A FILE OF COMPILER DESIGN LAB

At

BABA BANDA SINGH BAHADUR ENGINEERING COLLEGE

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE

AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science & Engineering)

SUBMITTED BY:

PRINCE KUMAR (2001308)

SUBMITTED TO:

PROF. ANUPMA SHARMA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BABA BANDA SINGH BAHADUR ENGINEERING COLLEGE,
FATEHGARH SAHIB

Table of contents

PRACTICAL NO. 1

AIM: -
Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language.

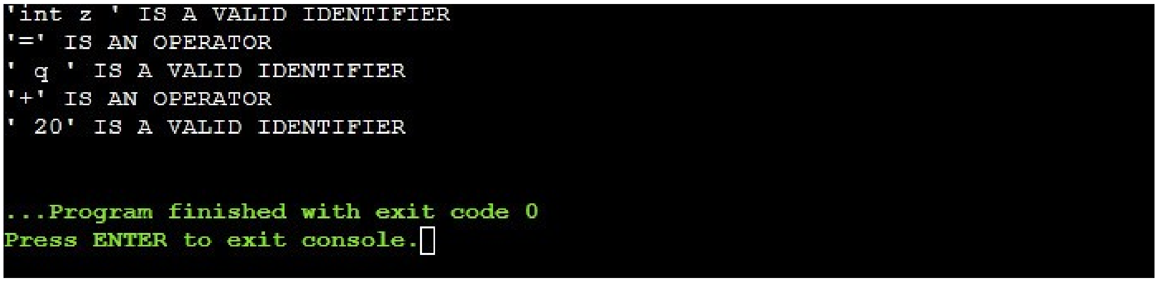
❖ INPUT

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
bool isDelimiter(char ch)
{
        if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
                ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
                ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
                ch == '[' || ch == ']' || ch == '{' || ch == '}')
                return (true);
        return (false);
}

bool isOperator(char ch)
{
        if (ch == '+' || ch == '-' || ch == '*' ||
                ch == '/' || ch == '>' || ch == '<' ||
                ch == '=')
                return (true);
        return (false);
}

bool validIdentifier(char* str)
{
        if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
                str[0] == '3' || str[0] == '4' || str[0] == '5' ||
                str[0] == '6' || str[0] == '7' || str[0] == '8' ||
                str[0] == '9' || isDelimiter(str[0]) == true)
                return (false);
        return (true);
```

```
}
bool isKeyword(char* str)
{
        if (!strcmp(str, "if") || !strcmp(str, "else") ||
                !strcmp(str, "while") || !strcmp(str, "do") ||
                !strcmp(str, "break") ||
                !strcmp(str, "continue") || !strcmp(str, "int")
                || !strcmp(str, "double") || !strcmp(str, "float")
                || !strcmp(str, "return") || !strcmp(str, "char")
                || !strcmp(str, "case") || !strcmp(str, "char")
                || !strcmp(str, "sizeof") || !strcmp(str, "long")
                || !strcmp(str, "short") || !strcmp(str, "typedef")
                || !strcmp(str, "switch") || !strcmp(str, "unsigned")
                || !strcmp(str, "void") || !strcmp(str, "static")
                || !strcmp(str, "struct") || !strcmp(str, "goto"))
                return (true);
        return (false);
}

bool isInteger(char* str)
{
        int i, len = strlen(str);

        if (len == 0)
                return (false);
        for (i = 0; i < len; i++) {
                if (str[i] != '0' && str[i] != '1' && str[i] != '2'
                        && str[i] != '3' && str[i] != '4' && str[i] != '5'
                        && str[i] != '6' && str[i] != '7' && str[i] != '8'
                        && str[i] != '9' || (str[i] == '-' && i > 0))
                        return (false);
        }
        return (true);
}
bool isRealNumber(char* str)
{
        int i, len = strlen(str);
        bool hasDecimal = false;
        if (len == 0)
        return (false); for (i = 0; i < len; i++) {
                if (str[i] != '0' && str[i] != '1' && str[i] != '2'
                        && str[i] != '3' && str[i] != '4' && str[i] != '5'
                        && str[i] != '6' && str[i] != '7' && str[i] != '8'
                        && str[i] != '9' && str[i] != '.' ||
                        (str[i] == '-' && i > 0))
                        return (false);
        if (str[i] == '.')
            hasDecimal = true;
    }
    return (hasDecimal);
}

char* subString(char* str, int left, int right)
{
    int i;
    char* subStr = (char*)malloc(
                sizeof(char) * (right - left + 2));

    for (i = left; i <= right; i++)
        subStr[i - left] = str[i];
    subStr[right - left + 1] = '\0';
    return (subStr);
}
void parse(char* str)
{
    int left = 0, right = 0;
    int len = strlen(str);

    while (right <= len && left <= right) {
        if (isDelimiter(str[right]) == false)
            right++;

        if (isDelimiter(str[right]) == true && left == right) {
            if (isOperator(str[right]) == true)
                printf("'%c' Is an Operator\n", str[right]);

            right++;
            left = right;
        } else if (isDelimiter(str[right]) == true && left != right
                || (right == len && left != right)) {
            char* subStr = subString(str, left, right - 1);
            if (isKeyword(subStr) == true)
                printf("'%s' Is a Keyword\n", subStr);
            else if (isInteger(subStr) == true)
                printf("'%s' Is a Integer\n", subStr);
            else if (isRealNumber(subStr) == true)
                printf("'%s' Is a Real Number\n", subStr);

            else if (validIdentifier(subStr) == true
                    && isDelimiter(str[right - 1]) == false)
```

```
                                        printf("'%s' Is a Valid Identifier\n",
                              subStr);

                    else if (validIdentifier(subStr) == false
                                        && isDelimiter(str[right - 1]) == false)
                              printf("'%s' Is Not a Valid Identifier\n", subStr);
                    left = right;
          }
     }
     return;
}

int main()
{
     char str[100] = "int z = 20 + q;";
  parse(str);
     return (0);
}
```

❖ <u>OUTPUT</u>



```
'int z ' IS A VALID IDENTIFIER
'=' IS AN OPERATOR
' q ' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
' 20' IS A VALID IDENTIFIER


...Program finished with exit code 0
Press ENTER to exit console.
```

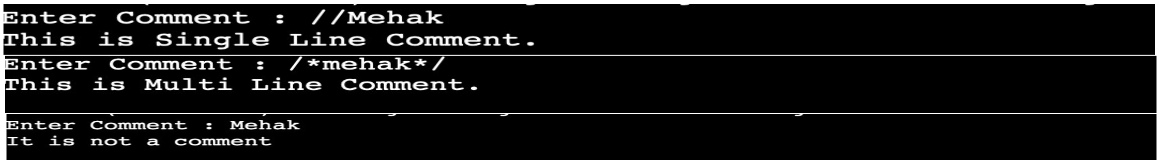
PRACTICAL NO. 2

AIM: -
Write a C program to identify whether a given line is a comment or not.

❖ INPUT

```
#include <stdio.h>
int main() {
   char comment[50];
   int i = 2, a = 0;
   printf("Enter Comment : ");
   gets(comment);
   if(comment[0] == '/'){
      if(comment[1] == '/'){
         printf("This is Single Line Comment.");}
      else if(comment[1] == '*'){
         for(i = 2; i <= 50; i++){
            if(comment[i]=='*' && comment[i + 1] == '/'){
               printf("This is Multi Line Comment.");
               a = 1;
               break;}
            else{
               continue;
            }}
         if(a == 0){
            printf("It is not a comment");
         }}
      else{
         printf("It is not a comment");
      }}
   else{
      printf("It is not a comment");}
    return 0;
}
```

❖ OUTPUT



