0x0L, 0x40000000L);
      case 47:
         jjmatchedKind = 62;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x10000L);
```

```
      case 58:
         return jjStopAtPos(0, 77);
      case 59:
         return jjStopAtPos(0, 89);
      case 60:
         jjmatchedKind = 66;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x100011L);
      case 61:
         jjmatchedKind = 78;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x40L);
      case 62:
         jjmatchedKind = 67;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x200022L);
      case 63:
         return jjStopAtPos(0, 76);
      case 64:
         return jjStopAtPos(0, 93);
      case 91:
         return jjStopAtPos(0, 52);
      case 93:
         return jjStopAtPos(0, 53);
      case 94:
         jjmatchedKind = 72;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x1000000L);
      case 97:
         return jjMoveStringLiteralDfa1_0(0x4000L, 0x0L);
      case 98:
         return jjMoveStringLiteralDfa1_0(0x10000000000L, 0x0L);
      case 99:
         return jjMoveStringLiteralDfa1_0(0x21002400000L, 0x0L);
      case 100:
         return jjMoveStringLiteralDfa1_0(0x44040000000L, 0x0L);
      case 101:
         return jjMoveStringLiteralDfa1_0(0x400210000L, 0x0L);
      case 102:
         return jjMoveStringLiteralDfa1_0(0x2020000000L, 0x0L);
      case 103:
         return jjMoveStringLiteralDfa1_0(0x800000000000L, 0x0L);
      case 105:
         return jjMoveStringLiteralDfa1_0(0x208000000L, 0x0L);
      case 108:
         return jjMoveStringLiteralDfa1_0(0x10000000L, 0x0L);
      case 114:
         return jjMoveStringLiteralDfa1_0(0x200000008000L, 0x0L);
      case 115:
         return jjMoveStringLiteralDfa1_0(0x1008840a0000L, 0x0L);
      case 116:
         return jjMoveStringLiteralDfa1_0(0x100000L, 0x0L);
      case 117:
         return jjMoveStringLiteralDfa1_0(0x100040000L, 0x0L);
      case 118:
         return jjMoveStringLiteralDfa1_0(0x1800000L, 0x0L);
      case 119:
         return jjMoveStringLiteralDfa1_0(0x8000000000L, 0x0L);
      case 123:
         return jjStopAtPos(0, 54);
      case 124:
         jjmatchedKind = 73;
         return jjMoveStringLiteralDfa1_0(0x0L, 0x400800L);
      case 125:
         return jjStopAtPos(0, 55);
      case 126:
         return jjStopAtPos(0, 60);
      default :
         return jjMoveNfa_0(0, 0);
   }
}
private int jjMoveStringLiteralDfa1_0(long active0, long active1)
{
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) {
```

```
   jjStopStringLiteralDfa_0(0, active0, active1);
   return 1;
}
switch(curChar)
{
   case 38:
      if ((active1 & 0x400L) != 0L)
         return jjStopAtPos(1, 74);
      break;
   case 43:
      if ((active0 & 0x4000000000000L) != 0L)
         return jjStopAtPos(1, 46);
      break;
   case 45:
      if ((active0 & 0x8000000000000L) != 0L)
         return jjStopAtPos(1, 47);
      break;
   case 46:
      return jjMoveStringLiteralDfa2_0(active0, 0L, active1, 0x40000000L);
   case 60:
      if ((active1 & 0x1L) != 0L)
      {
         jjmatchedKind = 64;
         jjmatchedPos = 1;
      }
      return jjMoveStringLiteralDfa2_0(active0, 0L, active1, 0x100000L);
   case 61:
      if ((active1 & 0x10L) != 0L)
         return jjStopAtPos(1, 68);
      else if ((active1 & 0x20L) != 0L)
         return jjStopAtPos(1, 69);
      else if ((active1 & 0x40L) != 0L)
         return jjStopAtPos(1, 70);
      else if ((active1 & 0x80L) != 0L)
         return jjStopAtPos(1, 71);
      else if ((active1 & 0x8000L) != 0L)
         return jjStopAtPos(1, 79);
      else if ((active1 & 0x10000L) != 0L)
         return jjStopAtPos(1, 80);
      else if ((active1 & 0x20000L) != 0L)
         return jjStopAtPos(1, 81);
      else if ((active1 & 0x40000L) != 0L)
         return jjStopAtPos(1, 82);
      else if ((active1 & 0x80000L) != 0L)
         return jjStopAtPos(1, 83);
      else if ((active1 & 0x400000L) != 0L)
         return jjStopAtPos(1, 86);
      else if ((active1 & 0x800000L) != 0L)
         return jjStopAtPos(1, 87);
      else if ((active1 & 0x1000000L) != 0L)
         return jjStopAtPos(1, 88);
      break;
   case 62:
      if ((active0 & 0x1000000000000L) != 0L)
         return jjStopAtPos(1, 48);
      else if ((active1 & 0x2L) != 0L)
      {
         jjmatchedKind = 65;
         jjmatchedPos = 1;
      }
      return jjMoveStringLiteralDfa2_0(active0, 0L, active1, 0x200000L);
   case 97:
      return jjMoveStringLiteralDfa2_0(active0, 0x1000000000L, active1, 0L);
   case 101:
      return jjMoveStringLiteralDfa2_0(active0, 0x240000008000L, active1, 0L);
   case 102:
      if ((active0 & 0x200000000L) != 0L)
         return jjStartNfaWithStates_0(1, 33, 42);
      break;
   case 104:
      return jjMoveStringLiteralDfa2_0(active0, 0x8006000000L, active1, 0L);
```

```
      case 105:
         return jjMoveStringLiteralDfa2_0(active0, 0x100080000000L, active1, 0L);
      case 108:
         return jjMoveStringLiteralDfa2_0(active0, 0x420000000L, active1, 0L);
      case 110:
         return jjMoveStringLiteralDfa2_0(active0, 0x108240000L, active1, 0L);
      case 111:
         if ((active0 & 0x4000000000L) != 0L)
         {
            jjmatchedKind = 38;
            jjmatchedPos = 1;
         }
         return jjMoveStringLiteralDfa2_0(active0, 0xa2051c00000L, active1, 0L);
      case 114:
         return jjMoveStringLiteralDfa2_0(active0, 0x10000000000L, active1, 0L);
      case 116:
         return jjMoveStringLiteralDfa2_0(active0, 0xa0000L, active1, 0L);
      case 117:
         return jjMoveStringLiteralDfa2_0(active0, 0x4000L, active1, 0L);
      case 119:
         return jjMoveStringLiteralDfa2_0(active0, 0x800000000L, active1, 0L);
      case 120:
         return jjMoveStringLiteralDfa2_0(active0, 0x10000L, active1, 0L);
      case 121:
         return jjMoveStringLiteralDfa2_0(active0, 0x100000L, active1, 0L);
      case 124:
         if ((active1 & 0x800L) != 0L)
            return jjStopAtPos(1, 75);
         break;
      default :
         break;
   }
   return jjStartNfa_0(0, active0, active1);
}
private int jjMoveStringLiteralDfa2_0(long old0, long active0, long old1, long active1)
{
   if (((active0 &= old0) | (active1 &= old1)) == 0L)
      return jjStartNfa_0(0, old0, old1);
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) {
      jjStopStringLiteralDfa_0(1, active0, active1);
      return 2;
   }
   switch(curChar)
   {
      case 46:
         if ((active1 & 0x40000000L) != 0L)
            return jjStopAtPos(2, 94);
         break;
      case 61:
         if ((active1 & 0x100000L) != 0L)
            return jjStopAtPos(2, 84);
         else if ((active1 & 0x200000L) != 0L)
            return jjStopAtPos(2, 85);
         break;
      case 97:
         return jjMoveStringLiteralDfa3_0(active0, 0x2080000L, active1, 0L);
      case 101:
         return jjMoveStringLiteralDfa3_0(active0, 0x10000000000L, active1, 0L);
      case 102:
         return jjMoveStringLiteralDfa3_0(active0, 0x40000000000L, active1, 0L);
      case 103:
         return jjMoveStringLiteralDfa3_0(active0, 0x80008000L, active1, 0L);
      case 105:
         return jjMoveStringLiteralDfa3_0(active0, 0x8801040000L, active1, 0L);
      case 108:
         return jjMoveStringLiteralDfa3_0(active0, 0x800000L, active1, 0L);
      case 110:
         return jjMoveStringLiteralDfa3_0(active0, 0x20010400000L, active1, 0L);
      case 111:
         return jjMoveStringLiteralDfa3_0(active0, 0x24000000L, active1, 0L);
```

```
      case 112:
         return jjMoveStringLiteralDfa3_0(active0, 0x100000L, active1, 0L);
      case 114:
         if ((active0 & 0x2000000000L) != 0L)
            return jjStartNfaWithStates_0(2, 37, 42);
         return jjMoveStringLiteralDfa3_0(active0, 0x20000L, active1, 0L);
      case 115:
         return jjMoveStringLiteralDfa3_0(active0, 0x1500000000L, active1, 0L);
      case 116:
         if ((active0 & 0x8000000L) != 0L)
            return jjStartNfaWithStates_0(2, 27, 42);
         return jjMoveStringLiteralDfa3_0(active0, 0x280000014000L, active1, 0L);
      case 117:
         return jjMoveStringLiteralDfa3_0(active0, 0x40200000L, active1, 0L);
      case 122:
         return jjMoveStringLiteralDfa3_0(active0, 0x100000000000L, active1, 0L);
      default :
         break;
   }
   return jjStartNfa_0(1, active0, active1);
}
private int jjMoveStringLiteralDfa3_0(long old0, long active0, long old1, long active1)
{
   if (((active0 &= old0) | (active1 &= old1)) == 0L)
      return jjStartNfa_0(1, old0, old1);
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) {
      jjStopStringLiteralDfa_0(2, active0, 0L);
      return 3;
   }
   switch(curChar)
   {
      case 97:
         return jjMoveStringLiteralDfa4_0(active0, 0x500208000000L);
      case 98:
         return jjMoveStringLiteralDfa4_0(active0, 0x40000000L);
      case 100:
         if ((active0 & 0x1000000L) != 0L)
            return jjStartNfaWithStates_0(3, 24, 42);
         break;
      case 101:
         if ((active0 & 0x400000000L) != 0L)
            return jjStartNfaWithStates_0(3, 34, 42);
         else if ((active0 & 0x1000000000L) != 0L)
            return jjStartNfaWithStates_0(3, 36, 42);
         return jjMoveStringLiteralDfa4_0(active0, 0x100000110000L);
      case 103:
         if ((active0 & 0x10000000L) != 0L)
            return jjStartNfaWithStates_0(3, 28, 42);
         break;
      case 105:
         return jjMoveStringLiteralDfa4_0(active0, 0x100008000L);
      case 108:
         return jjMoveStringLiteralDfa4_0(active0, 0x8000000000L);
      case 109:
         if ((active0 & 0x200000L) != 0L)
            return jjStartNfaWithStates_0(3, 21, 42);
         break;
      case 110:
         return jjMoveStringLiteralDfa4_0(active0, 0x80000000L);
      case 111:
         if ((active0 & 0x4000L) != 0L)
            return jjStartNfaWithStates_0(3, 14, 42);
         else if ((active0 & 0x800000000000L) != 0L)
            return jjStartNfaWithStates_0(3, 43, 42);
         return jjMoveStringLiteralDfa4_0(active0, 0x40000L);
      case 114:
         if ((active0 & 0x2000000L) != 0L)
            return jjStartNfaWithStates_0(3, 25, 42);
         return jjMoveStringLiteralDfa4_0(active0, 0x4000000L);
      case 115:
```

```
         return jjMoveStringLiteralDfa4_0(active0, 0x400000L);
      case 116:
         return jjMoveStringLiteralDfa4_0(active0, 0x208000800000L);
      case 117:
         return jjMoveStringLiteralDfa4_0(active0, 0x2000000020000L);
      default :
         break;
   }
   return jjStartNfa_0(2, active0, 0L);
}
private int jjMoveStringLiteralDfa4_0(long old0, long active0)
{
   if (((active0 &= old0)) == 0L)
      return jjStartNfa_0(2, old0, 0L);
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) {
      jjStopStringLiteralDfa_0(3, active0, 0L);
      return 4;
   }
   switch(curChar)
   {
      case 99:
         return jjMoveStringLiteralDfa5_0(active0, 0x800020000L);
      case 100:
         return jjMoveStringLiteralDfa5_0(active0, 0x100000L);
      case 101:
         if ((active0 & 0x8000000000L) != 0L)
            return jjStartNfaWithStates_0(4, 39, 42);
         return jjMoveStringLiteralDfa5_0(active0, 0x80000000L);
      case 103:
         return jjMoveStringLiteralDfa5_0(active0, 0x100000000L);
      case 105:
         return jjMoveStringLiteralDfa5_0(active0, 0x20000080000L);
      case 107:
         if ((active0 & 0x10000000000L) != 0L)
            return jjStartNfaWithStates_0(4, 40, 42);
         break;
      case 108:
         return jjMoveStringLiteralDfa5_0(active0, 0x40000000L);
      case 110:
         if ((active0 & 0x40000L) != 0L)
            return jjStartNfaWithStates_0(4, 18, 42);
         break;
      case 111:
         return jjMoveStringLiteralDfa5_0(active0, 0x100000000000L);
      case 114:
         return jjMoveStringLiteralDfa5_0(active0, 0x200000010000L);
      case 115:
         return jjMoveStringLiteralDfa5_0(active0, 0x8000L);
      case 116:
         if ((active0 & 0x400000L) != 0L)
            return jjStartNfaWithStates_0(4, 22, 42);
         else if ((active0 & 0x4000000L) != 0L)
            return jjStartNfaWithStates_0(4, 26, 42);
         else if ((active0 & 0x20000000L) != 0L)
            return jjStartNfaWithStates_0(4, 29, 42);
         return jjMoveStringLiteralDfa5_0(active0, 0x800000L);
      case 117:
         return jjMoveStringLiteralDfa5_0(active0, 0x40000000000L);
      default :
         break;
   }
   return jjStartNfa_0(3, active0, 0L);
}
private int jjMoveStringLiteralDfa5_0(long old0, long active0)
{
   if (((active0 &= old0)) == 0L)
      return jjStartNfa_0(3, old0, 0L);
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) {
      jjStopStringLiteralDfa_0(4, active0, 0L);
```

