DX – The programming language

Archana Mandape

Dec 18th 2007

Table of Contents

Introduction

Goal

The main goal of DX language is to implement and achieve a simple and ease-of-use programming language. The language syntax is very similar to C++ and Java.

The programmer of DX language can do simple numerical computations. Moreover, the language also provides functions for file read and file write. It also supports print to standard output.

DX can be used to generate XML files from a delimited text dumped from a database table. User can read an input delimited file and add XML records to the output XML file.

Portability

DX language will run on various platforms. For portability DX will use ANTLR as the syntax recognizer, which is running on a Java virtual machine (JVM). The current version of our language will be compiled and code generated would be a Java class file. With the help of the portability of Java programming language, DX can be virtually run on any machine

Ease-of-use

The language will be simple and quick-to-start. Only basic and necessary language features would be included in the language specification. Meanwhile, most of the needs in numerical computations would be fulfilled.

Language Tutorial

This section describes main features of the DX language.

1.4.1 Data types

Besides some basic data types like integer, character, string and Boolean, DX supports additional data type: file.

```
e.g.  int a = 10;
      string txt = “teststring”;
      file fp = “file1.txt”                //file type
```

1.4.2 Basic Numeric operations

Basic numeric operations like addition, subtraction, multiplication and division are supported in DX language.
e.g.:

```
int a = 10;
int b = 18;
int c = 0;

c = a + b;
c = a + b * 3;
c = c - 10;
```

1.4.4 Program flow control

The statements for program flow control were implemented, including if-then-else and while. In addition, programs in our DX language can define and invoke functions, recursive calls, and pass function as parameters.

```
e.g. :    if ( m > n )  {
                statements
          }

          WHILE ( m < n )
          {
                statements
          }
```

1.4.5 Internal functions

The minimum but necessary set of internal functions such as fgetline, fprint are supported in DX langauge. The print function can used to print data to standard output. Also, createRecord and addXMLRecord functions can help to create an xml file.

```
e.g. :          file fp = “file1.txt”
                string test = fgetline(fp);                 // gets line from file
                test = createRecord(test);                  //creates xml tag names
                test = addXMLrecord(fp, test);              // add xml records’
                string xmlStr  = “xmlrec1;xmlrec1;xmlrec2”;   // you can also
add strings different than the program
                test = addXMLrecord(fp, xmlStr )
```

First DX program (GCD Algorithm)

```
                int main()
                {
                        int a = 20;
                        int b = 25;
                        WHILE ( a <> b)
                        {
                                if (a > b)
                                {
                                        a = a - b;
                                }
                                else
                                {
                                        b = b - a;
                                }
                        }

                        string test ;
                        test = print(a);                        // when executed will
                        print 20
                        return;
                }
```

In GCD algorithm example the language features such as the while and if loop and the numerical computations are illustrated. You can also see that the inbuilt print function allows the user to print the result value.

XML file generation example

The second example explaining the features of DX language is of the XML file generation. Using the “file” data type of DX and the file read and print functionality of DX, the input delimited file is read. The DX language supports two inbuilt functions.

CreateRecord – takes in a delimited text string and creates the XML tags of the output XML.

Then the <ROOT> tag is printed to the output “File2.xml” file using the file data type.

The next lines in the input file are read using “fgetline” function and the string is passed to **addRecordXML** function

A EOF file check on the output of fgetline function is made using a while loop.

Once the file read is done, the ending </ROOT> tag is appended to the output XML file.
This file “File2.xml” is generated in the directory

DX Program :

```
int main()
{
        int m = 0;
        file fp1  = "File1.txt";
        file fp2  = "File2.xml";

        string strText;
        string test;
        string opt = "append";
        strText = fgetline(fp1);
        string delim = ";";
        strText = createRecord(strText, delim);

        test = "<ROOT>";
        test = fprint(fp2, opt,test );

        strText = fgetline(fp1);

        WHILE (strText <> EOF)
        {
                test = addXMLRecord(strText, fp1, delim);

                test = fprint(fp2, opt,test );

                strText = fgetline(fp1);

        }
        strText = "</ROOT>";
        strText = fprint(fp2, opt, strText);
        string printStr = "Please check the output file File2.xml";
        printStr = print(printStr);
        return;
}
```

The third example is of XML file generation with a if condition.
If a particular string is found in the input file, it is not created as an XML record. This example uses the defined function “find” to search for a string. It returns an integer 1 if the search string is found.

DX program:

```
int main()
{
      int m = 0;
      file fp1  = "File1.txt";
      file fp2  = "File2.xml";

      string strText;
      string test;
      string opt = "append";
      strText = fgetline(fp1);
      string delim = ";";
      strText = createRecord(strText, delim);



      test = "<ROOT>";
      test = fprint(fp2, opt,test );

      strText = fgetline(fp1);
      int check = 0;
      WHILE (strText <> EOF)
      {
            string checkStr = "archana";
            string emptyStr = "";
            check = find(strText, checkStr);

            // find return 1 if string archana found
            // ADD record if checkstring is NOT Archana
            if ( check <> 1 )
            {
                  test = addXMLRecord(strText, fp1, delim);
                  test = fprint(fp2, opt,test );
            }
            strText = fgetline(fp1);

      }

      strText = "</ROOT>";
      strText = fprint(fp2, opt, strText);
      string printStr = "Please check the output file File2.xml" ;
      printStr = print(printStr);
      return();
}
```

DX Language Reference Manual

1. Introduction

DX is a language which has features to create an xml file when given an input file with delimiters. The language is also useful for writing programs which requires programming constructs such as if statement, while loops etc.

2. **Lexical conventions**

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators. In general blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens.
At least one of these characters is required to separate otherwise adjacent identifiers, constants, and certain operator pairs. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

3 **Comments**
The characters // introduce a comment, which terminates with the new line.
// has
no special meaning inside comment line.

4. Identifiers (names)
An identifier is a sequence of letters and digits; the first character must be Alphabetic. The underscore ‘‘_’’ counts
as alphabetic. Upper and lower case letters are considered different.
ID : LETTER (LETTER | DIGIT | '_')* ;

5. **Keywords**
The following identifiers are reserved for use as keywords, and may not be used otherwise:
WHILE
addXMLRecord
createRecord
if
then
else
print
fgetline
fprint
EOF

6 **Constants**
There are several kinds of constants, as follows:

6.1 Integer constants
An integer constant is a sequence of digits.
integer : digit+

6.2. string constants
A string is a sequence of ACSII characters surrounded by double quote

7. Separators
The following ASCII characters are separators:
{
}
;
,
(
)
[
]

8. Variables

A variable is declared using the syntax *type variable_name*. The variable name must begin with at least one alphabetic, then any combination of alphabetic, integer number and /or underscore '_' is optional.
Variablestatement : typename ID (initialiser)?
It has an optional initialiser which would assign initial values to the var.
The variable types can be any of int, char, string, Boolean and file.

9. Expression
The precedence of expression operators is the same as the order of the major subsections of this section (highest precedence first). Within each subsection, the operators have the same precedence. Left- or right-associativity is specified in each subsection for the operators discussed therein.

9.1 Primary expressions

9.1.1. identifier
An identifier is a primary expression. Its type is specified by its declaration.

9.1.2. integer constant

An integer constant is a primary expression.

9.1.3. string constant
A string is a primary expression.

9.1.4. expression
A parenthesized *expression* is a primary expression whose type and value are identical to those of the unadorned expression

9.2 Operators
9.2.1 Multiplicative operators
The multiplicative operators *, /, and % group lefttoright.

9.2.1.1 expression * expression
The binary * operator indicates multiplication.

9.2.1.2 expression / expression
The binary / operator indicates division.

9.2.2 Additive operators
The additive operators + and - group lefttoright.

9.2.2.1 expression + expression
The result is the sum of the expressions.

9.2.2.2 expression - expression
The result is the difference of the operands

9.2.3 Relational operators
The relational operators group lefttoright,
but this fact is not very useful; ‘‘a<b<c’’ does not mean what it seems to.
9.2.3.1 expression < expression
9.2.3.2 expression > expression
9.2.3.3 expression <= expression
9.2.3.4 expression >= expression
The operators < (less than), > (greater than), <= (less than or equal to) and
>=
(greater than or equal to) all yield 0 if the specified relation is false and 1 if it is true.
Operand conversion is exactly the same as for the + operator

9.2.4 Equality operators
9.2.4.1 expression == expression
9.2.4.2 expression != expression
9.2.4.3 lvalue = expression
The value of the expression replaces that of the object referred to by the lvalue.

10 Statements
Except as indicated, statements are executed in sequence.

10.1 Expression statement
Most statements are expression statements, which have the form
expression ; Usually expression statements are assignments or function calls.