```
Enter Comment : //Mehak
This is Single Line Comment.
Enter Comment : /*mehak*/
This is Multi Line Comment.
Enter Comment : Mehak
It is not a comment
```

PRACTICAL NO. 3

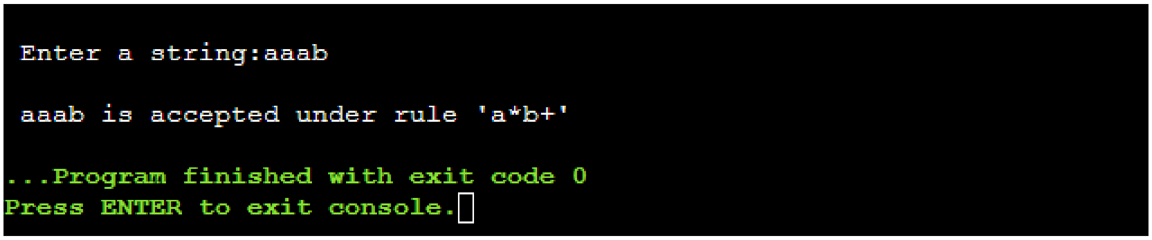
Aim: -

Write a C program to recognize string under ‘a’, ‘a*b+’, ‘abb’.

- **INPUT:**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char s[20],c;
int state=0,i=0;
printf("\n Enter a string:");
gets(s);
while(s[i]!='\0')
{
switch(state)
{
case 0: c=s[i++];
if(c=='a')
state=1;
else if(c=='b')
state=2;
else
state=6;
break;
case 1: c=s[i++];
if(c=='a')
state=3;
else if(c=='b')
state=4;
else
state=6;
break;
case 2: c=s[i++];
if(c=='a')
state=6;
else if(c=='b')
state=2;
else
state=6;
break;
case 3: c=s[i++];
if(c=='a')
state=3;
else if(c=='b')
state=2;
else
state=6;
break;
} case 4: c=s[i++];
if(c=='a')
state=6;
else if(c=='b')
state=5;
else
state=6;
break;
case 5: c=s[i++];
if(c=='a')
state=6;
else if(c=='b')
state=2;
else
state=6;
break;
case 6: printf("\n %s is not recognised.",s);}
}
if(state==1)
printf("\n %s is accepted under rule 'a'",s);
else if((state==2)||(state==4))
printf("\n %s is accepted under rule 'a*b+'",s);
else if(state==5)
printf("\n %s is accepted under rule 'abb'",s);
getch();
}
```

❖ **OUTPUT :**



```
 Enter a string:aaab

 aaab is accepted under rule 'a*b+'

...Program finished with exit code 0
Press ENTER to exit console.
```

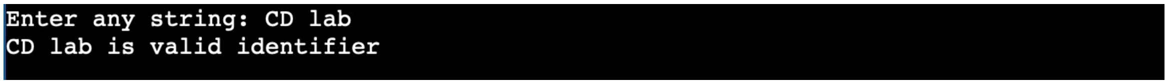
PRACTICAL NO. 4

AIM: -
Write a C program to test whether a given identifier is valid or not.

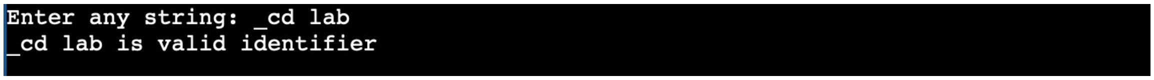
❖ INPUT

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main() {
char string[25];
int count=0,flag;
printf("Enter any string: ");
gets(string);
if( (string[0]>='a'&&string[0]<='z')||(string[0]>='A'&&string[0]<='Z')||(string[0]=='_'))
{
   for(int i=1;i<=strlen(string);i++) {
       if((string[i]>='a'&& string[i]<='z')||(string[i]>='A' && string[i]<='Z')||(string[i]>='0'&&
string[i]<='9')||(string[i]=='-')) {
   count++; }}
   if(count==strlen(string)) {
     flag=0;
   }}
else
{
   flag=1;}
if(flag==1)
   printf("%s is not valid identifier",string);
else
   printf("%s is valid identifier",string);
return 0;}
```

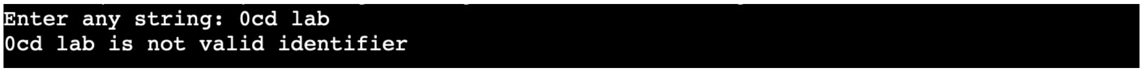
❖ OUTPUT



```
Enter any string: CD lab
CD lab is valid identifier
```



```
Enter any string: _cd lab
_cd lab is valid identifier
```



```
Enter any string: 0cd lab
0cd lab is not valid identifier
```

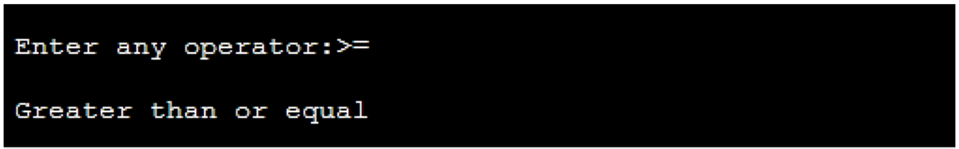
PRACTICAL NO.5

AIM:-
Write a C program to simulate lexical analyzer for validating operators.

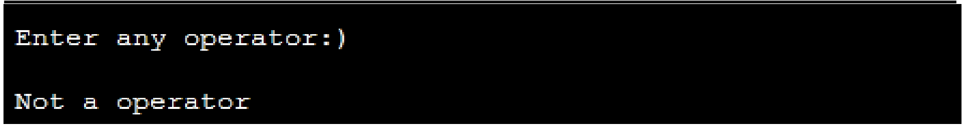
❖ INPUT:-

```
#include<stdio.h>
#include<conio.h>

void main()
{
char s[5];
printf("\n Enter any operator:");
gets(s);
switch(s[0])
{
case'>': if(s[1]=='=')
printf("\n Greater than or equal");
else
printf("\n Greater than");
break;
case'<': if(s[1]=='=')
printf("\n Less than or equal");
else
printf("\nLess than");
break;

case'=': if(s[1]=='=')
printf("\nEqual to");
else
printf("\nAssignment");
break;

case'!': if(s[1]=='=')
printf("\nNot Equal");
else
printf("\n Bit Not");
break;

case'&': if(s[1]=='&')
printf("\nLogical AND");
else
printf("\n Bitwise AND");
break;

case'|': if(s[1]=='|')
printf("\nLogical OR");
else
printf("\nBitwise OR");
break;

case'+': printf("\n Addition");
break;

case'-': printf("\nSubstraction");
break;

case'*': printf("\nMultiplication");
break;

case'/': printf("\nDivision");
break;

case'%': printf("Modulus");
break;

default: printf("\n Not a operator");
}
}
```

❖ OUTPUT:-



```
Enter any operator:>=

Greater than or equal
```



```
Enter any operator:)