```
      return 5;
   }
   switch(curChar)
   {
      case 99:
         if ((active0 & 0x80000L) != 0L)
            return jjStartNfaWithStates_0(5, 19, 42);
         break;
      case 100:
         if ((active0 & 0x80000000L) != 0L)
            return jjStartNfaWithStates_0(5, 31, 42);
         break;
      case 101:
         if ((active0 & 0x40000000L) != 0L)
            return jjStartNfaWithStates_0(5, 30, 42);
         return jjMoveStringLiteralDfa6_0(active0, 0x100000L);
      case 102:
         if ((active0 & 0x1000000000000L) != 0L)
            return jjStartNfaWithStates_0(5, 44, 42);
         break;
      case 104:
         if ((active0 & 0x800000000L) != 0L)
            return jjStartNfaWithStates_0(5, 35, 42);
         break;
      case 105:
         return jjMoveStringLiteralDfa6_0(active0, 0x800000L);
      case 108:
         return jjMoveStringLiteralDfa6_0(active0, 0x400000000000L);
      case 110:
         if ((active0 & 0x10000L) != 0L)
            return jjStartNfaWithStates_0(5, 16, 42);
         else if ((active0 & 0x2000000000000L) != 0L)
            return jjStartNfaWithStates_0(5, 45, 42);
         return jjMoveStringLiteralDfa6_0(active0, 0x201000000000L);
      case 116:
         if ((active0 & 0x20000L) != 0L)
            return jjStartNfaWithStates_0(5, 17, 42);
         return jjMoveStringLiteralDfa6_0(active0, 0x8000L);
      default :
         break;
   }
   return jjStartNfa_0(4, active0, 0L);
}
private int jjMoveStringLiteralDfa6_0(long old0, long active0)
{
   if (((active0 &= old0)) == 0L)
      return jjStartNfa_0(4, old0, 0L);
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) {
      jjStopStringLiteralDfa_0(5, active0, 0L);
      return 6;
   }
   switch(curChar)
   {
      case 101:
         return jjMoveStringLiteralDfa7_0(active0, 0x100008000L);
      case 102:
         if ((active0 & 0x100000L) != 0L)
            return jjStartNfaWithStates_0(6, 20, 42);
         break;
      case 108:
         return jjMoveStringLiteralDfa7_0(active0, 0x800000L);
      case 116:
         if ((active0 & 0x40000000000L) != 0L)
            return jjStartNfaWithStates_0(6, 42, 42);
         break;
      case 117:
         return jjMoveStringLiteralDfa7_0(active0, 0x200000000000L);
      default :
         break;
   }
```

```
   return jjStartNfa_0(5, active0, 0L);
}
private int jjMoveStringLiteralDfa7_0(long old0, long active0)
{
   if (((active0 &= old0)) == 0L)
      return jjStartNfa_0(5, old0, 0L);
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) {
      jjStopStringLiteralDfa_0(6, active0, 0L);
      return 7;
   }
   switch(curChar)
   {
      case 100:
         if ((active0 & 0x100000000L) != 0L)
            return jjStartNfaWithStates_0(7, 32, 42);
         break;
      case 101:
         if ((active0 & 0x800000L) != 0L)
            return jjStartNfaWithStates_0(7, 23, 42);
         else if ((active0 & 0x20000000000L) != 0L)
            return jjStartNfaWithStates_0(7, 41, 42);
         break;
      case 114:
         if ((active0 & 0x8000L) != 0L)
            return jjStartNfaWithStates_0(7, 15, 42);
         break;
      default :
         break;
   }
   return jjStartNfa_0(6, active0, 0L);
}
private int jjStartNfaWithStates_0(int pos, int kind, int state)
{
   jjmatchedKind = kind;
   jjmatchedPos = pos;
   try { curChar = input_stream.readChar(); }
   catch(java.io.IOException e) { return pos + 1; }
   return jjMoveNfa_0(state, pos + 1);
}
static final long[] jjbitVec0 = {
   0x0L, 0x0L, 0xffffffffffffffffL, 0xffffffffffffffffL
};
private int jjMoveNfa_0(int startState, int curPos)
{
   int startsAt = 0;
   jjnewStateCnt = 78;
   int i = 1;
   jjstateSet[0] = startState;
   int kind = 0x7fffffff;
   for (;;)
   {
      if (++jjround == 0x7fffffff)
         ReInitRounds();
      if (curChar < 64)
      {
         long l = 1L << curChar;
         do
         {
            switch(jjstateSet[--i])
            {
               case 68:
                  if (curChar == 42)
                  {
                     if (kind > 96)
                        kind = 96;
                  }
                  else if (curChar == 47)
                  {
                     if (kind > 95)
                        kind = 95;
```

```
      }
      if (curChar == 47)
      {
         if (kind > 7)
            kind = 7;
         jjCheckNAdd(70);
      }
      else if (curChar == 42)
      {
         if (kind > 4)
            kind = 4;
      }
      break;
   case 0:
      if ((0x3ff000000000000L & l) != 0L)
      {
         if (kind > 11)
            kind = 11;
         jjCheckNAddStates(0, 6);
      }
      else if ((0x2400L & l) != 0L)
      {
         if (kind > 9)
            kind = 9;
      }
      else if ((0x100000200L & l) != 0L)
      {
         if (kind > 8)
            kind = 8;
      }
      else if (curChar == 46)
      {
         if (kind > 49)
            kind = 49;
         jjCheckNAdd(74);
      }
      else if (curChar == 47)
         jjAddStates(7, 10);
      else if (curChar == 42)
         jjstateSet[jjnewStateCnt++] = 39;
      else if (curChar == 34)
         jjCheckNAddStates(11, 13);
      else if (curChar == 39)
         jjCheckNAddStates(14, 16);
      if ((0x3fe000000000000L & l) != 0L)
      {
         if (kind > 10)
            kind = 10;
         jjCheckNAddStates(17, 19);
      }
      else if (curChar == 48)
      {
         if (kind > 10)
            kind = 10;
         jjCheckNAddStates(20, 23);
      }
      else if (curChar == 13)
         jjstateSet[jjnewStateCnt++] = 2;
      break;
   case 78:
      if ((0xfffffff7bffffdbffL & l) != 0L)
         jjCheckNAddStates(11, 13);
      else if (curChar == 34)
      {
         if (kind > 13)
            kind = 13;
      }
      break;
   case 11:
      if ((0xfffffff7bffffdbffL & l) != 0L)
         jjCheckNAdd(12);
```

```
   else if (curChar == 39)
   {
      if (kind > 12)
         kind = 12;
   }
   break;
case 1:
   if ((0x2400L & l) != 0L && kind > 9)
      kind = 9;
   break;
case 2:
   if (curChar == 10 && kind > 9)
      kind = 9;
   break;
case 3:
   if (curChar == 13)
      jjstateSet[jjnewStateCnt++] = 2;
   break;
case 4:
   if ((0x3fe000000000000L & l) == 0L)
      break;
   if (kind > 10)
      kind = 10;
   jjCheckNAddStates(17, 19);
   break;
case 5:
   if ((0x3ff000000000000L & l) == 0L)
      break;
   if (kind > 10)
      kind = 10;
   jjCheckNAddStates(17, 19);
   break;
case 10:
   if (curChar == 39)
      jjCheckNAddStates(14, 16);
   break;
case 12:
   if (curChar == 39 && kind > 12)
      kind = 12;
   break;
case 14:
   if ((0x8001008400000000L & l) != 0L)
      jjCheckNAdd(12);
   break;
case 15:
   if ((0xff000000000000L & l) != 0L)
      jjCheckNAddTwoStates(12, 16);
   break;
case 16:
   if ((0xff000000000000L & l) != 0L)
      jjCheckNAddTwoStates(17, 12);
   break;
case 17:
   if ((0xff000000000000L & l) != 0L)
      jjCheckNAdd(12);
   break;
case 19:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAddTwoStates(12, 20);
   break;
case 20:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAddStates(24, 26);
   break;
case 21:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAdd(12);
   break;
case 22:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAddTwoStates(21, 12);
```

```
   break;
case 23:
   if (curChar == 34)
      jjCheckNAddStates(11, 13);
   break;
case 24:
   if ((0xffffff7bffffdbffL & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 25:
   if (curChar == 34 && kind > 13)
      kind = 13;
   break;
case 27:
   if ((0x8001008400000000L & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 28:
   if ((0xff000000000000L & l) != 0L)
      jjCheckNAddStates(27, 30);
   break;
case 29:
   if ((0xff000000000000L & l) != 0L)
      jjCheckNAddStates(31, 34);
   break;
case 30:
   if ((0xff000000000000L & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 32:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAddStates(35, 38);
   break;
case 33:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAddStates(39, 43);
   break;
case 34:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 35:
   if ((0x3ff000000000000L & l) != 0L)
      jjCheckNAddStates(44, 47);
   break;
case 36:
   if ((0x2400L & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 37:
   if (curChar == 10)
      jjCheckNAddStates(11, 13);
   break;
case 38:
   if (curChar == 13)
      jjstateSet[jjnewStateCnt++] = 37;
   break;
case 39:
   if (curChar == 47 && kind > 97)
      kind = 97;
   break;
case 40:
   if (curChar == 42)
      jjstateSet[jjnewStateCnt++] = 39;
   break;
case 42:
   if ((0x3ff000000000000L & l) == 0L)
      break;
   if (kind > 98)
      kind = 98;
   jjstateSet[jjnewStateCnt++] = 42;
```

```
   break;
case 43:
   if ((0x3ff0000000000000L & l) == 0L)
      break;
   if (kind > 11)
      kind = 11;
   jjCheckNAddStates(0, 6);
   break;
case 44:
   if ((0x3ff0000000000000L & l) != 0L)
      jjCheckNAddTwoStates(44, 45);
   break;
case 45:
   if (curChar != 46)
      break;
   if (kind > 11)
      kind = 11;
   jjCheckNAddStates(48, 50);
   break;
case 46:
   if ((0x3ff0000000000000L & l) == 0L)
      break;
   if (kind > 11)
      kind = 11;
   jjCheckNAddStates(48, 50);
   break;
case 48:
   if ((0x280000000000L & l) != 0L)
      jjCheckNAdd(49);
   break;
case 49:
   if ((0x3ff0000000000000L & l) == 0L)
      break;
   if (kind > 11)
      kind = 11;
   jjCheckNAddTwoStates(49, 50);
   break;
case 51:
   if ((0x3ff0000000000000L & l) != 0L)
      jjCheckNAddTwoStates(51, 52);
   break;
case 53:
   if ((0x280000000000L & l) != 0L)
      jjCheckNAdd(54);
   break;
case 54:
   if ((0x3ff0000000000000L & l) == 0L)
      break;
   if (kind > 11)
      kind = 11;
   jjCheckNAddTwoStates(54, 50);
   break;
case 55:
   if ((0x3ff0000000000000L & l) == 0L)
      break;
   if (kind > 11)
      kind = 11;
   jjCheckNAddStates(51, 53);
   break;
case 57:
   if ((0x280000000000L & l) != 0L)
      jjCheckNAdd(58);
   break;
case 58:
   if ((0x3ff0000000000000L & l) == 0L)
      break;
   if (kind > 11)
      kind = 11;
   jjCheckNAddTwoStates(58, 50);
   break;
case 59:
```

