XQ: An XML Query Language

Project Report

Kin Ng
kn2006@columbia.edu
December 17, 2003

Contents

1 Introduction

1.1 Background

The Extensible Markup Language (XML) is a markup language for documents containing structured information dervied from SGML (ISO 8879). The goal of XML is to provide many of SGML's benefits not available in Hyper Text Markup Language (HTML) and to provide them in a language that is easier to learn and use than complete SGML. In fact, XML is really a meta-language to provide a facility to define tags and the strutural relationship between them. Unlike HTML, XML has no pre-defined tag set and any preconceived sematics. All of the semantics of an XML document will either be defined will either be defined by the applications that process them or by stylesheets.

Over the past few years, XML has rapidly gained popularity as a formatting language for information. Key application areas for XML include information management, database management, e-commerce. In fact, Simple Object Access Protocol (SOAP), one of the fastest growing protocols adopted by many application servers, provides an open, extensible way for applications to communicate using XML-based messages over Hyper Text Transfer Protocol (HTTP). The simplicity and extensibility of XML allows SOAP to bridge heterogeneous components over the Internet and provides an open mechanism for frameworks that provide Web services

As more businesses choose to provide access to their databases and to exchange data with related businesses and organizations. We take a database view, as opposed to document view, of XML: we consider an XML document to be a database and a DTD to be a database schema. Howerver, XML standard does not address what techniques should be used for data extracting. XML query languages are needed to address this issue.

In this project, we are present a programming language, XQ, as a small trail in design and implementing of programming language for XML queries.

1.2 Related Works

Over the past few years, a lot of effort was dedicated to XML query languages research. A few languages were proposed. The commonly known ones are: XQL[1], XML-QL[2] and QUILT[3]. Their functionalities and syntax are quite similar. Currently, the W3C picked XQuery[4] (dervied from QUILT) as the query language of XML. The language is still a work in progress under the auspices of the W3C's XML Query working group. The syntax of XQuery is similar to other query language such as SQL[5].

2 Project Overview

2.1 Purpose

The purpose of XQ is to provide programmers with a simple-to-use XML document query language that allow programmers to express queries in a few simple statements.

2.2 Objectives

The main goal of the project is to sucessfully design and implement a XML query language. This language will be clean semantic, easy to setup, efficient and expandable.

2.2.1 Clean Semantic

The syntax of this language will able to express simple queries simply. In ordre to this objective, XQ has a C-like syntax. Since the syntax is based on a well-known and well-structure language. This language will be easily pickup by an experience database programming as well as a noivce application prgrammer. This will not increase the training and development cost and any XQ projects

2.2.2 Easy to Setup

This lauguage will limit its external dependence. Only minimum software and hardware will be required to installation to the language. Since the langauge is written in Java, the machine that execute XQ code must have Java 2 Runtime Environment setup.

2.2.3 Expandility

The implementation of the language will use a component-based design methodology. It will simplify the process of adding future enhancements.

2.3 Features

Besides the design goals we mentioned above, there are a few main features we will include in this language.

2.3.1 Data types

This laugauge will support data type such as *int*, *double*, *string*, *cursor* and *document.* The *cursor* data type is used to hold the result set from the *select* operation. The *document* type is used to represent the XML documents.

2.3.2 Control flow

The flows of control of this language will consist of selection statements such as *if-else, else-if* , iteration statements such *while* and *for* ,and jump statements such as *break* and *continue.*

2.3.3 Ability to return an XML

This language will able to return the result in XML format.

2.3.4 Ability to query attributes:

This language will allow programmers to query the attributes as well as the payload.

2.3.5 Preserve order

This language will allow programmers to use the *order by* clause to sort the result set in a particular order.

2.3.6 Projection and selection queries:

This lauguage will support projection and selection queries. Insert and update qeuries are not included this language.

2.3.7 Functions

This language will provide a number of internal functions such as *print.* It also will support user define functions.

2.3.8 Comment

C style comment[6] will be used for this language.

3 Tutorial

In this section, we are going to provide a sample program that uses XQ. This example is to serve as a tutorial on the usage of XQ. For detail language syntax specification, please refer to the language reference manual section of the document.

3.1 Sample Code

```
/* start of the sample code */
/* start of variable declaration section */
cursor cur;
string name;
int ssn;
double salary;
document doc;
/* end of variable declaration section */

/* start of the program statement body */
/* create document handler using data and DTD files */
load( "/home/johnd/students.txt ,"doc);

SELECT doc.ssn, doc.name, doc.salary FROM doc WHERE ssn = 123456789 USING
cur;

for (; ;) {
 FETCH cur INTO rcode, ssn, doc, salary;
  if (rcode != 100) {
          break;
  }
  print(ssn, name, salary);
}

/* start of the program statement body */
/* end of the sample code */
```

3.2 Sample Code Analysis

Let s'look at the same code in detial. All XQ program are divided into 2 sections: the variable declaration section and the program body section. Program must start with the variable declaration section and follow by the program body section.

Variables declaration section contain varible declaration statements only. All varibles must be declared using the variable declaration statements. Variable must

belong to one of the following data types: *cursor, string, int, double* and *document.* In this sample code, variables for all 5 data types are declared.

The program enters the the program body section after the last variable declaration statement. In general, the program body section is consist of 3 type of statements: the LOAD statement; the SELECT statement; and the FETCH statement. These three statements are used together to perform queries on XML documents.

First, the LOAD statements are used to bind XML documents with document varibles with document variables. After the LOAD statements, the document varibles have all information regarding the XML document.

Then the SELECT statements are used to run queries using the document variable and store the result sets into cursor variables. The SELECT statement has 4 clauses: the SELECT clause; the FROM clause; the WHERE clause; and the USING clause. The SELECT clause is used to specify which fields from the XML should be returned in the result set. The field list specify by the SELECT clause also control how variables should be binded in the FETCH statement. The WHERE clause is used specify which document variable should the query to run against. The FROM clause is used to specify the selection criteria of the result set. The USING clause is used to bind the result set to a cursor variable.

Finally, the FETCH statements are used to fetch information from cursor variables. The FETCH statement can also be divided into 2 clauses: the FETCH clause and the INTO clause. The FETCH clause takes a cursor variable so it can prepare to access the result set stored in the cursor variable. The INTO clause is used to bind a list of variables with the result set. The variable list first start with a status variable to return the status of the fetch operation. Then the rest of the variable list should will be binded to the field list specify by SELECT clause that generates the cursor. Value of the each field in the result set will fetch into the variable list.

4 Language Reference Manual

4.1 Lexical Conventions

4.1.1 Comments

A comment starts with the characters /* and terminates with the characters */. Comments do not nest, and they do not occur within string or character literals.

4.1.2 Identifiers

An identifiers is a series of letters and digits with the first character being a letter or the underscore. Upper and lower case letters are distinct.

4.1.3 Keywords

The following identifiers are keywords:

ASC	DISTINCT	Integer	SQLCODE
AVERAGE	else	INTO	String
break	FETCH	MAX	SUM
BY	document	MIN	Tag
CLOSE	float	NULL	WHERE
continue	for	OPEN	While
COUNT	FROM	ORDER	WITH
Cursor	function	Return	
DESC	if	SELECT	

4.1.4 Constants

4.1.4.1 Integer Constants

An integer is a series of digits is taken to be decimal.

4.1.4.2 Floating Constants

A floating constants consists of an integer part, a decimal point, a fraction part, and e or E and an optionally signed integer exponent. Either the integer part or the fraction part (not both) may be missing; either the decimal part of the e and the exponent (not both) may be missing.

4.1.4.3 String Literals

A string literals is a series of characters enclosed by double quotes, as in ". . . ". A double quote inside a string literal is represented by \ ".

4.1.5 Other Tokens

The following are the remaining tokens:

{	}	(	)	[	]	,	;	+	-
*	/	%	=	+=	-=	*=	/=	>=	<=
>	<	==	!=	!	&&	||	:	&	

4.2 Meaning of Identifies

4.2.1 Types

In this language, all variables are statically typed and must be declared before they are used. Types that can be interpreted as numbers will be referred to as arithmetic type. There are several basic types including:

cursor:	pointer to navigate within the query result set
document:	pointer to the data structures to hold the XML document
double:	64-bit IEEE floating point numbers
int:	32-bit integers
string:	sequence of characters

4.2.2 Scope

There are two kinds of scope: block and file. The scope of an identifier begins when its declaration is seen. Although it is impossible to have two declarations of the same identifier active in the same scope, no conflict occurs if the instances are in different scopes.

4.3 Conversions

4.3.1 Integer and Floating

When a floating value is converted to an integral value, the rounded value is preserved as long as it does not overflow. When an integral value is converted to a floating value, the value is preserved.

4.4 Expressions and Operators

4.4.1 Primary Expressions

4.4.1.1 Identifiers

An identifier is an lvalue expression.

4.4.1.2 Constants

A constant's type is determined by its form and value.

4.4.1.3 String Literals

A string literal's type is *string*.

4.4.1.4 Parenthesized Expressions

A parenthesized expressions' type and value are identical to those of the un-parenthesized expressions.

4.4.1.5 Function Calls

A function call contains a function identifier and a (possibly empty) comma-separated list of expressions that are the arguments to the function.

4.4.2 Arithmetic Expressions

4.4.2.1 Unary Operators

Expressions with unary operators associate from right to left. The result of the unary – operator is the negative of its operand. The result of the unary operator + is the value of the operand. The operand must be arithmetic type.

4.4.2.2 Multiplicative Operators

The multiplicative operators *, /, and % group from left to right. Operands of * and / must have arithmetic type. Operands of % must have integral type. The binary operators *, / and % represents multiplication, division and modulo.

4.4.2.3 Additive Operators

The additive operators + and - associate from left to right. The result of the + operator is the sum of the operands. The result of the - operator is the difference of the operands. The operands must be both arithmetic type.

4.4.2.4 Relational Expressions

The relational operators associate from left to right. The operators <, >, <=, >=, == and != represents greater than, less than or equal to, greater than or equal to, equal to and not equal to respectively. They all yield a result of type *integer* with value 0 if the specific relation is false and 1 if it is true.

4.4.2.5 Logical Expressions

The unary operator ! is the logical negation. Its result is 1 if the value of its operand is 0 and vice versa. The && and || operators represent logical AND and OR. The result from both operators has type integer. For && operator, if neither of the operands evaluates to 0, the result has a value of 1. Otherwise it has a value of 0. For || operator, if neither of the operands evaluates to 0, the result has a value of 1. Otherwise it has a value of 0.

4.4.2.6 Assignment Expressions

Assignment operators consists =, *=, /= , %=, +=, -= . They are all of associate from right to left. Assignment operators require a modifiable lvalue as their left operand. The type of an assignment expression is that of its unqualified left operand. The result is not an lvalue. Its value is the value stored in the left operand after the assignment.

4.5 Statements

A statement is a complete instruction to the computer. Statements are executed in sequence.

4.5.1 Expression Statements

Most statements are expression statements. They are evaluated for their side effects, such as assignments or function calls.

4.5.2 Conditional Statements

The conditional statements have the following form:

```
if (expression)
  statement
```

or

```
if (expression)
  statement
else
  statement
```

Conditional statements choose one of a set of statements to execute, based on the evaluation of the expression. The expression is the controlling expression. For both forms of the `if` statement, the first statement is executed if the controlling expression evaluates to nonzero. For the second form, the second statement is executed if the controlling expression evaluates to zero. An `else` clause that follows multiple sequential `else-less if` statements is associated with the most recent `if` statement in the same block.

4.5.3 Iterative Statements

Iteration statements execute the attached statement repeatedly until the controlling expression evaluates to zero.

4.5.3.1 For Statement

The *for* statement has the following form:

```
for (expression; expression; expression)
  statement
```

The first expression specifies initialization for the loop. The second expression is the controlling expression which is evaluated before each iteration. The third expression often specifies incrementation which is evaluated after each iteration.

4.5.3.2 While Statement

The *while* statement has the following form:

```
while (expression)
  statement
```

The controlling expression of a *while* statement is evaluated before each execution of the body.

4.5.3.3 Break Statement

The *break* statement can appear only in the body of an iteration statement. It transfers control to the statement immediately following the smallest enclosing iteration.

4.5.3.4 Continue Statement

The *continue* statement can appear only in the body of an iteration statement. It causes the *while* or *for* statement to the end of the loop.

4.5.3.5 Function Return Statement

A function returns to its caller by the *return* statement. When *return* is followed by an expression, the value is returned to the caller fo the function.

4.5.4 Query Statements

4.5.4.1 SELECT Statement

SELECT statement is used to specify the query operation and store the result set into a cursor-identifier.
The SELECT statement has the following form:

SELECT sel-clause
FROM doc-identifier
WHERE expression order-by-clause USING cursor-identifier

The *sel-clause* consists of a *list of identifiers* and performs a projection on the target document. Operational key words such as *AVERAGE, COUNT, SUM, MIN* and *MAX* can be applied to the *sel-clause* to perform aggregate operations. Another keyword *DISTINCT* can be used so there will not be duplicates in the result set.
The *where-clause* consists of expression and follows by an optional *order-by-clause*. The *where-clause* expression is used to perform a selection on the target document. The optional *order-by-clause* has the following form:

ORDER BY list of tag-identifiers ASC/DESC

The *order-by-clause* is used to sort the result set in order to retrieve in a particular order. *ASC* and *DESC* are used to represent ascending and descending order respectively.

The result set generate by the SELECT statement is stored in the *cusror-identifer* after the *USING* keyword.

4.5.4.2 FETCH Statement

The fetch statement has the following form:

FETCH cursor-identifier INTO identifier-list
The fetch statement retrieve information from result set through the a cursor type identifier. The result is loaded into a list of identifiers.

4.6 Declarations

A declaration specifies the interpretation given to a set of identifiers. Declarations have the following form:

declaration-specifiers init-declarator-list

The *init-declarator-list* is a comma-separated sequence of declarators, each of which can have an initializer. The declarators in the *init-declarator-list* consist of a sequence of type.

4.7 Function Definition

A function definition has the following form:

```
function func-identifier (identifier-list) {
  statement
}
```

func-identifier represents the name of the function. *Expression-list* is a list of identifier or arguments, separated by commas. The list of argument can be empty. The return statement should be used in function block in order to return a value.

4.8 Library Functions

4.8.1 Console Output Functions

4.8.1.1 Print Function

Print function takes an argument list and print them to the standard output. The return value is number of characters written or negative if an error occurred.

4.8.2 File I/O Functions

4.8.2.1 Load Function

The load function has the following form:

load(char-identifier, char- identifier)

The load function takes the name of the XML document and the DTD file as the first and second argument respectively, and returns a *document* type which is pointing to the data structures. If the function fails, the function returns NULL.

4.8.2.2 Release Function

The release function has the following form:

release(document-identifier)

The release function takes an document identifier as the argument, and free resources that are linked to the document identifier.

4.8.3 Mathematical Functions

The following mathematical functions are supported by this language:

Ceil(x)	smallest integer not less than x, as a float
exp(x)	exponential function e^x
Fabs(x)	absolute value $\|x\|$
floor(x)	largest integer not greater than x, as a float
Log(x, y)	logarithm of x with base y $log_y x$, $x > 0$
Pow(x, y)	x^y. A domain error occurs if x=0 and y =0, or if x<0 and y is not an integer

5 Project Plan

5.1 Development Plan

The development plan for this project is to use the waterfall development process. The project is divided into different phases and will only move to the next phase if the previous phase is successfully implemented. Different phases are outlined in below:

1. Development of lexer
2. Development of parser
3. Development of AST walker
4. Development of data types, symbol tables, control flow, expression handling and internal functions
5. Development of XML handling mechanism
6. Development of query logic

In this project, there are several advantages to use the waterfall development process over some other methodology.

1. Every phase has very clear deliverables
2. Requirements and scope are locked down and do not change
3. Each phase require the successfully implementation of the previous phase.

5.2 Programming Style Guide

5.2.1 Antlr

Formal coding convention for Antlr code was not developed in this project. The only convention used is all the tokens in the lexer class are in upper case.

5.2.2 Java

Coding conversion of for the Java portion in this project follows code conventions for Java programming language suggested by Sun.

5.3 Project Timeline

Sept 23	Submitted white paper proposal for XQ
Sept 24 – Oct 28	Research language grammar
Oct 28	Submit LRM
Oct 28 – Nov 7	Learn Antlr
Nov 8 – Nov 14	Implement Lexer
Nov 14 – Dec 21	Implement Parser
Dec 21 – Dec 12	Implement AST walker
Dec 12 – Dec 18	Implement language feature and project report

5.4 Roles and Responsibilities

All the development work in this project is solely done by the author of this report.

5.5 Development Environment

5.5.1 Machine & OS

This project is developed on an Intel-based P4 x86 (big endian) processor machine running Windows XP Home.

5.5.2 IDE

The IDE for this project is Eclipse 2.1.2 .

5.5.3 Antlr

The version of Antlr used in this project is 2.7.2. The Eclipse Antlr plug-ins used are Antlr Documentation 2.7.2, Antlr Core Plugin 1.0.3 and Antlr UI Plugin 1.0.4.

5.5.4 Java

The version of Java used in this project is Java 1.4.2. The Eclipse Java plug-in used is Java Development Tools Core 2.1.2.

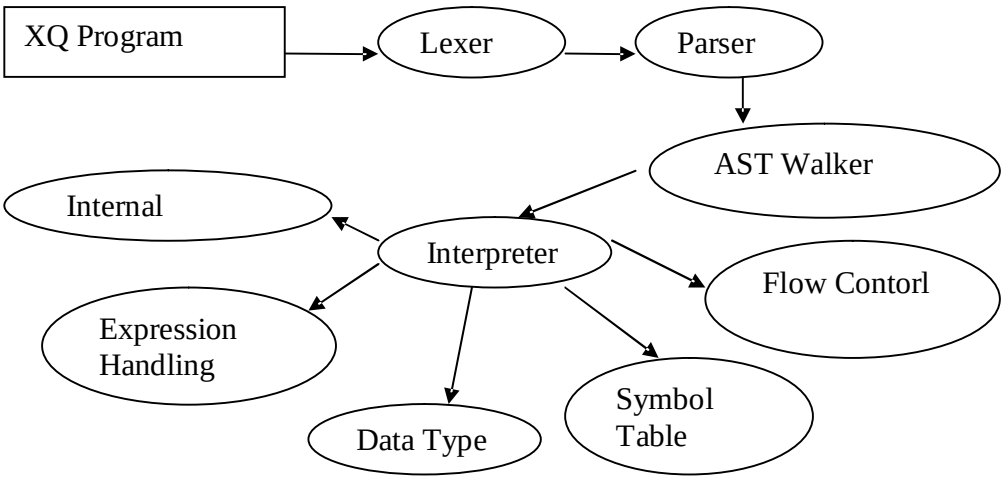
5.5.5 Revision Control

CVS was originally picked as the change (configuration) management tool. CVSNT was installed to work with Eclipse for change management. However, the development has VirusScan 7.0 installed and it has a known issue with any windows based CVS tools. Due to the time constraint of this project, the project decided to go along without any change management tools.

6 Architectural Design

6.1 Major Componets

There are several major componets in this project. They are interacted in the block diagram below.



6.1.1 Lexer

The Lexer perform lexical analysis on the input XQ program. It turns the program into tokens using a set of grammar rules. The output of the Lexer is passed to the Parser for parsing.

6.1.2 Parser

The Parser takes in the stream of tokens that are prepared by the Lexer. It then uses the tokens to construct a abstract syntax tree (AST) using a set of grammar rules.

6.1.3 AST Walker

The AST Walker is used to perform a depth-first search on the AST . It also has a set of grammar rules to specify the actions needed when certain tree nodes are encounted. In this project, the actions invoked by the AST Walker are the functions provided by the Interpreter.

6.1.4 Interpreter

The Interpreter module acts as a interface between the AST walker and the rest of the program. The interpreter abstracted the backend implementation of XQ.

6.1.5 Internal Functions

All the internal functions module suppport by XQ are implemented in the Internal Functions module. It is called by the Interpreter.

6.1.6 Symbol Table

The Symbol Table tracks all the symbol used in the program. Symbol name and their properties are stored in a HashMap data structure for quick lookup.

6.1.7 Data Type

All the data types and their properties are implemented in the Data Type module. Expressions that returns data type value are also handled by this module.

6.1.8 Control Flow

The Control Flow module handles the control flow. It handles the selection, iteration and control logic in the program.

7 Test Plan

Each of the phases mentioned in the development plan should have a corresponding test plan to verify the deliverables fulfill all the requirements. However, due to the time constraint of the project, majority of the effort was dedicated to the code development and mininal testing was done to verify the correctness of the deliverables. If more time was given to the project, functional and unit tests should be implemented.

8 Lesson Learn

The post-implementation analysis of this project can be summarized as below:

8.1 Resource Constraints

The design did not take into consideration that the resource available to this project. Unlike other projects in the class which consist of 4 team members, this project has only 1 person. A less aggressive design should be used to compensate the shortage of resource. The lesson learn is to understand the resource requirement at the beginning to avoid non-realistic design.

8.2 Not Afraid to Seek Help

This project spent a lot of time on designing the AST walker. The project team tried to understand the concept through online tutorial material but the resource on this subject is very limited. After about 1 week of investigation, the team final understand the key concept of the tree walker is to perform a DSF on the AST prepared by the parser. However, the progress of the project has been greatly affected by this. The time wasted on understanding the AST could have been saved by seeking help from the instructor or TA. The lesson learn in this case is not afraid to seek help. Instructors and TA will do a much better job on explaning the material than online tutorial.

8.3 Test Plan

Test plan should be done as early as possible. Once all the requirements are finalized, the test plan should be outlined before the development phases start. This will help identify defects in the early stages and reduce the cost required to fix them.

8.4 Milestones

Identify project milestones and put them into the project timeline. A constant status check should be preform to examine the health of the project. This will prevent the project to fall behind schedule.

8.5 Scheduling

One of the problem this project face was all the development work are only done on the weekend. Sometimes it requires a lot of time to recall what was left off from the previous week. A more even development scheduling should be done to mininize time needed to refresh knowledge.

8.6 Project Documentation

Better project documentation should be done to keep track of open issues and development progress.

9 Outstanding Issue Log

This section is used to capture all the outstanding issues with the project.

1. For loop is not implemented
2. Switch-case statements are not implemented.
3. Function call is not implemented.
4. Because function call is not implemented, print() and load() are temporary implemented as statements.
5. The XML handling logic is not implemented.
6. The qeury logic is not implemented.

10 Bibliography

[1] XQL, J. Robie, J. Lapp, D. Schach. *XML Query Language (XQL)*. See http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[2] XML-QL, Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. *A Query Language for XML*. See http://www.research.att.com/~mff/files/final.html

[3] QUILT, Don Chamberlin, Jonathan Robie, and Daniela Florescu. *Quilt: an XML Query Language for Heterogeneous Data Sources*. In *Lecture Notes in Computer Science*, Springer-Verlag, Dec. 2000. Also available at http://www.almaden.ibm.com/cs/people/chamberlin/quilt.html.

[4] XQuery, World Wide Web Consortium. *XQueryX, Version 1.0*. W3C Working Draft, 7 June 2001. See http://www.w3.org/TR/xqueryx

[5] SQL, International Organization for Standardization (ISO). *Information Technology-Database Language SQL*. Standard No. ISO/IEC 9075:1999

[6] C Style Comment, Brian W. Kernighan, and Dennis M. Ritchie. *The C Programming Language, Second Edition.*

[7] ANTLR, http://www.antlr.org

[8] Xerces-J, http://xml.apache.org/xerces-j

[9] Document Object Model, http://www.w3.org/DOM

[10] Junit, http://www.junit.org

[11] CVS, tttp://www.cvshome.org

[12] Sun JAVA Coding Conventions, http://java.sun.com/docs/codeconv

A. Language Syntax

Lexical Rules

ws à (' ' | '\t')+
nl à ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
comment à "/*" ((nl) | ~('\n' | '\r'))* "*/"
string à '"' (~('"') | ('"' '"'))* '"'
digits à '0' .. '9' +
number à ((digit)+ (. (digit)*)? ! . (digit+) ((e|E) (+|-)? (digit)+)?
id à alpha (alpha | digit)*

Syntactic Rules

start_condition à declaration_list statement_list
statement à assign_statement | if_statement | iteration_statement | break_statement |
continue_statement | compound_statement | fetch_statement |
select_statement
statement_list à (statement)+
compound_statement à '{' (statement_list)? '}'
fetch_statement à fetch_clause using_clause ';'
fetch_clause à 'FETCH' id into_cluase ';'
into_clause à 'INTO' id (',' id)+
select_statement à select_clause from clause where_clause using_clause ';'
from_clause à 'FROM' id
select_clause à 'SELECT' id (',' id)
where_clause à 'WHERE' id
using_clause à 'USING' id
if_statement à 'if' '(' 'expression' ')' 'statement list ('else' statement_list)?
break_statement à 'break' ';'
continue_statement à 'continue' ';'
iteration_statement à while_loop
while_loop à 'while' ('expression') 'statement_list
assign_statement à assign_expression ';'
assign_expression à assign_id ('++' | '--' | assignment_operator expression)
assign_id à id (['expression'])?
assignment_operator à assign | add_assign | sub_assign | div_assign |
star_assign | mod_assign
expression à logical_and_expression ('||' logical_and_expression)*
logical_and_expression à equality_expression ('&&' equality_expression)*
equality_expression à relational_expression (('==' | '!=') relational_expression)*
relational_expression à additive_expression
(('<' | '>' | '>=' | '<=') additive_expression)*

```
additive_expression à multiplicative_expression
                        (( +‘ |’ - ‘)’multiplicative_expression )*
multiplicative_expression à unary_expression
                                (( *‘ |’ / ‘|’ % ) ùnary_expression )*
unary_expression à ( primary_expression | unary_operator primary_expression)
unary_operator à +‘ |’ - ‘|’ ! ‘ ’
 primary_expression à id ( [ ‘eexpression ] ‘)? | string
                        | ( ‘eexpression ) ‘) ’
declaration_list à ( declaration )+
declaration à type_specifier id
        ( [ ‘eexpression ] ‘)? ; ‘ ’
type_specifier à inft |’ dóuble |’ cúrsor |’ string ’
```