Not a operator
```

PRACTICAL.6

AIM: -

Implement the lexical analyzer using JLex, flex or other lexical analyzer generating tools.

- **INPUT: -**

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_TOKEN_LENGTH 100

/* Symbolic constants for token types */
#define TOKEN_IDENTIFIER 1
#define TOKEN_KEYWORD 2
#define TOKEN_NUMBER 3
#define TOKEN_OPERATOR 4

/* Symbolic constants for keywords */
#define KEYWORD_IF 1
#define KEYWORD_ELSE 2
#define KEYWORD_WHILE 3
#define KEYWORD_FOR 4

/* Symbolic constants for operators */
#define OPERATOR_PLUS 1
#define OPERATOR_MINUS 2
#define OPERATOR_MULT 3
#define OPERATOR_DIV 4
%}
/* Regular expressions for identifiers, keywords, and numerical literals */
digit [0-9]
letter [a-zA-Z]
id {letter}({letter}|{digit})*keyword if|else|while|for
number {digit}+(\.{digit}+)?([eE][+-]?{digit}+)?

/* Regular expressions for operators *
/plus \+
minus –
mult \*
div /
%%
/* Rules for identifiers and keywords */
{id} {
 yylval.strval = strdup(yytext);if (!yylval.strval)
```

```
  {
   fprintf(stderr, "Out of memory\n");exit(1);
  }
  if (strcmp(yytext, "if") == 0) {return TOKEN_KEYWORD;
  } else if (strcmp(yytext, "else") == 0) {return TOKEN_KEYWORD;
  } else if (strcmp(yytext, "while") == 0) {return TOKEN_KEYWORD;
  } else if (strcmp(yytext, "for") == 0) {return TOKEN_KEYWORD;
  }
   else {
   return TOKEN_IDENTIFIER;
  }}

/* Rules for numerical literals */
{number} {
  yylval.numval = atof(yytext);
  return TOKEN_NUMBER;
}
/* Rules for operators */
{plus} {
  return OPERATOR_PLUS;
}
{minus} {
  return OPERATOR_MINUS;
}
{mult} {
  return OPERATOR_MULT;
}
{div} {
  return OPERATOR_DIV;
}

/* Rule for whitespace */
[ \t\n] {
  /* ignore whitespace */
}
/* Rule for invalid characters */
{
printf(stderr, "Invalid character: %s\n", yytext);exit(1);
%%
/* Main program */int main(void) {
  int token_type;

  /* Loop through tokens in input */
  while (token_type = yylex()) { switch (token_type) {
     case TOKEN_IDENTIFIER:
      printf("Identifier: %s\n", yylval.strval);free(yylval.strval);
      break;
     case TOKEN_KEYWORD:
      printf("Keyword: %s\n", yylval.strval);free(yylval.strval);
```

```
          break;
         case TOKEN_NUMBER:
          printf("Number: %f\n", yylval.numval);break;
         case OPERATOR_PLUS:
          printf("Operator: +\n");break;
         case OPERATOR_MINUS:
          printf("Operator: -\n");break;
         case OPERATOR_MULT:
          printf("Operator: *\n");break;
         case OPERATOR_DIV:
          printf("Operator: /\n");break;
       }
      }
     return 0; }
```

INPUT.

```
int x = 5;
 if (x > 0) {
  printf("x is positive");
}
else: {
  printf("x is non-positive");}
```

OUTPUT.

```
Keyword: int
Identifier: x
Operator: =
 Number: 5.000000
Keyword: if
Operator: (
Identifier: x
Operator: >
Number: 0.000000
Operator: )
Operator: {
 Keyword: printf
Operator: (
"\"x is positive\""
Operator: )
 Operator: ;
Operator: }
 Keyword: else
 Operator: {
Keyword: printf
Operator: (
"\"x is non-positive\""
Operator: )
Operator: ;
Operator: }
```

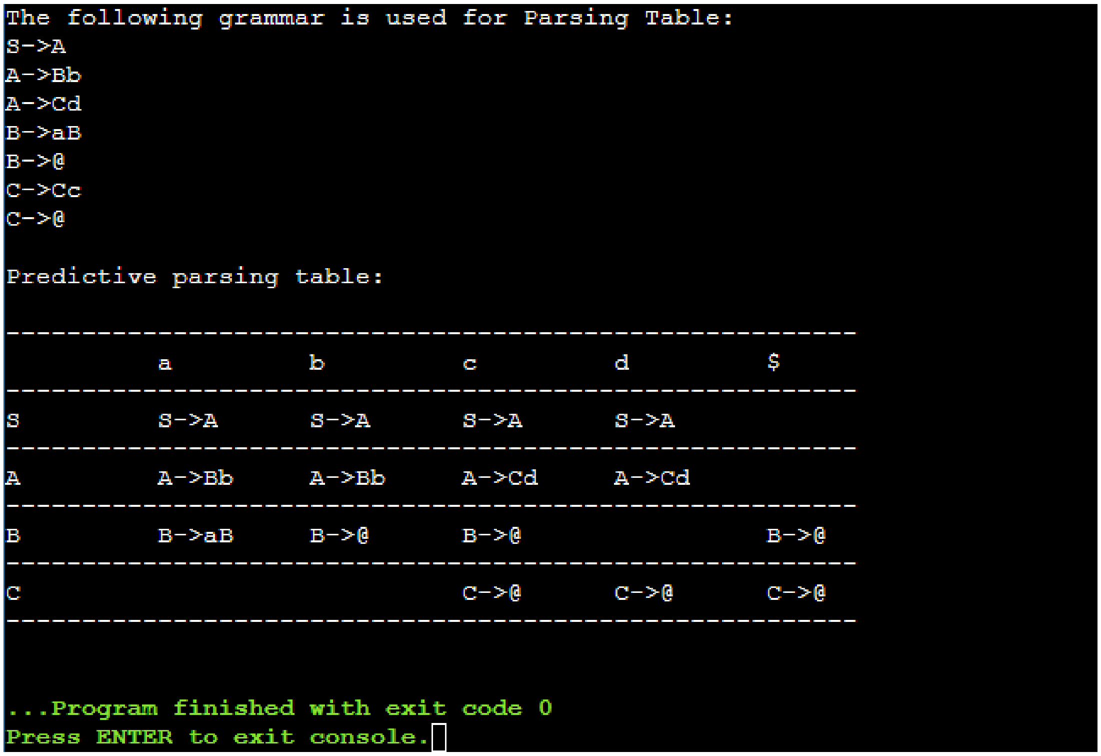
PRACTICAL NO. 7

AIM: -
Write a C program for implementing the functionalities of predictive parser for the mini language.