```
      if (curChar != 48)
         break;
      if (kind > 10)
         kind = 10;
      jjCheckNAddStates(20, 23);
      break;
   case 60:
      if ((0xff000000000000L & l) == 0L)
         break;
      if (kind > 10)
         kind = 10;
      jjCheckNAddStates(54, 56);
      break;
   case 64:
      if ((0x3ff000000000000L & l) == 0L)
         break;
      if (kind > 10)
         kind = 10;
      jjAddStates(57, 59);
      break;
   case 67:
      if (curChar == 47)
         jjAddStates(7, 10);
      break;
   case 69:
      if (curChar != 47)
         break;
      if (kind > 7)
         kind = 7;
      jjCheckNAdd(70);
      break;
   case 70:
      if ((0xfffffffffffdbffL & l) == 0L)
         break;
      if (kind > 7)
         kind = 7;
      jjCheckNAdd(70);
      break;
   case 71:
      if (curChar == 47 && kind > 95)
         kind = 95;
      break;
   case 72:
      if (curChar == 42 && kind > 96)
         kind = 96;
      break;
   case 73:
      if (curChar != 46)
         break;
      if (kind > 49)
         kind = 49;
      jjCheckNAdd(74);
      break;
   case 74:
      if ((0x3ff000000000000L & l) == 0L)
         break;
      if (kind > 11)
         kind = 11;
      jjCheckNAddStates(60, 62);
      break;
   case 76:
      if ((0x2800000000000L & l) != 0L)
         jjCheckNAdd(77);
      break;
   case 77:
      if ((0x3ff000000000000L & l) == 0L)
         break;
      if (kind > 11)
         kind = 11;
      jjCheckNAddTwoStates(77, 50);
      break;
```

```
         default : break;
      }
   } while(i != startsAt);
}
else if (curChar < 128)
{
   long l = 1L << (curChar & 077);
   do
   {
      switch(jjstateSet[--i])
      {
         case 0:
         case 42:
            if ((0x7fffffe87ffffeL & l) == 0L)
               break;
            if (kind > 98)
               kind = 98;
            jjCheckNAdd(42);
            break;
         case 78:
            if ((0xffffffffefffffffL & l) != 0L)
               jjCheckNAddStates(11, 13);
            else if (curChar == 92)
               jjAddStates(63, 67);
            break;
         case 11:
            if ((0xffffffffefffffffL & l) != 0L)
               jjCheckNAdd(12);
            else if (curChar == 92)
               jjAddStates(68, 70);
            break;
         case 6:
         case 61:
         case 65:
            if ((0x100000001000L & l) == 0L)
               break;
            if (kind > 10)
               kind = 10;
            jjCheckNAdd(7);
            break;
         case 7:
            if ((0x20000000200000L & l) != 0L && kind > 10)
               kind = 10;
            break;
         case 8:
         case 62:
         case 66:
            if ((0x20000000200000L & l) == 0L)
               break;
            if (kind > 10)
               kind = 10;
            jjCheckNAdd(9);
            break;
         case 9:
            if ((0x100000001000L & l) != 0L && kind > 10)
               kind = 10;
            break;
         case 13:
            if (curChar == 92)
               jjAddStates(68, 70);
            break;
         case 14:
            if ((0x54404610000000L & l) != 0L)
               jjCheckNAdd(12);
            break;
         case 18:
            if ((0x1000000001000000L & l) != 0L)
               jjstateSet[jjnewStateCnt++] = 19;
            break;
         case 19:
            if ((0x7e0000007eL & l) != 0L)
```

```
      jjCheckNAddTwoStates(12, 20);
   break;
case 20:
   if ((0x7e0000007eL & l) != 0L)
      jjCheckNAddStates(24, 26);
   break;
case 21:
   if ((0x7e0000007eL & l) != 0L)
      jjCheckNAdd(12);
   break;
case 22:
   if ((0x7e0000007eL & l) != 0L)
      jjCheckNAddTwoStates(21, 12);
   break;
case 24:
   if ((0xfffffffffeffffffL & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 26:
   if (curChar == 92)
      jjAddStates(63, 67);
   break;
case 27:
   if ((0x544046100000000L & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 31:
   if ((0x1000000010000000L & l) != 0L)
      jjstateSet[jjnewStateCnt++] = 32;
   break;
case 32:
   if ((0x7e0000007eL & l) != 0L)
      jjCheckNAddStates(35, 38);
   break;
case 33:
   if ((0x7e0000007eL & l) != 0L)
      jjCheckNAddStates(39, 43);
   break;
case 34:
   if ((0x7e0000007eL & l) != 0L)
      jjCheckNAddStates(11, 13);
   break;
case 35:
   if ((0x7e0000007eL & l) != 0L)
      jjCheckNAddStates(44, 47);
   break;
case 47:
   if ((0x2000000020L & l) != 0L)
      jjAddStates(71, 72);
   break;
case 50:
   if ((0x104000001040L & l) != 0L && kind > 11)
      kind = 11;
   break;
case 52:
   if ((0x2000000020L & l) != 0L)
      jjAddStates(73, 74);
   break;
case 56:
   if ((0x2000000020L & l) != 0L)
      jjAddStates(75, 76);
   break;
case 63:
   if ((0x1000000010000000L & l) != 0L)
      jjCheckNAdd(64);
   break;
case 64:
   if ((0x7e0000007eL & l) == 0L)
      break;
   if (kind > 10)
      kind = 10;
```

```
                  jjCheckNAddStates(57, 59);
                  break;
               case 70:
                  if (kind > 7)
                     kind = 7;
                  jjstateSet[jjnewStateCnt++] = 70;
                  break;
               case 75:
                  if ((0x2000000020L & l) != 0L)
                     jjAddStates(77, 78);
                  break;
               default : break;
            }
         } while(i != startsAt);
      }
      else
      {
         int i2 = (curChar & 0xff) >> 6;
         long l2 = 1L << (curChar & 077);
         do
         {
            switch(jjstateSet[--i])
            {
               case 78:
               case 24:
                  if ((jjbitVec0[i2] & l2) != 0L)
                     jjCheckNAddStates(11, 13);
                  break;
               case 11:
                  if ((jjbitVec0[i2] & l2) != 0L)
                     jjstateSet[jjnewStateCnt++] = 12;
                  break;
               case 70:
                  if ((jjbitVec0[i2] & l2) == 0L)
                     break;
                  if (kind > 7)
                     kind = 7;
                  jjstateSet[jjnewStateCnt++] = 70;
                  break;
               default : break;
            }
         } while(i != startsAt);
      }
      if (kind != 0x7fffffff)
      {
         jjmatchedKind = kind;
         jjmatchedPos = curPos;
         kind = 0x7fffffff;
      }
      ++curPos;
      if ((i = jjnewStateCnt) == (startsAt = 78 - (jjnewStateCnt = startsAt)))
         return curPos;
      try { curChar = input_stream.readChar(); }
      catch(java.io.IOException e) { return curPos; }
   }
}
private int jjMoveStringLiteralDfa0_1()
{
   return jjMoveNfa_1(0, 0);
}
private int jjMoveNfa_1(int startState, int curPos)
{
   int startsAt = 0;
   jjnewStateCnt = 8;
   int i = 1;
   jjstateSet[0] = startState;
   int kind = 0x7fffffff;
   for (;;)
   {
      if (++jjround == 0x7fffffff)
         ReInitRounds();
```

```
if (curChar < 64)
{
   long l = 1L << curChar;
   do
   {
      switch(jjstateSet[--i])
      {
         case 0:
            if (kind > 2)
               kind = 2;
            if ((0x2400L & l) != 0L)
            {
               if (kind > 3)
                  kind = 3;
            }
            if (curChar == 13)
               jjstateSet[jjnewStateCnt++] = 6;
            break;
         case 1:
            if ((0x2400L & l) != 0L && kind > 2)
               kind = 2;
            break;
         case 2:
            if (curChar == 10 && kind > 2)
               kind = 2;
            break;
         case 3:
            if (curChar == 13)
               jjstateSet[jjnewStateCnt++] = 2;
            break;
         case 4:
            if (kind > 2)
               kind = 2;
            break;
         case 5:
            if ((0x2400L & l) != 0L && kind > 3)
               kind = 3;
            break;
         case 6:
            if (curChar == 10 && kind > 3)
               kind = 3;
            break;
         case 7:
            if (curChar == 13)
               jjstateSet[jjnewStateCnt++] = 6;
            break;
         default : break;
      }
   } while(i != startsAt);
}
else if (curChar < 128)
{
   long l = 1L << (curChar & 077);
   do
   {
      switch(jjstateSet[--i])
      {
         case 0:
            if (kind > 2)
               kind = 2;
            if (curChar == 92)
               jjAddStates(79, 80);
            break;
         case 4:
            if (kind > 2)
               kind = 2;
            break;
         default : break;
      }
   } while(i != startsAt);
}
```

```
      else
      {
         int i2 = (curChar & 0xff) >> 6;
         long l2 = 1L << (curChar & 077);
         do
         {
            switch(jjstateSet[--i])
            {
               case 0:
                  if ((jjbitVec0[i2] & l2) != 0L && kind > 2)
                     kind = 2;
                  break;
               default : break;
            }
         } while(i != startsAt);
      }
      if (kind != 0x7fffffff)
      {
         jjmatchedKind = kind;
         jjmatchedPos = curPos;
         kind = 0x7fffffff;
      }
      ++curPos;
      if ((i = jjnewStateCnt) == (startsAt = 8 - (jjnewStateCnt = startsAt)))
         return curPos;
      try { curChar = input_stream.readChar(); }
      catch(java.io.IOException e) { return curPos; }
   }
}
private int jjMoveStringLiteralDfa0_2()
{
   return jjMoveNfa_2(0, 0);
}
private int jjMoveNfa_2(int startState, int curPos)
{
   int startsAt = 0;
   jjnewStateCnt = 3;
   int i = 1;
   jjstateSet[0] = startState;
   int kind = 0x7fffffff;
   for (;;)
   {
      if (++jjround == 0x7fffffff)
         ReInitRounds();
      if (curChar < 64)
      {
         long l = 1L << curChar;
         do
         {
            switch(jjstateSet[--i])
            {
               case 0:
                  if (kind > 5)
                     kind = 5;
                  if (curChar == 42)
                     jjstateSet[jjnewStateCnt++] = 1;
                  break;
               case 1:
                  if (curChar == 47 && kind > 6)
                     kind = 6;
                  break;
               case 2:
                  if (curChar == 42)
                     jjstateSet[jjnewStateCnt++] = 1;
                  break;
               default : break;
            }
         } while(i != startsAt);
      }
      else if (curChar < 128)
      {
```

```
         long l = 1L << (curChar & 077);
         do
         {
            switch(jjstateSet[--i])
            {
               case 0:
                  kind = 5;
                  break;
               default : break;
            }
         } while(i != startsAt);
      }
      else
      {
         int i2 = (curChar & 0xff) >> 6;
         long l2 = 1L << (curChar & 077);
         do
         {
            switch(jjstateSet[--i])
            {
               case 0:
                  if ((jjbitVec0[i2] & l2) != 0L && kind > 5)
                     kind = 5;
                  break;
               default : break;
            }
         } while(i != startsAt);
      }
      if (kind != 0x7fffffff)
      {
         jjmatchedKind = kind;
         jjmatchedPos = curPos;
         kind = 0x7fffffff;
      }
      ++curPos;
      if ((i = jjnewStateCnt) == (startsAt = 3 - (jjnewStateCnt = startsAt)))
         return curPos;
      try { curChar = input_stream.readChar(); }
      catch(java.io.IOException e) { return curPos; }
   }
}
static final int[] jjnextStates = {
   44, 45, 51, 52, 55, 56, 50, 68, 69, 71, 72, 24, 25, 26, 11, 13,
   12, 5, 6, 8, 60, 61, 62, 63, 21, 12, 22, 24, 25, 26, 29, 24,
   30, 25, 26, 24, 25, 26, 33, 24, 34, 25, 26, 35, 24, 34, 25, 26,
   46, 47, 50, 55, 56, 50, 60, 61, 62, 64, 65, 66, 74, 75, 50, 27,
   28, 31, 36, 38, 14, 15, 18, 48, 49, 53, 54, 57, 58, 76, 77, 1,
   3,
};

/** Token literal values. */
public static final String[] jjstrLiteralImages = {
"", null, null, null, null, null, null, null, null, null, null,
null, "\141\165\164\157", "\162\145\147\151\163\164\145\162",
"\145\170\164\145\162\156", "\163\164\162\165\143\164", "\165\156\151\157\156",
"\163\164\141\164\151\143", "\164\171\160\145\144\145\146", "\145\156\165\155",
"\143\157\156\163\164",
"\166\157\154\141\164\151\154\145", "\166\157\151\144", "\143\150\141\162",
"\163\150\157\162\164",
"\151\156\164", "\154\157\156\147", "\146\154\157\141\164", "\144\157\165\142\154\145",
"\163\151\147\156\145\144", "\165\156\163\151\147\156\145\144", "\151\146",
"\145\154\163\145",
"\163\167\151\164\143\150", "\143\141\163\145", "\146\157\162", "\144\157",
"\167\150\151\154\145",
"\142\162\145\141\153", "\143\157\156\164\151\156\165\145",
"\144\145\146\141\165\154\164",
"\147\157\164\157", "\163\151\172\145\157\146", "\162\145\164\165\162\156", "\53\53",
"\55\55",
"\55\76", null, "\50", "\51", "\133", "\135", "\173", "\175", "\46", "\52", "\53",
"\55", "\176", "\41", "\57", "\45", "\74\74", "\76\76", "\74", "\76", "\74\75",
"\76\75", "\75\75", "\41\75", "\136", "\174", "\46\46", "\174\174", "\77", "\72", "\75",
```

