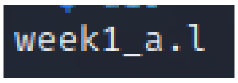
## Experiment-1

### 1.1) Write a lex program whose output is same as input

### Program:

```
[22A91A4409@Linux compilerdesign] $ vi week1_a.l
[22A91A4409@Linux compilerdesign] $ dir
```

**Output:**



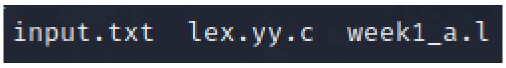
```
week1_a.l
```

**Program:**

```
%%
. {fprintf(yyout, "%s",yytext);}
%%
int main(){
extern FILE *yyin, *yyout;
yyin = fopen("input.txt","r");
yyout = fopen("output.txt","w");
yylex();
return 0;
}
```

```
[22A91A4409@Linux compilerdesign] $ lex week1_a.l
[22A91A4409@Linux compilerdesign] $ dir
```

**Output:**



```
input.txt  lex.yy.c  week1_a.l
```

```
[22A91A4409@Linux compilerdesign] $ gcc lex.yy.c -ll
[22A91A4409@Linux compilerdesign] $ vi input.txt
```

**Input text:**

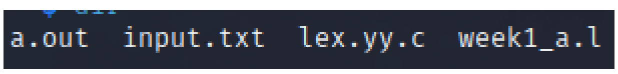
Dinesh kumar

22A91A4409

DS

[22A91A4409@Linux compilerdesign] $ dir

**Output:**

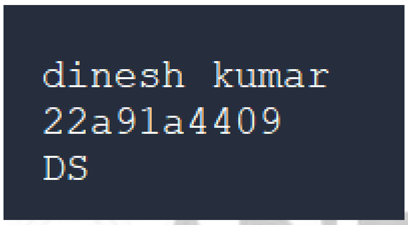


```
a.out  input.txt  lex.yy.c  week1_a.l
```

[22A91A4409@Linux compilerdesign] $ ./a.out input

[22A91A4409@Linux compilerdesign] $ cat output.txt

**Output:**



```
dinesh kumar
22a91a4409
DS
```

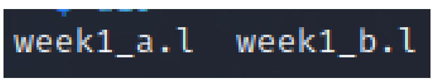
**1.2) Write a lex program which removes white spaces from its input file**

**Program:**

[22A91A4409@Linux compilerdesign] $ vi week1_b.l

[22A91A4409@Linux compilerdesign] $ dir

**Output:**



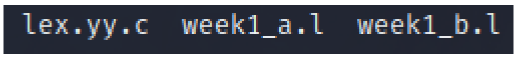
```
week1_a.l  week1_b.l
```

**Program:**

```
%%
\t+ {;}
. {fprintf (yyout, "%s", yytext);}
%%
int main(){
extern FILE *yyin, *yyout;
yyin = fopen("input.txt","r");
yyout = fopen("output.txt","w");
yylex();
return 0;
}
```

```
[22A91A4409@Linux compilerdesign] $ lex week1_b.l
[22A91A4409@Linux compilerdesign] $ dir
```

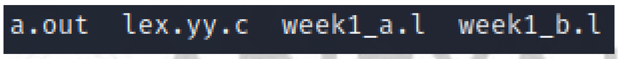
**Output:**



```
lex.yy.c  week1_a.l  week1_b.l
```

```
[22A91A4409@Linux compilerdesign] $ gcc lex.yy.c -ll
[22A91A4409@Linux compilerdesign] $ dir
```

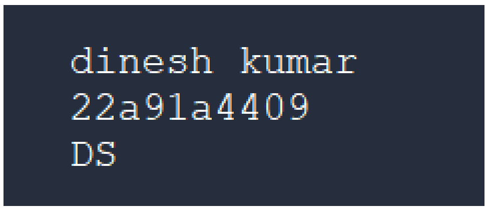
**Output:**



```
a.out  lex.yy.c  week1_a.l  week1_b.l
```

```
[22A91A4409@Linux compilerdesign] $ ./a.out input.txt
[22A91A4409@Linux compilerdesign] $ cat output.txt
```