❖ INPUT: -

```
#include <stdio.h>
#include <string.h>
char prol[7][10] = { "S", "A", "A", "B", "B", "C", "C" };
char pror[7][10] = { "A", "Bb", "Cd", "aB", "@", "Cc", "@" };
char prod[7][10] = { "S->A", "A->Bb", "A->Cd", "B->aB", "B->@", "C->Cc", "C->@" };
char first[7][10] = { "abcd", "ab", "cd", "a@", "@", "c@", "@" };
char follow[7][10] = { "$", "$", "$", "a$", "b$", "c$", "d$" };
char table[5][6][10];
intnumr(char c)
{ switch (c) ,{
case 'S':
return 0;
case 'A':
return 1;
case 'B':
return 2;
case 'C':
return 3;
case 'a':
return 0;
case 'b':
return 1;
case 'c':
return 2;
case 'd':
return 3;
case '$':
return 4; }
return (2); }
int main() {
int i, j, k;
for (i = 0; i < 5; i++)
for (j = 0; j < 6; j++)
strcpy(table[i][j], " ");
printf("The following grammar is used for Parsing Table:\n");
for (i = 0; i < 7; i++)
printf("%s\n", prod[i]);
printf("\nPredictive parsing table:\n");
fflush(stdin);
for (i = 0; i < 7; i++){
    k = strlen(first[i]);
for (j = 0; j < 10; j++)
if (first[i][j] != '@')
strcpy(table[numr(prol[i][0]) + 1][numr(first[i][j]) + 1], prod[i]);}
for (i = 0; i < 7; i++){
if (strlen(pror[i]) == 1){
if (pror[i][0] == '@'){
        k = strlen(follow[i]);
for (j = 0; j < k; j++)
strcpy(table[numr(prol[i][0]) + 1][numr(follow[i][j]) + 1], prod[i]);}}}
strcpy(table[0][0], " ");
strcpy(table[0][1], "a");
strcpy(table[0][2], "b");
strcpy(table[0][3], "c");
strcpy(table[0][4], "d");
strcpy(table[0][5], "$");
strcpy(table[1][0], "S");
strcpy(table[2][0], "A");
strcpy(table[3][0], "B");

strcpy(table[4][0], “C");

printf("\n------------------------------------------------------- \n");

for (i = 0; i < 5; i++)for (j = 0; j < 6; j++){

printf("%-10s", table[i][j]);if (j == 5)

printf("\n------------------------------------------------------- \n");

    }}
```

- ## OUTPUT :-



```
The following grammar is used for Parsing Table:
S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@

Predictive parsing table:

--------------------------------------------------------
          a         b         c         d         $
--------------------------------------------------------
S         S->A      S->A      S->A      S->A
--------------------------------------------------------
A         A->Bb     A->Bb     A->Cd     A->Cd
--------------------------------------------------------
B         B->aB     B->@      B->@                B->@
--------------------------------------------------------
C                             C->@      C->@      C->@
--------------------------------------------------------


...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL NO. 8(a)

Aim: -

Write a C Program for constructing LL (1) Parser.

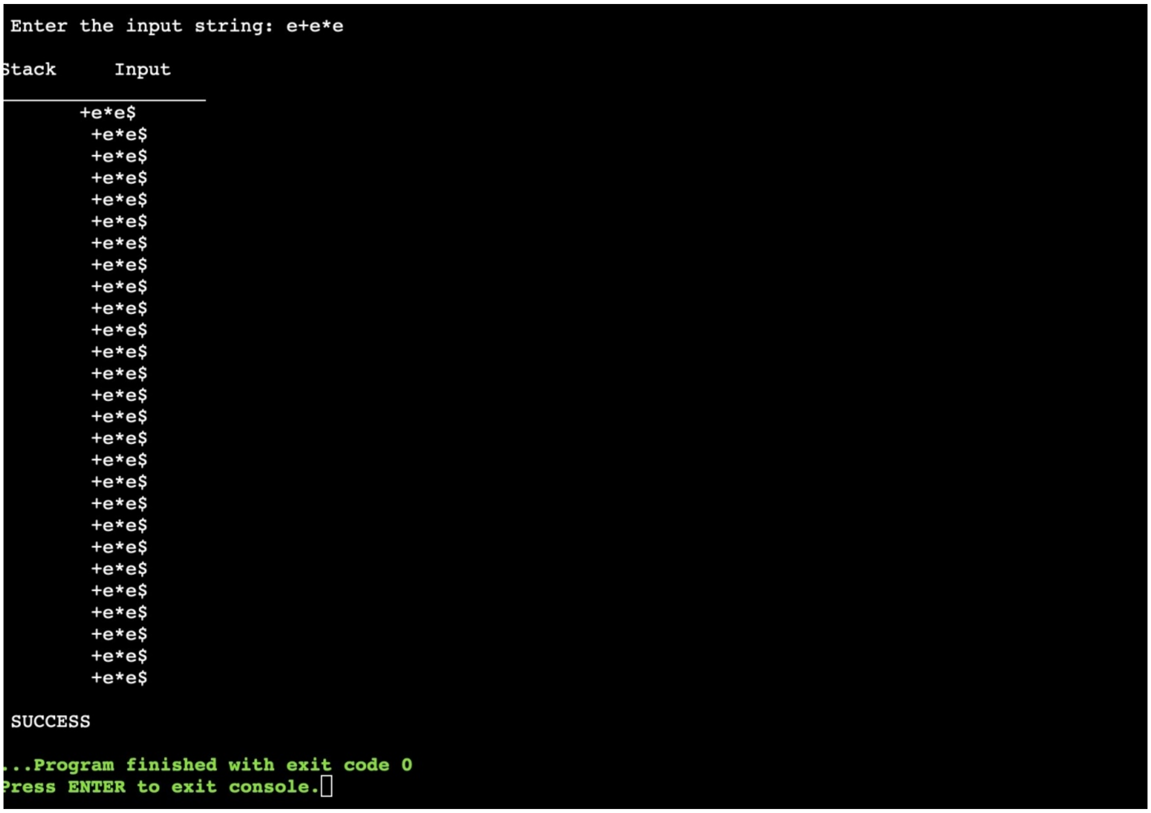
- **INPUT:**

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

char s[20],stack[20];

void main()

{

char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," "," 
","n","*fc"," a ","n","n","i"," "," ","(e)"," "," "};

int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};

int i,j,k,n,str1,str2;

printf("\n Enter the input string: ");

scanf("%s",s);

strcat(s,"$");

n=strlen(s);

stack[0]='$';

stack[1]='e';

i=1;

j=0;

printf("\nStack     Input\n");

printf("__________________\n");

while((stack[i]!='$')&&(s[j]!='$'))

{

if(stack[i]==s[j])
```

```
{
i--;
j++;
}
switch(stack[i])
{
case 'e': str1=0;
break;
case 'b': str1=1;
break;
case 't': str1=2;
break;
case 'c': str1=3;
break;
case 'f': str1=4;
break;
}
switch(s[j])
{
case 'i': str2=0;
break;
case '+': str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
case '$': str2=5;
break;
}
if(m[str1][str2][0]=='\0')
{
printf("\nERROR");
}
else if(m[str1][str2][0]=='n')
i--;
else if(m[str1][str2][0]=='i')
stack[i]='i';
else
{
for(k=size[str1][str2]-1;k>=0;k--)
{
stack[i]=m[str1][str2][k];
i++;
}
i--;
}
for(k=0;k<=i;k++)
printf(" %c",stack[k]);
printf("       ");
```

```
for(k=j;k<=n;k++)

printf("%c",s[k]);

printf(" \n ");

}

printf("\n SUCCESS");

getch();

  }
```

