PROGRAM: 2. Write a C program to design a lexical analyzer for given language, which should ignore the redundant spaces, tabs, new lines, find the tokens and also count the number of lines using C program.

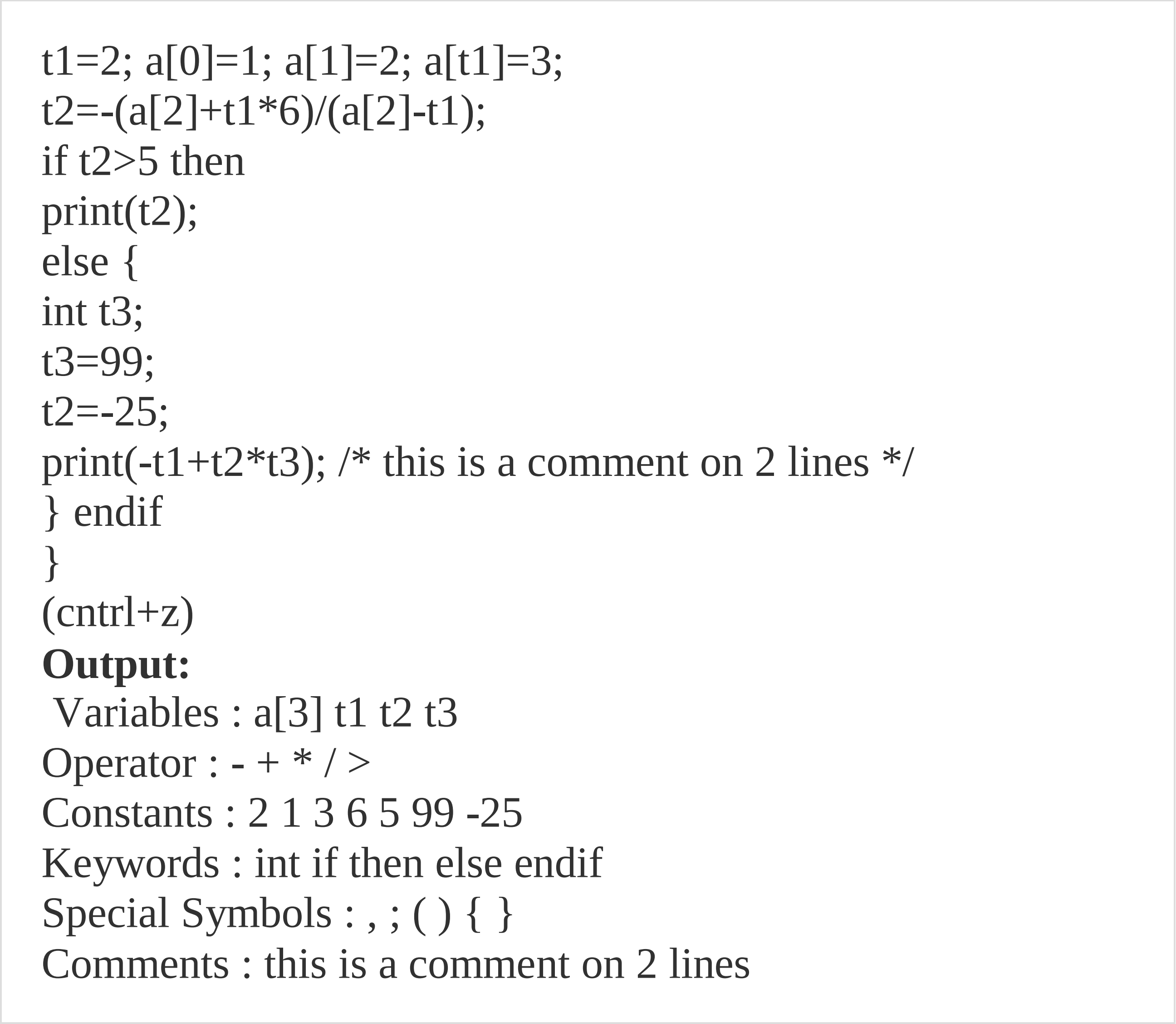
```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",s
tr)==0||
strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)
==0||strcmp("double",str)==0||

strcmp("static",str)==0||strcmp("switch",str)==0||strcmp("case
",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str);
}

main()
{
FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
printf("\nEnter the c program");/*gets(st1);*/
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
```

```
f3=fopen("specialchar","w");
while((c=getc(f1))!=EOF){
if(isdigit(c))
{
tokenvalue=c-'0';
c=getc(f1);
while(isdigit(c)){
tokenvalue*=10+c-'0';
c=getc(f1);
}
num[i++]=tokenvalue;
ungetc(c,f1);
}
else if(isalpha(c))
{
putc(c,f2);
c=getc(f1);
while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
{
putc(c,f2);
c=getc(f1);
}
putc(' ',f2);
ungetc(c,f1);
}
else if(c==' '||c=='\t')
printf(" ");
else
if(c=='\n')
lineno++;
else
putc(c,f3);
}
```

```
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe no's in the program are");
for(j=0;j<i;j++)
printf("%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiersare:");
while((c=getc(f2))!=EOF){
if(c!=' ')
str[k++]=c;
else
{
str[k]='\0';
keyword(str);
k=0;
}
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\nSpecial characters are");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d",lineno);
}
```

Input:

```
Enter Program $ for termination:
{
int a[3],t1,t2;
```



```
t1=2; a[0]=1; a[1]=2; a[t1]=3;
t2=-(a[2]+t1*6)/(a[2]-t1);
if t2>5 then
print(t2);
else {
int t3;
t3=99;
t2=-25;
print(-t1+t2*t3); /* this is a comment on 2 lines */
} endif
}
(cntrl+z)
```

Output:

```
 Variables : a[3] t1 t2 t3
Operator : - + * / >
Constants : 2 1 3 6 5 99 -25
Keywords : int if then else endif
Special Symbols : , ; ( ) { }
Comments : this is a comment on 2 lines
```

4. Program

Write a C program to recognize strings under 'a*|abb'

