Programming Languages and Translators
COMS W4115

Prof. Stephen Edwards

TweaXML

Final Report

By:
Kaushal Kumar
(kk2457@columbia.edu)

Srinivasa Valluripalli
(sv2232@columbia.edu)

Contents

1. Introduction

Abstract

In this document, we present the description, tutorial, sample programs and test classes for TweaXML, a high level language for manipulation of XML documents. We will discuss the motivation behind the idea and key features of the language.

Introduction

TweaXML is a simple language that helps us manipulate and perform various operations on XML documents, like extracting specific data, performing arithmetic/string operations on the extracted data and saving the data in a desired format to a file.

The Extensible Markup Language (XML) is a general-purpose markup language. It is classified as an extensible language because it allows the users to define their own tags. XML is designed primarily to share information across heterogeneous systems, regardless of the specific architecture or language of the systems. It is adopted as an interoperable language across various technologies and component vendors, for example, it is used to share data across J2EE components and .NET components.

Since XML is highly customizable and contains user-defined tags, the technologies to parse an XML document are quite complex. For example, Java provides APIs, like SAX and DOM Parsers to parse XML documents, but they are quite complex and require a thorough knowledge of Java. Therefore we have implemented TweaXML, a language which will be easy to use and will require minimal amount of a prior knowledge of programming to get started right away. TweaXML will enable a novice user to manipulate and operate on XML documents, extract its data, and do arithmetic/string operations on it and save it in a desired format.

2. Language Features

Data Types
string – Data type containing data of character and string types.
file – Data type for writing file on a file system.
node – Data type for a XML element.
int – Standard int data type.

Operators
Arithmetic Operators:
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
= (assignment)
== (equal)
> (greater than)
< (less than)
>= (greater than or equal to)
<= (less than or equal to)
!= (not equal to)

Scope Operators:
"{" , "}"
"(" , ")"
// comment

Looping constructs:
If() {…} else {…}
While(){…}
FOREACH(){…}

Methods available on STRING, NODE and FILE data-types:

STRING:

If the string variable represents a number, arithmetic operations can be done on it using following inbuilt functions:

add str1 str2 - adds two numbers. Concatenates them if both of them are not numbers.

subtract str1 str2 – subtracts the two numbers represented by str1 and str2. It throws a runtime exception if they are not numbers.

multiply str1 str2 – multiplies the two numbers represented by str1 and str2. It throws a runtime exception if they are not numbers.

divide str1 str2 – divide the two numbers represented by str1 and str2. It throws a runtime exception if they are not numbers.

NODE:

getchild node path_to_child_node – gives an arraylist of child nodes defined by xpath (given as the 2nd argument).

length nodes[] – gives the number of nodes present in the nodes array.

getvalue node – gives the value of the node (in string format).

FILE:

open path_of_input_file – opens the input xml file to read and returns the root node.

create path_of_output_file – creates an output file to write and returns a file data type.

Close file_name – closes the output file after all the operations are done.

3. Language Tutorial

TweaXML is designed to handle XML files, extract data from it, do arithmetic/string operations on it and write the data in an output format in desired format. This tutorial will help programmers learn TweaXML as quickly and easily as possible.

Before we get started on the programs, lets look at the input file for the following set of the programs.

```
<students>
	<student>
		<name>kaushal</name>
		<homework1>85</homework1>
		<homework2>85</homework2>
		<midterm>70</midterm>
		<final>90</final>
	</student>
	<student>
		<name>Srini</name>
		<homework1>80</homework1>
		<homework2>85</homework2>
		<midterm>87</midterm>
		<final>95</final>
	</student>
</students>
```

This input file contains the information about each student's marks in a class for various exams. The final aim is to parse this input xml file and extract the data and calculate average marks of each student.

2.1 TweaXML program:

This program shows how to open an XML file to read, extract the child nodes of a node and get the value of a node. Also it will show how to use various statements like if, else, while etc.

```
start(){
	file output;
	node rootNode;
	output = create "AvgMarks.csv";
	rootNode = open "marks_data.xml";
```

```
	node studentNodes[];
	studentNodes = getchild rootNode "student";
	int len;
	len = length studentNodes;
	if(len > 0)
	{
		int j;
		j=0;
		while(j < len)
		{
			node nameNode[], homework1Node[], homework2Node[],
midtermNode[], finalNode[];
			string name, homework1Marks, homework2Marks, midtermMarks,
finalMarks;

			nameNode = getchild studentNodes[j] "name";
			homework1Node = getchild studentNodes[j] "homework1";
			homework2Node = getchild studentNodes[j] "homework2";
			midtermNode = getchild studentNodes[j] "midterm";
			finalNode = getchild studentNodes[j] "final";

			name = getvalue nameNode[0];
			homework1Marks = getvalue homework1Node[0];
			homework2Marks = getvalue homework2Node[0];
			midtermMarks = getvalue midtermNode[0];
			finalMarks = getvalue finalNode[0];

			string totalMarks;
			totalMarks = add homework1Marks homework2Marks;
			totalMarks = add totalMarks midtermMarks;
			totalMarks = add totalMarks finalMarks;

			string avgMarks;
			avgMarks = divide totalMarks "4";

			print output name;
			print output "\t";
			print output avgMarks;
			print output "\n";
			j = j + 1;
		}
	}
	close output;
}
```

Program start: All TweaXML programs are contained within start(){} statement. So the program starts on line 1, with start() statement.

Declarations:

All the variables should be declared before a value can be assigned to them. Example of declarations for various data types:

file output;
node rootNode;
node studentNodes[];
int len;
string totalMarks;

"if" statement:

If statement takes an expression as a parameter, which should evaluate to true or false.

"while" statement:

"while" statement takes an expression as a parameter, which should evaluate to true or false. If expression evaluates to true, the loop will be executed, otherwise not.

Opening an input xml file to read:

rootNode = open "marks_data.xml";

open statement can be used to open an input file to read. It returns the root node of the xml file, which can be used later in the program.

Getting a child node of a node:

studentNodes = getchild rootNode "student";

getchild function can be used to get children of a node. It takes xpath of the child node and returns an array of all the nodes which matches the xpath criterion.

Getting length of an array:

len = length studentNodes;

length function can be used to find the length of an array.

Getting value of a node:

name = getvalue nameNode[0];

getvalue gives the value of the node in string format.

Operations on the string values:

- add str1 str2

        - if both str1 and str2 are numbers, it will add them up and return a string representation of the number. If not, it will return the concatenation of the strings.
    - subtract str1 str2
        - returns str1 – str2. If any one of these two strings is not a number, it will throw runtime exception.
    - multiply str1 str2
        - returns str1 * str2. If any one of these two strings is not a number, it will throw runtime exception.
    - divide str1 str2
        - returns str1 – str2. If any one of these two strings is not a number, it will throw runtime exception.

Writing to the output file:

print output name;

print function can be used to write anything in the output file.

Closing an output file:

c*lose output;*

It finishes the processing of the output file and closes it.

4. Language Manual

Introduction

The following document is a reference manual for TweaXML, a language to perform ope rations on XML documents.

Lexical Conventions

Tokens

Tokens are divided as follows, Identifiers, Keywords, Constants, Operators, Other Separa tors.

Blanks, horizontal and vertical tabs, newlines, form feeds and comments are considered as “white space”. They have no semantic meaning, and are used only as token separators. The input is validated into various tokens that are available.

Comments

Comments are defined as a sequence of characters that begin with the sequence /* and en d with the sequence */. Comments cannot be nested and cannot occur inside a string or ch aracter literals.

Identifier

Identifiers are typically variables and they comprise of alphabets and numbers. Identifiers should essentially start with a character between [A…Z] or [a…z]. All succeeding charac terscan be characters from the sets, [A…Z], [a…z], [0…9] or an underscore (_).

Keywords

Following are the reserved keywords,

string	file	open	close	
node		getvalue	getchild	
if	else	return	continue	
while	break	void		print
add	subtract	multiply	divide	

Constants

Number Constants

Being an XML document, there is no point in having separate data types for integer, float and double etc. All data types are dealed in terms if strings and inbuilt functions (add, subtract, multiply and divided) are provided to do arithmetic/string operations on them.

String Constants

Every constant that is not a number is a string constant. Strings are enclosed in double quotes " ". The escape sequence '\n' denotes a new line and it can be used in any string constants.

Punctuations

These are the punctuation symbols in TweaXML.

; statement end
, argument separator
" " string constant
{} function body or block delimiter
() function argument list

Data Types

TweaXML has four data types available,

int - This data type represents integers.

string This data type is can hold multiple characters of variable length.

file This data type represents an output file, on which print operations can be performed.

node Using this data type we can access the nodes in the documents.

Expressions

Expressions are all evaluated left to right. In case of logical expressions, if the evaluation of the first n elements of the expression is enough for evaluating the whole expression its elf, the n+1th element will not be evaluated.

Arithmetic Expressions

Arithmetic expressions include operations such as +, -
, *, / between number data types. For example, a + b * c - d

Conditional Expressions

Conditional Expressions are those involving comparison operators such as <, <=, >, >=, = = and !=. These can be used in if, while and foreach loops. The associativity of these oper ators is from left to right. All of these operators are binary operators that work on two ope rands. Examples are a<b, b>=d.

Logical Expressions

Logical expressions return a boolean result i.e. either true or false. The supported operato rs are AND (&&), OR (||) and NOT (!).

All of the above expressions are evaluated from left to right and standard precedence rule s are applicable throughout the languge. ! has the highest precendence, followed by && a nd then by ||.

Operators

Unary Operators

Unary operators are applied on the expressions on their right.

Minus sign: □ expression
The result of the minus sign □ operator is the negative value of expression. expression m ust be of type number.

Logical negation: ! expression

The result of the logical negation operator ! is the constant 1 of type number if expression value is 0; constant 0 of type number if expression value is otherwise. Expression must be of type number.

Arithmetic Operators

Multiplication –

expr1 * expr2 returns a value of type number. expr1 and expr2 must be of type number an d this operator returns the product of the two values.

Division –

expr1 / expr2 returns a value of type number. Division by 0 is forbidden and will produce a runtime error. expr1 and expr2 must be of type number and this operator returns the div ision of the two values.

Addition –

expr1 + expr2 returns a value of type number if both expressions are of number type. But + can be performed also on strings to concatenate two strings.

Subtraction –

expr1 □ expr2 returns a value of type number. expr1 and expr2 must be of type number a nd this operator returns the difference between the two
values.

Relational Operators

Greater than □ expr1 > expr2
Lesser than □ expr1 < expr2
Greater than or equal to □ expr1 >= expr2
Lesser than or equal to □ expr1 <= expr2

All the above operators return 1 if true and 0 if false. Both expr1 and expr2 should be of t ype number or string (a combination is not allowed).

In case of number, expr1 > expr2 and expr1 < expr2 will return true (1) and false (0) resp ectively, if expr1 is numerically greater than expr2. Similarly, expr1 >= expr2 and expr1 <= expr2 will return true (1) and false (0) respectively, if expr1 is numerically greater tha n or equal to expr2.

In case of strings, expr1 > expr2 and expr1 < expr2 will return true (1) and false (0) respe ctively, if expr1 comes before expr2 alphabetically. Similarly, expr1 >= expr2 and expr1 <= expr2 will return true (1) and false (0) respectively, if expr1 comes before or is same a s expr2 alphabetically.

Equality Operators

Equality □ expr1 == expr2
Inequality □ expr1!= expr2

The above operators return 1 if true and 0 if false. Both the expressions expr1 and expr2 s hould be of type number or string (a combination is not allowed).

Logical Operators

AND □ expr1 && expr2
Logical AND returns 1 if only both expressions are true and returns false for all other cas es.

OR □ expr1 || expr2
Logical OR returns 1 if both the expressions are true or either of the expressions is true. It returns 0 if both expressions are false.

NOT □ !(expr)

Logical NOT returns 1 if the expression is false and 0 if the expression is true.

Operator Precedence

( )
!
* /
+ □
< > <= >= == !=

&& ||
=
,

All binary operators associate left to right whereas the unary operators associate right to left.

Variable Declaration

The number and string variables are declared similar to Java.

number n1, n2;

string s1;

Assignment Statements

Variables can be assigned with either expressions or constants.

number n1 = expression;
number n2 = 9;

Conditional Statements

The conditional statements in this language are if else,

```
if ( condition)
{
        statement 1;
        statement 2;
}
else
{
        statement 1;
        statement 2;
}
```

Loop Statements

while (condition)

{
statement 1;
statement 2;
}

The while block will be executed only if ‘condition’ evaluates to a value other than 0.

print Statement

print statements can be used to write the output to a file. The syntax is:

print output “hello world”;

Jump statement

Jump statements are used to jump to a different set of statements altogether. They affect t he control flow immediately without any condition check as in iterative or conditional sta tements.

return : used in functions to return a single value or void.

continue : used to continue with the next iteration in a while loop construct.

break : used to break out of a while loop immediately.

Scope rules in TweaXML

Lexical Scope

Identifiers in different name space do not interfere with each other. In TweaXML, there a re two separate name spaces available: variables name space and functions name space. T o clarify the explanation of the lexical scope, the meaning of a block should be defined. A block is a set of statements grouped by curly brackets, {}. Thus, a block of statements is executed like a single statement syntactically.

An identifier can be declared once in a block within a name space.
Identifiers can be used several times in its block. If the same name of an identifier is decl ared outside the block, the identifier in the outer block will be undermined in the current

block. Thus, TweaXML follows static scoping, which means the life of an identifier begi ns with its declaration and ends at the end of its block.

Language defined Functions:

open "input.xml"

This function opens a file to read. Filename should be the location of the file on th e disk and it should be in xml file format. The function opens the file, parses it and returns the root node of the xml document.

create "output.csv"

This function creates an output file named "output.csv" and returns the file datatype, on which print operations can be performed.

getchild node "xpath_to_child_node"

This function returns an arraylist of all the child nodes, satisfying the xpath criteria.

length childnodes[]

This function returns the length of the array.

getvalue node

This function returns the value of the given node.

print output "hello"

This function writes the given string in the output file.

add str1 str2

if both str1 and str2 are numbers, it will add them up and return a string representation of the number. If not, it will return the concatenation of the strings.

subtract str1 str2

returns str1 – str2. If any one of these two strings is not a number, it will throw runtime exception.

multiply str1 str2

returns str1 * str2. If any one of these two strings is not a number, it will throw runtime exception.

divide str1 str2

returns str1 – str2. If any one of these two strings is not a number, it will throw runtime exception.