10.2 Conditional statement
The four forms of the conditional statement are

```
    if expression { statement }
    if expression { statement } else { statement }
```

In all cases, the expression is evaluated, and if it is non-zero, the first substatement is executed. In the second if expression is zero, then the substatement in the else is executed.

10.3 While statement
The while statement has the form
while expression { statement }
The substatement is executed repeatedly so long as the value of the expression remains
nonzero. The test takes place before each execution of the statement.

11 Scope rules
All components of the DX program must be part of one file and must be compiled at the same time. Variables can be global or local to a function. Therefore, there are two kinds of scopes to consider: first, the lexical scope of the variable; and second, the scope of the dependency.

11.1 Lexical scope
DX can be considered similar to a scripting language, and also uses block

structuring. Variables declared are visible within its scope.

11.2. Scope of dependency
Some built-in functions have some dependencies on the variables that need to be declared and used before its use. This is specifically for XML building functionality.
The input and output files need to be specified. The createRecord function needs to be called to define the tags of the XML.

Project Plan

1 Team Responsibilities

Archana Mandape – design, development, testing and documentation.

1.1 Software project environment

The program is developed in Java. The lexer and parser are written in Antlr, and will be translated to java code. The helper functions are written in Java.

1.2 Operating systems

Since this project is developed in pure Java, it can be used anywhere that a Java VM is running. I have developed it on Windows.

1.3 Java 1.4

Java is a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language".

1.4 Antlr

The language parser is written in Antlr, a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing C++ or Java actions."

Front end:

ANTLR used to design Lexer, Parser, and Walkers for both code generation and semantic checking

Back end:

Java 1.4 used for Java classes used in DX.

Architectural Design

The DX compiler consists of these components: a lexer that reads the user input or program files to tokens, a parser that analyzes the syntactic structure of the program and converts it to an abstract syntax tree, a tree walker that traverses the AST and generates code which is Java code. The java code as usual runs in JAVA runtime environment.

Figure 1 shows its block diagram of the DX compiler and the environment.

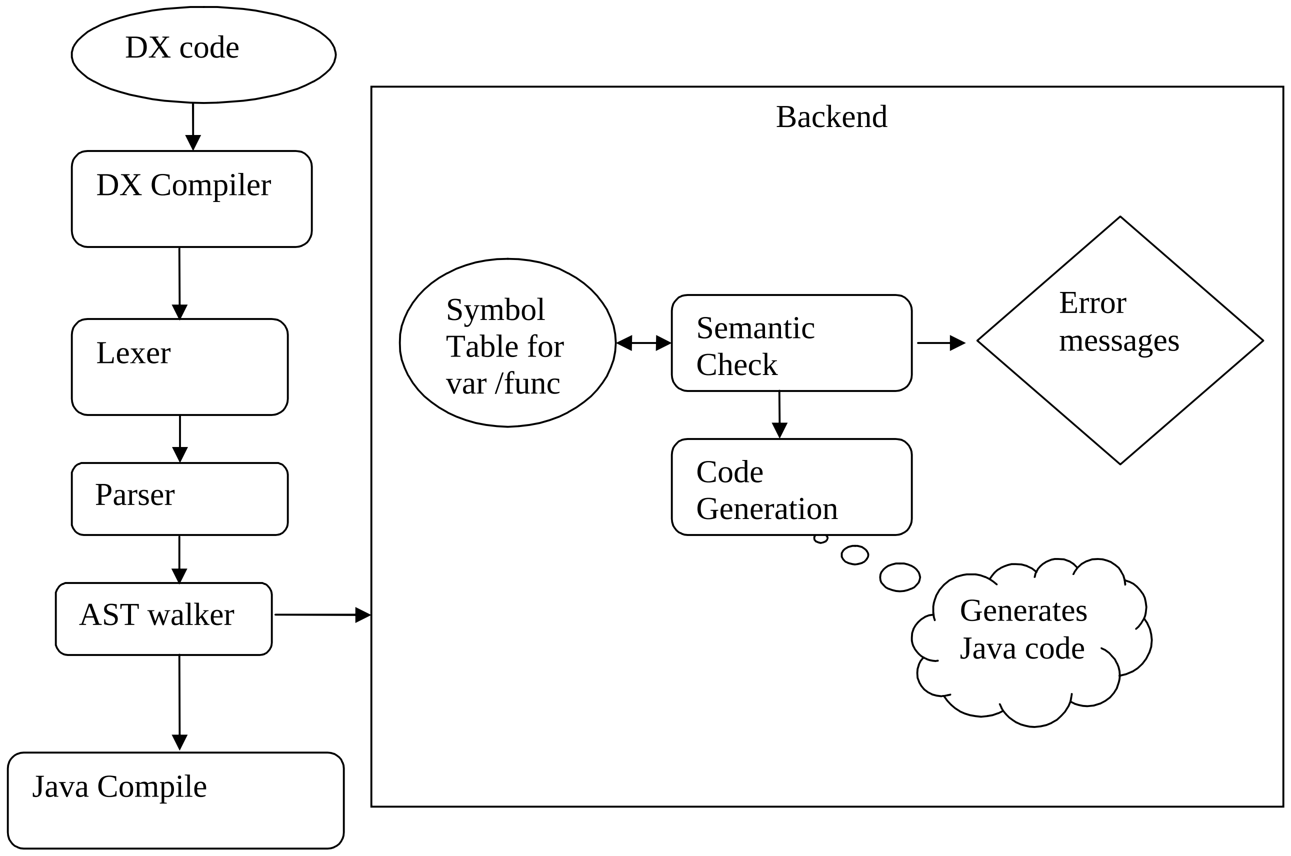

Fig 1: Architecture of DX Compiler

The Lexer, parser and treewalker are implemented by ANLTR
They are implemented in .g file. The main method is in Main class which takes the input file and then initializes the lexer, the parser and the walker.

The lexer is used for tokenization and the the result of the Lexer, a stream of tokens, is then passed into the Parser.

The Parser takes the input token stream and after matching nodes with the grammar of the language builds an AST (abstract syntax tree) for the input file. The parser returns any syntactic error.

The Semantic Checker

There are three tables used in this part of the compiler.
The global symbol table is used to store variables accessible by all parts of the program. The second symbol table is used for local scope.
These two symbol table contain the name and the type and a flag to indicate whether the variable has been initialized. An error is thrown if the variable is used before being initialized. One more table is used to store the function information in the program.

The information about the function such as name, its arguments and the return type is stored in this table.

The following checks are made during the compilation of the program

- Keywords are not allowed as variables or function names
- No redeclaration of variables.
- Variables cannot be used before being declared.
- Functions are called with same no of arguments.
- Variables are not used before being initialized
- Does not allow variables of different types to be assigned to each other
- Return statement expected and make sure that it returns the right type.

Code Generator

When the AST is being walked the code is generated at the same time. Java code is generated in a .java file that defines a java class. A wrapper java class file is also generated which uses the java class generated by the code generator. This wrapper class accesses the class file generated.
The generated Java code can then be compiled by the Java compiler which can executed in any Java runtime environment.

Test plan

The testing was done during phases and mostly done as unit testing.

Any changes done to the grammar or during tree walking were tested with different test cases.
Errors were introduced deliberately in the code to be compiled and tested to make sure the error messages were sent out during the syntax semantic analysis.

Messages were printed for debugging during the tree walking phrase.