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

 main()
```

```
{ char s[20],c;
 int state=0,i=0;
 //clrscr();
 printf("\n Enter a string:");
gets(s);
 while(s[i]!='\0')
 {
switch(state)
 {
    case 0: c=s[i++];
 if(c=='a')
    state=1;
 else
   state=4;
 break;
 case 1: c=s[i++];
 if(c=='a')
   state=1;
 else if(c=='b')
    state=2;
 else
 }
 state=4;
 break;
 case 2: c=s[i++];
 if(c=='b')
 state=3;
 else
 state=4;
 break;
 case 3: if((c=s[i++])!='\0')
  state=4;
 else if(c=='b')
 state=2;
 //else state=6
 break;
 case 4: c=s[i++];
 printf("\n %s is not recognised.",s);
 exit(0);
 }
 }
 if(state==1)
 printf("\n %s is accepted under rule 'a'",s);
```

```
 else if(state==3)
 printf("\n %s is accepted under rule 'abb'",s);
 else
 printf("\n %s is not accepted",s);
 return 0;
 }
```

```
Input : Enter a String: aaaabb
Output: aaaabb is accepted under rule 'a*|abb'
Enter a string: cdgs cdgs is not recognized
```

5. Write a C program to construct a recursive descent parser for an expression

```
#include
 #include
 char input[10];
int i=0,error=0;
void E();
 void T();
voidEprime();
voidTprime();
void F();
void main()
{
clrscr();
```

```
printf("Enter an arithmetic expression :\n");

gets(input);

 E();

if(strlen(input)==i&&error==0)

printf("\nAccepted..!!!");

 else

printf("\nRejected..!!!");

 getch();

 }

 void E()

 {

 T();

 Eprime();


 }

 voidEprime()

{

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

{

i++;

T();

 Eprime();

 }

 }

 void T()

 {

 F();
 Tprime();


}
 voidTprime()
{
 if(input[i]=='*')
{
 i++;
 F();
 Tprime();
}
 }
 void F()
 {
 if(input[i]=='(')
{
 i++;
 E();
 if(input[i]==')') i++;
}
 else if(isalpha(input[i]))
{
 i++;
 while(isalnum(input[i])||input[i]=='_')
i++;
 }
 else error=1;
```

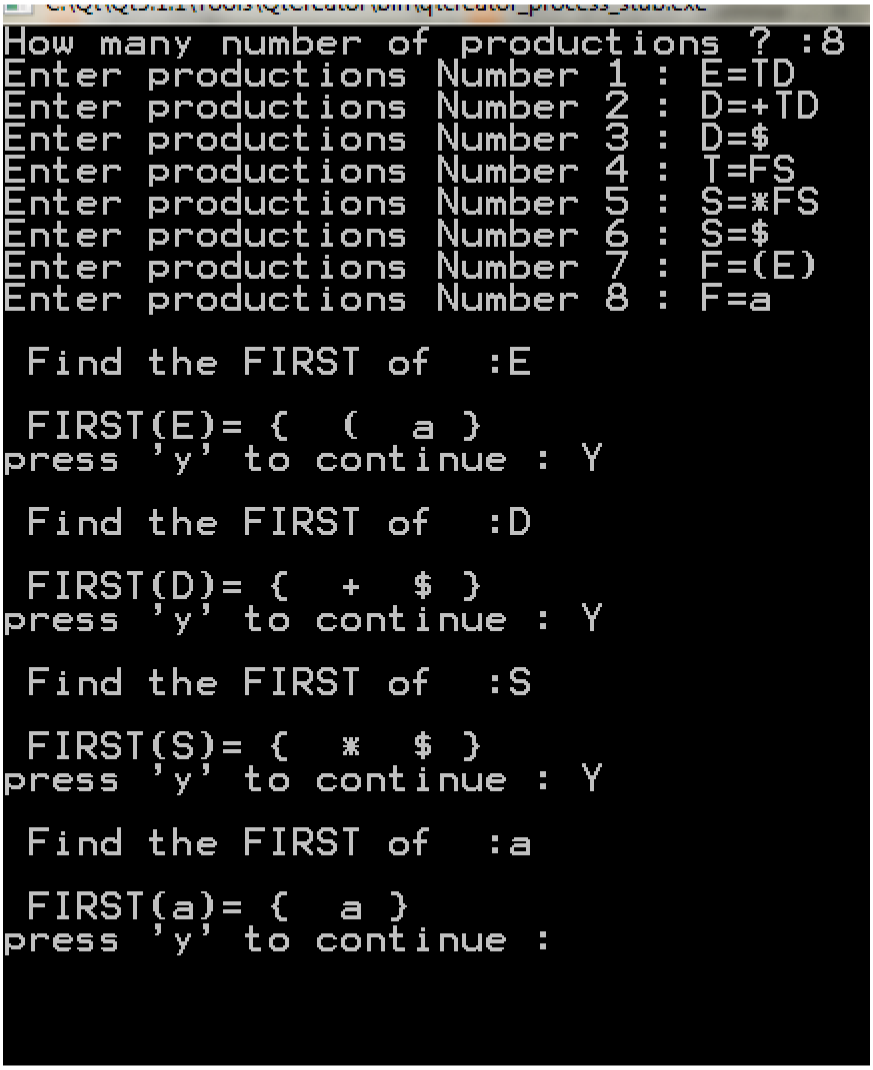
```
}
```

```
OUTPUT 1) Enter an arithmetic expression :

sum+month*interest Accepted..!!!