B. Code Listing

Here is all the source code related to this project:

```
-rw-rw-rw-  1 Kin     5        37810 Dec 17 03:50 XQAntlrLexer.java
-rw-rw-rw-  1 Kin     5        49665 Dec 17 03:50 XQAntlrParser.java
-rw-rw-rw-  1 Kin     5         1757 Dec 17 03:50 XQAntlrTokenTypes.java
-rw-rw-rw-  1 Kin     5         1286 Dec 17 03:50 XQAntlrTokenTypes.txt
-rw-rw-rw-  1 Kin     5        30554 Dec 17 03:50 XQAntlrWalker.java
-rw-rw-rw-  1 Kin     5         1764 Dec 17 03:50 XQAntlrWalkerTokenTypes.java
-rw-rw-rw-  1 Kin     5         1298 Dec 17 03:50 XQAntlrWalkerTokenTypes.txt
-rw-rw-rw-  1 Kin     5         3147 Dec 16 04:43 XQDataType.java
-rw-rw-rw-  1 Kin     5         2746 Dec 15 02:05 XQDouble.java
-rw-rw-rw-  1 Kin     5          470 Dec 14 04:12 XQException.java
-rw-rw-rw-  1 Kin     5         1850 Dec 14 21:06 XQFunction.java
-rw-rw-rw-  1 Kin     5         3820 Dec 15 02:03 XQInt.java
-rw-rw-rw-  1 Kin     5         7343 Dec 15 00:10 XQInternalFunction.java
-rw-rw-rw-  1 Kin     5         6117 Dec 17 01:50 XQInterpreter.java
-rw-rw-rw-  1 Kin     5        36717 Dec 14 04:27 XQLexer.java
-rw-rw-rw-  1 Kin     5         1821 Dec 14 21:21 XQMain.java
-rw-rw-rw-  1 Kin     5        61900 Dec 14 04:27 XQParser.java
-rw-rw-rw-  1 Kin     5         1080 Dec 14 21:27 XQString.java
-rw-rw-rw-  1 Kin     5         1421 Dec 16 00:12 XQSymbolTable.java
-rw-rw-rw-  1 Kin     5          874 Dec  9 04:43 XQTokenTypes.java
-rw-rw-rw-  1 Kin     5          583 Dec  9 04:43 XQTokenTypes.txt
-rw-rw-rw-  1 Kin     5          729 Dec 14 20:56 XQVariable.java
-rw-rw-rw-  1 Kin     5         1822 Dec 14 04:27 XQWalkerTokenTypes.java
-rw-rw-rw-  1 Kin     5         1337 Dec 14 04:27 XQWalkerTokenTypes.txt
-rw-rw-rw-  1 Kin     5         7553 Dec 18 18:41 grammar.g
-rw-rw-rw-  1 Kin     5         6559 Dec 17 02:38 walker.g
```

```
/*
 * grammar.g : the lexer and the parser, in ANTLR grammar.
 *
 * @author Kin Ng - kn2006@columbia.edu
 *
 */

class XQParser extends Parser;

options{
        k = 2;
        buildAST=true;
        exportVocab = XQAntlr;
}

tokens {
        NStartCondition;
}

{
  int nr_error = 0;
  public void reportError( String s ) {
      super.reportError( s );
      nr_error++;
   }
  public void reportError( RecognitionException e ) {
      super.reportError( e );
      nr_error++;
   }
}

interpretation_unit:
        ( start_condition   | /* nothing */)
        EOF!
        ;

start_condition:
        "main"^ LPAREN! RPAREN! LCURLY^
        (declaration_list (statement_list)?)?
        RCURLY!
        { #start_condition = #(#[NStartCondition,"NStartCondition"],#start_condition); }
        ;

declaration_list
        :       (declaration)+
        ;

declaration:
        type_specifier declarator_list SEMI!
        { #declaration = #([NDeclaration, "NDeclaration"], #declaration);}
        ;

declarator
        : ID
        ( LBRACKET! INTLIT RBRACKET!
        { #declarator = #([NDeclareList, "NDeclareList"], #declarator); }
        )?
        ;
```

```
type_specifier
            :
            (       INTEGER
            |       DOUBLE
            |       STRING
            |       DOCUMENT
            |       CURSOR
            )
            ;

statement:
            assign_statement
            ;

assignment
      : l_value ( ASGN^ | PLUSEQ^ | MINUSEQ^ | MULTEQ^ | LDVEQ^
                | MODEQ^ | RDVEQ^
              ) expression SEMI!
      ;


class XQLexer extends Lexer;

options{
   k = 2;
   charVocabulary = '\3'..'\377';
   testLiterals = false;
   exportVocab = XQAntlr;
}

tokens {
// token keywords
   IF                   = "if";
            ELSE    = "else";
            SWITCH = "switch";
            CASE    = "case";
            DEFAULT         = "default";
            WHILE   = "while";
            DO                  = "do";
            FOR                 = "for";
            INTEGER         = "integer";
            FLOAT   = "float";
            STRING  = "string";
            DOCUMENT = "document";
            CURSOR   = "cursor";
}

{
   int nr_error = 0;
   public void reportError( String s ) {
      super.reportError( s );
      nr_error++;
   }
   public void reportError( RecognitionException e ) {
      super.reportError( e );
      nr_error++;
   }
```

```
}

protected
ALPHA: 'a'..'z' | 'A'..'Z' | '_' ;

protected
DIGIT: '0'..'9';

WS: (' ' | '\t')+ { $setType(Token.SKIP); };

NL: ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
   { $setType(Token.SKIP); newline(); };

COMMENT: "/*"
                        ( options {greedy=false;} : (NL)
                        | ~( '\n' | '\r' )
                        )* "*/"
                        { $setType(Token.SKIP); };

INTLIT:  (DIGIT)+ ;

STRING_LITERAL: ""!
                                ( "" ""! | ~(""|'\n'|'\r') )*
                                ( ""! | // nothing -- write error message
                                );

ID  options { testLiterals = true; }:
            ALPHA (ALPHA|DIGIT)*
   ;


// Operators
DOT        : '.'  ;
COLON      : ':'  ;
SEMI       : ';'  ;
COMMA      : ','  ;
EQUALS     : '='  ;
LBRACKET   : '['  ;
RBRACKET   : ']'  ;
LPAREN     : '('  ;
RPAREN     : ')'  ;
NOT_EQUALS : "/=" ;
LT         : '<'  ;
LTE        : "<=" ;
GT         : '>'  ;
GTE        : ">=" ;
PLUS       : '+'  ;
MINUS      : '-'  ;
TIMES      : '*'  ;
DIV        : '/'  ;
LCURLY     : '{'  ;
RCURLY     : '}'  ;
PLUSEQ                  : "+=";
MINUSEQ    : "-=";
MULTEQ     : "*=";
RDVEQ      : "/=";
MODEQ      : "%=";
```

```
/*
 * walker.g : the AST walker , in ANTLR grammar.
 *
 * @author Kin Ng - kn2006@columbia.edu
 *
 */

{
import java.io.*;
import java.util.*;
}

class XQAntlrWalker extends TreeParser;

options {
                importVocab = XQAntlr;
                buildAST=false;
}

{
   static XQDataType null_data = new XQDataType( "<NULL>" );
   XQInterpreter ipt = new XQInterpreter();
}


start_condition
        :       #(NStartCondition declaration_list statement_list)
        ;

statement_list
        :       ( statement )+
        ;

assign_statement
        :       assign_expression
        ;

assign_expression
{
        XQDataType aid, e;
        XQInt i = new XQInt(1);
}
        :
        (       #( ASSIGN aid=assign_ID e=expression )
                { aid.print(); System.out.println("before assign-->"+aid.name); ipt.assign(aid, e);
                        System.out.println("after assign"); }
        |       #( ADD_ASSIGN aid=assign_ID e=expression )  { aid.add(e); }
        |       #( SUB_ASSIGN aid=assign_ID e=expression )      { aid.sub(e); }
        |       #( DIV_ASSIGN aid=assign_ID e=expression )      { aid.ldiv(e); }
        |       #( STAR_ASSIGN aid=assign_ID e=expression )     { aid.mul(e); }
        |       #( MOD_ASSIGN aid=assign_ID e=expression )      { aid.rem(e); }
        |       #( INC_OP aid=assign_ID )                                       { aid.add(i); }
        |       #( DEC_OP aid=assign_ID )                                       { aid.sub(i); }
        )
        ;

assign_ID returns [ XQDataType r ]
{
```

```
 r = null_data;
 XQDataType e;
 XQDataType [] x;
}
        :       sid:ID { r = ipt.getVariable(sid.getText());
                         System.out.println(sid.getText()+"--"+r.name);
                         r.print();
                         }
        |       #(NAssignList id:ID e=expression)
                { x = ipt.getArrayVariable(id.getText());
                  r = x[XQInt.intValue(e)];
                 }
        ;

expression returns [ XQDataType r ]
{
 r = null_data;
}
        :       r=logical_or_expression
        |       r=logical_and_expression
        |       r=equality_expression
        |       r=relational_expression
        |       r=additive_expression
        |       r=multiplicative_expression
        |       r=unary_expression
        |       r=primary_expression
        ;

logical_or_expression returns [ XQDataType r ]
{
        XQDataType a, b;
        r = null_data;

}
        :       #( LOR a=expression b=expression ) { r = a.or(b); }
        ;

logical_and_expression returns [ XQDataType r ]
{
        XQDataType a, b;
        r = null_data;
}
        :       #( LAND a=expression b=expression )      { r = a.and(b); }
        ;

equality_expression returns [ XQDataType r ]
{
        XQDataType a, b;
        r = null_data;
}
        :       #( EQUAL a=expression b=expression )     { r = a.eq(b); }
        |       #( NOT_EQUAL a=expression b=expression ) { r = a.ne(b); }
        ;

relational_expression returns [ XQDataType r ]
{
        XQDataType a, b;
        r = null_data;
```

```
}
        :       #( LT a=expression b=expression )    { r = a.lt(b); }
        |       #( GT a=expression b=expression )  { r = a.gt(b); }
        |       #( LTE a=expression b=expression )          { r = a.le(b); }
        |       #( GTE a=expression b=expression )          { r = a.ge(b); }
        ;

additive_expression returns [ XQDataType r ]
{
        XQDataType a, b;
        r = null_data;
}
        :       #( PLUS a=expression b=expression )  { r = a.add(b); }
        |       #( MINUS a=expression b=expression ) { r = a.minus(b); }
        ;

multiplicative_expression returns [ XQDataType r ]
{
        XQDataType a, b;
        r = null_data;
}
        :       #( STAR a=expression b=expression ) { r = a.times(b); }
        |       #( DIV a=expression b=expression )  { r = a.lfracts(b); }
        |       #( MOD a=expression b=expression )  { r = a.modulus(b); }
        ;

unary_expression returns [ XQDataType r ]
{
        r = null_data;
}
        :       #(NUnary uo:unary_operator r=primary_expression )
        {
                if (  uo.getType() == #MINUS )
                        r.uminus();
                else if (uo.getType() == #LNOT )
                        r.not();
        }
        ;

unary_operator
        :       PLUS
        |       MINUS
        |       LNOT
        ;

primary_expression returns [ XQDataType r ]
{
                XQDataType x[];
                XQDataType a;
                r = null_data;
}
        :
        ( i:ID  {r = ipt.getVariable(i.getText()); }
        | #(NListDeclaration il:ID a=expression )
                {x = ipt.getArrayVariable(il.getText());
                 r = x[XQInt.intValue(a)];
                }
        |       sl:STRING_LITERAL { r = new XQString(sl.getText());}
```

```
        |       ic:INT_CONST      { r = ipt.getNumber(ic.getText()); }
        |       dc:DOUBLE_CONST   { r = ipt.getNumber(dc.getText()); }
        | #( LPAREN r=expression )
        )       ;

declaration_list
        :       ( declaration )+
        ;

declaration
{
        int size = 0;
        XQDataType a = null_data;
}
        :
        #(NDeclaration t:type_specifier
        idec:ID ( a=expression  {size = XQInt.intValue(a);} )?
        )
        {
                System.out.println("Im hhere");
                if (t.getType() == #INT) {
                        if (size == 0 ) {
                                XQInt x = new XQInt(0);
                                x.setName(idec.getText());
                                System.out.println("Im hhere-->"+idec.getText()+"<---");
                                ipt.declareVariable(idec.getText(), x);
                                System.out.println("Im hhere");
                        }
                        else {
                                XQInt [] xs = new XQInt[size];
                                for (int i = 0; i < size; i++) {
                                        xs[i] = new XQInt(0);
                                        xs[i].setName(idec.getText());
                                }
                                ipt.declareArrayVariable(idec.getText(), xs);
                        }

                }
                else if (t.getType() == #DOUBLE ) {
                        if (size == 0 ) {
                                XQDouble x = new XQDouble(0);
                                x.setName(idec.getText());
                                ipt.declareVariable( idec.getText(), x);
                        }
                        else {
                                XQDouble [] xs = new XQDouble[size];
                                for (int i = 0; i < size; i++) {
                                        xs[i] = new XQDouble(0);
                                        xs[i].setName(idec.getText());
                                }
                                ipt.declareArrayVariable(idec.getText(), xs);
                        }
                }
                else if (t.getType() == #STRING)  {
                        if (size == 0 ) {
                                XQString x = new XQString("");
                                x.setName(idec.getText());
                                ipt.declareVariable( idec.getText(), x);
```

```
                        }
                        else {
                                XQString [] xs = new XQString[size];
                                for (int i = 0; i < size; i++) {
                                        xs[i] = new XQString("");
                                        xs[i].setName(idec.getText());
                                }
                                ipt.declareArrayVariable(idec.getText(), xs);
                        }
                }
        }
        ;

break_statement
        : #(BREAK {ipt.setBreak(null); })
        ;

continue_statement
        : #(CONTINUE { ipt.setContinue( null ); } )
        ;

iteration_statement
        :       whileLoop
        ;

whileLoop
{
        XQDataType a, b;
}
        : #(WHILE a=expression s:statement_list) {
                while ( ipt.canProceed() )
    {
        statement_list(#s);
        ipt.whileNext(null);
    }
    ipt.whileEnd(null);
        }
        ;

type_specifier
        :
        (       INT
        |       DOUBLE
        |       STRING
        )
        ;

statement
        :       assign_statement
        |       if_statement
        |       iteration_statement
        |       break_statement
        |       continue_statement
        ;

compound_statement
        :       #( LCURLY
            (statement_list)? )
```

```
            ;

if_statement
{
            XQDataType a;
}
            : #("if" a=expression thenp:. (elsep:.)?)
      {
       if (!( a instanceof XQInt )) {
           a.error( "if: expression should be integer value" );
                                    return;
       }
       if ( XQInt.intValue(a) != 0 )
            statement_list( #thenp );
       else if ( null != elsep )
            statement_list( #elsep );
       }
      ;


// start of XQAntlrLexer.java
// $ANTLR : "parser.g" -> "XQAntlrLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class XQAntlrLexer extends antlr.CharScanner implements XQAntlrTokenTypes, TokenStream
 {

   int nr_error = 0;
   public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
   public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
```

```
   }
public XQAntlrLexer(InputStream in) {
	this(new ByteBuffer(in));
}
public XQAntlrLexer(Reader in) {
	this(new CharBuffer(in));
}
public XQAntlrLexer(InputBuffer ib) {
	this(new LexerSharedInputState(ib));
}
public XQAntlrLexer(LexerSharedInputState state) {
	super(state);
	caseSensitiveLiterals = true;
	setCaseSensitive(true);
	literals = new Hashtable();
	literals.put(new ANTLRHashString("for", this), new Integer(72));
	literals.put(new ANTLRHashString("print", this), new Integer(6));
	literals.put(new ANTLRHashString("select", this), new Integer(14));
	literals.put(new ANTLRHashString("string", this), new Integer(51));
	literals.put(new ANTLRHashString("load", this), new Integer(10));
	literals.put(new ANTLRHashString("continue", this), new Integer(21));
	literals.put(new ANTLRHashString("fetch", this), new Integer(13));
	literals.put(new ANTLRHashString("while", this), new Integer(22));
	literals.put(new ANTLRHashString("from", this), new Integer(74));
	literals.put(new ANTLRHashString("cursor", this), new Integer(52));
	literals.put(new ANTLRHashString("break", this), new Integer(19));
	literals.put(new ANTLRHashString("return", this), new Integer(73));
	literals.put(new ANTLRHashString("if", this), new Integer(17));
	literals.put(new ANTLRHashString("double", this), new Integer(50));
	literals.put(new ANTLRHashString("int", this), new Integer(49));
	literals.put(new ANTLRHashString("using", this), new Integer(16));
	literals.put(new ANTLRHashString("else", this), new Integer(18));
}

public Token nextToken() throws TokenStreamException {
	Token theRetToken=null;
tryAgain:
	for (;;) {
		Token _token = null;
		int _ttype = Token.INVALID_TYPE;
		resetText();
		try {   // for char stream error handling
			try {   // for lexical error handling
				switch ( LA(1)) {
				case '\t':  case ' ':
				{
					mWS(true);
					theRetToken=_returnToken;
					break;
				}
				case '\n':  case '\r':
				{
					mNL(true);
					theRetToken=_returnToken;
					break;
				}
				case ':':
				{
```

```
            mCOLON(true);
            theRetToken=_returnToken;
            break;
        }
        case ',':
        {
            mCOMMA(true);
            theRetToken=_returnToken;
            break;
        }
        case ';':
        {
            mSEMI(true);
            theRetToken=_returnToken;
            break;
        }
        case '.':
        {
            mDOT(true);
            theRetToken=_returnToken;
            break;
        }
        case '(':
        {
            mLPAREN(true);
            theRetToken=_returnToken;
            break;
        }
        case ')':
        {
            mRPAREN(true);
            theRetToken=_returnToken;
            break;
        }
        case '[':
        {
            mLBRACKET(true);
            theRetToken=_returnToken;
            break;
        }
        case ']':
        {
            mRBRACKET(true);
            theRetToken=_returnToken;
            break;
        }
        case '{':
        {
            mLCURLY(true);
            theRetToken=_returnToken;
            break;
        }
        case '}':
        {
            mRCURLY(true);
            theRetToken=_returnToken;
            break;
        }
```

```
case '&':
{
		mLAND(true);
		theRetToken=_returnToken;
		break;
}
case '|':
{
		mLOR(true);
		theRetToken=_returnToken;
		break;
}
case '"':
{
		mSTRING_LITERAL(true);
		theRetToken=_returnToken;
		break;
}
case '0': case '1': case '2': case '3':
case '4': case '5': case '6': case '7':
case '8': case '9':
{
		mNUMBER(true);
		theRetToken=_returnToken;
		break;
}
case 'A': case 'B': case 'C': case 'D':
case 'E': case 'F': case 'G': case 'H':
case 'I': case 'J': case 'K': case 'L':
case 'M': case 'N': case 'O': case 'P':
case 'Q': case 'R': case 'S': case 'T':
case 'U': case 'V': case 'W': case 'X':
case 'Y': case 'Z': case '_': case 'a':
case 'b': case 'c': case 'd': case 'e':
case 'f': case 'g': case 'h': case 'i':
case 'j': case 'k': case 'l': case 'm':
case 'n': case 'o': case 'p': case 'q':
case 'r': case 's': case 't': case 'u':
case 'v': case 'w': case 'x': case 'y':
case 'z':
{
		mID(true);
		theRetToken=_returnToken;
		break;
}
default:
		if ((LA(1)=='/') && (LA(2)=='*')) {
			mCOMMENT(true);
			theRetToken=_returnToken;
		}
		else if ((LA(1)=='=') && (LA(2)=='=')) {
			mEQUAL(true);
			theRetToken=_returnToken;
		}
		else if ((LA(1)=='!') && (LA(2)=='=')) {
			mNOT_EQUAL(true);
			theRetToken=_returnToken;
		}
```

```
else if ((LA(1)=='<') && (LA(2)=='=')) {
		mLTE(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='>') && (LA(2)=='=')) {
		mGTE(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='/') && (LA(2)=='=')) {
		mDIV_ASSIGN(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='*') && (LA(2)=='=')) {
		mSTAR_ASSIGN(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='%') && (LA(2)=='=')) {
		mMOD_ASSIGN(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (LA(2)=='=')) {
		mADD_ASSIGN(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (LA(2)=='=')) {
		mSUB_ASSIGN(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (LA(2)=='+')) {
		mINC_OP(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (LA(2)=='-')) {
		mDEC_OP(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (LA(2)=='<')) {
		mLSHIFT(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='=') && (true)) {
		mASSIGN(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (true)) {
		mLT(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='>') && (true)) {
		mGT(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='%') && (true)) {
		mMOD(true);
		theRetToken=_returnToken;
}
else if ((LA(1)=='/') && (true)) {
		mDIV(true);
```

```
                                        theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='+') && (true)) {
                                        mPLUS(true);
                                        theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='-') && (true)) {
                                        mMINUS(true);
                                        theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='*') && (true)) {
                                        mSTAR(true);
                                        theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='!') && (true)) {
                                        mLNOT(true);
                                        theRetToken=_returnToken;
                                    }
                                else {
                                    if    (LA(1)==EOF_CHAR)    {uponEOF();    _returnToken    =
makeToken(Token.EOF_TYPE);}
                                else    {throw    new    NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
                                }
                                }
                                if ( _returnToken==null ) continue tryAgain; // found SKIP token
                                _ttype = _returnToken.getType();
                                _ttype = testLiteralsTable(_ttype);
                                _returnToken.setType(_ttype);
                                return _returnToken;
                        }
                        catch (RecognitionException e) {
                                throw new TokenStreamRecognitionException(e);
                        }
                }
                catch (CharStreamException cse) {
                        if ( cse instanceof CharStreamIOException ) {
                                throw                                                    new
TokenStreamIOException(((CharStreamIOException)cse).io);
                        }
                        else {
                                throw new TokenStreamException(cse.getMessage());
                        }
                }
        }
}

        public final void mWS(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = WS;
                int _saveIndex;

                {
                int _cnt4628=0;
                _loop4628:
                do {
                        switch ( LA(1)) {
```

```
                        case ' ':
                        {
                                match(' ');
                                break;
                        }
                        case '\t':
                        {
                                match('\t');
                                break;
                        }
                        default:
                        {
                                if ( _cnt4628>=1 ) { break _loop4628; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                        }
                        }
                        _cnt4628++;
                } while (true);
                }
                if ( inputState.guessing==0 ) {
                        _ttype = Token.SKIP;
                }
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                        _token = makeToken(_ttype);
                        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
                }
                _returnToken = _token;
        }

        public final void mNL(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = NL;
                int _saveIndex;

                {
                boolean synPredMatched4632 = false;
                if (((LA(1)=='\r') && (LA(2)=='\n'))) {
                        int _m4632 = mark();
                        synPredMatched4632 = true;
                        inputState.guessing++;
                        try {
                                {
                                match('\r');
                                match('\n');
                                }
                        }
                        catch (RecognitionException pe) {
                                synPredMatched4632 = false;
                        }
                        rewind(_m4632);
                        inputState.guessing--;
                }
                if ( synPredMatched4632 ) {
                        match('\r');
                        match('\n');
                }
                else if ((LA(1)=='\n')) {
```

```
                    match('\n');
                }
                else if ((LA(1)=='\r') && (true)) {
                    match('\r');
                }
                else {
                    throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
                }

                }
                if ( inputState.guessing==0 ) {
                    _ttype = Token.SKIP; newline();
                }
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
                }
                _returnToken = _token;
        }

        public final void mCOMMENT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = COMMENT;
                int _saveIndex;

                match("/*");
                {
                _loop4637:
                do {
                    // nongreedy exit test
                    if ((LA(1)=='*') && (LA(2)=='/')) break _loop4637;
                    if ((_tokenSet_0.member(LA(1))) && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
                        {
                        match(_tokenSet_0);
                        }
                    }
                    else if ((LA(1)=='\n'||LA(1)=='\r')) {
                        {
                        mNL(false);
                        }
                    }
                    else {
                        break _loop4637;
                    }

                } while (true);
                }
                match("*/");
                if ( inputState.guessing==0 ) {
                    _ttype = Token.SKIP;
                }
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
                }
                _returnToken = _token;
```

```
	}

	public final void mASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = ASSIGN;
		int _saveIndex;

		match('=');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mCOLON(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COLON;
		int _saveIndex;

		match(':');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mCOMMA(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COMMA;
		int _saveIndex;

		match(',');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSEMI(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SEMI;
		int _saveIndex;

		match(';');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	public final void mDOT(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DOT;
		int _saveIndex;

		match('.');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLPAREN(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LPAREN;
		int _saveIndex;

		match('(');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRPAREN(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RPAREN;
		int _saveIndex;

		match(')');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLBRACKET(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LBRACKET;
		int _saveIndex;

		match('[');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRBRACKET(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
```

```
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RBRACKET;
		int _saveIndex;

		match(']');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLCURLY(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LCURLY;
		int _saveIndex;

		match('{');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRCURLY(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RCURLY;
		int _saveIndex;

		match('}');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mEQUAL(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = EQUAL;
		int _saveIndex;

		match("==");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mNOT_EQUAL(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NOT_EQUAL;
```

```
		int _saveIndex;

		match("!=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLTE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LTE;
		int _saveIndex;

		match("<=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LT;
		int _saveIndex;

		match("<");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mGTE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = GTE;
		int _saveIndex;

		match(">=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mGT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = GT;
		int _saveIndex;
```

```
		match(">");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mMOD(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = MOD;
		int _saveIndex;

		match('%');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mDIV(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DIV;
		int _saveIndex;

		match('/');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mPLUS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = PLUS;
		int _saveIndex;

		match('+');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mDIV_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DIV_ASSIGN;
		int _saveIndex;

		match("/=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
```

```
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mSTAR_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = STAR_ASSIGN;
            int _saveIndex;

            match("*=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mMOD_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MOD_ASSIGN;
            int _saveIndex;

            match("%=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mADD_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = ADD_ASSIGN;
            int _saveIndex;

            match("+=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mMINUS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MINUS;
            int _saveIndex;

            match('-');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
```

```
		}
		_returnToken = _token;
	}

	public final void mSUB_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SUB_ASSIGN;
		int _saveIndex;

		match("-=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSTAR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STAR;
		int _saveIndex;

		match('*');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mINC_OP(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = INC_OP;
		int _saveIndex;

		match("++");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mDEC_OP(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DEC_OP;
		int _saveIndex;

		match("--");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
```

```
	}

	public final void mLSHIFT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LSHIFT;
		int _saveIndex;

		match("<<");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLAND(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LAND;
		int _saveIndex;

		match("&&");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLNOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LNOT;
		int _saveIndex;

		match('!');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLOR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LOR;
		int _saveIndex;

		match("||");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	public final void mSTRING_LITERAL(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STRING_LITERAL;
		int _saveIndex;

		match("");
		{
		_loop4673:
		do {
			if ((LA(1)=='\\')) {
				mEscape(false);
			}
			else if ((_tokenSet_1.member(LA(1)))) {
				matchNot("");
			}
			else {
				break _loop4673;
			}

		} while (true);
		}
		match("");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mEscape(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = Escape;
		int _saveIndex;

		match('\\');
		{
		switch ( LA(1)) {
		case 'a':
		{
			match('a');
			break;
		}
		case 'b':
		{
			match('b');
			if ( inputState.guessing==0 ) {
				text.setLength(_begin); text.append("\b");
			}
			break;
		}
		case 'f':
		{
			match('f');
			if ( inputState.guessing==0 ) {
				text.setLength(_begin); text.append("\f");
			}
```

```
		break;
	}
	case 'n':
	{
		match('n');
		if ( inputState.guessing==0 ) {
			text.setLength(_begin); text.append("\n");
		}
		break;
	}
	case 'r':
	{
		match('r');
		if ( inputState.guessing==0 ) {
			text.setLength(_begin); text.append("\r");
		}
		break;
	}
	case 't':
	{
		match('t');
		if ( inputState.guessing==0 ) {
			text.setLength(_begin); text.append("\t");
		}
		break;
	}
	case 'v':
	{
		match('v');
		break;
	}
	case '"':
	{
		match('"');
		if ( inputState.guessing==0 ) {
			text.setLength(_begin); text.append("\"");
		}
		break;
	}
	case '\'':
	{
		match('\'');
		if ( inputState.guessing==0 ) {
			text.setLength(_begin); text.append("\'");
		}
		break;
	}
	case '\\':
	{
		match('\\');
		if ( inputState.guessing==0 ) {
			text.setLength(_begin); text.append("\\");
		}
		break;
	}
	case '?':
	{
		match('?');
```

```
                    break;
                }
                case '0':  case '1':  case '2':  case '3':
                {
                    {
                    matchRange('0','3');
                    }
                    {
                    if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
                        mDIGIT(false);
                        {
                        if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <=
'\u00ff'))) {
                            mDIGIT(false);
                        }
                        else if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && (true)) {
                        }
                        else {
                            throw        new        NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                        }

                        }
                    }
                    else if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && (true)) {
                    }
                    else {
                        throw   new   NoViableAltForCharException((char)LA(1),   getFilename(),
getLine(), getColumn());
                    }

                    }
                    break;
                }
                case '4':  case '5':  case '6':  case '7':
                {
                    {
                    matchRange('4','7');
                    }
                    {
                    if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
                        mDIGIT(false);
                    }
                    else if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && (true)) {
                    }
                    else {
                        throw   new   NoViableAltForCharException((char)LA(1),   getFilename(),
getLine(), getColumn());
                    }

                    }
                    break;
                }
                case 'x':
                {
                    match('x');
                    {
                    int _cnt4682=0;
```

```
                        _loop4682:
                        do {
                                if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <=
'\u00ff'))) {
                                        mDIGIT(false);
                                }
                                else if (((LA(1) >= 'a' && LA(1) <= 'f')) && ((LA(2) >= '\u0003' && LA(2)
<= '\u00ff'))) {
                                        matchRange('a','f');
                                }
                                else if (((LA(1) >= 'A' && LA(1) <= 'F')) && ((LA(2) >= '\u0003' && LA(2)
<= '\u00ff'))) {
                                        matchRange('A','F');
                                }
                                else {
                                        if ( _cnt4682>=1 ) { break _loop4682; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                                }

                                _cnt4682++;
                        } while (true);
                        }
                        break;
                }
                default:
                {
                        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
                }
                }
                }
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                        _token = makeToken(_ttype);
                        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
                }
                _returnToken = _token;
        }

        protected final void mDIGIT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = DIGIT;
                int _saveIndex;

                matchRange('0','9');
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                        _token = makeToken(_ttype);
                        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
                }
                _returnToken = _token;
        }

        protected final void mDOUBLE_CONST(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = DOUBLE_CONST;
                int _saveIndex;
```

```
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mINT_CONST(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = INT_CONST;
		int _saveIndex;

		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mNUMBER(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NUMBER;
		int _saveIndex;

		{
		boolean synPredMatched4691 = false;
		if ((((LA(1) >= '0' && LA(1) <= '9')) && (_tokenSet_2.member(LA(2))))) {
			int _m4691 = mark();
			synPredMatched4691 = true;
			inputState.guessing++;
			try {
				{
				{
				int _cnt4690=0;
				_loop4690:
				do {
					if (((LA(1) >= '0' && LA(1) <= '9'))) {
						mDIGIT(false);
					}
					else {
						if ( _cnt4690>=1 ) { break _loop4690; } else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
					}

					_cnt4690++;
				} while (true);
				}
				mDOT(false);
				}
			}
			catch (RecognitionException pe) {
				synPredMatched4691 = false;
			}
			rewind(_m4691);
			inputState.guessing--;
		}
```

```
		if ( synPredMatched4691 ) {
			{
			{
			int _cnt4694=0;
			_loop4694:
			do {
				if (((LA(1) >= '0' && LA(1) <= '9'))) {