Sample Program:

Input XML file:

```
<customer-records>
    <orders>
        <customer>
            <name>John</name>
            <order>
                <order-number>1</order-number>
                <amount>100</amount>
            </order>
            <order>
                <order-number>10</order-number>
                <amount>50</amount>
            </order>
            <order>
                <order-number>12</order-number>
                <amount>70</amount>
            </order>
            <order>
                <order-number>15</order-number>
                <amount>150</amount>
            </order>
        </customer>
        <customer>
            <name>Jack</name>
            <order>
                <order-number>1</order-number>
                <amount>100</amount>
            </order>
            <order>
                <order-number>10</order-number>
                <amount>5890</amount>
            </order>
            <order>
                <order-number>12</order-number>
                <amount>7023</amount>
            </order>
            <order>
                <order-number>15</order-number>
                <amount>15056</amount>
            </order>
        </customer>
</customer-records>
```

TweaXML program:

```
start(){
	file output;
	node rootNode;
	output = create "totalAmounts.csv";
	rootNode = open "order_data.xml";

	node childNodes[];
	childNodes = getchild rootNode "orders/customer";
	int len;
	len = length childNodes;
	if(len > 0)
	{
		int j;
		j=0;
		while(j < len)
		{
			node nextNode;
			nextNode = childNodes[j];

			node nameNodes[];
			nameNodes = getchild childNodes[j] "name";

			string nextName;
			nextName = getvalue nameNodes[0];

			node amountNodes[];
			amountNodes = getchild childNodes[j] "order/amount";
			string totalAmount;
			totalAmount = "0";
			int amountNodesLength;
			amountNodesLength = length amountNodes;

			if(amountNodesLength > 0){
				int k;
				k = 0;
				while(k < amountNodesLength){
					string nextAmount;
					nextAmount = getvalue amountNodes[k];
					totalAmount = add totalAmount nextAmount;
					k = k + 1;
				}
			}
```

```
                print output nextName;
                print output "\t";
                print output totalAmount;
                print output "\n";
                j = j + 1;
            }
        }
        close output;
}
```

This program reads the input xml file (given above) containing order data for a shop, parses it, calculates total amount of the orders for each customer and then prints it in a tab-delimited output file.

5. Project Plan

Project Processes

Our team being a 2 member team, interaction and collaboration of work was relatively easy. We first decided that language should be made simple and useful for multiplatform users. We came up with the idea of a language for manipulating XML documents, which a common file format for cross platform data exchange. We met regularly to discuss the project progress and also corresponded with the TA to inform our progress. We began, developing our language, first with the development of grammar for our language. Once the grammar has been finalized, we moved on to the functional implementation of the language.

Project Timeline

Following is the timeline for our project

Date	Milestone
9/15/2007	Team formed
9/25/2007	Project Proposal submitted
10/18/2007	LRM submitted
11/10/2007	Lexer/Parser completed
11/30/2007	Tree walker completed
12/10/2007	Code generation completed
12/11/2007	Testing started
12/14/2007	Errors rectified and code improved
12/16/2007	Testing completed

Roles and Responsibilities

Kaushal Kumar

- Tree Walker
- Code Generation
- Helper class for code generation (CodeGen.java)
- Java classes representing various data types and symbol tables.
- Testing

Srinivasa Valluripalli

- Lexer
- Parser
- Documentation

Software Development Environment

Implementation Language – Java 5.0
Lexer, Parser & Tree walker – ANTLR 3.0
Version Control – CVS
Integrated Development Environment – Eclipse 3.3 with ANTLR plug-in

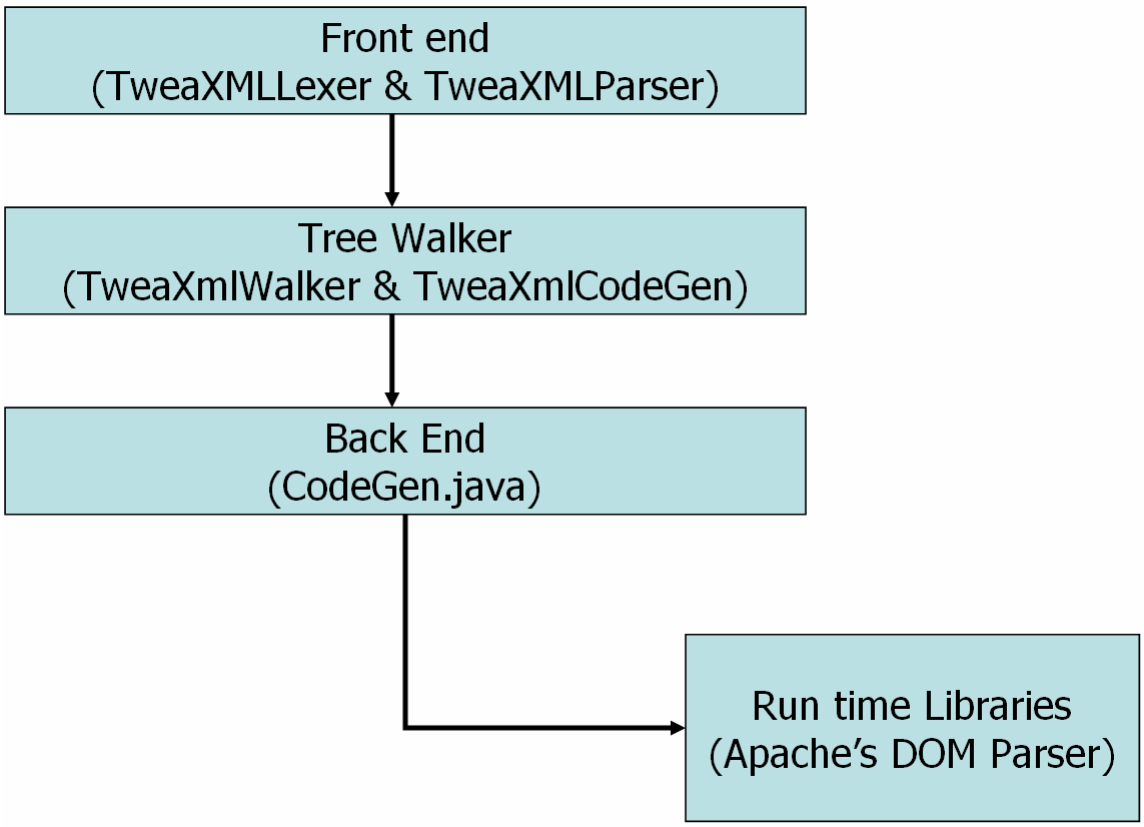
Project Log

Following is the project log.

Date	Milestone
9/15/2007	Team formed
9/18/2007	Project meeting
9/21/2007	Language chosen
9/24/2007	Project proposal finalized
9/25/2007	Project Proposal submitted
9/30/2007	Project meeting for grammar discussion
10/5/2007	Project meeting for LRM

10/10/2007	First version of Lexer/Parser completed
10/15/2007	First LRM of LRM completed
10/17/2007	LRM review and revision
10/18/2007	Language Reference Manual submitted
10/25/2007	Project meeting for grammar discussion
10/30/2007	Grammar finalized
11/6/2007	Lexer/Parser finalized
11/10/2007	Lexer/Parser completed
11/20/2007	Project meeting for progress update
11/30/2007	Tree walker completed
12/10/2007	Code generation completed
12/11/2007	Testing started
12/14/2007	Errors rectified and code improved
12/16/2007	Testing completed

6. Architectural Design



7. Testing

a. Testing if-else statements:

```
start(){
        int i;
        i = 10;
        int j;
        j = 0;
        if(i > 0)
        {
                j = j + 1;
        }
        else
        {
                j = j - 1;
        }
}
```

b. Testing while loop:

```
start(){
    int i;
    i = 0;
    while(j < 10)
    {
            j = j + 1;
    }

}
```

c. Opening a file to read

```
start(){
    node rootNode;
    rootNode = open "marks_data.xml";
}
```

d. Opening/closing a file to write

```
start(){
    file output;
    output = create "output.txt";
```

```
    print output “hello”;

    close output;
}
```

e. Getting child of a node

```
start(){
    file output;
    node rootNode;
    output = create "AvgMarks.csv";
    rootNode = open "marks_data.xml";

    node studentNodes[];
    studentNodes = getchild rootNode "student";
    int len;
    len = length studentNodes;
    print output len;

    close output;
}
```

f. Full test program, combining all the functionalities:

```
start(){
    file output;
    node rootNode;
    output = create "totalAmounts.csv";
    rootNode = open "order_data.xml";

    node childNodes[];
    childNodes = getchild rootNode "orders/customer";
    int len;
    len = length childNodes;
    if(len > 0)
    {
            int j;
            j=0;
            while(j < len)
            {
                    node nextNode;
                    nextNode = childNodes[j];

                    node nameNodes[];
                    nameNodes = getchild childNodes[j] "name";
```

```
                    string nextName;
                    nextName = getvalue nameNodes[0];

                    node amountNodes[];
                    amountNodes = getchild childNodes[j] "order/amount";
                    string totalAmount;
                    totalAmount = "0";
                    int amountNodesLength;
                    amountNodesLength = length amountNodes;

                    if(amountNodesLength > 0){
                        int k;
                        k = 0;
                        while(k < amountNodesLength){
                            string nextAmount;
                            nextAmount = getvalue amountNodes[k];
                            totalAmount = add totalAmount nextAmount;
                            k = k + 1;
                        }
                    }
                    print output nextName;
                    print output "\t";
                    print output totalAmount;
                    print output "\n";
                    j = j + 1;
                }
            }
            close output;
        }
```

8. Lessons Learned

Kaushal Kumar

I learnt a great deal of compiler construction concepts. Learning ANTLR was fun and challenging too at the same time. Going through the whole process of doing lexical analysis, semantic analysis, tree generation etc gave me a great insight into the internal working of a compiler. Now I can truly appreciate the complexities in any other compiler.

I learnt a lesson of being agile and proactive while doing this course. I had to wished to implement many many more functionalities in this language than what I actually could. If I had been a little faster in learning these new concepts, I could have managed to do all the things I had hoped to do.

Srinivasa Valluripalli

Although I have taken a similar course during my under graduation, I have not implemented a language then. Developing a language of our own is a great idea. During this course I learned how easy it can be to develop a language with tool such as ANTLR. As my part of the project, I was responsible for writing the grammar for lexer and parser using ANTLR grammar. I learned and understood the concepts of writing proper grammar for a language and was able to successfully write grammar for lexer and parser. Also I understood the importance of schedule. One has to strictly follow the schedule in order to finish the things on time.