## Output:



```
dinesh kumar
22a91a4409
DS
```

## Experiment-2:

## 2.1)

## Program:

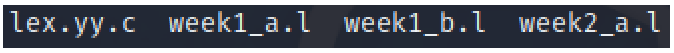
```
[22A91A4409@Linux compilerdesign] $ vi week2_a.l
```

**Program:**

```
%{
#include<stdio.h>
%}
delim [ |\t]
ws {delim}+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
num {digit}+(\.{digit}+)?(E[+|-]?{digit}+)?
%%
{ws} {printf("delimiter");}
If|else|then|int {printf("%s is a keyword", yytext);}
{id} {printf("%s is an identifier",yytext);}
{num} {printf("it is a number");}
%%
int main(){
yylex();
return 0;
}
```

```
[22A91A4409@Linux compilerdesign] $ lex week2_a.l
[22A91A4409@Linux compilerdesign] $ dir
```

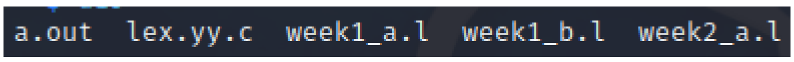
**Output:**



```
lex.yy.c  week1_a.l  week1_b.l  week2_a.l
```

```
[22A91A4409@Linux compilerdesign] $ gcc lex.yy.c -ll
[22A91A4409@Linux compilerdesign] $ dir
```

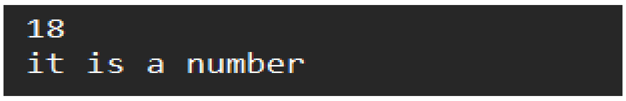
**Output:**



```
a.out  lex.yy.c  week1_a.l  week1_b.l  week2_a.l
```

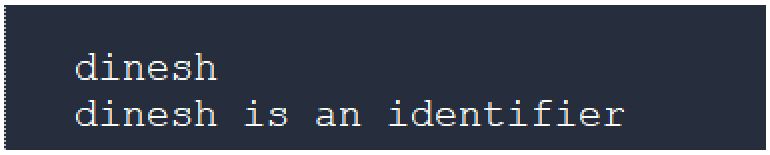
```
[22A91A4409@Linux compilerdesign] $ ./a.out
```