2) Enter an arithmetic expression :

sum+avg*+interest Rejected..!!
```

6.) Write a C program to simulate FIRST of a given Context Free Grammar.

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[20][20];
main()
{
    int i;
    char choice;
    char c;
    char result[30];
    printf("How many number of productions ? :");
    scanf(" %d",&numOfProductions);
    for(i=0;i<numOfProductions;i++)//read production string eg: E=E+T
    {
        printf("Enter productions Number %d : ",i+1);
        scanf(" %s",productionSet[i]);
    }
    do
    {
        printf("\n Find the FIRST of  :");
        scanf(" %c",&c);
        FIRST(result,c); //Compute FIRST; Get Answer in 'result' array
        printf("\n FIRST(%c)= { ",c);
        for(i=0;result[i]!='\0';i++)
        printf(" %c ",result[i]);       //Display result
        printf("}\n");
         printf("press 'y' to continue : ");
        scanf(" %c",&choice);
    }
    while(choice=='y'||choice =='Y');
}
/*
 *Function FIRST:
 *Compute the elements in FIRST(c) and write them
 *in Result Array.
 */
void FIRST(char* Result,char c)
{
    int i,j,k;
    char subResult[20];
    int foundEpsilon;
    subResult[0]='\0';
    Result[0]='\0';
    //If X is terminal, FIRST(X) = {X}.
    if(!(isupper(c)))
    {
        addToResultSet(Result,c);
               return ;
    }
    //If X is non terminal
    //Read each production
    for(i=0;i<numOfProductions;i++)
    {
//Find production with X as LHS
        if(productionSet[i][0]==c)
        {
//If X → ε is a production, then add ε to FIRST(X).
 if(productionSet[i][2]=='$') addToResultSet(Result,'$');
            //If X is a non-terminal, and X → Y1 Y2 … Yk
            //is a production, then add a to FIRST(X)
            //if for some i, a is in FIRST(Yi),
            //and ε is in all of FIRST(Y1), …, FIRST(Yi-1).
      else
            {
                j=2;
                while(productionSet[i][j]!='\0')
                {
                foundEpsilon=0;
                FIRST(subResult,productionSet[i][j]);
                for(k=0;subResult[k]!='\0';k++)
                    addToResultSet(Result,subResult[k]);
                 for(k=0;subResult[k]!='\0';k++)
                     if(subResult[k]=='$')
                     {
                         foundEpsilon=1;
                         break;
                     }
                 //No ε found, no need to check next element
                 if(!foundEpsilon)
                     break;
                 j++;
                }
            }
    }
}
    return ;
}
/* addToResultSet adds the computed
 *element to result set.
 *This code avoids multiple inclusion of elements
  */
void addToResultSet(char Result[],char val)
{
    int k;
    for(k=0 ;Result[k]!='\0';k++)
        if(Result[k]==val)
            return;
    Result[k]=val;
    Result[k+1]='\0';
}
```

Output


C:\Qt\Qt5.1.1\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
How many number of productions ? :8
Enter productions Number 1 : E=TD
Enter productions Number 2 : D=+TD
Enter productions Number 3 : D=$
Enter productions Number 4 : T=FS
Enter productions Number 5 : S=*FS
Enter productions Number 6 : S=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=a

 Find the FIRST of  :E

 FIRST(E)= {  (  a }
press 'y' to continue : Y

 Find the FIRST of  :D

 FIRST(D)= {  +  $ }
press 'y' to continue : Y

 Find the FIRST of  :S

 FIRST(S)= {  *  $ }
press 'y' to continue : Y

 Find the FIRST of  :a

 FIRST(a)= {  a }
press 'y' to continue :
```

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

#include<string.h>

int n,m=0,p,i=0,j=0;

char a[10][10],followResult[10];

void follow(char c);

void first(char c);

void addToResult(char);
```

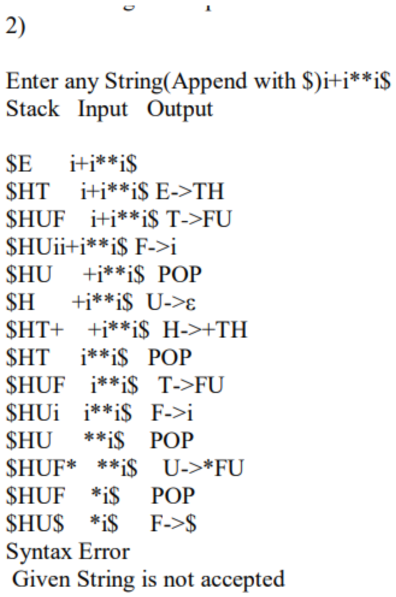
```
int main()
{
 int i;
 int choice;
 char c,ch;
 printf("Enter the no.of productions: ");
scanf("%d", &n);
 printf(" Enter %d productions\nProduction with multiple terms should be give as separate productions \n", n);
 for(i=0;i<n;i++)
  scanf("%s%c",a[i],&ch);
    // gets(a[i]);
 do
 {
  m=0;
  printf("Find FOLLOW of -->");
  scanf(" %c",&c);
  follow(c);
  printf("FOLLOW(%c) = { ",c);
  for(i=0;i<m;i++)
   printf("%c ",followResult[i]);
  printf(" }\n");
  printf("Do you want to continue(Press 1 to
continue....)?");
 scanf("%d%c",&choice,&ch);
 }
 while(choice==1);
}
void follow(char c)
{
    if(a[0][0]==c)addToResult('$');
 for(i=0;i<n;i++)
 {
  for(j=2;j<strlen(a[i]);j++)
   {
   if(a[i][j]==c)
    {
    if(a[i][j+1]!='\0')first(a[i][j+1]);
    if(a[i][j+1]=='\0'&&c!=a[i][0])
     follow(a[i][0]);
   }
  }
 }
}
void first(char c)
{
      int k;
                if(!(isupper(c)))
                    //f[m++]=c;
                    addToResult(c);
                for(k=0;k<n;k++)
                {
                if(a[k][0]==c)
                {
                if(a[k][2]=='$') follow(a[i][0]);
                else if(islower(a[k][2]))
                    //f[m++]=a[k][2];
                    addToResult(a[k][2]);
                else first(a[k][2]);
                }
                }
}
void  addToResult(char c)
{
    int i;
    for( i=0;i<=m;i++)
        if(followResult[i]==c)
            return;
   followResult[m++]=c;
}
```