Lessson learnt

It was a great learning experience while developing my own language and its compiler. I was able to visualize the many things involved in the development of other programming languages that we use everyday.
As regards the actual development take one step at a time. Start with a smaller basic syntax first and then add more functionality on top of it. I started going through the ANTLR syntax. The hardest part was documentation. I should have started early on the documentation

DLexer.g

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

class DXlexer extends Lexer;
options {
    testLiterals = false; // By default, don't check tokens against keywords
    k = 2;                // Need to decide when strings literals end
    charVocabulary = '\3'..'\377'; // Accept all eight-bit ASCII characters
    exportVocab=DX; // Call its vocabulary "DX"
}

tokens{

ADDRECORD = "addRecord";
//CREATERECORD = "createRecord";
READRECORD = "readRecord" ;
ROOTELEM ;


}
PLUS   : '+' ;
EQUAL  : '=' ;
MINUS  : '-' ;
TIMES  : '*' ;
DIV    : '/' ;
OR     : "||";
AND    : "&&";
LTE    : "<=";
NOT    : '!';
SEMI   : ';' ;
COMMA  : ',' ;
LPAREN : '{' ;
RPAREN : '}' ;
LBRACKET : '[' ;
RBRACKET : ']' ;
OPAREN          : '(' ;
CPAREN: ')' ;
GT        : '>' ;
GTE       : ">=" ;
EQUALS : "==" ;
NOT_EQUALS  :  "<>" ;
LT     : '<' ;



protected LETTER : ( 'a'..'z' | 'A'..'Z' ) ;
protected DIGIT  : '0'..'9' ;

ID
options {
    testLiterals = true;
}
    : LETTER (LETTER | DIGIT | '_')* ;

NUMBER : (DIGIT)+;
```

```
// Strings are "like this ""double quotes"" doubled to include them"
// Note that testLiterals are false so we don't have to worry about
// strings such as "if"
STRING :  '"' ( '"' '"'! | ~('"'))*  '"';


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

COMMENT
: ("//") (' '..'~' | '\t')* WS { $setType(Token.SKIP); };




class DXParser extends Parser;
options {

                buildAST = true;
                exportVocab=DX; // Call its vocabulary "DX"
                //defaultErrorHandler = true; // Don't generate parser error handlers
                k=3;
 }

tokens {
  STATEMENTS;PARAMLIST; ARGLIST;  DECLS;
TYPENAME;VAR_DECLARATION;VARLIST;FUNCDECL;ARG;ARGS;RETTYPE;BODY;IF;WHILE;RETURN;
FUNCCALL;SIGNED;
 }

startRule
: (decl)*
;

decl
:((typeName|"void") ID OPAREN) =>funcDecl
| variableDeclaration
;

funcDecl!
: (t:typeName|v:"void")
                i:ID OPAREN! a:args CPAREN! LPAREN! b:body RPAREN!
{ #funcDecl = #(#[FUNCDECL, "PROC_DECL"], t, v, i, a, b);}
;

decls : (decl)* { #decls = #([DECLS, "DECLS"], #decls); } ;


args
 : (arg (COMMA! arg)*)?
 ;

arg!
 : t:typeName i:ID a:(LBRACKET expr  RBRACKET)*
 {#arg = #(#[ARG,"ARG"],t, i, a);}
;
```

```
body!
: (b: babyBody
{#body = #(#[BODY,"BODY"], b);})?
;

babyBody :
(variableDeclaration | statement)+
;

variableDeclaration!
:t:typeName v:variabeList SEMI!
{#variableDeclaration = #(#[VAR_DECLARATION,"VAR_DECL"],t,v);}
;

variabeList
:
variabeInit(COMMA! variabeInit)*
;
variabeInit
: ID (LBRACKET expr  RBRACKET)* (EQUAL^ (expr|STRING))?
;

typeName
: typeNames
;
typeNames
: "int"
| "string"
| "boolean"
| "char"
| "float"
| "file"
;

//array:
//(LBRACKET! expr  RBRACKET!)*
//{#array =#([ARRAY,"array"], array); }
//;

statement  :  r:returnStatement
        {#statement = #(#[RETURN,"RETURN"],r);}
        | i:ifStatement
        {#statement = #(#[IF, "IF"], i);}
        | w:while_stmt
        {#statement = #(#[WHILE,"WHILE"],w);}
        //| printStatement
        | a:assignmentStatement
        {#statement = #(#[ASSIGN,"ASSIGN"],a);}
        | f: funccall SEMI!
        {#statement = #(#[FUNCCALL,"FUNC_CALL"],f);}
        | createFunc
        | addRecFunc
        | assignmentState
        {#statement = #(#[ASSIGN,"ASSIGN"],a);}

        ;


builtinfunc :        createFunc
          |          addRecFunc
          |   readRecFunc
          ;

createFunc :    CREATERECORD^ "(" (STRING| expr )")";

addRecFunc  :        ADDRECORD^ "(" (STRING| expr )")";

readRecFunc :        READRECORD^ "(" (STRING| expr )? ")" ;

stmt_list
        : ( LPAREN (stmt_single | ifStatement | while_stmt)+ RPAREN)
        | stmt_single;

stmt_single
        :
         assignmentStatement
        | assignmentState
        | builtinfunc
        | SEMI;



initialiser:
        ASSIGN expr
        ;

variableReference
  : ID
    ( LBRACKET expr RBRACKET
    | DOT ID
    )*
  ;

assignmentStatement
  : variableReference EQUAL^ (expr|STRING)  SEMI
  ;
paramlist :
        (parameter (COMMA! parameter)*)?
        { #paramlist = #([PARAMLIST, "paramlist"], #paramlist); }
        ;

parameter :
        ID;

funccall:
  ID OPAREN! (arglist)? CPAREN! ;

arglist:
(ID) (COMMA! (ID))*
{ #arglist = #([ARGLIST, "arglist"], #arglist); }
;

exitStatement
  : "exit" "when" expr
  ;

ifStatement :
        "if"! OPAREN! expr  CPAREN! (LPAREN! body RPAREN!)(("else") => "else"! (LPAREN! body
RPAREN!) | /*empty*/ )
;

while_stmt:
        ("WHILE"! OPAREN ! expr CPAREN!
                (LPAREN! body RPAREN))
   ;

 endStatement
   : "end" SEMI
   ;

returnStatement
  : "return" OPAREN! ( expr | STRING )? CPAREN!   SEMI !
  ;

assignmentState
                :  (accessval | "ROOTELEM" | "DELIM" | "RECNAME" | "RECATTRIB" | "RECVAL" | "INPUTFILE" |
"XMLFILE" ) EQUAL^ (expr| STRING |builtinfunc) SEMI ;

eof
: "EOF";

expr
    : ID EQUAL^ expr0
    | expr0

    ;

expr0 : expr1 ((EQUALS^ |NOT_EQUALS^ |GT^ |GTE^ |LT^ |LTE^| OR^ | AND^ ) expr1)* ;

expr1 : expr2 ( (PLUS^ | MINUS^) expr2 )* ;

expr2 : expr3 ( (TIMES^ | DIV^) expr3 )* ;

expr3
    : ID
    | "("! expr ")"!
    | NOT ID
    | NUMBER
    | MINUS^ expr3
    | keywords
    | eof
    | f:funccall
{#expr3 = #(#[FUNCCALL, "FUNC_CALL"], f);}
    ;

keywords :  accessval|"ROOTELEM" | "DELIM" | "RECNAME" | "RECATTRIB" | "RECVAL" | "INPUTFILE" |
"XMLFILE" ;

accessval : "RECVAL"^ LBRACKET NUMBER RBRACKET;




class DXWalker extends TreeParser;
options {
                importVocab = DX;
}
{

// maintain variable and function declarations using Hashtables

HashforFunction fh = new HashforFunction();
HashforFunction dxh = new HashforFunction();
Hashtable global = new java.util.Hashtable();
Hashtable local = new java.util.Hashtable();
Hashtable keywords = new java.util.Hashtable();
Hashtable functions = new java.util.Hashtable();   // for functions in the program
Hashtable dxhash = new Hashtable();                                    // for user defined functions



Vector DXVector = new Vector();
int comArgs = 0;
boolean mainFound = false;


boolean checkReturn = false;
boolean misMatch = false;
boolean iminfunccall = false;

CodeWriter output = new CodeWriter();
Stack varStack = new Stack();

boolean GenerateCode = true;
boolean boolargs = false;

String filename = "DX";
}


startRule :
{
        //userdefined functions - dxhash table

        DXVector.add("string");
        DXVector.add("string");
        DXVector.add("file");
        DXVector.add("string");
        dxhash = dxh.putFunction(dxhash, "addXMLRecord", DXVector);

        DXVector = new Vector();
        DXVector.add("string");
        DXVector.add("string");
        DXVector.add("string");
        dxhash = dxh.putFunction(dxhash, "createRecord", DXVector);

        DXVector = new Vector();
        DXVector.add("string");
        DXVector.add("file");
        dxhash = dxh.putFunction(dxhash, "fgetline", DXVector);

        DXVector = new Vector();
        DXVector.add("string");
        DXVector.add("file");
        DXVector.add("string");
        DXVector.add("string");
        dxhash = dxh.putFunction(dxhash, "fprint", DXVector);

        DXVector = new Vector();
        DXVector.add("string");
        DXVector.add("string");
        dxhash = dxh.putFunction(dxhash, "print", DXVector);

        DXVector = new Vector();
        DXVector.add("int");
        DXVector.add("string");
        DXVector.add("string");
```

```
		dxhash = dxh.putFunction(dxhash, "find", DXVector);