❖ OUTPUT:



```
 Enter the input string: e+e*e

Stack     Input
_________________
      +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$
       +e*e$

 SUCCESS

...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL NO. 8(b)

Aim: -

Write a C Program for constructing recursive descent parsing.

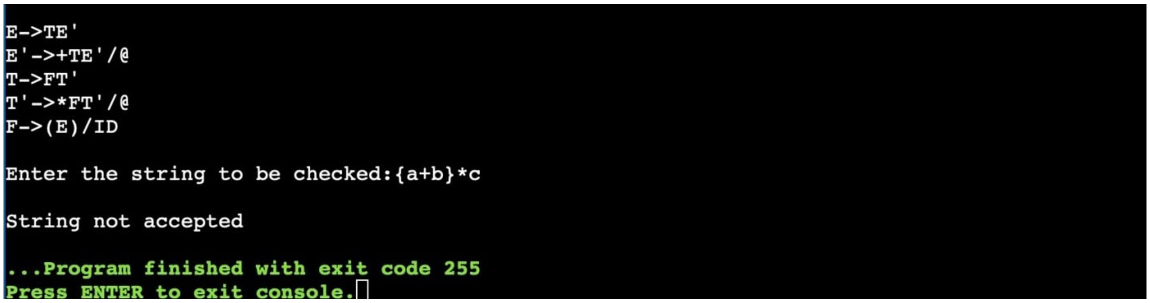
❖ **INPUT:**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char  input[100];
int i,l;
void main()
{
printf("\nRecursive descent parsing for the following grammar\n");
printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/ID\n");
printf("\nEnter the string to be checked:");
gets(input);
if(E())
{
if(input[i+1]=='\0')
printf("\nString is accepted");
else
printf("\nString is not accepted");
}
else
printf("\nString not accepted");
getch();
}
```

```
E(){
 if(T())
{

 f(EP())

return(1);

else

 return(0);

}

else

return(0);

} EP()

{

if(input[i]=='+')

{ i++;

if(T())

{ if(EP())

return(1);

else

return(0);

}

else

return(0);

}

else return(1);

} T()

{ if(F())

{ if(TP())

return(1);
else return(0);}

else return(0);

} TP()

{

if(input[i]=='*')

{ i++;

if(F())

{ if(TP())

return(1);else

return(0);

}
else return(0);}

else return(1);

} F()

{

if(input[i]=='(')

{ i++;

if(E())

{

if(input[i]==')')

{ i++;

return(1);

}

else return(0);

}

else return(0);

}
```

```
else if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')
{
i++;
return(1);
}
else
return(0);
}
```

❖ OUTPUT:



```
E->TE'
E'->+TE'/@
T->FT'
T'->*FT'/@
F->(E)/ID

Enter the string to be checked:{a+b}*c

String not accepted

...Program finished with exit code 255
Press ENTER to exit console.
```

PRACTICAL NO. 9

Aim: -

Write a C program to implement LALR Parsing.

- **INPUT :**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
void push(char *,int *,char);
char stacktop(char *);
void isproduct(char,char);
int ister(char);
int  isnter(char);
int isstate(char);
void error();
void isreduce(char,char);
char pop(char *,int *);
void printt(char *,int *,char [],int);
void rep(char [],int);
struct action
{
char row[6][5];
};
const struct action A[12]={
{"sf","emp","emp","se","emp","emp"},
{"emp","sg","emp","emp","emp","acc"},
{"emp","rc","sh","emp","rc","rc"},
{"emp","re","re","emp","re","re"},
{"sf","emp","emp","se","emp","emp"},
{"emp","rg","rg","emp","rg","rg"},
{"sf","emp","emp","se","emp","emp"},
{"sf","emp","emp","se","emp","emp"},
{"emp","sg","emp","emp","sl","emp"},
{"emp","rb","sh","emp","rb","rb"},
{"emp","rb","rd","emp","rd","rd"},
{"emp","rf","rf","emp","rf","rf"}
};
struct gotol
```

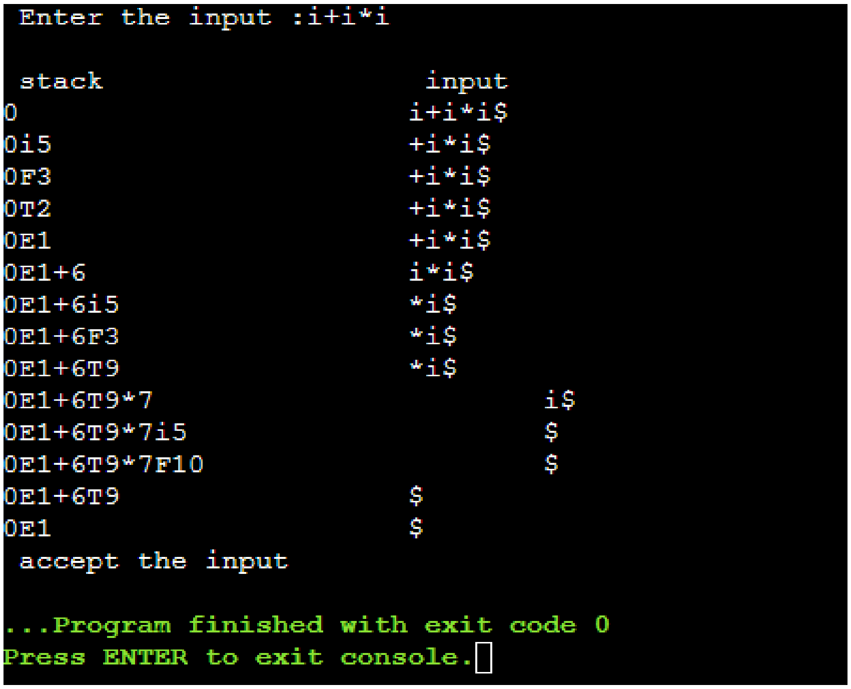
```
{
char r[3][4];
};
const struct gotol G[12]={
{"b","c","d"},
{"emp","emp","emp"},
{"emp","emp","emp"},
{"emp","emp","emp"},
{"i","c","d"},
{"emp","emp","emp"},
{"emp","j","d"},
{"emp","emp","k"},
{"emp","emp","emp"},
{"emp","emp","emp"},
};
char ter[6]={'i','+','*',')','(','$'};
char nter[3]={'E','T','F'};
char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};
char stack[100];
int  top=-1;
char temp[10];
struct grammar
{
char left;
char right[5];
};
const struct grammar rl[6]={
{'E',"e+T"},
{'E',"T"},
{'T',"T*F"},
{'T',"F"},
{'F',"(E)"},
{'F',"i"},
};
void main()
{
char inp[80],x,p,dl[80],y,bl='a';
int i=0,j,k,l,n,m,c,len;
printf(" Enter the input :");
scanf("%s",inp);
len=strlen(inp);
inp[len]='$';
```

```
inp[len+1]='\0';
push(stack,&top,bl);
printf("\n stack \t\t\t input");
printt(stack,&top,inp,i);
do
{
x=inp[i];
p=stacktop(stack);
isproduct(x,p);
if(strcmp(temp,"emp")==0)
error();
if(strcmp(temp,"acc")==0)
break;
else
{
if(temp[0]=='s')
{
push(stack,&top,inp[i]);
push(stack,&top,temp[1]);
i++;
}
else
{
if(temp[0]=='r')
{
j=isstate(temp[1]);
strcpy(temp,rl[j-2].right);
dl[0]=rl[j-2].left;
dl[1]='\0';
n=strlen(temp);
for(k=0;k<2*n;k++)
pop(stack,&top);
for(m=0;dl[m]!='\0';m++)
push(stack,&top,dl[m]);
l=top;
y=stack[l-1];
isreduce(y,dl[0]);
for(m=0;temp[m]!='\0';m++)
push(stack,&top,temp[m]);
}
}
}
printt(stack,&top,inp,i);
}while(inp[i]!='\0');
if(strcmp(temp,"acc")==0)
printf(" \n accept the input ");
else
printf(" \n do not accept the input ");
getch();
}
void push(char *s,int *sp,char item)
{
if(*sp==100)
printf(" stack is full ");
else
{
*sp=*sp+1;
s[*sp]=item;
}
}
char stacktop(char *s)
{
char i;
i=s[top];
return i;
}
void isproduct(char x,char p)
{
int k,l;
k=ister(x);
l=isstate(p);
strcpy(temp,A[l-1].row[k-1]);
}
int ister(char x)
{
int i;
for(i=0;i<6;i++)
if(x==ter[i])
return i+1;
return 0;
}
int isnter(char x)
{
int i;
```

