|  |  |  |
| --- | --- | --- |
| **Compiler Design Laboratory**  |  | | --- | | **Course No:** CSE 3212 | |  |   Project Name: Implementation of Syntax Analyzer |

|  |
| --- |
| **Submitted By:** Ishrak islam zarif  ROLL : 1307025 |

**Problem Statement:**

In this lab, we were assigned to write a program to design a syntax analyzer of our own that will include the followings:

1. Expression evaluation
2. Control flow (if then else)
3. Switch –case statement
4. Loop (optional but strongly recommended)

**Flex Program:**

%{

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "1307025.tab.h"

%}

%%

[-]?[0-9]\*[.]?[0-9]+ { yylval.flt = atof(yytext); return NUM; }

[a-z] { yylval.dec = \*yytext - 'a'; return ID; }

"main method" { return MAIN; }

"assign" { return ASSIGN; }

"if" { return IF; }

"else" { return ELSE; }

"var" { return VAR; }

[~].+ { return HEADER; }

"sin" { return SIN; }

"cos" { return COS; }

"tan" { return TAN; }

"log" { return LOG; }

"print" { return PRINT; }

"fact" { return FACT; }

"gcd" { return GCD; }

"lcm" { return LCM; }

"loop" { return LOOP; }

"switch" { return SWITCH; }

"case" { return CASE; }

"default" { return DEFAULT; }

[-+/\*^<>,(){}:`] { return \*yytext; }

[ \t\n]\* {}

"--".+ {}

"$\*"(.\*|\n)\*"\*$" {}

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

%%

int main(){

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

yyparse();

return 0;

}

**Bison Program:**

%{

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define pi 3.14159265358979323846264

double sym[26];

double default\_val;

double switch\_val;

double flag=0;

%}

%union{

int dec;

double flt;

}

%token<dec> ID

%token<flt> NUM

%token VAR MAIN IF ELSE HEADER SIN COS TAN LOG PRINT FACT GCD LCM ASSIGN SWITCH CASE DEFAULT LOOP

%type<flt> statement expression start

%nonassoc IFX

%nonassoc ELSE

%left '<' '>'

%left '+' '-'

%left '\*' '/'

%right '^'

%left UMINUS

%left FACT

%%

program: headers MAIN '(' ')' '{' fstatement '}'

;

headers: /\* empty \*/

| headers HEADER

;

fstatement: /\* empty \*/

| fstatement statement

;

statement: '`'

| declaration '`'

| expression '`' { printf("%lf\n", $1); }

| ID ASSIGN expression '`' { sym[$1] = $3; }

| IF ':' expression ':' expression '`' %prec IFX {

if($3) printf("value of expression in IF: %lf\n", $5);

else printf("condition value zero in IF block\n");

}

| IF ':' expression ':' expression '`' ELSE expression '`' {

if($3)

printf("value of expression in IF: %lf\n", $5);

else

printf("value of expression in ELSE: %lf\n", $8);

}

| switchcase {

if(flag == 0)

printf("%lf\n", default\_val);

}

| LOOP '(' ID ':' expression ',' expression ')' '(' expression '`' ')' {

int i;

for(i=$5; i<=$7; i++)

printf("value of expression in loop: %lf\n", $10);

}

;

declaration : VAR ids

;

ids : ids ',' ID

| ID

;

expression: NUM { $$ = $1; }

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

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

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

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

| expression '/' expression {

if($3)

$$ = $1 / $3;

else{

$$ = 0;

printf("\ndivision by zero\n");

}

}

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

| '-' expression %prec UMINUS { $2 = -$2; }

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

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

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

| FACT '(' NUM ')' { $$ = factorial((int)$3) }

| GCD '(' NUM ',' NUM ')' { $$ = GCD\_fn($3, $5) }

| LCM '(' NUM ',' NUM ')' { $$ = LCM\_fn($3, $5) }

| SIN '(' expression ')' { $$ = sin($3 \* pi / 180); }

| COS '(' expression ')' { $$ = cos($3 \* pi / 180); }

| TAN '(' expression ')' { $$ = tan($3 \* pi / 180); }

| LOG '(' expression ')' { $$ = log10($3); }

| PRINT '(' expression ')' { $$ = $3; }

;

switchcase: start DEFAULT ':' expression '`' { default\_val = $4; }

;

start: SWITCH '(' expression ')' ':' { switch\_val = $3; }

| start CASE '(' expression ')' ':' expression '`' {

if($4 == switch\_val){

printf("%lf\n", $7);

flag=1;

}

}

%%

int yywrap(){

return 1;

}

yyerror(char \*s){

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

}

int factorial(int num){

int i;

long x=1;

for(i=1; i<=num; i++){

x=x\*i;

}

return x;

}

int GCD\_fn(double n1, double n2){

int i, gcd, mul, n3, n4;

n3=(int)n1;

n4=(int)n2;

mul=(n3\*n4);

for(i=1; i <= n3 && i <= n4; ++i)

{

if(n3%i==0 && n4%i==0)

gcd = i;

}

return gcd;

}

int LCM\_fn(double n1, double n2){

int i, gcd, mul, lcm, n3, n4;

n3=(int)n1;

n4=(int)n2;

mul=(n3\*n4);

for(i=1; i <= n3 && i <= n4; ++i)

{

if(n3%i==0 && n4%i==0)

gcd = i;

}

lcm=(mul/gcd);

return lcm;

}

**Input:**

~MyHeader1

~MyHeader2

main method()

{

--single line comment

$\*

multi line comment

\*$

var a, b, c, d, e, f, g, i`

c assign sin(90)`

a assign -2 - 5`

print(a)`

b assign a ^ 2`

print(b)`

print(c)`

if:2 > 1:

1 + 2`

if:5 < 2:

5 - 2`

else

1 + ( 2 ^ 3 ) \* 2`

switch(2):

case(1): 100`

case(2): 200`

case(3): 300`

default: 500`

d assign log(1000)`

loop(i : 1, 5)(

print(d)`

)

e assign fact(5)`

print(e)`

f assign gcd(5, 10)`

print(f)`

g assign lcm(5, 10)`

print(g)`

}

**Output:**

-7.000000

49.000000

1.000000

value of expression in IF: 3.000000

value of expression in ELSE: 17.000000

200.000000

value of expression in loop: 3.000000

value of expression in loop: 3.000000

value of expression in loop: 3.000000

value of expression in loop: 3.000000

value of expression in loop: 3.000000

120.000000

5.000000

10.000000

**Conclusion:**

In this lab I have learned about the lexical analyzer and syntax analyzer. I have also learned how it works. I have also implemented it to find the type, identifier, operators, and reserve words. At the time of implementing this syntax analyzer I have faced a lot of problem and errors. By debugging the program, I have removed some of the errors. I have tried my level best to make my program error fee. From this assignment I have known about the compiler design and top down parsing technique. I have also implemented the top down parsing technique in C language. At the time of implementation, I have faced some problems and solved these problems. At the end I became successful to implement the grammar using top down parsing technique and evaluate the value by using syntax analyzer.