					mDIGIT(false);
				}
				else {
					if ( _cnt4694>=1 ) { break _loop4694; } else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
				}

				_cnt4694++;
			} while (true);
			}
			mDOT(false);
			{
			_loop4696:
			do {
				if (((LA(1) >= '0' && LA(1) <= '9'))) {
					mDIGIT(false);
				}
				else {
					break _loop4696;
				}

			} while (true);
			}
			}
			if ( inputState.guessing==0 ) {
				_ttype = DOUBLE_CONST;
			}
		}
		else if (((LA(1) >= '0' && LA(1) <= '9')) && (true)) {
			{
			int _cnt4698=0;
			_loop4698:
			do {
				if (((LA(1) >= '0' && LA(1) <= '9'))) {
					mDIGIT(false);
				}
				else {
					if ( _cnt4698>=1 ) { break _loop4698; } else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
				}

				_cnt4698++;
			} while (true);
			}
			if ( inputState.guessing==0 ) {
				_ttype = INT_CONST;
			}
		}
		else {
			throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());
```

```
		}

		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mID(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = ID;
		int _saveIndex;

		{
		switch ( LA(1)) {
		case 'a': case 'b': case 'c': case 'd':
		case 'e': case 'f': case 'g': case 'h':
		case 'i': case 'j': case 'k': case 'l':
		case 'm': case 'n': case 'o': case 'p':
		case 'q': case 'r': case 's': case 't':
		case 'u': case 'v': case 'w': case 'x':
		case 'y': case 'z':
		{
			matchRange('a','z');
			break;
		}
		case 'A': case 'B': case 'C': case 'D':
		case 'E': case 'F': case 'G': case 'H':
		case 'I': case 'J': case 'K': case 'L':
		case 'M': case 'N': case 'O': case 'P':
		case 'Q': case 'R': case 'S': case 'T':
		case 'U': case 'V': case 'W': case 'X':
		case 'Y': case 'Z':
		{
			matchRange('A','Z');
			break;
		}
		case '_':
		{
			match('_');
			break;
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
		}
		}
		}
		{
		_loop4702:
		do {
			switch ( LA(1)) {
			case 'a': case 'b': case 'c': case 'd':
			case 'e': case 'f': case 'g': case 'h':
```

```
                    case 'i': case 'j': case 'k': case 'l':
                    case 'm': case 'n': case 'o': case 'p':
                    case 'q': case 'r': case 's': case 't':
                    case 'u': case 'v': case 'w': case 'x':
                    case 'y': case 'z':
                    {
                            matchRange('a','z');
                            break;
                    }
                    case 'A': case 'B': case 'C': case 'D':
                    case 'E': case 'F': case 'G': case 'H':
                    case 'I': case 'J': case 'K': case 'L':
                    case 'M': case 'N': case 'O': case 'P':
                    case 'Q': case 'R': case 'S': case 'T':
                    case 'U': case 'V': case 'W': case 'X':
                    case 'Y': case 'Z':
                    {
                            matchRange('A','Z');
                            break;
                    }
                    case '_':
                    {
                            match('_');
                            break;
                    }
                    case '0': case '1': case '2': case '3':
                    case '4': case '5': case '6': case '7':
                    case '8': case '9':
                    {
                            matchRange('0','9');
                            break;
                    }
                    default:
                    {
                            break _loop4702;
                    }
                    }
            } while (true);
            }
            _ttype = testLiteralsTable(_ttype);
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
    }


    private static final long[] mk_tokenSet_0() {
            long[] data = new long[8];
            data[0]=-9224L;
            for (int i = 1; i<=3; i++) { data[i]=-1L; }
            return data;
    }
    public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
    private static final long[] mk_tokenSet_1() {
            long[] data = new long[8];
            data[0]=-17179869192L;
```

```
		data[1]=-268435457L;
		for (int i = 2; i<=3; i++) { data[i]=-1L; }
		return data;
	}
	public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
	private static final long[] mk_tokenSet_2() {
		long[] data = { 288019269919178752L, 0L, 0L, 0L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());

	}


// start of XQAntlParser.java
// $ANTLR : "parser.g" -> "XQAntlrParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class XQAntlrParser extends antlr.LLkParser       implements XQAntlrTokenTypes
 {

   int nr_error = 0;
   public void reportError( String s ) {
       super.reportError( s );
       nr_error++;
    }
   public void reportError( RecognitionException e ) {
       super.reportError( e );
       nr_error++;
    }

protected XQAntlrParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public XQAntlrParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
```

```
}

protected XQAntlrParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public XQAntlrParser(TokenStream lexer) {
  this(lexer,2);
}

public XQAntlrParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

        public final void start_condition() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST start_condition_AST = null;

                try {      // for error handling
                        declaration_list();
                        astFactory.addASTChild(currentAST, returnAST);
                        statement_list();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(Token.EOF_TYPE);
                        start_condition_AST = (AST)currentAST.root;
                        start_condition_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NStartCondition,"NStartCondition")).add(start_condition_AST));
                        currentAST.root = start_condition_AST;
                        currentAST.child = start_condition_AST!=null
&&start_condition_AST.getFirstChild()!=null ?
                                start_condition_AST.getFirstChild() : start_condition_AST;
                        currentAST.advanceChildToEnd();
                        start_condition_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_0);
                }
                returnAST = start_condition_AST;
        }

        public final void declaration_list() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST declaration_list_AST = null;

                try {      // for error handling
                        {
```

```
                int _cnt4620=0;
                _loop4620:
                do {
                        if (((LA(1) >= INT && LA(1) <= CURSOR))) {
                                declaration();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                if ( _cnt4620>=1 ) { break _loop4620; } else {throw new
NoViableAltException(LT(1), getFilename());}
                        }

                        _cnt4620++;
                } while (true);
                }
                declaration_list_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_1);
        }
        returnAST = declaration_list_AST;
    }

    public final void statement_list() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST statement_list_AST = null;

        try {      // for error handling
                {
                int _cnt4560=0;
                _loop4560:
                do {
                        if ((_tokenSet_1.member(LA(1))) && (_tokenSet_2.member(LA(2)))) {
                                statement();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                if ( _cnt4560>=1 ) { break _loop4560; } else {throw new
NoViableAltException(LT(1), getFilename());}
                        }

                        _cnt4560++;
                } while (true);
                }
                statement_list_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
        }
        returnAST = statement_list_AST;
    }
```

```
public final void statement() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST statement_AST = null;

	try {	// for error handling
	  switch ( LA(1)) {
	  case ID:
	  {
			assign_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
	  }
	  case IF:
	  {
			if_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
	  }
	  case WHILE:
	  {
			iteration_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
	  }
	  case BREAK:
	  {
			break_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
	  }
	  case CONTINUE:
	  {
			continue_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
	  }
	  case LCURLY:
	  {
			compound_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
	  }
	  case PRINT:
	  {
			print_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
	  }
	  case LOAD:
```

```
                {
                        load_statement();
                        astFactory.addASTChild(currentAST, returnAST);
                        statement_AST = (AST)currentAST.root;
                        break;
                }
                case FETCH:
                {
                        fetch_statement();
                        astFactory.addASTChild(currentAST, returnAST);
                        statement_AST = (AST)currentAST.root;
                        break;
                }
                case SELECT:
                {
                        select_statement();
                        astFactory.addASTChild(currentAST, returnAST);
                        statement_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
        }
        returnAST = statement_AST;
}

public final void assign_statement() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST assign_statement_AST = null;

        try {      // for error handling
                assign_expression();
                astFactory.addASTChild(currentAST, returnAST);
                match(SEMI);
                assign_statement_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
        }
        returnAST = assign_statement_AST;
}

public final void if_statement() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
```

```
	AST if_statement_AST = null;

	try {	// for error handling
		AST tmp3_AST = null;
		tmp3_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp3_AST);
		match(IF);
		match(LPAREN);
		expression();
		astFactory.addASTChild(currentAST, returnAST);
		match(RPAREN);
		statement_list();
		astFactory.addASTChild(currentAST, returnAST);
		{
		if ((LA(1)==ELSE) && (_tokenSet_1.member(LA(2)))) {
			match(ELSE);
			statement_list();
			astFactory.addASTChild(currentAST, returnAST);
		}
		else if ((_tokenSet_3.member(LA(1))) && (_tokenSet_4.member(LA(2)))) {
		}
		else {
			throw new NoViableAltException(LT(1), getFilename());
		}

		}
		if_statement_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		consume();
		consumeUntil(_tokenSet_3);
	}
	returnAST = if_statement_AST;
}

public final void iteration_statement() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST iteration_statement_AST = null;

	try {	// for error handling
		whilLoop();
		astFactory.addASTChild(currentAST, returnAST);
		iteration_statement_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		consume();
		consumeUntil(_tokenSet_3);
	}
	returnAST = iteration_statement_AST;
}

public final void break_statement() throws RecognitionException, TokenStreamException {

	returnAST = null;
```

```
		ASTPair currentAST = new ASTPair();
		AST break_statement_AST = null;

		try {	// for error handling
			AST tmp7_AST = null;
			tmp7_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp7_AST);
			match(BREAK);
			match(SEMI);
			break_statement_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_3);
		}
		returnAST = break_statement_AST;
}

public final void continue_statement() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST continue_statement_AST = null;

		try {	// for error handling
			AST tmp9_AST = null;
			tmp9_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp9_AST);
			match(CONTINUE);
			match(SEMI);
			continue_statement_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_3);
		}
		returnAST = continue_statement_AST;
}

public final void compound_statement() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST compound_statement_AST = null;

		try {	// for error handling
			AST tmp11_AST = null;
			tmp11_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp11_AST);
			match(LCURLY);
			{
			switch ( LA(1)) {
			case LCURLY:
			case PRINT:
			case LOAD:
			case ID:
```

```
                    case FETCH:
                    case SELECT:
                    case IF:
                    case BREAK:
                    case CONTINUE:
                    case WHILE:
                    {
                            statement_list();
                            astFactory.addASTChild(currentAST, returnAST);
                            break;
                    }
                    case RCURLY:
                    {
                            break;
                    }
                    default:
                    {
                            throw new NoViableAltException(LT(1), getFilename());
                    }
                    }
                    }
                    match(RCURLY);
                    compound_statement_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_3);
            }
            returnAST = compound_statement_AST;
}

public final void print_statement() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST print_statement_AST = null;

            try {      // for error handling
                    AST tmp13_AST = null;
                    tmp13_AST = astFactory.create(LT(1));
                    astFactory.makeASTRoot(currentAST, tmp13_AST);
                    match(PRINT);
                    match(LPAREN);
                    primary_expression();
                    astFactory.addASTChild(currentAST, returnAST);
                    {
                    _loop4565:
                    do {
                            if ((LA(1)==COMA)) {
                                    match(COMA);
                                    primary_expression();
                                    astFactory.addASTChild(currentAST, returnAST);
                            }
                            else {
                                    break _loop4565;
                            }
```

```
                } while (true);
                }
                match(RPAREN);
                print_statement_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
        }
        returnAST = print_statement_AST;
}

public final void load_statement() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST load_statement_AST = null;

        try {      // for error handling
                AST tmp17_AST = null;
                tmp17_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp17_AST);
                match(LOAD);
                match(LPAREN);
                AST tmp19_AST = null;
                tmp19_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp19_AST);
                match(ID);
                match(COMA);
                {
                switch ( LA(1)) {
                case ID:
                {
                        AST tmp21_AST = null;
                        tmp21_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp21_AST);
                        match(ID);
                        break;
                }
                case STRING_LITERAL:
                {
                        AST tmp22_AST = null;
                        tmp22_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp22_AST);
                        match(STRING_LITERAL);
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                match(RPAREN);
                load_statement_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
```

```
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
            }
            returnAST = load_statement_AST;
        }

        public final void fetch_statement() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST fetch_statement_AST = null;

            try {      // for error handling
                fetch_clause();
                astFactory.addASTChild(currentAST, returnAST);
                using_clause();
                astFactory.addASTChild(currentAST, returnAST);
                fetch_statement_AST = (AST)currentAST.root;
                fetch_statement_AST =
                        (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NFetch_statement,"NFetch_statement")).add(fetch_statement_AST));
                currentAST.root = fetch_statement_AST;
                currentAST.child = fetch_statement_AST!=null
&&fetch_statement_AST.getFirstChild()!=null ?
                    fetch_statement_AST.getFirstChild() : fetch_statement_AST;
                currentAST.advanceChildToEnd();
                fetch_statement_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
            }
            returnAST = fetch_statement_AST;
        }

        public final void select_statement() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST select_statement_AST = null;

            try {      // for error handling
                select_clause();
                astFactory.addASTChild(currentAST, returnAST);
                where_clause();
                astFactory.addASTChild(currentAST, returnAST);
                using_clause();
                astFactory.addASTChild(currentAST, returnAST);
                select_statement_AST = (AST)currentAST.root;
                select_statement_AST =
                        (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NSelect_statement,"NSelect_statement")).add(select_statement_AST))
;
                currentAST.root = select_statement_AST;
                currentAST.child = select_statement_AST!=null
&&select_statement_AST.getFirstChild()!=null ?
```

```
                        select_statement_AST.getFirstChild() : select_statement_AST;
                currentAST.advanceChildToEnd();
                select_statement_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
            }
            returnAST = select_statement_AST;
        }

        public final void primary_expression() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST primary_expression_AST = null;

            try {      // for error handling
                {
                switch ( LA(1)) {
                case ID:
                {
                        AST tmp24_AST = null;
                        tmp24_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp24_AST);
                        match(ID);
                        {
                        switch ( LA(1)) {
                        case LBRACKET:
                        {
                                match(LBRACKET);
                                expression();
                                astFactory.addASTChild(currentAST, returnAST);
                                match(RBRACKET);
                                primary_expression_AST = (AST)currentAST.root;
                                        primary_expression_AST   =   (AST)astFactory.make(
(new
ASTArray(2)).add(astFactory.create(NListDeclaration,"NListDeclaration")).add(primary_expression_AST));
                                currentAST.root = primary_expression_AST;
                                currentAST.child          =          primary_expression_AST!=null
&&primary_expression_AST.getFirstChild()!=null ?
                                        primary_expression_AST.getFirstChild()                :
primary_expression_AST;
                                currentAST.advanceChildToEnd();
                                break;
                        }
                        case COMA:
                        case RPAREN:
                        case SEMI:
                        case RBRACKET:
                        case LOR:
                        case LAND:
                        case EQUAL:
                        case NOT_EQUAL:
                        case LT:
                        case GT:
                        case LTE:
```

```
		case GTE:
		case PLUS:
		case MINUS:
		case STAR:
		case DIV:
		case MOD:
		{
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		break;
}
case STRING_LITERAL:
{
		AST tmp27_AST = null;
		tmp27_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp27_AST);
		match(STRING_LITERAL);
		break;
}
case DOUBLE_CONST:
{
		AST tmp28_AST = null;
		tmp28_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp28_AST);
		match(DOUBLE_CONST);
		break;
}
case INT_CONST:
{
		AST tmp29_AST = null;
		tmp29_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp29_AST);
		match(INT_CONST);
		break;
}
case LPAREN:
{
		AST tmp30_AST = null;
		tmp30_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp30_AST);
		match(LPAREN);
		expression();
		astFactory.addASTChild(currentAST, returnAST);
		match(RPAREN);
		break;
}
default:
{
		throw new NoViableAltException(LT(1), getFilename());
}
}
}
```

```
                primary_expression_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_5);
        }
        returnAST = primary_expression_AST;
}

public final void fetch_clause() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST fetch_clause_AST = null;

        try {      // for error handling
                AST tmp32_AST = null;
                tmp32_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp32_AST);
                match(FETCH);
                AST tmp33_AST = null;
                tmp33_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp33_AST);
                match(ID);
                {
                _loop4571:
                do {
                        if ((LA(1)==COMA)) {
                                match(COMA);
                                AST tmp35_AST = null;
                                tmp35_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp35_AST);
                                match(ID);
                        }
                        else {
                                break _loop4571;
                        }

                } while (true);
                }
                fetch_clause_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_6);
        }
        returnAST = fetch_clause_AST;
}

public final void using_clause() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST using_clause_AST = null;

        try {      // for error handling
```

```
                AST tmp36_AST = null;
                tmp36_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp36_AST);
                match(USING);
                AST tmp37_AST = null;
                tmp37_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp37_AST);
                match(ID);
                using_clause_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
        }
        returnAST = using_clause_AST;
}

public final void select_clause() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST select_clause_AST = null;

        try {      // for error handling
                AST tmp38_AST = null;
                tmp38_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp38_AST);
                match(SELECT);
                AST tmp39_AST = null;
                tmp39_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp39_AST);
                match(ID);
                {
                match(COMA);
                AST tmp41_AST = null;
                tmp41_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp41_AST);
                match(ID);
                }
                select_clause_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_7);
        }
        returnAST = select_clause_AST;
}

public final void where_clause() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST where_clause_AST = null;

        try {      // for error handling
                AST tmp42_AST = null;
```

```
                tmp42_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp42_AST);
                match(WHERE);
                AST tmp43_AST = null;
                tmp43_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp43_AST);
                match(ID);
                where_clause_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_6);
        }
        returnAST = where_clause_AST;
}

public final void expression() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expression_AST = null;

        try {      // for error handling
                logical_and_expression();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop4592:
                do {
                        if ((LA(1)==LOR)) {
                                AST tmp44_AST = null;
                                tmp44_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp44_AST);
                                match(LOR);
                                logical_and_expression();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop4592;
                        }

                } while (true);
                }
                expression_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_8);
        }
        returnAST = expression_AST;
}

public final void whilLoop() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST whilLoop_AST = null;
```

```
		try {	// for error handling
			AST tmp45_AST = null;
			tmp45_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp45_AST);
			match(WHILE);
			match(LPAREN);
			expression();
			astFactory.addASTChild(currentAST, returnAST);
			match(RPAREN);
			statement_list();
			astFactory.addASTChild(currentAST, returnAST);
			whilLoop_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_3);
		}
		returnAST = whilLoop_AST;
}

public final void assign_expression() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST assign_expression_AST = null;
		AST a_AST = null;

		try {	// for error handling
			assign_ID();
			astFactory.addASTChild(currentAST, returnAST);
			{
			switch ( LA(1)) {
			case INC_OP:
			{
				AST tmp48_AST = null;
				tmp48_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp48_AST);
				match(INC_OP);
				break;
			}
			case DEC_OP:
			{
				AST tmp49_AST = null;
				tmp49_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp49_AST);
				match(DEC_OP);
				break;
			}
			case ASSIGN:
			case ADD_ASSIGN:
			case SUB_ASSIGN:
			case DIV_ASSIGN:
			case STAR_ASSIGN:
			case MOD_ASSIGN:
			{
				assignment_operator();
```

```
                                        a_AST = (AST)returnAST;
                                        expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                        assign_expression_AST = (AST)currentAST.root;
                                        assign_expression_AST = (AST)astFactory.make( (new
ASTArray(2)).add(a_AST).add(assign_expression_AST));
                                        currentAST.root = assign_expression_AST;
                                        currentAST.child = assign_expression_AST!=null
&&assign_expression_AST.getFirstChild()!=null ?
                                                assign_expression_AST.getFirstChild() :
assign_expression_AST;
                                        currentAST.advanceChildToEnd();
                                        break;
                                }
                                default:
                                {
                                        throw new NoViableAltException(LT(1), getFilename());
                                }
                                }
                                }
                                assign_expression_AST = (AST)currentAST.root;
                        }
                        catch (RecognitionException ex) {
                                reportError(ex);
                                consume();
                                consumeUntil(_tokenSet_9);
                        }
                        returnAST = assign_expression_AST;
                }

                public final void assign_ID() throws RecognitionException, TokenStreamException {

                        returnAST = null;
                        ASTPair currentAST = new ASTPair();
                        AST assign_ID_AST = null;

                        try {      // for error handling
                                AST tmp50_AST = null;
                                tmp50_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp50_AST);
                                match(ID);
                                {
                                switch ( LA(1)) {
                                case LBRACKET:
                                {
                                        match(LBRACKET);
                                        expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                        match(RBRACKET);
                                        assign_ID_AST = (AST)currentAST.root;
                                        assign_ID_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NAssignList,"NAssignList")).add(assign_ID_AST));
                                        currentAST.root = assign_ID_AST;
                                        currentAST.child = assign_ID_AST!=null
&&assign_ID_AST.getFirstChild()!=null ?
                                                assign_ID_AST.getFirstChild() : assign_ID_AST;
                                        currentAST.advanceChildToEnd();
                                        break;
```

```
                }
                case INC_OP:
                case DEC_OP:
                case ASSIGN:
                case ADD_ASSIGN:
                case SUB_ASSIGN:
                case DIV_ASSIGN:
                case STAR_ASSIGN:
                case MOD_ASSIGN:
                {
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                assign_ID_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_10);
        }
        returnAST = assign_ID_AST;
}

public final void assignment_operator() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST assignment_operator_AST = null;

        try {      // for error handling
                {
                switch ( LA(1)) {
                case ASSIGN:
                {
                        AST tmp53_AST = null;
                        tmp53_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp53_AST);
                        match(ASSIGN);
                        break;
                }
                case ADD_ASSIGN:
                {
                        AST tmp54_AST = null;
                        tmp54_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp54_AST);
                        match(ADD_ASSIGN);
                        break;
                }
                case SUB_ASSIGN:
                {
                        AST tmp55_AST = null;
                        tmp55_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp55_AST);
```

```
                        match(SUB_ASSIGN);
                        break;
                }
                case DIV_ASSIGN:
                {
                        AST tmp56_AST = null;
                        tmp56_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp56_AST);
                        match(DIV_ASSIGN);
                        break;
                }
                case STAR_ASSIGN:
                {
                        AST tmp57_AST = null;
                        tmp57_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp57_AST);
                        match(STAR_ASSIGN);
                        break;
                }
                case MOD_ASSIGN:
                {
                        AST tmp58_AST = null;
                        tmp58_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp58_AST);
                        match(MOD_ASSIGN);
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                assignment_operator_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_11);
        }
        returnAST = assignment_operator_AST;
}

public final void logical_and_expression() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST logical_and_expression_AST = null;

        try {      // for error handling
                equality_expression();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop4595:
                do {
                        if ((LA(1)==LAND)) {
                                AST tmp59_AST = null;
                                tmp59_AST = astFactory.create(LT(1));
```

```
                                        astFactory.makeASTRoot(currentAST, tmp59_AST);
                                        match(LAND);
                                        equality_expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                        break _loop4595;
                                }

                        } while (true);
                        }
                        logical_and_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_12);
                }
                returnAST = logical_and_expression_AST;
        }

        public final void equality_expression() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST equality_expression_AST = null;

                try {      // for error handling
                        relational_expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        _loop4599:
                        do {
                                if ((LA(1)==EQUAL||LA(1)==NOT_EQUAL)) {
                                        {
                                        switch ( LA(1)) {
                                        case EQUAL:
                                        {
                                                AST tmp60_AST = null;
                                                tmp60_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp60_AST);
                                                match(EQUAL);
                                                break;
                                        }
                                        case NOT_EQUAL:
                                        {
                                                AST tmp61_AST = null;
                                                tmp61_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp61_AST);
                                                match(NOT_EQUAL);
                                                break;
                                        }
                                        default:
                                        {
                                                throw new NoViableAltException(LT(1),
getFilename());
                                        }
                                        }
```

```
                                        }
                                        relational_expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                        break _loop4599;
                                }

                        } while (true);
                        }
                        equality_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_13);
                }
                returnAST = equality_expression_AST;
}

public final void relational_expression() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST relational_expression_AST = null;

                try {      // for error handling
                        additive_expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        _loop4603:
                        do {
                                if (((LA(1) >= LT && LA(1) <= GTE))) {
                                        {
                                        switch ( LA(1)) {
                                        case LT:
                                        {
                                                AST tmp62_AST = null;
                                                tmp62_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp62_AST);
                                                match(LT);
                                                break;
                                        }
                                        case GT:
                                        {
                                                AST tmp63_AST = null;
                                                tmp63_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp63_AST);
                                                match(GT);
                                                break;
                                        }
                                        case LTE:
                                        {
                                                AST tmp64_AST = null;
                                                tmp64_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp64_AST);
                                                match(LTE);
                                                break;
```

```
                                        }
                                        case GTE:
                                        {
                                                AST tmp65_AST = null;
                                                tmp65_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp65_AST);
                                                match(GTE);
                                                break;
                                        }
                                        default:
                                        {
                                                throw new NoViableAltException(LT(1),
getFilename());
                                        }
                                        }
                                        }
                                        additive_expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                        break _loop4603;
                                }

                        } while (true);
                        }
                        relational_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_14);
                }
                returnAST = relational_expression_AST;
        }

        public final void additive_expression() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST additive_expression_AST = null;

                try {      // for error handling
                        multiplicative_expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        _loop4607:
                        do {
                                if ((LA(1)==PLUS||LA(1)==MINUS)) {
                                        {
                                        switch ( LA(1)) {
                                        case PLUS:
                                        {
                                                AST tmp66_AST = null;
                                                tmp66_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp66_AST);