8) Construct a LL(1) parser for an expression

```
 #include
 #include
char str[25],st[25],*temp,v,ch,ch1;
char t[5][6][10]={"$","$","TX","TX","$","$", "+TX","$","$","$","e","e", "$","$","FY","FY","$","$",
"e","*FY","$","$","e","e", "$","$","i","(E)","$","$"};
int i,k,n,top=-1,r,c,m,flag=0;
void push(char t)
{
top++;
st[top]=t;
}
 char pop()
 {
 ch1=st[top];
 top--;
 return ch1;
 }
 main()
{
 printf("enter the string:\n");
scanf("%s",str);
n=strlen(str);
str[n++]='$';
 i=0;
 push('$');
push('E');
 printf("stack\tinput\toperation\n");
 while(i<=top;k++)
printf("%c",st[k]);
printf("\t");
for(k=i;k<n;k++)
printf("%c",str[k]);
printf("\t");
if(flag==1)
printf("pop");
 if(flag==2)
printf("%c->%s",ch,t[r][c]);
if(str[i]==st[top])
{
 flag=1;
ch=pop();
i++;
 }
 Else
 {
 flag=2;
if(st[top]=='E') r=0;
else if(st[top]=='X') r=1;
else if(st[top]=='T') r=2;
else if(st[top]=='Y') r=3;
else if(st[top]=='F') r=4;
else break; if(str[i]=='+') c=0;
else if(str[i]=='*') c=1;
else if(str[i]=='i') c=2;
else if(str[i]=='(') c=3;

else if(str[i]==')') c=4;

else if(str[i]=='$') c=5;

else

break;

if(strcmp(t[r][c],"$")==0)

 break;

 ch=pop();

temp=t[r][c];

 m=strlen(temp);

if(strcmp(t[r][c],"e")!=0)

 {

 for(k=m-1;k>=0;k--)

push(temp[k]);

}

 }

 printf("\n");

}

 if(i==n)

printf("\nparsed successfully");

else

printf("\nnot parsed");

 }
```

OUTPUT

```
1)
Enter any String(Append with $)i+i*i$
Stack           Input           Output
$E              i+i*i$
$HT             i+i*i$          E->TH
$HUF            i+i*i$          T->FU
$HUi            i+i*i$          F->i
$HU              +i*i$          POP
$H               +i*i$          U->ε
$HT+             +i*i$          H->+TH
$HT                i*i$          POP
$HUF               i*i$   T->FU
$HUi               i*i$   F->i
$HU                 *i$   POP
$HUF*               *i$   U->*FU
$HUF                 i$    POP
$HUi                 i$    F->i
$HU                   $     POP
$H                    $     U->ε
$                     $     H->ε
```



```
2)

Enter any String(Append with $)i+i**i$
Stack  Input  Output

$E     i+i**i$
$HT    i+i**i$ E->TH
$HUF   i+i**i$ T->FU
$HUii+i**i$ F->i
$HU    +i**i$  POP
$H     +i**i$  U->ε
$HT+   +i**i$  H->+TH
$HT    i**i$   POP
$HUF   i**i$   T->FU
$HUi   i**i$   F->i
$HU    **i$    POP
$HUF*   **i$    U->*FU
$HUF    *i$     POP
$HU$    *i$     F->$
Syntax Error
 Given String is not accepted
```

9) Write a C program to implement a shift-reduce parser.

```
#include"stdio.h"
#include"stdlib.h"
#include"string.h"
char ip_sym[15],stack[15];
int ip_ptr=0,st_ptr=0,len,i;
char temp[2],temp2[2];
char act[15];
void check();
void main()
{
printf("\n\t\t SHIFT REDUCE PARSER\n");
printf("\n GRAMMER\n");
printf("\n E->E+E\n E->E/E");
printf("\n E->E*E\n E->a/b");
printf("\n enter the input symbol:\t");
gets(ip_sym);
printf("\n\t stack implementation table");
printf("\n stack\t\t input symbol\t\t action");
printf("\n______\t\t ____________\t\t ______\n");
printf("\n $\t\t%s$\t\t\t--",ip_sym);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
len=strlen(ip_sym);

 for(i=0;i<=len-1;i++)
 {
 stack[st_ptr]=ip_sym[ip_ptr];
 stack[st_ptr+1]='\0';
 ip_sym[ip_ptr]=' ';
 ip_ptr++;
 printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
 strcpy(act,"shift ");
 temp[0]=ip_sym[ip_ptr];
 temp[1]='\0';
 strcat(act,temp);
 check();
 st_ptr++;
 }
 check();
 }
 void check()
 {
 int flag=0;
 temp2[0]=stack[st_ptr];
 temp2[1]='\0';
 if((isalpha(temp2[0])))
 {
```

```
 stack[st_ptr]='E';
printf("\n $%s\t\t%s$\t\t\tE->%s",stack,ip_sym,temp2);
 flag=1;
}
if((!strcmp(temp2,"+"))||(!strcmp(temp2,"*"))||(!strcmp(temp2,"/")))
{
 flag=1;
}
if((!strcmp(stack,"E+E"))||(!strcmp(stack,"E/E"))||(!strcmp(stack,"E*E")))
{
if(!strcmp(stack,"E+E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
}
else if(!strcmp(stack,"E/E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t %s$\t\t\tE->E/E",stack,ip_sym);
}
else
{
strcpy(stack,"E");
printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
```

