Windshield

Windows Shell Script

Columbia University
COMS W4115 Programming Languages and Translators
Spring 2007

Prof. Stephen A. Edwards

Team members

Wei-Yun Ma	**wm2174**	**wm2174@columbia.edu**
Tony Wang	**tw2174**	**tw2174@columbia.edu**
Tzu-Jung Liu	**tl2263**	**tl2263@columbia.edu**

Chapter 1

Overview

On command line-based operating systems, shell scripts are especially useful. Through the evolution of versions, shell programming has been consistently integrating new functions, such as variables, loops, conditions, strengthening shell scripts to be capable of much more. One of the important features of shell scripts is its ability to “glue” software, which is available under its current operating system (shell scripts are also known as glue languages). Due to the rise of GUI, WWW, and component frameworks, we need a better, more powerful alternative to integrate software.

With the likes of Bourne Shell, Bourne-Again Shell(bash), Korne Shell, UNIX had enjoyed a great variety of powerful shell scripting language. Windows on the other hand, utilizes batch files and software such as 4DOS, 4NT to strengthen the former; however, still can not program as powerfully as UNIX shell scripting language.

We therefore propose to implement Windows Shell Script(Windshield). Using the Bash script as reference and implementing the fundamentals, we try to develop a prototype of a Windows Shell Script and a corresponding compiler to simulate the functions of the interpreter. We also seek to add other useful functionalities to further enhance programming with Windshield. Our compiler will read in the source code, and produce a Perl file, executable under Windows.

Chapter 2

Language Tutorial

A shell program of Windshield, called a windows script, is an easy-to-use tool for building applications by "gluing" together system calls, tools, utilities, and compiled binaries. Virtually the entire repertoire of DOS commands, major UNIX commands, utilities, and tools is available for invocation by a shell script. If that were not enough, internal shell commands, such as testing and loop constructs, functions, give additional power and flexibility to scripts. Windows scripts lend themselves exceptionally well to administrative system tasks and other routine repetitive jobs not requiring the bells and whistles of a full-blown tightly structured programming language.

Writing Windshield is not hard to learn, since the scripts can be built in bite-sized sections and there is only a fairly small set of shell-specific operators and options to learn. The syntax is simple and straightforward, similar to that of invoking and chaining together utilities at the command line, and there are only a few "rules" to learn.

2.1 How to write Windshield and execute them

Following steps are required to write your codes and execute them:

(1) Install Perl interpreter in Windows (ActivePerl)
(2) Use any editor in windows like “notepad” or “UltraEdit” to write shell script.
(3) Use our compiler to compile the Windshield source codes to generate corresponding Perl program.
(4) Give a command to execute the Perl program.

2.2 My first Windshield Program- Variable Assign

You can give a name to a variable and assign its value. The name is a composition of letters and digit numbers. The value is either a number (integer of float) or a string. For example:

```
i=5;
j=”hello world”;
```

Each assign statement should end with a semicolon. You can also assign a variable’s value to another variable.
For example:

```
i=”hello world”;
j=i;
```

2.3 Arithmetic Operation

Arithmetic operation is straightforward. We have four arithmetic operators (+: plus, -: minus, *: multiply, /: divide). Parentheses could be used to represent the precedence.

For example:

```
i=5+7;
j=(2+3)*6;
```

2.4 String Concatenation

Multiple strings can be concatenated by using string concatenation. These string could be "…", a number (regarded as a string too) or a ID which represents a string. A white space is used as the operator. For example,

```
i="hello" "world"; // i="hello world"
j=i "good morning"; // j="hello world good morning"
```

2.5 Conditional Statement

A conditional statement can be either an if statement or an case statement.

<u>If statement</u>

An if statement determines its' control flow based on the logical expression. For example,

```
i=3;
if [ i<5 ]
   then
        k=6;
else
   k=8;
fi
```

The "else" statement is optional.

<u>Case statement</u>

Case statements' control flow is determined by the value of a given variable. For example,

```
i=7;
case i in
    1)
          k=6;
           break;
    2)
          k=7;
          break;
    *)    // when i is not 1 or 2
          k=8;
          break;
esac
```

There may be zero or more (number or string) options, and the * option is the default option, which is optional too. However, a case statement may not contain no options, thus it either contains both or one of the two.

2.6 Loop Statement

A loop statement is either a for-loop or a while-loop. Statement is this category augments the original statement definition by allowing two extra statements: the break (break;) and continue (continue;) statements.

For-loop

It starts at the value of the first assignment statement, incrementing by one each loop, and stops once either: the logical statement is false, or when the second assign statement is reached. For example,

```
for (i=0 ; i<6 ; i=i+1 )
do
      k=”This is number” i;
done
```

<u>While-loop</u>

A while loop iterates until the logical expression it's based on is unsatisfied.

```
i=0
while (i<6)
do
    k=”This is number” i;
    i=i+1;
done
```

2.7 System Call

System allows you to facilitate the built-in functions, it also serves as a mean to call external programs. You can also use pipeline and IO-redirection. For example,

```
myarg=5;
system(echo “hello world”);
system(echo “hello world” | myprogram.exe myarg > output.txt);
```

2.8 Function

2.8.1 Function Definition

Function in Windshield needs to be defined before they are used. Arguments inside a function are referred to using the argument token. It is optional for a function to return a value. For example,

```
function transferT[]
{
   temp = $1 * 9 / 4 + 32; // $1 means the first argument
   return temp;
}   // a function converts from Celsius to Fahrenheit
```

2.8.1 Function Call

Function calls are function names followed by zero or more arguments, the arguments are separated by ‘,’ and enclosed in ‘[‘ and ‘]’.They evaluate to a certain value and type depending on the arguments and function definition. For example,

```
tempC = 32
tempF = transferT[tempC];
```

Chapter 3

Language Reference Manual

3.1 Lexical Conventions

3.1.1 Comments

There are two ways to write comments: Single-line comments start with the special character ‘#’ and ends at the end of the line. Multi-line comments starts with ‘#/’ and ends at the first matching ‘/#’.

3.1.2 Identifiers

An identifier is a sequence of letters, digits and underscores. The first character must be a letter. Identifiers are case-sensitive.

3.1.3 Arguments

To refer to arguments inside a user-defined function, the user must use the following form: a ‘$’ followed by the corresponding argument number, i.e. the first argument is $1, whereas the fifteenth is $15.

3.1.4 Keywords

Below is a table of keywords, they may not be used as variable names when writing a program in Windshield.

break	case	continue	copy (cp)	del (rm)
dir (ls)	do	done	echo	elif
else	find (grep)	function	for	if
in	move	system	then	type (cat)
return	while			

3.1.5 Numbers

A number consists of a sequence of one or more digits, with an optional decimal part. The decimal part contains a ‘.’ followed by another sequence of one of more digits.

3.1.6 String

String literals are characters contained in a pair of double quotes. To include a double quote (“) inside the string itself, the user needs to type two consecutive double quote.

3.1.7 Others

{	}	[	]	(	)
+	-	*	/	\\\\	:
= =	!=	>	>>	<	>=
<=	=	.	,	;	&&
||	|	-d	-e		

3.2 Types

Variables do not need to be declared to a specific type upon creation. Instead its’ type is automatically determined when that variable is assigned a value.

The underlying types in Windshield are: string, number, function.

3.3 Expressions

3.3.1 Primary expressions

Primary expressions includes constants (string and number), identifiers, and function calls. These expressions also serve as base elements to form more complex expressions, including: arithmetic expressions, relational expressions, logical expressions, and string concatenations.

Constants

Constants are either strings or numbers, which are as defined in lexical conventions, constants are evaluated to themselves.

Identifiers

An identifier in an expression is bounded to a value and will be evaluated to a specific type.

Function calls

Function calls are function names followed by zero or more arguments, the arguments are separated by ‘,’ and enclosed in ‘[‘ and ‘]’.They evaluate to a certain value and type depending on the arguments and function definition.

3.3.2 Arithmetic expressions

Arithmetic expressions is recursively defined as: Either a number or a function call that returns a number, or two sequential arithmetic expressions combined with a middle arithmetic operator (+: plus, -: minus, *: multiply, /: divide).

3.3.3 Relational expressions

Relational expressions can be either an arithmetic expression or two relational expressions combined with a middle relational operator (< , > , <= , >= , = = , !=).

3.3.4 Logical expressions

A logical expression can be either an relational expression, or two logical expressions combined with a logical operator in the middle. (Logical operators: &&, ||).

3.3.5 String concatenation

String concatenation contains a single string literal, or a sequence of constants, IDs, separated by white spaces.

3.4 Statements

3.4.1 Conditional statements

A conditional statement can be either an if statement or an case statement.