9. Appendix

Lexer.g

```
header{
	package tweaxml;
}

/**
 * @author sv2232
 */
class TweaXMLLexer extends Lexer;
	options {
		k = 2;
		testLiterals = false;
		exportVocab = TWEAXML;
		charVocabulary = '\3'..'\377';
	}

	LPAREN : '(';
	RPAREN : ')';
	LCURLY : '{';
	RCURLY : '}';
	LBRACKET : '[';
	RBRACKET : ']';
	COMMA : ',';
	DQUOTE : '"';
	SEMI! : ';';

	STRCON : '"' (~('"' | '\n'))* '"' ;

	NOT : '!';
	PLUS : '+';
	MINUS : '-';
	UMINUS : '~';
	TIMES : '*';
	DIV : '/';
	ASSIGN : '=';
	EQ : "==";
	INEQ : "!=";
	AND : "&&";
	OR : "||";
	GT : '>';
	GEQ : ">=";
	LT : '<';
```

```
	LEQ : "<=";
	COLON : ':';

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

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

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

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

	MLCOMMENT : "/*" (options {greedy = false;} : .)* "*/"
		{ $setType(Token.SKIP); };


/**
 * @author kk2457
 */
class TweaXMLParser extends Parser;
	options {
		k = 2;
		buildAST = true;
		exportVocab = TWEAXML;
	}
	tokens {
		CODE;
		FUNC_PROTOS;
		FUNC_BODY;
		DECLS;
		ARGS;
		ARG;
		OPSTMTS;
		CONTROLSTMT;
		ELSESTMT;
	}

	program :(func_prototypes)* main (function)* EOF!
		{ #program = #([CODE], program); }
	;
```

```
main : "start"^ LPAREN! RPAREN! body
;

body : LCURLY! (stmt)* RCURLY
;

expr : conditional_expr ( AND^ conditional_expr
| OR^ conditional_expr)*
;

conditional_expr : plus_minus_expr ( GT^ plus_minus_expr
| GEQ^ plus_minus_expr
| LT^ plus_minus_expr
| LEQ^ plus_minus_expr
| EQ^ plus_minus_expr
| INEQ^ plus_minus_expr)*
;

plus_minus_expr : times_div_expr ( PLUS^ times_div_expr
| MINUS^ times_div_expr)*;
times_div_expr : primary_expr ( TIMES^ primary_expr | DIV^
primary_expr )*
;

primary_expr : (NOT^|UMINUS^)? ((ID (array_expr)?) | NUMB)
| LPAREN! expr RPAREN!
| function_call
;

array_expr : LBRACKET plus_minus_expr RBRACKET
;

blank_array : LBRACKET RBRACKET
;

func_prototypes : ("void"|type(blank_array)?) ID functionprog SEMI!
{ #func_prototypes = #([FUNC_PROTOS], func_prototypes); }
;
function : ("void"|type(blank_array)?) ID functionprog body
{ #function = #([FUNC_BODY], function); }
;
functionprog : LPAREN! func_paras RPAREN!
{ #functionprog = #([ARGS], functionprog); }
;
func_paras : (param (COMMA! param)*)?
;
```

```
param: type ID (blank_array)?
{ #param = #([ARG], param); }
;
type : "int" | "number" | "string" | "node" | "file"
;

function_call : ID LPAREN (((expr|STRCON) (COMMA! (expr|STRCON))*)?)
RPAREN
;
function_call_stmt : function_call SEMI!
;

int_stmt : "int" ID int_tail SEMI!
;
int_tail : (int_decl_array)?
(COMMA! ID (int_decl_array)?)*
;
int_decl_array : LBRACKET (NUMB)? RBRACKET
;

number_stmt : "number" ID number_tail SEMI!
;
number_tail : (decl_array)?
(COMMA! ID (decl_array)?)*
;
decl_array : LBRACKET (NUMB)? RBRACKET
;

file_stmt : "file" ID file_tail SEMI!
;
file_tail : (decl_array)?
(COMMA! ID (decl_array)?)*
;

node_stmt : "node" ID node_tail SEMI!
;
node_tail : (decl_array)?
(COMMA! ID (decl_array)?)*
;
string_stmt : "string" ID string_tail SEMI!
;
string_tail : (decl_array)?
(COMMA! ID (decl_array)?)*
;
decls_stmt:decl_stmt
{#decls_stmt = #([DECLS], decls_stmt);}
```

```
;
decl_stmt : number_stmt
| int_stmt
| node_stmt
| string_stmt
| file_stmt
;

contains_stmt : "contains"^ (ID(array_expr)?|function_call)
(STRCON | function_call)
;
highlight_stmt : "highlight"^ (ID(array_expr)?|function_call )
(STRCON|ID(array_expr)?|function_call)
;
extractpage_stmt : "extractpage"^ (ID(array_expr)?|function_call)
(numid | function_call)
;
create_stmt : "create"^ (STRCON|ID (array_expr)?|function_call)
;
open_stmt : "open"^ (STRCON|ID (array_expr)?|function_call)
;
set_stmt : "setauthor"^ (ID(array_expr)?|function_call)
(STRCON|(ID(array_expr)?)|function_call)
;
get_stmt : "getauthor"^ (function_call | ID(array_expr)?)
;
op_stmts : (highlight_stmt SEMI!
| set_stmt SEMI!
| print_stmt SEMI!
| close_stmt SEMI!
| totext_stmt SEMI!)
{#op_stmts = #([OPSTMTS], op_stmts); }
;

conditional_stmt : "if"^ LPAREN! expr RPAREN! (body) (elsepart)?
{#conditional_stmt = #([CONTROLSTMT], conditional_stmt); }
;
elsepart: (options {greedy=true;} : "else"^ (body))
{ #elsepart = #([ELSESTMT], elsepart); }
;
iteration_stmt : "while"^ LPAREN! expr RPAREN! (body)
{#iteration_stmt = #([CONTROLSTMT], iteration_stmt); }
;
jump_stmt : "return"^ (expr|STRCON)? SEMI!
| "continue"^ SEMI!
| "break"^ SEMI!
```

```
;
length_stmt: "length"^ (ID(array_expr)?|function_call)
;
print_stmt : "print"^ ID
(((expr)=>(expr)|(strcon_expr)=>(strcon_expr)|STRCON|(ID(array_expr)
?)|function_call))
;
close_stmt : "close"^ (ID(array_expr)?|function_call)
;
totext_stmt : "totextfile"^ (ID(array_expr)?|function_call)
(STRCON|(ID(array_expr)?)|function_call)
;
getchild_stmt : "getchild"^ (ID(array_expr)?|function_call)
(STRCON|(ID(array_expr)?)|function_call)
;
add_stmt : "add"^ (ID(array_expr)?|function_call)
(STRCON|(ID(array_expr)?)|function_call)
;
subtract_stmt : "subtract"^ (ID(array_expr)?|function_call)
(STRCON|(ID(array_expr)?)|function_call)
;
multiply_stmt : "multiply"^ (ID(array_expr)?|function_call)
(STRCON|(ID(array_expr)?)|function_call)
;
divide_stmt : "divide"^ (ID(array_expr)?|function_call)
(STRCON|(ID(array_expr)?)|function_call)
;
getvalue_stmt: "getvalue"^ (ID(array_expr)?|function_call)
;

strcon_expr : (
STRCON
| NUMB
| (ID (array_expr)?)
| function_call
)
(((PLUS^ (
STRCON
| NUMB
| (ID (array_expr)?)
| function_call
)
)*)
|(EQ^|INEQ^)(STRCON
| NUMB
| (ID (array_expr)?)
```

```
        | function_call
        ))
        ;

        id_assign_stmt : (ID|(ID LBRACKET plus_minus_expr RBRACKET))
        ASSIGN^
        ( (expr) => expr
        | (strcon_expr) => strcon_expr
        | get_stmt
        | contains_stmt
        | extractpage_stmt
        | create_stmt
        | open_stmt
        | length_stmt
        | getchild_stmt
        | add_stmt
        | subtract_stmt
        | multiply_stmt
        | divide_stmt
        | getvalue_stmt
        ) SEMI!
        ;
        numid : ID(array_expr)?
        | NUMB
        ;

        stmt : id_assign_stmt
        | decls_stmt
        | conditional_stmt
        | iteration_stmt
        | jump_stmt
        | op_stmts
        | function_call_stmt
        ;
```