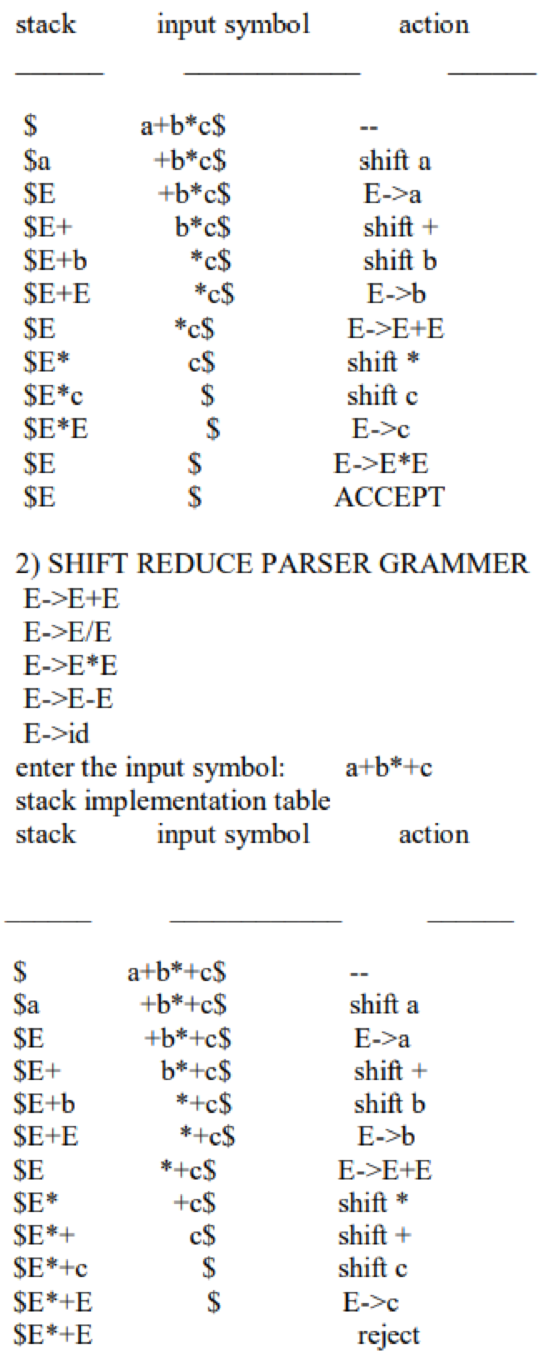
```
 }
 else
 {
 strcpy(stack,"E");
 printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
```

```
 }
 flag=1;
 st_ptr=0;
 }
 if(!strcmp(stack,"E")&&ip_ptr==len)
 {
 printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);
 exit(0);
 }
 if(flag==0)
 {
 printf("\n $%s\t\t%s$\t\t\tReject",stack,ip_sym);
 exit(0);
 }
 return;
 }
```

OUTPUT:

1)

```
SHIFT REDUCE PARSER GRAMMER
 E->E+E
 E->E/E
 E->E*E
 E->E-E
 E->id
enter the input symbol:        a+b*c
stack implementation table
```



```
stack        input symbol        action
______       ____________        ______

$            a+b*c$              --
$a            +b*c$              shift a
$E            +b*c$              E->a
$E+            b*c$              shift +
$E+b            *c$              shift b
$E+E            *c$              E->b
$E             *c$              E->E+E
$E*             c$              shift *
$E*c             $              shift c
$E*E             $              E->c
$E              $              E->E*E
$E              $              ACCEPT

2) SHIFT REDUCE PARSER GRAMMER
 E->E+E
 E->E/E
 E->E*E
 E->E-E
 E->id
enter the input symbol:      a+b*+c
stack implementation table
stack        input symbol        action


______       ____________        ______

$            a+b*+c$             --
$a            +b*+c$             shift a
$E            +b*+c$             E->a
$E+            b*+c$             shift +
$E+b            *+c$             shift b
$E+E            *+c$             E->b
$E             *+c$             E->E+E
$E*             +c$             shift *
$E*+             c$             shift +
$E*+c             $             shift c
$E*+E             $             E->c
$E*+E                            reject
```

10. Write a C program to verify whether the given grammar is Operator precedence grammar or not?

```
#include<stdlib.h>

#include<stdio.h>

#include<string.h>



// function f to exit from the loop
// if given condition is not true
void f()
{
        printf("Not operator grammar");
        exit(0);
}

void main()
{
        char grm[20][20], c;

        // Here using flag variable,
        // considering grammar is not operator grammar
        int i, n, j = 2, flag = 0;

        // taking number of productions from user
        scanf("%d", &n);
        for (i = 0; i < n; i++)
                scanf("%s", grm[i]);

        for (i = 0; i < n; i++) {
                c = grm[i][2];

                while (c != '\0') {

                        if (grm[i][3] == '+' || grm[i][3] == '-'
                                || grm[i][3] == '*' || grm[i][3] == '/')
                flag = 1;

            else {

                flag = 0;
                f();
            }

            if (c == '$') {
                flag = 0;
                f();
            }

            c = grm[i][++j];
        }
    }

    if (flag == 1)
        printf("Operator grammar");
}
```

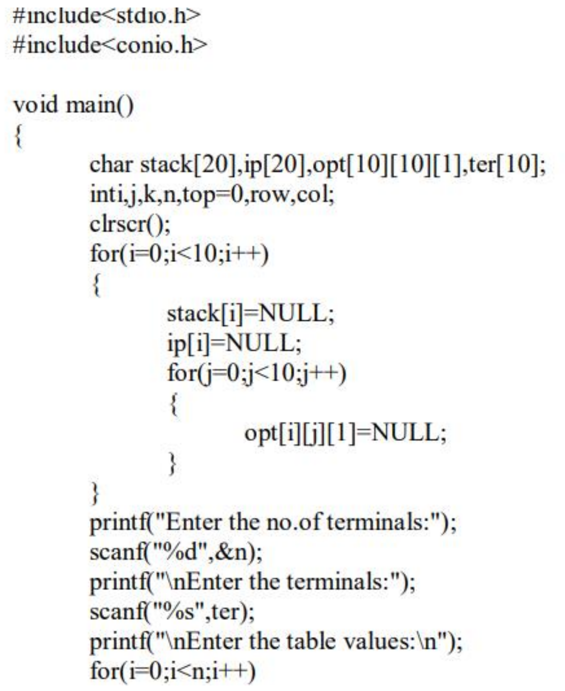
```
Input :3
A=A*A
B=AA
A=$

Output : Not operator grammar
```

```
Input :2
A=A/A
B=A+A
```

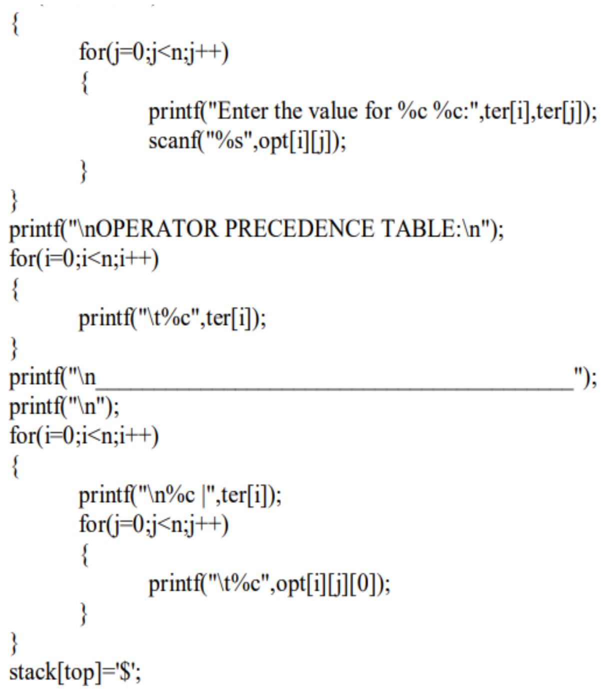
```
Output : Operator grammar
```

$ is a null production here which are also not allowed in operator grammars.