<u>If statement</u>

An if statement determines its' control flow based on the logical expression. More specifically, it takes the following form:

```
if [ logical expression ]
   then (statement)*
elif [ logical expression ]
   then (statement)*
else
   (statement)*
fi
```

In which there can be zero or more "elif" statements, and the "else" statement is optional.

<u>Case statement</u>

Case statements' control flow is determined by the value of a given variable:

```
case identifier in
number|string)
   (statement)*
*)
   (statement)*
esac
```

There may be zero or more (number or string) options, and the * option is the default option is optional too. However, a case statement may not contain no options, thus it either contains both or one of the two.

3.4.2 Loop statements

A loop statements is either a for-loop or a while-loop. Statement is this category augments the original statement definition by allowing two extra statements: the break (break;) and continue (continue;) statements.

<u>For-loop</u>

The for loop takes the following form, it starts at the value of the first assignment statement, incrementing by one each loop, and stops once either: the logical statement is false, or when the second assign statement is reached.

```
for (assign statement ; logical statement ; assign statement )
   do
         (statement | break | continue)*
   done
```

<u>While-loop</u>

A while loop iterates until the logical expression it’s based on is unsatisfied.

```
while [logical expression]
   do
         (statement | break | continue)*
   done
```

3.4.3 Assign statement

An assign statement assigns an a value to an identifier. The right-value to be assigned is either: a constant, a string literal, an arithmetic expression, or a function call:

```
ID = right-value;
```

3.4.4 Function call

A function call can be statement alone. It needs to be defined before it can be used, in addition, the definition has to appear earlier in the program before it can be called.

3.4.5 Return statement

A return statement has two functionalities, inside the main scope, a return statement exits the program. Inside a certain function, a return statement terminates that function, and returns the followed primary expression. It takes the following form:

return primary_expression;

3.4.6 System call

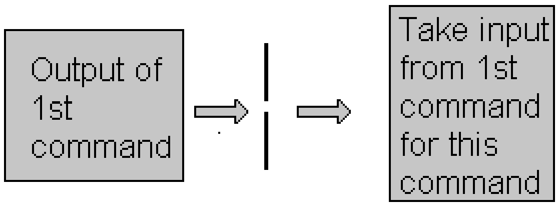
System allows the user to facilitate the built-in functions, it also serves as a mean to call external functions:

system (instruction_unit);

The instruction unit above could be either a built-in function or a external function. Note that these instruction units can be used in a more flexible way, in which they can be connected with '>' , '<' and '|':

system(instruction _unit (connector instruction _unit)*);

In which the connector is either IO-redirection operators '>' , '<' or the instruction operator '|'. Below is a brief flow graph of how the pipeline operator works.



3.5 User defined functions

Function in Windshield needs to be defined before they are used. Arguments inside a function are referred to using the argument token (in the lexical rules). It is optional for a function to return a value:

```
function ID[ ]{

    (statement | return statement)*

}
```

3.6 Built-in functions

Built-in functions and their corresponding unix command are listed below:

Function name	UNIX command	Description
“echo” string	echo	Prints out a string to screen.
dir	ls	Lists the files in the current directory
“type” file	cat	Prints out the contents of a file
“copy” source destination	cp	Copies a file from source to destination
move source destination	mv	Moves a file
del file	rm	Deletes a file
find	grep	Tries to match user specified words.

Chapter 4

Project Plan

Project Windshield didn't have a smooth start. During the first two weeks, we had problems forming a team, we thought we'd have 5 members, it turned out we had to go with three. Though we had to re-consider our project proposal, as a result meetings were easily arranged.

Communication wise, we used e-mail, online messengers, and phone. On the code aspect, CVS was used to maintain keep all members up to date. The development speed was started off slowly and accelerated towards the end.

4.1 Project Log

Date	Progress
1 / 29 / 2007	Brainstorming
2 / 05 / 2007	Project proposal completed
3 / 01 / 2007	LRM completed
3 / 20 / 2007	Lexer completed, Parser started
4 / 03 / 2007	Parser integration , non-determinism fixing
4 / 10 / 2007	Walker structure discussion
4 / 20 / 2007	Code generating initial step
4 / 25 / 2007	More code generating
4/ / 29 / 2007	Static semantic analysis complete
5 / 01 / 2007	Final walker integration
5 / 02 / 2007	Wss Translator completed
5 / 05 / 2007	Testing
5 / 06 / 2007	Report completed
5 / 07 / 2007	Presentation, Report final amendment

4.2 Programming style

The way we programmed Windshield is unique. We did not divide the work into front-end, back-end, testing and documentation. Instead, we all participated in the two core part of the translator - parser and code generating, gaining understanding of what each field is like.

We divided work through “segments” of the parser and walker, while trying to maintain independency, we also helped each other out with problems such as non-determinisms and code clashes.

This way of dividing the work forms an interesting characteristic of our final product – most of the code lies in the “grammar.g” file.

4.3 Software environment

4.3.1 Java

The java versions were kept up to date, by the time of finishing the project, the version installed was: “1.6.0_01”. With the aid of ANTLR, Java is the main programming language we programmed in.

4.3.2 ANTLR

We used ANLR version 2.7.6 instead of the newest version 2.7.7, which is the newest version we could find an Eclipse plug-in for.

4.3.3 Eclipse

Eclipse is an IDE which makes java programming efficient, it also has many external plug-ins so that coding various programming languages under Eclipse is possible. The plug-in we used is named “ANTLREclipse” (http://antlreclipse.sourceforge.net/). Warnings such as non-determinisms in ANTLR code were notified instantly in Eclipse.

4.3.4 CVS

CVS was setup under the account “tw2174” at the clic machines, and all other members were given the privilege to access and write to file, as well as updating. CVS is a very efficient tool, that merge changes when it sees fit, some troubles were saved towards the end

4.3.5 Operating system

All members wrote their codes under the operating system Windows XP.

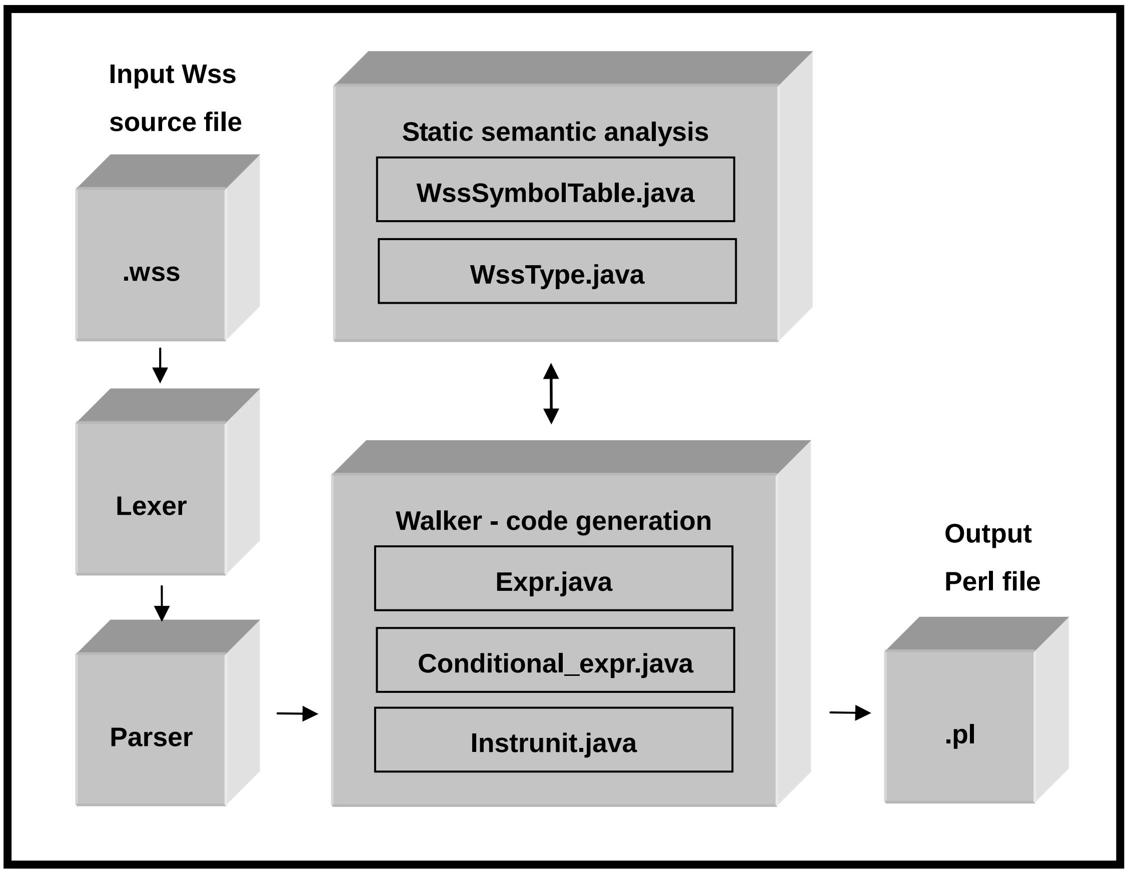
4.4 Team responsibilities

Lexer		Tony
Parser	Loops and Iterations, Expression integration	Tzu-Jung
	Assignment, Function, Arithmetic expression	Tony
	Loops and Iterations, System call, Built-in Functions	Wei-Yun
Code Generation	Loops and Iterations	Tzu-Jung
	Expressions, System call, Built-in functions	Wei-Yun
	Assignment, Functions	Tony
Static Semantic Analysis		Tony
Testing		Tzu-Jung
Documentation	Chapter 1, 2 , 6	Wei-Yun
	Chapter 3, 4, 5	Tony
	Chapter 7 , 6	Tzu-Jung