```
for(i=0;i<3;i++)
if(x==nter[i])
return i+1;
return 0;
}
int isstate(char p)
{
int i;
for(i=0;i<12;i++)
if(p==states[i])
return i+1;
return 0;
}
void error()
{
printf(" error in the input ");
exit(0);
}
void isreduce(char x,char p)
{
int k,l;
k=isstate(x);
l=isnter(p);
strcpy(temp,G[k-1].r[l-1]);
}
char pop(char *s,int *sp)
{
char item;
if(*sp==-1)
printf(" stack is empty ");
else
{
item=s[*sp];
*sp=*sp-1;
}
return item;
}
void printt(char *t,int *p,char inp[],int i)
{
int r;
printf("\n");
for(r=0;r<=*p;r++)
```

```
rep(t,r);
printf("\t\t\t");
for(r=i;inp[r]!='\0';r++)
printf("%c",inp[r]);
}
void rep(char t[],int r)
{
char c;
c=t[r];
switch(c)
{
case 'a': printf("0");
break;
case 'b': printf("1");
break;
case 'c': printf("2");
break;
case 'd': printf("3");
break;
case 'e': printf("4");
break;
case 'f': printf("5");
break;
case 'g': printf("6");
break;
case 'h': printf("7");
break;
case 'm': printf("8");
break;
case 'j': printf("9");
break;
case 'k': printf("10");
break;
case 'l': printf("11");
break;
default :printf("%c",t[r]);
break;
}
}
```

❖ **OUTPUT :**



```
 Enter the input :i+i*i

 stack                   input
0                       i+i*i$
0i5                     +i*i$
0F3                     +i*i$
0T2                     +i*i$
0E1                     +i*i$
0E1+6                   i*i$
0E1+6i5                 *i$
0E1+6F3                 *i$
0E1+6T9                 *i$
0E1+6T9*7                       i$
0E1+6T9*7i5                     $
0E1+6T9*7F10                    $
0E1+6T9                 $
0E1                     $
 accept the input

...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL NO. 10(a)

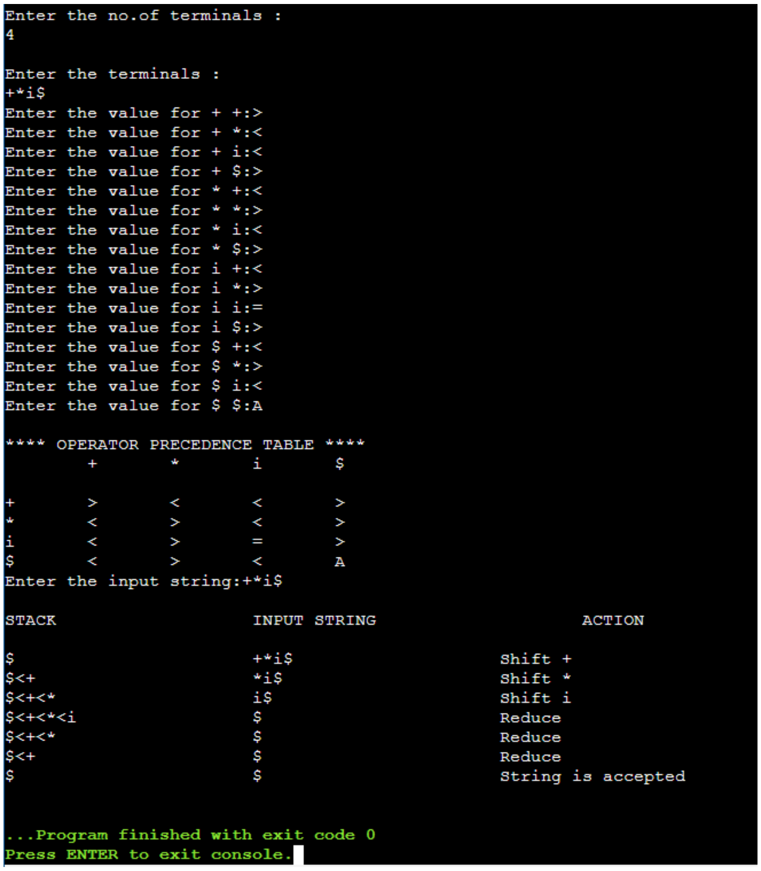
Aim: -

Write a C program to implement operator precedence parsing.

- **INPUT :**

```
#include<stdio.h>
#include<string.h>
#include<conio.h>

void main() {
  char stack[20], ip[20], opt[10][10][1], ter[10];
  int i, j, k, n, top = 0, col, row;
  for (i = 0; i < 10; i++)
  {
    stack[i] = NULL;
    ip[i] = NULL;
    for (j = 0; j < 10; j++)
    {
      opt[i][j][1] = NULL;
    }
  }
  printf("Enter the no.of terminals :\n");
  scanf("%d", & n);
  printf("\nEnter the terminals :\n");
  scanf("%s", & ter);
// printf("\nEnter the table values :\n");
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      printf("Enter the value for %c %c:", ter[i], ter[j]);
      scanf("%s", opt[i][j]);
    }
  }
  printf("\n**** OPERATOR PRECEDENCE TABLE ****\n");
  for (i = 0; i < n; i++) {
    printf("\t%c", ter[i]);
  }
  printf("\n");
  for (i = 0; i < n; i++) {
    printf("\n%c",   ter[i]);
    for (j = 0; j < n; j++) {
      printf("\t%c", opt[i][j][0]);
    }
  }
  stack[top] = '$';
  printf("\nEnter the input string:");
  scanf("%s", ip);
  i = 0;
  printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
  printf("\n%s\t\t\t%s\t\t\t", stack, ip);
  while (i <= strlen(ip)) {
    for (k = 0; k < n; k++) {
    if (stack[top] == ter[k])
        col = k;
      if (ip[i] == ter[k])
        row = k;
    }
    if ((stack[top] == '$') && (ip[i] == '$')) {
      printf("String is accepted\n");
      break;
    } else if ((opt[col][row][0] == '<') || (opt[col][row][0] == '=')) {
      stack[++top] = opt[col][row][0];
      stack[++top] = ip[i];
      printf("Shift %c", ip[i]);
      i++;
    } else {
      if (opt[col][row][0] == '>') {
        while (stack[top] != '<') {
          --top;
        }
        top = top - 1;
        printf("Reduce");
      } else {
        printf("\nString is not accepted");
        break;
      }
    }
    printf("\n");
    for (k = 0; k <= top; k++) {
      printf("%c", stack[k]);
    }
    printf("\t\t\t");
    for (k = i; k < strlen(ip); k++) {
      printf("%c", ip[k]);
    }
    printf("\t\t\t");
  }
getch();
}
```

❖ OUTPUT :



```
Enter the no.of terminals :
4

Enter the terminals :
+*i$
Enter the value for + +:>
Enter the value for + *:<
Enter the value for + i:<
Enter the value for + $:>
Enter the value for * +:<
Enter the value for * *:>
Enter the value for * i:<
Enter the value for * $:>
Enter the value for i +:<
Enter the value for i *:>
Enter the value for i i:=
Enter the value for i $:>
Enter the value for $ +:<
Enter the value for $ *:>
Enter the value for $ i:<
Enter the value for $ $:A

**** OPERATOR PRECEDENCE TABLE ****
        +       *       i       $

+       >       <       <       >
*       <       >       <       >
i       <       >       =       >
$       <       >       <       A
Enter the input string:+*i$

STACK                   INPUT STRING                    ACTION

$                       +*i$                    Shift +
$<+                     *i$                     Shift *
$<+<*                   i$                      Shift i
$<+<*<i                 $                       Reduce
$<+<*                   $                       Reduce
$<+                     $                       Reduce
$                       $                       String is accepted


...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL NO. 10(b)

Aim: -

b) Write a C program to implement Program semantic rules to calculate the expression that takes an expression with digits, + and * and computes the value.

- **INDEX:**