11.)Write a C program to implement a Operator precedence parser.

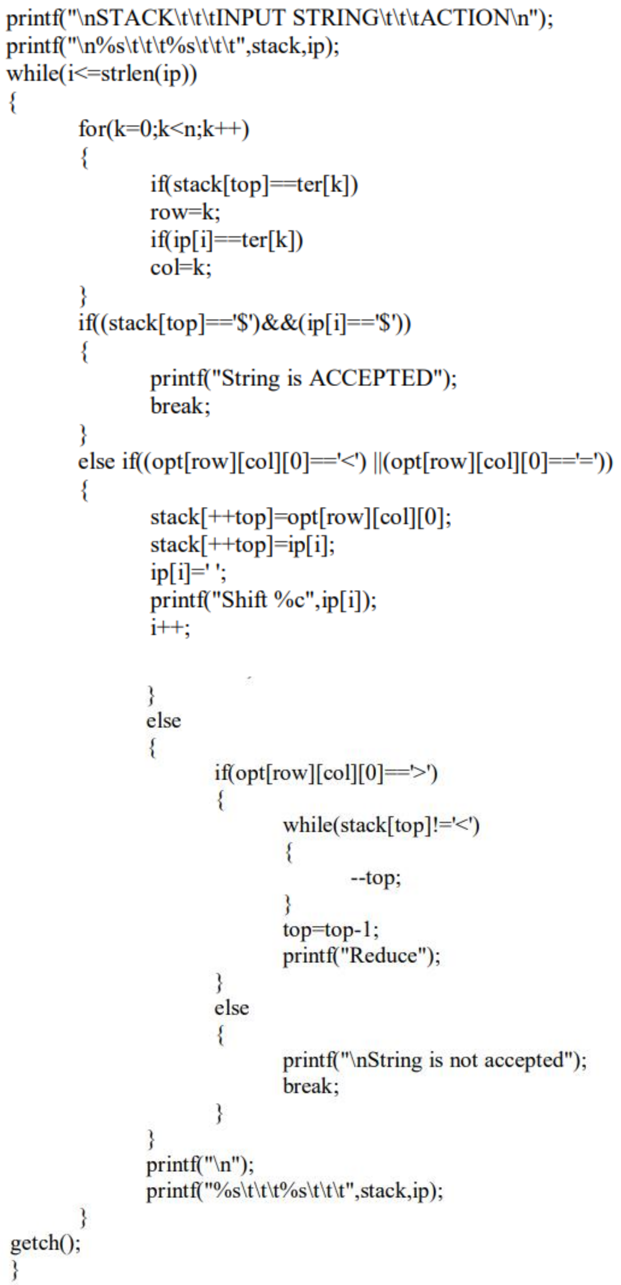


```
#include<stdio.h>
#include<conio.h>

void main()
{
        char stack[20],ip[20],opt[10][10][1],ter[10];
        inti,j,k,n,top=0,row,col;
        clrscr();
        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:");
        scanf("%d",&n);
        printf("\nEnter the terminals:");
        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("\nOPERATOR PRECEDENCE TABLE:\n");
for(i=0;i<n;i++)
{
        printf("\t%c",ter[i]);
}
printf("\n_________________________________________________");
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]='$';
```

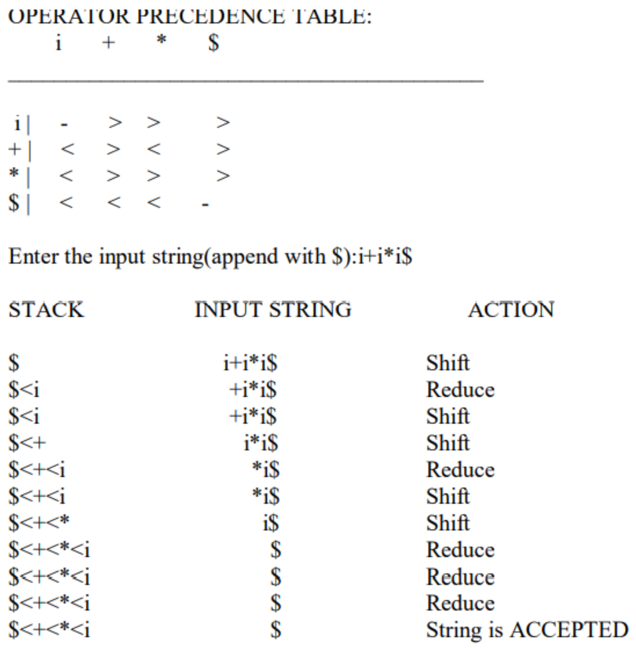
```
printf("\n\nEnter the input string(append with $):");

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])
                row=k;
                if(ip[i]==ter[k])
                col=k;
        }
        if((stack[top]=='$')&&(ip[i]=='$'))
        {
                printf("String is ACCEPTED");
                break;
        }
        else if((opt[row][col][0]=='<') ||(opt[row][col][0]=='='))
        {
                stack[++top]=opt[row][col][0];
                stack[++top]=ip[i];
                ip[i]=' ';
                printf("Shift %c",ip[i]);
                i++;

                }
                else
                {
                        if(opt[row][col][0]=='>')
                        {
                                while(stack[top]!='<')
                                {
                                        --top;
                                }
                                top=top-1;
                                printf("Reduce");
                        }
                        else
                        {
                                printf("\nString is not accepted");
                                break;
                        }
                }
                printf("\n");
                printf("%s\t\t\t%s\t\t\t",stack,ip);
        }
getch();
}
```