Chapter 5

Architectural Design

5.1 Block Diagram



The Windshield(WSS) translator will translate source ".wss" files into a Perl program, the architecture is as shown in the above graph. The lexer produces tokens, and the parser produces an AST tree, should there be any syntactic error, the parser will display an error.

5.2 Scoping and symbol table

The scoping rule in Windshield is static, there may be two scopes inside a program, one is the main scope, the other is the scope in the function. Scoping is maintained by a symbol table (WssSymbolTable.java) to do type checking, each table has a parent table (topmost parent = null). When a variable is used as a right-value, it will first try to retrieve it from the current scope, before it looks at it's parent scope, the searching stops as soon as variable is found or when it reaches the topmost table.

5.3 Static semantic analysis

Neither variables nor functions were specified a type or return type, therefore these will have to be determined implicitly. A type class (WssType.java) is used to help run type checking. There are three types – number, string, function. The function type also has a "return type", which can be either number, string, void or unable to determine.

The symbol table serves as the golden reference for type check, there are two ways a name will be pushed onto the hash table, type checking is also done at this time:

I . When a variable is assigned:

Ex. i = 1 + 2 * j / function_one[2,4,6] ;

An assign statement contains an ID, an assign symbol, and a right expression. The ID is the name we will use as key to push onto the table, the content of that key – the type, needs to be determined by the right expression – either a string concatenation or an arithmetic expression. Here's where type checking comes in:

(1) String concatenation : in this case we need only assign the type of the ID to string, no type checking is needed, since string can concatenate with numbers also.

(2) Arithmetic expression: here we are expecting the type to be number, but we need to make sure all operands are consistent (numeric). An operand can be a literal, where inconsistency is quickly spotted once a string is found, but it can also be an ID or a function call. When it is an ID, we need to check the table for it's type. For a function call however, it's a bit more complicated, explained in the next section (II.).

In the big picture, each expression is assigned to either a number or not a number. Once it is not a number, an error occurs.

II. When a function is defined.

```
Ex. function temp[ ] {
            i = 1+$2;
            return i;
      }
```

Similar to variable assign, where the user need not specify the variable type, function definitions does not specify the return type. This is solved using the magic of the return statement.

When we first see a function definition, we push the function name, with the type function and return type void onto the symbol table. If a return statement is not seen, the return value stays void, if a return statement is seen, we then determine the return value of the function through that statement, return of the literals are straightforward, return of ID requires looking up of the current function symbol table to determine the type, we then know what's the return type of a function call in (I.).

Other than the above, additional type checking includes usage of undeclared variables and functions, or misusage of a function as a variable and the other way round. These checks are done using the "get" method of the symbol table.

One thing to note is when type checking involves arguments (ex. $1, $2). A conservative approach is taken: In arithmetic expressions, arguments are allowed since the values are determined by the program and run time. Due to the same reason, function calls that returns arguments have their return type set to unable to determine, and are not caught if used in an arithmetic expression.

5.4 Error messages

Given the implementation in the last section, when a type error occurs, an error message is displayed on the screen, with its' error type number, explanation, and which variable caused the problem. The table on the next page is a summarize of these errors:

Error Type Number	**Description**
Error type #1	Use of undeclared variable
Error type #2	Call of undefined function
Error type #3	Variable used as a function
Error type #4	Invalid operand in arithmetic expression (a string or an ID that contains string)
Error type #5	Invalid operand in arithmetic expression (a function that either returns void or returns a string)

Chapter 6

Some Implementation Tricks

In this chapter, we will write some parts of the implementation we find worthy of mentioning.

6.1 Expression and Control Flow

In the implementation process, the most frequent problems we faced are nondeterminism. The occurrence of nondeterminism would affect the accuracy of the parser, which further affects the accuracy of the walker and generating codes. Most of the nondeterminism problems can be fixed by adjusting our ANLTR codes. But some are hard to completely solve. Take a representative example, Our Arithmetic and Logical expressions are recursively defined. In our original design, we use parentheses to represent the precedence of operation either in Arithmetic or Logical expressions, such as

(5>2 && ((6+2)*5>3 || 2<5))

But we immediately faced the problem of nondeterminism. The parser can not make sure the following part of "&&" is an Arithmetic expression or Logical expression. For solving this problem, we finally decided to change our language syntax about this part- using parentheses in Arithmetic expressions, and using brackets in Logical expressions. So the above example become

[5>2 && [(6+2)*5>3 || 2<5]]

We found this syntax is just the same as the syntax in Bash shell of UNIX. Probably the designer of Bash shell design this writing syntax based on the same reason.

Besides nondeterminism problem, another interesting case is how to implement "case" (switch) in Windshield. Because our generated code- perl does not have"case" or "switch" in its syntax. SO we used the skills we learned in class- using multiple "if" statements to implement one "case". Following are two examples:

```
WSS:
i=2;
case i in
1)
      k=3;
      break;
2)
      e=0;
      break;
*)
      w=0;
esac
```

$\longrightarrow$

```
Perl:
$i=2;
if ($i==1){
    $k=3;
    }
elsif ($i==2){
          $e=0;
        }
    else{
              $w=0;
          }
```

```
WSS:
i=0;
case i in
*)
      w=0;
esac
```

$\longrightarrow$

```
Perl:
$i=0;
$w=0;
```

6.2 System Call

The most challenging implementation issue lies on checking of grammar of build-in function, pipeline and IO-redirection from syntax perspective, and checking the declaration of variables in system call from semantic perspective. For future expansion, we designed an organized structure in ALTLR in parser and walker.

In the following table, you can find we divided the body of system call into multiple instruction units based on PIPE sign. Each instruction unit is either a build-in function or a outside program. We also check the accuracy of Build-in function usages but allow outside programs to have more flexibility of usages. The design structure is clear and easy to expend more build-in functions in the future.

```
system_stmt
        : "system"^ LPAR! system_body RPAR!;
system_body
        : instr_unit (PIPE instr_unit)* ;
instr_unit
        : buildinfunc_stmt (io_redirection)?
        | outsideprogram_stmt (io_redirection)?
        ;
buildinfunc_stmt
        : "echo" (STRING|NUMBER|ID|ID_IN_SYSTEM)
        | "copy" (program_name|ID_IN_SYSTEM) (program_name|ID_IN_SYSTEM)
        …
outsideprogram_stmt
        : program_name (STRING|NUMBER|program_name|ID_IN_SYSTEM)* ;
io_redirection
        : GT program_name (LT program_name)?
        | GGT program_name (LT program_name)?
        | LT program_name ((GT|GGT) program_name)?
        ;
program_name
        : ID^ (DOT ID)?
```

ANLTR code for system call in parser

Chapter 7

A small program of Windshield – Temperature Transfer

```
function transferT[]
{
        temp = $1 * 9 / 4 + 32;
        return temp;
} #a function converts from Celsius to Fahrenheit
i=0;
for(i=0;i<3;i=i+1)
do
        case i in
        0)
                country = "Taiwan";
                tempC = 32; #Celsius
                break;
        1)
                country = "Japan";
                tempC = 20; #Celsius
                break;
        2)
                country = "England";
                tempC = 12; #Celsius
                break;
        esac

        separate = "----------";
        tempF = transferT[tempC]; #convert from Celsius to Fahrenheit
        system(echo $country); #print country name
        system(echo $tempF); #print Fahrenheit degree
        system(cat $country); #print some information
        system(grep $country diary.txt); #search for country name
        system(echo $separate);
done
```

It’s generated Perl code

```
sub transferT{
$temp=((($_[0]*9)/4)+32);
return $temp;
}
$i=0;
for ($i=0 ; ($i<3) ; $i=($i+1))
{if ($i==0)
{
$country="Taiwan";
$tempC=32;
}
elsif ($i==1)
{
$country="Japan";
$tempC=20;
}
elsif ($i==2)
{
$country="England";
$tempC=12;
}
$separate="----------";
$tempF=transferT($tempC,);
system "echo $country";
system "echo $tempF";
system "type    $country";
system "find \"$country\" diary.txt";
system "echo $separate";
}
```

Chapter 8

Testing

In this chapter, we will introduce some correct and incorrect testing cases of different part such as arithmetic, control flow and conditional expression, variables and functions and system call. We do not list all cases because a large part is done by parser. We just list some cases that bother us when we construct the grammer.g.