                                                match(PLUS);
                                                break;
                                        }
```

```
                                        case MINUS:
                                        {
                                                AST tmp67_AST = null;
                                                tmp67_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp67_AST);
                                                match(MINUS);
                                                break;
                                        }
                                        default:
                                        {
                                                throw new NoViableAltException(LT(1),
getFilename());
                                        }
                                        }
                                        }
                                        multiplicative_expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                        break _loop4607;
                                }

                        } while (true);
                        }
                        additive_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_15);
                }
                returnAST = additive_expression_AST;
        }

        public final void multiplicative_expression() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST multiplicative_expression_AST = null;

                try {      // for error handling
                        unary_expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        _loop4611:
                        do {
                                if (((LA(1) >= STAR && LA(1) <= MOD))) {
                                        {
                                        switch ( LA(1)) {
                                        case STAR:
                                        {
                                                AST tmp68_AST = null;
                                                tmp68_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp68_AST);
                                                match(STAR);
                                                break;
                                        }
                                        case DIV:
```

```
                                        {
                                                AST tmp69_AST = null;
                                                tmp69_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp69_AST);
                                                match(DIV);
                                                break;
                                        }
                                        case MOD:
                                        {
                                                AST tmp70_AST = null;
                                                tmp70_AST = astFactory.create(LT(1));
                                                astFactory.makeASTRoot(currentAST, tmp70_AST);
                                                match(MOD);
                                                break;
                                        }
                                        default:
                                        {
                                                throw new NoViableAltException(LT(1),
getFilename());
                                        }
                                        }
                                        }
                                        unary_expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                        break _loop4611;
                                }

                        } while (true);
                        }
                        multiplicative_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_16);
                }
                returnAST = multiplicative_expression_AST;
        }

        public final void unary_expression() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST unary_expression_AST = null;
                AST u_AST = null;

                try {      // for error handling
                        {
                        switch ( LA(1)) {
                        case LPAREN:
                        case ID:
                        case STRING_LITERAL:
                        case DOUBLE_CONST:
                        case INT_CONST:
                        {
                                primary_expression();
```

```
                                astFactory.addASTChild(currentAST, returnAST);
                                break;
                        }
                        case PLUS:
                        case MINUS:
                        case LNOT:
                        {
                                unary_operator();
                                u_AST = (AST)returnAST;
                                astFactory.addASTChild(currentAST, returnAST);
                                primary_expression();
                                astFactory.addASTChild(currentAST, returnAST);
                                unary_expression_AST = (AST)currentAST.root;
                                unary_expression_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NUnary,"NUnary")).add(unary_expression_AST));
                                currentAST.root = unary_expression_AST;
                                currentAST.child = unary_expression_AST!=null
&&unary_expression_AST.getFirstChild()!=null ?
                                        unary_expression_AST.getFirstChild() :
unary_expression_AST;
                                currentAST.advanceChildToEnd();
                                break;
                        }
                        default:
                        {
                                throw new NoViableAltException(LT(1), getFilename());
                        }
                        }
                        }
                        unary_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_17);
                }
                returnAST = unary_expression_AST;
        }

        public final void unary_operator() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST unary_operator_AST = null;

                try {      // for error handling
                        switch ( LA(1)) {
                        case PLUS:
                        {
                                AST tmp71_AST = null;
                                tmp71_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp71_AST);
                                match(PLUS);
                                unary_operator_AST = (AST)currentAST.root;
                                break;
                        }
                        case MINUS:
                        {
```

```
                        AST tmp72_AST = null;
                        tmp72_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp72_AST);
                        match(MINUS);
                        unary_operator_AST = (AST)currentAST.root;
                        break;
                }
                case LNOT:
                {
                        AST tmp73_AST = null;
                        tmp73_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp73_AST);
                        match(LNOT);
                        unary_operator_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_18);
        }
        returnAST = unary_operator_AST;
}

public final void declaration() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST declaration_AST = null;

        try {      // for error handling
                type_specifier();
                astFactory.addASTChild(currentAST, returnAST);
                AST tmp74_AST = null;
                tmp74_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp74_AST);
                match(ID);
                {
                switch ( LA(1)) {
                case LBRACKET:
                {
                        match(LBRACKET);
                        expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(RBRACKET);
                        break;
                }
                case SEMI:
                {
                        break;
                }
                default:
```

```
                        {
                                throw new NoViableAltException(LT(1), getFilename());
                        }
                        }
                        }
                        match(SEMI);
                        declaration_AST = (AST)currentAST.root;
                        declaration_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NDeclaration,"NDeclaration")).add(declaration_AST));
                        currentAST.root = declaration_AST;
                        currentAST.child = declaration_AST!=null
&&declaration_AST.getFirstChild()!=null ?
                                declaration_AST.getFirstChild() : declaration_AST;
                        currentAST.advanceChildToEnd();
                        declaration_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_19);
                }
                returnAST = declaration_AST;
        }

        public final void type_specifier() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST type_specifier_AST = null;

                try {      // for error handling
                        {
                        switch ( LA(1)) {
                        case INT:
                        {
                                AST tmp78_AST = null;
                                tmp78_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp78_AST);
                                match(INT);
                                break;
                        }
                        case DOUBLE:
                        {
                                AST tmp79_AST = null;
                                tmp79_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp79_AST);
                                match(DOUBLE);
                                break;
                        }
                        case STRING:
                        {
                                AST tmp80_AST = null;
                                tmp80_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp80_AST);
                                match(STRING);
                                break;
                        }
                        case CURSOR:
```

```
                {
                        AST tmp81_AST = null;
                        tmp81_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp81_AST);
                        match(CURSOR);
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                type_specifier_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_20);
        }
        returnAST = type_specifier_AST;
}

public final void imaginary_token() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST imaginary_token_AST = null;

        try {     // for error handling
                switch ( LA(1)) {
                case NRootAST:
                {
                        AST tmp82_AST = null;
                        tmp82_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp82_AST);
                        match(NRootAST);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
                case NDeclarator:
                {
                        AST tmp83_AST = null;
                        tmp83_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp83_AST);
                        match(NDeclarator);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
                case NDeclaration:
                {
                        AST tmp84_AST = null;
                        tmp84_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp84_AST);
                        match(NDeclaration);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
```

```
case NUnionDeclaration:
{
		AST tmp85_AST = null;
		tmp85_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp85_AST);
		match(NUnionDeclaration);
		imaginary_token_AST = (AST)currentAST.root;
		break;
}
case NUnary:
{
		AST tmp86_AST = null;
		tmp86_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp86_AST);
		match(NUnary);
		imaginary_token_AST = (AST)currentAST.root;
		break;
}
case NFetch_statement:
{
		AST tmp87_AST = null;
		tmp87_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp87_AST);
		match(NFetch_statement);
		imaginary_token_AST = (AST)currentAST.root;
		break;
}
case NSelect_statement:
{
		AST tmp88_AST = null;
		tmp88_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp88_AST);
		match(NSelect_statement);
		imaginary_token_AST = (AST)currentAST.root;
		break;
}
case NEmptyStatement:
{
		AST tmp89_AST = null;
		tmp89_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp89_AST);
		match(NEmptyStatement);
		imaginary_token_AST = (AST)currentAST.root;
		break;
}
case NInitializer:
{
		AST tmp90_AST = null;
		tmp90_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp90_AST);
		match(NInitializer);
		imaginary_token_AST = (AST)currentAST.root;
		break;
}
case NExterntype:
{
		AST tmp91_AST = null;
		tmp91_AST = astFactory.create(LT(1));
```

```
            astFactory.addASTChild(currentAST, tmp91_AST);
            match(NExterntype);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NCodeBlock:
{
            AST tmp92_AST = null;
            tmp92_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp92_AST);
            match(NCodeBlock);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NForCon:
{
            AST tmp93_AST = null;
            tmp93_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp93_AST);
            match(NForCon);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NForBlock:
{
            AST tmp94_AST = null;
            tmp94_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp94_AST);
            match(NForBlock);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NForDeclaration:
{
            AST tmp95_AST = null;
            tmp95_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp95_AST);
            match(NForDeclaration);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NDeclareList:
{
            AST tmp96_AST = null;
            tmp96_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp96_AST);
            match(NDeclareList);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NListDeclaration:
{
            AST tmp97_AST = null;
            tmp97_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp97_AST);
            match(NListDeclaration);
            imaginary_token_AST = (AST)currentAST.root;
            break;
```

```
                }
                case NAssignList:
                {
                        AST tmp98_AST = null;
                        tmp98_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp98_AST);
                        match(NAssignList);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
                case NStartCondition:
                {
                        AST tmp99_AST = null;
                        tmp99_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp99_AST);
                        match(NStartCondition);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
                case NStatement:
                {
                        AST tmp100_AST = null;
                        tmp100_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp100_AST);
                        match(NStatement);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_0);
        }
        returnAST = imaginary_token_AST;
}


public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "'{'",
        "'}'",
        "\"print\"",
        "'('",
        "COMA",
        "')'",
        "\"load\"",
        "an identifier",
        "STRING_LITERAL",
        "\"fetch\"",
```

```
"\"select\"",
"WHERE",
"\"using\"",
"\"if\"",
"\"else\"",
"\"break\"",
"';'",
"\"continue\"",
"\"while\"",
"INC_OP",
"DEC_OP",
"'['",
"']'",
"ASSIGN",
"ADD_ASSIGN",
"SUB_ASSIGN",
"DIV_ASSIGN",
"STAR_ASSIGN",
"MOD_ASSIGN",
"LOR",
"LAND",
"EQUAL",
"NOT_EQUAL",
"LT",
"GT",
"LTE",
"GTE",
"PLUS",
"MINUS",
"STAR",
"DIV",
"MOD",
"LNOT",
"DOUBLE_CONST",
"INT_CONST",
"\"int\"",
"\"double\"",
"\"string\"",
"\"cursor\"",
"NRootAST",
"NDeclarator",
"NDeclaration",
"NUnionDeclaration",
"NUnary",
"NFetch_statement",
"NSelect_statement",
"NEmptyStatement",
"NInitializer",
"NExterntype",
"NCodeBlock",
"NForCon",
"NForBlock",
"NForDeclaration",
"NDeclareList",
"NListDeclaration",
"NAssignList",
"NStartCondition",
"NStatement",
```

```
		"\"for\"",
		"\"return\"",
		"\"from\"",
		"WS",
		"NL",
		"COMMENT",
		"':'",
		"COMMA",
		"DOT",
		"'<<'",
		"Escape",
		"DIGIT",
		"a number"
};

protected void buildTokenTypeASTClassMap() {
		tokenTypeToASTClassMap=null;
};

private static final long[] mk_tokenSet_0() {
		long[] data = { 2L, 0L};
		return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
		long[] data = { 6974544L, 0L};
		return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
private static final long[] mk_tokenSet_2() {
		long[] data = { 8522460400L, 0L};
		return data;
}
public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());
private static final long[] mk_tokenSet_3() {
		long[] data = { 7236722L, 0L};
		return data;
}
public static final BitSet _tokenSet_3 = new BitSet(mk_tokenSet_3());
private static final long[] mk_tokenSet_4() {
		long[] data = { 8522722546L, 0L};
		return data;
}
public static final BitSet _tokenSet_4 = new BitSet(mk_tokenSet_4());
private static final long[] mk_tokenSet_5() {
		long[] data = { 70360222401280L, 0L};
		return data;
}
public static final BitSet _tokenSet_5 = new BitSet(mk_tokenSet_5());
private static final long[] mk_tokenSet_6() {
		long[] data = { 65536L, 0L};
		return data;
}
public static final BitSet _tokenSet_6 = new BitSet(mk_tokenSet_6());
private static final long[] mk_tokenSet_7() {
		long[] data = { 32768L, 0L};
		return data;
}
```

```
public static final BitSet _tokenSet_7 = new BitSet(mk_tokenSet_7());
private static final long[] mk_tokenSet_8() {
	long[] data = { 68157952L, 0L};
	return data;
}
public static final BitSet _tokenSet_8 = new BitSet(mk_tokenSet_8());
private static final long[] mk_tokenSet_9() {
	long[] data = { 1048576L, 0L};
	return data;
}
public static final BitSet _tokenSet_9 = new BitSet(mk_tokenSet_9());
private static final long[] mk_tokenSet_10() {
	long[] data = { 8480882688L, 0L};
	return data;
}
public static final BitSet _tokenSet_10 = new BitSet(mk_tokenSet_10());
private static final long[] mk_tokenSet_11() {
	long[] data = { 499178279016576L, 0L};
	return data;
}
public static final BitSet _tokenSet_11 = new BitSet(mk_tokenSet_11());
private static final long[] mk_tokenSet_12() {
	long[] data = { 8658092544L, 0L};
	return data;
}
public static final BitSet _tokenSet_12 = new BitSet(mk_tokenSet_12());
private static final long[] mk_tokenSet_13() {
	long[] data = { 25837961728L, 0L};
	return data;
}
public static final BitSet _tokenSet_13 = new BitSet(mk_tokenSet_13());
private static final long[] mk_tokenSet_14() {
	long[] data = { 128917176832L, 0L};
	return data;
}
public static final BitSet _tokenSet_14 = new BitSet(mk_tokenSet_14());
private static final long[] mk_tokenSet_15() {
	long[] data = { 2190501478912L, 0L};
	return data;
}
public static final BitSet _tokenSet_15 = new BitSet(mk_tokenSet_15());
private static final long[] mk_tokenSet_16() {
	long[] data = { 8787571245568L, 0L};
	return data;
}
public static final BitSet _tokenSet_16 = new BitSet(mk_tokenSet_16());
private static final long[] mk_tokenSet_17() {
	long[] data = { 70360222401024L, 0L};
	return data;
}
public static final BitSet _tokenSet_17 = new BitSet(mk_tokenSet_17());
private static final long[] mk_tokenSet_18() {
	long[] data = { 422212465072256L, 0L};
	return data;
}
public static final BitSet _tokenSet_18 = new BitSet(mk_tokenSet_18());
private static final long[] mk_tokenSet_19() {
	long[] data = { 8444249308294224L, 0L};
```

```
		return data;
	}
	public static final BitSet _tokenSet_19 = new BitSet(mk_tokenSet_19());
	private static final long[] mk_tokenSet_20() {
		long[] data = { 2048L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_20 = new BitSet(mk_tokenSet_20());

	}

// start of XQAntlrLexer.java
// $ANTLR : "parser.g" -> "XQAntlrLexer.java"$

public interface XQAntlrTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int LCURLY = 4;
	int RCURLY = 5;
	int PRINT = 6;
	int LPAREN = 7;
	int COMA = 8;
	int RPAREN = 9;
	int LOAD = 10;
	int ID = 11;
	int STRING_LITERAL = 12;
	int FETCH = 13;
	int SELECT = 14;
	int WHERE = 15;
	int USING = 16;
	int IF = 17;
	int ELSE = 18;
	int BREAK = 19;
	int SEMI = 20;
	int CONTINUE = 21;
	int WHILE = 22;
	int INC_OP = 23;
	int DEC_OP = 24;
	int LBRACKET = 25;
	int RBRACKET = 26;
	int ASSIGN = 27;
	int ADD_ASSIGN = 28;
	int SUB_ASSIGN = 29;
	int DIV_ASSIGN = 30;
	int STAR_ASSIGN = 31;
	int MOD_ASSIGN = 32;
	int LOR = 33;
	int LAND = 34;
	int EQUAL = 35;
	int NOT_EQUAL = 36;
	int LT = 37;
	int GT = 38;
	int LTE = 39;
	int GTE = 40;
	int PLUS = 41;
	int MINUS = 42;
	int STAR = 43;
	int DIV = 44;
```

```
        int MOD = 45;
        int LNOT = 46;
        int DOUBLE_CONST = 47;
        int INT_CONST = 48;
        int INT = 49;
        int DOUBLE = 50;
        int STRING = 51;
        int CURSOR = 52;
        int NRootAST = 53;
        int NDeclarator = 54;
        int NDeclaration = 55;
        int NUnionDeclaration = 56;
        int NUnary = 57;
        int NFetch_statement = 58;
        int NSelect_statement = 59;
        int NEmptyStatement = 60;
        int NInitializer = 61;
        int NExterntype = 62;
        int NCodeBlock = 63;
        int NForCon = 64;
        int NForBlock = 65;
        int NForDeclaration = 66;
        int NDeclareList = 67;
        int NListDeclaration = 68;
        int NAssignList = 69;
        int NStartCondition = 70;
        int NStatement = 71;
        int FOR = 72;
        int RETURN = 73;
        int FROM = 74;
        int WS = 75;
        int NL = 76;
        int COMMENT = 77;
        int COLON = 78;
        int COMMA = 79;
        int DOT = 80;
        int LSHIFT = 81;
        int Escape = 82;
        int DIGIT = 83;
        int NUMBER = 84;
}

// start of  "XQAntlrWalker.java"
// $ANTLR : "walker.g" -> "XQAntlrWalker.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

import java.io.*;
```

```
import java.util.*;


public class XQAntlrWalker extends antlr.TreeParser       implements XQAntlrWalkerTokenTypes
 {

    static XQDataType null_data = new XQDataType( "<NULL>" );
    XQInterpreter ipt = new XQInterpreter();
public XQAntlrWalker() {
        tokenNames = _tokenNames;
}

        public final void start_condition(AST _t) throws RecognitionException {

                AST start_condition_AST_in = (AST)_t;

                try {      // for error handling
                    AST __t4705 = _t;
                    AST tmp1_AST_in = (AST)_t;
                    match(_t,NStartCondition);
                    _t = _t.getFirstChild();
                    declaration_list(_t);
                    _t = _retTree;
                    statement_list(_t);
                    _t = _retTree;
                    _t = __t4705;
                    _t = _t.getNextSibling();
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    if (_t!=null) {_t = _t.getNextSibling();}
                }
                _retTree = _t;
        }

        public final void declaration_list(AST _t) throws RecognitionException {

                AST declaration_list_AST_in = (AST)_t;

                try {      // for error handling
                    {
                    int _cnt4751=0;
                    _loop4751:
                    do {
                        if (_t==null) _t=ASTNULL;
                        if ((_t.getType()==NDeclaration)) {
                            declaration(_t);
                            _t = _retTree;
                        }
                        else {
                            if  (  _cnt4751>=1  )  {  break  _loop4751;  }  else  {throw  new
NoViableAltException(_t);}
                        }

                        _cnt4751++;
                    } while (true);
                    }
                }
```

```
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		_retTree = _t;
	}

	public final void statement_list(AST _t) throws RecognitionException {

		AST statement_list_AST_in = (AST)_t;

		try {	// for error handling
			{
			int _cnt4708=0;
			_loop4708:
			do {
				if (_t==null) _t=ASTNULL;
				if ((_tokenSet_0.member(_t.getType()))) {
					statement(_t);
					_t = _retTree;
				}
				else {
					if ( _cnt4708>=1 ) { break _loop4708; } else {throw new
NoViableAltException(_t);}
				}

				_cnt4708++;
			} while (true);
			}
		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		_retTree = _t;
	}

	public final void statement(AST _t) throws RecognitionException {

		AST statement_AST_in = (AST)_t;

		try {	// for error handling
			if (_t==null) _t=ASTNULL;
			switch ( _t.getType()) {
			case INC_OP:
			case DEC_OP:
			case ASSIGN:
			case ADD_ASSIGN:
			case SUB_ASSIGN:
			case DIV_ASSIGN:
			case STAR_ASSIGN:
			case MOD_ASSIGN:
			{
				assign_statement(_t);
				_t = _retTree;
				break;
			}
			case IF:
```

```
                {
                        if_statement(_t);
                        _t = _retTree;
                        break;
                }
                case WHILE:
                {
                        iteration_statement(_t);
                        _t = _retTree;
                        break;
                }
                case BREAK:
                {
                        break_statement(_t);
                        _t = _retTree;
                        break;
                }
                case CONTINUE:
                {
                        continue_statement(_t);
                        _t = _retTree;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
}

public final void assign_statement(AST _t) throws RecognitionException {

        AST assign_statement_AST_in = (AST)_t;

        try {      // for error handling
                assign_expression(_t);
                _t = _retTree;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
}

public final void assign_expression(AST _t) throws RecognitionException {

        AST assign_expression_AST_in = (AST)_t;

                XQDataType aid, e;
                XQInt i = new XQInt(1);
```

```
		try {	// for error handling
			{
			if (_t==null) _t=ASTNULL;
			switch ( _t.getType()) {
			case ASSIGN:
			{
				AST __t4712 = _t;
				AST tmp2_AST_in = (AST)_t;
				match(_t,ASSIGN);
				_t = _t.getFirstChild();
				aid=assign_ID(_t);
				_t = _retTree;
				e=expression(_t);
				_t = _retTree;
				_t = __t4712;
				_t = _t.getNextSibling();
				aid.print();		System.out.println("before		assign-->"+aid.name);
ipt.assign(aid, e);
								System.out.println("after assign");
				break;
			}
			case ADD_ASSIGN:
			{
				AST __t4713 = _t;
				AST tmp3_AST_in = (AST)_t;
				match(_t,ADD_ASSIGN);
				_t = _t.getFirstChild();
				aid=assign_ID(_t);
				_t = _retTree;
				e=expression(_t);
				_t = _retTree;
				_t = __t4713;
				_t = _t.getNextSibling();
				aid.add(e);
				break;
			}
			case SUB_ASSIGN:
			{
				AST __t4714 = _t;
				AST tmp4_AST_in = (AST)_t;
				match(_t,SUB_ASSIGN);
				_t = _t.getFirstChild();
				aid=assign_ID(_t);
				_t = _retTree;
				e=expression(_t);
				_t = _retTree;
				_t = __t4714;
				_t = _t.getNextSibling();
				aid.sub(e);
				break;
			}
			case DIV_ASSIGN:
			{
				AST __t4715 = _t;
				AST tmp5_AST_in = (AST)_t;
				match(_t,DIV_ASSIGN);
				_t = _t.getFirstChild();
```

```
            aid=assign_ID(_t);
            _t = _retTree;
            e=expression(_t);
            _t = _retTree;
            _t = __t4715;
            _t = _t.getNextSibling();
            aid.ldiv(e);
            break;
}
case STAR_ASSIGN:
{
            AST __t4716 = _t;
            AST tmp6_AST_in = (AST)_t;
            match(_t,STAR_ASSIGN);
            _t = _t.getFirstChild();
            aid=assign_ID(_t);
            _t = _retTree;
            e=expression(_t);
            _t = _retTree;
            _t = __t4716;
            _t = _t.getNextSibling();
            aid.mul(e);
            break;
}
case MOD_ASSIGN:
{
            AST __t4717 = _t;
            AST tmp7_AST_in = (AST)_t;
            match(_t,MOD_ASSIGN);
            _t = _t.getFirstChild();
            aid=assign_ID(_t);
            _t = _retTree;
            e=expression(_t);
            _t = _retTree;
            _t = __t4717;
            _t = _t.getNextSibling();
            aid.rem(e);
            break;
}
case INC_OP:
{
            AST __t4718 = _t;
            AST tmp8_AST_in = (AST)_t;
            match(_t,INC_OP);
            _t = _t.getFirstChild();
            aid=assign_ID(_t);
            _t = _retTree;
            _t = __t4718;
            _t = _t.getNextSibling();
            aid.add(i);
            break;
}
case DEC_OP:
{
            AST __t4719 = _t;
            AST tmp9_AST_in = (AST)_t;
            match(_t,DEC_OP);
            _t = _t.getFirstChild();
```

```
                                        aid=assign_ID(_t);
                                        _t = _retTree;
                                        _t = __t4719;
                                        _t = _t.getNextSibling();
                                        aid.sub(i);
                                        break;
                                }
                                default:
                                {
                                        throw new NoViableAltException(_t);
                                }
                                }
                                }
                        }
                        catch (RecognitionException ex) {
                                reportError(ex);
                                if (_t!=null) {_t = _t.getNextSibling();}
                        }
                        _retTree = _t;
                }

                public final  XQDataType  assign_ID(AST _t) throws RecognitionException {
                         XQDataType r ;

                        AST assign_ID_AST_in = (AST)_t;
                        AST sid = null;
                        AST id = null;

                        r = null_data;
                        XQDataType e;
                        XQDataType [] x;


                        try {      // for error handling
                                if (_t==null) _t=ASTNULL;
                                switch ( _t.getType()) {
                                case ID:
                                {
                                        sid = (AST)_t;
                                        match(_t,ID);
                                        _t = _t.getNextSibling();
                                        r = ipt.getVariable(sid.getText());
                                                                                System.out.println(sid.getText()+"--
"+r.name);
                                                                                r.print();

                                        break;
                                }
                                case NAssignList:
                                {
                                        AST __t4721 = _t;
                                        AST tmp10_AST_in = (AST)_t;
                                        match(_t,NAssignList);
                                        _t = _t.getFirstChild();
                                        id = (AST)_t;
                                        match(_t,ID);
                                        _t = _t.getNextSibling();
                                        e=expression(_t);
```

```
                        _t = _retTree;
                        _t = __t4721;
                        _t = _t.getNextSibling();
                        x = ipt.getArrayVariable(id.getText());
                                        r = x[XQInt.intValue(e)];

                        break;
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return r ;
}

public final  XQDataType  expression(AST _t) throws RecognitionException {
         XQDataType r ;

        AST expression_AST_in = (AST)_t;

        r = null_data;


        try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case LOR:
                {
                        r=logical_or_expression(_t);
                        _t = _retTree;
                        break;
                }
                case LAND:
                {
                        r=logical_and_expression(_t);
                        _t = _retTree;
                        break;
                }
                case EQUAL:
                case NOT_EQUAL:
                {
                        r=equality_expression(_t);
                        _t = _retTree;
                        break;
                }
                case LT:
                case GT:
                case LTE:
                case GTE:
                {
                        r=relational_expression(_t);
```

```
                    _t = _retTree;
                    break;
                }
                case PLUS:
                case MINUS:
                {
                    r=additive_expression(_t);
                    _t = _retTree;
                    break;
                }
                case STAR:
                case DIV:
                case MOD:
                {
                    r=multiplicative_expression(_t);
                    _t = _retTree;
                    break;
                }
                case NUnary:
                {
                    r=unary_expression(_t);
                    _t = _retTree;
                    break;
                }
                case LPAREN:
                case ID:
                case STRING_LITERAL:
                case DOUBLE_CONST:
                case INT_CONST:
                case NListDeclaration:
                {
                    r=primary_expression(_t);
                    _t = _retTree;
                    break;
                }
                default:
                {
                    throw new NoViableAltException(_t);
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return r ;
}

public final  XQDataType  logical_or_expression(AST _t) throws RecognitionException {
         XQDataType r ;

        AST logical_or_expression_AST_in = (AST)_t;

                XQDataType a, b;
                r = null_data;
```

```
		try {	// for error handling
			AST __t4724 = _t;
			AST tmp11_AST_in = (AST)_t;
			match(_t,LOR);
			_t = _t.getFirstChild();
			a=expression(_t);
			_t = _retTree;
			b=expression(_t);
			_t = _retTree;
			_t = __t4724;
			_t = _t.getNextSibling();
			r = a.or(b);
		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		_retTree = _t;
		return r ;
	}

	public final  XQDataType  logical_and_expression(AST _t) throws RecognitionException {
		 XQDataType r ;

		AST logical_and_expression_AST_in = (AST)_t;

			XQDataType a, b;
			r = null_data;


		try {	// for error handling
			AST __t4726 = _t;
			AST tmp12_AST_in = (AST)_t;
			match(_t,LAND);
			_t = _t.getFirstChild();
			a=expression(_t);
			_t = _retTree;
			b=expression(_t);
			_t = _retTree;
			_t = __t4726;
			_t = _t.getNextSibling();
			r = a.and(b);
		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		_retTree = _t;
		return r ;
	}

	public final  XQDataType  equality_expression(AST _t) throws RecognitionException {
		 XQDataType r ;

		AST equality_expression_AST_in = (AST)_t;

			XQDataType a, b;
```

```
		r = null_data;


	try {	// for error handling
		if (_t==null) _t=ASTNULL;
		switch ( _t.getType()) {
		case EQUAL:
		{
			AST __t4728 = _t;
			AST tmp13_AST_in = (AST)_t;
			match(_t,EQUAL);
			_t = _t.getFirstChild();
			a=expression(_t);
			_t = _retTree;
			b=expression(_t);
			_t = _retTree;
			_t = __t4728;
			_t = _t.getNextSibling();
			r = a.eq(b);
			break;
		}
		case NOT_EQUAL:
		{
			AST __t4729 = _t;
			AST tmp14_AST_in = (AST)_t;
			match(_t,NOT_EQUAL);
			_t = _t.getFirstChild();
			a=expression(_t);
			_t = _retTree;
			b=expression(_t);
			_t = _retTree;
			_t = __t4729;
			_t = _t.getNextSibling();
			r = a.ne(b);
			break;
		}
		default:
		{
			throw new NoViableAltException(_t);
		}
		}
	}
	catch (RecognitionException ex) {
		reportError(ex);
		if (_t!=null) {_t = _t.getNextSibling();}
	}
	_retTree = _t;
	return r ;
}

public final  XQDataType  relational_expression(AST _t) throws RecognitionException {
	 XQDataType r ;

	AST relational_expression_AST_in = (AST)_t;

		XQDataType a, b;
		r = null_data;
```

```
try {      // for error handling
          if (_t==null) _t=ASTNULL;
          switch ( _t.getType()) {
          case LT:
          {
                    AST __t4731 = _t;
                    AST tmp15_AST_in = (AST)_t;
                    match(_t,LT);
                    _t = _t.getFirstChild();
                    a=expression(_t);
                    _t = _retTree;
                    b=expression(_t);
                    _t = _retTree;
                    _t = __t4731;
                    _t = _t.getNextSibling();
                    r = a.lt(b);
                    break;
          }
          case GT:
          {
                    AST __t4732 = _t;
                    AST tmp16_AST_in = (AST)_t;
                    match(_t,GT);
                    _t = _t.getFirstChild();
                    a=expression(_t);
                    _t = _retTree;
                    b=expression(_t);
                    _t = _retTree;
                    _t = __t4732;
                    _t = _t.getNextSibling();
                    r = a.gt(b);
                    break;
          }
          case LTE:
          {
                    AST __t4733 = _t;
                    AST tmp17_AST_in = (AST)_t;
                    match(_t,LTE);
                    _t = _t.getFirstChild();
                    a=expression(_t);
                    _t = _retTree;
                    b=expression(_t);
                    _t = _retTree;
                    _t = __t4733;
                    _t = _t.getNextSibling();
                    r = a.le(b);
                    break;
          }
          case GTE:
          {
                    AST __t4734 = _t;
                    AST tmp18_AST_in = (AST)_t;
                    match(_t,GTE);
                    _t = _t.getFirstChild();
                    a=expression(_t);
                    _t = _retTree;
                    b=expression(_t);
```

```
                        _t = _retTree;
                        _t = __t4734;
                        _t = _t.getNextSibling();