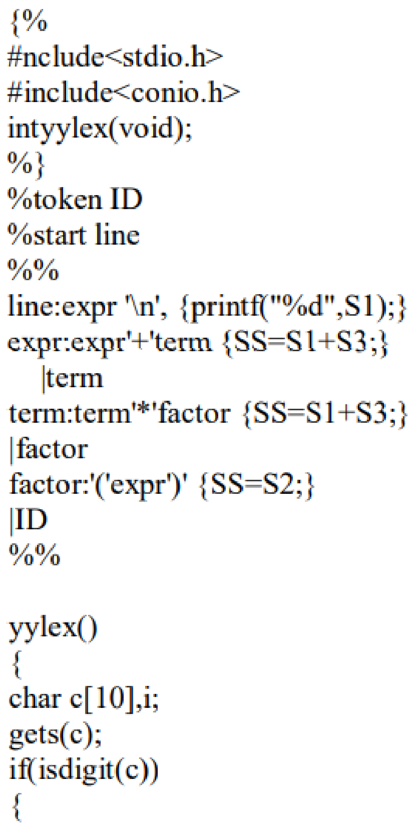
OUTPUT

```
Enter the no.of terminals:4

Enter the terminals:i+*$

Enter the table values:
Enter the value for i i:
-
Enter the value for i +:>
Enter the value for i *:>
Enter the value for i $:>
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 * *:>
Enter the value for * $:>
Enter the value for $ i:<
Enter the value for $ +:<
Enter the value for $ *:<
Enter the value for $ $:-
```



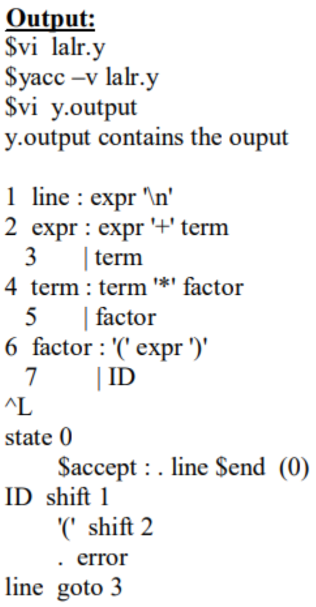
```
OPERATOR PRECEDENCE TABLE:
      i     +     *     $
________________________________________

 i |   -     >   >       >
 + |   <     >   <       >
 * |   <     >   >       >
 $ |   <     <   <     -

Enter the input string(append with $):i+i*i$

STACK              INPUT STRING            ACTION

$                      i+i*i$            Shift
$<i                     +i*i$            Reduce
$<i                     +i*i$            Shift
$<+                      i*i$            Shift
$<+<i                     *i$            Reduce
$<+<i                     *i$            Shift
$<+<*                      i$            Shift
$<+<*<i                     $            Reduce
$<+<*<i                     $            Reduce
$<+<*<i                     $            Reduce
$<+<*<i                     $            String is ACCEPTED
```

12. Write a C program to design a LALR bottom up parser for the given language.



```
{%
#nclude<stdio.h>
#include<conio.h>
intyylex(void);
%}
%token ID
%start line
%%
line:expr '\n', {printf("%d",S1);}
expr:expr'+'term {SS=S1+S3;}
    |term
term:term'*'factor {SS=S1+S3;}
|factor
factor:'('expr')' {SS=S2;}
|ID
%%

yylex()
{
char c[10],i;
gets(c);
if(isdigit(c))
{
```

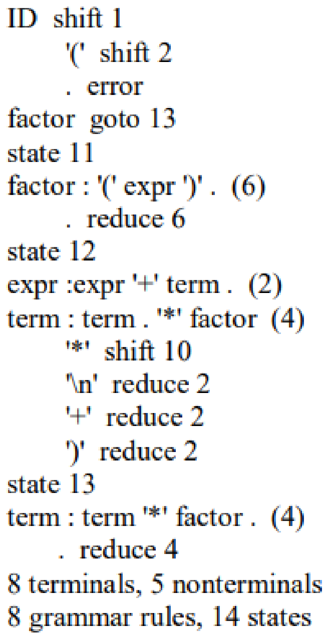
```
yylval=c;

return ID;

}

return c;

}
```



Output:

