**WEEK - 9 Intermediate Code Generation UsingYACC**

**Lab Program - 9**

1. Parser generator using YACC (Calculator)

Code:

**Calculator.l**

%{

#include "y.tab.h"

#include<math.h>

%}

%%

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

return NUMBER;

}

log|LOG {return LOG;}

ln {return nLOG;}

sin|SIN {return SINE;}

cos|COS {return COS;}

tan|TAN {return TAN;}

mem {return MEM;}

[\t];

\$; {return 0;}

\n|. {return yytext[0];}

**%%**

**Calculator.y**

%{

#include<stdio.h>

#include<math.h>

double memvar;

%}

%union

{

double dval;

}

%token<dval>NUMBER

%token<dval>MEM

%token LOG SINE nLOG COS TAN

%left '-''+'

%left '\*''/'

%right '^'

%left LOG SINE nLOG COS TAN

%nonassoc UMINUS

%type<dval> expression

%%

start: statement '\n'

|start statement '\n'

;

statement: MEM'='expression { memvar=$3;}

|expression {printf("answer=%g\n",$1);}

;

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

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

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

|expression'/'expression

{

if($3==0)

yyerror("divide by zero");

else

$$=$1/$3;}

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

;

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

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

|LOG expression {$$=log($2)/log(10);}

|nLOG expression {$$=log($2);}

|SINE expression {$$=sin($2\*3.14159/180);}

|COS expression {$$=cos($2\*3.14159/180);}

|TAN expression {$$=tan($2\*3.14159/180);}

|NUMBER { $$ = $1;}

|MEM {$$=memvar;}

;

%%

main()

{

printf("enter expression:");

yyparse();

}

int yyerror(char \*error)

{

fprintf(stderr,"%s\n",error);

}

yywrap() { return 1; }

OUTPUT:

$ ./calculator

Enter expression:

12 + 24

answer=36

Enter expression:

sin 30

answer=0.5

Enter expression:

log 100

answer=2

Enter expression:

10 / 0

divide by zero

Enter expression:

mem = 15

C program for Three address code generation

**Code:**

**three.l**

%{

#include "y.tab.h"

extern char yyval;

%}

number [0-9]+

letter [a-zA-Z]+

%%

{number} {yylval.sym=(char)yytext[0];return number;}

{letter} {yylval.sym=(char)yytext[0]; return letter; }

\n {return 0;}

. {return yytext[0];}

%%

**Three.y**

%{

#include<stdio.h>

#include<string.h>

int nIndex=0;

struct Intercode

{

char operand1;

char operand2;

char opera;

};

%}

%union

{

char sym;

}

%token <sym> letter number

%type <sym> expr

%left '-' '+'

%right '\*' '/'

%%

statement: letter '=' expr ';' { addtotable((char)$1,(char)$3,'=' ); }

| expr ;

;

expr: expr '+' expr { $$=addtotable((char)$1,(char)$3,'+');}

| expr '-' expr { $$=addtotable((char)$1,(char)$3,'-');}

| expr '\*' expr { $$=addtotable((char)$1,(char)$3, '\*');}

| expr '/' expr { $$=addtotable((char)$1,(char)$3,'/');}

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

| number { $$= (char)$1;}

| letter { $$= (char)$1;}

%%

yyerror(char \*s)

{

printf("%s",s);

exit (0);

}

struct Intercode code[20];

char addtotable(char operand1, char operand2,char opera)

{

char temp = 'A';

code[nIndex].operand1 = operand1;

code[nIndex].operand2 = operand2;

code[nIndex].opera = opera;

nIndex++;

temp++;

return temp;

}

threeaddresscode()

{

int nCnt=0;

char temp='A';

printf("\n\n\t three addrtess codes\n\n");

temp++;

while(nCnt<nIndex)

{

printf("%c:=\t",temp);

if (isalpha(code[nCnt].operand1))

printf("%c\t", code[nCnt].operand1);

else

printf("%c\t",temp);

printf("%c\t", code[nCnt].opera);

if (isalpha(code[nCnt].operand2))

printf("%c\t", code[nCnt].operand2);

else

printf("%c\t",temp);

printf("\n");

nCnt++;

temp++;