                        r = a.ge(b);
                        break;
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return r ;
}

public final  XQDataType  additive_expression(AST _t) throws RecognitionException {
         XQDataType r ;

        AST additive_expression_AST_in = (AST)_t;

                XQDataType a, b;
                r = null_data;


        try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case PLUS:
                {
                        AST __t4736 = _t;
                        AST tmp19_AST_in = (AST)_t;
                        match(_t,PLUS);
                        _t = _t.getFirstChild();
                        a=expression(_t);
                        _t = _retTree;
                        b=expression(_t);
                        _t = _retTree;
                        _t = __t4736;
                        _t = _t.getNextSibling();
                        r = a.add(b);
                        break;
                }
                case MINUS:
                {
                        AST __t4737 = _t;
                        AST tmp20_AST_in = (AST)_t;
                        match(_t,MINUS);
                        _t = _t.getFirstChild();
                        a=expression(_t);
                        _t = _retTree;
                        b=expression(_t);
                        _t = _retTree;
                        _t = __t4737;
```

```
                        _t = _t.getNextSibling();
                        r = a.minus(b);
                        break;
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return r ;
}

public final  XQDataType  multiplicative_expression(AST _t) throws RecognitionException {
         XQDataType r ;

        AST multiplicative_expression_AST_in = (AST)_t;

                XQDataType a, b;
                r = null_data;


        try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case STAR:
                {
                        AST __t4739 = _t;
                        AST tmp21_AST_in = (AST)_t;
                        match(_t,STAR);
                        _t = _t.getFirstChild();
                        a=expression(_t);
                        _t = _retTree;
                        b=expression(_t);
                        _t = _retTree;
                        _t = __t4739;
                        _t = _t.getNextSibling();
                        r = a.times(b);
                        break;
                }
                case DIV:
                {
                        AST __t4740 = _t;
                        AST tmp22_AST_in = (AST)_t;
                        match(_t,DIV);
                        _t = _t.getFirstChild();
                        a=expression(_t);
                        _t = _retTree;
                        b=expression(_t);
                        _t = _retTree;
                        _t = __t4740;
                        _t = _t.getNextSibling();
                        r = a.lfracts(b);
```

```
                    break;
                }
                case MOD:
                {
                    AST __t4741 = _t;
                    AST tmp23_AST_in = (AST)_t;
                    match(_t,MOD);
                    _t = _t.getFirstChild();
                    a=expression(_t);
                    _t = _retTree;
                    b=expression(_t);
                    _t = _retTree;
                    _t = __t4741;
                    _t = _t.getNextSibling();
                    r = a.modulus(b);
                    break;
                }
                default:
                {
                    throw new NoViableAltException(_t);
                }
                }
            }
            catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            _retTree = _t;
            return r ;
}

public final  XQDataType  unary_expression(AST _t) throws RecognitionException {
             XQDataType r ;

            AST unary_expression_AST_in = (AST)_t;
            AST uo = null;

                r = null_data;


            try {      // for error handling
                AST __t4743 = _t;
                AST tmp24_AST_in = (AST)_t;
                match(_t,NUnary);
                _t = _t.getFirstChild();
                uo = _t==ASTNULL ? null : (AST)_t;
                unary_operator(_t);
                _t = _retTree;
                r=primary_expression(_t);
                _t = _retTree;
                _t = __t4743;
                _t = _t.getNextSibling();

                                    if (  uo.getType() == MINUS )
                                        r.uminus();
                                    else if (uo.getType() == LNOT )
                                        r.not();
```

```
		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		_retTree = _t;
		return r ;
}

public final  XQDataType  primary_expression(AST _t) throws RecognitionException {
		 XQDataType r ;

		AST primary_expression_AST_in = (AST)_t;
		AST i = null;
		AST il = null;
		AST sl = null;
		AST ic = null;
		AST dc = null;

					XQDataType x[];
					XQDataType a;
					r = null_data;


		try {	// for error handling
			{
			if (_t==null) _t=ASTNULL;
			switch ( _t.getType()) {
			case ID:
			{
					i = (AST)_t;
					match(_t,ID);
					_t = _t.getNextSibling();
					r = ipt.getVariable(i.getText());
					break;
			}
			case NListDeclaration:
			{
					AST __t4747 = _t;
					AST tmp25_AST_in = (AST)_t;
					match(_t,NListDeclaration);
					_t = _t.getFirstChild();
					il = (AST)_t;
					match(_t,ID);
					_t = _t.getNextSibling();
					a=expression(_t);
					_t = _retTree;
					_t = __t4747;
					_t = _t.getNextSibling();
					x = ipt.getArrayVariable(il.getText());
								r = x[XQInt.intValue(a)];

					break;
			}
			case STRING_LITERAL:
			{
					sl = (AST)_t;
					match(_t,STRING_LITERAL);
```

```
                    _t = _t.getNextSibling();
                    r = new XQString(sl.getText());
                    break;
                }
                case INT_CONST:
                {
                    ic = (AST)_t;
                    match(_t,INT_CONST);
                    _t = _t.getNextSibling();
                    r = ipt.getNumber(ic.getText());
                    break;
                }
                case DOUBLE_CONST:
                {
                    dc = (AST)_t;
                    match(_t,DOUBLE_CONST);
                    _t = _t.getNextSibling();
                    r = ipt.getNumber(dc.getText());
                    break;
                }
                case LPAREN:
                {
                    AST __t4748 = _t;
                    AST tmp26_AST_in = (AST)_t;
                    match(_t,LPAREN);
                    _t = _t.getFirstChild();
                    r=expression(_t);
                    _t = _retTree;
                    _t = __t4748;
                    _t = _t.getNextSibling();
                    break;
                }
                default:
                {
                    throw new NoViableAltException(_t);
                }
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
        return r ;
}

public final void unary_operator(AST _t) throws RecognitionException {

        AST unary_operator_AST_in = (AST)_t;

        try {      // for error handling
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case PLUS:
                {
                    AST tmp27_AST_in = (AST)_t;
                    match(_t,PLUS);
```

```
                    _t = _t.getNextSibling();
                    break;
                }
                case MINUS:
                {
                    AST tmp28_AST_in = (AST)_t;
                    match(_t,MINUS);
                    _t = _t.getNextSibling();
                    break;
                }
                case LNOT:
                {
                    AST tmp29_AST_in = (AST)_t;
                    match(_t,LNOT);
                    _t = _t.getNextSibling();
                    break;
                }
                default:
                {
                    throw new NoViableAltException(_t);
                }
                }
            }
            catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            _retTree = _t;
}

public final void declaration(AST _t) throws RecognitionException {

            AST declaration_AST_in = (AST)_t;
            AST t = null;
            AST idec = null;

                int size = 0;
                XQDataType a = null_data;


            try {      // for error handling
                AST __t4753 = _t;
                AST tmp30_AST_in = (AST)_t;
                match(_t,NDeclaration);
                _t = _t.getFirstChild();
                t = _t==ASTNULL ? null : (AST)_t;
                type_specifier(_t);
                _t = _retTree;
                idec = (AST)_t;
                match(_t,ID);
                _t = _t.getNextSibling();
                {
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case LPAREN:
                case ID:
                case STRING_LITERAL:
                case LOR:
```

```
case LAND:
case EQUAL:
case NOT_EQUAL:
case LT:
case GT:
case LTE:
case GTE:
case PLUS:
case MINUS:
case STAR:
case DIV:
case MOD:
case DOUBLE_CONST:
case INT_CONST:
case NUnary:
case NListDeclaration:
{
        a=expression(_t);
        _t = _retTree;
        size = XQInt.intValue(a);
        break;
}
case 3:
{
        break;
}
default:
{
        throw new NoViableAltException(_t);
}
}
}
_t = __t4753;
_t = _t.getNextSibling();

                System.out.println("Im hhere");
                if (t.getType() == INT) {
                        if (size == 0 ) {
                                XQInt x = new XQInt(0);
                                x.setName(idec.getText());
                                System.out.println("Im                      hhere--
>"+idec.getText()+"<---");

                                ipt.declareVariable(idec.getText(), x);
                                System.out.println("Im hhere");
                        }
                        else {
                                XQInt [] xs = new XQInt[size];
                                for (int i = 0; i < size; i++) {
                                        xs[i] = new XQInt(0);
                                        xs[i].setName(idec.getText());
                                }
                                ipt.declareArrayVariable(idec.getText(), xs);
                        }

                }
                else if (t.getType() == DOUBLE ) {
                        if (size == 0 ) {
                                XQDouble x = new XQDouble(0);
```

```
                                        x.setName(idec.getText());
                                        ipt.declareVariable( idec.getText(), x);
                                }
                                else {
                                        XQDouble [] xs = new XQDouble[size];
                                        for (int i = 0; i < size; i++) {
                                                xs[i] = new XQDouble(0);
                                                xs[i].setName(idec.getText());
                                        }
                                        ipt.declareArrayVariable(idec.getText(), xs);
                                }
                        }
                        else if (t.getType() == STRING)  {
                                if (size == 0 ) {
                                        XQString x = new XQString("");
                                        x.setName(idec.getText());
                                        ipt.declareVariable( idec.getText(), x);
                                }
                                else {
                                        XQString [] xs = new XQString[size];
                                        for (int i = 0; i < size; i++) {
                                                xs[i] = new XQString("");
                                                xs[i].setName(idec.getText());
                                        }
                                        ipt.declareArrayVariable(idec.getText(), xs);
                                }
                        }

        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
}

public final void type_specifier(AST _t) throws RecognitionException {

        AST type_specifier_AST_in = (AST)_t;

        try {      // for error handling
                {
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case INT:
                {
                        AST tmp31_AST_in = (AST)_t;
                        match(_t,INT);
                        _t = _t.getNextSibling();
                        break;
                }
                case DOUBLE:
                {
                        AST tmp32_AST_in = (AST)_t;
                        match(_t,DOUBLE);
                        _t = _t.getNextSibling();
                        break;
                }
```

```
                case STRING:
                {
                        AST tmp33_AST_in = (AST)_t;
                        match(_t,STRING);
                        _t = _t.getNextSibling();
                        break;
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
}

public final void break_statement(AST _t) throws RecognitionException {

        AST break_statement_AST_in = (AST)_t;

        try {      // for error handling
                AST __t4756 = _t;
                AST tmp34_AST_in = (AST)_t;
                match(_t,BREAK);
                _t = _t.getFirstChild();
                ipt.setBreak(null);
                _t = __t4756;
                _t = _t.getNextSibling();
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
}

public final void continue_statement(AST _t) throws RecognitionException {

        AST continue_statement_AST_in = (AST)_t;

        try {      // for error handling
                AST __t4758 = _t;
                AST tmp35_AST_in = (AST)_t;
                match(_t,CONTINUE);
                _t = _t.getFirstChild();
                ipt.setContinue( null );
                _t = __t4758;
                _t = _t.getNextSibling();
        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
```

```
		_retTree = _t;
}

public final void iteration_statement(AST _t) throws RecognitionException {

		AST iteration_statement_AST_in = (AST)_t;

		try {	// for error handling
			whileLoop(_t);
			_t = _retTree;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		_retTree = _t;
}

public final void whileLoop(AST _t) throws RecognitionException {

		AST whileLoop_AST_in = (AST)_t;
		AST s = null;

			XQDataType a, b;


		try {	// for error handling
			AST __t4761 = _t;
			AST tmp36_AST_in = (AST)_t;
			match(_t,WHILE);
			_t = _t.getFirstChild();
			a=expression(_t);
			_t = _retTree;
			s = _t==ASTNULL ? null : (AST)_t;
			statement_list(_t);
			_t = _retTree;
			_t = __t4761;
			_t = _t.getNextSibling();

					while ( ipt.canProceed() )
			{
				statement_list(s);
				ipt.whileNext(null);
			}
			ipt.whileEnd(null);

		}
		catch (RecognitionException ex) {
			reportError(ex);
			if (_t!=null) {_t = _t.getNextSibling();}
		}
		_retTree = _t;
}

public final void if_statement(AST _t) throws RecognitionException {

		AST if_statement_AST_in = (AST)_t;
		AST thenp = null;
```

```
AST elsep = null;

        XQDataType a;


try {   // for error handling
        AST __t4769 = _t;
        AST tmp37_AST_in = (AST)_t;
        match(_t,IF);
        _t = _t.getFirstChild();
        a=expression(_t);
        _t = _retTree;
        thenp = (AST)_t;
        if ( _t==null ) throw new MismatchedTokenException();
        _t = _t.getNextSibling();
        {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case LCURLY:
        case RCURLY:
        case PRINT:
        case LPAREN:
        case COMA:
        case RPAREN:
        case LOAD:
        case ID:
        case STRING_LITERAL:
        case FETCH:
        case SELECT:
        case WHERE:
        case USING:
        case IF:
        case ELSE:
        case BREAK:
        case SEMI:
        case CONTINUE:
        case WHILE:
        case INC_OP:
        case DEC_OP:
        case LBRACKET:
        case RBRACKET:
        case ASSIGN:
        case ADD_ASSIGN:
        case SUB_ASSIGN:
        case DIV_ASSIGN:
        case STAR_ASSIGN:
        case MOD_ASSIGN:
        case LOR:
        case LAND:
        case EQUAL:
        case NOT_EQUAL:
        case LT:
        case GT:
        case LTE:
        case GTE:
        case PLUS:
        case MINUS:
        case STAR:
```

```
case DIV:
case MOD:
case LNOT:
case DOUBLE_CONST:
case INT_CONST:
case INT:
case DOUBLE:
case STRING:
case CURSOR:
case NRootAST:
case NDeclarator:
case NDeclaration:
case NUnionDeclaration:
case NUnary:
case NFetch_statement:
case NSelect_statement:
case NEmptyStatement:
case NInitializer:
case NExterntype:
case NCodeBlock:
case NForCon:
case NForBlock:
case NForDeclaration:
case NDeclareList:
case NListDeclaration:
case NAssignList:
case NStartCondition:
case NStatement:
case FOR:
case RETURN:
case FROM:
case WS:
case NL:
case COMMENT:
case COLON:
case COMMA:
case DOT:
case LSHIFT:
case Escape:
case DIGIT:
case NUMBER:
{
        elsep = (AST)_t;
        if ( _t==null ) throw new MismatchedTokenException();
        _t = _t.getNextSibling();
        break;
}
case 3:
{
        break;
}
default:
{
        throw new NoViableAltException(_t);
}
}
}
_t = __t4769;
```

```
                _t = _t.getNextSibling();

                if (!( a instanceof XQInt )) {
                a.error( "if: expression should be integer value" );
                                        return;
                }
                if ( XQInt.intValue(a) != 0 )
                statement_list( thenp );
                else if ( null != elsep )
                statement_list( elsep );

        }
        catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
        }
        _retTree = _t;
}

public final void compound_statement(AST _t) throws RecognitionException {

        AST compound_statement_AST_in = (AST)_t;

        try {      // for error handling
                AST __t4766 = _t;
                AST tmp38_AST_in = (AST)_t;
                match(_t,LCURLY);
                _t = _t.getFirstChild();
                {
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType()) {
                case IF:
                case BREAK:
                case CONTINUE:
                case WHILE:
                case INC_OP:
                case DEC_OP:
                case ASSIGN:
                case ADD_ASSIGN:
                case SUB_ASSIGN:
                case DIV_ASSIGN:
                case STAR_ASSIGN:
                case MOD_ASSIGN:
                {
                        statement_list(_t);
                        _t = _retTree;
                        break;
                }
                case 3:
                {
                        break;
                }
                default:
                {
                        throw new NoViableAltException(_t);
                }
                }
                }
```

```
                _t = __t4766;
                _t = _t.getNextSibling();
            }
            catch (RecognitionException ex) {
                reportError(ex);
                if (_t!=null) {_t = _t.getNextSibling();}
            }
            _retTree = _t;
    }


    public static final String[] _tokenNames = {
            "<0>",
            "EOF",
            "<2>",
            "NULL_TREE_LOOKAHEAD",
            "\"{\"",
            "\"}\"",
            "\"print\"",
            "\"(\"",
            "COMA",
            "\")\"",
            "\"load\"",
            "an identifier",
            "STRING_LITERAL",
            "\"fetch\"",
            "\"select\"",
            "WHERE",
            "\"using\"",
            "\"if\"",
            "\"else\"",
            "\"break\"",
            "\";\"",
            "\"continue\"",
            "\"while\"",
            "INC_OP",
            "DEC_OP",
            "\"[\"",
            "\"]\"",
            "ASSIGN",
            "ADD_ASSIGN",
            "SUB_ASSIGN",
            "DIV_ASSIGN",
            "STAR_ASSIGN",
            "MOD_ASSIGN",
            "LOR",
            "LAND",
            "EQUAL",
            "NOT_EQUAL",
            "LT",
            "GT",
            "LTE",
            "GTE",
            "PLUS",
            "MINUS",
            "STAR",
            "DIV",
            "MOD",
```

```
                "LNOT",
                "DOUBLE_CONST",
                "INT_CONST",
                "\"int\"",
                "\"double\"",
                "\"string\"",
                "\"cursor\"",
                "NRootAST",
                "NDeclarator",
                "NDeclaration",
                "NUnionDeclaration",
                "NUnary",
                "NFetch_statement",
                "NSelect_statement",
                "NEmptyStatement",
                "NInitializer",
                "NExterntype",
                "NCodeBlock",
                "NForCon",
                "NForBlock",
                "NForDeclaration",
                "NDeclareList",
                "NListDeclaration",
                "NAssignList",
                "NStartCondition",
                "NStatement",
                "\"for\"",
                "\"return\"",
                "\"from\"",
                "WS",
                "NL",
                "COMMENT",
                "':'",
                "COMMA",
                "DOT",
                "'<<'",
                "Escape",
                "DIGIT",
                "a number"
        };

        private static final long[] mk_tokenSet_0() {
                long[] data = { 8487829504L, 0L};
                return data;
        }
        public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
        }

// start of XQAntlrWalkerTokenTypes.java
// $ANTLR : "walker.g" -> "XQAntlrWalker.java"$

public interface XQAntlrWalkerTokenTypes {
        int EOF = 1;
        int NULL_TREE_LOOKAHEAD = 3;
        int LCURLY = 4;
        int RCURLY = 5;
        int PRINT = 6;
        int LPAREN = 7;
```

```
int COMA = 8;
int RPAREN = 9;
int LOAD = 10;
int ID = 11;
int STRING_LITERAL = 12;
int FETCH = 13;
int SELECT = 14;
int WHERE = 15;
int USING = 16;
int IF = 17;
int ELSE = 18;
int BREAK = 19;
int SEMI = 20;
int CONTINUE = 21;
int WHILE = 22;
int INC_OP = 23;
int DEC_OP = 24;
int LBRACKET = 25;
int RBRACKET = 26;
int ASSIGN = 27;
int ADD_ASSIGN = 28;
int SUB_ASSIGN = 29;
int DIV_ASSIGN = 30;
int STAR_ASSIGN = 31;
int MOD_ASSIGN = 32;
int LOR = 33;
int LAND = 34;
int EQUAL = 35;
int NOT_EQUAL = 36;
int LT = 37;
int GT = 38;
int LTE = 39;
int GTE = 40;
int PLUS = 41;
int MINUS = 42;
int STAR = 43;
int DIV = 44;
int MOD = 45;
int LNOT = 46;
int DOUBLE_CONST = 47;
int INT_CONST = 48;
int INT = 49;
int DOUBLE = 50;
int STRING = 51;
int CURSOR = 52;
int NRootAST = 53;
int NDeclarator = 54;
int NDeclaration = 55;
int NUnionDeclaration = 56;
int NUnary = 57;
int NFetch_statement = 58;
int NSelect_statement = 59;
int NEmptyStatement = 60;
int NInitializer = 61;
int NExterntype = 62;
int NCodeBlock = 63;
int NForCon = 64;
int NForBlock = 65;
```

```
		int NForDeclaration = 66;
		int NDeclareList = 67;
		int NListDeclaration = 68;
		int NAssignList = 69;
		int NStartCondition = 70;
		int NStatement = 71;
		int FOR = 72;
		int RETURN = 73;
		int FROM = 74;
		int WS = 75;
		int NL = 76;
		int COMMENT = 77;
		int COLON = 78;
		int COMMA = 79;
		int DOT = 80;
		int LSHIFT = 81;
		int Escape = 82;
		int DIGIT = 83;
		int NUMBER = 84;
}

// start of XQdataType.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 */

import java.io.PrintWriter;

public class XQDataType {
		String name; // used in hash table

		public XQDataType() {
			name = null;
		}

		public XQDataType(String name) {
			this.name = name;
		}

		public String typename() {
			return "unknown";
		}

		public XQDataType copy() {
			return new XQDataType();
		}

		public void setName(String name) {
			this.name = name;
		}
```

```
public XQDataType error(String msg) {
	throw new XQException(
		"illegal operation: "
			+ msg
			+ "( <"
			+ typename()
			+ "> "
			+ (name != null ? name : "<?>")
			+ " )");
}

public XQDataType error(XQDataType b, String msg) {
	if (null == b)
		return error(msg);
	throw new XQException(
		"illegal operation: "
			+ msg
			+ "( <"
			+ typename()
			+ "> "
			+ (name != null ? name : "<?>")
			+ " and "
			+ "<"
			+ typename()
			+ "> "
			+ (name != null ? name : "<?>")
			+ " )");
}

public void print(PrintWriter w) {
	if (name != null)
		w.print(name + " = ");
	w.println("<undefined>");
}

public void print() {
	print(new PrintWriter(System.out, true));
}

public void what(PrintWriter w) {
	w.print("<" + typename() + ">  ");
	print(w);
}

public void what() {
	what(new PrintWriter(System.out, true));
}

public XQDataType assign(XQDataType b) {
	return error(b, "=");
}

public XQDataType uminus() {
	return error("-");
}

public XQDataType plus(XQDataType b) {
```

```
		return error(b, "+");
}

public XQDataType add(XQDataType b) {
		return error(b, "+=");
}

public XQDataType minus(XQDataType b) {
		return error(b, "-");
}

public XQDataType sub(XQDataType b) {
		return error(b, "-=");
}

public XQDataType times(XQDataType b) {
		return error(b, "*");
}

public XQDataType mul(XQDataType b) {
		return error(b, "*=");
}

public XQDataType lfracts(XQDataType b) {
		return error(b, "/");
}

public XQDataType rfracts(XQDataType b) {
		return error(b, "/\'");
}

public XQDataType ldiv(XQDataType b) {
		return error(b, "/=");
}

public XQDataType rdiv(XQDataType b) {
		return error(b, "/\'=");
}

public XQDataType modulus(XQDataType b) {
		return error(b, "%");
}

public XQDataType rem(XQDataType b) {
		return error(b, "%=");
}

public XQDataType gt(XQDataType b) {
		return error(b, ">");
}

public XQDataType ge(XQDataType b) {
		return error(b, ">=");
}

public XQDataType lt(XQDataType b) {
		return error(b, "<");
}
```

```
	public XQDataType le(XQDataType b) {
		return error(b, "<=");
	}

	public XQDataType eq(XQDataType b) {
		return error(b, "==");
	}

	public XQDataType ne(XQDataType b) {
		return error(b, "!=");
	}

	public XQDataType and(XQDataType b) {
		return error(b, "&&");
	}

	public XQDataType or(XQDataType b) {
		return error(b, "||");
	}

	public XQDataType not() {
		return error("!");
	}
}

// start of XQDouble.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.io.PrintWriter;

class XQDouble extends XQDataType {
	double var;

	public XQDouble( double x ) {
		var = x;
	}

	public String typename() {
		return "double";
	}

	public XQDataType copy() {
		return new XQDouble( var );
	}
```

```
public static double doubleValue( XQDataType b ) {
        if ( b instanceof XQDouble )
                return ((XQDouble)b).var;
        if ( b instanceof XQInt )
                return (double) ((XQInt)b).var;
        b.error( "cast to double" );
        return 0;
}

public void print( PrintWriter w ) {
        if ( name != null )
                w.print( name + " = " );
        w.println( Double.toString( var ) );
}

public XQDataType uminus() {
        return new XQDouble( -var );
}

public XQDataType plus( XQDataType b ) {
        return new XQDouble( var + doubleValue(b) );
}

public XQDataType add( XQDataType b ) {
        var += doubleValue( b );
        return this;
}

public XQDataType minus( XQDataType b ) {
        return new XQDouble( var - doubleValue(b) );
}

public XQDataType sub( XQDataType b ) {
        var -= doubleValue( b );
        return this;
}

public XQDataType times( XQDataType b ) {
        return new XQDouble( var * doubleValue(b) );
}

public XQDataType mul( XQDataType b ) {
        var *= doubleValue( b );
        return this;
}

public XQDataType lfracts( XQDataType b ) {
        return new XQDouble( var / doubleValue(b) );
}

public XQDataType rfracts( XQDataType b ) {
        return lfracts( b );
}

public XQDataType ldiv( XQDataType b ) {
        var /= doubleValue(b);
        return this;
}
```

```
	public XQDataType rdiv( XQDataType b ) {
		return ldiv( b );
	}

	public XQDataType modulus( XQDataType b ) {
		return new XQDouble( var % doubleValue(b) );
	}


	public XQDataType rem( XQDataType b ) {
		var %= doubleValue( b );
		return this;
	}

	public XQDataType gt( XQDataType b ) {
		return new XQInt( var > doubleValue(b) ? 1 : 0);
	}

	public XQDataType ge( XQDataType b ) {
		return new XQInt( var >= doubleValue(b) ? 1 : 0);
	}

	public XQDataType lt( XQDataType b ) {
		return new XQInt( var < doubleValue(b) ? 1 : 0 );
	}

	public XQDataType le( XQDataType b ) {
		return new XQInt( var <= doubleValue(b) ? 1 : 0);
	}

	public XQDataType eq( XQDataType b ) {
		return new XQInt( var == doubleValue(b) ? 1 : 0 );
	}

	public XQDataType ne( XQDataType b ) {
		return new XQInt( var != doubleValue(b) ? 1 : 0);
	}
}

// start of XQException.java

/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

class XQException extends RuntimeException {
	XQException( String msg ) {
```

```
                System.err.println( "Error: " + msg );
        }
}

// start of XQFunction.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.io.PrintWriter;
import antlr.collections.AST;

class XQFunction extends XQDataType {
        // we need a reference to the AST for the function entry
        String[] args;
        AST body;           // body = null means an internal function.
        XQSymbolTable pst;   // the symbol table of static parent
        int id;             // for internal functions only

        public XQFunction( String name, String[] args,
                                                AST body, XQSymbolTable pst) {
                super( name );
                this.args = args;
                this.body = body;
                this.pst = pst;
        }

        public XQFunction( String name, int id ) {
                super( name );
                this.args = null;
                this.id = id;
                pst = null;
                body = null;
        }

        public final boolean isInternal() {
                return body == null;
        }

        public final int getInternalId() {
                return id;
        }

        public String typename() {
                return "function";
        }

        public XQDataType copy() {
```

```
                return new XQFunction( name, args, body, pst );
        }

        public void print( PrintWriter w ) {
                if ( body == null )
                {
                        w.println( name + " = <internal-function> #" + id );
                }
                else
                {
                        if ( name != null )
                                w.print( name + " = " );
                        w.print( "<function>(" );
                        for ( int i=0; ; i++ )
                        {
                                w.print( args[i] );
                                if ( i >= args.length - 1 )
                                        break;
                                w.print( "," );
                        }
                        w.println( ")" );
                }
        }

        public String[] getArgs() {
                return args;
        }

        public XQSymbolTable getParentSymbolTable() {
                return pst;
        }

        public AST getBody() {
                return body;
        }
}

// start of XQInt.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
import java.io.PrintWriter;

class XQInt extends XQDataType {
        int var;

        public XQInt( int x ) {
                var = x;
```

```
}

public String typename() {
	return "int";
}

public XQDataType copy() {
	return new XQInt( var );
}

public static int intValue( XQDataType b ) {
	if ( b instanceof XQDouble )
		return (int) ((XQDouble)b).var;
	if ( b instanceof XQInt )
		return ((XQInt)b).var;
	b.error( "cast to int" );
	return 0;
}

public void print( PrintWriter w ) {
	if ( name != null )
		w.print( name + " = " );
	w.println( Integer.toString( var ) );
}

public XQDataType uminus() {
	return new XQInt( -var );
}

public XQDataType plus( XQDataType b ) {
	if ( b instanceof XQInt )
		return new XQInt( var + intValue(b) );
	return new XQDouble( var + XQDouble.doubleValue(b) );
}

public XQDataType add( XQDataType b ) {
	var += intValue( b );
	return this;
}

public XQDataType minus( XQDataType b ) {
	if ( b instanceof XQInt )
		return new XQInt( var - intValue(b) );
	return new XQDouble( var - XQDouble.doubleValue(b) );
}

public XQDataType sub( XQDataType b ) {
	var -= intValue( b );
	return this;
}

public XQDataType times( XQDataType b ) {
	if ( b instanceof XQInt )
		return new XQInt( var * intValue(b) );
	return new XQDouble( var * XQDouble.doubleValue(b) );
}

public XQDataType mul( XQDataType b ) {
```

```
		var *= intValue( b );
		return this;
}

public XQDataType lfracts( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var / intValue(b) );
		return new XQDouble( var / XQDouble.doubleValue(b) );
}

public XQDataType rfracts( XQDataType b ) {
		return lfracts( b );
}

public XQDataType ldiv( XQDataType b ) {
		var /= intValue(b);
		return this;
}

public XQDataType rdiv( XQDataType b ) {
		return ldiv( b );
}

public XQDataType modulus( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var % intValue(b) );
		return new XQDouble( var % XQDouble.doubleValue(b) );
}


public XQDataType rem( XQDataType b ) {
		var %= intValue( b );
		return this;
}

public XQDataType gt( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var > intValue(b) ? 1 : 0);
		return b.lt( this );
}

public XQDataType ge( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var >= intValue(b) ? 1 : 0);
		return b.le( this );
}

public XQDataType lt( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var < intValue(b) ? 1 : 0);
		return b.gt( this );
}

public XQDataType le( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var <= intValue(b) ? 1 : 0);
		return b.ge( this );
}
```

```
	public XQDataType eq( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var == intValue(b) ? 1 : 0);
		return b.eq( this );
	}

	public XQDataType ne( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var != intValue(b) ? 1 : 0);
		return b.ne( this );
	}

	public XQDataType and( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var == 0 || intValue(b) == 0 ? 0 : 1);
		return error( b, "&&" );
	}

	public XQDataType or( XQDataType b ) {
		if ( b instanceof XQInt )
			return new XQInt( var != 0 || intValue(b) != 0 ? 1 : 0);
		return error( b, "||" );
	}

	public XQDataType not() {
		return new XQInt( var == 0 ? 1 : 0 );
	}

}


// start of XQInternalFuncton.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.util.Random;

class XQInternalFunction {

	static Random random = new Random();

	final static int f_print = 0;
	final static int f_what = 1;
	final static int f_load = 8;
	final static int f_random = 12;
	final static int f_min = 20;
```

```
final static int f_max = 21;
final static int f_round = 25;
final static int f_floor = 26;
final static int f_ceil = 27;
final static int f_int = 28;
final static int f_abs = 29;
final static int f_sqrt = 30;
final static int f_exp = 31;
final static int f_log = 32;
final static int f_pow = 33;
final static int f_sin = 34;
final static int f_cos = 35;
final static int f_tan = 36;
final static int f_asin = 37;
final static int f_acos = 38;
final static int f_atan = 39;

public static void register(XQSymbolTable st) {

	st.put("print", new XQFunction(null, f_print));
	st.put("what", new XQFunction(null, f_what));
	st.put("load", new XQFunction(null, f_load));
	st.put("random", new XQFunction(null, f_random));
	st.put("min", new XQFunction(null, f_min));
	st.put("max", new XQFunction(null, f_max));
	st.put("round", new XQFunction(null, f_round));
	st.put("floor", new XQFunction(null, f_floor));
	st.put("ceil", new XQFunction(null, f_ceil));
	st.put("int", new XQFunction(null, f_int));
	st.put("abs", new XQFunction(null, f_abs));
	st.put("sqrt", new XQFunction(null, f_sqrt));
	st.put("exp", new XQFunction(null, f_exp));
	st.put("log", new XQFunction(null, f_log));
	st.put("pow", new XQFunction(null, f_pow));
	st.put("sin", new XQFunction(null, f_sin));
	st.put("cos", new XQFunction(null, f_cos));
	st.put("tan", new XQFunction(null, f_tan));
	st.put("asin", new XQFunction(null, f_asin));
	st.put("acos", new XQFunction(null, f_acos));
	st.put("atan ", new XQFunction(null, f_atan));
	st.put("E", new XQDouble(Math.E));
	st.put("PI", new XQDouble(Math.PI));
}

private static boolean isIntList(XQDataType[] params) {
	for (int i = 0; i < params.length; i++)
		if (!(params[i] instanceof XQInt))
			return false;
	return true;
}

public static XQDataType run(
	XQSymbolTable st,
	int id,
	XQDataType[] params) {
	switch (id) {
		case f_print :
			for (int i = 0; i < params.length; i++)
```

```
                                        params[i].print();
                                return null;

                        case f_what :
                                if (params.length > 0) {
                                        for (int i = 0; i < params.length; i++)
                                                params[i].what();
                                } else
                                        st.what();
                                return null;
                        case f_load :

                                // Need work in here

                                return null;

                        case f_random :
                                if (0 == params.length)
                                        return new XQDouble(random.nextDouble());
                                if (1 == params.length) {
                                        if (params[0] instanceof XQDouble)
                                                return new XQDouble(
                                                        random.nextDouble()
                                                                *
XQDouble.doubleValue(params[0]));

                                        if (params[0] instanceof XQInt)
                                                return new XQInt(
                                                        random.nextInt(XQInt.intValue(params[0])));
                                }
                                throw new XQException("random() accepts 0-2  parameter(s)");

                        case f_min :
                                if (params.length > 0) {
                                        if (isIntList(params)) {
                                                int x = XQInt.intValue(params[0]);
                                                for (int i = 1; i < params.length; i++) {
