**Compiler Lab Report**

**Experiment: 04**

**Submitted By**

Md. Mushfekur Rahman

Roll – 01

15th Batch

Dept. of Computer Science & Engineering

University of Dhaka

**Name of the Experiment**

Make a Calculator using Flex and Bison

**Source Code (Lex)**

%{

#define YYSTYPE double

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

void yyerror(char \*);

extern double yylval;

#include "y.tab.h"

%}

%%

SQRT { return SQRT; }

LOG { return LOG; }

EXP { return EXP; }

[a-zA-Z\_]+ { yylval = \*yytext - 'a';

return VAR; }

[0-9.]\*[0-9]+ { yylval = atof(yytext);

return NUM; }

[-+()=^/\*\n] { return \*yytext; }

\[ { return \*yytext; }

] { return \*yytext; }

[ \t] ;

[$] { return 0; }

. yyerror("invalid character");

%%

int main(int argc, char \*argv[]) {

yyin = fopen(argv[2], "r");

yylex();

fclose( yyin );

return 0;

}

int yywrap(void) {

return 1;

}

**Source Code (Bison)**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

typedef double (\*func\_t) (double);

struct symrec {

char \*name;

int type;

union {

double var;

func\_t fnctptr;

} value;

struct symrec \*next;

};

typedef struct symrec symrec;

extern symrec \*sym\_table;

symrec \*putsym (char const \*, int);

symrec \*getsym (char const \*);

void yyerror(char const \*s);

int yylex(void);

%}

%union { double val; symrec \*tptr; }

%token <val> NUM

%token <tptr> VAR FUNC

%type <val> expr

%right '='

%left '-' '+'

%left '\*' '/'

%left NEG

%right '^'

%%

input:

| input line

;

line:

'\n' { /\* do nothing \*/ }

| expr '\n' { printf("%.10g\n",$1); }

| error '\n' { yyerrok; }

;

expr:

NUM { $$ = $1; }

| VAR { $$ = $1->value.var; }

| VAR '=' expr { $$ = $3; $1->value.var = $3; }

| FUNC '(' expr ')' { $$ = (\*($1->value.fnctptr))($3); }

| expr '+' expr { $$ = $1 + $3; }

| expr '-' expr { $$ = $1 - $3; }

| expr '\*' expr { $$ = $1 \* $3; }

| expr '/' expr { if( $3 ) { $$ = $1 / $3; }

else { $$ = 1; fprintf(stderr, "%s", "Divide by zero error!");}

}

| '-' expr %prec NEG { $$ = -$2; }

| expr '^' expr { $$ = pow($1, $3); }

| '(' expr ')' { $$ = $2; }

| '[' expr ']' { $$ = $2; }

;

%%

struct init {

char const \*fname;

double (\*fnct) (double);

};

struct init const lib\_funcs[] = {

"SIN", sin,

"ASIN", asin,

"COS", cos,

"ACOS", acos,

"TAN", tan,

"ATAN", atan,

"LOG", log,

"EXP", exp,

"SQRT", sqrt,

0, 0

};

symrec \*sym\_table;

void init\_table(void) {

int i;

for(i=0; lib\_funcs[i].fname!=0; i++) {

symrec \*ptr = putsym(lib\_funcs[i].fname, FUNC);

ptr->value.fnctptr = lib\_funcs[i].fnct;

}

}

int main(int argc, char \*argv[]) {

init\_table();

return yyparse();

}

void yyerror(char const \*s) {

printf("%s\n",s);

}

int yylex(void)

{

int c;

while((c = getchar ()) == ' ' || c == '\t') continue;

if((c == EOF) || (c == '$')) return 0;

if(c == '.' || isdigit (c)) {

ungetc (c, stdin);

scanf("%lf", &yylval.val);

return NUM;

}

if( isalpha(c) || (c=='\_')) {

static size\_t length = 40;

static char \*symbuf = 0;

symrec \*s;

if( !symbuf ) symbuf = (char \*) malloc (length + 1);

int i = 0;

do {

if(i == length) {

length \*= 2;

symbuf = (char \*) realloc (symbuf, length + 1);

}

symbuf[i++] = c;

c = getchar();

}

while( isalnum(c) || (c=='\_'));

ungetc (c, stdin);

symbuf[i] = 0;

s = getsym (symbuf);

if( !s ) s = putsym (symbuf, VAR);

yylval.tptr = s;

return s->type;

}

return c;

}

symrec\* putsym (char const \*sym\_name, int sym\_type) {

symrec\* ptr = (symrec \*) malloc (sizeof (symrec));

ptr->name = (char \*) malloc (strlen (sym\_name) + 1);

strcpy (ptr->name,sym\_name);

ptr->type = sym\_type;

ptr->value.var = 0;

ptr->next = (struct symrec \*)sym\_table;

sym\_table = ptr;

return ptr;

}

symrec\* getsym (char const \*sym\_name) {

symrec\* ptr;

for(ptr = sym\_table; ptr != (symrec \*) 0;

ptr = (symrec \*)ptr->next)

if( !strcmp (ptr->name,sym\_name) ) return ptr;

return 0;

}

**Sample Input Output**

**2/3+1**

= 1.666667

**(4\*(1+2)^3)/6**

=18.000000

**a = 2 + 6.5 / 3.25**

**a**

= 4.000000

**my\_variable = SQRT( a ) + 6**

**my\_variable/a**

= 5.000000

**LOG( 512.00) \* EXP(1.00)**

=24.464536