8.1 Arithmetic

The most challenging implementation issue lies on checking of grammar of build-in function, p

Correct case:

Precedence, association and left-right parenthesis are ok.

```
i = 4+6*3+(3*(4+5)+3);
 (   ( = i ( + ( + 4 ( * 6 3 ) ) ( + ( * 3 ( + 4 5 ) ) 3 ) ) ) ) )
$i=((4+(6*3))+((3*(4+5))+3));
```

Concatenating strings.

```
foo = "abc" "def";
$foo="abc"."def";
```

Incorrect cases:

```
i = 4+6*5++5;
line 1:9: unexpected token: +
i = 4+6/*5+5;
line 1:7: unexpected token: *
foo = "abc"+"def";
unexpected token: +
```

8.2 Control Flow and Conditional Expression

Because control flow (if, while, for and case) and conditional expression are bound together and very like each other, we show some cases included both here.

◆ Correct cases:

if:

```
i=0;
if [i>9 && [ i<3 || i>5 && i<7 ] ]
then
        i=0;
elif [i==4]
then
        i=1;
else
        i=50;
fi
(    ( = i 0 ) ( if ( && ( > i 9 ) ( || ( < i 3 ) ( && ( > i 5 ) ( < i 7 ) ) ) ) ( then ( = i 0 ) )
( elif ( == i 4 ) ( then ( = i 1 ) ) ) ( else ( = i 50 ) ) ) )
$i=0;
if (($i>9) && (($i<3) || (($i>5) && ($i<7))))
{
$i=0;
}
elsif ($i==4)
{
$i=1;
}
else
{
$i=50;
}
```

case:

We confert from case statements into if statement in pler. Because perl does not have switch or case statements. This concept was taught in the control flow class. By the way, we remove all “break” in the case statement when we convert it into if statements.

```
i=2;
case i in
1)
        k=3;
        break;
```

```
2)
        e=0;
        break;
*)
        w=0;
esac
  (    ( = i 2 ) ( case i ( 1 ( = k 3 ) break ) ( 2 ( = e 0 ) break ) ( * ( = w 0 ) ) ) )
$i=2;
if ($i==1)
{
$k=3;
break;
}
elsif ($i==2)
{
$e=0;
break;
}
else
{
$w=0;
}
```

When there is only default selection in case statement, we won’t use if statement but directly shows statements in this default selection. Because you have no choice but implement these statements.

```
i=2;
case i in
*)
        w=0;
esac
  (    ( = i 2 ) ( case i ( * ( = w 0 ) ) ) )
$i=2;
$w=0;
```

for:

```
i=0;
j=0;
for (i=0;i<10;i=i+1)
```

```
do
        j = j+1;
done
 (    ( = i 0 ) ( = j 0 ) ( for ( = i 0 ) ( < i 10 ) ( = i ( + i 1 ) ) ( do ( = j ( + j 1 ) ) ) ) ) )
$i=0;
$j=0;
for ($i=0 ; ($i<10) ; $i=($i+1))
{
$j=($j+1);
}
```

◆ Incorrect case:

It is illegal to change from assign to conditional expressions or from conditional to assign expressions

```
i=0;
if [i>9 && [ i=3 || i>5 && i<7 ] ]
then
        i=0;
fi
line 2:14: unexpected token: i
line 2:16: unexpected token: 3
for (i<0;i<10;i=i+1)
line 3:7: expecting ASGN, found '<'
for (i=0;i=10;i=i+1)
line 3:10: unexpected token: i
line 3:12: unexpected token: 10
```

The parser can not allow any selections after the default selection.

```
i=2;
case i in
*)
        w=0;
1)
        e=0;
        break;
esac
line 5:1: expecting "esac", found '1'
```

Here is a nested and complicated case:

```
i=0;
j=0;
if [i>9 && [ i<3 || i>5 && i<7 ] ]
then
    if [j==0]
    then
        for (i=0;i<10;i=i+1)
        do
            while [i<6]
            do
                j = i+j;
                case j in
                3)
                    i = i+1;
                    break;
                5)
                    i = i+2;
                    break;
                esac
            done
        done
    fi
elif [i==4]
then
    i=1;
else
    i=50;
fi
```

```
(  ( = i 0 ) ( = j 0 ) ( if ( && ( > i 9 ) ( || ( < i 3 ) ( && ( > i 5 )
( < i 7 ) ) ) ) ( then ( if ( == j 0 ) ( then ( for ( = i 0 ) ( < i 10 )
( = i ( + i 1 ) ) ( do ( while ( < i 6 ) ( do ( = j ( + i j ) ) ( case
j ( 3 ( = i ( + i 1 ) ) break ) ( 5 ( = i ( + i 2 ) ) break ) ) ) ) ) ) ) )
( elif ( == i 4 ) ( then ( = i 1 ) ) ) ( else ( = i 50 ) ) ) )
```

```
$i=0;
$j=0;
if (($i>9) && (($i<3) || (($i>5) && ($i<7))))
{
```

```
if ($j==0)
{
for ($i=0 ; ($i<10) ; $i=($i+1))
{
while ($i<6)
{
$j=($i+$j);
if ($j==3)
{
$i=($i+1);
break;
}
elsif ($j==5)
{
$i=($i+2);
break;
}
}
}
}
}
elsif ($i==4)
{
$i=1;
}
else
{
$i=50;
}
```

8.3 Variable and Function

There are five kinds of error messages about variables and functions:

```
!! WSS Error (Type #1): Use of undeclared variable
!! WSS Error (Type #2): Call of undefined function
!! WSS Error (Type #3): Variable used as a function name
!! WSS Error (Type #4): Invalid operand in arithmetic operation
   (a non-numeric operand)
```

```
!! WSS Error (Type #5): Invalid operand in arithmetic operation
  (The function - returns a non-numeric operand <string or void?> )
```

◆ Correct cases:

Declare y first then you can do anything on y.

```
y = 1;
x = y;
```

y is checked type before any operation. Because it is illegal that “number + string”.

```
x = 1 + y;
```

A function has to be declared before user use it. Also, the return type must be checked before any operation.

```
function funky[]
{
    b = "123";
    return b;
}
z = "abc" funky[];
sub funky{
$a="123";
return $a;
}
$z="abc".funky(1,2,);
```

the result is: z = “abc123”

```
function funky[]
{
    a = $1 + $2;
    return a;
}
z = 1 + funky[1,2];
sub funky{
$a=($_[0]+$_[1]);
return $a;
}
$z=(1+funky(1,2,));
```

the result is: z = 4

◆ Incorrect case:

```
x = y; //error
!! WSS Error (Type #1): Use of undeclared variable - y
funky[1,2]; //the undeclared function
!! WSS Error (Type #2): Call of undefined function - funky
x = 1;
z = 1+x; //ok, both are numbers
z = 1+ x[111]; //error
!! WSS Error (Type #3): Variable used as a function name - x
!! (x is a number not a function! )
i = "hahaha";
j = i;
k = 1+j; //error
!! WSS Error (Type #4): Invalid operand in arithmetic operation -  j
!! ( j is a non-numeric operand )
function temp[]{
    what = "Ye";
    return what;
}
y = 1+temp[]; //error
!! WSS Error (Type #5): Invalid operand in arithmetic operation - temp
!! (The function - temp returns a non-numeric operand <string or void?> )
```

8.4 System Call

◆ Correct cases:

```
system(aaa.exe bbb.txt ccc < ddd > kkk);
system "aaa.exe bbb.txt ccc < ddd > kkk";
system(grep abc kkk.txt > bbb.txt);
system "grep abc kkk.txt > bbb.txt";
system(aaa.exe ccc | bbb.exe ddd eee | hhh.exe iii jjj kkk);
system "aaa.exe ccc | bbb.exe ddd eee | hhh.exe iii jjj kkk";
p="bbb.txt";
system(aaa.exe $p ccc < ddd > kkk); // $p means this is a veriable
system(aaa.exe p ccc < ddd > kkk); // p means this is only “p”
$p="bbb.txt";
system "aaa.exe $p ccc < ddd > kkk"; // perl see $p as “bbb.txt”
system "aaa.exe p ccc < ddd > kkk"; // perl see p as “p”
```

Here are two real cases where append_aaa.exe, append_bbb.exe and append_ccc.exe are .exe files which read words from the screen and append “aaa”, “bbb” and “ccc” then print on the screen. The “good” file contains “good”.

```
system(append_aaa < good | append_bbb | append_ccc);
system "append_aaa < good | append_bbb | append_ccc";
```

result: “goodaaabbbccc”.