TweaXmlCodeGen.g

```
header{
        package tweaxml;
}
{
        import java.util.*;
}
```

```
/**
 * @author kk2457
 */
class TweaXmlCodeGen extends TreeParser;
	options {
		importVocab = TWEAXML;
		defaultErrorHandler = true;
	}
	{
		TweaXmlOut out = new TweaXmlOut();
		boolean declFlag = false;
		boolean controlFlag = false;
		boolean returnFlag = false;
		boolean hasMoreCode = false;
		Type curRetType = null;
		ArrayList<Type> tempForFunc = new ArrayList<Type>(1);
		Env env = new Env();
		boolean condFlag = false;
	}



	elsebody throws CompilerException
	:#(ELSESTMT elsebody)
	|#("else" { env.addNewScope();} mainBody RCURLY
	{env.deleteCurrentScope();})
	//|//mainBody
	;



	opstmts throws CompilerException
	{
		Variable var = null;
		Variable var2 = null;
		Type ltype = null;
		Type rtype = null;
	}
	:#("print" printFunc)

	| #("close" closeFunc)
	;
```

```
rhsOfAssign returns [Variable rexpr] throws CompilerException
{
        Variable var = null;
        Variable var2 = null;
        Type rtype = null;
        Type ltype = null;
        rexpr = null;
        Variable lhs = null;
        Variable rhs = null;
        String fname = null;
        String l, r, varName = null;
        ArrayList<Type> funcArgTypes = null;
}
:#(PLUS lhs = rhsOfAssign rhs = rhsOfAssign)
        {
                l = lhs.getVarName().getName();
                r = rhs.getVarName().getName();
                varName = CodeGen.expressionGen(new
                        StringBuffer("("+l),new StringBuffer(r+")"),new
                                StringBuffer("+")).toString();
                rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
                Iden(varName));
        }
|#(MINUS lhs = rhsOfAssign rhs = rhsOfAssign)
        {
                l = lhs.getVarName().getName();
                r = rhs.getVarName().getName();
                if(condFlag)
                {
                        varName = CodeGen.expressionGenIntToBool(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("-")).toString();
                }else
                {
                        varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("-")).toString();
                }
                rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
        }
|#(TIMES lhs = rhsOfAssign rhs = rhsOfAssign)
        {
                l = lhs.getVarName().getName();
                r = rhs.getVarName().getName();
                if(condFlag)
```

```
                        {
                                varName = CodeGen.expressionGenIntToBool(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("*")).toString();
                        }else
                        {
                                varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("*")).toString();
                        }
                        rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
                }
        |#(DIV lhs = rhsOfAssign rhs = rhsOfAssign)
                {
                        l = lhs.getVarName().getName();
                        r = rhs.getVarName().getName();
                        if(condFlag)
                        {
                                varName = CodeGen.expressionGenIntToBool(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("/")).toString();
                        }else
                        {
                                varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("/")).toString();
                        }
                        rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
                }
        |#(AND lhs = rhsOfAssign rhs = rhsOfAssign)
                {
                        l = lhs.getVarName().getName();
                        r = rhs.getVarName().getName();
                        if(condFlag)
                        {
                                varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("&&")).toString();
                        }else
                        {
                                varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("&&")).toString();
                        }
                        rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
                }
        |#(OR lhs = rhsOfAssign rhs = rhsOfAssign)
                {
                        l = lhs.getVarName().getName();
```

```
                        r = rhs.getVarName().getName();
                        if(condFlag)
                        {
                                varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("||")).toString();
                        }else
                        {
                                varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("||")).toString();
                        }
                        rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
                        Iden(varName));
                }
        |#(LT lhs = rhsOfAssign rhs = rhsOfAssign)
                {
                        l = lhs.getVarName().getName();
                        r = rhs.getVarName().getName();
                        if(condFlag)
                        {
                                varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("<")).toString();
                        }else
                        {
                                varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("<")).toString();
                        }
                        rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
                        Iden(varName));
                }
        |#(LEQ lhs = rhsOfAssign rhs = rhsOfAssign)
                {
                        l = lhs.getVarName().getName();
                        r = rhs.getVarName().getName();
                        if(condFlag)
                        {
                                varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("<=")).toString();
                        }else
                        {
                                varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("<=")).toString();
                        }
                        rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
                }
        |#(GT lhs = rhsOfAssign rhs = rhsOfAssign)
```

```
		{
			l = lhs.getVarName().getName();
			r = rhs.getVarName().getName();
			if(condFlag)
			{
				varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer(">")).toString();
			}else
			{
				varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer(">")).toString();
			}
			rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
		}
	|#(GEQ lhs = rhsOfAssign rhs = rhsOfAssign)
		{
			l = lhs.getVarName().getName();
			r = rhs.getVarName().getName();
			if(condFlag)
			{
				varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer(">=")).toString();
			}else
			{
				varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer(">=")).toString();
			}
			rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
		}
	|#(EQ lhs = rhsOfAssign rhs = rhsOfAssign)
		{
			l = lhs.getVarName().getName();
			r = rhs.getVarName().getName();
			if(lhs.getVarType() instanceof TypeFile)
			{
			}
			else if(lhs.getVarType() instanceof TypeString)
			{
				if(condFlag){
					varName = CodeGen.stringEqualityBool(new
StringBuffer("("+l),new StringBuffer(r+")")).toString();
				}else{
					varName = CodeGen.stringEqualityInt(new
StringBuffer("("+l),new StringBuffer(r+")")).toString();
```

```
                        }
                    }
                    else
                    {
                        if(condFlag){
                            varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("==")).toString();
                        }
                        else
                        {
                            varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("==")).toString();
                        }
                    }
                    rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
                }
        |#(INEQ lhs = rhsOfAssign rhs = rhsOfAssign)
                {
                    l = lhs.getVarName().getName();
                    r = rhs.getVarName().getName();
                    if(lhs.getVarType() instanceof TypeString)
                    {
                        if(condFlag)
                        {
                            varName = CodeGen.stringInequalityBool(new
StringBuffer("("+l),new StringBuffer(r+")")).toString();
                        }else
                        {
                            varName = CodeGen.stringInequalityInt(new
StringBuffer("("+l),new StringBuffer(r+")")).toString();
                        }
                    }
                    else
                    {
                        if(condFlag)
                        {
                            varName = CodeGen.expressionGen(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("!=")).toString();
                        }else
                        {
                            varName = CodeGen.expressionGenBoolToInt(new
StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("!=")).toString();
                        }
                    }
```

```
                        rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
Iden(varName));
                }
        |#(UMINUS rhs = rhsOfAssign)
                {
                        r = rhs.getVarName().getName();
                        varName = CodeGen.unaryMinus(new
StringBuffer("("+r+")")).toString();
                        rexpr = new Variable(Type.newInstance(rhs.getVarType()),new
Iden(varName));
                }
        |#(NOT rhs = rhsOfAssign)
                {
                        r = rhs.getVarName().getName();
                        if(condFlag)
                        {
                                varName = CodeGen.unaryNotBool(new
StringBuffer("("+r+")")).toString();
                        }else
                        {
                                varName = CodeGen.unaryNotInt(new
StringBuffer("("+r+")")).toString();
                        }
                        rexpr = new Variable(Type.newInstance(rhs.getVarType()),new
Iden(varName));
                }
        |#("length" var = getId)
                {
                        String vName = null;
                        if(var.getVarType() instanceof TypeArray)
                        {
                                vName = CodeGen.arrayLength(new

        StringBuffer(var.getVarName().getName())).toString();
                        }else
                        {
                                throw new CompilerException("length is applicable to only
arrays");
                        }
                        rexpr = new Variable(new TypeInt(), new Iden(vName));
                }
        |#("open" var = getId)
                {
                        String vName = CodeGen.fileOpen(new
StringBuffer(var.getVarName().getName())).toString();
                        rexpr = new Variable(new TypeNode(), new Iden(vName));
```

```
            }
        |#("create" var = getId)
            {
                String vName = CodeGen.fileCreate(new
StringBuffer(var.getVarName().getName())).toString();
                rexpr = new Variable(new TypeFile(), new Iden(vName));
            }
        |#("add" var = getId var2 = getId)
            {
                String vName=null;
                if(var.getVarType() instanceof TypeString && var2.getVarType()
                instanceof TypeString){
                    vName = CodeGen.add(new
StringBuffer(var.getVarName().getName()),new
StringBuffer(var2.getVarName().getName())).toString();
                }
                else{
                    throw new CompilerException("only string parameters can
be passed to add function.");
                }
                rexpr = new Variable(new TypeString(),new Iden(vName));
            }
        |#("subtract" var = getId var2 = getId)
            {
                String vName=null;
                if(var.getVarType() instanceof TypeString && var2.getVarType()
                instanceof TypeString){
                    vName = CodeGen.subtract(new
StringBuffer(var.getVarName().getName()),new
StringBuffer(var2.getVarName().getName())).toString();
                }
                else{
                    throw new CompilerException("only string parameters can
be passed to subtract function.");
                }
                rexpr = new Variable(new TypeString(),new Iden(vName));
            }
        |#("multiply" var = getId var2 = getId)
            {
                String vName=null;
                if(var.getVarType() instanceof TypeString && var2.getVarType()
                instanceof TypeString){
                    vName = CodeGen.multiply(new
StringBuffer(var.getVarName().getName()),new
StringBuffer(var2.getVarName().getName())).toString();
                }
```

```
                        else{
                                throw new CompilerException("only string parameters can
be passed to multiply function.");
                        }
                        rexpr = new Variable(new TypeString(),new Iden(vName));
                }
        |#("divide" var = getId var2 = getId)
                {
                        String vName=null;
                        if(var.getVarType() instanceof TypeString && var2.getVarType()
                        instanceof TypeString){
                                vName = CodeGen.divide(new
StringBuffer(var.getVarName().getName()),new
StringBuffer(var2.getVarName().getName())).toString();
                        }
                        else{
                                throw new CompilerException("only string parameters can
be passed to divide function.");
                        }
                        rexpr = new Variable(new TypeString(),new Iden(vName));
                }
        |#("getchild" var = getId var2 = getId)
                {
                        String vName=null;
                        if(var.getVarType() instanceof TypeString && var2.getVarType()
                        instanceof TypeString){
                        }
                        if(var.getVarType() instanceof TypeNode && var2.getVarType()
                                instanceof TypeString)
                        {
                                vName = CodeGen.getChildStmt(new
StringBuffer(var.getVarName().getName()),new
StringBuffer(var2.getVarName().getName())).toString();
                        }
                        rexpr = new Variable(new TypeString(),new Iden(vName));
                }
        |#("getvalue" var = getId)
                {
                        String vName = null;
                        if(var.getVarType() instanceof TypeNode)
                        {
                                vName = CodeGen.getValueStmt(new

        StringBuffer(var.getVarName().getName())).toString();
                        }else
                        {
```

```
                        throw new CompilerException("getvalue is only applicable
to nodes.");
                }
                rexpr = new Variable(new TypeString(), new Iden(vName));
        }
    |(var = getId)
        {
                if(var == null)
                        System.out.println("VAR IS NULL!!!!");
                rexpr = var;
        }
    ;


    getId returns [Variable v] throws CompilerException
    {
        String id = null;
        SymbolTable cur =env.getCurrentScope();
        String arrExpr = null;
        boolean isFunc = false ;
        ArrayList<Type> func = new ArrayList<Type>();
        v = null;
        ArrayList<Variable> argList = null;
    }
    :str:ID ((LBRACKET arrExpr = checkArrayExpr) | (LPAREN {isFunc = true;}
argList = checkFunc))?
        {
                if(isFunc)
                {
                        id = str.getText();
                        Prototype pt = env.getProto(id);
                        v                                =                                new
Variable(Type.newInstance(pt.getReturnType()),new Iden(id));
                        v.setInitialized();
                }
                else
                {
                        id = str.getText();
                        v = Variable.newInstance(cur.get(id));
                        if(arrExpr!=null)
                        {
                                if(v.getVarType().getType().startsWith(Type.INT))
                                        v.setVarType(new TypeInt());

    if(v.getVarType().getType().startsWith(Type.STRING))
                                        v.setVarType(new TypeString());
```

```
	if(v.getVarType().getType().startsWith(Type.FILE))
                                        v.setVarType(new TypeFile());

	if(v.getVarType().getType().startsWith(Type.NODE))
                                        v.setVarType(new TypeNode());
                                    v.setVarName(new

	Iden(v.getVarName().getName()+"["+arrExpr+"]"));
                                }
                        }
                }
	|str2:NUMB	{v	=	new	Variable(new	TypeInt(),new
Iden(str2.getText()));v.setInitialized();}
	|str3:STRCON	{v	=	new	Variable(new	TypeString(),new
Iden(str3.getText()));v.setInitialized();}
	;


	checkFunc returns[ArrayList<Variable> args]throws CompilerException
		{
			Variable t = null;
			args = new ArrayList<Variable>();
			ArrayList<Variable> temp = new ArrayList<Variable>();
		}
	:RPAREN
		{
		}
	|(t=rhsOfAssign)
		{
			if(t!=null)
				args.add(t);
		}
		(temp = checkFunc)
			{
				for(int i=0;i<temp.size();i++)
					args.add(temp.get(i));
			}
	;




	checkArrayExpr returns[String arrExp] throws CompilerException
```

```
	{
		Variable t = null;
		arrExp = null;
	}
	:(t = rhsOfAssign RBRACKET)
		{
			arrExp = t.getVarName().getName();
		}
	;


	printFunc throws CompilerException
	{
		Variable var = null;
		Variable var2 = null;
	}
	:(var = getId var2 = getId)
		{
			out.append(CodeGen.printInFile(new
StringBuffer(var.getVarName().getName()),                                    new