```
"\52\75", "\57\75", "\45\75", "\53\75", "\55\75", "\74\74\75", "\76\76\75", "\174\75",
"\46\75", "\136\75", "\73", "\54", "\47", "\42", "\100", "\56\56\56", null, null, null,
null, null, null, null, null, null, null, null, null, null, null, null, null, null,
null, null, null, null, null, null, null, };

/** Lexer state names. */
public static final String[] lexStateNames = {
   "DEFAULT",
   "INSIDE_PREPROCESSOR",
   "INSIDE_BLOCK_COMMENT",
};

/** Lex State array. */
public static final int[] jjnewLexState = {
   -1, 1, -1, 0, 2, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1,
   -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1,
   -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1,
   -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -
1, -1, -1, -1,
   -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
};
static final long[] jjtoToken = {
   0xffffffffffffc01L, 0x7fffffffL,
};
static final long[] jjtoSkip = {
   0x3c8L, 0x0L,
};
static final long[] jjtoSpecial = {
   0xc8L, 0x0L,
};
static final long[] jjtoMore = {
   0x36L, 0x0L,
};
protected SimpleCharStream input_stream;
private final int[] jjrounds = new int[78];
private final int[] jjstateSet = new int[156];
private final StringBuilder jjimage = new StringBuilder();
private StringBuilder image = jjimage;
private int jjimageLen;
private int lengthOfMatch;
protected char curChar;
/** Constructor. */
public CParserTokenManager(SimpleCharStream stream){
   if (SimpleCharStream.staticFlag)
      throw new Error("ERROR: Cannot use a static CharStream class with a non-static
lexical analyzer.");
   input_stream = stream;
}

/** Constructor. */
public CParserTokenManager(SimpleCharStream stream, int lexState){
   this(stream);
   SwitchTo(lexState);
}

/** Reinitialise parser. */
public void ReInit(SimpleCharStream stream)
{
   jjmatchedPos = jjnewStateCnt = 0;
   curLexState = defaultLexState;
   input_stream = stream;
   ReInitRounds();
}
private void ReInitRounds()
{
   int i;
   jjround = 0x80000001;
   for (i = 78; i-- > 0;)
```

```
      jjrounds[i] = 0x80000000;
}

/** Reinitialise parser. */
public void ReInit(SimpleCharStream stream, int lexState)
{
   ReInit(stream);
   SwitchTo(lexState);
}

/** Switch to specified lex state. */
public void SwitchTo(int lexState)
{
   if (lexState >= 3 || lexState < 0)
      throw new TokenMgrError("Error: Ignoring invalid lexical state : " + lexState + ".
State unchanged.", TokenMgrError.INVALID_LEXICAL_STATE);
   else
      curLexState = lexState;
}

protected Token jjFillToken()
{
   final Token t;
   final String curTokenImage;
   final int beginLine;
   final int endLine;
   final int beginColumn;
   final int endColumn;
   String im = jjstrLiteralImages[jjmatchedKind];
   curTokenImage = (im == null) ? input_stream.GetImage() : im;
   beginLine = input_stream.getBeginLine();
   beginColumn = input_stream.getBeginColumn();
   endLine = input_stream.getEndLine();
   endColumn = input_stream.getEndColumn();
   t = Token.newToken(jjmatchedKind, curTokenImage);

   t.beginLine = beginLine;
   t.endLine = endLine;
   t.beginColumn = beginColumn;
   t.endColumn = endColumn;

   return t;
}

int curLexState = 0;
int defaultLexState = 0;
int jjnewStateCnt;
int jjround;
int jjmatchedPos;
int jjmatchedKind;

/** Get the next Token. */
public Token getNextToken()
{
  Token specialToken = null;
  Token matchedToken;
  int curPos = 0;

  EOFLoop :
  for (;;)
  {
   try
   {
      curChar = input_stream.BeginToken();
   }
   catch(java.io.IOException e)
   {
      jjmatchedKind = 0;
      matchedToken = jjFillToken();
      matchedToken.specialToken = specialToken;
      return matchedToken;
```

```
}
image = jjimage;
image.setLength(0);
jjimageLen = 0;

for (;;)
{
  switch(curLexState)
  {
    case 0:
      jjmatchedKind = 0x7fffffff;
      jjmatchedPos = 0;
      curPos = jjMoveStringLiteralDfa0_0();
      break;
    case 1:
      jjmatchedKind = 0x7fffffff;
      jjmatchedPos = 0;
      curPos = jjMoveStringLiteralDfa0_1();
      break;
    case 2:
      jjmatchedKind = 0x7fffffff;
      jjmatchedPos = 0;
      curPos = jjMoveStringLiteralDfa0_2();
      break;
  }
  if (jjmatchedKind != 0x7fffffff)
  {
     if (jjmatchedPos + 1 < curPos)
        input_stream.backup(curPos - jjmatchedPos - 1);
     if ((jjtoToken[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
     {
        matchedToken = jjFillToken();
        matchedToken.specialToken = specialToken;
    if (jjnewLexState[jjmatchedKind] != -1)
      curLexState = jjnewLexState[jjmatchedKind];
        return matchedToken;
     }
     else if ((jjtoSkip[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
     {
        if ((jjtoSpecial[jjmatchedKind >> 6] & (1L << (jjmatchedKind & 077))) != 0L)
        {
           matchedToken = jjFillToken();
           if (specialToken == null)
              specialToken = matchedToken;
           else
           {
              matchedToken.specialToken = specialToken;
              specialToken = (specialToken.next = matchedToken);
           }
           SkipLexicalActions(matchedToken);
        }
        else
           SkipLexicalActions(null);
      if (jjnewLexState[jjmatchedKind] != -1)
        curLexState = jjnewLexState[jjmatchedKind];
        continue EOFLoop;
     }
     jjimageLen += jjmatchedPos + 1;
   if (jjnewLexState[jjmatchedKind] != -1)
     curLexState = jjnewLexState[jjmatchedKind];
     curPos = 0;
     jjmatchedKind = 0x7fffffff;
     try {
        curChar = input_stream.readChar();
        continue;
     }
     catch (java.io.IOException e1) { }
  }
  int error_line = input_stream.getEndLine();
  int error_column = input_stream.getEndColumn();
  String error_after = null;
```

```
     boolean EOFSeen = false;
     try { input_stream.readChar(); input_stream.backup(1); }
     catch (java.io.IOException e1) {
        EOFSeen = true;
        error_after = curPos <= 1 ? "" : input_stream.GetImage();
        if (curChar == '\n' || curChar == '\r') {
           error_line++;
           error_column = 0;
        }
        else
           error_column++;
     }
     if (!EOFSeen) {
        input_stream.backup(1);
        error_after = curPos <= 1 ? "" : input_stream.GetImage();
     }
     throw new TokenMgrError(EOFSeen, curLexState, error_line, error_column, error_after,
curChar, TokenMgrError.LEXICAL_ERROR);
   }
  }
}

void SkipLexicalActions(Token matchedToken)
{
   switch(jjmatchedKind)
   {
      default :
         break;
   }
}
private void jjCheckNAdd(int state)
{
   if (jjrounds[state] != jjround)
   {
      jjstateSet[jjnewStateCnt++] = state;
      jjrounds[state] = jjround;
   }
}
private void jjAddStates(int start, int end)
{
   do {
      jjstateSet[jjnewStateCnt++] = jjnextStates[start];
   } while (start++ != end);
}
private void jjCheckNAddTwoStates(int state1, int state2)
{
   jjCheckNAdd(state1);
   jjCheckNAdd(state2);
}

private void jjCheckNAddStates(int start, int end)
{
   do {
      jjCheckNAdd(jjnextStates[start]);
   } while (start++ != end);
}

}

/* Generated By:JavaCC: Do not edit this line. CParserTreeConstants.java Version 5.0 */
package nql.cd.frontend;

public interface CParserTreeConstants
{
  public int JJTPARSE = 0;
  public int JJTEXTERNALDECLARATION = 1;
  public int JJTFUNCTIONDEFINITION = 2;
  public int JJTFUNCTIONDEFINITIONLOOKAHEAD = 3;
  public int JJTDECLARATIONLIST = 4;
  public int JJTDECLARATION = 5;
  public int JJTDECLARATIONSPECIFIERS = 6;
```

```
public int JJTINITDECLARATORLIST = 7;
public int JJTINITDECLARATOR = 8;
public int JJTINITIALIZER = 9;
public int JJTINITIALIZERLIST = 10;
public int JJTTYPENAME = 11;
public int JJTSTORAGECLASSSPECIFIER = 12;
public int JJTTYPESPECIFIER = 13;
public int JJTTYPEDEFNAME = 14;
public int JJTTYPEQUALIFIER = 15;
public int JJTSTRUCTORUNIONSPECIFIER = 16;
public int JJTSTRUCTORUNION = 17;
public int JJTSTRUCTDECLARATIONLIST = 18;
public int JJTSTRUCTDECLARATION = 19;
public int JJTVOID = 20;
public int JJTSTRUCTDECLARATORLIST = 21;
public int JJTSTRUCTDECLARATOR = 22;
public int JJTENUMSPECIFIER = 23;
public int JJTENUMERATORLIST = 24;
public int JJTENUMERATOR = 25;
public int JJTDECLARATOR = 26;
public int JJTPOINTER = 27;
public int JJTDIRECTDECLARATOR = 28;
public int JJTPARAMETERTYPELIST = 29;
public int JJTPARAMETERLIST = 30;
public int JJTPARAMETERDECLARATION = 31;
public int JJTABSTRACTDECLARATOR = 32;
public int JJTDIRECTABSTRACTDECLARATOR = 33;
public int JJTCOMPOUNDSTATEMENT = 34;
public int JJTLABELEDSTATEMENT = 35;
public int JJTJUMPSTATEMENT = 36;
public int JJTEXPRESSIONSTATEMENT = 37;
public int JJTIFSTATEMENT = 38;
public int JJTSWITCHSTATEMENT = 39;
public int JJTFORSTATEMENT = 40;
public int JJTWHILESTATEMENT = 41;
public int JJTDOWHILESTATEMENT = 42;
public int JJTEXPRESSION = 43;
public int JJTASSIGNMENTEXPRESSION = 44;
public int JJTCONDITIONALEXPRESSION = 45;
public int JJTLOGICALOREXPRESSION = 46;
public int JJTLOGICALANDEXPRESSION = 47;
public int JJTBITWISEOREXPRESSION = 48;
public int JJTBITWISEXOREXPRESSION = 49;
public int JJTBITWISEANDEXPRESSION = 50;
public int JJTEQUALITYEXPRESSION = 51;
public int JJTRELATIONALEXPRESSION = 52;
public int JJTSHIFTEXPRESSION = 53;
public int JJTADDITIVEEXPRESSION = 54;
public int JJTMULTIPLICATIVEEXPRESSION = 55;
public int JJTCASTEXPRESSION = 56;
public int JJTUNARYEXPRESSION = 57;
public int JJTPOSTFIXEXPRESSION = 58;
public int JJTIDENTIFIER = 59;
public int JJTINTEGER = 60;
public int JJTREAL = 61;
public int JJTSTRING = 62;
public int JJTCHARACTER = 63;


public String[] jjtNodeName = {
  "parse",
  "externalDeclaration",
  "functionDefinition",
  "functionDefinitionLookahead",
  "declarationList",
  "declaration",
  "declarationSpecifiers",
  "initDeclaratorList",
  "initDeclarator",
  "initializer",
  "initializerList",
```

```
    "typeName",
    "storageClassSpecifier",
    "typeSpecifier",
    "typedefName",
    "typeQualifier",
    "structOrUnionSpecifier",
    "structOrUnion",
    "structDeclarationList",
    "structDeclaration",
    "void",
    "structDeclaratorList",
    "structDeclarator",
    "enumSpecifier",
    "enumeratorList",
    "enumerator",
    "declarator",
    "pointer",
    "directDeclarator",
    "parameterTypeList",
    "parameterList",
    "parameterDeclaration",
    "abstractDeclarator",
    "directAbstractDeclarator",
    "compoundStatement",
    "labeledStatement",
    "jumpStatement",
    "expressionStatement",
    "ifStatement",
    "switchStatement",
    "forStatement",
    "whileStatement",
    "doWhileStatement",
    "expression",
    "assignmentExpression",
    "conditionalExpression",
    "logicalORExpression",
    "logicalANDExpression",
    "bitwiseORExpression",
    "bitwiseXORExpression",
    "bitwiseANDExpression",
    "equalityExpression",
    "relationalExpression",
    "shiftExpression",
    "additiveExpression",
    "multiplicativeExpression",
    "castExpression",
    "unaryExpression",
    "postfixExpression",
    "identifier",
    "integer",
    "real",
    "string",
    "character",
  };
}
/* JavaCC - OriginalChecksum=51b456a78f07f50edf2c2a44a2b0f33a (do not edit this line) */