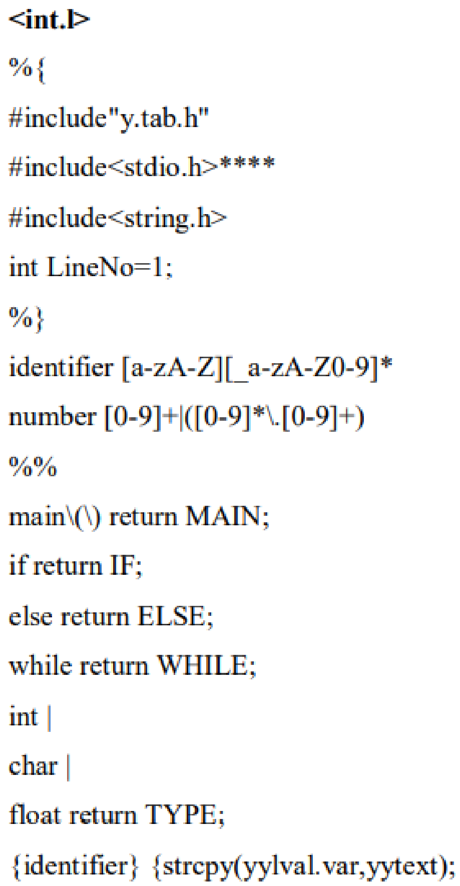
```
$vi  lalr.y
$yacc –v lalr.y
$vi  y.output
y.output contains the ouput

1  line : expr '\n'
2  expr : expr '+' term
  3       | term
4  term : term '*' factor
  5       | factor
6  factor : '(' expr ')'
  7         | ID
^L
state 0
       $accept : . line $end  (0)
ID  shift 1
       '('  shift 2
       .  error
line  goto 3
```

```
exprgoto 4
term  goto 5
state 1
factor : ID .  (7)
        .  reduce 7
state 2
factor : '(' . expr ')'  (6)
ID  shift 1
        '('  shift 2
        .  error
exprgoto 7
term  goto 5
factor  goto 6
state 3
        $accept : line . $end  (0)
        $end  accept
state 4
line :expr . '\n'  (1)
expr :expr . '+' term  (2)
        '\n'  shift 8
        '+'  shift 9
        .  error
state 5

expr : term .  (3)
term : term . '*' factor  (4)
       '*'  shift 10
       '\n'  reduce 3
       '+'  reduce 3
       ')'  reduce 3
state 6
term : factor .  (5)
       .  reduce 5
state 7
expr :expr . '+' term  (2)
factor : '(' expr . ')'  (6)
       '+'  shift 9
       ')'  shift 11
       .  error
state 8
line :expr '\n' .  (1)
       .  reduce 1
state 9
expr :expr '+' . term  (2)
ID  shift 1
       '('  shift 2
       .  error
term  goto 12
factor  goto 6
state 10
term : term '*' . factor  (4)
```



```
ID  shift 1
      '('  shift 2
      .  error
factor  goto 13
state 11
factor : '(' expr ')' .  (6)
      .  reduce 6
state 12
expr :expr '+' term .  (2)
term : term . '*' factor  (4)
      '*'  shift 10
      '\n'  reduce 2
      '+'  reduce 2
      ')'  reduce 2
state 13
term : term '*' factor .  (4)
     .  reduce 4
8 terminals, 5 nonterminals
8 grammar rules, 14 states
```

13. Write a program to convert the BNF rules into YACC form and write code to generate abstract syntax tree.

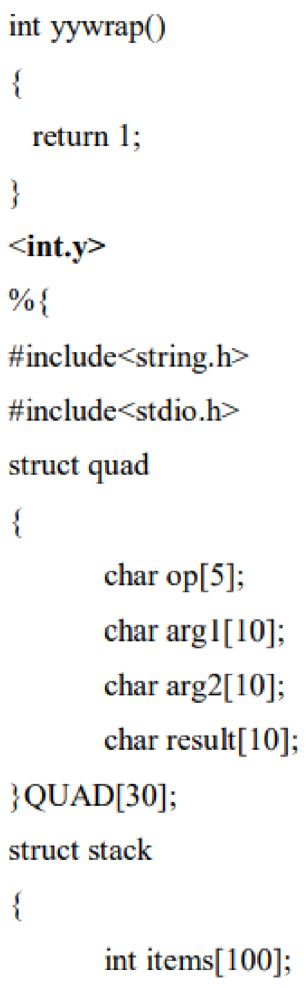


<int.l>

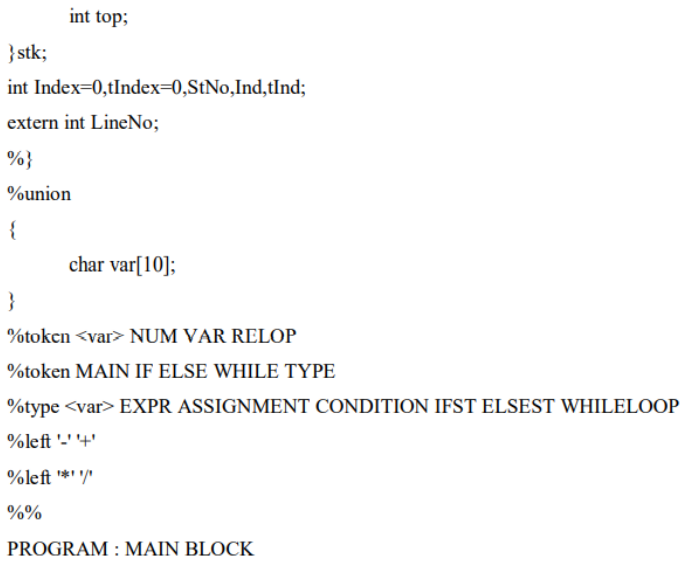
```
%{
#include"y.tab.h"
#include<stdio.h>****
#include<string.h>
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
```

```
return NUM;

}
```

```
\< |
\>|
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;
}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```



```
int yywrap()
{
  return 1;
}
<int.y>
%{
#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 Arg1  Arg2 Result" "\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);
}
```

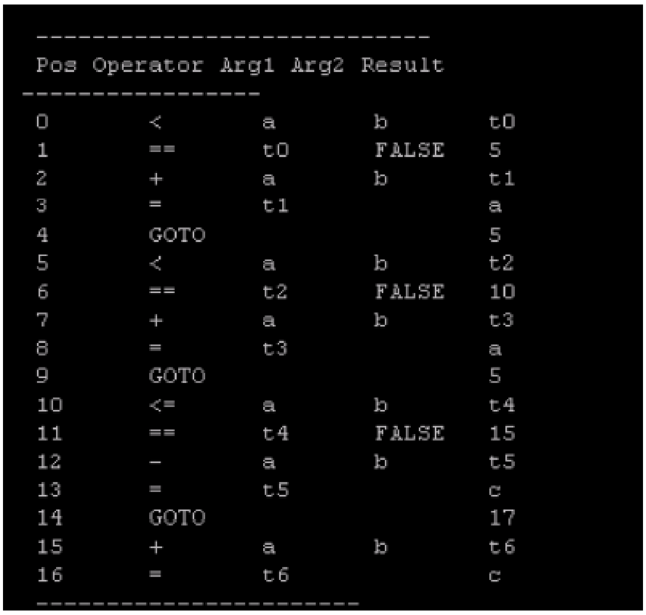
Input:

```
$vi test1.c
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;
}
}
```

Output:

```
$lex int.l
$yacc –d –v int.y
$gcc lex.yy.c y.tab.c  –lm
$./a.out test1.c
```



```
 ----------------------------
 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
 -----------------------
```