}}

main()

{

printf("enter expression");

yyparse();

threeaddresscode();

}

yywrap()

{

return 1;

**}**

**Output:**

$ lex three.l

$ yacc -d three.y

$ gcc lex.yy.c y.tab.c -ll -lm

$ ./a.out

enter expression (a\*b)+(c\*d)

three addrtess codes

B:= a \* b

C:= c \* d

D:= B + B

YACC program for generate Quadruple.

**Code:**

**/\* lex program \*/**

%{

#include "y.tab.h"

extern char yyval;

%}

number [0-9]+

letter [a-zA-Z]+

%%

{number} {yylval.sym=(char)yytext[0];return number;}

{letter} {yylval.sym=(char)yytext[0]; return letter; }

\n {return 0;}

. {return yytext[0];}

**%%**

**/\*yaac program \*/**

%{

#include<stdio.h>

#include<string.h>

int nIndex=0;

struct Intercode

{

char operand1;

char operand2;

char opera;

};

%}

%union

{

char sym;

}

%token <sym> letter number

%type <sym> expr

%left '-' '+'

%right '\*' '/'

%%

Statement: letter '=' expr ';' { addtotable((char)$1,(char)$3,'=' ); }

| expr ;

;

expr: expr '+' expr { $$=addtotable((char)$1,(char)$3,'+');}

| expr '-' expr { $$=addtotable((char)$1,(char)$3,'-');}

| expr '\*' expr { $$=addtotable((char)$1,(char)$3, '\*');}

| expr '/' expr { $$=addtotable((char)$1,(char)$3,'/');}

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

| number { $$= (char)$1;}

| letter { $$= (char)$1;}

%%

yyerror(char \*s)

{

printf("%s",s);

exit (0);

}

struct Intercode code[20];

char addtotable(char operand1, char operand2,char opera)

{

char temp = 'A';

code[nIndex].operand1 = operand1;

code[nIndex].operand2 = operand2;

code[nIndex].opera = opera;

nIndex++;

temp++;

return temp;

}

threeaddresscode()

{

int nCnt=0;

char temp='A';

printf("\n\n\t three addrtess codes\n\n");

temp++;

while(nCnt<nIndex)

{

printf("%c:=\t",temp);

if (isalpha(code[nCnt].operand1))

printf("%c\t", code[nCnt].operand1);

else

printf("%c\t",temp);

printf("%c\t", code[nCnt].opera);

if (isalpha(code[nCnt].operand2))

printf("%c\t", code[nCnt].operand2);

else

printf("%c\t",temp);

printf("\n");

nCnt++;

temp++;

}

}

void quadruples()

{

int nCnt=0;

char temp = 'A';

temp++;

printf("\n\n\t Quardruples \n");

printf("\n ID OPERATOR OPERAND1 OPERAND2\n");

while(nCnt<nIndex)

{

printf("\n (%d) \t %c \t",nCnt,code[nCnt].opera);

if(isalpha(code[nCnt].operand1))

printf("%c\t", code[nCnt].operand1);

else

printf("%c\t",temp);

printf("%c\t", code[nCnt].opera);

if(isalpha(code[nCnt].operand2))

printf("%c\t", code[nCnt].operand2);

else

printf("%c\t",temp);

printf("%c\t",temp);

printf("\n");

nCnt++;

temp++;

}

}

main()

{

printf("enter expression");

yyparse();

threeaddresscode();

quadruples();

}

yywrap()

{

return 1;

**}**

**Output:**

$ lex lexfile.l

$ yacc –d yaccfile.y

$cc lex.yy.c y.tab.c –ll –ly –lm

$./a.out

Enter expr a+b\*c+d

Three address code

B=b\*c

C=a+B

D=B+d

ID OPERATOR OPERAND1 OPERAND2

(0) \* b c B

(1) + a B C

(2) + B d D