**SUVASH CHANDRA YADAV:**

**119CS0178**

**6.Design a simple calculator.**

***yacc – program***

%{

#include<stdio.h>

int regs[26];

int base;

%}

%union { int a; }

%token DIGIT LETTER

%left '|'

%left '&'

%left '+' '-'

%left '\*' '/' '%'

%left UMINUS /\*supplies precedence for unary minus \*/

%type <a> stat expr number DIGIT LETTER

%% /\* beginning of rules section \*/

list: list stat '\n'

|

list error '\n'

{

yyerrok;

}

| /\*empty \*/

;

stat: expr

{

printf("%d\n",$1);

}

|

LETTER '=' expr

{

regs[$1] = $3;

}

;

expr: '(' expr ')'

{

$$ = $2;

}

|

expr '\*' expr

{

$$ = $1 \* $3;

}

|

expr '/' expr

{

$$ = $1 / $3;

}

|

expr '%' expr

{

$$ = $1 % $3;

}

|

expr '+' expr

{

$$ = $1 + $3;

}

|

expr '-' expr

{

$$ = $1 - $3;

}

|

expr '&' expr

{

$$ = $1 & $3;

}

|

expr '|' expr

{

$$ = $1 | $3;

}

|

'-' expr %prec UMINUS

{

$$ = -$2;

}

|

LETTER

{

$$ = regs[$1];

}

|

number

;

number: DIGIT

{

$$ = $1;

base = ($1==0) ? 8 : 10;

}

|

number DIGIT

{

$$ = base \* $1 + $2;

}

;

%%

main()

{

return(yyparse());

}

yyerror(s)

char \*s;

{

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

}

yywrap()

{

return(1);

}

***Lex- Program***

%{

#include <stdio.h>

#include "calc.tab.h"

int c;

%}

%%

" " ;

[a-z] {

c = yytext[0];

yylval.a = c - 'a';

return(LETTER);

}

[0-9] {

c = yytext[0];

yylval.a = c - '0';

return(DIGIT);

}

[^a-z0-9\b] {

c = yytext[0];

return(c);

}

%%

Input: Output:

5+5 10

5\*8 40

7**. Write a YACC program to check whether the given grammar is valid or not. Consider an input expression and convert it to postfix form.**

***Yacc File***

%{

#include<stdio.h>

int yylex(void);

void yyerror(char \*s);

int k=0;

int i;

char sym[26];

%}

%union

{

char dval;

}

%token <dval> NUMBER

%left '+' '-'

%left '\*' '/'

%nonassoc UMINUS

%type <dval> state

%type <dval> exp

%%

start: state '\n'

| start state '\n'

;

state : exp {

printf("Postfix: ");

for(i = 0;i<k;i++){

printf("%c",sym[i]);

sym[i]=' ';

}

k=0;

printf("\n");

}

;

exp : NUMBER {sym[k]=(char)$$;k++}

| exp '+' exp {sym[k]='+';k++}

| exp '-' exp {sym[k]='-';k++}

| exp '\*' exp {sym[k]='\*';k++}

| exp '/' exp {sym[k]='/';k++}

;

%%

void yyerror(char \*str){

printf("Invalid Character...");

}

int main()

{

yyparse();

return(0);

}

***Lex file***

%{

#include <stdio.h>

#include <math.h>

#include "Q2.tab.h"

%}

%%

[0-9]+ {

yylval.dval=yytext[0];

return NUMBER;

}

[\t];

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

"quit" return 0;

%%

int yywrap(void){

return 1;

}

**INPUT**  **OUTPUT:**

2+3\*4/5-7 234\*5/+7-

2++3 INVALID

**8. Write a program to Implement YACC for Subset of C (for loop) statement.**

***Yacc program***

%{

/Program for YACC Specification/

#include <stdio.h>

int flag=0;

%}

%token FOR OPBR CLBR SEMIC RELOP EQU ID NUM INC DEC

%%

S:FOR OPBR E1 SEMIC E2 SEMIC E3 CLBR {printf("Accepted!");flag=1;}

;

E1:ID EQU ID

| ID EQU NUM

;

E2:ID RELOP ID

| ID RELOP NUM

;

E3:ID INC

| ID DEC

;

%%

main()

{

yyparse();

}

yyerror(constchar \*msg)

{

if(flag==0);

printf("Not Accepted!");

}

***Lex program***

%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}

%%

for {return (FOR);}

"(" {return (OPBR);}

")" {return (CLBR);}

";" {return (SEMIC);}

"=" {return (EQU);}

"<"|">" {return (RELOP);}

"++" {return (INC);}

"--" {return (DEC);}

[a-zA-Z]+ {yylval=yytext[0]; return(ID);}

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

%%

yywrap()

{

return 1;

}

**9. Write a LR parser program in C. Define the data structure for the parsing table in such a way that it can be initialised easily (manually) for a given grammar. Take a simple grammar, eg., expression grammar, compute the parsing table entries by hand using the steps discussed in the class, and initialize the table in your program with these values. Try to parse input expressions scanned by a lexical analyser (which can be easily created using flex). The output of the parser should be SUCCESS or FAILURE depending on the input. In case of FAILURE the parser should indicate the incorrect token in the input.**

CODE:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char stack[30];

int TOP = -1;

void PUSH(char c)

{

TOP++;

stack[TOP] = c;

}

char POP()

{

char c;

if(TOP != -1)

{

c = stack[TOP];

TOP--;

return c;

}

return 'x';

}

void print()

{

int i;

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

for(i=0; i<=TOP; i++)

printf("%c",stack[i]);

}

int main()

{

int i,j,k,l;

char s1[20], s2[20], c1, c2, c3;

printf("\n\n Left Recursive Parser");

printf("\n Enter the expression");

scanf("%s",s1);

l = strlen(s1);

j = 0;

printf("\n $");

for(i = 0; i< l ; i++)

{

if(s1[i] == 'i' && s1[i+1] =='d')

{

s1[i] = ' ';

s1[i+1] = 'E';

print();

printf("id");

PUSH('E');

print();

}

else if(s1[i] == '+' || s1[i] == '-' || s1[i] == '\*' || s1[i] == '/' || s1[i] == 'd')

{

PUSH(s1[i]);

print();

}

}

print();

l = strlen(s2);

while(l != 0)

{

c1 = POP();

if(c1 == 'x')

{

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

break;

}

if( c1 == '+' || c1 == '-' || c1 == '\*' || c1 == '/')

{

c3 = POP();

if(c3 != 'E')

{

printf("error");

exit(0);

}

else

{

PUSH('E');

print();

}

}

c2 = c1;

}

return 0;

}

**INPUT:** **OUTPUT :**

A+B\*C VALID EXPRESSION