◆ Incorrect cases:

Only four kinds of operands are correct, “ > < ” “ >> < ” “ < > ” and “ < >> ”. All other combinations are illegal.

```
system(aaa.exe bbb.txt ccc < ddd < yyy > kkk);
unexpected token: <
system(aaa.exe bbb.txt ccc << ddd > kkk);
line 1:29: expecting ID, found '<'
```

Chapter 9

Lessons Learned

9.1 Wei-Yun Ma

I am glad I took this class as my one of core classes. I have done research on NLP for several years before I went to Columbia. I have been hoping to enhance my knowledge about parser skills. In this course, I learned how to implement a parser, a walker and gained valuable experience in ALTLR. I also learned how to carefully design a language. This process of implementation is challenging but interesting!

9.2 Tzu-Jung Liu

There are only three people in our team, so every one has to do lots of work. It is very good to me because the more you do the more you can learn. We discuss what we done before the meeting and teach to each other. I have learned the whole part of lexir, parser and tree walker by the part I done and discussions, especially the part of type and semantic checking. I start understand the reason why we should declare a variables or function type before we use it. And how to implement the concept of scop.

Finally I understand how to construct a programming language out of nothing and some concepts taught in classes. Also, I have learned how to do this through ANTLR and JAVA code.

I also learned a very important thing, team work. The best team work is to finish the part you are assigned and consider how to teach your teammates what you done and do not change the meeting time if you do not really have other important thing to do. And the most important thing is be patient and discuss to each other when some different opinions occured.

9.3 Tony Wang

PLT is composed of theory and practice, without doing the implementation, one could barely say he know PLT. I started off from knowing the theory, not the practice, to gradually gaining grasp of the big idea. Previous projects are resourceful references, I’d especially recommend the MX and the dragon book language.

Working with a team is definitely great experience gained, if you read through our labor division, you’d find our division quite unique. We tried to seek independence among dependence, that given, frequent communication is a requisite.

One last word: it’s human nature to start late, but be prepared, it’s a good load of work!