StringBuffer(var2.getVarName().getName())));
		}
	;


	closeFunc throws CompilerException
	{
		Variable v = null;
	}
	:(v = rhsOfAssign)
		{
			out.append(CodeGen.closeFile(new
StringBuffer(v.getVarName().getName())));
		}
	;







	declHandler throws CompilerException
	{
		SymbolTable cur = env.getCurrentScope();
		int arrDim=-1;
```

```
		String token = new String();
	}
	:(
		"int"
		(
			token = strdecl arrDim = checkArray
			{
				Type t = new TypeInt();
				if(arrDim>=0)
				{
					out.append(CodeGen.arrayDecl(new
						StringBuffer(t.getType()),new
StringBuffer(token),new StringBuffer(Integer.toString(arrDim))));
					t = new TypeArray(t);
				}
				else
				{
					out.append(CodeGen.varDeclaration(new
StringBuffer(t.getType()), new StringBuffer(token)));
				}
				out.append(CodeGen.getEndStmt());
				Variable v = new Variable(t,new Iden(token));
				if(arrDim>=0)
					v.setInitialized();
				cur.put(token,v);
			}
		)*

		|"number"
		(
			token = strdecl arrDim = checkArray
			{
				Type t = new TypeNumber();
				if(arrDim>=0)
				{
					out.append(CodeGen.arrayDecl(new
						StringBuffer(t.getType()),new
StringBuffer(token),new StringBuffer(Integer.toString(arrDim))));
					t = new TypeArray(t);
				}
				else
				{
					out.append(CodeGen.varDeclaration(new
StringBuffer(t.getType()), new StringBuffer(token)));
				}
				out.append(CodeGen.getEndStmt());
```

```
                        Variable v = new Variable(t,new Iden(token));
                        if(arrDim>=0)
                                v.setInitialized();
                        cur.put(token,v);
                }
        )*

        |"file"
        (
                token = strdecl arrDim = checkArray
                {
                        Type t = new TypeFile();
                        if(arrDim>=0){
                                t = new TypeArray(t);
                        }
                        else
                        {
                                out.append(CodeGen.varDeclaration(new
StringBuffer(t.getType()), new StringBuffer(token)));
                        }
                        out.append(CodeGen.getEndStmt());
                        Variable v = new Variable(t,new Iden(token));
                        if(arrDim>=0)
                                v.setInitialized();
                        cur.put(token,v);
                }
        )*

        |"node"
        (
                token = strdecl arrDim = checkArray
                {
                        Type t = new TypeNode();
                        if(arrDim>=0){
                                out.append(CodeGen.arrayDecl(new
StringBuffer(t.getType()),new                                   StringBuffer(token),new
StringBuffer(Integer.toString(arrDim))));
                                t = new TypeArray(t);
                        }
                        else
                        {
                                out.append(CodeGen.varDeclaration(new
StringBuffer(t.getType()), new StringBuffer(token)));
                        }
                        out.append(CodeGen.getEndStmt());
                        Variable v = new Variable(t,new Iden(token));
```

```
                        if(arrDim>=0)
                                v.setInitialized();
                        cur.put(token,v);
                }
        )*

        |"string"
        (
                token = strdecl arrDim = checkArray
                {
                        Type t = new TypeString();
                        if(arrDim>=0)
                        {
                                out.append(CodeGen.arrayDecl(new
StringBuffer(t.getType()),new                   StringBuffer(token),new
StringBuffer(Integer.toString(arrDim))));
                                t = new TypeArray(t);
                        }
                        else
                        {
                                out.append(CodeGen.varDeclaration(new
StringBuffer(t.getType()), new StringBuffer(token)));
                        }
                        out.append(CodeGen.getEndStmt());
                        Variable v = new Variable(t,new Iden(token));
                        if(arrDim>=0)
                                v.setInitialized();
                        cur.put(token,v);
                }
        )*
    )
    {
        env.refreshCurrentScope(cur);
    }
    declHandler

    |
    ;


    strdecl returns [String s]
    {
        s=null;
    }
    :str2:ID {s=str2.getText();}
    ;
```

```
checkArray returns[int arrDim]{
	arrDim = -1;
}
:LBRACKET arrDim = checkArray
|str:NUMB RBRACKET {arrDim = Integer.parseInt(str.getText());}
| RBRACKET {arrDim=0;}
|{arrDim=-1;}
;




func_body throws CompilerException
{
	Prototype p=null;
	Type ret=null;
	String name=null;
	ArrayList<Variable> list=null;
	SymbolTable cur =null;
	returnFlag = false;
}
:#( FUNC_BODY { }
(
	ret = ret_type) (ret_type_array { ret = new TypeArray(ret); } )?
	(name = func_name)
	(list = args)
		{
			p = env.getProto(name);
			env.addNewScope();
			cur = env.getCurrentScope();
			for(int i=0; i<list.size(); i++)
			{
				Variable v = list.get(i);
				v.setInitialized();
				cur.put(v.getVarName().getName(),v);
			}
			env.refreshCurrentScope(cur);
		}
	(mainBody)
	{
		if(hasMoreCode ==true)
		{
		}
		if(curRetType == null && ret.getType()!=null)
```

```
                        {
                        }
                        else
                        {
                                if(!ret.getType().equals(curRetType.getType()))
                                {
                                }
                        }
                }
                RCURLY {out.append("}\n\n");hasMoreCode = false ; returnFlag = false;curRetType = null ;
                env.deleteCurrentScope();}
        )
        (func_body)
        |
        ;




        func_protos throws CompilerException
        {
                Prototype p;
                Type ret;
                String name;
                ArrayList<Variable> list;
        }
        :#( FUNC_PROTOS
                (ret = ret_type) (ret_type_array { ret = new TypeArray(ret); } )? (name = func_name) (list = args)
                {
                        p = new Prototype(ret, name, list);
                        env.addProto(p);
                        //env.printAllProtos();
                }
        )
        func_protos
        |
        ;

        ret_type_array
        {
        }
        :LBRACKET ret_type_array
        |RBRACKET
        ;
```

```
	ret_type returns[Type t]{
		t = null;
	}
	:"string" {t = new TypeString();}
	|"file" {t = new TypeFile();}
	|"node" {t = new TypeNode();}
	|"void" {t = new TypeVoid();}
	|"int" {t = new TypeInt();}
	|"number" {t = new TypeNumber();}
	;


	func_name returns[String s]{
		s = null;
	}
	:(str:ID {s = str.getText();})
	;


	args returns[ArrayList<Variable> list]{
		Variable v;
		list = new ArrayList<Variable>();
	}
	:#( ARGS (v = arg {list.add(v);})* )
	;


	arg returns [Variable var]{
		var = null;
		Type t;
		Iden i;
	}
	:#(ARG (t = argType) (i = argId) (ret_type_array {t = new TypeArray(t);} )? {var
= new Variable(t,i);} )
	;

	argType returns[Type t]{
		t = null;
	}
	:"string" {t = new TypeString();}
	|"file" {t = new TypeFile();}
	|"node" {t = new TypeNode();}
	|"int" {t = new TypeInt();}
	|"number" {t = new TypeNumber();}
	;
```

```
argId returns[Iden i]{
        i = null;
}
:
(str:ID {i = new Iden(str.getText());})
;




program throws CompilerException
{
        out.append("import java.io.*;\n");
        out.append("import java.util.ArrayList;\n");
        out.append("import org.apache.xerces.parsers.DOMParser;\n");
        out.append("import org.w3c.dom.Document;\n");
        out.append("import org.w3c.dom.Node;\n");
        out.append("import org.w3c.dom.NodeList;\n");
        out.append("public class "+TweaXmlOut.fileName+"{\n");
        out.append(
                        "public static String add(String str1, String str2){\n"+
                        "       try{\n"+
                        "               float str1Val = Float.parseFloat(str1);\n"+
                        "               try{\n"+
                        "                       float str2Val = Float.parseFloat(str2);\n"+
                        "                       return String.valueOf(str1Val+str2Val);\n"+
                        "               }catch(Exception e1){\n"+
                        "               }\n"+
                        "       }catch(Exception e){\n"+
                        "       }\n"+
                        "       return str1.concat(str2);\n"+
                        "}\n"+
                        "\n"+
                        "public static String subtract(String str1, String str2){\n"+
                        "       float str1Val = Float.parseFloat(str1);\n"+
                        "       float str2Val = Float.parseFloat(str2);\n"+
                        "       return String.valueOf(str1Val-str2Val);\n"+
                        "}\n"+
```

```
"\n"+
"public static String multiply(String str1, String str2){\n"+
"		float str1Val = Float.parseFloat(str1);\n"+
"		float str2Val = Float.parseFloat(str2);\n"+
"		return String.valueOf(str1Val*str2Val);\n"+
"}\n"+
"\n"+
"public static String divide(String str1, String str2){\n"+
"	float str1Val = Float.parseFloat(str1);\n"+
"	float str2Val = Float.parseFloat(str2);\n"+
"	return String.valueOf(str1Val/str2Val);\n"+
"}\n"+
"public static int getLength(Node[] nodes){\n"+
"	if(nodes != null)\n"+
"		return nodes.length;\n"+
"	else\n"+
"		return 0;\n"+
"}\n"+
"\n"+
"public static int getLength(String[] nodes){\n"+
"	if(nodes != null)\n"+
"		return nodes.length;\n"+
"	else\n"+
"		return 0;\n"+
"}\n"+
"\n"+
"public static int getLength(int[] nodes){\n"+
"	if(nodes != null)\n"+
"		return nodes.length;\n"+
"	else\n"+
"		return 0;\n"+
"}\n"+
"\n"+
"public static int getLength(float[] nodes){\n"+
"	if(nodes != null)\n"+
"		return nodes.length;\n"+
"	else\n"+
"		return 0;\n"+
"}\n"+
"\n"+
```

```
"public static Node[] getChild(Node node, String nodeName){\n"+
"	if(node == null){\n"+
"		return new Node[0];\n"+
"	}\n"+
"	\n"+
"	if(nodeName.contains(\"/\")){\n"+
"		String nextChildName = nodeName.substring(0, nodeName.indexOf(\"/\"));\n"+
"		String restOfChilds = nodeName.substring(nodeName.indexOf(\"/\")+1, nodeName.length());\n"+
"		NodeList childNodes = node.getChildNodes();\n"+
"		ArrayList matchingNodesArr = new ArrayList();\n"+
"		Node[] matchingNodes = new Node[0];\n"+
"		if(childNodes != null && childNodes.getLength() > 0){\n"+
"			for(int i=0; i<childNodes.getLength(); i++){\n"+
"				Node nextChild = childNodes.item(i);\n"+
"
	if(nextChild.getNodeName().equals(nextChildName)){\n"+
"					Node[] foundNodes = getChild(nextChild, restOfChilds);\n"+
"					if(foundNodes != null && foundNodes.length > 0)\n"+
"						for(int j =0; j<foundNodes.length; j++){\n"+
"
	matchingNodesArr.add(foundNodes[j]);\n"+
"					}\n"+
"				}\n"+
"			}\n"+
"			if(matchingNodesArr.size() > 0){\n"+
"				matchingNodes = new Node[matchingNodesArr.size()];\n"+
"				for(int i=0; i<matchingNodesArr.size(); i++){\n"+
"
	matchingNodes[i] = (Node)matchingNodesArr.get(i);\n"+
"				}\n"+
"			}\n"+
```

```
                                        "                }\n"+
                                        "                return matchingNodes;\n"+
                                        "        }else{\n"+
                                        "                Node[] matchingNodes = new
Node[0];\n"+
                                        "                ArrayList matchingNodesArr = new
ArrayList();\n"+
                                        "                NodeList childNodes =
node.getChildNodes();\n"+
                                        "                if(childNodes != null &&
childNodes.getLength() > 0){\n"+
                                        "                        for(int i=0;
i<childNodes.getLength(); i++){\n"+
                                        "                                Node nextChild =
childNodes.item(i);\n"+
                                        "
        if(nextChild.getNodeName().equals(nodeName)){\n"+
                                        "
        matchingNodesArr.add(nextChild);\n"+
                                        "                                }\n"+
                                        "                        }\n"+
                                        "                        matchingNodes = new
Node[matchingNodesArr.size()];\n"+
                                        "                        for(int i=0;
i<matchingNodesArr.size(); i++){\n"+
                                        "                                matchingNodes[i] =
(Node)matchingNodesArr.get(i);\n"+
                                        "                        }\n"+
                                        "                }\n"+
                                        "                return matchingNodes;\n"+
                                        "        }\n"+
                                        "}\n"+
                                        "public static String getValue(Node node){\n"+
                                        "        if(node != null)\n"+
                                        "                return node.getTextContent();\n"+
                                        "        else\n"+
                                        "                return null;\n"+
                                        "}\n"+
                                        "public      static      Node      getRootNode(String
fileName){\n"+
                                        "    try {\n"+
                                        "      DOMParser p = new DOMParser();\n"+
                                        "      p.parse(fileName);\n"+
                                        "      Document doc = p.getDocument();\n"+
                                        "      return doc.getFirstChild();\n"+
                                        "    } catch (Exception e) {e.printStackTrace();}\n"+
```

```
                                        "          \n"+
                                        "          return null;\n"+
                                        "}\n"+
                                        "public static void main(String[] args) {\n"+
                                        "    try {\n"


        );
}

: #(CODE
        (func_protos)
        (mainBody)
        {
        }
        (func_body)
        {
        }
)
{}
;

mainBody throws CompilerException
{
        Variable dummy = null;
        String jump;
}
: #("start" {
                env.addNewScope();
        }
        mainBody RCURLY
        {   env.deleteCurrentScope();
                out.append("    } catch (Exception e) {e.printStackTrace();}\n");
                out.append("  }\n");
                out.append("}\n");
        })
| #( DECLS {declFlag = false;}declHandler {out.append("");{declFlag =
        false; }} ) mainBody
| #( ASSIGN
        {
                String lhsId = null ;
                Type rhsType = null;
                Type lhsType = null;
                SymbolTable cur = env.getCurrentScope();
                String arrExpr=null;
                Variable rExpr = null;
```

```
		}
		lhsId = strdecl (LBRACKET arrExpr = checkArrayExpr)? rExpr = rhsOfAssign
		{
			Variable lhs = Variable.newInstance(cur.get(lhsId));
			if(arrExpr!=null){
				lhsId = lhsId + "["+arrExpr+"]";
			}
			if(lhs.getVarType().getType().equals(Type.FILE) && rExpr.getVarType().getType().equals(Type.STRING)){
				out.append(CodeGen.assignment(new StringBuffer(lhsId),new StringBuffer(rExpr.getVarName().getName())));
			}
			else{
				out.append(CodeGen.assignment(new StringBuffer(lhsId),new StringBuffer(rExpr.getVarName().getName())));
			}
			out.append(CodeGen.getEndStmt());
		}
	) { }mainBody

	|#(OPSTMTS opstmts)mainBody

	|#(CONTROLSTMT controlstmts ) mainBody

	|jump = jump_stmts {
		if(!controlFlag)
		{
		}
	} mainBody

	|return_stmt

	|
	;
	return_stmt throws CompilerException
	{
		Variable curRetVar = null;
	}

	:#("return" {out.append("return "); returnFlag = true ; }
	{
	}
	(curRetVar = rhsOfAssign)?
	{
		if(curRetVar!=null){
```

```
                curRetType = curRetVar.getVarType();
                out.append(curRetVar.getVarName().getName());
            }
            else
                curRetType = null;
            out.append(CodeGen.getEndStmt());
            if(curRetType==null)
            {
                curRetType = new TypeVoid();
            }
            else
            {
            }
        }) checkformore
        ;

        checkformore throws CompilerException
        :
        | { hasMoreCode = false;}
        |mainBody { hasMoreCode = true; }
        ;


        controlstmts throws CompilerException
        {
            Variable t = null;
            String jump;
        }

        :#("if"
            {
                env.addNewScope();
                condFlag = true;
            }
            (t=rhsOfAssign)
            {
                out.append(CodeGen.ifStmt(CodeGen.intExprEval(new
StringBuffer(t.getVarName().getName()))));
                condFlag = false;
            }mainBody
            RCURLY{out.append("}\n");env.deleteCurrentScope();}(elsebody)?)

        |#("while"
            {
                env.addNewScope();
                condFlag = true;
```

```
                }
                (t=rhsOfAssign)
                {
                        out.append(CodeGen.whileStmt(CodeGen.intExprEval(new
StringBuffer(t.getVarName().getName()))));
                        condFlag = false;
                }
                mainBody RCURLY {out.append("}\n");env.deleteCurrentScope();}
        )
        {
                if(!t.getVarType().getType().equals(Type.INT))
                {
                }
        }
        ;

        jump_stmts returns[String j]
        {
                j = null;
        }
        :"continue" { out.append("continue;\n"); j = "continue"; }
        |"break" { out.append("break;\n"); j = "break" ; }
        ;
```