                                                        int y = XQInt.intValue(params[i]);
                                                        if (y < x)
                                                                x = y;
                                                }
                                                return new XQInt(x);
                                        }

                                        double x = XQDouble.doubleValue(params[0]);
                                        for (int i = 1; i < params.length; i++) {
                                                double y = XQDouble.doubleValue(params[i]);
                                                if (y < x)
                                                        x = y;
                                        }
                                        return new XQDouble(x);
                                }
                                throw new XQException("min() accepts 1  parameter(s)");

                        case f_max :
                                if (params.length > 0) {
                                        if (isIntList(params)) {
                                                int x = XQInt.intValue(params[0]);
```

```
                        for (int i = 1; i < params.length; i++) {
                                int y = XQInt.intValue(params[i]);
                                if (y > x)
                                        x = y;
                        }
                        return new XQInt(x);
                }

                double x = XQDouble.doubleValue(params[0]);
                for (int i = 1; i < params.length; i++) {
                        double y = XQDouble.doubleValue(params[i]);
                        if (y > x)
                                x = y;
                }
                return new XQDouble(x);
        }
        throw new XQException("max() accepts 1  parameter(s)");

case f_pow :
        if (params.length != 2)
                throw new XQException("$1() accepts 2   parameter(s)");
        ;
        double pwr = XQDouble.doubleValue(params[1]);

        return new XQDouble(
                Math.pow(XQDouble.doubleValue(params[0]), pwr));

case f_round :
        if (params.length != 1)
                throw new XQException("round() accepts 1   parameter(s)");
        ;
        return new XQDouble(
                Math.round(XQDouble.doubleValue(params[0])));

case f_ceil :
        if (params.length != 1)
                throw new XQException("ceil() accepts 1   parameter(s)");
        ;
        return new XQDouble(Math.ceil(XQDouble.doubleValue(params[0])));

case f_floor :
        if (params.length != 1)
                throw new XQException("floor() accepts 1   parameter(s)");
        ;
        return new XQDouble(
                Math.floor(XQDouble.doubleValue(params[0])));

case f_abs :
        if (params.length != 1)
                throw new XQException("abs() accepts 1   parameter(s)");
        ;
        return new XQDouble(Math.abs(XQDouble.doubleValue(params[0])));

case f_sqrt :
        if (params.length != 1)
                throw new XQException("sqrt() accepts 1   parameter(s)");
        ;
        return new XQDouble(Math.sqrt(XQDouble.doubleValue(params[0])));
```

```
                        case f_exp :
                                if (params.length != 1)
                                        throw new XQException("exp() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.exp(XQDouble.doubleValue(params[0])));

                        case f_log :
                                if (params.length != 1)
                                        throw new XQException("log() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.log(XQDouble.doubleValue(params[0])));

                        case f_sin :
                                if (params.length != 1)
                                        throw new XQException("sin() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.sin(XQDouble.doubleValue(params[0])));

                        case f_cos :
                                if (params.length != 1)
                                        throw new XQException("cos() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.cos(XQDouble.doubleValue(params[0])));

                        case f_tan :
                                if (params.length != 1)
                                        throw new XQException("tan() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.tan(XQDouble.doubleValue(params[0])));

                        case f_asin :
                                if (params.length != 1)
                                        throw new XQException("asin() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.asin(XQDouble.doubleValue(params[0])));

                        case f_acos :
                                if (params.length != 1)
                                        throw new XQException("acos() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.acos(XQDouble.doubleValue(params[0])));

                        case f_atan :
                                if (params.length != 1)
                                        throw new XQException("atan() accepts 1   parameter(s)");
                                ;
                                return new XQDouble(Math.atan(XQDouble.doubleValue(params[0])));

                        default :
                                throw new XQException("unknown internal function");
                }
        }

}

// start of XQInterpreter.java
/*
```

```
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.util.*;
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;

class XQInterpreter {
	XQSymbolTable symt;

	final static int fc_none = 0;
	final static int fc_break = 1;
	final static int fc_continue = 2;
	final static int fc_return = 3;

	private int control = fc_none;
	private String label;

	private Random random = new Random();

	public XQInterpreter() {
		symt = new XQSymbolTable(null);
		registerInternal();
	}

	public static XQDataType getNumber(String s) {
		if (s.indexOf('.') >= 0 || s.indexOf('e') >= 0 || s.indexOf('E') >= 0)
			return new XQDouble(Double.parseDouble(s));
		return new XQInt(Integer.parseInt(s));
	}

	public boolean declareArrayVariable(String s, XQDataType[] d) {
		XQDataType[] x = symt.getArrayValue(s);
		if (x == null) {
			symt.setArrayValue(s, d);
		} else {
			x[0].error("Deplicate variable declaration:" + s);
			return false;
		}
		return true;
	}

	public XQDataType[] getArrayVariable(String s) {
		// default static scoping
```

```
		XQDataType[] x = symt.getArrayValue(s);
		if (x == null) {
			x[0].error("Undefined array:" + s);
		}
		return x;
	}

	public boolean setArrayVariable(String s, XQDataType[] d) {
		XQDataType[] x = symt.getArrayValue(s);
		if (x != null)
			symt.setArrayValue(s, d);
		else {
			x[0].error("Array not declared:" + s);
			return false;
		}
		return true;
	}

	public boolean declareVariable(String s, XQDataType d) {
		// default static scoping
		System.out.println("dV:b4 size--"+String.valueOf(symt.size()));
		XQDataType x = symt.getValue(s);
		System.out.println("dV:after size--"+String.valueOf(symt.size()));
		if (null == x) {
			System.out.println("before set value-->"+d.name);
			symt.setValue(s, d);
			System.out.println("after set value");
		}
		else {
			x.error("Undefined variable:" + s);
			return false;
		}
		return true;
	}

	public XQDataType getVariable(String s) {
		// default static scoping
		System.out.println("getV: s="+s);
		XQDataType x = symt.getValue(s);
		System.out.println("after get-->"+x.name+"--"+String.valueOf(symt.size()));
		x.print();
		if (null == x)
			return x.error("Undefined variable:" + s);

		return x;
	}

	public boolean setVariable(String s, XQDataType d) {
		XQDataType x = symt.getValue(s);
		if (x != null)
			symt.setValue(s, d);
		else {
			x.error("variable not declared:" + s);
			return false;
		}
		return true;
	}
```

```
	public XQDataType rvalue(XQDataType a) {
		if (null == a.name)
			return a;
		return a.copy();
	}

	public XQDataType deepRvalue(XQDataType a) {
		if (null == a.name)
			return a;

		return a.copy();
	}

	public XQDataType assign(XQDataType a, XQDataType b) {

		XQDataType x = symt.getValue(a.name);

		if (null != x) {
			XQDataType t = deepRvalue(b);
			t.setName(a.name);
			symt.setValue(t.name, t); // scope?
			System.out.println("after setvalue-->"+a.name);
			return t;
		}

		System.out.println("error condition");
		return a.error(b, "=");
	}

//missing
	public XQDataType funcInvoke(
//			XQAntlrWalker walker,
			XQDataType func,
			XQDataType[] params)
			throws antlr.RecognitionException {

			// func must be an existing function
			if (!(func instanceof XQFunction))
				return func.error("not a function");

			// Is this function an internal function?
			// Names of formal args are not necessary for internal functions.
			if (((XQFunction) func).isInternal())
				return execInternal(((XQFunction) func).getInternalId(), params);

			// otherwise check numbers of actual and formal arguments
			String[] args = ((XQFunction) func).getArgs();
			if (args.length != params.length)
				return func.error("unmatched length of parameters");

			// create a new symbol table
			symt = new XQSymbolTable(((XQFunction) func).getParentSymbolTable());

			// assign actual parameters to formal arguments
			for (int i = 0; i < args.length; i++) {
				XQDataType d = rvalue(params[i]);
				d.setName(args[i]);
				symt.setValue(args[i], d);
```

```
		}

		// call the function body
		//missing
		//			XQDataType r = walker.expr( ((XQFunction)func).getBody() );
		XQDataType r = new XQDataType("<NULL>");

		// no break or continue can go through the function
		if (control == fc_break || control == fc_continue)
			throw new XQException("nowhere to break or continue");

		// if a return was called
		if (control == fc_return)
			tryResetFlowControl(((XQFunction) func).name);

		// remove this symbol table and return
		symt = symt.parent();

		return r;
	}

	public void funcRegister(String name, String[] args, AST body) {
		symt.put(name, new XQFunction(name, args, body, symt));
	}

	public void setBreak(String label) {
		this.label = label;
		control = fc_break;
	}

	public void setContinue(String label) {
		this.label = label;
		control = fc_continue;
	}

	public void setReturn(String label) {
		this.label = label;
		control = fc_return;
	}

	public void tryResetFlowControl(String loop_label) {
		if (null == label || label.equals(loop_label))
			control = fc_none;
	}

	public void whileNext(String loop_label) {
		if (control == fc_continue)
			tryResetFlowControl(loop_label);
	}

	public void whileEnd(String loop_label) {
		if (control == fc_break)
			tryResetFlowControl(loop_label);
	}

	public boolean canProceed() {
		return control == fc_none;
	}
```

```
		public XQDataType execInternal(int id, XQDataType[] params) {
			return XQInternalFunction.run(symt, id, params);
		}

		public void registerInternal() {
			XQInternalFunction.register(symt);
		}
}

// start of XQLexer.java
// $ANTLR : "parser.g" -> "XQLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class XQLexer extends antlr.CharScanner implements XQAntlrTokenTypes, TokenStream
 {

   int nr_error = 0;
   public void reportError( String s ) {
       super.reportError( s );
       nr_error++;
    }
   public void reportError( RecognitionException e ) {
       super.reportError( e );
       nr_error++;
    }
public XQLexer(InputStream in) {
		this(new ByteBuffer(in));
}
public XQLexer(Reader in) {
		this(new CharBuffer(in));
}
public XQLexer(InputBuffer ib) {
		this(new LexerSharedInputState(ib));
}
```

```
public XQLexer(LexerSharedInputState state) {
	super(state);
	caseSensitiveLiterals = true;
	setCaseSensitive(true);
	literals = new Hashtable();
	literals.put(new ANTLRHashString("main", this), new Integer(4));
	literals.put(new ANTLRHashString("switch", this), new Integer(10));
	literals.put(new ANTLRHashString("case", this), new Integer(11));
	literals.put(new ANTLRHashString("for", this), new Integer(18));
	literals.put(new ANTLRHashString("false", this), new Integer(48));
	literals.put(new ANTLRHashString("true", this), new Integer(47));
	literals.put(new ANTLRHashString("endl", this), new Integer(78));
	literals.put(new ANTLRHashString("string", this), new Integer(53));
	literals.put(new ANTLRHashString("do", this), new Integer(17));
	literals.put(new ANTLRHashString("bool", this), new Integer(51));
	literals.put(new ANTLRHashString("char", this), new Integer(49));
	literals.put(new ANTLRHashString("while", this), new Integer(16));
	literals.put(new ANTLRHashString("if", this), new Integer(14));
	literals.put(new ANTLRHashString("double", this), new Integer(52));
	literals.put(new ANTLRHashString("int", this), new Integer(50));
	literals.put(new ANTLRHashString("default", this), new Integer(13));
	literals.put(new ANTLRHashString("cout", this), new Integer(77));
	literals.put(new ANTLRHashString("else", this), new Integer(15));
}

public Token nextToken() throws TokenStreamException {
	Token theRetToken=null;
tryAgain:
	for (;;) {
		Token _token = null;
		int _ttype = Token.INVALID_TYPE;
		resetText();
		try {   // for char stream error handling
			try {   // for lexical error handling
				switch ( LA(1)) {
				case '\t':  case ' ':
				{
					mWS(true);
					theRetToken=_returnToken;
					break;
				}
				case '\n':  case '\r':
				{
					mNL(true);
					theRetToken=_returnToken;
					break;
				}
				case ':':
				{
					mCOLON(true);
					theRetToken=_returnToken;
					break;
				}
				case ',':
				{
					mCOMMA(true);
					theRetToken=_returnToken;
					break;
```

```
}
case ';':
{
		mSEMI(true);
		theRetToken=_returnToken;
		break;
}
case '.':
{
		mDOT(true);
		theRetToken=_returnToken;
		break;
}
case '(':
{
		mLPAREN(true);
		theRetToken=_returnToken;
		break;
}
case ')':
{
		mRPAREN(true);
		theRetToken=_returnToken;
		break;
}
case '[':
{
		mLBRACKET(true);
		theRetToken=_returnToken;
		break;
}
case ']':
{
		mRBRACKET(true);
		theRetToken=_returnToken;
		break;
}
case '{':
{
		mLCURLY(true);
		theRetToken=_returnToken;
		break;
}
case '}':
{
		mRCURLY(true);
		theRetToken=_returnToken;
		break;
}
case '&':
{
		mLAND(true);
		theRetToken=_returnToken;
		break;
}
case '|':
{
		mLOR(true);
```

```
            theRetToken=_returnToken;
            break;
}
case '"':
{
            mSTRING_LITERAL(true);
            theRetToken=_returnToken;
            break;
}
case '0': case '1': case '2': case '3':
case '4': case '5': case '6': case '7':
case '8': case '9':
{
            mNUMBER(true);
            theRetToken=_returnToken;
            break;
}
case 'A': case 'B': case 'C': case 'D':
case 'E': case 'F': case 'G': case 'H':
case 'I': case 'J': case 'K': case 'L':
case 'M': case 'N': case 'O': case 'P':
case 'Q': case 'R': case 'S': case 'T':
case 'U': case 'V': case 'W': case 'X':
case 'Y': case 'Z': case '_': case 'a':
case 'b': case 'c': case 'd': case 'e':
case 'f': case 'g': case 'h': case 'i':
case 'j': case 'k': case 'l': case 'm':
case 'n': case 'o': case 'p': case 'q':
case 'r': case 's': case 't': case 'u':
case 'v': case 'w': case 'x': case 'y':
case 'z':
{
            mID(true);
            theRetToken=_returnToken;
            break;
}
default:
            if ((LA(1)=='/') && (LA(2)=='*')) {
                        mCOMMENT(true);
                        theRetToken=_returnToken;
            }
            else if ((LA(1)=='=') && (LA(2)=='=')) {
                        mEQUAL(true);
                        theRetToken=_returnToken;
            }
            else if ((LA(1)=='!') && (LA(2)=='=')) {
                        mNOT_EQUAL(true);
                        theRetToken=_returnToken;
            }
            else if ((LA(1)=='<') && (LA(2)=='=')) {
                        mLTE(true);
                        theRetToken=_returnToken;
            }
            else if ((LA(1)=='>') && (LA(2)=='=')) {
                        mGTE(true);
                        theRetToken=_returnToken;
            }
            else if ((LA(1)=='/') && (LA(2)=='=')) {
```

```
            mDIV_ASSIGN(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='*') && (LA(2)=='=')) {
            mSTAR_ASSIGN(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='%') && (LA(2)=='=')) {
            mMOD_ASSIGN(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (LA(2)=='=')) {
            mADD_ASSIGN(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (LA(2)=='=')) {
            mSUB_ASSIGN(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (LA(2)=='<')) {
            mLSHIFT(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='=') && (true)) {
            mASSIGN(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='<') && (true)) {
            mLT_OP(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='>') && (true)) {
            mGT(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='%') && (true)) {
            mMOD(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='/') && (true)) {
            mDIV(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (true)) {
            mPLUS(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (true)) {
            mMINUS(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='*') && (true)) {
            mSTAR(true);
            theRetToken=_returnToken;
}
else if ((LA(1)=='!') && (true)) {
            mLNOT(true);
            theRetToken=_returnToken;
```

```
                                        }
                                else {
                                        if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken = makeToken(Token.EOF_TYPE);}
                                else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                                }
                                }
                                if ( _returnToken==null ) continue tryAgain; // found SKIP token
                                _ttype = _returnToken.getType();
                                _ttype = testLiteralsTable(_ttype);
                                _returnToken.setType(_ttype);
                                return _returnToken;
                        }
                        catch (RecognitionException e) {
                                throw new TokenStreamRecognitionException(e);
                        }
                }
                catch (CharStreamException cse) {
                        if ( cse instanceof CharStreamIOException ) {
                                throw new TokenStreamIOException(((CharStreamIOException)cse).io);
                        }
                        else {
                                throw new TokenStreamException(cse.getMessage());
                        }
                }
        }
}

        public final void mWS(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = WS;
                int _saveIndex;

                {
                int _cnt1508=0;
                _loop1508:
                do {
                        switch ( LA(1)) {
                        case ' ':
                        {
                                match(' ');
                                break;
                        }
                        case '\t':
                        {
                                match('\t');
                                break;
                        }
                        default:
                        {
                                if ( _cnt1508>=1 ) { break _loop1508; } else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                        }
                        }
                        _cnt1508++;
```

```
		} while (true);
		}
		if ( inputState.guessing==0 ) {
			_ttype = Token.SKIP;
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mNL(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NL;
		int _saveIndex;

		{
		boolean synPredMatched1512 = false;
		if (((LA(1)=='\r') && (LA(2)=='\n'))) {
			int _m1512 = mark();
			synPredMatched1512 = true;
			inputState.guessing++;
			try {
				{
				match('\r');
				match('\n');
				}
			}
			catch (RecognitionException pe) {
				synPredMatched1512 = false;
			}
			rewind(_m1512);
			inputState.guessing--;
		}
		if ( synPredMatched1512 ) {
			match('\r');
			match('\n');
		}
		else if ((LA(1)=='\n')) {
			match('\n');
		}
		else if ((LA(1)=='\r') && (true)) {
			match('\r');
		}
		else {
			throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
		}

		}
		if ( inputState.guessing==0 ) {
			_ttype = Token.SKIP; newline();
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
```

```
		}
		_returnToken = _token;
	}

	public final void mCOMMENT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COMMENT;
		int _saveIndex;

		match("/*");
		{
		_loop1517:
		do {
			// nongreedy exit test
			if ((LA(1)=='*') && (LA(2)=='/')) break _loop1517;
			if ((_tokenSet_0.member(LA(1))) && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
				{
				match(_tokenSet_0);
				}
			}
			else if ((LA(1)=='\n'||LA(1)=='\r')) {
				{
				mNL(false);
				}
			}
			else {
				break _loop1517;
			}

		} while (true);
		}
		match("*/");
		if ( inputState.guessing==0 ) {
			_ttype = Token.SKIP;
		}
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = ASSIGN;
		int _saveIndex;

		match('=');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	public final void mCOLON(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COLON;
		int _saveIndex;

		match(':');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mCOMMA(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COMMA;
		int _saveIndex;

		match(',');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSEMI(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SEMI;
		int _saveIndex;

		match(';');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mDOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DOT;
		int _saveIndex;

		match('.');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLPAREN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
```

```
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LPAREN;
		int _saveIndex;

		match('(');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRPAREN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RPAREN;
		int _saveIndex;

		match(')');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLBRACKET(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LBRACKET;
		int _saveIndex;

		match('[');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRBRACKET(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RBRACKET;
		int _saveIndex;

		match(']');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLCURLY(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LCURLY;
```

```
		int _saveIndex;

		match('{');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRCURLY(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RCURLY;
		int _saveIndex;

		match('}');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mEQUAL(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = EQUAL;
		int _saveIndex;

		match("==");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mNOT_EQUAL(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NOT_EQUAL;
		int _saveIndex;

		match("!=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLTE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LTE;
		int _saveIndex;
```

```
		match("<=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLT_OP(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LT_OP;
		int _saveIndex;

		match("<");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mGTE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = GTE;
		int _saveIndex;

		match(">=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mGT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = GT;
		int _saveIndex;

		match(">");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mMOD(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = MOD;
		int _saveIndex;

		match('%');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
```

```
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mDIV(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIV;
            int _saveIndex;

            match('/');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mPLUS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = PLUS;
            int _saveIndex;

            match('+');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mDIV_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = DIV_ASSIGN;
            int _saveIndex;

            match("/=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
            }
            _returnToken = _token;
        }

        public final void mSTAR_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = STAR_ASSIGN;
            int _saveIndex;

            match("*=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
```

```
		}
		_returnToken = _token;
	}

	public final void mMOD_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = MOD_ASSIGN;
		int _saveIndex;

		match("%=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mADD_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = ADD_ASSIGN;
		int _saveIndex;

		match("+=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mMINUS(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = MINUS;
		int _saveIndex;

		match('-');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSUB_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = SUB_ASSIGN;
		int _saveIndex;

		match("-=");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
```

```
	}

	public final void mSTAR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STAR;
		int _saveIndex;

		match('*');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLSHIFT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LSHIFT;
		int _saveIndex;

		match("<<");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLAND(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LAND;
		int _saveIndex;

		match("&&");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mLNOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LNOT;
		int _saveIndex;

		match('!');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}
```

```
	public final void mLOR(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = LOR;
		int _saveIndex;

		match("||");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mSTRING_LITERAL(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = STRING_LITERAL;
		int _saveIndex;

		match("");
		{
		_loop1551:
		do {
			if ((LA(1)=='\\')) {
				mEscape(false);
			}
			else if ((_tokenSet_1.member(LA(1)))) {
				matchNot("");
			}
			else {
				break _loop1551;
			}

		} while (true);
		}
		match("");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mEscape(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = Escape;
		int _saveIndex;

		match('\\');
		{
		switch ( LA(1)) {
		case 'a':
		{
			match('a');
			break;
		}
```

```
case 'b':
{
	match('b');
	if ( inputState.guessing==0 ) {
		text.setLength(_begin); text.append("\b");
	}
	break;
}
case 'f':
{
	match('f');
	if ( inputState.guessing==0 ) {
		text.setLength(_begin); text.append("\f");
	}
	break;
}
case 'n':
{
	match('n');
	if ( inputState.guessing==0 ) {
		text.setLength(_begin); text.append("\n");
	}
	break;
}
case 'r':
{
	match('r');
	if ( inputState.guessing==0 ) {
		text.setLength(_begin); text.append("\r");
	}
	break;
}
case 't':
{
	match('t');
	if ( inputState.guessing==0 ) {
		text.setLength(_begin); text.append("\t");
	}
	break;
}
case 'v':
{
	match('v');
	break;
}
case '"':
{
	match('"');
	if ( inputState.guessing==0 ) {
		text.setLength(_begin); text.append("\"");
	}
	break;
}
case '\'':
{
	match('\'');
	if ( inputState.guessing==0 ) {
		text.setLength(_begin); text.append("\'");
```

```
                    }
                    break;
                }
                case '\\':
                {
                    match('\\');
                    if ( inputState.guessing==0 ) {
                        text.setLength(_begin); text.append("\\");
                    }
                    break;
                }
                case '?':
                {
                    match('?');
                    break;
                }
                case '0':  case '1':  case '2':  case '3':
                {
                    {
                    matchRange('0','3');
                    }
                    {
                    if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
                        mDIGIT(false);
                        {
                        if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <=
'\u00ff'))) {
                            mDIGIT(false);
                        }
                        else if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && (true)) {
                        }
                        else {
                            throw        new        NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                        }

                        }
                    }
                    else if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && (true)) {
                    }
                    else {
                        throw   new   NoViableAltForCharException((char)LA(1),   getFilename(),
getLine(), getColumn());
                    }

                    }
                    break;
                }
                case '4':  case '5':  case '6':  case '7':
                {
                    {
                    matchRange('4','7');
                    }
                    {
                    if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
                        mDIGIT(false);
                    }
                    else if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && (true)) {
```

```
                        }
                        else {
                                throw new NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());
                        }

                        }
                        break;
                }
                case 'x':
                {
                        match('x');
                        {
                        int _cnt1560=0;
                        _loop1560:
                        do {
                                if (((LA(1) >= '0' && LA(1) <= '9')) && ((LA(2) >= '\u0003' && LA(2) <=
'\u00ff'))) {
                                        mDIGIT(false);
                                }
                                else if (((LA(1) >= 'a' && LA(1) <= 'f')) && ((LA(2) >= '\u0003' && LA(2)
<= '\u00ff'))) {
                                        matchRange('a','f');
                                }
                                else if (((LA(1) >= 'A' && LA(1) <= 'F')) && ((LA(2) >= '\u0003' && LA(2)
<= '\u00ff'))) {
                                        matchRange('A','F');
                                }
                                else {
                                        if ( _cnt1560>=1 ) { break _loop1560; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                                }

                                _cnt1560++;
                        } while (true);
                        }
                        break;
                }
                default:
                {
                        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
                }
                }
                }
                if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                        _token = makeToken(_ttype);
                        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
                }
                _returnToken = _token;
        }

        protected final void mDIGIT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
                int _ttype; Token _token=null; int _begin=text.length();
                _ttype = DIGIT;
                int _saveIndex;
```

```
		matchRange('0','9');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mDOUBLE_CONST(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DOUBLE_CONST;
		int _saveIndex;

		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	protected final void mINT_CONST(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = INT_CONST;
		int _saveIndex;

		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mNUMBER(boolean _createToken) throws RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = NUMBER;
		int _saveIndex;

		{
		boolean synPredMatched1569 = false;
		if ((((LA(1) >= '0' && LA(1) <= '9')) && (_tokenSet_2.member(LA(2))))) {
			int _m1569 = mark();
			synPredMatched1569 = true;
			inputState.guessing++;
			try {
				{
				{
				int _cnt1568=0;
				_loop1568:
				do {
					if (((LA(1) >= '0' && LA(1) <= '9'))) {
						mDIGIT(false);
					}
					else {
						if ( _cnt1568>=1 ) { break _loop1568; } else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
```

```
                                        }

                                        _cnt1568++;
                                } while (true);
                                }
                                mDOT(false);
                                }
                        }
                        catch (RecognitionException pe) {
                                synPredMatched1569 = false;
                        }
                        rewind(_m1569);
                        inputState.guessing--;
                }
                if ( synPredMatched1569 ) {
                        {
                        {
                        int _cnt1572=0;
                        _loop1572:
                        do {
                                if (((LA(1) >= '0' && LA(1) <= '9'))) {
                                        mDIGIT(false);
                                }
                                else {
                                        if ( _cnt1572>=1 ) { break _loop1572; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                                }

                                _cnt1572++;
                        } while (true);
                        }
                        mDOT(false);
                        {
                        _loop1574:
                        do {
                                if (((LA(1) >= '0' && LA(1) <= '9'))) {
                                        mDIGIT(false);
                                }
                                else {
                                        break _loop1574;
                                }

                        } while (true);
                        }
                        }
                        if ( inputState.guessing==0 ) {
                                _ttype = DOUBLE_CONST;
                        }
                }
                else if (((LA(1) >= '0' && LA(1) <= '9')) && (true)) {
                        {
                        int _cnt1576=0;
                        _loop1576:
                        do {
                                if (((LA(1) >= '0' && LA(1) <= '9'))) {
                                        mDIGIT(false);
                                }
                                else {
```

```
                                        if ( _cnt1576>=1 ) { break _loop1576; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                                    }

                                    _cnt1576++;
                                } while (true);
                                }
                                if ( inputState.guessing==0 ) {
                                    _ttype = INT_CONST;
                                }
                            }
                            else {
                                throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
                            }

                            }
                            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                                _token = makeToken(_ttype);
                                _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
                            }
                            _returnToken = _token;
                        }

                        public final void mID(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
                            int _ttype; Token _token=null; int _begin=text.length();
                            _ttype = ID;
                            int _saveIndex;

                            {
                            switch ( LA(1)) {
                            case 'a': case 'b': case 'c': case 'd':
                            case 'e': case 'f': case 'g': case 'h':
                            case 'i': case 'j': case 'k': case 'l':
                            case 'm': case 'n': case 'o': case 'p':
                            case 'q': case 'r': case 's': case 't':
                            case 'u': case 'v': case 'w': case 'x':
                            case 'y': case 'z':
                            {
                                matchRange('a','z');
                                break;
                            }
                            case 'A': case 'B': case 'C': case 'D':
                            case 'E': case 'F': case 'G': case 'H':
                            case 'I': case 'J': case 'K': case 'L':
                            case 'M': case 'N': case 'O': case 'P':
                            case 'Q': case 'R': case 'S': case 'T':
                            case 'U': case 'V': case 'W': case 'X':
                            case 'Y': case 'Z':
                            {
                                matchRange('A','Z');
                                break;
                            }
                            case '_':
                            {
                                match('_');
                                break;
```

```
		}
		default:
		{
			throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
		}
		}
		}
		{
		_loop1580:
		do {
			switch ( LA(1)) {
			case 'a': case 'b': case 'c': case 'd':
			case 'e': case 'f': case 'g': case 'h':
			case 'i': case 'j': case 'k': case 'l':
			case 'm': case 'n': case 'o': case 'p':
			case 'q': case 'r': case 's': case 't':
			case 'u': case 'v': case 'w': case 'x':
			case 'y': case 'z':
			{
				matchRange('a','z');
				break;
			}
			case 'A': case 'B': case 'C': case 'D':
			case 'E': case 'F': case 'G': case 'H':
			case 'I': case 'J': case 'K': case 'L':
			case 'M': case 'N': case 'O': case 'P':
			case 'Q': case 'R': case 'S': case 'T':
			case 'U': case 'V': case 'W': case 'X':
			case 'Y': case 'Z':
			{
				matchRange('A','Z');
				break;
			}
			case '_':
			{
				match('_');
				break;
			}
			case '0': case '1': case '2': case '3':