Appendix

Code of grammar.g

```
/*
 * File Name: grammar.g
 * Description: 1. WssLexer - ANTLR code for lexer
 *                              2. WssParser - ANTLR code for parser
 *                              3. WssWalker - ANTLR code for tree walker
 */

//    -------------------- 1.WssLexer --------------------
class WssLexer extends Lexer;

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

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

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

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

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

ID_IN_SYSTEM :
'$' LETTER (LETTER | DIGIT | '_')* ;

ARG :
'$' '1'..'9' (DIGIT)*;

NUMBER : (DIGIT)+ (DOT (DIGIT)+)?;

/*
 * To include a '"' inside a string, type two '"' when assigning
 */
STRING :    '"'! ( '"' '"'! | ~('"'))*    '"'!;

LBR    :          '{';
RBR    :          '}';
LSQ :  '[';
RSQ :  ']';
LPAR: '(';
RPAR: ')';
PLUS :'+';
MINUS :         '-';
MULT :          '*';
DIV :  '/';
IDIV :  '\\';
COLON : ':';
EQ :    "==";
```

```
NE :    "!=";
GT :    '>';
GGT :      ">>";
LT :     '<';
GE :    ">=";
LE :    "<=";
ASGN:           '=';
DOT :  '.';
COMMA :      ',';
SEMI: ';';
AND : "&&";
OR :    "||";
PIPE :  '|';
DFLAG : "-d";
EFLAG : "-e";




//      --------------------- 2.WssParser ---------------------
class WssParser extends Parser;
options {
        buildAST = true;
        k = 2;
}

tokens {
    STATEMENT;
}

program
        : (statement | function_decl)* EOF!
                { #program = #([STATEMENT],program); }
        ;

statement
        : if_stmt
        | case_stmt
        | for_stmt
        | while_stmt

        | return_stmt

        | assign_stmt SEMI!

        | function_call SEMI!

        | system_stmt SEMI!

        | SEMI! //empty statement is possible

        ;


// ----------if statement ----------

if_stmt

        : "if"^ LSQ! condition_OR_expr RSQ! then_stmt (elseif_stmt)* (else_stmt)? "fi"!

        ;


elseif_stmt

        : "elif"^ LSQ! condition_OR_expr RSQ! then_stmt

        ;


then_stmt

        : "then"^ (statement)+;


else_stmt

        : "else"^ (statement)+;



// ---------- case statement ----------

case_stmt

        : "case"^ ID (case_expr)+ (case_default)? "esac"!

        ;


case_expr

        : "in"! (case_choice)*;


case_choice

        : (STRING^|NUMBER^) RPAR! (statement)* break_stmt;


case_default

        : MULT^ RPAR! (statement)*;
```

```
// ---------- for statement ----------
for_stmt
	: "for"^ LPAR! assign_stmt SEMI! condition_OR_expr SEMI! assign_stmt RPAR! do_stmt "done"!
	;

do_stmt
	: "do"^ (loop_stmt)+;

loop_stmt
	   : statement
	| break_stmt
	| continue_stmt
	   ;

// ---------- while statement ----------
while_stmt
	: "while"^ LSQ! condition_OR_expr RSQ! do_stmt "done"!;

// ---------- break statement ----------
break_stmt
	: "break"^ SEMI!;

// ---------- continue statement ----------
continue_stmt
	: "continue"^ SEMI!;

// ---------- assignment statements ----------
assign_stmt
	: ID ASGN^ (expr_plus_minus |str_concat)
	;

// string (w/ optional concatenation)
str_concat
	   : STRING ((STRING|atom)+)?
	   ;



// ---------- function call ----------
function_call
```

```
        : ID LSQ^ arg_list RSQ!
        ;

arg_list
        : (atom (COMMA! atom)*)?
        ;

// ---------- system statement ----------

system_stmt
      : "system"^ LPAR! system_body RPAR!;

system_body
      : instr_unit (PIPE instr_unit)* ;

// ---------- instruction unit ----------

instr_unit
      : buildinfunc_stmt (io_redirection)?
      | outsideprogram_stmt (io_redirection)?
      ;

buildinfunc_stmt
      : "echo" (STRING|NUMBER|ID|ID_IN_SYSTEM)
      | "dir" (MULT|DOT|program_name|ID_IN_SYSTEM)*
      | "ls" (MULT|DOT|program_name|ID_IN_SYSTEM)*
      | "type" (program_name|ID_IN_SYSTEM)*
      | "cat" (program_name|ID_IN_SYSTEM)*
      | "copy" (program_name|ID_IN_SYSTEM) (program_name|ID_IN_SYSTEM)
      | "cp" (program_name|ID_IN_SYSTEM) (program_name|ID_IN_SYSTEM)
      | "move" (program_name|ID_IN_SYSTEM) (program_name|ID_IN_SYSTEM)
      | "mv" (program_name|ID_IN_SYSTEM) (program_name|ID_IN_SYSTEM)
      | "del" (MULT|DOT|program_name|ID_IN_SYSTEM)*
      | "rm" (MULT|DOT|program_name|ID_IN_SYSTEM)*
      | "grep" (ID|ID_IN_SYSTEM) (MULT|DOT|program_name|ID_IN_SYSTEM)*
      ;

outsideprogram_stmt
      : program_name (STRING|NUMBER|program_name|ID_IN_SYSTEM)* ;
```

```
io_redirection
    : GT program_name (LT program_name)?
    | GGT program_name (LT program_name)?
    | LT program_name ((GT|GGT) program_name)?
    ;

program_name
    : ID^ (DOT ID)?
    ;

// Function declaration
function_decl
        : "function"^ ID LSQ! RSQ! function_body
        ;

function_body
        :       LBR! (statement)* RBR!
        ;

return_stmt
        :       "return"^ (ID|ARG) SEMI!
        ;

// The expressions : (logic & arithmetic)

condition_OR_expr
    : condition_AND_expr (OR^ condition_AND_expr)*
    ;

condition_AND_expr
        : conditional_atom (AND^ conditional_atom)*
        ;

conditional_atom
    : expr_plus_minus (EQ^|NE^|GT^|LT^|GE^|LE^) expr_plus_minus
    | LSQ! condition_OR_expr RSQ!
    ;
```

```
expr_plus_minus
        : expr_mult_div ( (PLUS^ | MINUS^) expr_mult_div )*
        ;

expr_mult_div
        : atom ( (MULT^ | DIV^) atom )*
        ;

atom
        : ID
        | ARG
        | NUMBER
        | LPAR! expr_plus_minus RPAR!
        | MINUS^ atom
        | function_call
        ;

//      --------------------- 3.WssWalker ---------------------
class WssWalker extends TreeParser;
{
            WssSymbolTable current = new WssSymbolTable(null,true);
            String current_func = "";
            String current_ID = "";
            String wssProgram = "";
            boolean errorFound = false;
}

program returns [String x]
{
            x = "";
}
            : #(STATEMENT (func_decl|statement)*) { x = wssProgram;}
            ;

func_decl
{
            current = new WssSymbolTable(current,false);
}
            : #("function" ID
                    {
                            current.parent.put(#ID.getText(),WssType.Function);
                            current_func = #ID.getText();
                            wssProgram = wssProgram+"sub " + #ID.getText() + "{\n";
                            //System.err.println("sub " + #ID.getText() + "{");
                    }
                (statement)*
                    {
                            wssProgram = wssProgram + "}\n";
                    }
            )
{ current = current.parent; }
        ;

statement
        : assign_stmt {wssProgram = wssProgram + ";\n";}
        | if_stmt
        | for_stmt
        | while_stmt
        | case_stmt
        | function_call
        | system_stmt
        | return_stmt
        | break_stmt
        | continue_stmt
        ;

assign_stmt
{ Expr a,b; }
        : #(ASGN ID {current_ID = #ID.getText();}    a=expr
                    {
                        String s = a.gen();
                        if (s.charAt(0) == '"'){
                            current.put(#ID.getText(),WssType.String);
                        }
                                wssProgram = wssProgram + "$"+#ID.getText()+"="+a.gen();
                          }
                            (b=expr
                        {
```

```
                                    wssProgram = wssProgram + "."+b.gen();
                            }
                        )*
            )
          ;



condition_expr returns [ Condition_expr ce ]
{
    ce=null;
    Condition_expr a,b;
    Expr c,d;
}
        : #(AND a=condition_expr b=condition_expr
        { String s="("+a.gen()+" && "+b.gen()+")";
                ce=new Condition_expr(s); }     )
        | #(OR a=condition_expr b=condition_expr
          { String s="("+a.gen()+" || "+b.gen()+")";
                 ce=new Condition_expr(s); }     )
        | #(EQ c=expr d=expr
          { String s="("+c.gen()+"=="+d.gen()+")";
                 ce=new Condition_expr(s); }     )
        | #(NE c=expr d=expr
          { String s="("+c.gen()+"!="+d.gen()+")";
                 ce=new Condition_expr(s); }     )
        | #(GT c=expr d=expr
          { String s="("+c.gen()+">"+d.gen()+")";
                 ce=new Condition_expr(s); }     )
        | #(LT c=expr d=expr
          { String s="("+c.gen()+"<"+d.gen()+")";
                 ce=new Condition_expr(s); }     )
        | #(GE c=expr d=expr
          { String s="("+c.gen()+">="+d.gen()+")";
                 ce=new Condition_expr(s); }     )
        | #(LE c=expr d=expr
          { String s="("+c.gen()+"<="+d.gen()+")";
                 ce=new Condition_expr(s); }     )
        ;
```

```
// If statement
if_stmt
{ Condition_expr a; }
        : #("if"
                {
                        wssProgram = wssProgram + "if";
                        //System.err.print("if ");
                }
                   a=condition_expr { wssProgram = wssProgram + a.gen()+"\n{\n"; }
                   then_stmt (elsif_stmt)* (else_stmt)?
           )
        ;

then_stmt
        : #("then" (statement)+) { wssProgram = wssProgram + "}\n"; }
        ;

elsif_stmt
{ Condition_expr a; }
        : #("elif" { wssProgram = wssProgram + "elsif "; }
          a=condition_expr { wssProgram = wssProgram + a.gen()+"\n{\n"; }
          #("then" (statement)+) { wssProgram = wssProgram + "}\n"; } )
        ;

else_stmt
        : #("else" { wssProgram = wssProgram + "else\n{\n"; }
          (statement)+ { wssProgram = wssProgram +"}\n"; } )
        ;

// For statement
for_stmt
{ Condition_expr a; }
        : #("for" { wssProgram = wssProgram + "for ("; }
          assign_stmt a=condition_expr { wssProgram = wssProgram + " ; "+a.gen()+" ; "; }
          assign_stmt { wssProgram = wssProgram + ")\n{"; }
          #("do" (statement)+ { wssProgram = wssProgram +"}"; }) )
        ;

// While_stmt
```

```
while_stmt
{ Condition_expr a; }
        : #("while" { wssProgram = wssProgram +"while "; }
         a=condition_expr
         { wssProgram = wssProgram + a.gen() + "\n"; wssProgram = wssProgram + "{\n"; }
         #("do" (statement)+ { wssProgram = wssProgram + "}\n"; }) )
        ;


// Case_stmt
case_stmt
   { Expr a;
      String b; }
   : #("case"
         ID { if(current.get(#ID.getText()) == null){
                              errorFound = true;
                         System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+#ID.getText()+"\n");


         }
                b="$"+#ID.getText(); }
         ((
         (#(NUMBER { wssProgram = wssProgram +"if ("+b+"=="+#NUMBER.getText()+")\n{\n"; } (statement)+
{ wssProgram = wssProgram + "}\n"; } )
         |#(STRING { wssProgram = wssProgram + "if ("+b+" eq \""+#STRING.getText()+"\")\n{\n"; } (statement)+
{ wssProgram = wssProgram +"}\n"; } )
         )


         (#(NUMBER { wssProgram = wssProgram + "elsif ("+b+"=="+#NUMBER.getText()+")\n{\n"; } (statement)+
{ wssProgram = wssProgram +"}\n"; } )
         |#(STRING { wssProgram = wssProgram +"elsif ("+b+" eq \""+#STRING.getText()+"\")\n{\n"; } (statement)+
{ wssProgram = wssProgram + "}\n"; } )
         )*


         (#(MULT { wssProgram = wssProgram+"else\n{\n"; } (statement)+ { wssProgram = wssProgram + "}\n"; } ))?
         ) | (#(MULT { wssProgram = wssProgram +"\n"; } (statement)+ { wssProgram = wssProgram + "\n"; } )
            ) )
         )
   ;
```

```
// Function_call
function_call
{
        Expr f;
}
        :#(LSQ ID {
                                WssType thisFun = current.get(#ID.getText());
                                if(thisFun == null){
                                errorFound = true;
                              System.err.println(" !! WSS Error (Type #2): Call of undefined function - "+#ID.getText()+"\n");

                        }
                         else{
                                if(thisFun.getType() != 3){
                                errorFound = true;
                                System.err.println(" !! WSS Error (Type #3): Variable used as a function name - "+#ID.getText());