TweaXmlWalker.g

```
header{
        package tweaxml;
}

{
        import java.util.*;
}


/**
 * @author kk2457
 */
class TweaXmlWalker extends TreeParser;

        options {
                importVocab = TWEAXML;
```

```
		defaultErrorHandler = true;
	}
	{
		TweaXmlOut out = new TweaXmlOut();
		boolean declFlag = false;
		boolean controlFlag = false;
		boolean returnFlag = false;
		boolean hasMoreCode = false;
		int lnum = 0;
		Type curRetType = null;
		ArrayList<Type> tempForFunc = new ArrayList<Type>(1);
		Env env = new Env();
	}



	printFunc throws CompilerException
	{
		Variable var = null;
		Variable var2 = null;
		Type ltype = null;
		Type rtype = null;
	}
	:(var = getId var2 = getId)
	{
		ltype = var.getVarType();
		rtype = var2.getVarType();
		if(!ltype.getType().equals(Type.FILE) &&
			!(rtype.getType().equals(Type.STRING)
				|| rtype.getType().equals(Type.INT)
				|| rtype.getType().equals(Type.NUMBER)))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types "+ltype.getType()+" can't be used in print function'");
		}
	}
	;

	closeFunc throws CompilerException
	{
		Type t = null;
	}
	:(t = rhsOfAssign)
	{
	if(!t.getType().equals(Type.FILE))
```

```
            throw new CompilerException("Line "+lnum+": Incompatible type arguments for close statement");
        }
        ;

        declHandler throws CompilerException
        {
            SymbolTable cur = env.getCurrentScope();
            boolean isArray;
            String token = new String();
        }
        :(
            "int"
            (
                token = strdecl isArray = checkArray
                {
                    Type t = new TypeInt();
                    if(isArray)
                        t = new TypeArray(t);
                    Variable v = new Variable(t,new Iden(token));
                    if(isArray)
                        v.setInitialized();
                    cur.put(token,v);
                }
            )*
            |"number"
            (
                token = strdecl isArray = checkArray
                {
                    Type t = new TypeNumber();
                    if(isArray)
                        t = new TypeArray(t);
                    Variable v = new Variable(t,new Iden(token));
                    if(isArray)
                        v.setInitialized();
                    cur.put(token,v);
                }
            )*
            |"node"
            (
                token = strdecl isArray = checkArray
                {
                    Type t = new TypeNode();
                    if(isArray)
                        t = new TypeArray(t);
                    Variable v = new Variable(t,new Iden(token));
```

```
                        if(isArray)
                                v.setInitialized();
                        cur.put(token,v);
                }
        )*
        |"string"
        (
                token = strdecl isArray = checkArray
                {
                Type t = new TypeString();
                if(isArray)
                        t = new TypeArray(t);
                Variable v = new Variable(t,new Iden(token));
                if(isArray)
                        v.setInitialized();
                cur.put(token,v);
                }
        )*
        |"file"
        (
                token = strdecl isArray = checkArray
                {
                Type t = new TypeFile();
                if(isArray)
                        t = new TypeArray(t);
                Variable v = new Variable(t,new Iden(token));
                if(isArray)
                        v.setInitialized();
                cur.put(token,v);
                }
        )*
)
{
env.refreshCurrentScope(cur);
}
declHandler
|
;

strdecl returns [String s]
{
        s=null;
}
:str2:ID {s=str2.getText();lnum = str2.getLine();}
;
```

```
checkArray returns[boolean isArray]{
	isArray = false;
	String s;
	s = null;
}
:LBRACKET isArray = checkArray|NUMB isArray = checkArray
|RBRACKET {isArray = true;}
|{isArray = false;}
;

func_protos throws CompilerException
{
	Prototype p;
	Type ret;
	String name;
	ArrayList<Variable> list;
}
:#( FUNC_PROTOS
(ret = ret_type) (ret_type_array { ret = new TypeArray(ret); } )? (name = func_name) (list = args)
{
	p = new Prototype(ret, name, list);
	if(env.getProto(name)!=null)
		throw new CompilerException("Line "+lnum+": Prototype with name: "+name+" already exists");
	env.addProto(p);
}
)
func_protos
|
;

ret_type_array
{
}
:LBRACKET ret_type_array
|RBRACKET
;

ret_type returns[Type t]
{
```

```
		t = null;
	}
	:"string" {t = new TypeString();}
	|"file" {t = new TypeFile();}
	|"node" {t = new TypeNode();}
	|"void" {t = new TypeVoid();}
	|"int" {t = new TypeInt();}
	|"number" {t = new TypeNumber();}
	;

	func_name returns[String s]
	{
		s = null;
	}
	:(str:ID {s = str.getText();lnum = str.getLine();})
	;

	args returns[ArrayList<Variable> list]
	{
		Variable v;
		list = new ArrayList<Variable>();
	}
	:#( ARGS (v = arg {list.add(v);})* )
	;

	arg returns [Variable var]
	{
		var = null;
		Type t;
		Iden i;
	}
	:#(ARG (t = argType) (i = argId) (ret_type_array {t = new TypeArray(t);} )? {var
= new Variable(t,i);} )
	;

	argType returns[Type t]
	{
		t = null;
	}
	:"string" {t = new TypeString();}
	|"file" {t = new TypeFile();}
	|"node" {t = new TypeNode();}
	|"int" {t = new TypeInt();}
	|"number" {t = new TypeNumber();}
	;
```

```
argId returns[Iden i]
{
        i = null;
}
:
(str:ID {i = new Iden(str.getText());lnum = str.getLine();})
;




func_body throws CompilerException
{
        Prototype p=null;
        Type ret=null;
        String name=null;
        ArrayList<Variable> list=null;
        SymbolTable cur =null;
        returnFlag = false;
}
:#( FUNC_BODY { }
        (ret = ret_type) (ret_type_array { ret = new TypeArray(ret); } )? (name =
func_name) (list = args)
        {
                returnFlag = true;
                p = new Prototype(ret,name,list);
                if(!env.protoExists(p))
                {
                        throw new CompilerException("Line "+lnum+": Prototype
for the function '"+p.toString()+"' is not defined");
                }
                if(env.getProto(name).getIsdefined())
                {
                        throw new CompilerException("Line "+lnum+":
Redefinition of function - "+p.toString());
                }
                else
                {
                        p.setIsdefined();
                        env.refreshProto(p);
                }
                env.addNewScope();
                cur = env.getCurrentScope();
                for(int i=0; i<list.size(); i++)
```

```
                {
                        Variable v = list.get(i);
                        v.setInitialized();
                        cur.put(v.getVarName().getName(),v);
                }
                env.refreshCurrentScope(cur);
        }
        (mainBody)
        {
                if(hasMoreCode ==true)
                {
                        throw new CompilerException("Line "+lnum+":
Unreachable code after return statement in function "+name);
                }
                if(curRetType == null && ret.getType()!=null)
                {
                        if(!ret.getType().equals(Type.VOID))
                                throw new CompilerException("Line "+lnum+":
Return Statement for function " + name +" required");
                }
                else
                {

                        if(!ret.getType().equals(curRetType.getType()))
                        {
                                throw new CompilerException("Line "+lnum+":
Return Type mismatch for function "+name);
                        }
                }
        }
        RCURLY
        {
                hasMoreCode = false ;
                returnFlag = false;
                curRetType = null ;
                env.deleteCurrentScope();
        }
)
(func_body)
| // nothing
;
```

```
program throws CompilerException
{
}
: #(CODE
        (func_protos)
        (mainBody)
        {
        }
        (func_body)
        {
                Iterator it = env.funcProtos.keySet().iterator();
                while(it.hasNext())
                {
                        String pname = (String)it.next();
                        Prototype pt = env.funcProtos.get(pname);
                        if(pt.isUsed() && !pt.getIsdefined())
                        {
                                throw new CompilerException("Line "+lnum+":
Function "+pt.getProtoName()+" has been used but not defined");
                        }
                }
        }
)
{}
;




mainBody throws CompilerException
{
        Variable dummy = null;
        String jump;
}
: #("start"
        {
                env.addNewScope();
        }
        mainBody RCURLY {env.deleteCurrentScope();})
        | #( DECLS {declFlag = false;}declHandler {

                {declFlag = false; }
```

```
		} ) mainBody
	| #( ASSIGN
	{
		String lhsId = null ;
		Type rhsType = null;
		Type lhsType = null;
		SymbolTable cur = env.getCurrentScope();
		boolean arrFlag=false;
		Variable v = null;
	}
	lhsId = strdecl (LBRACKET arrFlag = checkArrayExpr)?
	{
		v = Variable.newInstance(cur.get(lhsId));
		if(v == null)
		{
			throw new CompilerException("Line "+lnum+": Variable
'"+lhsId+"' not declared");
		}
		lhsType = Type.newInstance(v.getVarType());
		if(arrFlag)
		{
			if(!lhsType.getType().endsWith("]"))
				throw new CompilerException("Line "+lnum+":
Non array variable "+v.getVarName().getName()+" being indexed");
			if(lhsType.getType().startsWith(Type.INT))
				lhsType = new TypeInt();
			if(lhsType.getType().startsWith(Type.NUMBER))
				lhsType = new TypeNumber();
			if(lhsType.getType().startsWith(Type.NODE))
				lhsType = new TypeNode();
			if(lhsType.getType().startsWith(Type.STRING))
				lhsType = new TypeString();
			if(lhsType.getType().startsWith(Type.FILE))
				lhsType = new TypeFile();
		}
	}
	(rhsType = rhsOfAssign)
	{
		if(lhsType.getType().equals(Type.FILE)
&&
rhsType.getType().equals(Type.STRING))
		{
		}
		else
		{
			if(!lhsType.getType().equals(rhsType.getType()))
			{
```

```
                                        throw new CompilerException("Line "+lnum+": Type mismatch between "+lhsType.getType()+" and "+rhsType.getType()+" for =");
                                }
                        }
                        v.setInitialized();
                        cur.replaceVar(lhsId,v);
                        env.refreshCurrentScope(cur);
                }
        ) { }

        mainBody
        |#(OPSTMTS opstmts)mainBody
        |#(CONTROLSTMT controlstmts ) mainBody
        |(dummy = getId) mainBody
        |jump = jump_stmts
        {
                if(!controlFlag)
                {
                        throw new CompilerException("Line "+lnum+": Statement "+jump+" is wrongly used here, allowed only in while block");
                }
        } mainBody
        |return_stmt
        |
        ;




                return_stmt throws CompilerException
        {
        }
        :#("return"
                {
                        if(!returnFlag)
                        {
                                throw new CompilerException("Line "+lnum+": Return Statement is used wrongly. Only allowed inside user defined function bodies.");
                        }
                }
                (curRetType = rhsOfAssign)?
                {
                        if(curRetType==null)
                        {
                                curRetType = new TypeVoid();
```

```
                }
                else
                {
                }
        }
) checkformore
;


checkformore throws CompilerException
:

| { hasMoreCode = false;}

|mainBody { hasMoreCode = true; }
;




controlstmts throws CompilerException
{
        Type t=null;
        String jump;
}
:#("if"
        {
                env.addNewScope();
        }
        (t=rhsOfAssign)
        {
                if(!t.getType().equals(Type.INT))
                {
                        throw new CompilerException("Line "+lnum+": Type
mismatch in if conditional");
                }
        }
        mainBody RCURLY{env.deleteCurrentScope();}(elsebody)?
)
|#("while"
        {
                controlFlag = true;
                env.addNewScope();
        }
```

```
		t=rhsOfAssign mainBody RCURLY {controlFlag = false;env.deleteCurrentScope();}
	)
	{
		if(!t.getType().equals(Type.INT))
		{
			throw new CompilerException("Line "+lnum+": Type mismatch in while conditional");
		}
	}
	;

	jump_stmts returns[String j]
	{
		j = null;
	}
	:"continue" { j = "continue"; }
	|"break" { j = "break" ; }
	;

	elsebody throws CompilerException
	:#(ELSESTMT elsebody)
	|#("else" { env.addNewScope();} mainBody RCURLY
	{env.deleteCurrentScope();})
	//|//mainBody
	;




	opstmts throws CompilerException
	{
		Variable var = null;
		Variable var2 = null;
		Type ltype = null;
		Type rtype = null;
	}
	:#("print" printFunc)
	| #("close" closeFunc)
	;
```

```
	rhsOfAssign returns [Type typ] throws CompilerException
	{
		typ=null;
		Variable var = null;
		Variable var2 = null;
		Type rtype = null;
		Type ltype = null;
		ArrayList<Type> funcArgTypes = null;
	}
	:#(PLUS ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		typ = ltype.getCompatibleType(rtype, "+");
		if(typ == null)
			throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator +");
	}

	|#(MINUS ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		typ = ltype.getCompatibleType(rtype, "-");
		if(typ == null)
			throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator -");
	}

	|#(TIMES ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		typ = ltype.getCompatibleType(rtype, "*");
		if(typ == null)
			throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator *");
	}
	|#(DIV ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		typ = ltype.getCompatibleType(rtype, "/");
		if(typ == null)
			throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator /");
	}

	|#(AND ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
		{
```

```
                throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator &&");
                }
                typ = new TypeInt();
        }

        |#(OR ltype = rhsOfAssign rtype = rhsOfAssign)
        {
                if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
                {
                        throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator ||");
                }
                typ = new TypeInt();
        }

        |#(LT ltype = rhsOfAssign rtype = rhsOfAssign)
        {
                if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
                {
                        throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator <");
                }
                typ = new TypeInt();
        }

        |#(LEQ ltype = rhsOfAssign rtype = rhsOfAssign)
        {
                if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
                {
                        throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator <=");
                }
                typ = new TypeInt();
        }

        |#(GT ltype = rhsOfAssign rtype = rhsOfAssign)
        {
                if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
                {
                        throw new CompilerException("Line "+lnum+": Incompatible types"+ltype.getType()+", "+rtype.getType()+" for operator >");
                }
                typ = new TypeInt();
        }
```

```
	|#(GEQ ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
		{
			throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator >=");
		}
		typ = new TypeInt();
	}

	|#(EQ ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
		{

	if(!ltype.getType().equals(Type.STRING)||!rtype.getType().equals(Type.STRING))
				throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator ==");
		}
		typ = new TypeInt();
	}

	|#(INEQ ltype = rhsOfAssign rtype = rhsOfAssign)
	{
		if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.INT))
		{

	if(!ltype.getType().equals(Type.STRING)||!rtype.getType().equals(Type.STRING))
				throw new CompilerException("Line "+lnum+": Incompatible types "+ltype.getType()+", "+rtype.getType()+" for operator !=");
		}
		typ = new TypeInt();
	}

	|#("length" var = getId)
	{
		rtype = var.getVarType();
		if(!rtype.getType().endsWith("[]"))
		{
			throw new CompilerException("Line "+lnum+": Incompatible types '"+rtype.getType()+"' for length statement");
		}
		typ = new TypeInt();
	}
```

```
	|#("create" var = getId)
	{
		rtype = var.getVarType();
		if(!(rtype.getType().equals(Type.STRING)))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types "'+rtype.getType()+"' for create statement");
		}
		typ = new TypeFile();
	}

	|#("open" var = getId)
	{
		rtype = var.getVarType();
		if(!(rtype.getType().equals(Type.STRING)))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types "'+rtype.getType()+"' for open statement");
		}
		typ = new TypeNode();
	}

	|#("contains" var = getId var2=getId)
		{
			ltype = var.getVarType();
			rtype = var2.getVarType();
			if(!ltype.getType().equals(Type.NODE)                    ||
!rtype.getType().equals(Type.STRING))
			{
				throw new CompilerException("Line "+lnum+":
Incompatible types "+ltype.getType()+","+rtype.getType()+" for contains statement");
			}
		}
	|#("add" var = getId var2 = getId)
	{
		ltype = var.getVarType();
		rtype = var2.getVarType();
		if(!ltype.getType().equals(Type.STRING)                    ||
!rtype.getType().equals(Type.STRING))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types "'+ltype.getType()+", "+rtype.getType()+"' for add statement");
		}
		typ = new TypeString();
	}
```

```
	|#("subtract" var = getId var2 = getId)
	{
		ltype = var.getVarType();
		rtype = var2.getVarType();
		if(!ltype.getType().equals(Type.STRING) ||
!rtype.getType().equals(Type.STRING))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types '"+ltype.getType()+", "+rtype.getType()+"' for subtract statement");
		}
		typ = new TypeString();
	}
	|#("multiply" var = getId var2 = getId)
	{
		ltype = var.getVarType();
		rtype = var2.getVarType();
		if(!ltype.getType().equals(Type.STRING) ||
!rtype.getType().equals(Type.STRING))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types '"+ltype.getType()+", "+rtype.getType()+"' for multiply statement");
		}
		typ = new TypeString();
	}
	|#("divide" var = getId var2 = getId)
	{
		ltype = var.getVarType();
		rtype = var2.getVarType();
		if(!ltype.getType().equals(Type.STRING) ||
!rtype.getType().equals(Type.STRING))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types '"+ltype.getType()+", "+rtype.getType()+"' for divide statement");
		}
		typ = new TypeString();
	}
	|#("getchild" var = getId var2 = getId)
	{
		ltype = var.getVarType();
		rtype = var2.getVarType();
		if(!ltype.getType().equals(Type.NODE) ||
!rtype.getType().equals(Type.STRING))
		{
			throw new CompilerException("Line "+lnum+": Incompatible
types '"+ltype.getType()+", "+rtype.getType()+"' for getchild statement");
		}
```

```
                typ = ltype.getCompatibleType(rtype, "getchild");
        }
        |#("getvalue" var = getId)
        {
                rtype = var.getVarType();
                if(!rtype.getType().equals(Type.NODE))
                {
                        throw new CompilerException("Line "+lnum+": Incompatible
types '"+rtype.getType()+"' for getvalue statement");
                }
                typ = new TypeString();
        }

        |#(UMINUS rtype = rhsOfAssign)
        {
                if(!rtype.getType().equals(Type.INT))
                {
                        throw new CompilerException("Line "+lnum+": Wrong type for
the argument of Unary Minus ~");
                }
                typ = new TypeInt();
        }

        |#(NOT rtype = rhsOfAssign)
        {
                if(!rtype.getType().equals(Type.INT))
                {
                        throw new CompilerException("Line "+lnum+": Wrong type for
the argument of Operator NOT !");
                }
                typ = new TypeInt();
        }

        |(var = getId)
        {
                typ = var.getVarType();
        }
        ;
```

```
	getId returns [Variable v] throws CompilerException
	{
		String id = null;
		SymbolTable cur =env.getCurrentScope();
		boolean isItArray = false;
		boolean isFunc = false ;
		ArrayList<Type> func = new ArrayList<Type>(1);
		v = null;
		ArrayList<Type> argList = null;
	}
	:str:ID ((LBRACKET isItArray = checkArrayExpr) | (LPAREN {isFunc = true;}
argList = checkFunc))?
	{
		if(isFunc)
		{
			id = str.getText();
			lnum = str.getLine();
			if(env.getProto(id)==null)
			{
				throw new CompilerException("Line "+lnum+": Prototype
for the function "+id+" has not been declared before start");
			}
			Prototype pt = env.getProto(id);
			pt.setUsed(true);
			env.refreshProto(pt);
			ArrayList<Variable> pvars = pt.getArgList();
			if(argList == null)
				throw new CompilerException("Line "+lnum+":
Debugging!!!--> argList becomes null!!");
			if(pvars.size()!=argList.size())
			{
				throw new CompilerException("Line "+lnum+": The
number of function arguments for "+id+" do not match with function prototype");
			}
			for(int i=0;i<argList.size();i++)
			{
				if(argList.get(i) == null)
					System.out.println(i+" is null!!!!!!!!!!!!!!!");

	if(!argList.get(i).getType().equals(pvars.get(i).getVarType().getType()))
				{
					throw new CompilerException("Line "+lnum+":
	The types of function call arguments for the function "+id+" do not match with
the types in function prototype");
				}
			}
```

```
                        v = new Variable(Type.newInstance(pt.getReturnType()),new
                        Iden(id));
                        v.setInitialized();
                        argList.clear();
                }
                else
                {
                        id = str.getText();
                        lnum = str.getLine();
                        v = Variable.newInstance(cur.get(id));
                        if(v == null)
                                throw new CompilerException("Line "+lnum+": Variable
'"+id+"'not declared");
                        if(!v.isInitialized())
                                throw new CompilerException("Line "+lnum+": Variable
'"+id+"' may not have been initialized");
                        if(isItArray)
                        {
                                if(!v.getVarType().getType().endsWith("]"))
                                        throw new CompilerException("Line "+lnum+":
Non array variable "+v.getVarName().getName()+" being indexed");
                                if(v.getVarType().getType().startsWith(Type.INT))
                                        v.setVarType(new TypeInt());
                                else
                                {

        if(v.getVarType().getType().startsWith(Type.STRING))
                                                v.setVarType(new TypeString());
                                        else
                                        {

        if(v.getVarType().getType().startsWith(Type.NODE))
                                                        v.setVarType(new TypeNode());

        if(v.getVarType().getType().startsWith(Type.FILE))
                                                        v.setVarType(new TypeFile());
                                        }
                                }
                        }
                }
        }

        |str2:NUMB {v = new Variable(new TypeInt(),new
Iden(str2.getText()));v.setInitialized();lnum = str2.getLine();}
```

```
	|str3:STRCON {v = new Variable(new TypeString(),new Iden(str3.getText()));
v.setInitialized();lnum = str3.getLine();}
	;



	checkFunc returns[ArrayList<Type> args]throws CompilerException
	{
		Type t = null;
		args = new ArrayList<Type>();
		ArrayList<Type> temp = new ArrayList<Type>();
	}
	:RPAREN
	{
	}

	|(t=rhsOfAssign)
	{
		if(t == null)
			System.out.println("in checkFunc! : got type = NULL!!!");
		args.add(t);
	} (temp = checkFunc)
	{
		for(int i=0;i<temp.size();i++)
			args.add(temp.get(i));
	}
	;




	checkArrayExpr returns[boolean b] throws CompilerException
	{
		b=false;
		Type t = null;
	}
	:RBRACKET {b=true;}
	|(t = rhsOfAssign)
	{
		if(!t.getType().equals(Type.INT))
		{
			throw new CompilerException("Line "+lnum+": Expression inside
array not int");
		}
	}
	(b=checkArrayExpr)
```