			case '4': case '5': case '6': case '7':
			case '8': case '9':
			{
				matchRange('0','9');
				break;
			}
			default:
			{
				break _loop1580;
			}
			}
		} while (true);
		}
		_ttype = testLiteralsTable(_ttype);
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
		}
```

```
		_returnToken = _token;
	}


	private static final long[] mk_tokenSet_0() {
		long[] data = new long[8];
		data[0]=-9224L;
		for (int i = 1; i<=3; i++) { data[i]=-1L; }
		return data;
	}
	public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
	private static final long[] mk_tokenSet_1() {
		long[] data = new long[8];
		data[0]=-17179869192L;
		data[1]=-268435457L;
		for (int i = 2; i<=3; i++) { data[i]=-1L; }
		return data;
	}
	public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
	private static final long[] mk_tokenSet_2() {
		long[] data = { 288019269919178752L, 0L, 0L, 0L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());

	}


// start of XQMain.java

/*
 * Created on Dec 9, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;

public class XQMain {

	public static void execFile(String filename) {
		try {
			InputStream input =
				(null != filename)
					? (InputStream) new FileInputStream(filename)
```

```
                                        : (InputStream) System.in;

                        XQAntlrLexer lexer = new XQAntlrLexer(input);

                        XQAntlrParser parser = new XQAntlrParser(lexer);

                        // Parse the input program
                        parser.start_condition();

                        if (lexer.nr_error > 0 || parser.nr_error > 0) {
                                System.err.println("Parsing errors found. Stop.");
                                return;
                        }

                        CommonAST tree = (CommonAST) parser.getAST();

                        XQAntlrWalker walker = new XQAntlrWalker();
                        // Traverse the tree created by the parser

                        walker.start_condition(tree);

                } catch (IOException e) {
                        System.err.println("Error: I/O: " + e);
                } catch (RecognitionException e) {
                        System.err.println("Error: Recognition: " + e);
                } catch (TokenStreamException e) {
                        System.err.println("Error: Token stream: " + e);
                } catch (Exception e) {
                        System.err.println("Error: " + e);
                }
        }

        public static void main(String[] args) {

                if (args.length >= 1)
                        execFile(args[args.length - 1]);
                System.exit(0);
        }
}

// start of XQParser.java
// $ANTLR : "parser.g" -> "XQParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
```

```
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class XQParser extends antlr.LLkParser       implements XQAntlrTokenTypes
 {

   int nr_error = 0;
   public void reportError( String s ) {
       super.reportError( s );
       nr_error++;
    }
   public void reportError( RecognitionException e ) {
       super.reportError( e );
       nr_error++;
    }

protected XQParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public XQParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected XQParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public XQParser(TokenStream lexer) {
  this(lexer,2);
}

public XQParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

        public final void start_condition() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST start_condition_AST = null;

                try {      // for error handling
                          match(LITERAL_main);
                          match(LPAREN);
                          match(RPAREN);
                          match(LCURLY);
                          declaration_list();
```

```
                    astFactory.addASTChild(currentAST, returnAST);
                    statement_list();
                    astFactory.addASTChild(currentAST, returnAST);
                    match(RCURLY);
                    match(Token.EOF_TYPE);
                    start_condition_AST = (AST)currentAST.root;
                    start_condition_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NStartCondition,"NStartCondition")).add(start_condition_AST));
                    currentAST.root = start_condition_AST;
                    currentAST.child = start_condition_AST!=null
&&start_condition_AST.getFirstChild()!=null ?
                            start_condition_AST.getFirstChild() : start_condition_AST;
                    currentAST.advanceChildToEnd();
                    start_condition_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_0);
            }
            returnAST = start_condition_AST;
    }

    public final void declaration_list() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST declaration_list_AST = null;

            try {      // for error handling
                    {
                    int _cnt1491=0;
                    _loop1491:
                    do {
                            if (((LA(1) >= CHAR && LA(1) <= STRING))) {
                                    declaration();
                                    astFactory.addASTChild(currentAST, returnAST);
                            }
                            else {
                                    if ( _cnt1491>=1 ) { break _loop1491; } else {throw new
NoViableAltException(LT(1), getFilename());}
                            }

                            _cnt1491++;
                    } while (true);
                    }
                    declaration_list_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_1);
            }
            returnAST = declaration_list_AST;
    }

    public final void statement_list() throws RecognitionException, TokenStreamException {
```

```
		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST statement_list_AST = null;

		try {	// for error handling
			{
			int _cnt1423=0;
			_loop1423:
			do {
				if ((_tokenSet_1.member(LA(1)))) {
					statement();
					astFactory.addASTChild(currentAST, returnAST);
				}
				else {
					if ( _cnt1423>=1 ) { break _loop1423; } else {throw new
NoViableAltException(LT(1), getFilename());}
				}

				_cnt1423++;
			} while (true);
			}
			statement_list_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_2);
		}
		returnAST = statement_list_AST;
	}

	public final void statement() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST statement_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case SEMI:
			case SWITCH:
			case IF:
			{
				selection_statement();
				astFactory.addASTChild(currentAST, returnAST);
				statement_AST = (AST)currentAST.root;
				break;
			}
			case WHILE:
			case DO:
			case FOR:
			{
				iteration_statement();
				astFactory.addASTChild(currentAST, returnAST);
				statement_AST = (AST)currentAST.root;
				break;
			}
			default:
```

```
                        if ((LA(1)==ID) && (_tokenSet_3.member(LA(2)))) {
                                assign_statement();
                                astFactory.addASTChild(currentAST, returnAST);
                                statement_AST = (AST)currentAST.root;
                        }
                        else if ((LA(1)==ID) && (LA(2)==DOT)) {
                                call_statement();
                                astFactory.addASTChild(currentAST, returnAST);
                                statement_AST = (AST)currentAST.root;
                        }
                else {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_4);
        }
        returnAST = statement_AST;
}

public final void assign_statement() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST assign_statement_AST = null;

        try {      // for error handling
                assign_expression();
                astFactory.addASTChild(currentAST, returnAST);
                match(SEMI);
                assign_statement_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_4);
        }
        returnAST = assign_statement_AST;
}

public final void call_statement() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST call_statement_AST = null;

        try {      // for error handling
                call_expression();
                astFactory.addASTChild(currentAST, returnAST);
                AST tmp8_AST = null;
                tmp8_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp8_AST);
                match(SEMI);
                call_statement_AST = (AST)currentAST.root;
        }
```

```
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_4);
		}
		returnAST = call_statement_AST;
}

public final void selection_statement() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST selection_statement_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
			case SEMI:
			{
				match(SEMI);
				selection_statement_AST = (AST)currentAST.root;
				break;
			}
			case SWITCH:
			{
				switchBlock();
				astFactory.addASTChild(currentAST, returnAST);
				selection_statement_AST = (AST)currentAST.root;
				break;
			}
			case IF:
			{
				ifThenBlock();
				astFactory.addASTChild(currentAST, returnAST);
				selection_statement_AST = (AST)currentAST.root;
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1), getFilename());
			}
			}
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_4);
		}
		returnAST = selection_statement_AST;
}

public final void iteration_statement() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST iteration_statement_AST = null;

		try {	// for error handling
			switch ( LA(1)) {
```

```
                    case WHILE:
                    {
                            AST tmp10_AST = null;
                            tmp10_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp10_AST);
                            match(WHILE);
                            match(LPAREN);
                            condition();
                            astFactory.addASTChild(currentAST, returnAST);
                            match(RPAREN);
                            codeBlock();
                            astFactory.addASTChild(currentAST, returnAST);
                            iteration_statement_AST = (AST)currentAST.root;
                            break;
                    }
                    case DO:
                    {
                            AST tmp13_AST = null;
                            tmp13_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp13_AST);
                            match(DO);
                            codeBlock();
                            astFactory.addASTChild(currentAST, returnAST);
                            match(WHILE);
                            match(LPAREN);
                            condition();
                            astFactory.addASTChild(currentAST, returnAST);
                            match(RPAREN);
                            match(SEMI);
                            iteration_statement_AST = (AST)currentAST.root;
                            break;
                    }
                    case FOR:
                    {
                            for_loop();
                            astFactory.addASTChild(currentAST, returnAST);
                            iteration_statement_AST = (AST)currentAST.root;
                            break;
                    }
                    default:
                    {
                            throw new NoViableAltException(LT(1), getFilename());
                    }
                    }
            }
            catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_4);
            }
            returnAST = iteration_statement_AST;
}

public final void switchBlock() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST switchBlock_AST = null;
```

```
		try {	// for error handling
			match(SWITCH);
			match(LPAREN);
			expression();
			astFactory.addASTChild(currentAST, returnAST);
			match(RPAREN);
			match(LCURLY);
			{
			_loop1427:
			do {
				if ((LA(1)==CASE)) {
					case_statement();
					astFactory.addASTChild(currentAST, returnAST);
				}
				else {
					break _loop1427;
				}

			} while (true);
			}
			{
			switch ( LA(1)) {
			case DEFAULT:
			{
				default_statement();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			case RCURLY:
			{
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1), getFilename());
			}
			}
			}
			match(RCURLY);
			switchBlock_AST = (AST)currentAST.root;
			switchBlock_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NSwitchBlock,"NSwitchBlock")).add(switchBlock_AST));
			currentAST.root = switchBlock_AST;
			currentAST.child = switchBlock_AST!=null
&&switchBlock_AST.getFirstChild()!=null ?
				switchBlock_AST.getFirstChild() : switchBlock_AST;
			currentAST.advanceChildToEnd();
			switchBlock_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_4);
		}
		returnAST = switchBlock_AST;
	}
```

```
	public final void ifThenBlock() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST ifThenBlock_AST = null;

		try {	// for error handling
			ifBlock();
			astFactory.addASTChild(currentAST, returnAST);
			{
			if ((LA(1)==ELSE) && (_tokenSet_1.member(LA(2)))) {
				elseBlock();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else if ((_tokenSet_4.member(LA(1))) && (_tokenSet_5.member(LA(2)))) {
			}
			else {
				throw new NoViableAltException(LT(1), getFilename());
			}

			}
			ifThenBlock_AST = (AST)currentAST.root;
			ifThenBlock_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NIfThenBlock,"NIfThenBlock")).add(ifThenBlock_AST));
			currentAST.root = ifThenBlock_AST;
			currentAST.child = ifThenBlock_AST!=null
&&ifThenBlock_AST.getFirstChild()!=null ?
				ifThenBlock_AST.getFirstChild() : ifThenBlock_AST;
			currentAST.advanceChildToEnd();
			ifThenBlock_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_4);
		}
		returnAST = ifThenBlock_AST;
	}

	public final void expression() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST expression_AST = null;

		try {	// for error handling
			logical_and_expression();
			astFactory.addASTChild(currentAST, returnAST);
			{
			_loop1463:
			do {
				if ((LA(1)==LOR)) {
					AST tmp23_AST = null;
					tmp23_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp23_AST);
					match(LOR);
					logical_and_expression();
					astFactory.addASTChild(currentAST, returnAST);
```

```
                                        }
                                        else {
                                                break _loop1463;
                                        }

                                } while (true);
                                }
                                expression_AST = (AST)currentAST.root;
                        }
                        catch (RecognitionException ex) {
                                reportError(ex);
                                consume();
                                consumeUntil(_tokenSet_6);
                        }
                        returnAST = expression_AST;
                }

                public final void case_statement() throws RecognitionException, TokenStreamException {

                        returnAST = null;
                        ASTPair currentAST = new ASTPair();
                        AST case_statement_AST = null;

                        try {      // for error handling
                                match(CASE);
                                condition();
                                astFactory.addASTChild(currentAST, returnAST);
                                match(COLON);
                                codeBlock();
                                astFactory.addASTChild(currentAST, returnAST);
                                case_statement_AST = (AST)currentAST.root;
                                case_statement_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NCaseStatement,"NCaseStatement")).add(case_statement_AST));
                                currentAST.root = case_statement_AST;
                                currentAST.child = case_statement_AST!=null
&&case_statement_AST.getFirstChild()!=null ?
                                        case_statement_AST.getFirstChild() : case_statement_AST;
                                currentAST.advanceChildToEnd();
                                case_statement_AST = (AST)currentAST.root;
                        }
                        catch (RecognitionException ex) {
                                reportError(ex);
                                consume();
                                consumeUntil(_tokenSet_7);
                        }
                        returnAST = case_statement_AST;
                }

                public final void default_statement() throws RecognitionException, TokenStreamException {

                        returnAST = null;
                        ASTPair currentAST = new ASTPair();
                        AST default_statement_AST = null;

                        try {      // for error handling
                                match(DEFAULT);
                                match(COLON);
                                codeBlock();
```

```
                    astFactory.addASTChild(currentAST, returnAST);
                    default_statement_AST = (AST)currentAST.root;
                    default_statement_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NDefaultStatement,"NDefaultStatement")).add(default_statement_AST)
);
                    currentAST.root = default_statement_AST;
                    currentAST.child = default_statement_AST!=null
&&default_statement_AST.getFirstChild()!=null ?
                        default_statement_AST.getFirstChild() : default_statement_AST;
                    currentAST.advanceChildToEnd();
                    default_statement_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_2);
                }
                returnAST = default_statement_AST;
        }

        public final void condition() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST condition_AST = null;

                try {      // for error handling
                    expression();
                    astFactory.addASTChild(currentAST, returnAST);
                    condition_AST = (AST)currentAST.root;
                    condition_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NCondition,"NCondition")).add(condition_AST));
                    currentAST.root = condition_AST;
                    currentAST.child = condition_AST!=null &&condition_AST.getFirstChild()!=null ?
                        condition_AST.getFirstChild() : condition_AST;
                    currentAST.advanceChildToEnd();
                    condition_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_8);
                }
                returnAST = condition_AST;
        }

        public final void codeBlock() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST codeBlock_AST = null;

                try {      // for error handling
                    {
                    statement();
                    astFactory.addASTChild(currentAST, returnAST);
                    }
                    codeBlock_AST = (AST)currentAST.root;
```

```
                    codeBlock_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NCodeBlock,"NCodeBlock")).add(codeBlock_AST));
                    currentAST.root = codeBlock_AST;
                    currentAST.child = codeBlock_AST!=null &&codeBlock_AST.getFirstChild()!=null
?
                        codeBlock_AST.getFirstChild() : codeBlock_AST;
                    currentAST.advanceChildToEnd();
                    codeBlock_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_4);
                }
                returnAST = codeBlock_AST;
        }

        public final void ifBlock() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST ifBlock_AST = null;

                try {      // for error handling
                    match(IF);
                    match(LPAREN);
                    condition();
                    astFactory.addASTChild(currentAST, returnAST);
                    match(RPAREN);
                    codeBlock();
                    astFactory.addASTChild(currentAST, returnAST);
                    ifBlock_AST = (AST)currentAST.root;
                    ifBlock_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NIfBlock,"NIfBlock")).add(ifBlock_AST));
                    currentAST.root = ifBlock_AST;
                    currentAST.child = ifBlock_AST!=null &&ifBlock_AST.getFirstChild()!=null ?
                            ifBlock_AST.getFirstChild() : ifBlock_AST;
                    currentAST.advanceChildToEnd();
                    ifBlock_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_4);
                }
                returnAST = ifBlock_AST;
        }

        public final void elseBlock() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST elseBlock_AST = null;

                try {      // for error handling
                    match(ELSE);
                    codeBlock();
                    astFactory.addASTChild(currentAST, returnAST);
```

```
                elseBlock_AST = (AST)currentAST.root;
                elseBlock_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NElseBlock,"NElseBlock")).add(elseBlock_AST));
                currentAST.root = elseBlock_AST;
                currentAST.child = elseBlock_AST!=null &&elseBlock_AST.getFirstChild()!=null ?
                        elseBlock_AST.getFirstChild() : elseBlock_AST;
                currentAST.advanceChildToEnd();
                elseBlock_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_4);
            }
            returnAST = elseBlock_AST;
        }

        public final void for_loop() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST for_loop_AST = null;

            try {      // for error handling
                AST tmp32_AST = null;
                tmp32_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp32_AST);
                match(FOR);
                match(LPAREN);
                for_expression();
                astFactory.addASTChild(currentAST, returnAST);
                match(SEMI);
                loop_cntrl();
                astFactory.addASTChild(currentAST, returnAST);
                for_loop_AST = (AST)currentAST.root;
            }
            catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_4);
            }
            returnAST = for_loop_AST;
        }

        public final void for_expression() throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST for_expression_AST = null;

            try {      // for error handling
                switch ( LA(1)) {
                case ID:
                {
                        assign_expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        for_expression_AST = (AST)currentAST.root;
                        break;
```

```
                }
                case CHAR:
                case INT:
                case BOOL:
                case DOUBLE:
                case STRING:
                {
                        for_declaration();
                        astFactory.addASTChild(currentAST, returnAST);
                        for_expression_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_9);
        }
        returnAST = for_expression_AST;
}

public final void loop_cntrl() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST loop_cntrl_AST = null;

        try {      // for error handling
                condition();
                astFactory.addASTChild(currentAST, returnAST);
                match(SEMI);
                forBlock();
                astFactory.addASTChild(currentAST, returnAST);
                loop_cntrl_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_4);
        }
        returnAST = loop_cntrl_AST;
}

public final void assign_expression() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST assign_expression_AST = null;
        AST a_AST = null;

        try {      // for error handling
                assign_ID();
                astFactory.addASTChild(currentAST, returnAST);
```

```
                    assignment_operator();
                    a_AST = (AST)returnAST;
                    expression();
                    astFactory.addASTChild(currentAST, returnAST);
                    assign_expression_AST = (AST)currentAST.root;
                    assign_expression_AST = (AST)astFactory.make( (new
ASTArray(2)).add(a_AST).add(assign_expression_AST));
                    currentAST.root = assign_expression_AST;
                    currentAST.child = assign_expression_AST!=null
&&assign_expression_AST.getFirstChild()!=null ?
                        assign_expression_AST.getFirstChild() : assign_expression_AST;
                    currentAST.advanceChildToEnd();
                    assign_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_10);
                }
                returnAST = assign_expression_AST;
        }

        public final void for_declaration() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST for_declaration_AST = null;

                try {      // for error handling
                    type_specifier();
                    astFactory.addASTChild(currentAST, returnAST);
                    declarator();
                    astFactory.addASTChild(currentAST, returnAST);
                    match(ASSIGN);
                    expression();
                    astFactory.addASTChild(currentAST, returnAST);
                    for_declaration_AST = (AST)currentAST.root;
                    for_declaration_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NForDeclaration,"NForDeclaration")).add(for_declaration_AST));
                    currentAST.root = for_declaration_AST;
                    currentAST.child = for_declaration_AST!=null
&&for_declaration_AST.getFirstChild()!=null ?
                        for_declaration_AST.getFirstChild() : for_declaration_AST;
                    currentAST.advanceChildToEnd();
                    for_declaration_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_9);
                }
                returnAST = for_declaration_AST;
        }

        public final void type_specifier() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
```

```
AST type_specifier_AST = null;

try {      // for error handling
          {
          switch ( LA(1)) {
          case CHAR:
          {
                    AST tmp37_AST = null;
                    tmp37_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp37_AST);
                    match(CHAR);
                    break;
          }
          case INT:
          {
                    AST tmp38_AST = null;
                    tmp38_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp38_AST);
                    match(INT);
                    break;
          }
          case BOOL:
          {
                    AST tmp39_AST = null;
                    tmp39_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp39_AST);
                    match(BOOL);
                    break;
          }
          case DOUBLE:
          {
                    AST tmp40_AST = null;
                    tmp40_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp40_AST);
                    match(DOUBLE);
                    break;
          }
          case STRING:
          {
                    AST tmp41_AST = null;
                    tmp41_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp41_AST);
                    match(STRING);
                    break;
          }
          default:
          {
                    throw new NoViableAltException(LT(1), getFilename());
          }
          }
          }
          type_specifier_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
          reportError(ex);
          consume();
          consumeUntil(_tokenSet_11);
}
```

```
		returnAST = type_specifier_AST;
	}

	public final void declarator() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST declarator_AST = null;

		try {	// for error handling
			AST tmp42_AST = null;
			tmp42_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp42_AST);
			match(ID);
			{
			switch ( LA(1)) {
			case LBRACKET:
			{
				match(LBRACKET);
				expression();
				astFactory.addASTChild(currentAST, returnAST);
				match(RBRACKET);
				declarator_AST = (AST)currentAST.root;
				declarator_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NDeclareList,"NDeclareList")).add(declarator_AST));
				currentAST.root = declarator_AST;
				currentAST.child = declarator_AST!=null
&&declarator_AST.getFirstChild()!=null ?
					declarator_AST.getFirstChild() : declarator_AST;
				currentAST.advanceChildToEnd();
				break;
			}
			case SEMI:
			case ASSIGN:
			case COMMA:
			{
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1), getFilename());
			}
			}
			}
			declarator_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_12);
		}
		returnAST = declarator_AST;
	}

	public final void forBlock() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
```

```
		AST forBlock_AST = null;
		AST ax_AST = null;
		AST st_AST = null;

		try {	// for error handling
			{
			assign_expression();
			ax_AST = (AST)returnAST;
			match(RPAREN);
			statement();
			st_AST = (AST)returnAST;
			}
			forBlock_AST = (AST)currentAST.root;
			forBlock_AST = (AST)astFactory.make( (new
ASTArray(3)).add(astFactory.create(NForBlock,"NForBlock")).add(st_AST).add(ax_AST));
			currentAST.root = forBlock_AST;
			currentAST.child = forBlock_AST!=null &&forBlock_AST.getFirstChild()!=null ?
				forBlock_AST.getFirstChild() : forBlock_AST;
			currentAST.advanceChildToEnd();
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_4);
		}
		returnAST = forBlock_AST;
	}

	public final void assign_ID() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST assign_ID_AST = null;

		try {	// for error handling
			AST tmp46_AST = null;
			tmp46_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp46_AST);
			match(ID);
			{
			switch ( LA(1)) {
			case LBRACKET:
			{
				match(LBRACKET);
				expression();
				astFactory.addASTChild(currentAST, returnAST);
				match(RBRACKET);
				assign_ID_AST = (AST)currentAST.root;
				assign_ID_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NAssignList,"NAssignList")).add(assign_ID_AST));
				currentAST.root = assign_ID_AST;
				currentAST.child = assign_ID_AST!=null
&&assign_ID_AST.getFirstChild()!=null ?
					assign_ID_AST.getFirstChild() : assign_ID_AST;
				currentAST.advanceChildToEnd();
				break;
			}
			case ASSIGN:
```

```
                case ADD_ASSIGN:
                case SUB_ASSIGN:
                case DIV_ASSIGN:
                case STAR_ASSIGN:
                case MOD_ASSIGN:
                {
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                assign_ID_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_13);
        }
        returnAST = assign_ID_AST;
}

public final void assignment_operator() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST assignment_operator_AST = null;

        try {      // for error handling
                {
                switch ( LA(1)) {
                case ASSIGN:
                {
                        AST tmp49_AST = null;
                        tmp49_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp49_AST);
                        match(ASSIGN);
                        break;
                }
                case ADD_ASSIGN:
                {
                        AST tmp50_AST = null;
                        tmp50_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp50_AST);
                        match(ADD_ASSIGN);
                        break;
                }
                case SUB_ASSIGN:
                {
                        AST tmp51_AST = null;
                        tmp51_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp51_AST);
                        match(SUB_ASSIGN);
                        break;
                }
                case DIV_ASSIGN:
```

```
                {
                        AST tmp52_AST = null;
                        tmp52_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp52_AST);
                        match(DIV_ASSIGN);
                        break;
                }
                case STAR_ASSIGN:
                {
                        AST tmp53_AST = null;
                        tmp53_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp53_AST);
                        match(STAR_ASSIGN);
                        break;
                }
                case MOD_ASSIGN:
                {
                        AST tmp54_AST = null;
                        tmp54_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp54_AST);
                        match(MOD_ASSIGN);
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                assignment_operator_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_14);
        }
        returnAST = assignment_operator_AST;
}

public final void call_expression() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST call_expression_AST = null;

        try {      // for error handling
                AST tmp55_AST = null;
                tmp55_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp55_AST);
                match(ID);
                {
                int _cnt1457=0;
                _loop1457:
                do {
                        if ((LA(1)==DOT)) {
                                AST tmp56_AST = null;
                                tmp56_AST = astFactory.create(LT(1));
                                astFactory.addASTChild(currentAST, tmp56_AST);
```

```
match(DOT);
AST tmp57_AST = null;
tmp57_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp57_AST);
match(ID);
{
switch ( LA(1)) {
case LPAREN:
{
	AST tmp58_AST = null;
	tmp58_AST = astFactory.create(LT(1));
	astFactory.addASTChild(currentAST, tmp58_AST);
	match(LPAREN);
	{
	switch ( LA(1)) {
	case LPAREN:
	case ID:
	case PLUS:
	case MINUS:
	case LNOT:
	case STRING_LITERAL:
	case DOUBLE_CONST:
	case INT_CONST:
	case TRUE:
	case FALSE:
	{
		argument_expression_list();
		astFactory.addASTChild(currentAST,
returnAST);
		break;
	}
	case RPAREN:
	{
		break;
	}
	default:
	{
		throw	new	NoViableAltException(LT(1),
getFilename());
	}
	}
	}
	AST tmp59_AST = null;
	tmp59_AST = astFactory.create(LT(1));
	astFactory.addASTChild(currentAST, tmp59_AST);
	match(RPAREN);
	break;
}
case LBRACKET:
{
	AST tmp60_AST = null;
	tmp60_AST = astFactory.create(LT(1));
	astFactory.addASTChild(currentAST, tmp60_AST);
	match(LBRACKET);
	{
	switch ( LA(1)) {
	case LPAREN:
	case ID:
```

```
                                        case PLUS:
                                        case MINUS:
                                        case LNOT:
                                        case STRING_LITERAL:
                                        case DOUBLE_CONST:
                                        case INT_CONST:
                                        case TRUE:
                                        case FALSE:
                                        {
                                                argument_expression_list();
                                                astFactory.addASTChild(currentAST,
returnAST);
                                                break;
                                        }
                                        case RBRACKET:
                                        {
                                                break;
                                        }
                                        default:
                                        {
                                                throw     new     NoViableAltException(LT(1),
getFilename());
                                        }
                                        }
                                        }
                                        AST tmp61_AST = null;
                                        tmp61_AST = astFactory.create(LT(1));
                                        astFactory.addASTChild(currentAST, tmp61_AST);
                                        match(RBRACKET);
                                        break;
                                }
                                case EOF:
                                case RPAREN:
                                case RCURLY:
                                case SEMI:
                                case COLON:
                                case RBRACKET:
                                case DOT:
                                case COMMA:
                                case LOR:
                                case LAND:
                                case EQUAL:
                                case NOT_EQUAL:
                                case LT_OP:
                                case GT:
                                case LTE:
                                case GTE:
                                case PLUS:
                                case MINUS:
                                case STAR:
                                case DIV:
                                case MOD:
                                {
                                        break;
                                }
                                default:
                                {
```

```
                                                throw new NoViableAltException(LT(1),
getFilename());
                                        }
                                        }
                                        }
                                }
                                else {
                                        if ( _cnt1457>=1 ) { break _loop1457; } else {throw new
NoViableAltException(LT(1), getFilename());}
                                }

                                _cnt1457++;
                        } while (true);
                        }
                        call_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_15);
                }
                returnAST = call_expression_AST;
        }

        public final void argument_expression_list() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST argument_expression_list_AST = null;

                try {      // for error handling
                        expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        _loop1460:
                        do {
                                if ((LA(1)==COMMA)) {
                                        AST tmp62_AST = null;
                                        tmp62_AST = astFactory.create(LT(1));
                                        astFactory.addASTChild(currentAST, tmp62_AST);