**Output:**



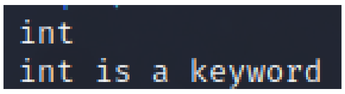
```
18
it is a number
```

**Output:**



```
dinesh
dinesh is an identifier
```

**Output:**



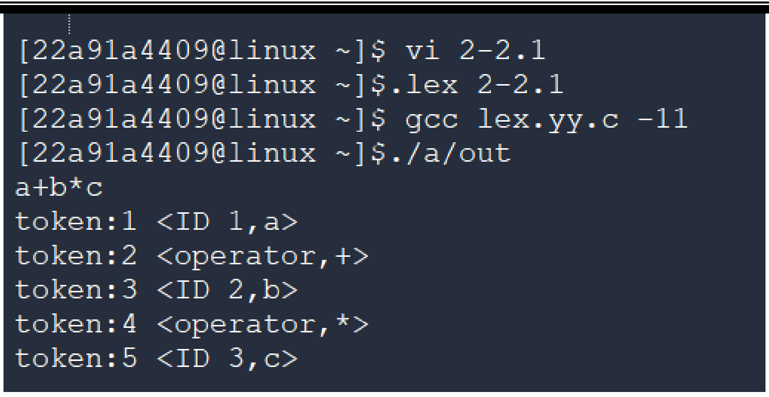
```
int
int is a keyword
```

**2.2) Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines.**

**Program:**

```
%{
#include<stdio.h>
int i=0,id=0;
%}
%%
[#].*[<].*[>]\n {}
[ \t\n]+ {}
\/\/.*\n {}
\/\*(.*\n)*.*\*\/ {}
auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int|long|
register|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|volatile|whil
e
 {printf("token: %d < keyword, %s >\n",++i,yytext);}
[+\-\*\/%<>] {printf("token: %d < operator, %s >\n",++i,yytext);}
[();{}] {printf("token: %d < special char, %s >\n",++i,yytext);}
[0-9]+ {printf("token: %d < constant, %s >\n",++i,yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {printf("token: %d < ID %d, %s >\n",++i,++id,yytext);}
^[^a-zA-Z_] {printf("ERROR INVALID TOKEN %s\n",yytext);}
%%
```

**Output:**



```
[22a91a4409@linux ~]$ vi 2-2.1
[22a91a4409@linux ~]$.lex 2-2.1
[22a91a4409@linux ~]$ gcc lex.yy.c -11
[22a91a4409@linux ~]$./a/out
a+b*c
token:1 <ID 1,a>
token:2 <operator,+>
token:3 <ID 2,b>
token:4 <operator,*>
token:5 <ID 3,c>
```

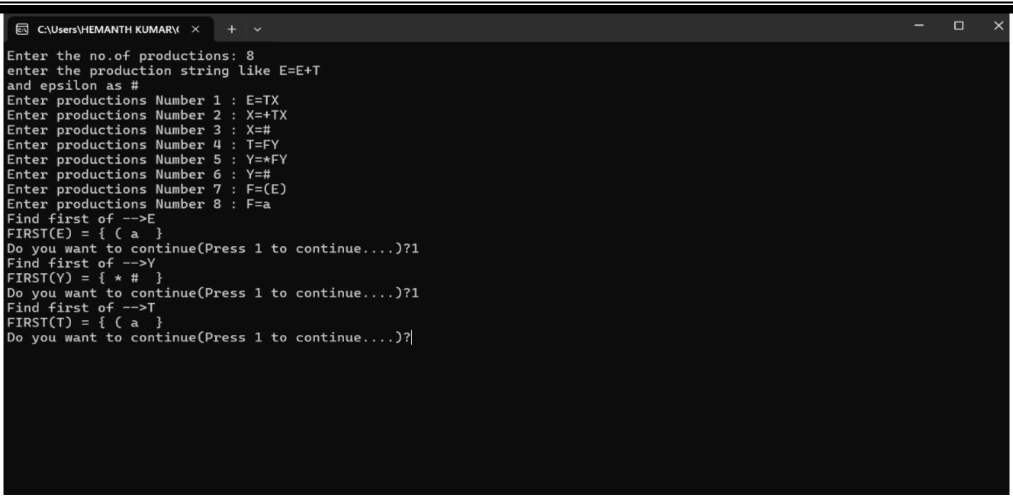
## Week-3

## Program:

## a)First()

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int nop,m=0;
char prod[10][10],res[10];
void first(char c);
void result(char);
int main()
{       int I,choice;
        char c;
        printf("Enter the no.of productions: ");
        scanf("%d", &nop);
        printf("enter the production string like E=E+T \nand epsilon as #\n");
        for(i=0;i<nop;i++)
        {       printf("Enter productions Number %d : ",i+1);
                scanf("%s",prod[i]);}do{       m=0;
                memset(res,'\0',sizeof(res));
                printf("Find first of -->");
                scanf(" %c",&c);
                first(c);
                printf("FIRST(%c) = { ",c);
                for(i=0;i<m;i++)
                        printf("%c ",res[i]);
                printf(" }\n");
                printf("Do you want to continue(Press 1 to continue....)?");
                scanf("%d",&choice);
        }while(choice==1);
        return 0;}
void first(char c)
{       int k;
        if(!(isupper(c)))
                result(c);
        for(k=0;k<nop;k++)
        {if(prod[k][0]==c)
                {if(prod[k][2]=='#')
                                result('#');
                        else if(prod[k][2]==c)
                                return ;
                        else
                                first(prod[k][2]);}}}
void result(char c)
{       int i;
        for( i=0;i<=m;i++)
                if(res[i]==c)
                        return;
        res[m++]=c;}
```

**Output:**