                                System.err.println(" !! (" + #ID.getText()+" is a "+thisFun.toString()+" not a function! )\n");
                            }
                         }
                         wssProgram = wssProgram + #ID.getText()+"(";
                         }
           (f=expr { wssProgram = wssProgram+ f.gen()+","; })*
           { wssProgram = wssProgram + ");\n"; }
        )
        ;

// system statement
system_stmt
{
   Instrunit a,b;
}
   : #( "system" { wssProgram = wssProgram + "system \""; }
          a=instr_unit { wssProgram = wssProgram + a.gen(); }
          ( PIPE b=instr_unit { wssProgram = wssProgram+" | "+b.gen(); } )*
       { wssProgram = wssProgram + "\";\n"; }
     )
   ;
```

```
// instruction unit

instr_unit returns [ Instrunit iu ]
{
    iu=null;
}

// outside program

  : #( ID { String s = #ID.getText(); }
                              ( DOT ID { s=s+"."+#ID.getText(); } )?
      )
     ( STRING { s=s+" "+#STRING.getText(); }
       | NUMBER { s=s+" "+#NUMBER.getText(); }
       | #( ID { s=s+" "+#ID.getText(); }
                       ( DOT ID { s=s+"."+#ID.getText(); } )?
           )
       | ID_IN_SYSTEM
          {
              String a=#ID_IN_SYSTEM.getText();
              String b=a.substring(1,a.length());
              if(current.get(b) == null){
                          errorFound = true;
                   System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
              }
              else
                  s=s+" "+#ID_IN_SYSTEM.getText();
          }
       | GT #( ID { s=s+" > "+#ID.getText(); }
                       ( DOT ID { s=s+"."+#ID.getText(); } )?
               )
       | GGT #( ID { s=s+" >> "+#ID.getText(); }
                       ( DOT ID { s=s+"."+#ID.getText(); } )?
               )
       | LT #( ID { s=s+" < "+#ID.getText(); }
                       ( DOT ID { s=s+"."+#ID.getText(); } )?
               )
     )*
     { iu=new Instrunit(s); }
```

```
        |

// buildin function

        #( "echo" { String s = "echo"; } )
        ( STRING { s=s+" "+#STRING.getText(); }
           | NUMBER { s=s+" "+#NUMBER.getText(); }
           | ID { s=s+" "+#ID.getText(); }
           | ID_IN_SYSTEM
               {
                    String a=#ID_IN_SYSTEM.getText();
                    String b=a.substring(1,a.length());
                    if(current.get(b) == null){
                              errorFound = true;
                         System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
                    }
                    else
                        s=s+" "+#ID_IN_SYSTEM.getText();
               }
        )
        ( GT #( ID { s=s+" > "+#ID.getText(); }
                         ( DOT ID { s=s+"."+#ID.getText(); } )?
                 )
           | GGT #( ID { s=s+" >> "+#ID.getText(); }
                         ( DOT ID { s=s+"."+#ID.getText(); } )?
                 )
        )?
        { iu=new Instrunit(s); }

        |

        #( "dir" { String s = "dir "; } )
        (
           MULT { s=s+"*"; }
           | DOT { s=s+"."; }
           | #( ID { s=s+#ID.getText(); }
                         ( DOT ID { s=s+"."+#ID.getText(); } )?
             )
```

```
   | ID_IN_SYSTEM
       {
            String a=#ID_IN_SYSTEM.getText();
            String b=a.substring(1,a.length());
            if(current.get(b) == null){
                          errorFound = true;
                   System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
            }
            else
                s=s+" "+#ID_IN_SYSTEM.getText();
       }
)*
( GT #( ID { s=s+" > "+#ID.getText(); }
                  ( DOT ID { s=s+"."+#ID.getText(); } )?
           )
   | GGT #( ID { s=s+" >> "+#ID.getText(); }
                  ( DOT ID { s=s+"."+#ID.getText(); } )?
           )
)?
{ iu=new Instrunit(s); }

|

#( "ls" { String s = "dir "; } )
(
   MULT { s=s+"*"; }
   | DOT { s=s+"."; }
   | #( ID { s=s+#ID.getText(); }
                  ( DOT ID { s=s+"."+#ID.getText(); } )?
       )
   | ID_IN_SYSTEM
       {
            String a=#ID_IN_SYSTEM.getText();
            String b=a.substring(1,a.length());
            if(current.get(b) == null){
                          errorFound = true;
                   System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
            }
            else
```

```
                s=s+" "+#ID_IN_SYSTEM.getText();
        }
)*
(    GT #( ID { s=s+" > "+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
            )
   | GGT #( ID { s=s+" >> "+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
            )
)?
{ iu=new Instrunit(s); }


|


#( "type" { String s = "type "; } )
(
   #( ID { s=s+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
        )
   | ID_IN_SYSTEM
       {
            String a=#ID_IN_SYSTEM.getText();
            String b=a.substring(1,a.length());
            if(current.get(b) == null){
                        errorFound = true;
                    System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
            }
            else
                s=s+" "+#ID_IN_SYSTEM.getText();
       }
)
( GT #( ID { s=s+" > "+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
            )
   | GGT #( ID { s=s+" >> "+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
            )
)?
{ iu=new Instrunit(s); }
```

```
|

#( "cat" { String s = "type "; } )
(
   #( ID { s=s+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
        )
   | ID_IN_SYSTEM
      {
           String a=#ID_IN_SYSTEM.getText();
           String b=a.substring(1,a.length());
           if(current.get(b) == null){
                        errorFound = true;
                  System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
           }
           else
               s=s+" "+#ID_IN_SYSTEM.getText();
      }
)
( GT #( ID { s=s+" > "+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
             )
   | GGT #( ID { s=s+" >> "+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
             )
)?
{ iu=new Instrunit(s); }

|

#( "copy" { String s = "copy "; } )
(#( ID { s=s+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
 ) | ID_IN_SYSTEM
        {
           String a=#ID_IN_SYSTEM.getText();
           String b=a.substring(1,a.length());
           if(current.get(b) == null){
```

```
                    errorFound = true;
                System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
            }
            else
                s=s+" "+#ID_IN_SYSTEM.getText();
        } )
((#( ID { s=s+" "+#ID.getText(); }
                ( DOT ID { s=s+"."+#ID.getText(); } )?
  )) | ID_IN_SYSTEM
          {
            String a=#ID_IN_SYSTEM.getText();
            String b=a.substring(1,a.length());
            if(current.get(b) == null){
                    errorFound = true;
                System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
            }
            else
                s=s+" "+#ID_IN_SYSTEM.getText();
          } )

{ iu=new Instrunit(s); }


|


#( "cp" { String s = "copy "; } )
(#( ID { s=s+#ID.getText(); }
                ( DOT ID { s=s+"."+#ID.getText(); } )?
  ) | ID_IN_SYSTEM
        {
            String a=#ID_IN_SYSTEM.getText();
            String b=a.substring(1,a.length());
            if(current.get(b) == null){
                    errorFound = true;
                System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
            }
            else
                s=s+" "+#ID_IN_SYSTEM.getText();
        } )
((#( ID { s=s+" "+#ID.getText(); }
                         ( DOT ID { s=s+"."+#ID.getText(); } )?
  )) | ID_IN_SYSTEM
            {
              String a=#ID_IN_SYSTEM.getText();
              String b=a.substring(1,a.length());
              if(current.get(b) == null){
                              errorFound = true;
                        System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
              }
              else
                    s=s+" "+#ID_IN_SYSTEM.getText();
            } )

{ iu=new Instrunit(s); }


|


#( "move" { String s = "move "; } )
(#( ID { s=s+#ID.getText(); }
                         ( DOT ID { s=s+"."+#ID.getText(); } )?
  ) | ID_IN_SYSTEM
          {
              String a=#ID_IN_SYSTEM.getText();
              String b=a.substring(1,a.length());
              if(current.get(b) == null){
                              errorFound = true;
                        System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
              }
              else
                    s=s+" "+#ID_IN_SYSTEM.getText();
          } )
((#( ID { s=s+" "+#ID.getText(); }
                         ( DOT ID { s=s+"."+#ID.getText(); } )?
  )) | ID_IN_SYSTEM
            {
              String a=#ID_IN_SYSTEM.getText();
              String b=a.substring(1,a.length());
              if(current.get(b) == null){
                              errorFound = true;
                    System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
               }
               else
                    s=s+" "+#ID_IN_SYSTEM.getText();
             } )

{ iu=new Instrunit(s); }


|


#( "mv" { String s = "move "; } )
(#( ID { s=s+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
  ) | ID_IN_SYSTEM
          {
               String a=#ID_IN_SYSTEM.getText();
               String b=a.substring(1,a.length());
               if(current.get(b) == null){
                              errorFound = true;
                    System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
               }
               else
                    s=s+" "+#ID_IN_SYSTEM.getText();
          } )
((#( ID { s=s+" "+#ID.getText(); }
                    ( DOT ID { s=s+"."+#ID.getText(); } )?
  )) | ID_IN_SYSTEM
            {
               String a=#ID_IN_SYSTEM.getText();
               String b=a.substring(1,a.length());
               if(current.get(b) == null){
                              errorFound = true;
                    System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
               }
               else
                    s=s+" "+#ID_IN_SYSTEM.getText();
             } )

{ iu=new Instrunit(s); }
```

```
|

#( "del" { String s = "del "; } )
(
   MULT { s=s+"*"; }
   | DOT { s=s+"."; }
   | #( ID { s=s+#ID.getText(); }
                  ( DOT ID { s=s+"."+#ID.getText(); } )?
       )
   | ID_IN_SYSTEM
      {
          String a=#ID_IN_SYSTEM.getText();
          String b=a.substring(1,a.length());
          if(current.get(b) == null){
                      errorFound = true;
                 System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
          }
          else
              s=s+" "+#ID_IN_SYSTEM.getText();
      }
)*
{ iu=new Instrunit(s); }

|

#( "rm" { String s = "del "; } )
(
   MULT { s=s+"*"; }
   | DOT { s=s+"."; }
   | #( ID { s=s+#ID.getText(); }
                  ( DOT ID { s=s+"."+#ID.getText(); } )?
       )
   | ID_IN_SYSTEM
      {
          String a=#ID_IN_SYSTEM.getText();
          String b=a.substring(1,a.length());
          if(current.get(b) == null){
                      errorFound = true;
```

```
                        System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
                }
                else
                      s=s+" "+#ID_IN_SYSTEM.getText();
          }
)*
{ iu=new Instrunit(s); }

|

#( "grep" { String s = "grep "; } )
( #( ID { s=s+#ID.getText()+" "; } )
| ID_IN_SYSTEM { s=s+#ID_IN_SYSTEM.getText()+" "; } )
(
    MULT { s=s+"*"; }
    | DOT { s=s+"."; }
    | #( ID { s=s+#ID.getText(); }
                          ( DOT ID { s=s+"."+#ID.getText(); } )?
         )
    | ID_IN_SYSTEM
          {
                String a=#ID_IN_SYSTEM.getText();
                String b=a.substring(1,a.length());
                if(current.get(b) == null){
                                errorFound = true;
                        System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+b+"\n");
                }
                else
                      s=s+" "+#ID_IN_SYSTEM.getText();