		DXVector = new Vector();
		DXVector.add("string");
		dxhash = dxh.putFunction(dxhash, "getline", DXVector);

		output.newWriter(filename + "MainFile.java");

		keywords.put("addXMLRecord", "addXMLRecord");
		keywords.put("createRecord", "createRecord");
		keywords.put("fgetline", "fgetline") ;
		keywords.put("fprint", "fprint");
		keywords.put("getline", "getline");
		keywords.put("print", "print");
		keywords.put("find", "find");
		keywords.put("boolean", "boolean");
		keywords.put("cd", "cd");
		keywords.put("char", "char");
		keywords.put("else", "else");
		keywords.put("false", "false");
		keywords.put("file", "file");
		keywords.put("for", "for");
		keywords.put("if", "if");
		keywords.put("int", "int");
		keywords.put("ls", "ls");
		keywords.put("mkdir", "mkdir");
		keywords.put("overwrite", "overwrite");
		keywords.put("put", "put");
		keywords.put("return", "return");
		keywords.put("rm", "rm");
		keywords.put("rmdir", "rmdir");
		keywords.put("string", "string");
		keywords.put("true", "true");
		keywords.put("void", "void");
		keywords.put("while", "while");

		//java's keywords

		if(GenerateCode)
		{
			//open your files
			//output.WriteCode("import DX_File.*;\n");
			output.WriteCode("public class " + filename + "MainFile { \n");
			output.WriteCode("DX_Funcs DXfuncs = new DX_Funcs(); \n");
		}
	}
	(variableDeclaration[global]|funcDecl)*
	{
		if(mainFound == false)
		System.err.println("\n***You have not declared a main function***");
		if(GenerateCode)
		{
			output.WriteCode("}\n"); //Close our MainMethod class
			output.closeWriter();
			output.newWriter(filename + ".java");
			output.WriteCode("public class " + filename + "{\n");
			output.WriteCode("public static void main(String args[]){\n");

			if(comArgs == 0)
			{
				output.WriteCode(filename + "MainFile m = new " + filename + "MainFile();\n");
				output.WriteCode("}\n");
```

```
                }
                else{
                        output.WriteCode(filename + "MainFile m = new " + filename + "MainFile(args[0]");

                        for (int i = 1; i < comArgs; i++){
                                output.WriteCode(", args[" + i + "]");
                        }
                        output.WriteCode(");\n");
                        output.WriteCode("}\n");

                }
                output.WriteCode("}");
                output.closeWriter();
        }
}
;


arg
:#(argRoot:ARG
{
        AST leftChild = argRoot.getFirstChild();
        output.WriteCode(leftChild.getText() + " ");
        leftChild = leftChild.getNextSibling();
        output.WriteCode(leftChild.getText());
}
)
;
variableDeclaration[Hashtable variableHash]
{String name;Vector v = new Vector(); boolean init = false; int count = 0;}
: #(vardeclroot:VAR_DECLARATION
{
                AST type = vardeclroot.getFirstChild();
                AST rightChildren = type.getNextSibling();

                if(GenerateCode)
                {
                        if((type.getText()).compareTo("file")==0)
                                output.WriteCode("DX_File ");
                        else
                        {
                                if((type.getText()).compareTo("string")==0)
                                        output.WriteCode("String"  + " ");
                                else
                                        output.WriteCode(type.getText() + " ");
                        }


                }

                while(rightChildren != null)
                {
                        if(count > 0)
                        {
                        { if(GenerateCode)
                                if (rightChildren.getText().compareTo("[")==0  ||
rightChildren.getText().compareTo("]")==0  )
                                {
                                }
                                else
                                if (rightChildren.getNextSibling() != null)
                                {
                                        if (rightChildren.getNextSibling().getText().compareTo("]")==0 )
                                        {
                                        }
                                        else
                                                output.WriteCode(", ");
                                }
                                else
                                        output.WriteCode(", ");
                        }
                        }
                        if (((rightChildren.getText()).compareTo("="))==0)
                        {

                                name = (rightChildren.getFirstChild()).getText();
                                init = true;
                                if(GenerateCode)
                                {
                                                output.WriteCode(name + " = ");
                                        if((type.getText()).compareTo("file")==0)
                                                output.WriteCode(" new DX_File(");
                                }
                                if((type.getText()).compareTo("file")==0)
                                        expr((rightChildren.getFirstChild()).getNextSibling(), "string" );
                                else
                                        expr((rightChildren.getFirstChild()).getNextSibling(),type.getText());
                                if((type.getText()).compareTo("file")==0)
                                        output.WriteCode(" ) ");
                        }
                        else
                        {
                                name = rightChildren.getText();
                                if(GenerateCode)
                                        output.WriteCode(name);
                        }
                        if(keywords.containsKey(name))
                                System.err.println("\nvariable [" + name +"]: is a keyword");
                        else if(!variableHash.containsKey(name))
                        {
                                v.add(type.getText());
                                Boolean tb = new Boolean(init);
                                v.add(tb);
                                variableHash.put(name, v);
                        }
                        else
                                System.err.println("\nvariable [" + name + " ] already declared");
                        count++;
                        rightChildren = rightChildren.getNextSibling();
                }
                if(GenerateCode)
                        output.WriteCode(";\n");
        }
)
;

funcDecl
{Vector v = new Vector(); }
:#(func:FUNCDECL
{
                AST typeNode = func.getFirstChild();
                String type = new String(typeNode.getText());
                local.put("return", new String(type));
                v.add(type);
```

```
        AST funcNode = typeNode.getNextSibling();
        String key = new String(funcNode.getText());
        if(key.equals("main"))
        {
                mainFound = true;
        }
/***********************************************/
if(GenerateCode)
{
        if(key.equals("main"))
        {
                output.WriteCode("public " + filename + "MainFile(");
        }
        else
        {
                if(type.equals("string"))
                        output.WriteCode("String " + key + "(");
                else if(type.equals("file"))
                        output.WriteCode("DX_File " + key + "(");
                else
                        output.WriteCode(type + " " + key + "(");
        }
}
/***********************************************/
if(keywords.containsKey(key))
{
        System.err.println("\n[" + key + "]: is a reserved word and cannot be used as a function name");
}

AST funcStuff = funcNode.getNextSibling();
/*vector stores the args of the funcDecl---*/
if(funcStuff != null && (funcStuff.getText()).equals("ARG"))
{
        String argType = (funcStuff.getFirstChild()).getText();
        String argName = ((funcStuff.getFirstChild()).getNextSibling()).getText();
        Vector varv = new Vector();

        varv.add(argType);
        varv.add("false");
        local.put(argName, varv);
        v.add(argType);
        if(GenerateCode)
        {
                        if(argType.equals("string"))
                                output.WriteCode("String " + argName);
                        else if(argType.equals("file"))
                                output.WriteCode("DX_File " + argName);
                        else
                                output.WriteCode(argType + " " + argName);
        }
        funcStuff = funcStuff.getNextSibling();
        while(funcStuff != null && (((funcStuff.getText()).compareTo("ARG"))==0))
        {
                argType = (funcStuff.getFirstChild()).getText();
                v.add(argType);
                argName = ((funcStuff.getFirstChild()).getNextSibling()).getText();

                varv.clear();
                varv.add(argType);
                varv.add("false");
                local.put(argName, varv);
                funcStuff = funcStuff.getNextSibling();
                if(GenerateCode)
                {
                        output.WriteCode(",");
                        if(argType.equals("string"))
                                output.WriteCode("String " + argName);
                        else if(argType.equals("file"))
                                output.WriteCode("DX_File " + argName);
                        else
                                output.WriteCode(argType + " " + argName);
                }
        }
}
output.WriteCode(")");
output.WriteCode("{");


if(key.equals("main"))
        comArgs = v.size() - 1;
if(functions.containsKey(key))
{
        LinkedList declareFunc = new LinkedList();
        declareFunc = (LinkedList)functions.get(key);
        Iterator iter = declareFunc.iterator();
        boolean same = true;
        int i = 1;
        while (iter.hasNext())
        {
                Vector checkv = (Vector)iter.next();
                for (i=1; i < (checkv.size()-1); i++)
                {
                        if(v.elementAt(i) != null)
                        {
                                if(!(((String)v.elementAt(i)).equals((String)(checkv.elementAt(i)))))
                                {
                                        same = false;
                                }
                        }
                }
                if( ((same == true) && (v.size() > checkv.size())) || (v.size() < checkv.size()) )
                        same = false;
        }
        if((same == true))
                System.err.println("function [" + key + "]:function is already declared!");
}
functions = fh.putFunction(functions, key,v);
if(GenerateCode)
{
}
else
{
        output.WriteCode(") { \n");
}
if (funcStuff != null)
{
        body(funcStuff);
        output.closeWriter();
        output.newWriter(filename + "MainFile.java");
}
if(GenerateCode)
{
        output.WriteCode("} \n");
```

