**Project Homework 1**

**Please understand the program’s structure and design, your future coding work will be based on it.**

**For some of these questions, you could test your answer by running the code!**

**All of these questions have been discussed in the lab except some needing your trying, if you feel too much lost, find a friend in this class for help, but the solution should be finished independently.**

Question 1:

In the *parser\_hw1.l* file, how to understand the following sentences:

**EXP ([Ee][-+]?[0-9]+)**

This code takes one in one expression in the beginning. It takes E or e, one – or +, and then at least one more number. This will look like 10e+65768735, which is a float exponent.

**[a-zA-Z][a-zA-Z0-9]\***

This code can start with any lowercase or uppercase letter, followed by any number of letters or digits from 0-9. This can be a word or a variable.

**[0-9]+"."[0-9]\*{EXP}?**

This code accepts at least one digit, followed by a period, followed by any number of digits 0-9. This can then be assigned to a pattern name EXP.

**"."?[0-9]+{EXP}?**

This code begins with an optional decimal point, followed by at least one number 0-9. It can then be assigned to a pattern name EXP. It is a decimal. One example: .0

Please give each of them one sentence of English explanation and one language example.

Question 2:

In the *parser\_hw1.l* file, why the scanner could understand “>” and “>=”; or “=” and “==” without ambiguity? Write down the proper code for scanning the symbols “ **&** ”, “ **&&** ” and “ **&&[a-zA-Z][a-zA-Z0-9]\*** ”, if these symbols exist.

The scanner can differentiate between >, >=, =, and == because all of these symbols will get different yylval.fn values

[&] {return yytext[0]; }

“&&” {return ANDOP; }

&&[a-zA-Z][a-zA-Z0-9]\* { yylval.strval = strdup(yytext); return NAME; }

Question 3:

If we have keyword “**for**” and array type(for example, arr[3]; the data type is not required and the length is required as a constant integer), write down the proper scanning code for them.

“for” [a-zA-Z]+[a-zA-Z0-9]\* \[ [0-9]+ \]

Question 4:

In the *parser\_hw1.l* file, all the comparison operation and function have a value as “**fn**”, you could see it from for example:

**">" { yylval.fn = 1; return CMP; }**

**"sqrt" { yylval.fn = B\_sqrt; return FUNC; }**

Which part of codes in the *parser\_hw1.y* file support this feature and how it is implemented?

%union {

struct ast\* a;

double d;

struct symbol\* s; /\* which symbol \*/

struct symlist\* sl;

int fn; /\* which function \*/

}

This code is responsible for dealing with the fn values. When scanned in, each of these gets a value in yylval.fn. While building the tree, the compiler will see “<” or “sqrt”, and then perform the respective comparison function or square root function, which are defined elsewhere in the code. The order of precedence is also considered when implementing this feature.

What will happen and why if the code for “else” is:

**"else" {yylval.fn = 2; return ELSE; }**

Earlier in the file, code has “<” equal to yylval. Therefore, when the parser sees else it will instead perform a less-than operation.

Question 5:

In the *parser\_hw1.y* file, please list all the terminals and all the non-terminals.

Non-terminals: < > <> == >= <= if then else while do let sqrt exp log print list stmt expList symList prog

Terminals: + - \* / = | , ; ( ) NAME NUMBER . EOL

Question 6:

In the *parser\_hw1.y* file, how does the code show the precedence of different operations? Totally how many precedence levels are in this gramma? Which has higher precedence, comparison or assignment?

%nonassoc <fn> CMP

%right '='

%left '+' '-'

%left '\*' '/'

%nonassoc '|' UMINUS

This code establishes the order of precedence. There are 5 levels of precedence, and they are ordered from lowest to highest precedence. Comparison is listed first, so it has the lowest order of precedence. Assignment comes after, so it has a higher precedence than assignment.

Question 7:

Based on your understanding of the grammar, is the following program working or not?

let func(begin, size, step) = while size > 0 do begin = begin + step; size = size – 1;;

If this is not working, please modify it to function properly and use the grammar rules with the C file code to explain the reason.

This program will not work because of numerous syntax errors and the wrong ordering of do-while elements. I would reorder it to look like this:

let func(begin, size, step)

do {

begin = begin + step;

size = size – 1;

} while (size > 0) ;

Question 8:

Write three of your defined functions and nest them like inside func2, func1 is called; inside func3, func2 is called; then call all your functions one by one and print the result.

Let func2(a) = x = 0; func1(x);

Let func1(x) = x = x + 1; func3(x);

Let func3(x) = x = x + 1;;

Question 9:

To show the fully understanding of the grammar, write a function to update three global variables defined before and do nothing more. So this function works like a procedure.

X = 0

Y = 1

Z = 3

let myFunc(a) = x = x + 1; y = y + 2; z = z + 3; 1;;

Question 10:

After analysis on the design of the “parser”, explain how it deals and manages the local and global variables. What is the serious flaw in this design? Use code examples for explanation.

/\* symbol table \*/

struct symbol { /\* a variable name \*/

char\* name;

double value;

struct ast\* func; /\* stmt for the function \*/

struct symlist\* syms; /\* list of dummy args \*/

};

This parser stores al variables and functions in the symTable. It stores all of them on the same level, so all variables and functions are technically global variables because they can all be modified. To implement locality, you would need a more complex structures that gives each variable and functions a specific level.