```
    %
Aparser.l>
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+ {yylval.dval=atof(yytext);
return DIGIT;
}
\n. return yytext[0];
%%
<parser.y>
%{
/*This YACC specification file generates the LALR parser for the program
considered in experiment 4.*/
#include<stdio.h>
%} %union
{
double dval;
}
%token <dval> DIGIT
%type <dval> expr
%type <dval> term
%type <dval> factor
%%
line: expr '\n' {
printf("%g\n",$1); };

expr: expr '+' term {$$=$1 + $3;}
| term
term: term '*' factor {$$=$1 * $3;}
| factor
factor: '('expr')' {$$=$2 ;}
| DIGIT
%%
int main()
{
yyparse();
}
yyerror(char *s)
{
printf("%s",s);
}
```

INPUT & OUTPUT:

```
$lex parser.l
$yacc -d parser.y
$cc lex.yy.c y.tab.c -11 -lm
$./a.out
2+3
5.0000
```

PRACTICAL NO. 11

Aim: -

Convert the BNF rules into YACC form and write code to generate abstract syntax tree for the mini language specified in Note 1.

- ❖ **INDEX:**

LEX PART

```
%{
#include"y.tab.h"
#include<stdio.h>
#include<string.h>
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```

YACC PART

```
%{
#include<string.h>
#include<stdio.h>
struct quad
```

```
{char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];
struct stack
{int items[100];
int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}
%union
{char var[10];}
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-"+'
%left '*"/'
%%
PROGRAM : MAIN BLOCK, ;
BLOCK: '{' CODE '}';
CODE: BLOCK | STATEMENT CODE | STATEMENT;
STATEMENT: DESCT ';'
| ASSIGNMENT ';'
| CONDST
| WHILEST;
DESCT: TYPE VARLIST;
VARLIST: VAR ',' VARLIST | VAR;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);};
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
| EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$,$2);}
| VAR
| NUM;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
```

```
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);}
| IFST ELSEST;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;}
BLOCK { strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;}
| VAR
| NUM;
WHILEST: WHILELOOP{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);};
WHILELOOP: WHILE'('CONDITION ')'{
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
%%
extern FILE *yyin;
int main(int argc,char *argv[]){
FILE *fp;
int i;
if(argc>1){
fp=fopen(argv[1],"r");
if(!fp){
printf("\n File not found");
exit(0);}
yyin=fp;
}
yyparse();
printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator \tArg1 \tArg2
\tResult""\n\t\t--------------------");
for(i=0;i<Index;i++){
printf("\n\t\t %d\t %s\t %s\t
%s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result); }printf("\n\t\t
-----------------------");
printf("\n\n"); return 0; }
void push(int data)
{ stk.top++;
if(stk.top==100)
{printf("\n Stack overflow\n");
exit(0);}
stk.items[stk.top]=data;}
int pop(){
int data;
if(stk.top==-1){
printf("\n Stack underflow\n");
exit(0);}
data=stk.items[stk.top--];
return data;}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10]){
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);
strcpy(QUAD[Index].arg2,arg2);
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result);}
yyerror(){
printf("\n Error on line no:%d",LineNo);}
```

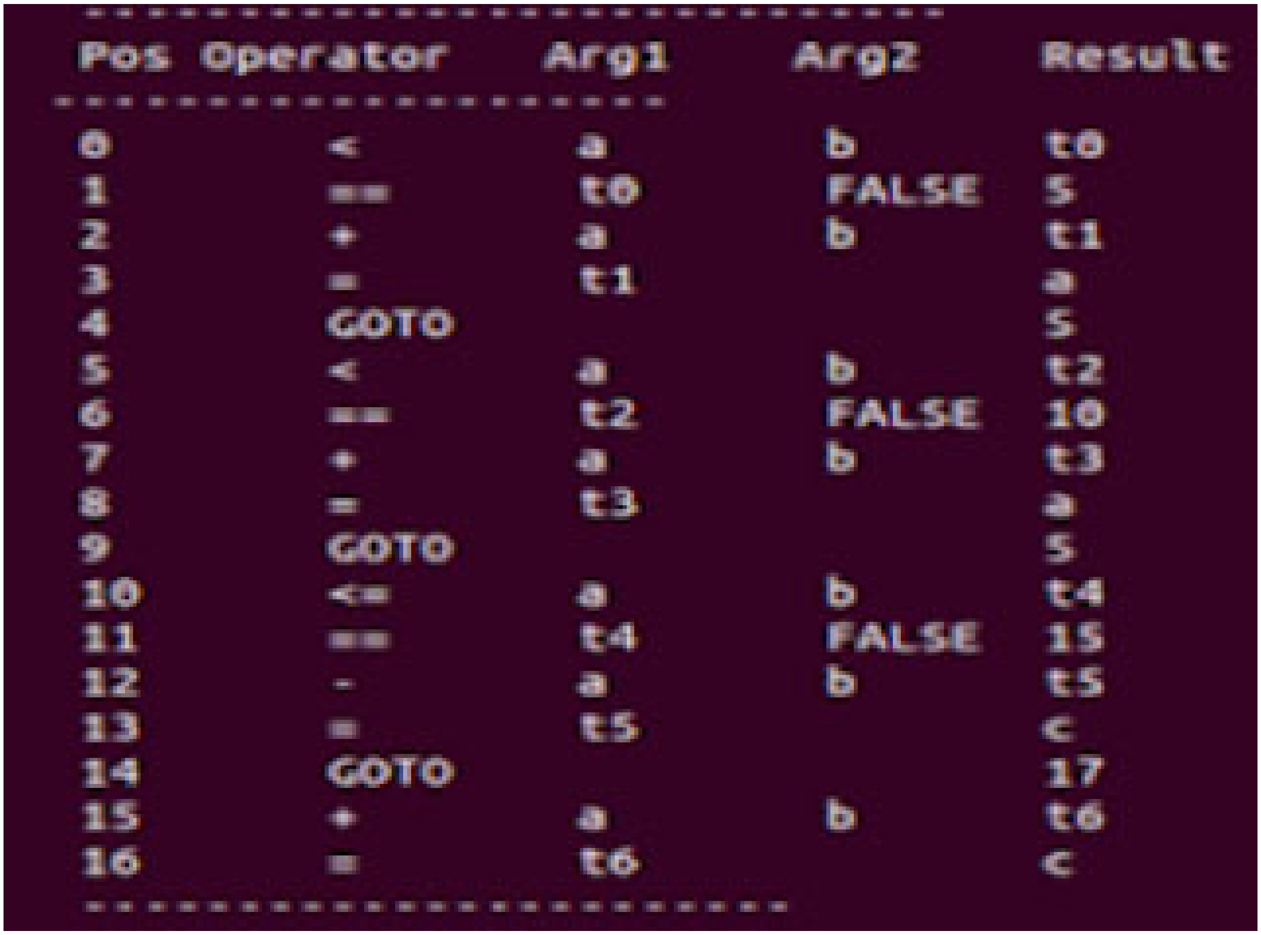
INPUT:

```
main()
{
int a,b,c;
if(a<b){
a=a+b;}
while(a<b){
a=a+b;}
if(a<=b){
c=a-b;}
else{
c=a+b;}}
```