```
		}

		if(!((String)(local.get("return"))).equals("void")   && checkReturn != true )
			System.err.println("function [" + key + "]: A return statement expected");
		if(((String)(local.get("return"))).equals("void") && checkReturn == true)
			System.err.println("function [" + key + "]: This function returns void - unwanted return statement
");
		local.clear();
		checkReturn = false;
	}
	)
	;

	expr[String mytype] {String ourType = "";} :ID
	{
		if(GenerateCode)
		{
			if (boolargs == false )
				output.WriteCode(#ID.getText());
		}

		if(!global.containsKey(#ID.getText()) && !local.containsKey(#ID.getText()) || mytype.compareTo("")==0)
		{

			if(iminfunccall == true)
				misMatch = true;
			else
				System.err.println("\nvariable [" + #ID.getText() + "]: is not declared/initialized");
		}
		else
		{

			if (local.containsKey(#ID.getText()))
			{
				Vector v = (Vector)local.get(#ID.getText());

				if(v.elementAt(1).equals(new Boolean(true)))
				{
					ourType = (String)v.elementAt(0);
					if (!(ourType.equals(mytype)))
					{
						if(iminfunccall == true)
							misMatch = true;
						else
							System.err.println("variable [" + #ID.getText() + "]:is " +
	ourType +" but should be " + mytype);
					}
				}
				else
				{
					if(iminfunccall == true)
						misMatch = true;
					else
						System.err.println("variable [" + #ID.getText() + "]: is not
	initialised");
					}
				}
				else
				{
					Vector v = (Vector)global.get(#ID.getText());
					if( v.elementAt(1).equals(new Boolean(true)))
					{
```

```
                                                ourType = (String)v.elementAt(0);
                                                if(!(ourType.equals(mytype)))
                                                {
                                                        if(iminfunccall == true)
                                                                misMatch = true;
                                                        else
                                                                System.err.println("variable [" + #ID.getText() +
"]:is " + ourType +" but should be " + mytype);
                                                }
                                        }
                                        else
                                        {
                                                if(iminfunccall == true)
                                                        misMatch = true;
                                                else
                                                        System.err.println("variable [" + #ID.getText() + "]: you
haven't initialized this guy globally");
                                        }
                                }
                        }
}
| NUMBER
{
        if(!(mytype.equals("int")))
        {
                if(iminfunccall == true)
                        misMatch = true;
                else
                        System.err.println("[]: expected type " + mytype + " got " + #NUMBER.getText());
        }

        if(GenerateCode)
                        output.WriteCode(#NUMBER.getText());

        }
| STRING
{
        if(!(mytype.equals("string")))
        {
                if(iminfunccall == true)
                        misMatch = true;
                else
                        System.err.println("[]: expected type" + mytype + " got " + #STRING.getText());
        }

        if(GenerateCode)
        {
                output.WriteCode(#STRING.getText());
        }

}
| CHAR
{
        if(!(mytype.equals("char")))
        {
                if(iminfunccall == true)
                        misMatch = true;
                else
                        System.err.println("[]: expected " + mytype + " got " + #CHAR.getText());
        }

        if(GenerateCode)
```

```
				output.WriteCode(#CHAR.getText());

			}
		| FILE
		{
			if(!(mytype.equals("file")))
			{
				if(iminfunccall == true)
					misMatch = true;
				else
					System.err.println("[]: expected " + mytype + " got " + #FILE.getText());
			}

			if(GenerateCode)
				output.WriteCode(#FILE.getText());

		}
		| "true"
		{
			if(!(mytype.equals("boolean")))
			{
				if(iminfunccall == true)
					misMatch = true;
				else
					System.err.println("[True]: expected " + mytype + " got true");
			}

			if(GenerateCode)
				output.WriteCode("true");

		}
		| "false"
		{
			if(!(mytype.equals("boolean")))
			{
				if(iminfunccall == true)
					misMatch = true;
				else
					System.err.println("[False]: expected " + mytype + " got false");
			}

			if(GenerateCode)
				output.WriteCode("false");

		}
		| #(fcallroot:FUNCCALL
		{
			iminfunccall = true;
			AST leftChild = fcallroot.getFirstChild();
			String key = new String(leftChild.getText());
			Vector v = new Vector();
			misMatch = false;
			String rettype = "";

			if(functions.containsKey(key) || dxhash.containsKey(key))
			{

				LinkedList declareFunc = new LinkedList();
				if(functions.containsKey(key))
					declareFunc = (LinkedList)functions.get(key);
				else
				{
```

```
                declareFunc = (LinkedList)dxhash.get(key);
        }
        Iterator iter = declareFunc.iterator();
        int i = 1;
        boolean foundMatch = false;
        AST nextArg = leftChild.getNextSibling().getFirstChild();

        while(nextArg != null)
        {
                v.add(nextArg);
                nextArg = (AST) nextArg.getNextSibling();
        }

        while (iter.hasNext() && foundMatch == false)
        {
                Vector checkv = (Vector)iter.next();
                String thistype = (String)checkv.elementAt(0);

                if( mytype.equals(thistype) || mytype.equals("") )
                {
                        for (i=1; i < (checkv.size()-1); i++)
                        {
                                if(v.elementAt(i-1) != null)
                                {
                                        boolargs = true;
                                        expr((AST)v.elementAt(i-1),(String)checkv.elementAt(i));
                                        boolargs = false;
                                }
                        }
                        if( (misMatch == false) && (v.size() == checkv.size()-1))
                        {
                                foundMatch = true;
                                rettype = (String)checkv.elementAt(0);
                        }
                }
        }

        if(foundMatch == false)
        {
                System.out.println("function[" + key + "()]: not declared.");
        }
}
else
{
        System.out.println("function [" + key + "]: not declared");
}
if(GenerateCode)
{
        if(dxhash.containsKey(key))
        {
                output.WriteCode("DXfuncs.");
        }
        output.WriteCode(key + "(");
        if(!v.isEmpty())
        {
                output.WriteCode((String)v.elementAt(0).toString());
                for(int i = 1; i < v.size(); i++)
                {
                        output.WriteCode(", " + (String)v.elementAt(i).toString());
                }
        }
        output.WriteCode(");\n");
```

```
		}

		/**************************************************/
		//clear flags
		misMatch = false;
		iminfunccall = false;
	}
	)
	//*****************************************************************************************
	| #(eqroot:EQUAL
	{

		AST leftChild = eqroot.getFirstChild();
		if(local.containsKey(leftChild.getText()))
		{
			Vector v = (Vector)local.get(leftChild.getText());
			v.setElementAt(new Boolean(true), 1);
		}
		else if(global.containsKey(leftChild.getText()))
		{
			Vector v = (Vector)global.get(leftChild.getText());
			v.setElementAt(new Boolean(true), 1);
		}
		expr(leftChild, mytype);


		AST rightChildren = leftChild.getNextSibling();
		if (rightChildren.getText().compareTo("[") == 0 )
		{
			if(GenerateCode)
			{		output.WriteCode(rightChildren.getText());
			}
			rightChildren = rightChildren.getNextSibling();
			while(rightChildren != null)
			{
				if (rightChildren.getText().compareTo("]") == 0 )
				{
					output.WriteCode(rightChildren.getText());
					if(GenerateCode)
					{
						if(rightChildren.getNextSibling().getText().compareTo("[") != 0 )
						{
							output.WriteCode(" = ");
						}
						if(mytype.compareTo("file")==0)
								output.WriteCode("new DX_File(");
					}
					expr(leftChild.getNextSibling(), mytype);
				}
				else
				{
					output.WriteCode(rightChildren.getText());

				}
				rightChildren = rightChildren.getNextSibling();
			}
		}
		else
		{
			if(GenerateCode)
			{
				output.WriteCode(" = ");
```

```
						if(mytype.compareTo("file")==0)
							output.WriteCode("new DX_File(");
				}
		}
		expr(leftChild.getNextSibling(), mytype);
}
)
| #(eofroot:"EOF"
{

		output.WriteCode(" null ");
}
)

| #(plusroot:PLUS
{
		AST leftChild = plusroot.getFirstChild();
		if(mytype.compareTo("int") != 0)
				System.err.println("\n[?] This expression returns an int, but you're trying to assign it to a " +
mytype);

		if(GenerateCode)
				output.WriteCode("(");

		expr(leftChild, "int");

		if(GenerateCode)
				output.WriteCode(" + ");

		expr(leftChild.getNextSibling(), "int");

		if(GenerateCode)
				output.WriteCode(")");