/* Generated By:JavaCC: Do not edit this line. CParserVisitor.java Version 5.0 */
package nql.cd.frontend;

public interface CParserVisitor
{
  public Object visit(SimpleNode node, Object data);
  public Object visit(ASTparse node, Object data);
  public Object visit(ASTexternalDeclaration node, Object data);
  public Object visit(ASTfunctionDefinition node, Object data);
  public Object visit(ASTfunctionDefinitionLookahead node, Object data);
  public Object visit(ASTdeclarationList node, Object data);
  public Object visit(ASTdeclaration node, Object data);
  public Object visit(ASTdeclarationSpecifiers node, Object data);
  public Object visit(ASTinitDeclaratorList node, Object data);
```

```
  public Object visit(ASTinitDeclarator node, Object data);
  public Object visit(ASTinitializer node, Object data);
  public Object visit(ASTinitializerList node, Object data);
  public Object visit(ASTtypeName node, Object data);
  public Object visit(ASTstorageClassSpecifier node, Object data);
  public Object visit(ASTtypeSpecifier node, Object data);
  public Object visit(ASTtypedefName node, Object data);
  public Object visit(ASTtypeQualifier node, Object data);
  public Object visit(ASTstructOrUnionSpecifier node, Object data);
  public Object visit(ASTstructOrUnion node, Object data);
  public Object visit(ASTstructDeclarationList node, Object data);
  public Object visit(ASTstructDeclaration node, Object data);
  public Object visit(ASTstructDeclaratorList node, Object data);
  public Object visit(ASTstructDeclarator node, Object data);
  public Object visit(ASTenumSpecifier node, Object data);
  public Object visit(ASTenumeratorList node, Object data);
  public Object visit(ASTenumerator node, Object data);
  public Object visit(ASTdeclarator node, Object data);
  public Object visit(ASTpointer node, Object data);
  public Object visit(ASTdirectDeclarator node, Object data);
  public Object visit(ASTparameterTypeList node, Object data);
  public Object visit(ASTparameterList node, Object data);
  public Object visit(ASTparameterDeclaration node, Object data);
  public Object visit(ASTabstractDeclarator node, Object data);
  public Object visit(ASTdirectAbstractDeclarator node, Object data);
  public Object visit(ASTcompoundStatement node, Object data);
  public Object visit(ASTlabeledStatement node, Object data);
  public Object visit(ASTjumpStatement node, Object data);
  public Object visit(ASTexpressionStatement node, Object data);
  public Object visit(ASTifStatement node, Object data);
  public Object visit(ASTswitchStatement node, Object data);
  public Object visit(ASTforStatement node, Object data);
  public Object visit(ASTwhileStatement node, Object data);
  public Object visit(ASTdoWhileStatement node, Object data);
  public Object visit(ASTexpression node, Object data);
  public Object visit(ASTassignmentExpression node, Object data);
  public Object visit(ASTconditionalExpression node, Object data);
  public Object visit(ASTlogicalORExpression node, Object data);
  public Object visit(ASTlogicalANDExpression node, Object data);
  public Object visit(ASTbitwiseORExpression node, Object data);
  public Object visit(ASTbitwiseXORExpression node, Object data);
  public Object visit(ASTbitwiseANDExpression node, Object data);
  public Object visit(ASTequalityExpression node, Object data);
  public Object visit(ASTrelationalExpression node, Object data);
  public Object visit(ASTshiftExpression node, Object data);
  public Object visit(ASTadditiveExpression node, Object data);
  public Object visit(ASTmultiplicativeExpression node, Object data);
  public Object visit(ASTcastExpression node, Object data);
  public Object visit(ASTunaryExpression node, Object data);
  public Object visit(ASTpostfixExpression node, Object data);
  public Object visit(ASTidentifier node, Object data);
  public Object visit(ASTinteger node, Object data);
  public Object visit(ASTreal node, Object data);
  public Object visit(ASTstring node, Object data);
  public Object visit(ASTcharacter node, Object data);
}
/* JavaCC - OriginalChecksum=7da48192cbde959bde3cddf30a62a9ad (do not edit this line) */

/* Generated By:JavaCC: Do not edit this line. JJTCParserState.java Version 5.0 */
package nql.cd.frontend;

public class JJTCParserState {
  private java.util.List<Node> nodes;
  private java.util.List<Integer> marks;

  private int sp;         // number of nodes on stack
  private int mk;         // current mark
  private boolean node_created;

  public JJTCParserState() {
    nodes = new java.util.ArrayList<Node>();
```

```
  marks = new java.util.ArrayList<Integer>();
  sp = 0;
  mk = 0;
}

/* Determines whether the current node was actually closed and
   pushed.  This should only be called in the final user action of a
   node scope.  */
public boolean nodeCreated() {
  return node_created;
}

/* Call this to reinitialize the node stack.  It is called
   automatically by the parser's ReInit() method. */
public void reset() {
  nodes.clear();
  marks.clear();
  sp = 0;
  mk = 0;
}

/* Returns the root node of the AST.  It only makes sense to call
   this after a successful parse. */
public Node rootNode() {
  return nodes.get(0);
}

/* Pushes a node on to the stack. */
public void pushNode(Node n) {
  nodes.add(n);
  ++sp;
}

/* Returns the node on the top of the stack, and remove it from the
   stack.  */
public Node popNode() {
  if (--sp < mk) {
    mk = marks.remove(marks.size()-1);
  }
  return nodes.remove(nodes.size()-1);
}

/* Returns the node currently on the top of the stack. */
public Node peekNode() {
  return nodes.get(nodes.size()-1);
}

/* Returns the number of children on the stack in the current node
   scope. */
public int nodeArity() {
  return sp - mk;
}


public void clearNodeScope(Node n) {
  while (sp > mk) {
    popNode();
  }
  mk = marks.remove(marks.size()-1);
}


public void openNodeScope(Node n) {
  marks.add(mk);
  mk = sp;
  n.jjtOpen();
}


/* A definite node is constructed from a specified number of
   children.  That number of nodes are popped from the stack and
```

```
     made the children of the definite node.  Then the definite node
     is pushed on to the stack. */
  public void closeNodeScope(Node n, int num) {
    mk = marks.remove(marks.size()-1);
    while (num-- > 0) {
      Node c = popNode();
      c.jjtSetParent(n);
      n.jjtAddChild(c, num);
    }
    n.jjtClose();
    pushNode(n);
    node_created = true;
  }


  /* A conditional node is constructed if its condition is true.  All
     the nodes that have been pushed since the node was opened are
     made children of the conditional node, which is then pushed
     on to the stack.  If the condition is false the node is not
     constructed and they are left on the stack. */
  public void closeNodeScope(Node n, boolean condition) {
    if (condition) {
      int a = nodeArity();
      mk = marks.remove(marks.size()-1);
      while (a-- > 0) {
        Node c = popNode();
        c.jjtSetParent(n);
        n.jjtAddChild(c, a);
      }
      n.jjtClose();
      pushNode(n);
      node_created = true;
    } else {
      mk = marks.remove(marks.size()-1);
      node_created = false;
    }
  }
}
/* JavaCC - OriginalChecksum=857616b5a2089fbcc3c5d7d4a9d7053f (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. Node.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

/* All AST nodes must implement this interface.  It provides basic
   machinery for constructing the parent and child relationships
   between nodes. */

public
interface Node {

  /** This method is called after the node has been made the current
    node.  It indicates that child nodes can now be added to it. */
  public void jjtOpen();

  /** This method is called after all the child nodes have been
    added. */
  public void jjtClose();

  /** This pair of methods are used to inform the node of its
    parent. */
  public void jjtSetParent(Node n);
  public Node jjtGetParent();

  /** This method tells the node to add its argument to the node's
    list of children.  */
  public void jjtAddChild(Node n, int i);
```

```
  /** This method returns a child node.  The children are numbered
     from zero, left to right. */
  public Node jjtGetChild(int i);

  /** Return the number of children the node has. */
  public int jjtGetNumChildren();

  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data);
}
/* JavaCC - OriginalChecksum=4f2d842a280979a186aeab9dabe26aea (do not edit this line) */

/* Generated By:JavaCC: Do not edit this line. ParseException.java Version 5.0 */
/* JavaCCOptions:KEEP_LINE_COL=null */
package nql.cd.frontend;

/**
 * This exception is thrown when parse errors are encountered.
 * You can explicitly create objects of this exception type by
 * calling the method generateParseException in the generated
 * parser.
 *
 * You can modify this class to customize your error reporting
 * mechanisms so long as you retain the public fields.
 */
public class ParseException extends Exception {

  /**
   * The version identifier for this Serializable class.
   * Increment only if the <i>serialized</i> form of the
   * class changes.
   */
  private static final long serialVersionUID = 1L;

  /**
   * This constructor is used by the method "generateParseException"
   * in the generated parser.  Calling this constructor generates
   * a new object of this type with the fields "currentToken",
   * "expectedTokenSequences", and "tokenImage" set.
   */
  public ParseException(Token currentTokenVal,
                        int[][] expectedTokenSequencesVal,
                        String[] tokenImageVal
                       )
  {
    super(initialise(currentTokenVal, expectedTokenSequencesVal, tokenImageVal));
    currentToken = currentTokenVal;
    expectedTokenSequences = expectedTokenSequencesVal;
    tokenImage = tokenImageVal;
  }

  /**
   * The following constructors are for use by you for whatever
   * purpose you can think of.  Constructing the exception in this
   * manner makes the exception behave in the normal way - i.e., as
   * documented in the class "Throwable".  The fields "errorToken",
   * "expectedTokenSequences", and "tokenImage" do not contain
   * relevant information.  The JavaCC generated code does not use
   * these constructors.
   */

  public ParseException() {
    super();
  }

  /** Constructor with message. */
  public ParseException(String message) {
    super(message);
  }
```

```
  /**
   * This is the last token that has been consumed successfully.  If
   * this object has been created due to a parse error, the token
   * followng this token will (therefore) be the first error token.
   */
  public Token currentToken;

  /**
   * Each entry in this array is an array of integers.  Each array
   * of integers represents a sequence of tokens (by their ordinal
   * values) that is expected at this point of the parse.
   */
  public int[][] expectedTokenSequences;

  /**
   * This is a reference to the "tokenImage" array of the generated
   * parser within which the parse error occurred.  This array is
   * defined in the generated ...Constants interface.
   */
  public String[] tokenImage;

  /**
   * It uses "currentToken" and "expectedTokenSequences" to generate a parse
   * error message and returns it.  If this object has been created
   * due to a parse error, and you do not catch it (it gets thrown
   * from the parser) the correct error message
   * gets displayed.
   */
  private static String initialise(Token currentToken,
                           int[][] expectedTokenSequences,
                           String[] tokenImage) {
    String eol = System.getProperty("line.separator", "\n");
    StringBuffer expected = new StringBuffer();
    int maxSize = 0;
    for (int i = 0; i < expectedTokenSequences.length; i++) {
      if (maxSize < expectedTokenSequences[i].length) {
        maxSize = expectedTokenSequences[i].length;
      }
      for (int j = 0; j < expectedTokenSequences[i].length; j++) {
        expected.append(tokenImage[expectedTokenSequences[i][j]]).append(' ');
      }
      if (expectedTokenSequences[i][expectedTokenSequences[i].length - 1] != 0) {
        expected.append("...");
      }
      expected.append(eol).append("    ");
    }
    String retval = "Encountered \"";
    Token tok = currentToken.next;
    for (int i = 0; i < maxSize; i++) {
      if (i != 0) retval += " ";
      if (tok.kind == 0) {
        retval += tokenImage[0];
        break;
      }
      retval += " " + tokenImage[tok.kind];
      retval += " \"";
      retval += add_escapes(tok.image);
      retval += " \"";
      tok = tok.next;
    }
    retval += "\" at line " + currentToken.next.beginLine + ", column " +
currentToken.next.beginColumn;
    retval += "." + eol;
    if (expectedTokenSequences.length == 1) {
      retval += "Was expecting:" + eol + "    ";
    } else {
      retval += "Was expecting one of:" + eol + "    ";
    }
    retval += expected.toString();
    return retval;
  }
```

```
  /**
   * The end of line string for this machine.
   */
  protected String eol = System.getProperty("line.separator", "\n");