          }
)*
( GT #( ID { s=s+" > "+#ID.getText(); }
                          ( DOT ID { s=s+"."+#ID.getText(); } )?
              )
    | GGT #( ID { s=s+" >> "+#ID.getText(); }
                          ( DOT ID { s=s+"."+#ID.getText(); } )?
              )
)?
```

```
      { iu=new Instrunit(s); }
   ;


// Return_stmt
return_stmt
          : #("return"
                    (
                     ID {
                                   if(current.get(#ID.getText()) == null){
                                             errorFound = true;
                                   System.err.println(" !! WSS Error (Type #1): Use of undeclared variable - "+#ID.getText()+"\n");
                                   }
                                   if(current.isRoot())
                                             wssProgram = wssProgram + "exit;\n";
                                   else{
                                                       current.parent.remove(current_func);
                            current.parent.put(current_func, new
                                                WssType("function",3,current.get(#ID.getText()).getReturnType() ));
                                                       wssProgram = wssProgram + "return $"+#ID.getText()+";\n";
                                             }
                           }
                     | ARG {
                                             if(current.isRoot())
                                                       wssProgram = wssProgram + "exit;\n";
                                             else{
                                                                 current.parent.remove(current_func);
                                                                 current.parent.put(current_func, new WssType("function",3,-99));


                                                                 String arg_text = #ARG.getText();
                                                                 String index = "";
                                                                 for(int i = 0; i < arg_text.length(); i++){
                                                                           if(arg_text.charAt(i) == '$')
                                                                                     ;
                                                                 else
                                                                           index = index+arg_text.charAt(i);
                                                                 }
                                                                 int temp = new Integer(index);
                                                                 index = Integer.toString(temp -1);
                                                                 wssProgram = wssProgram + "return $_["+index+"];\n";
```

```
                                    }
                                }
                | NUMBER {
                                if(current.isRoot())
                                            wssProgram = wssProgram + "exit;\n";
                                    else{
                                                current.parent.remove(current_func);
                                                current.parent.put(current_func, new WssType("function",3,1));
                                                wssProgram = wssProgram + "return $"+#NUMBER.getText()+";\n";
                                            }

                            }
                | STRING {
                                if(current.isRoot())
                                            wssProgram = wssProgram + "exit;\n";
                                    else{
                                                current.parent.remove(current_func);
                                                current.parent.put(current_func, new WssType("function",3,2));
                                                wssProgram = wssProgram + "return $"+#STRING.getText()+";\n";
                                            }
                            }
                 )
                )
            ;

// Break_stmt
break_stmt
        : #("break" { wssProgram = wssProgram +"break;\n"; })
        ;


// Continue_stmt
continue_stmt
        : #("continue" { wssProgram = wssProgram + "continue;\n"; })
        ;


expr returns [ Expr e ]
{
    e=null;
    Expr a,b;
```

```
}
        : NUMBER { e=new Expr(#NUMBER.getText()); e.type = 1; current.put(current_ID,WssType.Number);}
        | #(PLUS a=expr b=expr {
                                        if (a.type!=1)
                                        {
                                            errorFound = true;
                                            String k=a.gen();
                                            k=k.replace('$',' ');
                                    System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                                    System.err.println(" !! ("+k+" is a non-numeric operand ) \n");
                                        }
                                        if (b.type!=1)
                                        {
                                            errorFound = true;
                                            String k=b.gen();
                                            k=k.replace('$',' ');
                                     System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                                     System.err.println(" !! ("+k+" is a non-numeric operand ) \n");
                                        }

                                        String s="("+a.gen()+"+"+b.gen()+")";
                                        e=new Expr(s);
                                        e.type = 1;
                                    } )
        | #(MINUS a=expr b=expr {
                                        if (a.type!=1)
                                        {
                                            errorFound = true;
                                            String k=a.gen();
                                            k=k.replace('$',' ');
                                     System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                                      System.err.println(" !! ("+k+" is a non-numeric operand ) \n");
                                        }
                                        if (b.type!=1)
                                        {
                                            errorFound = true;
```

```
                                String k=b.gen();
                                k=k.replace('$',' ');
                        System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                         System.err.println(" !! ("+k+" is a non-numeric operand ) \n");
                            }

                            String s="("+a.gen()+"-"+b.gen()+")";
                            e=new Expr(s);
                            e.type = 1;
                        } )
| #(MULT a=expr b=expr {
                            if (a.type!=1)
                            {
                                errorFound = true;
                                String k=a.gen();
                                k=k.replace('$',' ');
                        System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                             System.err.println(" !! ("+k+" is a non-numeric operand ) \n");

                            }
                            if (b.type!=1)
                            {
                                errorFound = true;
                                String k=b.gen();
                                k=k.replace('$',' ');
                        System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                             System.err.println(" !! ("+k+" is a non-numeric operand ) \n");

                            }

                            String s="("+a.gen()+"*"+b.gen()+")";
                            e=new Expr(s);
                            e.type = 1;
                       } )
| #(DIV a=expr b=expr {
                            if (a.type!=1)
```

```
                                                  {
                                                        errorFound = true;
                                                        String k=a.gen();
                                                        k=k.replace('$',' ');
                                          System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                                                    System.err.println(" !! ("+k+" is a non-numeric operand ) \n");

                                                  }
                                                  if (b.type!=1)
                                                  {
                                                        errorFound = true;
                                                        String k=b.gen();
                                                        k=k.replace('$',' ');
                                           System.err.println(" !! WSS Error (Type #4): Invalid operand in arithmetic operation - "+k);

                                                    System.err.println(" !! ("+k+" is a non-numeric operand ) \n");

                                                  }

                                                  String s="("+a.gen()+"/"+b.gen()+")";
                                                  e=new Expr(s);
                                                  e.type = 1;
                                            } )
| ID {
          int tempType = current.get(#ID.getText()).getReturnType();
            if(tempType == 1)
                  current.put(current_ID,WssType.Number);
            else
                  current.put(current_ID,WssType.String);

        e=new Expr("$"+#ID.getText());
        e.type=tempType;
       }
| ARG { String arg_text = #ARG.getText();
        String index = "";
        for(int i = 0; i < arg_text.length(); i++){
            if(arg_text.charAt(i) == '$')
                  ;
```

```
                    else
                            index = index+arg_text.charAt(i);
               }
               int temp = new Integer(index);
               index = Integer.toString(temp -1);
               e = new Expr("$_["+index+"]");
          }
     | #(LSQ ID {
                                   WssType thisFun = current.get(#ID.getText());
                                   if(thisFun == null){
                                   errorFound = true;
                    System.err.println(" !! WSS Error (Type #2): Call of undefined function - "+#ID.getText()+"\n");
                                   }
                                   else{
                                           if(thisFun.getType() != 3){
                                                   errorFound = true;
                            System.err.println(" !! WSS Error (Type #3): Variable used as a function name - "+#ID.getText());

                            System.err.println(" !! (" + #ID.getText()+" is a "+thisFun.toString()+" not a function! )\n");
                                           }

                                           if((thisFun.getReturnType()==0) || (thisFun.getReturnType()==2)){
                                                   errorFound = true;
                    System.err.println(" !! WSS Error (Type #5): Invalid operand in arithmetic operation - "+#ID.getText());

            System.err.println(" !! (The function - "+#ID.getText()+" returns a non-numeric operand <string or void?> )\n");
                            }
                                   }
                                   String s = #ID.getText()+"("; }
        (a=expr { s = s+a.gen()+","; })*
        { s=s+")"; e = new Expr(s); }
     )
   | STRING { String s = ""+#STRING.getText()+"";
              e=new Expr(s);
              e.type=2; }
     ;
```