}
)
| #(timesroot:TIMES
{
		AST leftChild = timesroot.getFirstChild();
		if(mytype.compareTo("int") != 0)
				System.err.println("\n[?] This expression returns an int, but you're trying to assign it to a " +
mytype);

		if(GenerateCode)
				output.WriteCode("(");

		expr(leftChild, "int");

		if(GenerateCode)
				output.WriteCode(" * ");

		expr(leftChild.getNextSibling(), "int");

		if(GenerateCode)
				output.WriteCode(")");

}
)
| #(neqroot:NOT_EQUALS
{
		AST leftChild = neqroot.getFirstChild();
```

```
        if (local.containsKey(leftChild.getText()))
        {
                Vector v = (Vector)local.get(leftChild.getText());
                mytype = (String)v.elementAt(0);
        }
        else if(global.containsKey(leftChild.getText()))
        {
                Vector v = (Vector)global.get(leftChild.getText());
                mytype = (String)v.elementAt(0);
        }
        else
        mytype = "undeclared variable";

        if(GenerateCode)
                output.WriteCode("(");
        expr(leftChild, mytype);                            //TBD was int

        if(GenerateCode)
                output.WriteCode(" != ");

        expr(leftChild.getNextSibling(), mytype);           //TBD was int
        if(GenerateCode)
                output.WriteCode(")");

}
)
| #(divroot:DIV
{
        AST leftChild = divroot.getFirstChild();
        if(mytype.compareTo("int") != 0)
                System.err.println("\n[?] This expression returns an int, but you're trying to assign it to a " +
mytype);

        if(GenerateCode)
                output.WriteCode("(");

        expr(leftChild, "int");

        if(GenerateCode)
                output.WriteCode(" / ");

        expr(leftChild.getNextSibling(), "int");

        if(GenerateCode)
                output.WriteCode(")");

}
)
| #(orroot:OR
{
        AST leftChild = orroot.getFirstChild();
        System.out.println("leftchild" + leftChild.getNextSibling().getText());
        if(!mytype.equals("boolean"))
                System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);

        if(GenerateCode)
                output.WriteCode("(");

        expr(leftChild, "boolean");
```

```
		if(GenerateCode)
			output.WriteCode(" || ");

		expr(leftChild.getNextSibling(), "boolean");

		if(GenerateCode)
			output.WriteCode(")");
	}
	)
	| #(androot:AND
	{
		AST leftChild = androot.getFirstChild();
		if(!mytype.equals("boolean"))
			System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);

		if(GenerateCode)
			output.WriteCode("(");

		expr(leftChild, "boolean");

		if(GenerateCode)
			output.WriteCode(" && ");

		expr(leftChild.getNextSibling(), "boolean");

		if(GenerateCode)
			output.WriteCode(")");
	}
	)
	| #(gtroot:GT
	{
		AST leftChild = gtroot.getFirstChild();
		if(!mytype.equals("boolean"))
			System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);

		if(GenerateCode)
			output.WriteCode("(");

		expr(leftChild, "int");

		if(GenerateCode)
			output.WriteCode(" > ");

		expr(leftChild.getNextSibling(), "int");

		if(GenerateCode)
			output.WriteCode(")");

	}
	)
	| #(gteroot:GTE
	{
		AST leftChild = gteroot.getFirstChild();
		if(!mytype.equals("boolean"))
			System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);

		if(GenerateCode)
			output.WriteCode("(");
		expr(leftChild, "int");

		if(GenerateCode)
			output.WriteCode(" >= ");

		expr(leftChild.getNextSibling(), "int");

		if(GenerateCode)
			output.WriteCode(")");

	}
	)
	| #(ltroot:LT
	{
		AST leftChild = ltroot.getFirstChild();
		if(!mytype.equals("boolean"))
			System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);

		if(GenerateCode)
			output.WriteCode("(");

		expr(leftChild, "int");

		if(GenerateCode)
			output.WriteCode(" < ");

		expr(leftChild.getNextSibling(), "int");

		if(GenerateCode)
			output.WriteCode(")");

	}
	)
	| #(lteroot:LTE
	{
		AST leftChild = lteroot.getFirstChild();
		if(!mytype.equals("boolean"))
			System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);


		if(GenerateCode)
			output.WriteCode("(");

		expr(leftChild, "int");

		if(GenerateCode)
			output.WriteCode(" <= ");

		expr(leftChild.getNextSibling(), "int");

		if(GenerateCode)
			output.WriteCode(")");

	}
	)
	| #(notroot:NOT
	{
		AST leftChild = notroot.getNextSibling();
		if(!mytype.equals("boolean"))
			System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);

		if(GenerateCode)
			output.WriteCode(" !(");

		expr(leftChild, "boolean");

		if(GenerateCode)
			output.WriteCode(")");

}
)
| #(eqsroot:EQUALS
{
		AST leftChild = eqsroot.getFirstChild();
		if(!mytype.equals("boolean"))
			System.err.println("\n[?] This expression returns a boolean,but you're trying to assign it to a " +
mytype);

		int typeNum = leftChild.getType();
		String typeName = leftChild.getText();
		if(typeNum >= 49 && typeNum <= 52)
			mytype = "int";
		else if((typeNum >= 41 && typeNum <= 48) || typeNum == 53)
			mytype = "boolean";
		else if(typeNum == 19)
		{
			if (local.containsKey(typeName))
			{
				Vector v = (Vector)local.get(typeName);
				mytype = (String)v.elementAt(0);
			}
			else if(global.containsKey(typeName))
			{
				Vector v = (Vector)global.get(typeName);
				mytype = (String)v.elementAt(0);
			}
			else
				mytype = "undeclared variable";
		}
		else if(typeNum == 32) mytype = "int";
		else if(typeNum == 33) mytype = "string";
		else if(typeNum == 31) mytype = "char";
		else if(typeName.compareTo("true")==0 || typeName.compareTo("false")==0) mytype = "boolean";

		expr(leftChild, mytype);
		if(GenerateCode)
		{
			if(mytype.compareTo("file")==0 || mytype.compareTo("string")==0)
				output.WriteCode(".equals( ");
			else
				output.WriteCode(" == ");
		}

		expr(leftChild.getNextSibling(), mytype);

		if(GenerateCode)
		{
			if(mytype.compareTo("file")==0 || mytype.compareTo("string")==0)
				output.WriteCode(")");
		}
}
)
| #(minusroot:MINUS
{
		if (((minusroot.getFirstChild()).getNextSibling()) != null)
		{
				AST leftChild = minusroot.getFirstChild();
				if(mytype.compareTo("int") != 0)
						System.err.println("\n[?] This expression returns an int, but you're trying to assign it to a 
" + mytype);

				if(GenerateCode)
						output.WriteCode("(");

				expr(leftChild, "int");

				if(GenerateCode)
						output.WriteCode(" - ");

				expr(leftChild.getNextSibling(), "int");

				if(GenerateCode)
						output.WriteCode(")");
		}
		else
		{

				if(GenerateCode)
						output.WriteCode(" -(");

				AST leftChild = minusroot.getFirstChild();
				expr(leftChild, "int");

				if(GenerateCode)
						output.WriteCode(")");

		}
}
)
;


statement {String ourType = "";}
: #(ifRoot:IF
{
		if(GenerateCode)
				output.WriteCode("if( ");

				AST leftChild = ifRoot.getFirstChild();
				expr(leftChild, "boolean");
				varStack.push(local.clone());

				if(GenerateCode)
						output.WriteCode(")\n{\n\t");

				body(leftChild.getNextSibling());

				if(GenerateCode)
						output.WriteCode("}\n");

				local = (Hashtable)varStack.pop();
```

```
                    AST rightChild = leftChild.getNextSibling();
                    //Check for an else block
                    if(rightChild.getNextSibling() != null)
                    {

                              if(GenerateCode)
                                        output.WriteCode("else \n{\n");

                              varStack.push(local.clone());
                              body(rightChild.getNextSibling());
                              local = (Hashtable)varStack.pop();

                              if(GenerateCode)
                                        output.WriteCode("}\n");

                    }
}
)
|#(whileRoot:WHILE
{
          if(GenerateCode)
                    output.WriteCode("while( ");

          AST leftChild = whileRoot.getFirstChild();
          expr(leftChild, "boolean");                              //Changed now TBD was boolean
          varStack.push(local.clone());

          if(GenerateCode)
                    output.WriteCode(")\n{\n");

          body(leftChild.getNextSibling());

          if(GenerateCode)
                    output.WriteCode("}\n");

          local = (Hashtable)varStack.pop();
}
)
|#(assignRoot:ASSIGN
{
          AST firstChild = (assignRoot.getFirstChild()).getFirstChild();
          if(global.containsKey(firstChild.getText()) ||local.containsKey(firstChild.getText()))
          {
                    if (local.containsKey(firstChild.getText()))
                    {
                              Vector v = (Vector)local.get(firstChild.getText());
                              ourType = (String)v.elementAt(0);
                    }
                    else
                    {
                              Vector v = (Vector)global.get(firstChild.getText());
                              ourType = (String)v.elementAt(0);
                    }
          }
          expr(assignRoot.getFirstChild(), ourType);

          if(GenerateCode)
                    output.WriteCode(";\n");