  /**
   * Used to convert raw characters to their escaped version
   * when these raw version cannot be used as part of an ASCII
   * string literal.
   */
  static String add_escapes(String str) {
      StringBuffer retval = new StringBuffer();
      char ch;
      for (int i = 0; i < str.length(); i++) {
        switch (str.charAt(i))
        {
           case 0 :
              continue;
           case '\b':
              retval.append("\\b");
              continue;
           case '\t':
              retval.append("\\t");
              continue;
           case '\n':
              retval.append("\\n");
              continue;
           case '\f':
              retval.append("\\f");
              continue;
           case '\r':
              retval.append("\\r");
              continue;
           case '\"':
              retval.append("\\\"");
              continue;
           case '\'':
              retval.append("\\\'");
              continue;
           case '\\':
              retval.append("\\\\");
              continue;
           default:
              if ((ch = str.charAt(i)) < 0x20 || ch > 0x7e) {
                 String s = "0000" + Integer.toString(ch, 16);
                 retval.append("\\u" + s.substring(s.length() - 4, s.length()));
              } else {
                 retval.append(ch);
              }
              continue;
        }
      }
      return retval.toString();
   }

}
/* JavaCC - OriginalChecksum=6371b52b18f41e332301b83079642d68 (do not edit this line) */

/* Generated By:JavaCC: Do not edit this line. SimpleCharStream.java Version 5.0 */
/* JavaCCOptions:STATIC=false,SUPPORT_CLASS_VISIBILITY_PUBLIC=true */
package nql.cd.frontend;

/**
 * An implementation of interface CharStream, where the stream is assumed to
 * contain only ASCII characters (without unicode processing).
 */

public class SimpleCharStream
{
/** Whether parser is static. */
```

```
  public static final boolean staticFlag = false;
  int bufsize;
  int available;
  int tokenBegin;
/** Position in buffer. */
  public int bufpos = -1;
  protected int bufline[];
  protected int bufcolumn[];

  protected int column = 0;
  protected int line = 1;

  protected boolean prevCharIsCR = false;
  protected boolean prevCharIsLF = false;

  protected java.io.Reader inputStream;

  protected char[] buffer;
  protected int maxNextCharInd = 0;
  protected int inBuf = 0;
  protected int tabSize = 8;

  protected void setTabSize(int i) { tabSize = i; }
  protected int getTabSize(int i) { return tabSize; }


  protected void ExpandBuff(boolean wrapAround)
  {
    char[] newbuffer = new char[bufsize + 2048];
    int newbufline[] = new int[bufsize + 2048];
    int newbufcolumn[] = new int[bufsize + 2048];

    try
    {
      if (wrapAround)
      {
        System.arraycopy(buffer, tokenBegin, newbuffer, 0, bufsize - tokenBegin);
        System.arraycopy(buffer, 0, newbuffer, bufsize - tokenBegin, bufpos);
        buffer = newbuffer;

        System.arraycopy(bufline, tokenBegin, newbufline, 0, bufsize - tokenBegin);
        System.arraycopy(bufline, 0, newbufline, bufsize - tokenBegin, bufpos);
        bufline = newbufline;

        System.arraycopy(bufcolumn, tokenBegin, newbufcolumn, 0, bufsize - tokenBegin);
        System.arraycopy(bufcolumn, 0, newbufcolumn, bufsize - tokenBegin, bufpos);
        bufcolumn = newbufcolumn;

        maxNextCharInd = (bufpos += (bufsize - tokenBegin));
      }
      else
      {
        System.arraycopy(buffer, tokenBegin, newbuffer, 0, bufsize - tokenBegin);
        buffer = newbuffer;

        System.arraycopy(bufline, tokenBegin, newbufline, 0, bufsize - tokenBegin);
        bufline = newbufline;

        System.arraycopy(bufcolumn, tokenBegin, newbufcolumn, 0, bufsize - tokenBegin);
        bufcolumn = newbufcolumn;

        maxNextCharInd = (bufpos -= tokenBegin);
      }
    }
    catch (Throwable t)
    {
      throw new Error(t.getMessage());
    }


    bufsize += 2048;
```

```
    available = bufsize;
    tokenBegin = 0;
  }

  protected void FillBuff() throws java.io.IOException
  {
    if (maxNextCharInd == available)
    {
      if (available == bufsize)
      {
        if (tokenBegin > 2048)
        {
          bufpos = maxNextCharInd = 0;
          available = tokenBegin;
        }
        else if (tokenBegin < 0)
          bufpos = maxNextCharInd = 0;
        else
          ExpandBuff(false);
      }
      else if (available > tokenBegin)
        available = bufsize;
      else if ((tokenBegin - available) < 2048)
        ExpandBuff(true);
      else
        available = tokenBegin;
    }

    int i;
    try {
      if ((i = inputStream.read(buffer, maxNextCharInd, available - maxNextCharInd)) == -
1)
      {
        inputStream.close();
        throw new java.io.IOException();
      }
      else
        maxNextCharInd += i;
      return;
    }
    catch(java.io.IOException e) {
      --bufpos;
      backup(0);
      if (tokenBegin == -1)
        tokenBegin = bufpos;
      throw e;
    }
  }

/** Start. */
  public char BeginToken() throws java.io.IOException
  {
    tokenBegin = -1;
    char c = readChar();
    tokenBegin = bufpos;

    return c;
  }

  protected void UpdateLineColumn(char c)
  {
    column++;

    if (prevCharIsLF)
    {
      prevCharIsLF = false;
      line += (column = 1);
    }
    else if (prevCharIsCR)
    {
      prevCharIsCR = false;
```

```
      if (c == '\n')
      {
        prevCharIsLF = true;
      }
      else
        line += (column = 1);
    }

    switch (c)
    {
      case '\r' :
        prevCharIsCR = true;
        break;
      case '\n' :
        prevCharIsLF = true;
        break;
      case '\t' :
        column--;
        column += (tabSize - (column % tabSize));
        break;
      default :
        break;
    }

    bufline[bufpos] = line;
    bufcolumn[bufpos] = column;
  }

/** Read a character. */
  public char readChar() throws java.io.IOException
  {
    if (inBuf > 0)
    {
      --inBuf;

      if (++bufpos == bufsize)
        bufpos = 0;

      return buffer[bufpos];
    }

    if (++bufpos >= maxNextCharInd)
      FillBuff();

    char c = buffer[bufpos];

    UpdateLineColumn(c);
    return c;
  }

  @Deprecated
  /**
   * @deprecated
   * @see #getEndColumn
   */

  public int getColumn() {
    return bufcolumn[bufpos];
  }

  @Deprecated
  /**
   * @deprecated
   * @see #getEndLine
   */

  public int getLine() {
    return bufline[bufpos];
  }

  /** Get token end column number. */
```

```
  public int getEndColumn() {
    return bufcolumn[bufpos];
  }

  /** Get token end line number. */
  public int getEndLine() {
     return bufline[bufpos];
  }

  /** Get token beginning column number. */
  public int getBeginColumn() {
    return bufcolumn[tokenBegin];
  }

  /** Get token beginning line number. */
  public int getBeginLine() {
    return bufline[tokenBegin];
  }

/** Backup a number of characters. */
  public void backup(int amount) {

    inBuf += amount;
    if ((bufpos -= amount) < 0)
      bufpos += bufsize;
  }

  /** Constructor. */
  public SimpleCharStream(java.io.Reader dstream, int startline,
  int startcolumn, int buffersize)
  {
    inputStream = dstream;
    line = startline;
    column = startcolumn - 1;

    available = bufsize = buffersize;
    buffer = new char[buffersize];
    bufline = new int[buffersize];
    bufcolumn = new int[buffersize];
  }

  /** Constructor. */
  public SimpleCharStream(java.io.Reader dstream, int startline,
                          int startcolumn)
  {
    this(dstream, startline, startcolumn, 4096);
  }

  /** Constructor. */
  public SimpleCharStream(java.io.Reader dstream)
  {
    this(dstream, 1, 1, 4096);
  }

  /** Reinitialise. */
  public void ReInit(java.io.Reader dstream, int startline,
  int startcolumn, int buffersize)
  {
    inputStream = dstream;
    line = startline;
    column = startcolumn - 1;

    if (buffer == null || buffersize != buffer.length)
    {
      available = bufsize = buffersize;
      buffer = new char[buffersize];
      bufline = new int[buffersize];
      bufcolumn = new int[buffersize];
    }
    prevCharIsLF = prevCharIsCR = false;
    tokenBegin = inBuf = maxNextCharInd = 0;
```

```
    bufpos = -1;
  }

  /** Reinitialise. */
  public void ReInit(java.io.Reader dstream, int startline,
                     int startcolumn)
  {
    ReInit(dstream, startline, startcolumn, 4096);
  }

  /** Reinitialise. */
  public void ReInit(java.io.Reader dstream)
  {
    ReInit(dstream, 1, 1, 4096);
  }
  /** Constructor. */
  public SimpleCharStream(java.io.InputStream dstream, String encoding, int startline,
  int startcolumn, int buffersize) throws java.io.UnsupportedEncodingException
  {
    this(encoding == null ? new java.io.InputStreamReader(dstream) : new
java.io.InputStreamReader(dstream, encoding), startline, startcolumn, buffersize);
  }

  /** Constructor. */
  public SimpleCharStream(java.io.InputStream dstream, int startline,
  int startcolumn, int buffersize)
  {
    this(new java.io.InputStreamReader(dstream), startline, startcolumn, buffersize);
  }

  /** Constructor. */
  public SimpleCharStream(java.io.InputStream dstream, String encoding, int startline,
                          int startcolumn) throws java.io.UnsupportedEncodingException
  {
    this(dstream, encoding, startline, startcolumn, 4096);
  }

  /** Constructor. */
  public SimpleCharStream(java.io.InputStream dstream, int startline,
                          int startcolumn)
  {
    this(dstream, startline, startcolumn, 4096);
  }

  /** Constructor. */
  public SimpleCharStream(java.io.InputStream dstream, String encoding) throws
java.io.UnsupportedEncodingException
  {
    this(dstream, encoding, 1, 1, 4096);
  }

  /** Constructor. */
  public SimpleCharStream(java.io.InputStream dstream)
  {
    this(dstream, 1, 1, 4096);
  }

  /** Reinitialise. */
  public void ReInit(java.io.InputStream dstream, String encoding, int startline,
                          int startcolumn, int buffersize) throws
java.io.UnsupportedEncodingException
  {
    ReInit(encoding == null ? new java.io.InputStreamReader(dstream) : new
java.io.InputStreamReader(dstream, encoding), startline, startcolumn, buffersize);
  }

  /** Reinitialise. */
  public void ReInit(java.io.InputStream dstream, int startline,
                          int startcolumn, int buffersize)
  {
    ReInit(new java.io.InputStreamReader(dstream), startline, startcolumn, buffersize);
```

```
  }

  /** Reinitialise. */
  public void ReInit(java.io.InputStream dstream, String encoding) throws
java.io.UnsupportedEncodingException
  {
    ReInit(dstream, encoding, 1, 1, 4096);
  }

  /** Reinitialise. */
  public void ReInit(java.io.InputStream dstream)
  {
    ReInit(dstream, 1, 1, 4096);
  }
  /** Reinitialise. */
  public void ReInit(java.io.InputStream dstream, String encoding, int startline,
                     int startcolumn) throws java.io.UnsupportedEncodingException
  {
    ReInit(dstream, encoding, startline, startcolumn, 4096);
  }
  /** Reinitialise. */
  public void ReInit(java.io.InputStream dstream, int startline,
                     int startcolumn)
  {
    ReInit(dstream, startline, startcolumn, 4096);
  }
  /** Get token literal value. */
  public String GetImage()
  {
    if (bufpos >= tokenBegin)
      return new String(buffer, tokenBegin, bufpos - tokenBegin + 1);
    else
      return new String(buffer, tokenBegin, bufsize - tokenBegin) +
                            new String(buffer, 0, bufpos + 1);
  }

  /** Get the suffix. */
  public char[] GetSuffix(int len)
  {
    char[] ret = new char[len];

    if ((bufpos + 1) >= len)
      System.arraycopy(buffer, bufpos - len + 1, ret, 0, len);
    else
    {
      System.arraycopy(buffer, bufsize - (len - bufpos - 1), ret, 0,
                                                          len - bufpos - 1);
      System.arraycopy(buffer, 0, ret, len - bufpos - 1, bufpos + 1);
    }

    return ret;
  }

  /** Reset buffer when finished. */
  public void Done()
  {
    buffer = null;
    bufline = null;
    bufcolumn = null;
  }