COMPILATION

```
lex filename.l
yacc -d filename.y
cc lex.yy.c y.tab.c -w
./a.out
```

OUTPUT



```
 ---------------------------------
 Pos Operator    Arg1     Arg2    Result
-----------------------
 0        <        a       b      t0
 1        ==       t0      FALSE  5
 2        +        a       b      t1
 3        =        t1             a
 4        GOTO                    5
 5        <        a       b      t2
 6        ==       t2      FALSE  10
 7        +        a       b      t3
 8        =        t3             a
 9        GOTO                    5
 10       <=       a       b      t4
 11       ==       t4      FALSE  15
 12       -        a       b      t5
 13       =        t5             c
 14       GOTO                    17
 15       +        a       b      t6
 16       =        t6             c
 --------------------------
```

PRACTICAL NO. 12

Aim: -

Write a C program to generate machine code from abstract syntax tree generated by the parser. The instruction set specified in Note 2 may be considered as the target code.

- **INDEX:**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int label[20];
int no=0;
int main()
{ FILE *fp1,*fp2;
char fname[10],op[10],ch;
char operand1[8],operand2[8],result[8];
int i=0,j=0;
printf("\n Enter filename of the intermediate code");
scanf("%s",&fname);
fp1=fopen(fname,"r");
fp2=fopen("target.txt","w");
if(fp1==NULL || fp2==NULL)
{ printf("\n Error opening the file");
exit(0); }
while(!feof(fp1)) {
fprintf (fp2, "\n");
fscanf (fp1, "%s", op);
i++;
if (check_label (i))
  fprintf (fp2, "\nlabel#%d", i);
if (strcmp (op, "print") == 0)
  {
    fscanf (fp1, "%s", result);
    fprintf (fp2, "\n\t OUT %s", result);
  }
if (strcmp (op, "goto") == 0)
  {
    fscanf (fp1, "%s %s", operand1, operand2);
    fprintf (fp2, "\n\t JMP %s,label#%s", operand1, operand2);
    label[no++] = atoi (operand2);
  }
if (strcmp (op, "[]=") == 0)
  {
    fscanf (fp1, "%s %s %s", operand1, operand2, result);
    fprintf (fp2, "\n\t STORE %s[%s],%s", operand1, operand2, result);
  }
```

```
if (strcmp (op, "uminus") == 0)
  {
    fscanf (fp1, "%s %s", operand1, result);
    fprintf (fp2, "\n\t LOAD -%s,R1", operand1);
    fprintf (fp2, "\n\t STORE R1,%s", result);
  }
switch (op[0])  {
  case '*':
    fscanf (fp1, "%s %s %s", operand1, operand2, result);
    fprintf (fp2, "\n \t LOAD", operand1);
    fprintf (fp2, "\n \t LOAD %s,R1", operand2);
    fprintf (fp2, "\n \t MUL R1,R0");
    fprintf (fp2, "\n \t STORE R0,%s", result);
    break;
  case '+':
    fscanf (fp1, "%s %s %s", operand1, operand2, result);
    fprintf (fp2, "\n \t LOAD %s,R0", operand1);
    fprintf (fp2, "\n \t LOAD %s,R1", operand2);
    fprintf (fp2, "\n \t ADD R1,R0");
    fprintf (fp2, "\n \t STORE R0,%s", result);
    break;
case '-':
    fscanf (fp1, "%s %s %s", operand1, operand2, result);
    fprintf (fp2, "\n \t LOAD %s,R0", operand1);
    fprintf (fp2, "\n \t46
LOAD %s,R1", operand2);
    fprintf (fp2, "\n \t SUB R1,R0");
    fprintf (fp2, "\n \t STORE R0,%s", result);
    break;
  case '/':
    fscanf (fp1, "%s %s s", operand1, operand2, result);
    fprintf (fp2, "\n \t LOAD %s,R0", operand1);
    fprintf (fp2, "\n \t LOAD %s,R1", operand2);
    fprintf (fp2, "\n \t DIV R1,R0");
    fprintf (fp2, "\n \t STORE R0,%s", result);
    break;
  case '%':
    fscanf (fp1, "%s %s %s", operand1, operand2, result);
    fprintf (fp2, "\n \t LOAD %s,R0", operand1);
    fprintf (fp2, "\n \t LOAD %s,R1", operand2);
    fprintf (fp2, "\n \t DIV R1,R0");
    fprintf (fp2, "\n \t STORE R0,%s", result);
    break;
  case '=':
    fscanf (fp1, "%s %s", operand1, result);
    fprintf (fp2, "\n\t STORE %s %s", operand1, result);
    break;
  case '>':
    j++;
    fscanf (fp1, "%s %s %s", operand1, operand2, result);
    fprintf (fp2, "\n \t LOAD %s,R0", operand1);
    fprintf (fp2, "\n\t JGT %s,label#%s", operand2, result);
    label[no++] = atoi (result);
    break;
  case '<':
    fscanf (fp1, "%s %s %s", operand1, operand2, result);
    fprintf (fp2, "\n \t LOAD %s,R0", operand1);
    fprintf (fp2, "\n\t JLT %s,label#%d", operand2, result);
    label[no++] = atoi (result);
    break;
  }}
fclose (fp2);
fclose (fp1);
fp2 = fopen ("target.txt", "r");
if (fp2 == NULL)
  {
    printf ("Error opening the file\n");
    exit (0);  }
do  {
    ch = fgetc (fp2);
    printf ("%c", ch);
  }
while (ch != EOF);
fclose (fp1);
return 0;
47
}
int
check_label (int k)
{
  int i;
  for (i = 0; i < no; i++)
    {
      if (k == label[i])
          return 1;
    }
  return 0;
}
```

INPUT & OUTPUT: $vi int.txt

```
=t1 2
[]=a 0 1
 []=a 1 2
 []=a 2 3
*t1 6 t2
+a[2] t2 t3 –
a[2] t1 t2
/t3 t2 t2
uminus t2 t2
print t2
goto t2 t3
=t3 99
uminus 25 t2
 *t2 t3 t3
uminus t1 t1
+t1 t3 t4
print t4 48
```

Output: Enter filename of the intermediate code: int.txt

```
 STORE t1,2
 STORE a[0],1
STORE a[1],2
STORE a[2],3

 LOAD t1,R0
 LOAD 6,R1
 ADD R1,R0
 STORE R0,t3

 LOAD a[2],R0
 LOAD t2,R1
 ADD R1,R0
 STORE R0,t3

 LOAD a[t2],R0
 LOAD t1,R1
 SUB R1,R0
 STORE R0,t2
LOAD t3,R0

 LOAD t2,R1
 DIV R1,R0
STORE R0,t2
```

```
LOAD t2,R1
STORE R1,t2
 LOAD t2,R0
JGT 5,label#11
 Label#11: OUT t2
 JMP t2,label#13
 Label#13: STORE t3,99
LOAD 25,R1
 STORE R1,t2

 LOAD t2,R0
 LOAD t3,R1
MUL R1,R0
STORE R0,t3

LOAD t1,R1
STORE R1,t1

 LOAD t1,R0
 LOAD t3,R1
ADD R1,R0
STORE R0,t4
OUT t4
```