;

CodeGen.java

```
package tweaxml;

import java.util.ArrayList;

/**
 * @author kk2457
 *
 */
public class CodeGen {

	public static StringBuffer getLParen() {
		return new StringBuffer("(");
	}

	public static StringBuffer getRParen() {
		return new StringBuffer(")");
	}

	public static StringBuffer getLCurly() {
		return new StringBuffer("{");
	}

	public static StringBuffer getRCurly() {
		return new StringBuffer("}");
	}

	public static StringBuffer getLBracket() {
		return new StringBuffer("[");
	}

	public static StringBuffer getRBracket() {
		return new StringBuffer("]");
	}

	public static StringBuffer getEndStmt() {
		return new StringBuffer(";\n");
	}

	public static StringBuffer whileStmt(StringBuffer expr) {
```

```
        return new StringBuffer("while(" + expr + ") {\n");
}

public static StringBuffer ifStmt(StringBuffer expr) {
        return new StringBuffer("if(" + expr + ") {\n");
}

public static StringBuffer arrayLength(StringBuffer varname) {
        StringBuffer foo = new StringBuffer();
        return foo.append("getLength("+varname+"); \n");
}

public static StringBuffer print(StringBuffer p) {
        StringBuffer foo = new StringBuffer();
        return foo.append("System.out.print(").append(p).append(");\n");
}

public static StringBuffer printInFile(StringBuffer p1, StringBuffer p2) {
        StringBuffer foo = new StringBuffer();
        return foo.append(p1+".write("+p2+");\n");
}

public static StringBuffer add(StringBuffer p1, StringBuffer p2) {
        StringBuffer foo = new StringBuffer();
        return foo.append("add("+p1+", "+p2+");\n");
}

public static StringBuffer subtract(StringBuffer p1, StringBuffer p2) {
        StringBuffer foo = new StringBuffer();
        return foo.append("subtract("+p1+", "+p2+");\n");
}

public static StringBuffer multiply(StringBuffer p1, StringBuffer p2) {
        StringBuffer foo = new StringBuffer();
        return foo.append("multiply("+p1+", "+p2+");\n");
}

public static StringBuffer divide(StringBuffer p1, StringBuffer p2) {
        StringBuffer foo = new StringBuffer();
        return foo.append("divide("+p1+", "+p2+");\n");
}

public static StringBuffer closeFile(StringBuffer p) {
        StringBuffer foo = new StringBuffer();
        return foo.append(p+".flush();\n"+p+".close();\n");
}
```

```
	public static StringBuffer println(StringBuffer p) {
		StringBuffer foo = new StringBuffer();
		return foo.append("System.out.println(").append(p).append(");\n");
	}

	public static StringBuffer assignment(StringBuffer lexpr, StringBuffer rexpr) {
		StringBuffer foo = new StringBuffer();
		foo.append(lexpr);
		foo.append("=");
		foo.append(rexpr);
		return foo;
	}

	public static StringBuffer fileCreate(StringBuffer expr) {
		StringBuffer foo = new StringBuffer();
		foo.append("new BufferedWriter(new FileWriter(new File(" + expr +
"")))");
		return foo;
	}

	public static StringBuffer fileOpen(StringBuffer expr) {
		StringBuffer foo = new StringBuffer();
		foo.append("getRootNode(" + expr + ")");
		return foo;
	}


	public static StringBuffer varDeclaration(StringBuffer type,
			StringBuffer varname) {
		StringBuffer foo = new StringBuffer();
		if (type.toString().equals(Type.INT))
			foo.append("int " + varname.toString());
		if (type.toString().equals(Type.NUMBER))
			foo.append("float " + varname.toString());
		if (type.toString().equals(Type.STRING))
			foo.append("String " + varname.toString());
		if (type.toString().equals(Type.FILE))
			foo.append("BufferedWriter " + varname.toString() + "= null");
		if (type.toString().equals(Type.NODE))
			foo.append("Node " + varname.toString() + "= null");
		return foo;
	}

	public static StringBuffer arrayDecl(StringBuffer type,
			StringBuffer varname, StringBuffer array_dim) {
```

```
		StringBuffer foo = new StringBuffer();
		if (type.toString().equals(Type.INT)) {
			foo.append(Type.JAVA_INT + "[] " + varname.toString());
			foo.append(" = ");
			foo.append("new " + Type.JAVA_INT + "[" + array_dim + "]");
		}
		if (type.toString().equals(Type.STRING)) {
			foo.append(Type.JAVA_STRING + "[] " + varname.toString());
			foo.append(" = ");
			foo.append("new " + Type.JAVA_STRING + "[" + array_dim +
"]");
		}
		if (type.toString().equals(Type.NODE)) {
			int dim = -1;
			if(array_dim != null){
				try{
					dim = Integer.parseInt(array_dim.toString());
				}catch(Exception e){}
			}
			if(dim > 0){
				foo.append("Node[] " + varname.toString()+" = new
Node["+dim+"]");
			}else{
				foo.append("Node[] " + varname.toString()+" = null");
			}
		}
		return foo;
	}

	public static StringBuffer getChildStmt(StringBuffer lexpr, StringBuffer rexpr){
		StringBuffer foo = new StringBuffer();
		foo.append("getChild("+lexpr+", "+rexpr+")");
		return foo;
	}

	public static StringBuffer getValueStmt(StringBuffer lexpr){
		StringBuffer foo = new StringBuffer();
		foo.append("getValue("+lexpr+")");
		return foo;
	}

	public static StringBuffer mainStmt() {
		StringBuffer foo = new StringBuffer();
		foo
			.append("public static void main(String[] args) throws
Exception {\n");
```

```
        return foo;
}

public static StringBuffer expressionGen(StringBuffer lexpr,
                StringBuffer rexpr, StringBuffer operator) {
        StringBuffer foo = new StringBuffer();
        foo.append(lexpr);
        foo.append(operator);
        foo.append(rexpr);
        return foo;
}

public static StringBuffer unaryMinus(StringBuffer expr) {
        StringBuffer foo = new StringBuffer();
        foo.append("-").append(expr);
        return foo;
}

public static StringBuffer unaryNotBool(StringBuffer expr) {
        StringBuffer foo = new StringBuffer();
        foo.append("(" + expr + "==0)");
        return foo;
}

public static StringBuffer unaryNotInt(StringBuffer expr) {
        StringBuffer foo = new StringBuffer();
        foo.append("(" + expr + "!=0)?0:1");
        return foo;
}

public static StringBuffer elseStmt() {
        StringBuffer foo = new StringBuffer();
        foo.append("else{\n");
        return foo;
}

public static StringBuffer functionCall(StringBuffer name,
                ArrayList<Variable> list) {
        StringBuffer foo = new StringBuffer();
        foo.append(name + "(");
        for (int i = 0; i < list.size(); i++) {
                Variable v = list.get(i);
                foo.append(v.getVarName().getName());
                if (i != list.size() - 1)
                        foo.append(",");
        }
```

```
        foo.append(")");
        return foo;
}

public static StringBuffer functionBody(StringBuffer retType,
                StringBuffer name, ArrayList<Variable> list)
                throws CompilerException {
        StringBuffer foo = new StringBuffer();
        foo.append("static " + retType + " " + name + "(");
        for (int i = 0; i < list.size(); i++) {
                Variable v = list.get(i);
                foo.append(v.getJavaType() + " ");
                foo.append(v.getVarName().getName());
                if (i != list.size() - 1)
                        foo.append(",");
        }
        foo.append(") throws Exception{\n");
        return foo;
}

public static StringBuffer stringEqualityInt(StringBuffer lexpr,
                StringBuffer rexpr) {
        StringBuffer foo = new StringBuffer();
        foo.append("(" + lexpr + ".equals(");
        foo.append(rexpr + " )");
        foo.append(")?1:0");
        return foo;
}

public static StringBuffer stringInequalityInt(StringBuffer lexpr,
                StringBuffer rexpr) {
        StringBuffer foo = new StringBuffer();
        foo.append("(" + lexpr + ".equals(");
        foo.append(rexpr + " )");
        foo.append(")?0:1");
        return foo;
}

public static StringBuffer stringEqualityBool(StringBuffer lexpr,
                StringBuffer rexpr) {
        StringBuffer foo = new StringBuffer();
        foo.append(lexpr + ".equals(");
        foo.append(rexpr + " )");
        return foo;
}
```

```
public static StringBuffer stringInequalityBool(StringBuffer lexpr,
                StringBuffer rexpr) {
        StringBuffer foo = new StringBuffer();
        foo.append("!" + lexpr + ".equals(");
        foo.append(rexpr + " )");
        return foo;
}

public static StringBuffer expressionGenBoolToInt(StringBuffer lexpr,
                StringBuffer rexpr, StringBuffer operator) {
        StringBuffer foo = new StringBuffer();
        if (operator.toString().equals("&&")) {
                foo.append("( (" + lexpr + "!= 0)&&(" + rexpr + " != 0) )?1:0");
                return foo;
        }
        if (operator.toString().equals("||")) {
                foo.append("( (" + lexpr + "!= 0)||(" + rexpr + " != 0) )?1:0");
                return foo;
        }
        foo.append("(" + lexpr);
        foo.append(operator);
        foo.append(rexpr + ")");
        foo.append("?1:0");
        return foo;
}

public static StringBuffer expressionGenIntToBool(StringBuffer lexpr,
                StringBuffer rexpr, StringBuffer operator) {
        StringBuffer foo = new StringBuffer();
        foo.append("( (" + lexpr + operator + rexpr + ") != 0 )");
        return foo;
}

public static StringBuffer intExprEval(StringBuffer expr) {
        if (!isBooleanExpr(expr)) {
                StringBuffer foo = new StringBuffer();
                foo.append("(" + expr + ") != 0");
                return foo;
        }
        return expr;
}

private static boolean isBooleanExpr(StringBuffer expr) {
        if ((expr.toString().lastIndexOf("<") == -1)
                        && (expr.toString().lastIndexOf("<=") == -1)
                        && (expr.toString().lastIndexOf(">") == -1)
```

```
                                && (expr.toString().lastIndexOf(">=") == -1)
                                && (expr.toString().lastIndexOf("&&") == -1)
                                && (expr.toString().lastIndexOf("||") == -1)
                                && (expr.toString().lastIndexOf("!") == -1)
                                && (expr.toString().lastIndexOf("==") == -1))
                        return false;
                return true;
        }
}
```

CompilerException.java

```
package tweaxml;

        /**
         * @author kk2457
         */
public class CompilerException extends Exception {
        private static final long serialVersionUID = -5159407844883712 46L;

        public CompilerException(String message) {
                super(message);
        }

}
```