  /**
   * Method to adjust line and column numbers for the start of a token.
   */
  public void adjustBeginLineColumn(int newLine, int newCol)
  {
    int start = tokenBegin;
    int len;

    if (bufpos >= tokenBegin)
    {
```

```
      len = bufpos - tokenBegin + inBuf + 1;
    }
    else
    {
      len = bufsize - tokenBegin + bufpos + 1 + inBuf;
    }

    int i = 0, j = 0, k = 0;
    int nextColDiff = 0, columnDiff = 0;

    while (i < len && bufline[j = start % bufsize] == bufline[k = ++start % bufsize])
    {
      bufline[j] = newLine;
      nextColDiff = columnDiff + bufcolumn[k] - bufcolumn[j];
      bufcolumn[j] = newCol + columnDiff;
      columnDiff = nextColDiff;
      i++;
    }

    if (i < len)
    {
      bufline[j] = newLine++;
      bufcolumn[j] = newCol + columnDiff;

      while (i++ < len)
      {
        if (bufline[j = start % bufsize] != bufline[++start % bufsize])
          bufline[j] = newLine++;
        else
          bufline[j] = newLine;
      }
    }

    line = bufline[j];
    column = bufcolumn[j];
  }

}
/* JavaCC - OriginalChecksum=92d4471c23d34d70b9cd87366cddaf16 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. SimpleNode.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class SimpleNode extends nql.cd.intermediate.NotSimpleNode implements Node {

  protected Node parent;
  protected Node[] children;
  protected int id;
  protected Object value;
  protected CParser parser;

  public SimpleNode(int i) {
    id = i;
  }

  public SimpleNode(CParser p, int i) {
    this(i);
    parser = p;
  }

  public void jjtOpen() {
  }

  public void jjtClose() {
  }
```

```
  public void jjtSetParent(Node n) { parent = n; }
  public Node jjtGetParent() { return parent; }

  public void jjtAddChild(Node n, int i) {
    if (children == null) {
      children = new Node[i + 1];
    } else if (i >= children.length) {
      Node c[] = new Node[i + 1];
      System.arraycopy(children, 0, c, 0, children.length);
      children = c;
    }
    children[i] = n;
  }

  public Node jjtGetChild(int i) {
    return children[i];
  }

  public int jjtGetNumChildren() {
    return (children == null) ? 0 : children.length;
  }

  public void jjtSetValue(Object value) { this.value = value; }
  public Object jjtGetValue() { return value; }

  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data)
{
    return visitor.visit(this, data);
  }

  /** Accept the visitor. **/
  public Object childrenAccept(CParserVisitor visitor, Object data)
{
    if (children != null) {
      for (int i = 0; i < children.length; ++i) {
        children[i].jjtAccept(visitor, data);
      }
    }
    return data;
  }

  /* You can override these two methods in subclasses of SimpleNode to
     customize the way the node appears when the tree is dumped.  If
     your output uses more than one line you should override
     toString(String), otherwise overriding toString() is probably all
     you need to do. */

  public String toString() { return CParserTreeConstants.jjtNodeName[id]; }
  public String toString(String prefix) { return prefix + toString(); }

  /* Override this method if you want to customize how the node dumps
     out its children. */

  public void dump(String prefix, StringBuffer sb) {
    sb.append(toString(prefix) + "\n");
    if (children != null) {
      for (int i = 0; i < children.length; ++i) {
        SimpleNode n = (SimpleNode)children[i];
        if (n != null) {
          n.dump(prefix + "|    ", sb);
        }
      }
    }
  }
}

/* JavaCC - OriginalChecksum=39f6635b8a7d8045f895c3ee71fcd5d2 (do not edit this line) */

/* Generated By:JavaCC: Do not edit this line. Token.java Version 5.0 */
```

```
/* JavaCCOptions:TOKEN_EXTENDS=,KEEP_LINE_COL=null,SUPPORT_CLASS_VISIBILITY_PUBLIC=true
*/
package nql.cd.frontend;

/**
 * Describes the input token stream.
 */

public class Token implements java.io.Serializable {

  /**
   * The version identifier for this Serializable class.
   * Increment only if the <i>serialized</i> form of the
   * class changes.
   */
  private static final long serialVersionUID = 1L;

  /**
   * An integer that describes the kind of this token.  This numbering
   * system is determined by JavaCCParser, and a table of these numbers is
   * stored in the file ...Constants.java.
   */
  public int kind;

  /** The line number of the first character of this Token. */
  public int beginLine;
  /** The column number of the first character of this Token. */
  public int beginColumn;
  /** The line number of the last character of this Token. */
  public int endLine;
  /** The column number of the last character of this Token. */
  public int endColumn;

  /**
   * The string image of the token.
   */
  public String image;

  /**
   * A reference to the next regular (non-special) token from the input
   * stream.  If this is the last token from the input stream, or if the
   * token manager has not read tokens beyond this one, this field is
   * set to null.  This is true only if this token is also a regular
   * token.  Otherwise, see below for a description of the contents of
   * this field.
   */
  public Token next;

  /**
   * This field is used to access special tokens that occur prior to this
   * token, but after the immediately preceding regular (non-special) token.
   * If there are no such special tokens, this field is set to null.
   * When there are more than one such special token, this field refers
   * to the last of these special tokens, which in turn refers to the next
   * previous special token through its specialToken field, and so on
   * until the first special token (whose specialToken field is null).
   * The next fields of special tokens refer to other special tokens that
   * immediately follow it (without an intervening regular token).  If there
   * is no such token, this field is null.
   */
  public Token specialToken;

  /**
   * An optional attribute value of the Token.
   * Tokens which are not used as syntactic sugar will often contain
   * meaningful values that will be used later on by the compiler or
   * interpreter. This attribute value is often different from the image.
   * Any subclass of Token that actually wants to return a non-null value can
   * override this method as appropriate.
   */
  public Object getValue() {
```

```
    return null;
  }

  /**
   * No-argument constructor
   */
  public Token() {}

  /**
   * Constructs a new token for the specified Image.
   */
  public Token(int kind)
  {
    this(kind, null);
  }

  /**
   * Constructs a new token for the specified Image and Kind.
   */
  public Token(int kind, String image)
  {
    this.kind = kind;
    this.image = image;
  }

  /**
   * Returns the image.
   */
  public String toString()
  {
    return image;
  }

  /**
   * Returns a new Token object, by default. However, if you want, you
   * can create and return subclass objects based on the value of ofKind.
   * Simply add the cases to the switch for all those special cases.
   * For example, if you have a subclass of Token called IDToken that
   * you want to create if ofKind is ID, simply add something like :
   *
   *    case MyParserConstants.ID : return new IDToken(ofKind, image);
   *
   * to the following switch statement. Then you can cast matchedToken
   * variable to the appropriate type and use sit in your lexical actions.
   */
  public static Token newToken(int ofKind, String image)
  {
    switch(ofKind)
    {
      default : return new Token(ofKind, image);
    }
  }

  public static Token newToken(int ofKind)
  {
    return newToken(ofKind, null);
  }

}
/* JavaCC - OriginalChecksum=3c2217312c61719164f6466472d88e57 (do not edit this line) */

/* Generated By:JavaCC: Do not edit this line. TokenMgrError.java Version 5.0 */
/* JavaCCOptions: */
package nql.cd.frontend;

/** Token Manager Error. */
public class TokenMgrError extends Error
{

  /**
   * The version identifier for this Serializable class.
```

```
 * Increment only if the <i>serialized</i> form of the
 * class changes.
 */
private static final long serialVersionUID = 1L;

/*
 * Ordinals for various reasons why an Error of this type can be thrown.
 */

/**
 * Lexical error occurred.
 */
static final int LEXICAL_ERROR = 0;

/**
 * An attempt was made to create a second instance of a static token manager.
 */
static final int STATIC_LEXER_ERROR = 1;

/**
 * Tried to change to an invalid lexical state.
 */
static final int INVALID_LEXICAL_STATE = 2;

/**
 * Detected (and bailed out of) an infinite loop in the token manager.
 */
static final int LOOP_DETECTED = 3;

/**
 * Indicates the reason why the exception is thrown. It will have
 * one of the above 4 values.
 */
int errorCode;

/**
 * Replaces unprintable characters by their escaped (or unicode escaped)
 * equivalents in the given string
 */
protected static final String addEscapes(String str) {
  StringBuffer retval = new StringBuffer();
  char ch;
  for (int i = 0; i < str.length(); i++) {
    switch (str.charAt(i))
    {
      case 0 :
        continue;
      case '\b':
        retval.append("\\b");
        continue;
      case '\t':
        retval.append("\\t");
        continue;
      case '\n':
        retval.append("\\n");
        continue;
      case '\f':
        retval.append("\\f");
        continue;
      case '\r':
        retval.append("\\r");
        continue;
      case '\"':
        retval.append("\\\"");
        continue;
      case '\'':
        retval.append("\\\'");
        continue;
      case '\\':
        retval.append("\\\\");
        continue;
```

```
        default:
          if ((ch = str.charAt(i)) < 0x20 || ch > 0x7e) {
            String s = "0000" + Integer.toString(ch, 16);
            retval.append("\\u" + s.substring(s.length() - 4, s.length()));
          } else {
            retval.append(ch);
          }
          continue;
       }
     }
     return retval.toString();
  }

  /**
   * Returns a detailed message for the Error when it is thrown by the
   * token manager to indicate a lexical error.
   * Parameters :
   *    EOFSeen     : indicates if EOF caused the lexical error
   *    curLexState : lexical state in which this error occurred
   *    errorLine   : line number when the error occurred
   *    errorColumn : column number when the error occurred
   *    errorAfter  : prefix that was seen before this error occurred
   *    curchar     : the offending character
   * Note: You can customize the lexical error message by modifying this method.
   */
  protected static String LexicalError(boolean EOFSeen, int lexState, int errorLine, int
errorColumn, String errorAfter, char curChar) {
    return("Lexical error at line " +
          errorLine + ", column " +
          errorColumn + ".  Encountered: " +
          (EOFSeen ? "<EOF> " : ("\"" + addEscapes(String.valueOf(curChar)) + "\"") + "
(" + (int)curChar + "), ") +
          "after : \"" + addEscapes(errorAfter) + "\"");
  }

  /**
   * You can also modify the body of this method to customize your error messages.
   * For example, cases like LOOP_DETECTED and INVALID_LEXICAL_STATE are not
   * of end-users concern, so you can return something like :
   *
   *     "Internal Error : Please file a bug report .... "
   *
   * from this method for such cases in the release version of your parser.
   */
  public String getMessage() {
    return super.getMessage();
  }

  /*
   * Constructors of various flavors follow.
   */

  /** No arg constructor. */
  public TokenMgrError() {
  }

  /** Constructor with message and reason. */
  public TokenMgrError(String message, int reason) {
    super(message);
    errorCode = reason;
  }

  /** Full Constructor. */
  public TokenMgrError(boolean EOFSeen, int lexState, int errorLine, int errorColumn,
String errorAfter, char curChar, int reason) {
    this(LexicalError(EOFSeen, lexState, errorLine, errorColumn, errorAfter, curChar),
reason);
  }
}
/* JavaCC - OriginalChecksum=d469e12a941be6cc53538205c8d7062d (do not edit this line) */
```

```
/* Generated By:JJTree: Do not edit this line. ASTabstractDeclarator.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTabstractDeclarator extends SimpleNode {
  public ASTabstractDeclarator(int id) {
    super(id);
  }

  public ASTabstractDeclarator(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=f2d89110590caec4a789ab8860b38b8d (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTadditiveExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTadditiveExpression extends SimpleNode {
  public ASTadditiveExpression(int id) {
    super(id);
  }

  public ASTadditiveExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=61b958300658079dd5d6c8c57612422e (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTassignmentExpression.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTassignmentExpression extends SimpleNode {
  public ASTassignmentExpression(int id) {
    super(id);
  }

  public ASTassignmentExpression(CParser p, int id) {
    super(p, id);
  }
```

```
  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=dd9b6a7484c1a13fc3d76a46213e5e1a (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTbitwiseANDExpression.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTbitwiseANDExpression extends SimpleNode {
  public ASTbitwiseANDExpression(int id) {
    super(id);
  }

  public ASTbitwiseANDExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=1f056d1b74f8665edce44546f9c407a6 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTbitwiseORExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTbitwiseORExpression extends SimpleNode {
  public ASTbitwiseORExpression(int id) {
    super(id);
  }

  public ASTbitwiseORExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=1bb9ed185d4626be71523cdaf69cfffb (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTbitwiseXORExpression.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTbitwiseXORExpression extends SimpleNode {
  public ASTbitwiseXORExpression(int id) {
    super(id);
  }
```

```
  public ASTbitwiseXORExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=174138aa5f471eaa5d1244b99408b3b6 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTcastExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTcastExpression extends SimpleNode {
  public ASTcastExpression(int id) {
    super(id);
  }

  public ASTcastExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=16f0331e8d24f6dd1573aa7ef5af980b (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTcharacter.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTcharacter extends SimpleNode {
  public ASTcharacter(int id) {
    super(id);
  }

  public ASTcharacter(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=efd4d49094c97d92ab5e71470cd1140f (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTcompoundStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
```

```
class ASTcompoundStatement extends SimpleNode {
  public ASTcompoundStatement(int id) {
    super(id);
  }

  public ASTcompoundStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=ca5862bc4834a97d09d0d3c7ac7bf950 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTconditionalExpression.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTconditionalExpression extends SimpleNode {
  public ASTconditionalExpression(int id) {
    super(id);
  }

  public ASTconditionalExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=4156d1d230c0ec06e5cbd5fe783f909e (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTdeclaration.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTdeclaration extends SimpleNode {
  public ASTdeclaration(int id) {
    super(id);
  }

  public ASTdeclaration(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=31cc4ab17ca20e12ace0771b5f3ab3ea (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTdeclarationList.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
```