}
)
|#(returnRoot:RETURN
{
        if(GenerateCode)
                output.WriteCode("return");

        checkReturn = true;
        ourType = (String)local.get("return");
        if (returnRoot.getFirstChild() != null )
        {
                AST leftChild = returnRoot.getFirstChild();
                if ( leftChild.getNextSibling() != null )
                {
                        expr(leftChild.getNextSibling(), ourType);
                }
        }
        if(GenerateCode)
                output.WriteCode(";\n");
}
)
;
body
:#(BODY ( variableDeclaration[local] | statement)*)
;
```

Main.java

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class Main {
  public static void main(String args[]) {

   if(args.length != 1)
   {
   System .err.println ("USAGE: java Main <fileName>.dx" );
   System .exit(0);
   }
   try{
   BufferedReader r = new BufferedReader (new FileReader (args[0]));
   DXlexer lexer = new DXlexer (r);
   lexer.setFilename (args[0]);
   DXParser parser = new DXParser (lexer);
   parser.startRule ();
   CommonAST t = (CommonAST ) parser .getAST ();
   System.out.println (t.toStringList ());

   //ASTFrame frame = new ASTFrame("AST", t);
   //frame.setVisible(true);

   DXWalker walker = new DXWalker ();
   walker.startRule (parser .getAST ());
   System.out.println ("DONE");

    } catch(Exception e) { System.err.println("Exception: "+e); }
  }
}
```

HashFunction.java

```
import java.util.*;
public class HashforFunction
{
        LinkedList bucket ;
        public HashforFunction ()
        {
                bucket = new LinkedList ();
        }
        public Hashtable putFunction (Hashtable hash1, String key1, Vector argList )
        {
                bucket = new LinkedList ();

                if(hash1.containsKey (key1))
                {
                        bucket = (LinkedList )hash1.get(key1);
                }
                else
                {
                        bucket .clear();
                }
                bucket.addFirst (argList );
                hash1.put((String)key1,  (LinkedList)bucket);
                return hash1;
        }
}
```

CodeWrite.java

```
import java.io.*;
public class CodeWriter
{
        public CodeWriter ()
        {
        }
        public void newWriter (String filename )
        {
                try{
                        System .out.println (filename );
                        runF = new File(filename );
                        runFile = new FileWriter (runF, true);
                }
                catch(IOException e){
                System .err.println ("io exception: " + e);
                }
        }
        public void WriteCode(String text)
        {
                try{
                        System .out.println ("File Output: " + text);
                        runFile .write(text);
                }
                catch(IOException e)
                {
                        System .err.println ("io exception: " + e);
                }
        }
        public void closeWriter ()
        {
                try{
                    runFile .close();
                }
                catch(IOException e)
                {
                    System .err.println ("io exception: " + e);
                }
        }
        File runF;
        FileWriter runFile ;
}
```

DX_File.java

```
import java.io.*;

public class DX_File
{
        private File me;
        private BufferedReader in;
        private BufferedWriter out;
        private int currentLine ;

        /**
        * This is the constructor for the DX_File class. The constructor will open
        * the file if it exists or create a new file with the name that is given if
        * it does not.
        */

        public DX_File(String pathname )
        {
                me = new File(pathname );
                try
                {
                        if (!me.exists ())
                                me.createNewFile ();
                }
                catch(IOException e)
                {
                        System .err.println ("Unable to create file with the following " + "pathname: " +
me.getAbsolutePath ());
                }
                currentLine = 1;
        }
        /**
        * Gets the next line in the file and then increments the counter so that
        * next time the method is call the following line is returned.
        *
        * @return the next line of the file
        */
        public String getline ()
        {
                String line = new String ();
                try
                {
                        in = new BufferedReader (new FileReader (me));
                }
                catch(FileNotFoundException e)
                {
                        System .err.println ("Unable to open file " + me.getAbsolutePath () + " for reading" );
                }
                try
                {
                        for(int i = 0; i < currentLine ; i++)
                        {
                                line = in.readLine ();
                        }
                        currentLine ++;
                        in.close();
                }
                catch(IOException e)
                {
                        System .err.println ("Unable to read from " + me.getAbsolutePath ());
                }
                if(line == null)
                {
                        currentLine = 1;
                        /*
                        try
                        {
                                in = new BufferedReader (new FileReader (me));
                        }
                        catch(FileNotFoundException e)
                        {
                                System .err.println ("Unable to open file " + me.getAbsolutePath () + " for
reading" );
                        }
                        try
                        {
                                for(int i = 0; i < currentLine ; i++)
                                {
                                        line = in.readLine ();
                                }
                                currentLine ++;
                                in.close ();
                        }
                        catch(IOException e)
                        {
                                System .err.println ("Unable to read from " + me.getAbsolutePath ());
                        }
                        */
                }

                if ( currentLine == 1 )
                        return "";
                else
                        return line;
        }
        /**
        * Checks to see if two files are the same.
        *
        * @param f the file to compare to
        * @return true if the files are the same and false if the files are not
        */
        public boolean equals (DX_File f)

        {
                if(me.compareTo (f.me) == 0)
                        return true;
                else
```

```
				return false;
}
/**
* Gives you the name of the file.
*
* @return a string that contains the name of the file.
*/
public String getName ()
{
		return me.getName ();
}
/**
* Gives you the name of the file.
*
* @return a string that contains the absolute path of the file.
*/
public String pathToString ()
{
		return me.toString ();
}
/**
* Gives access to the entire content of a file.
*
* @return a string containing all the contents of the file.
*/
public String getContent ()
{
		String contents = new String ();
		try
		{
				in = new BufferedReader (new FileReader (me));
		}
		catch(FileNotFoundException e)
		{
				System .err.println ("Unable to open file " + me.getAbsolutePath () + " for reading" );
		}
		try
		{
				while (in.ready())
				{
						contents += (in.readLine () + "\n");
				}
				in.close();
		}
		catch(IOException e)
		{
				System .err.println ("Unable to read from " + me.getAbsolutePath ());
		}
		return contents ;
}

/**
* Writes to the file, overwriting previous content.
*
* @param s the string to write to the file
* @return true if write was sucessful, false if failed
*/
public boolean writeContent (String s)
{
		try
		{
				out = new BufferedWriter (new FileWriter (me));
		}
		catch(IOException e)
		{
			System .err.println ("Unable to open file " + me.getAbsolutePath () + " for writing" );
			return false;
		}
		try
		{
			out.write(s,0,s.length ());
			out.flush();
			out.close();
		}
		catch(IOException e)
		{
			System .err.println ("Unable to write to " + me.getAbsolutePath ());
			return false;
		}
		return true;
	}
	/**
	* Writes to the file, appending previous content.
	*
	* @param s the string to write to the file
	* @return true if write was sucessful, false if failed
	*/
	public boolean appendContent (String s)
	{
		String contents = new String ();
		try
		{
			in = new BufferedReader (new FileReader (me));
		}
		catch(FileNotFoundException e)
		{
			System .err.println ("Unable to open file " + me.getAbsolutePath () +
			" for reading" );
		}
		try
		{
			while (in.ready()) {
			contents += (in.readLine () + "\n");
			}
			in.close();
		}
		catch(IOException e)
		{
			System .err.println ("Unable to read from " + me.getAbsolutePath ());
		}
		contents += s;
		try
		{
			out = new BufferedWriter (new FileWriter (me));
		}
		catch(IOException e)
		{
			System .err.println ("Unable to open file " + me.getAbsolutePath () +
			" for writing" );
			return false;
		}
		try
		{
				out.write(contents ,0,contents .length ());
				out.close();
			}
			catch(IOException e)
			{
				System .err.println ("Unable to write to " + me.getAbsolutePath ());
				return false;
			}
			return true;
		}
		}
```