Env.java

```
package tweaxml;

import java.util.Enumeration;
import java.util.Stack;
import java.util.Hashtable;

/**
 * @author kk2457
 */
public class Env {
        Hashtable<String, Prototype> funcProtos;

        Stack<SymbolTable> stackTable;

        public Env() {
                funcProtos = new Hashtable<String, Prototype>();
```

```
		stackTable = new Stack<SymbolTable>();
	}

	public void addProto(Prototype p) {
		if (p == null || p.getProtoName() == null || p.getProtoName() == "") {
			System.out.println("Invalid parameters in Prototype");
			return;
		}
		funcProtos.put(p.getProtoName(), p);
	}

	public Prototype getProto(String name) {
		return funcProtos.get(name);
	}

	public void refreshProto(Prototype p) {
		addProto(p);
	}

	public boolean protoExists(Prototype p) {
		Prototype in = getProto(p.getProtoName());
		if (in == null)
			return false;
		if (!p.getReturnType().getType().equals(in.getReturnType().getType()))
			return false;
		if (p.getArgSize() != in.getArgSize())
			return false;
		for (int i = 0; i < p.getArgSize(); i++) {
			Variable ofp = p.getArgAtPosition(i);
			Variable ofin = in.getArgAtPosition(i);
			if
(!ofp.getVarType().getType().equals(ofin.getVarType().getType()))
				return false;
		}
		return true;
	}

	public void printAllProtos() {
		Enumeration<Prototype> e = funcProtos.elements();
		while (e.hasMoreElements()) {
			Prototype p = e.nextElement();
			System.out.println(p.toString());
		}
	}

	public void addNewScope() {
```

```
		SymbolTable st;
		SymbolTable outer;
		if (stackTable.empty())
			outer = null;
		else
			outer = stackTable.peek();
		st = new SymbolTable(outer);
		stackTable.push(st);
	}

	public SymbolTable deleteCurrentScope() {
		if (!stackTable.empty())
			return stackTable.pop();
		return null;
	}

	public SymbolTable getCurrentScope() {
		if (!stackTable.empty())
			return stackTable.peek();
		return null;
	}

	public void refreshCurrentScope(SymbolTable tab) {
		this.deleteCurrentScope();
		this.stackTable.push(tab);
	}

	public void printAllScopes() {
		if (stackTable.empty()) {
			System.out.println("No scopes present!!");
			return;
		}
		System.out.println("current size of stack = " + stackTable.size());
		for (int i = 0; i < stackTable.size(); i++) {
			System.out.println("Scope: " + i);
			SymbolTable t = stackTable.get(i);
			t.printTable();
			System.out.println();
		}
	}
}
```

Iden.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class Iden {
	private String name;

	public Iden(String n) {
		this.name = n;
	}

	public String getName() {
		return name;
	}

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

	public static Iden newInstance(Iden i) {
		Iden result = new Iden(i.getName());
		return result;
	}
}
```

Prototype.java

```
package tweaxml;

import java.util.ArrayList;

/**
 * @author kk2457
 */
public class Prototype {
	private Type return_type;

	private ArrayList<Variable> arg_list;

	private String protoName;

	private boolean isdefined;
```

```
private boolean isUsed;

public Prototype(Type t, String name, ArrayList<Variable> list) {
	this.return_type = t;
	this.arg_list = list;
	this.protoName = name;
	this.isdefined = false;
	this.isUsed = false;
}

public Type getReturnType() {
	return return_type;
}

public ArrayList<Variable> getArgList() {
	return arg_list;
}

public void setReturnType(Type t) {
	return_type = t;
}

public void setArgList(ArrayList<Variable> t) {
	arg_list = t;
}

public void addArgument(Variable t) {
	arg_list.add(t);
}

public int getArgSize() {
	return arg_list.size();
}

public Variable getArgAtPosition(int position) {
	if (position > this.getArgSize() - 1)
		return null;
	return arg_list.get(position);
}

public String getProtoName() {
	return protoName;
}

public void setProtoName(String protoName) {
	this.protoName = protoName;
```

```
	}

	public boolean getIsdefined() {
		return isdefined;
	}

	public void setIsdefined() {
		this.isdefined = true;
	}

	public String toString() {
		StringBuffer res = new StringBuffer();
		res.append(return_type.getType() + " ");
		res.append(protoName + "( ");
		Variable v;
		int i;
		for (i = 0; i < arg_list.size() - 1; i++) {
			v = arg_list.get(i);
			res.append(v.getVarType().getType() + " ");
			res.append(v.getVarName().getName() + ", ");
		}
		if (i < arg_list.size()) {
			v = arg_list.get(i);
			res.append(v.getVarType().getType() + " ");
			res.append(v.getVarName().getName() + " ");
		}
		res.append(");");
		return res.toString();
	}

	public boolean isUsed() {
		return isUsed;
	}

	public void setUsed(boolean isUsed) {
		this.isUsed = isUsed;
	}
}
```

SymbolTable.java

```
package tweaxml;

import java.util.Enumeration;
```

```
import java.util.Hashtable;

/**
 * @author kk2457
 */
public class SymbolTable {
	private Hashtable<String, Variable> table;

	protected SymbolTable outer;

	public SymbolTable(SymbolTable outer) {
		table = new Hashtable<String, Variable>();
		this.outer = outer;
	}

	public SymbolTable() {
	}

	public void put(String token, Variable v) throws CompilerException {
		if (get(token) != null) {
			throw new CompilerException("Variable " + token
					+ " already declared");
		}
		this.table.put(token, v);
	}

	public void replaceVar(String token, Variable v) {
		if (this.table.containsKey(token))
			this.table.remove(token);
		this.table.put(token, v);
	}

	public Variable get(String token) {
		for (SymbolTable tab = this; tab != null; tab = tab.outer) {
			Variable id = tab.table.get(token);
			if (id != null)
				return id;
		}
		return null;
	}

	public Variable getFromCurrent(String token) {
		return this.table.get(token);
	}

	public void printTable() {
```

```
                Enumeration<String> en = table.keys();
                while (en.hasMoreElements()) {
                        String str = en.nextElement();
                        Variable v = table.get(str);
                        System.out.println(v.toString());
                }
        }
}
```

TweaXmlAST.java

```
package tweaxml;

import antlr.CommonAST;
import antlr.Token;

/**
 *
 * @author kk2457
 *
 */
public class TweaXmlAST extends CommonAST {
        private static final long serialVersionUID = -1936758840206008598L;
        private int line = 0;
        private int column = 0;

        public void initialize(Token tok) {
                super.initialize(tok);
                this.line = tok.getLine();
                this.column = tok.getColumn();
        }

        public int getLine() {
                return this.line;
        }

        public int getColumn() {
                return this.column;
        }
}
```

TweaXmlOut.java

```
package tweaxml;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.StringTokenizer;

/**
 *
 * @author kk2457
 *
 */
public class TweaXmlOut {
	private StringBuffer out;

	public static String origFileName;

	public static String fileName;

	public static String outName;

	public TweaXmlOut() {
		out = new StringBuffer();
	}

	public void append(String str) {
		out.append(str);
	}

	public void append(StringBuffer str) {
		out.append(str);
	}

	public static void initFileName(String name) {
		StringTokenizer st = new StringTokenizer(name, ".");
		fileName = new String(st.nextToken());
		origFileName = fileName;
		int ind = fileName.lastIndexOf(File.separator);
		if (ind > -1) {
			fileName = fileName.substring(ind + 1);
		}
	}
```

```
	public StringBuffer getOut() {
		return out;
	}

	public void writeToJava() {
		outName = fileName + ".java";
		try {
			FileOutputStream fo = new FileOutputStream(outName);
			PrintStream p = new PrintStream(fo);
			p.print(this.out.toString());
		} catch (FileNotFoundException e) {
			System.out.println("Cannot create output file");
			System.exit(0);
			e.printStackTrace();
		}
	}
}
```

Type.java

```
package tweaxml;

/**
 * @author kk2457
 */
public abstract class Type {
	public static final String INT = "int";

	public static final String NUMBER = "number";

	public static final String STRING = "string";

	public static final String NODE = "node";


	public static final String FILE = "file";

	public static final String VOID = "void";

	public static final String JAVA_INT = "int";

	public static final String JAVA_STRING = "String";
```

```
	public static final String JAVA_VOID = "void";

	protected String type;

	public String getType() {
		return this.type;
	}

	public static Type newInstance(Type t) {
		Type result = null;
		String type = t.getType();
		if (type.startsWith(Type.INT))
			result = new TypeInt();
		if (type.startsWith(Type.NUMBER))
			result = new TypeNumber();
		if (type.startsWith(Type.FILE))
			result = new TypeFile();
		if (type.startsWith(Type.NODE))
			result = new TypeNode();
		if (type.startsWith(Type.VOID)) {
			result = new TypeVoid();
			return result;
		}
		if (type.startsWith(Type.STRING))
			result = new TypeString();
		if (type.endsWith("[]"))
			result = new TypeArray(result);
		return result;
	}

	public Type implicitCast(Type t1, Type t2) throws CompilerException {
		if ((t1.getType() == Type.STRING && t2.getType() == Type.INT)
				|| (t1.getType() == Type.INT && t2.getType() ==
Type.STRING))
			return new TypeString();
		else
			throw new CompilerException("Invalid implicit type cast");
	}

	public Type getCompatibleType(Type rtype, String op) {
		String ltyp = this.getType();
		String rtyp = rtype.getType();
		if (op.equals("+")) {
			if (ltyp.equals(Type.INT)) {
				if (rtyp.equals(Type.INT))
					return new TypeInt();
```

```
                        if (rtyp.equals(Type.STRING))
                            return new TypeString();
                    }
                    if (ltyp.equals(Type.STRING)) {
                        if (rtyp.equals(Type.INT))
                            return new TypeString();
                        if (rtyp.equals(Type.STRING))
                            return new TypeString();
                    }
                    if (ltyp.equals(Type.FILE)) {
                        if (rtyp.equals(Type.FILE))
                            return new TypeFile();
                    }
                    if (ltyp.equals(Type.NODE)) {
                        if (rtyp.equals(Type.NODE))
                            return new TypeNode();
                    }
                }
                if (op.equals("-") || op.equals("*") || op.equals("/")) {
                    if (ltyp.equals(Type.INT)) {
                        if (rtyp.equals(Type.INT))
                            return new TypeInt();
                    }
                }
                if (op.equals("getchild")) {
                    if (ltyp.equals(Type.STRING)) {
                        if (rtyp.equals(Type.STRING))
                            return new TypeInt();
                        if (rtyp.equals(Type.NODE)) {
                            Type t = new TypeNode();
                            return new TypeArray(t);
                        }
                    }
                    if (ltyp.equals(Type.NODE)) {
                        Type t = new TypeNode();
                        return new TypeArray(t);
                    }
                }
                return null;
        }
}
```

TypeArray.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class TypeArray extends Type {
	public static final String TYPE_AR_INT = "int[]";

	public static final String TYPE_AR_STRING = "string[]";

	public static final String TYPE_AR_NODE = "node[]";

	public TypeArray(Type t) {
		this.type = t.getType() + "[]";
	}
}
```

TypeFile.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class TypeFile extends Type{
	public TypeFile() {
		this.type = Type.FILE;
	}
}
```

TypeInt.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class TypeInt extends Type {
	public TypeInt() {
```

```
                this.type = Type.INT;
        }
}
```

TypeNode.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class TypeNode extends Type{
        public TypeNode() {
                this.type = Type.NODE;
        }

}
```

TypeNumber.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class TypeNumber extends Type {
        public TypeNumber() {
                this.type = Type.NUMBER;
        }

}
```

TypeString.java

```
package tweaxml;
```

```
/**
 * @author kk2457
 */
public class TypeString extends Type {
	public TypeString() {
		this.type = Type.STRING;
	}
}
```

TypeVoid.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class TypeVoid extends Type {
	public TypeVoid() {
		this.type = Type.VOID;
	}
}
```

Variable.java

```
package tweaxml;

/**
 * @author kk2457
 */
public class Variable {
	private Type varType;
	private Iden varName;
	private boolean valPresent;

	public Variable(Type t, Iden i){
		varType = t;
		varName = i;
		valPresent = false;
	}
```

```
	public static Variable newInstance(Variable v){
		if(v == null)
			return null;
		Variable result = new Variable(Type.newInstance(v.getVarType()),
				Iden.newInstance(v.getVarName()));
		if(v.isInitialized())
			result.setInitialized();
		return result;
	}
	public Iden getVarName() {
		return varName;
	}
	public void setVarName(Iden var_name) {
		this.varName = var_name;
	}
	public Type getVarType() {
		return varType;
	}
	public String getJavaType() throws CompilerException{
		Type temp = this.varType;
		boolean arrFlag = false;
		if(this.varType.getType().endsWith("[]")){
			arrFlag = true;
			if(this.varType.getType().startsWith(Type.STRING))
				temp = new TypeString();
			if(this.varType.getType().startsWith(Type.INT))
				temp = new TypeInt();
		}
		if(temp instanceof TypeInt){
			if(arrFlag)
				return Type.JAVA_INT+"[]";
			else
				return Type.JAVA_INT;
		}
		if(temp instanceof TypeString){
			if(arrFlag)
				return Type.JAVA_STRING+"[]";
			else
				return Type.JAVA_STRING;
		}
		if(temp instanceof TypeVoid)
			return Type.JAVA_VOID;

		throw new CompilerException("Tweaxml Exception: Type does not
exist");
	}
```

```
	public void setVarType(Type var_type) {
		this.varType = var_type;
	}
	public boolean isInitialized(){
		return valPresent;
	}
	public void setInitialized(){
		valPresent = true;
	}
	public String toString(){
		return varType.getType() + " " + varName.getName() + " Init:" +
		this.isInitialized();
	}
}
```

Program to run .txml files using the lexer, parser, treewalker and codegeneration program:

Main.java

```
package tweaxml;

import java.io.*;
import antlr.CommonAST;

/**
 *
 * @author kk2457
 */
class Main {
	public static void main(String[] args) {
		TweaXmlOut outxx = null;
		try {
			String file = args[0];
			System.out.println(file);
			if(!file.endsWith(".txml")){
				System.err.println("Invalid file type!");
				System.exit(1);
			}
```

```
                        TweaXMLLexer lexer = new TweaXMLLexer(new
DataInputStream(new
                                        FileInputStream(args[0])));
                        System.out.println("Parsing the input file");
                        TweaXMLParser parser = new TweaXMLParser(lexer);
                        parser.setASTNodeClass("tweaxml.TweaXmlAST");
                        parser.program();
                        System.out.println("Generating AST");
                        CommonAST parseTree = (CommonAST)parser.getAST();
                        TweaXmlOut.initFileName(args[0]);
                        System.out.println("Walking over the AST");
                        TweaXmlWalker walker = new TweaXmlWalker();
                        walker.setASTNodeClass("tweaxml.TweaXmlAST");
                        walker.program(parseTree);
                        System.out.println("Generating Code...");
                        TweaXmlCodeGen cg = new TweaXmlCodeGen();
                        cg.program(parseTree);
                        cg.out.writeToJava();
                        System.out.println("Java output file is ready.  ");
                } catch(Exception e) {
                        System.err.println(e);
                        System.exit(0);
                }
        }
}
```