```
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTdeclarationList extends SimpleNode {
  public ASTdeclarationList(int id) {
    super(id);
  }

  public ASTdeclarationList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=059a8efb0c916179dafa8d6bacf64bbf (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTdeclarationSpecifiers.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTdeclarationSpecifiers extends SimpleNode {
  public ASTdeclarationSpecifiers(int id) {
    super(id);
  }

  public ASTdeclarationSpecifiers(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=551b9e9799ab057e6d22a131611fff52 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTdeclarator.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTdeclarator extends SimpleNode {
  public ASTdeclarator(int id) {
    super(id);
  }

  public ASTdeclarator(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
```

```
/* JavaCC - OriginalChecksum=c3f9e873ab385d32b667b73107d82855 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTdirectAbstractDeclarator.java Version
4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTdirectAbstractDeclarator extends SimpleNode {
  public ASTdirectAbstractDeclarator(int id) {
    super(id);
  }

  public ASTdirectAbstractDeclarator(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=1f97d16c52f0411e7df8ce2dc2eace68 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTdirectDeclarator.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTdirectDeclarator extends SimpleNode {
  public ASTdirectDeclarator(int id) {
    super(id);
  }

  public ASTdirectDeclarator(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=50cc1be60afa82bcc9ed86230e03c3c9 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTdoWhileStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTdoWhileStatement extends SimpleNode {
  public ASTdoWhileStatement(int id) {
    super(id);
  }

  public ASTdoWhileStatement(CParser p, int id) {
    super(p, id);
  }
```

```
  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=a93efb3d8a51e3a8d6fb5a3b7a530089 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTenumerator.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTenumerator extends SimpleNode {
  public ASTenumerator(int id) {
    super(id);
  }

  public ASTenumerator(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=af21416fbc9b4e55aa4ae2950fe1f401 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTenumeratorList.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTenumeratorList extends SimpleNode {
  public ASTenumeratorList(int id) {
    super(id);
  }

  public ASTenumeratorList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=66a5cea07f71b60a175d957ffc2d98c5 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTenumSpecifier.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTenumSpecifier extends SimpleNode {
  public ASTenumSpecifier(int id) {
    super(id);
  }

  public ASTenumSpecifier(CParser p, int id) {
```

```
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=73a2aba19042bd3a51a89fb9e9f5a890 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTequalityExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTequalityExpression extends SimpleNode {
  public ASTequalityExpression(int id) {
    super(id);
  }

  public ASTequalityExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=e6a94a9863753cee0df2c9a779c73e00 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTexpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTexpression extends SimpleNode {
  public ASTexpression(int id) {
    super(id);
  }

  public ASTexpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=1106df6bd80c656c100370811934dec1 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTexpressionStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTexpressionStatement extends SimpleNode {
  public ASTexpressionStatement(int id) {
```

```
    super(id);
  }

  public ASTexpressionStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=e593af7f82e68101c7e2cf2724d04096 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTexternalDeclaration.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTexternalDeclaration extends SimpleNode {
  public ASTexternalDeclaration(int id) {
    super(id);
  }

  public ASTexternalDeclaration(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=b0c20d6a625bed69f2a0250dceb46d43 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTforStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTforStatement extends SimpleNode {
  public ASTforStatement(int id) {
    super(id);
  }

  public ASTforStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=927ddf629b836ab47ded7368edf1f6c1 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTfunctionDefinition.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;
```

```
public
class ASTfunctionDefinition extends SimpleNode {
  public ASTfunctionDefinition(int id) {
    super(id);
  }

  public ASTfunctionDefinition(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=7fc27a6947a3698c02b51aa35d96a13c (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTfunctionDefinitionLookahead.java
Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTfunctionDefinitionLookahead extends SimpleNode {
  public ASTfunctionDefinitionLookahead(int id) {
    super(id);
  }

  public ASTfunctionDefinitionLookahead(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=088dd8c0c58f4ecd57e6f38f2e6058fa (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTidentifier.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTidentifier extends SimpleNode {
  public ASTidentifier(int id) {
    super(id);
  }

  public ASTidentifier(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=b7a3ba6ad1159de53fb44fe4121b6cd8 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTifStatement.java Version 4.3 */
```

```
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTifStatement extends SimpleNode {
  public ASTifStatement(int id) {
    super(id);
  }

  public ASTifStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=4fa0dc111881da33f8445ad3a9273f5f (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTinitDeclarator.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTinitDeclarator extends SimpleNode {
  public ASTinitDeclarator(int id) {
    super(id);
  }

  public ASTinitDeclarator(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=a6a276fcfc33f0e4b08fd9824e1165d9 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTinitDeclaratorList.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTinitDeclaratorList extends SimpleNode {
  public ASTinitDeclaratorList(int id) {
    super(id);
  }

  public ASTinitDeclaratorList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
```

```
}
/* JavaCC - OriginalChecksum=b8ea1d188161403bd7f9e38781271aeb (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTinitializer.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTinitializer extends SimpleNode {
  public ASTinitializer(int id) {
    super(id);
  }

  public ASTinitializer(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=ab896275d7d0fd8bd84229cf27ede50d (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTinitializerList.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTinitializerList extends SimpleNode {
  public ASTinitializerList(int id) {
    super(id);
  }

  public ASTinitializerList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=a924db4a5f0a53bb6fb23a7cd79ec4c0 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTinteger.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTinteger extends SimpleNode {
  public ASTinteger(int id) {
    super(id);
  }

  public ASTinteger(CParser p, int id) {
    super(p, id);
  }
```

```
  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=5e07f3497c44860cb787c0ae605d5e66 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTjumpStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTjumpStatement extends SimpleNode {
  public ASTjumpStatement(int id) {
    super(id);
  }

  public ASTjumpStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=918d4e83d9057160ef4d1a36b972dd2e (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTlabeledStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTlabeledStatement extends SimpleNode {
  public ASTlabeledStatement(int id) {
    super(id);
  }

  public ASTlabeledStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=f8cf18a4338bf6cb726ba38c922e561b (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTlogicalANDExpression.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTlogicalANDExpression extends SimpleNode {
  public ASTlogicalANDExpression(int id) {
    super(id);
  }
```

```
  public ASTlogicalANDExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=bcc6a04b19b8a9d73dd2808a2dbddb84 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTlogicalORExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTlogicalORExpression extends SimpleNode {
  public ASTlogicalORExpression(int id) {
    super(id);
  }

  public ASTlogicalORExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=ad4dd2d615e622850713600feba98a72 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTmultiplicativeExpression.java Version
4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTmultiplicativeExpression extends SimpleNode {
  public ASTmultiplicativeExpression(int id) {
    super(id);
  }

  public ASTmultiplicativeExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=4cf4480d9d29a46c8cd251abba3bcc5c (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTparameterDeclaration.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;
```

```
public
class ASTparameterDeclaration extends SimpleNode {
  public ASTparameterDeclaration(int id) {
    super(id);
  }

  public ASTparameterDeclaration(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=b6201ce6651975d8e68ae2e3d42c9129 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTparameterList.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTparameterList extends SimpleNode {
  public ASTparameterList(int id) {
    super(id);
  }

  public ASTparameterList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=fa976975d65e365ccf97cd36f5648609 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTparameterTypeList.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTparameterTypeList extends SimpleNode {
  public ASTparameterTypeList(int id) {
    super(id);
  }

  public ASTparameterTypeList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=c84bea334d6b34928db5fea1a51b9861 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTparse.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
```

```
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTparse extends SimpleNode {
  public ASTparse(int id) {
    super(id);
  }

  public ASTparse(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=f74f11123c5e59144d15f9d7308a3430 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTpointer.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTpointer extends SimpleNode {
  public ASTpointer(int id) {
    super(id);
  }

  public ASTpointer(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=44fa56088df6ae942babb87247f442fe (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTpostfixExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTpostfixExpression extends SimpleNode {
  public ASTpostfixExpression(int id) {
    super(id);
  }

  public ASTpostfixExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=4e3d946770313708b67b84ffa2bdc3de (do not edit this line) */
```

```
/* Generated By:JJTree: Do not edit this line. ASTreal.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTreal extends SimpleNode {
  public ASTreal(int id) {
    super(id);
  }

  public ASTreal(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=8bbaf117fe5e6b09c872b29930edd035 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTrelationalExpression.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTrelationalExpression extends SimpleNode {
  public ASTrelationalExpression(int id) {
    super(id);
  }

  public ASTrelationalExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=264406fe0ef2ada2fdbe2c3e3b6636e7 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTshiftExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTshiftExpression extends SimpleNode {
  public ASTshiftExpression(int id) {
    super(id);
  }

  public ASTshiftExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
```

```
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=f68f81776c3e0d1d5232fce9f080c58e (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstorageClassSpecifier.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTstorageClassSpecifier extends SimpleNode {
  public ASTstorageClassSpecifier(int id) {
    super(id);
  }

  public ASTstorageClassSpecifier(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=91b9d52a56826f0f37ea9f14dbf1b6fa (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstring.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTstring extends SimpleNode {
  public ASTstring(int id) {
    super(id);
  }

  public ASTstring(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=c08180f14063ee7ed42cb543eec25c56 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstructDeclaration.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTstructDeclaration extends SimpleNode {
  public ASTstructDeclaration(int id) {
    super(id);
  }

  public ASTstructDeclaration(CParser p, int id) {
```

```
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=d7bae3111812ea89984e02b1c7ebec8e (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstructDeclarationList.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTstructDeclarationList extends SimpleNode {
  public ASTstructDeclarationList(int id) {
    super(id);
  }

  public ASTstructDeclarationList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=0c1780b399f19225d4bc9de446092a1e (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstructDeclarator.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTstructDeclarator extends SimpleNode {
  public ASTstructDeclarator(int id) {
    super(id);
  }

  public ASTstructDeclarator(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=024a3a3917f02e4a990c4cda38a5c4c5 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstructDeclaratorList.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
```

```
class ASTstructDeclaratorList extends SimpleNode {
  public ASTstructDeclaratorList(int id) {
    super(id);
  }

  public ASTstructDeclaratorList(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=146c743da2314418fa0cd563337e06df (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstructOrUnion.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTstructOrUnion extends SimpleNode {
  public ASTstructOrUnion(int id) {
    super(id);
  }

  public ASTstructOrUnion(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=cbbca209eaf4ac53c19918d987d879d0 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTstructOrUnionSpecifier.java Version 4.3
*/
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTstructOrUnionSpecifier extends SimpleNode {
  public ASTstructOrUnionSpecifier(int id) {
    super(id);
  }

  public ASTstructOrUnionSpecifier(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=c5accd063e81dfe3332450f030411f5f (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTswitchStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
```

```
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTswitchStatement extends SimpleNode {
  public ASTswitchStatement(int id) {
    super(id);
  }

  public ASTswitchStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=4a7c549318daec58676eea53c9c8a638 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTtypedefName.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTtypedefName extends SimpleNode {
  public ASTtypedefName(int id) {
    super(id);
  }

  public ASTtypedefName(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=2b9f17a79deb04be424f1166f8357fce (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTtypeName.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTtypeName extends SimpleNode {
  public ASTtypeName(int id) {
    super(id);
  }

  public ASTtypeName(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=da8ef55337a3bcf71f707192f0845c82 (do not edit this line) */
```

```
/* Generated By:JJTree: Do not edit this line. ASTtypeQualifier.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTtypeQualifier extends SimpleNode {
  public ASTtypeQualifier(int id) {
    super(id);
  }

  public ASTtypeQualifier(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=6124d1808c8b5333cccc8acee51d0a8d (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTtypeSpecifier.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTtypeSpecifier extends SimpleNode {
  public ASTtypeSpecifier(int id) {
    super(id);
  }

  public ASTtypeSpecifier(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=403f0ab3d2b0e068d9a6ada1bf4d0c25 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTunaryExpression.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTunaryExpression extends SimpleNode {
  public ASTunaryExpression(int id) {
    super(id);
  }

  public ASTunaryExpression(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
```

```
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=6d21407a2e1ea41c2c3bcb9942e9a8a8 (do not edit this line) */

/* Generated By:JJTree: Do not edit this line. ASTwhileStatement.java Version 4.3 */
/*
JavaCCOptions:MULTI=true,NODE_USES_PARSER=false,VISITOR=true,TRACK_TOKENS=false,NODE_PREF
IX=AST,NODE_EXTENDS=nql.cd.intermediate.NotSimpleNode,NODE_FACTORY=,SUPPORT_CLASS_VISIBIL
ITY_PUBLIC=true */
package nql.cd.frontend;

public
class ASTwhileStatement extends SimpleNode {
  public ASTwhileStatement(int id) {
    super(id);
  }

  public ASTwhileStatement(CParser p, int id) {
    super(p, id);
  }


  /** Accept the visitor. **/
  public Object jjtAccept(CParserVisitor visitor, Object data) {
    return visitor.visit(this, data);
  }
}
/* JavaCC - OriginalChecksum=e7a4597355f34ce32c2befa9dfdde0a4 (do not edit this line) */
```