DX_Funcs.java

```
import java.io.*;
import java.util.*;

public class DX_Funcs {


		private Vector tagVector;
		public DX_Funcs ()
		{

		}
		//General Functions
		/**
		* Prints a line of text to the specified file.
		*
		* @param f the file to write to.
		* @param option either "append" or "overwirte"
		* @param line the line of text to write to the file
		*/
		public String fprint (DX_File f, String option , String line)
		{
			if(option .equals ("append" ))
			{
				f.appendContent (line);
			}
			else if(option .equals ("overwrite" ))
			{
				f.writeContent (line);
			}
			else
			{
				System .out.println ("bad call to " + "fprint()  : check the options supplied" );
			}
			return line;
		}


		public String fprint (DX_File f, String option , int line)
			{
				if(option .equals ("append" ))
				{
					System.out.println("appending content " + line);
					f.appendContent (Integer.toString(line));
				}
```

```
                else if(option .equals ("overwrite" ))
                {
                        f.writeContent (Integer.toString(line));
                }
                else
                {
                        System .out.println ("bad call to " + "fprint()  : check the options supplied" );
                }
                return Integer.toString(line);
}
/**
* Read record and stores tags for xml in the memory
*
* @param strTableRecord string from the table record.
* @param option strDelim is delimitor, Default is ;
*/

public String createRecord ( String strTableRecord, String strDelim )
{
        String strDelimiter = ";";

        if ( strDelim != null )
                strDelimiter = strDelim;

        if ( strTableRecord != null )
        {

                // parse the string and put it in the memory

                tagVector = new Vector();

        String dataInput = strTableRecord.trim();
                int startIndex = 0;
                int endIndex = 0;
                int index = 0;
                String strTag;



                while(index < dataInput.length() )
                {
                        endIndex = dataInput.indexOf(strDelimiter,index);

                        if(endIndex == -1)
                        {

                                strTag = new String( dataInput.substring(index) );
                                if ( strTag != null && strTag != "")
                                {
                                        tagVector.add(strTag);
                                        // Reset value of strTag
                                        strTag = "";
                                }
                                break;
                        }
                        else
                        {
                                strTag = new String( dataInput.substring(index,endIndex) );
                                index = ++endIndex;
                        }

                        if ( strTag != null && strTag != "")
```

```
                                        {
                                                tagVector.add(strTag);

                                                // Reset value of strTag
                                                strTag = "";
                                        }
                                }

                                index = 0;

                                while ( index < tagVector.size())
                                {
                                        index ++;
                                }

                        }
                        return "";
                }

                /**
                * Read record and put it in the tags,
                *
                * @param strTableRecord string from the table record.
                * @param option strDelim is delimitor, Default is ;
                */


                public String addXMLRecord ( String strRecord, DX_File outFile, String strDelim  )
                {
                        if ( strRecord == null  || outFile == null || tagVector == null || tagVector.isEmpty() == true) return
        null;

                        String strDelimiter = ";";
                        String returnString = "";

                        if ( strDelim != null )
                                strDelimiter = strDelim;

                        if ( strRecord != null )
                        {

                                // parse the string and prepare XML String

                        String dataInput = strRecord.trim();
                                int startIndex = 0;
                                int endIndex = 0;
                                int index = 0;
                                int vectorIndex = 0;
                                String strTagValue;
                                returnString = returnString + "<RECORD>";
                                while(index < dataInput.length() || vectorIndex < tagVector.size())
                                {

                                        endIndex = dataInput.indexOf(strDelimiter,index);
                                        if(endIndex == -1)
                                        {

                                                strTagValue = new String( dataInput.substring(index) );

                                                if ( strTagValue != null && strTagValue != "")
                                                {
                                                        String strTag = (String)tagVector.get(vectorIndex);
                                                        returnString = returnString + "<" + strTag + ">" +
        strTagValue + "</" + strTag + ">" ;

                                                        // Reset value of strTag and vectorIndex
                                                        vectorIndex ++;
                                                        strTag = "";
                                                }
                                                break;
                                        }
                                        else
                                        {
                                                strTagValue = new String( dataInput.substring(index,endIndex) );
                                                index = ++endIndex;
                                        }

                                        if ( strTagValue != null && strTagValue != "")
                                        {
                                                String strTag = (String)tagVector.get(vectorIndex);

                                                returnString = returnString + "<" + strTag + ">" +  strTagValue +
        "</" + strTag + ">";

                                                // Reset value of strTag and vectorIndex
                                                vectorIndex ++;
                                                strTag = "";
                                        }
                                }
                                returnString = returnString + "</RECORD>";
                        }
                        return returnString;

                }
                /**
                * Reads the next line of text from a file.
                ** @param f the file to read from.
                * @return a string containing the text read from the file.
                */
                public String fgetline (DX_File f)
                {

                        String retline = f.getline ();
                        if ( retline == "" )
                                return null;
                        else
                                return retline;
                }
                /**
                * Writes a line of text to standard out.
                *
                * @param line the line of text to be written.
                */
                public String print(String line)
                {
                        System .out.println(line);
                        System .out.flush();
                        return line;
                }
                public int find(String inputString, String checkStr)
                {

                                int i = inputString.indexOf (checkStr)          ;
                                if (i >=0 )
                                {
                                        return 1;
                                }
                                else
                                {
                                        checkStr = "" ;
                                        return 0;
                                }
                }
                public String print(int line)
                        {
                                System .out.println(Integer.toString(line));
                                System .out.flush();
                                return Integer.toString(line);
                }
                /**
                * Reads a line of text from standard in.
                *
                * @return a string containing the line of text.
                */
                public String getline ()
                {
                        String line = "";
                        try
                        {
                                BufferedReader lineIn = new BufferedReader (new InputStreamReader (System .in));
                                line = lineIn .readLine ();
                        }
                        catch(IOException e)
                        {
                                System .err.println ("Could not read from command line." );
                        }
                        return line;
                }
        }
```

Compile.bat

```
@echo Off

set src_home=C:\DX
set compiler="c:\j2sdk1.4.2_16\bin\javac"
set parser="c:\j2sdk1.4.2_16\bin\java"

echo.
echo Source tree is set to %src_home%

echo.
echo compiler is set to %compiler%

echo.
echo java path is set to %parser%
```

```
echo.
echo    Are the above settings correct?
echo.
echo        If so, Press `Y'
echo        If Not, Press 'N' to End the program and edit compile.bat

set choice=
set /p choice=

if '%choice%'=='N' goto :EOF
if '%choice%'=='n' goto :EOF

echo.
echo compiling the code ...

echo.

set CLASSPATH=%src_home%;%src_home%\antlr;.

cd %src_home%

call :clearall
call :compile   DX_File.java DX_Funcs.java CodeWriter.java HashforFunction.java
call :lexerparser antlr.Tool DxLexer.g
call :compile   DXParser.java
call :compile   DXlexer.java
call :compile   Main.java

goto :EOF


:lexerparser
echo parsing %2
%parser% %1 %2
goto :EOF


:compile
echo Compiling %1
cd %src_home%\
%compiler% %1 %2
goto :EOF
```

```
:clearall
echo %0
cd %src_home%\
del  *.class
goto :EOF
```

All the files need to be in the same directory.

Run.bat

```
@echo Off

set src_home=C:\DX

set java="c:\j2sdk1.4.2_16\bin\java"
set javac="c:\j2sdk1.4.2_16\bin\javac"
echo.
echo Source tree is set to %src_home%

echo.
echo java path is set to %java%

echo.
echo    Are the above settings correct?
echo.
echo        If so, Press `Y'
echo        If Not, Press 'N' to End the program and edit compile.bat

set choice=
set /p choice=

if '%choice%'=='N' goto :EOF
if '%choice%'=='n' goto :EOF

echo.

echo Please select the .dx program to be compiled. Enter 1 2 3 or 4
echo.  1. gcd.dx
echo   2. DXTest.dx
echo   3. DXfind.dx
echo   4. If you select 4 please enter the DX program to be compiled....

set choice=
set /p choice=
if '%choice%'=='1' set prg=gcd.dx
if '%choice%'=='2' set prg=DXTest.dx
if '%choice%'=='3' set prg=DXfind.dx
if '%choice%'=='4' goto  :fourchoice

goto :classpath

:fourchoice
echo enter program name
set prg=
set /p prg=


:classpath
set CLASSPATH=%src_home%;%src_home%\antlr;.

cd %src_home%

call :Execute Main %prg%

goto :EOF


:Execute

echo processing %2

del /F /Q DXMainFile.java
del /F /Q DX.java
del /F /Q File2.xml

echo Now compiling %2
%java% %1 %2
echo Compiled %2 DX program

echo Now compiling output java file
%javac% DXMainFile.java DX.java

echo Now executing DX
%java% DX

goto :EOF
```

File1.txt   for XML generation.

```
First_name;Last_name;city;email;phone_no
Tom ;Smith; Tampa; tomsmith@yahoo.com; 8133832844
Jerry ;Smith; New York; jerrysmith@google.com; 9149873456
Archana;Mandape;Tampa;archanamandape@yahoo.com;813972118
```