CSE 3212
Compiler Design Laboratory

submitted by

MD. Mahmud Ur Rahman

Roll no: 1107003

Introduction:

Flex (fast lexical analyzer generator) is a free software alternative to lex It is a computer program that generates lexical analyzers ("scanners" or "lexers"). It is frequently used with the free Bison parser generator. Unlike Bison, flex is not part of the GNU project. Flex was written in C by Vern Perxson around 1987.He was translating a Raftor generator, which had been led by Jef Poskanzer.

GNU bison, commonly known as Bison, is a parser generator that is part of the GNU project. Bison reads specification of a context free language, warns about any parsing ambiguities, and generates a parser (either in C, C++, or Java) which reads sequences of tokens and decides whether the sequence conforms to the syntax specified by the grammar. Bison by default generates LALR parsers but can also create GLR parser.

Keywords:

Keywords name	**Actual meaning**
add	+
sub	-
mul	*
div	/
_if	IF
_else	ELSE
>>	Greater than (>)
<<	Smaller than (<)
pow	POWER (^)
_fact	FACTORIAL (!)
_tan	TANGENT (tan)
_sin	SINE (sin)
_cos	COSINE (cos)
NUMBER	[0-9]+
VARIABLE	[a-z]

Flex file:

```
/* C Declarations */

%{
	#include<stdio.h>
	#include "1107003.tab.h"
	#include<stdlib.h>
	extern int yylval;
%}

/* RE and Actions */

%%

[0-9]+	{
		yylval = atoi(yytext);
		return NUM;
	}

[a-z] {
		yylval = *yytext - 'a';
		return	VAR;
	}

"_if" { return IF;	}
"_else"	{ return ELSE;  }
"_add_"  {return ADD;}
"_sub_" {return SUB;}
"_mul_" {return MUL;}
"_div_"  {return DIV;}
```

```
">>" {return GREATER;}
"<<" {return SMALLER;}
"_pow_"  {return POWER;}
"_fact"    {return FAC;}
"_tan"    {return TAN;}
"_sin"    {return SIN;}
"_cos"    {return COS;}
[-+/*<>=,();]  {
                    yylval = yytext[0];
                    return *yytext;
               }

[ \t\n]*     ;

.      {
               yyerror("Unknown Character.\n");
       }
%%

int yywrap()
{

return 1;
}

main(){
yyin=fopen("input.txt","r");
       yyparse();

}
```

Bison program:

```
/* C Declarations */

%{
	#include<stdio.h>
	#include<math.h>
	double PI = 3.1416;
	int sym[26];

%}



/* bison declarations */

%token NUM VAR ADD SUB MUL DIV IF ELSE GREATER SMALLER
POWER FAC SIN COS TAN
%nonassoc IFX
%nonassoc ELSE
%left GREATER SMALLER
%left ADD SUB POWER FAC SIN COS TAN
%left MUL DIV
```

```
/* Grammar rules and actions follow.  */

%%

program: /* EMPTY INPUT */

        | program statement
        ;

statement: ';'

        | expression ';'                    { printf("value of expression:
%d\n", $1); }

        | VAR '=' expression ';'            {
                                                sym[$1] = $3;
                                                printf("value of the variable:
%d\t\n",$3);
                                        }

        | IF '(' expression ')' expression ';' %prec IFX {
                                                if($3){
                                                        printf("\nvalue of
expression in if: %d\n",$5);
                                                }
                                                else{
                                                printf("condition value zero in
else block\n");
                                                }
```

```
                                        }

        | IF '(' expression ')' expression ';' ELSE expression ';' {
                                                        if($3){
                                                                printf("value of the
expression in if: %d\n",$5);
                                                        }
                                                        else{
                                                                printf("value of the
expression in else: %d\n",$8);
                                                        }
                                                }
        ;

expression: NUM                         { $$ = $1;      }

        | VAR                           { $$ = sym[$1]; }

        | expression POWER expression { $$ = pow($1,$3); }

        | expression FAC  {

         int total=1;
                int n=$1;
                for(;n>0;n--)
                {

                total*=n;
                }
                $$=total;
```

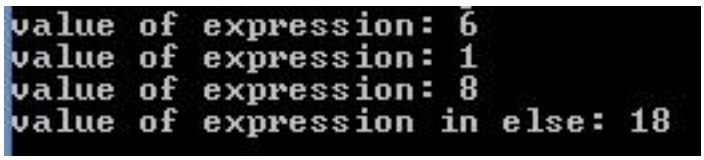
```
}

| TAN expression {

        $$=tan($2*(180/PI));
}

| SIN expression {

        $$=sin($2*(180/PI));
}

| COS expression {

        $$=cos($2*(180/PI));
}

| expression ADD expression    { $$ = $1 + $3; }

| expression SUB expression    { $$ = $1 - $3; }

| expression MUL expression    { $$ = $1 * $3; }

| expression DIV expression    {     if($3)
                               {
                                     $$ = $1 / $3;
                               }
                               else
                               {
```

```
                                        $$ = 0;
                                        printf("\ndivision by zero\t");
                                    }
                                }

        | expression SMALLER expression     { $$ = $1 < $3; }

        | expression GREATER expression     { $$ = $1 > $3; }

        | '(' expression ')'            { $$ = $2;}
        ;
%%

yyerror(char *s){
        printf( "%s\n", s);
}
```

Input File:

```
(2_add_5_sub_2)_mul_3_mul_2_div_10_fact;
_cos(0);
2_pow_3;
_if(2>>9)2_add_9;
_else 2_mul_9;
```

Output File:



Conclusion:

In this lab we were introduced about parsing and semantic analyzing. This knowledge will help us in future to develop not only compiler but also some other fields like natural language processing.