                                        match(COMMA);
                                        expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                        break _loop1460;
                                }

                        } while (true);
                        }
                        argument_expression_list_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_16);
                }
                returnAST = argument_expression_list_AST;
```

```
}

public final void logical_and_expression() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST logical_and_expression_AST = null;

	try {	// for error handling
		equality_expression();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop1466:
		do {
			if ((LA(1)==LAND)) {
				AST tmp63_AST = null;
				tmp63_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp63_AST);
				match(LAND);
				equality_expression();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop1466;
			}

		} while (true);
		}
		logical_and_expression_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		consume();
		consumeUntil(_tokenSet_17);
	}
	returnAST = logical_and_expression_AST;
}

public final void equality_expression() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST equality_expression_AST = null;

	try {	// for error handling
		relational_expression();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop1470:
		do {
			if ((LA(1)==EQUAL||LA(1)==NOT_EQUAL)) {
				{
				switch ( LA(1)) {
				case EQUAL:
				{
					AST tmp64_AST = null;
					tmp64_AST = astFactory.create(LT(1));
					astFactory.makeASTRoot(currentAST, tmp64_AST);
```

```
                                        match(EQUAL);
                                        break;
                                }
                                case NOT_EQUAL:
                                {
                                        AST tmp65_AST = null;
                                        tmp65_AST = astFactory.create(LT(1));
                                        astFactory.makeASTRoot(currentAST, tmp65_AST);
                                        match(NOT_EQUAL);
                                        break;
                                }
                                default:
                                {
                                        throw new NoViableAltException(LT(1),
getFilename());
                                }
                                }
                                }
                                relational_expression();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                                break _loop1470;
                        }

                } while (true);
                }
                equality_expression_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_18);
        }
        returnAST = equality_expression_AST;
}

public final void relational_expression() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST relational_expression_AST = null;

        try {     // for error handling
                additive_expression();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop1474:
                do {
                        if (((LA(1) >= LT_OP && LA(1) <= GTE))) {
                                {
                                switch ( LA(1)) {
                                case LT_OP:
                                {
                                        AST tmp66_AST = null;
                                        tmp66_AST = astFactory.create(LT(1));
                                        astFactory.makeASTRoot(currentAST, tmp66_AST);
                                        match(LT_OP);
```

```
                                        break;
                                    }
                                    case GT:
                                    {
                                        AST tmp67_AST = null;
                                        tmp67_AST = astFactory.create(LT(1));
                                        astFactory.makeASTRoot(currentAST, tmp67_AST);
                                        match(GT);
                                        break;
                                    }
                                    case LTE:
                                    {
                                        AST tmp68_AST = null;
                                        tmp68_AST = astFactory.create(LT(1));
                                        astFactory.makeASTRoot(currentAST, tmp68_AST);
                                        match(LTE);
                                        break;
                                    }
                                    case GTE:
                                    {
                                        AST tmp69_AST = null;
                                        tmp69_AST = astFactory.create(LT(1));
                                        astFactory.makeASTRoot(currentAST, tmp69_AST);
                                        match(GTE);
                                        break;
                                    }
                                    default:
                                    {
                                        throw new NoViableAltException(LT(1),
getFilename());
                                    }
                                    }
                                    }
                                    additive_expression();
                                    astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                    break _loop1474;
                                }

                            } while (true);
                            }
                            relational_expression_AST = (AST)currentAST.root;
                        }
                        catch (RecognitionException ex) {
                            reportError(ex);
                            consume();
                            consumeUntil(_tokenSet_19);
                        }
                        returnAST = relational_expression_AST;
                    }

                    public final void additive_expression() throws RecognitionException, TokenStreamException {

                        returnAST = null;
                        ASTPair currentAST = new ASTPair();
                        AST additive_expression_AST = null;
```

```
		try {	// for error handling
			multiplicative_expression();
			astFactory.addASTChild(currentAST, returnAST);
			{
			_loop1478:
			do {
				if ((LA(1)==PLUS||LA(1)==MINUS)) {
					{
					switch ( LA(1)) {
					case PLUS:
					{
						AST tmp70_AST = null;
						tmp70_AST = astFactory.create(LT(1));
						astFactory.makeASTRoot(currentAST, tmp70_AST);
						match(PLUS);
						break;
					}
					case MINUS:
					{
						AST tmp71_AST = null;
						tmp71_AST = astFactory.create(LT(1));
						astFactory.makeASTRoot(currentAST, tmp71_AST);
						match(MINUS);
						break;
					}
					default:
					{
						throw new NoViableAltException(LT(1),
getFilename());
					}
					}
					}
					multiplicative_expression();
					astFactory.addASTChild(currentAST, returnAST);
				}
				else {
					break _loop1478;
				}

			} while (true);
			}
			additive_expression_AST = (AST)currentAST.root;
		}
		catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_20);
		}
		returnAST = additive_expression_AST;
	}

	public final void multiplicative_expression() throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST multiplicative_expression_AST = null;

		try {	// for error handling
```

```
                    unary_expression();
                    astFactory.addASTChild(currentAST, returnAST);
                    {
                    _loop1482:
                    do {
                        if (((LA(1) >= STAR && LA(1) <= MOD))) {
                            {
                            switch ( LA(1)) {
                            case STAR:
                            {
                                AST tmp72_AST = null;
                                tmp72_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp72_AST);
                                match(STAR);
                                break;
                            }
                            case DIV:
                            {
                                AST tmp73_AST = null;
                                tmp73_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp73_AST);
                                match(DIV);
                                break;
                            }
                            case MOD:
                            {
                                AST tmp74_AST = null;
                                tmp74_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp74_AST);
                                match(MOD);
                                break;
                            }
                            default:
                            {
                                throw new NoViableAltException(LT(1),
getFilename());
                            }
                            }
                            }
                            unary_expression();
                            astFactory.addASTChild(currentAST, returnAST);
                        }
                        else {
                            break _loop1482;
                        }

                    } while (true);
                    }
                    multiplicative_expression_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_21);
                }
                returnAST = multiplicative_expression_AST;
            }
```

```
public final void unary_expression() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST unary_expression_AST = null;
	AST u_AST = null;

	try {	// for error handling
		{
		switch ( LA(1)) {
		case LPAREN:
		case ID:
		case STRING_LITERAL:
		case DOUBLE_CONST:
		case INT_CONST:
		case TRUE:
		case FALSE:
		{
			primary_expression();
			astFactory.addASTChild(currentAST, returnAST);
			break;
		}
		case PLUS:
		case MINUS:
		case LNOT:
		{
			unary_operator();
			u_AST = (AST)returnAST;
			astFactory.addASTChild(currentAST, returnAST);
			primary_expression();
			astFactory.addASTChild(currentAST, returnAST);
			unary_expression_AST = (AST)currentAST.root;
			unary_expression_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NUnary,"NUnary")).add(unary_expression_AST));
			currentAST.root = unary_expression_AST;
			currentAST.child = unary_expression_AST!=null
&&unary_expression_AST.getFirstChild()!=null ?
				unary_expression_AST.getFirstChild() :
unary_expression_AST;
			currentAST.advanceChildToEnd();
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		unary_expression_AST = (AST)currentAST.root;
	}
	catch (RecognitionException ex) {
		reportError(ex);
		consume();
		consumeUntil(_tokenSet_15);
	}
	returnAST = unary_expression_AST;
}
```

```
public final void primary_expression() throws RecognitionException, TokenStreamException {

	returnAST = null;
	ASTPair currentAST = new ASTPair();
	AST primary_expression_AST = null;

	try {	// for error handling
		{
		switch ( LA(1)) {
		case STRING_LITERAL:
		{
			AST tmp75_AST = null;
			tmp75_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp75_AST);
			match(STRING_LITERAL);
			break;
		}
		case DOUBLE_CONST:
		{
			AST tmp76_AST = null;
			tmp76_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp76_AST);
			match(DOUBLE_CONST);
			break;
		}
		case INT_CONST:
		{
			AST tmp77_AST = null;
			tmp77_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp77_AST);
			match(INT_CONST);
			break;
		}
		case LPAREN:
		{
			AST tmp78_AST = null;
			tmp78_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp78_AST);
			match(LPAREN);
			expression();
			astFactory.addASTChild(currentAST, returnAST);
			match(RPAREN);
			break;
		}
		case TRUE:
		{
			AST tmp80_AST = null;
			tmp80_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp80_AST);
			match(TRUE);
			break;
		}
		case FALSE:
		{
			AST tmp81_AST = null;
			tmp81_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp81_AST);
			match(FALSE);
```

```
                        break;
                    }
                    default:
                        if ((LA(1)==ID) && (_tokenSet_22.member(LA(2)))) {
                            AST tmp82_AST = null;
                            tmp82_AST = astFactory.create(LT(1));
                            astFactory.addASTChild(currentAST, tmp82_AST);
                            match(ID);
                            {
                            switch ( LA(1)) {
                            case LBRACKET:
                            {
                                match(LBRACKET);
                                expression();
                                astFactory.addASTChild(currentAST, returnAST);
                                match(RBRACKET);
                                primary_expression_AST = (AST)currentAST.root;
                                    primary_expression_AST =
(AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NListDeclaration,"NListDeclaration")).add(primary_expression_AST));
                                currentAST.root = primary_expression_AST;
                                currentAST.child = primary_expression_AST!=null
&&primary_expression_AST.getFirstChild()!=null ?
                                        primary_expression_AST.getFirstChild() :
primary_expression_AST;
                                currentAST.advanceChildToEnd();
                                break;
                            }
                            case EOF:
                            case RPAREN:
                            case RCURLY:
                            case SEMI:
                            case COLON:
                            case RBRACKET:
                            case COMMA:
                            case LOR:
                            case LAND:
                            case EQUAL:
                            case NOT_EQUAL:
                            case LT_OP:
                            case GT:
                            case LTE:
                            case GTE:
                            case PLUS:
                            case MINUS:
                            case STAR:
                            case DIV:
                            case MOD:
                            {
                                break;
                            }
                            default:
                            {
                                throw new NoViableAltException(LT(1),
getFilename());
                            }
                            }
                            }
```

```
                        }
                        else if ((LA(1)==ID) && (LA(2)==DOT)) {
                                call_expression();
                                astFactory.addASTChild(currentAST, returnAST);
                        }
                else {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                primary_expression_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_15);
        }
        returnAST = primary_expression_AST;
}

public final void unary_operator() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST unary_operator_AST = null;

        try {     // for error handling
                switch ( LA(1)) {
                case PLUS:
                {
                        AST tmp85_AST = null;
                        tmp85_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp85_AST);
                        match(PLUS);
                        unary_operator_AST = (AST)currentAST.root;
                        break;
                }
                case MINUS:
                {
                        AST tmp86_AST = null;
                        tmp86_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp86_AST);
                        match(MINUS);
                        unary_operator_AST = (AST)currentAST.root;
                        break;
                }
                case LNOT:
                {
                        AST tmp87_AST = null;
                        tmp87_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp87_AST);
                        match(LNOT);
                        unary_operator_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
```

```
                    }
                    }
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_23);
                }
                returnAST = unary_operator_AST;
            }

            public final void declaration() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST declaration_AST = null;

                try {      // for error handling
                    type_specifier();
                    astFactory.addASTChild(currentAST, returnAST);
                    declarator_list();
                    astFactory.addASTChild(currentAST, returnAST);
                    match(SEMI);
                    declaration_AST = (AST)currentAST.root;
                    declaration_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NDeclaration,"NDeclaration")).add(declaration_AST));
                    currentAST.root = declaration_AST;
                    currentAST.child = declaration_AST!=null
&&declaration_AST.getFirstChild()!=null ?
                        declaration_AST.getFirstChild() : declaration_AST;
                    currentAST.advanceChildToEnd();
                    declaration_AST = (AST)currentAST.root;
                }
                catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_24);
                }
                returnAST = declaration_AST;
            }

            public final void declarator_list() throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST declarator_list_AST = null;

                try {      // for error handling
                    declarator();
                    astFactory.addASTChild(currentAST, returnAST);
                    {
                    _loop1495:
                    do {
                        if ((LA(1)==COMMA)) {
                            match(COMMA);
                            declarator();
                            astFactory.addASTChild(currentAST, returnAST);
                        }
```

```
                        else {
                                break _loop1495;
                        }

                } while (true);
                }
                declarator_list_AST = (AST)currentAST.root;
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_9);
        }
        returnAST = declarator_list_AST;
}

public final void initializer_list() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST initializer_list_AST = null;

        try {      // for error handling
                {
                switch ( LA(1)) {
                case LPAREN:
                case ID:
                case PLUS:
                case MINUS:
                case LNOT:
                case STRING_LITERAL:
                case DOUBLE_CONST:
                case INT_CONST:
                case TRUE:
                case FALSE:
                {
                        expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                }
                case LCURLY:
                {
                        match(LCURLY);
                        {
                        expression();
                        astFactory.addASTChild(currentAST, returnAST);
                        {
                        _loop1502:
                        do {
                                if ((LA(1)==COMMA)) {
                                        match(COMMA);
                                        expression();
                                        astFactory.addASTChild(currentAST, returnAST);
                                }
                                else {
                                        break _loop1502;
                                }
```

```
                                        } while (true);
                                        }
                                        }
                                        match(RCURLY);
                                        break;
                                }
                                default:
                                {
                                        throw new NoViableAltException(LT(1), getFilename());
                                }
                                }
                                }
                                initializer_list_AST = (AST)currentAST.root;
                                initializer_list_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(NInitializer,"NInitializer")).add(initializer_list_AST));
                                currentAST.root = initializer_list_AST;
                                currentAST.child = initializer_list_AST!=null
&&initializer_list_AST.getFirstChild()!=null ?
                                        initializer_list_AST.getFirstChild() : initializer_list_AST;
                                currentAST.advanceChildToEnd();
                                initializer_list_AST = (AST)currentAST.root;
                        }
                        catch (RecognitionException ex) {
                                reportError(ex);
                                consume();
                                consumeUntil(_tokenSet_0);
                        }
                        returnAST = initializer_list_AST;
                }

                public final void imaginary_token() throws RecognitionException, TokenStreamException {

                        returnAST = null;
                        ASTPair currentAST = new ASTPair();
                        AST imaginary_token_AST = null;

                        try {      // for error handling
                                switch ( LA(1)) {
                                case NRootAST:
                                {
                                        AST tmp93_AST = null;
                                        tmp93_AST = astFactory.create(LT(1));
                                        astFactory.addASTChild(currentAST, tmp93_AST);
                                        match(NRootAST);
                                        imaginary_token_AST = (AST)currentAST.root;
                                        break;
                                }
                                case NDeclaration:
                                {
                                        AST tmp94_AST = null;
                                        tmp94_AST = astFactory.create(LT(1));
                                        astFactory.addASTChild(currentAST, tmp94_AST);
                                        match(NDeclaration);
                                        imaginary_token_AST = (AST)currentAST.root;
                                        break;
                                }
                                case NUnionDeclaration:
                                {
```

```
		AST tmp95_AST = null;
		tmp95_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp95_AST);
		match(NUnionDeclaration);
		imaginary_token_AST = (AST)currentAST.root;
		break;
	}
	case NUnary:
	{
		AST tmp96_AST = null;
		tmp96_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp96_AST);
		match(NUnary);
		imaginary_token_AST = (AST)currentAST.root;
		break;
	}
	case NEmptyStatement:
	{
		AST tmp97_AST = null;
		tmp97_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp97_AST);
		match(NEmptyStatement);
		imaginary_token_AST = (AST)currentAST.root;
		break;
	}
	case NInitializer:
	{
		AST tmp98_AST = null;
		tmp98_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp98_AST);
		match(NInitializer);
		imaginary_token_AST = (AST)currentAST.root;
		break;
	}
	case NExterntype:
	{
		AST tmp99_AST = null;
		tmp99_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp99_AST);
		match(NExterntype);
		imaginary_token_AST = (AST)currentAST.root;
		break;
	}
	case NCondition:
	{
		AST tmp100_AST = null;
		tmp100_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp100_AST);
		match(NCondition);
		imaginary_token_AST = (AST)currentAST.root;
		break;
	}
	case NCodeBlock:
	{
		AST tmp101_AST = null;
		tmp101_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp101_AST);
		match(NCodeBlock);
```

```
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NIfThenBlock:
{
            AST tmp102_AST = null;
            tmp102_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp102_AST);
            match(NIfThenBlock);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NIfBlock:
{
            AST tmp103_AST = null;
            tmp103_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp103_AST);
            match(NIfBlock);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NElseBlock:
{
            AST tmp104_AST = null;
            tmp104_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp104_AST);
            match(NElseBlock);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NForBlock:
{
            AST tmp105_AST = null;
            tmp105_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp105_AST);
            match(NForBlock);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NCaseStatement:
{
            AST tmp106_AST = null;
            tmp106_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp106_AST);
            match(NCaseStatement);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NDefaultStatement:
{
            AST tmp107_AST = null;
            tmp107_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp107_AST);
            match(NDefaultStatement);
            imaginary_token_AST = (AST)currentAST.root;
            break;
}
case NSwitchBlock:
```

```
{
        AST tmp108_AST = null;
        tmp108_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp108_AST);
        match(NSwitchBlock);
        imaginary_token_AST = (AST)currentAST.root;
        break;
}
case NCaseCondition:
{
        AST tmp109_AST = null;
        tmp109_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp109_AST);
        match(NCaseCondition);
        imaginary_token_AST = (AST)currentAST.root;
        break;
}
case NForDeclaration:
{
        AST tmp110_AST = null;
        tmp110_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp110_AST);
        match(NForDeclaration);
        imaginary_token_AST = (AST)currentAST.root;
        break;
}
case NDeclareList:
{
        AST tmp111_AST = null;
        tmp111_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp111_AST);
        match(NDeclareList);
        imaginary_token_AST = (AST)currentAST.root;
        break;
}
case NListDeclaration:
{
        AST tmp112_AST = null;
        tmp112_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp112_AST);
        match(NListDeclaration);
        imaginary_token_AST = (AST)currentAST.root;
        break;
}
case NAssignList:
{
        AST tmp113_AST = null;
        tmp113_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp113_AST);
        match(NAssignList);
        imaginary_token_AST = (AST)currentAST.root;
        break;
}
case NStartCondition:
{
        AST tmp114_AST = null;
        tmp114_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp114_AST);
```

```
                        match(NStartCondition);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
                case NStatement:
                {
                        AST tmp115_AST = null;
                        tmp115_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp115_AST);
                        match(NStatement);
                        imaginary_token_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
        }
        catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_0);
        }
        returnAST = imaginary_token_AST;
}


public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "\"main\"",
        "'('",
        "')'",
        "'{'",
        "'}'",
        "';'",
        "\"switch\"",
        "\"case\"",
        "':'",
        "\"default\"",
        "\"if\"",
        "\"else\"",
        "\"while\"",
        "\"do\"",
        "\"for\"",
        "ASSIGN",
        "an identifier",
        "'['",
        "']'",
        "ADD_ASSIGN",
        "SUB_ASSIGN",
        "DIV_ASSIGN",
        "STAR_ASSIGN",
        "MOD_ASSIGN",
        "DOT",
```

```
	"COMMA",
	"LOR",
	"LAND",
	"EQUAL",
	"NOT_EQUAL",
	"LT_OP",
	"GT",
	"LTE",
	"GTE",
	"PLUS",
	"MINUS",
	"STAR",
	"DIV",
	"MOD",
	"LNOT",
	"STRING_LITERAL",
	"DOUBLE_CONST",
	"INT_CONST",
	"\"true\"",
	"\"false\"",
	"\"char\"",
	"\"int\"",
	"\"bool\"",
	"\"double\"",
	"\"string\"",
	"NRootAST",
	"NDeclaration",
	"NUnionDeclaration",
	"NUnary",
	"NEmptyStatement",
	"NInitializer",
	"NExterntype",
	"NCondition",
	"NCodeBlock",
	"NIfThenBlock",
	"NIfBlock",
	"NElseBlock",
	"NForBlock",
	"NCaseStatement",
	"NDefaultStatement",
	"NSwitchBlock",
	"NCaseCondition",
	"NForDeclaration",
	"NDeclareList",
	"NListDeclaration",
	"NAssignList",
	"NStartCondition",
	"NStatement",
	"\"cout\"",
	"\"endl\"",
	"WS",
	"NL",
	"COMMENT",
	"'<<'",
	"Escape",
	"DIGIT",
	"a number"
};
```

```
protected void buildTokenTypeASTClassMap() {
	tokenTypeToASTClassMap=null;
};

private static final long[] mk_tokenSet_0() {
	long[] data = { 2L, 0L};
	return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
	long[] data = { 1525248L, 0L};
	return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
private static final long[] mk_tokenSet_2() {
	long[] data = { 256L, 0L};
	return data;
}
public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());
private static final long[] mk_tokenSet_3() {
	long[] data = { 262668288L, 0L};
	return data;
}
public static final BitSet _tokenSet_3 = new BitSet(mk_tokenSet_3());
private static final long[] mk_tokenSet_4() {
	long[] data = { 1568512L, 0L};
	return data;
}
public static final BitSet _tokenSet_4 = new BitSet(mk_tokenSet_4());
private static final long[] mk_tokenSet_5() {
	long[] data = { 554979026796322L, 0L};
	return data;
}
public static final BitSet _tokenSet_5 = new BitSet(mk_tokenSet_5());
private static final long[] mk_tokenSet_6() {
	long[] data = { 541070146L, 0L};
	return data;
}
public static final BitSet _tokenSet_6 = new BitSet(mk_tokenSet_6());
private static final long[] mk_tokenSet_7() {
	long[] data = { 10496L, 0L};
	return data;
}
public static final BitSet _tokenSet_7 = new BitSet(mk_tokenSet_7());
private static final long[] mk_tokenSet_8() {
	long[] data = { 4672L, 0L};
	return data;
}
public static final BitSet _tokenSet_8 = new BitSet(mk_tokenSet_8());
private static final long[] mk_tokenSet_9() {
	long[] data = { 512L, 0L};
	return data;
}
public static final BitSet _tokenSet_9 = new BitSet(mk_tokenSet_9());
private static final long[] mk_tokenSet_10() {
	long[] data = { 576L, 0L};
	return data;
```

```
}
public static final BitSet _tokenSet_10 = new BitSet(mk_tokenSet_10());
private static final long[] mk_tokenSet_11() {
	long[] data = { 1048576L, 0L};
	return data;
}
public static final BitSet _tokenSet_11 = new BitSet(mk_tokenSet_11());
private static final long[] mk_tokenSet_12() {
	long[] data = { 537395712L, 0L};
	return data;
}
public static final BitSet _tokenSet_12 = new BitSet(mk_tokenSet_12());
private static final long[] mk_tokenSet_13() {
	long[] data = { 260571136L, 0L};
	return data;
}
public static final BitSet _tokenSet_13 = new BitSet(mk_tokenSet_13());
private static final long[] mk_tokenSet_14() {
	long[] data = { 554978495168544L, 0L};
	return data;
}
public static final BitSet _tokenSet_14 = new BitSet(mk_tokenSet_14());
private static final long[] mk_tokenSet_15() {
	long[] data = { 8795560350530L, 0L};
	return data;
}
public static final BitSet _tokenSet_15 = new BitSet(mk_tokenSet_15());
private static final long[] mk_tokenSet_16() {
	long[] data = { 4194368L, 0L};
	return data;
}
public static final BitSet _tokenSet_16 = new BitSet(mk_tokenSet_16());
private static final long[] mk_tokenSet_17() {
	long[] data = { 1614811970L, 0L};
	return data;
}
public static final BitSet _tokenSet_17 = new BitSet(mk_tokenSet_17());
private static final long[] mk_tokenSet_18() {
	long[] data = { 3762295618L, 0L};
	return data;
}
public static final BitSet _tokenSet_18 = new BitSet(mk_tokenSet_18());
private static final long[] mk_tokenSet_19() {
	long[] data = { 16647197506L, 0L};
	return data;
}
public static final BitSet _tokenSet_19 = new BitSet(mk_tokenSet_19());
private static final long[] mk_tokenSet_20() {
	long[] data = { 274345235266L, 0L};
	return data;
}
public static final BitSet _tokenSet_20 = new BitSet(mk_tokenSet_20());
private static final long[] mk_tokenSet_21() {
	long[] data = { 1098978956098L, 0L};
	return data;
}
public static final BitSet _tokenSet_21 = new BitSet(mk_tokenSet_21());
private static final long[] mk_tokenSet_22() {
```

```
		long[] data = { 8795562447682L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_22 = new BitSet(mk_tokenSet_22());
	private static final long[] mk_tokenSet_23() {
		long[] data = { 545357768425504L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_23 = new BitSet(mk_tokenSet_23());
	private static final long[] mk_tokenSet_24() {
		long[] data = { 17451448557585920L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_24 = new BitSet(mk_tokenSet_24());

	}

// start of XQString.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.io.PrintWriter;

class XQString extends XQDataType {
	String var;

	public XQString( String str ) {
		this.var = str;
	}

	public String typename() {
		return "string";
	}

	public XQDataType copy() {
		return new XQString( var );
	}

	public void print( PrintWriter w ) {
		if ( name != null )
			w.print( name + " = " );
		w.print( var );
		w.println();
	}

	public XQDataType plus( XQDataType b ) {
		if ( b instanceof XQString )
```

```
                        return new XQString( var + ((XQString)b).var );

                return error( b, "+" );
        }

        public XQDataType add( XQDataType b ) {
                if ( b instanceof XQString )
                {
                        var = var + ((XQString)b).var;
                        return this;
                }

                return error( b, "+=" );
        }
}

//start of XQSymbolTable.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.util.*;
import java.io.PrintWriter;

class XQSymbolTable extends HashMap {

        XQSymbolTable parent;

        public XQSymbolTable( XQSymbolTable parent) {
                this.parent = parent;
        }

        public final XQSymbolTable parent() {
                return parent;
        }

        public final boolean containsVar( String name ) {
                return containsKey( name );
        }

        public final XQDataType getValue( String name) {
                return (XQDataType) get(name);
        }

        public final void setValue( String name, XQDataType data) {
                put( name, data );
        }
```

```
	public final void setArrayValue( String name, XQDataType [] data) {
			put( name, data );
	}

	public final XQDataType [] getArrayValue( String name) {
		return (XQDataType []) get(name);
	}

	public void what( PrintWriter output ) {
		for ( Iterator it = values().iterator() ; it.hasNext(); )
		{
			XQDataType d = ((XQDataType)(it.next()));
			if ( !( d instanceof XQFunction && ((XQFunction)d).isInternal() ) )
				d.what( output );
		}
	}

	public void what() {
		what( new PrintWriter( System.out, true ) );
	}
}

// start of XQTokenType.java
// $ANTLR : "grammar.g" -> "XQParser.java"$

public interface XQTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int NStartCondition = 4;
	int LITERAL_main = 5;
	int LPAREN = 6;
	int RPAREN = 7;
	int LCURLY = 8;
	int RCURLY = 9;
	int IF = 10;
	int ELSE = 11;
	int SWITCH = 12;
	int CASE = 13;
	int DEFAULT = 14;
	int WHILE = 15;
	int DO = 16;
	int FOR = 17;
	int INT = 18;
	int DOUBLE = 19;
	int STRING = 20;
	int ALPHA = 21;
	int DIGIT = 22;
	int WS = 23;
	int NL = 24;
	int COMMENT = 25;
	int INTLIT = 26;
	int STRING_LITERAL = 27;
	int ID = 28;
	int DOT = 29;
	int COLON = 30;
	int SEMI = 31;
	int COMMA = 32;
	int EQUALS = 33;
```

```
	int LBRACKET = 34;
	int RBRACKET = 35;
	int NOT_EQUALS = 36;
	int LT = 37;
	int LTE = 38;
	int GT = 39;
	int GTE = 40;
	int PLUS = 41;
	int MINUS = 42;
	int TIMES = 43;
	int DIV = 44;
}

// start of XQVariable.java
/*
 * Created on Dec 14, 2003
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * @author Kin
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

import java.io.PrintWriter;

class XQVariable extends XQDataType {
	public XQVariable(String name) {
		super(name);
	}

	public String typename() {
		return "undefined-variable";
	}

	public XQDataType copy() {
		throw new XQException("Variable " + name + " has not been defined");
	}

	public void print(PrintWriter w) {
		w.println(name + " = <undefined>");
	}
}


// start of XQWalkerTOkenTypes.java
// $ANTLR : "walker.g" -> "XQWalker.java"$

public interface XQWalkerTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int LITERAL_main = 4;
	int LPAREN = 5;
	int RPAREN = 6;
```

```
int LCURLY = 7;
int RCURLY = 8;
int SEMI = 9;
int SWITCH = 10;
int CASE = 11;
int COLON = 12;
int DEFAULT = 13;
int IF = 14;
int ELSE = 15;
int WHILE = 16;
int DO = 17;
int FOR = 18;
int ASSIGN = 19;
int ID = 20;
int LBRACKET = 21;
int RBRACKET = 22;
int ADD_ASSIGN = 23;
int SUB_ASSIGN = 24;
int DIV_ASSIGN = 25;
int STAR_ASSIGN = 26;
int MOD_ASSIGN = 27;
int DOT = 28;
int COMMA = 29;
int LOR = 30;
int LAND = 31;
int EQUAL = 32;
int NOT_EQUAL = 33;
int LT_OP = 34;
int GT = 35;
int LTE = 36;
int GTE = 37;
int PLUS = 38;
int MINUS = 39;
int STAR = 40;
int DIV = 41;
int MOD = 42;
int LNOT = 43;
int STRING_LITERAL = 44;
int DOUBLE_CONST = 45;
int INT_CONST = 46;
int TRUE = 47;
int FALSE = 48;
int CHAR = 49;
int INT = 50;
int BOOL = 51;
int DOUBLE = 52;
int STRING = 53;
int NRootAST = 54;
int NDeclaration = 55;
int NUnionDeclaration = 56;
int NUnary = 57;
int NEmptyStatement = 58;
int NInitializer = 59;
int NExterntype = 60;
int NCondition = 61;
int NCodeBlock = 62;
int NIfThenBlock = 63;
int NIfBlock = 64;
```

```
        int NElseBlock = 65;
        int NForBlock = 66;
        int NCaseStatement = 67;
        int NDefaultStatement = 68;
        int NSwitchBlock = 69;
        int NCaseCondition = 70;
        int NForDeclaration = 71;
        int NDeclareList = 72;
        int NListDeclaration = 73;
        int NAssignList = 74;
        int NStartCondition = 75;
        int NStatement = 76;
        int COUT = 77;
        int ENDL = 78;
        int WS = 79;
        int NL = 80;
        int COMMENT = 81;
        int LSHIFT = 82;
        int Escape = 83;
        int DIGIT = 84;
        int NUMBER = 85;
        int CHAR_LITERAL = 86;
}

// end of source doe
```