```
Enter the no.of productions: 8
enter the production string like E=E+T
and epsilon as #
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=#
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=#
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=a
Find first of -->E
FIRST(E) = { ( a  }
Do you want to continue(Press 1 to continue....)?1
Find first of -->Y
FIRST(Y) = { * #  }
Do you want to continue(Press 1 to continue....)?1
Find first of -->T
FIRST(T) = { ( a  }
Do you want to continue(Press 1 to continue....)?
```

## b)Follow()

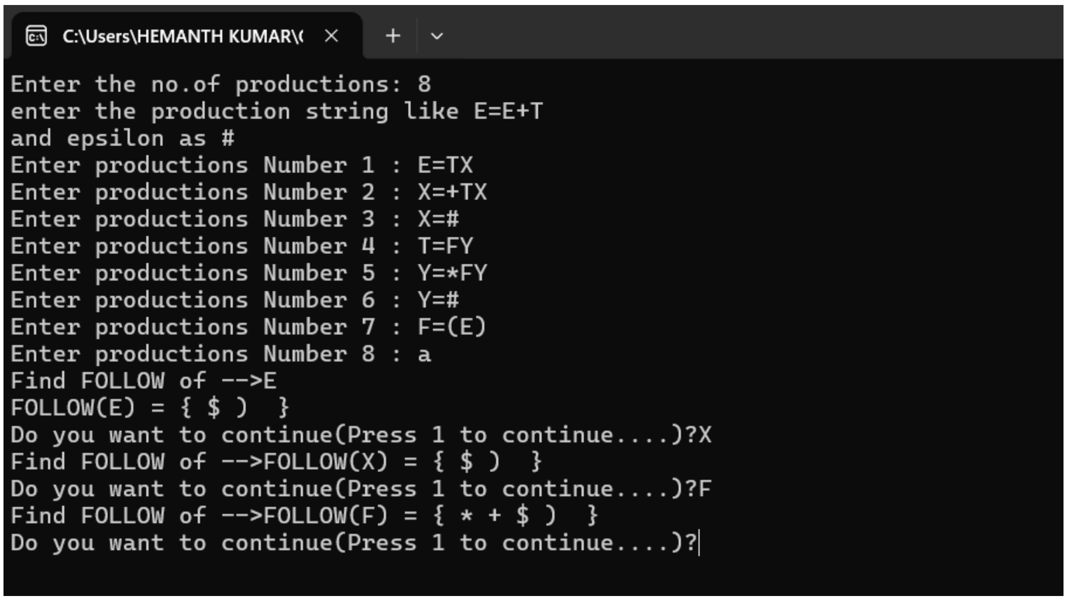
## Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int nop,m=0;
char prod[10][10],res[10];
void FOLLOW(char c);
void first(char c);
void result(char);
int main()
{       int I,choice;
        char c;
        printf("Enter the no.of productions: ");
        scanf("%d", &nop);
        printf("enter the production string like E=E+T \nand epsilon as #\n");
        for(i=0;i<nop;i++)
        {       printf("Enter productions Number %d : ",i+1);
                scanf("%s",prod[i]);}do{
```

```
                m=0;
                memset(res,'\0',sizeof(res));
                printf("Find FOLLOW of -->");
                scanf(" %c",&c);
                if(isupper(c))
                        FOLLOW(c);
                else
                {       printf("not possible\n");
                        return 0;}
                printf("FOLLOW(%c) = { ",c);
                for(i=0;i<m;i++)
                printf("%c ",res[i]);
                printf(" }\n");
                printf("Do you want to continue(Press 1 to continue....)?");
                scanf("%d",&choice);
        }while(choice==1);
        return 0;}
void FOLLOW(char c){
        int i,j;
        if(prod[0][0]==c)
                result('$');
        for(i=0;i<nop;i++)
        {for(j=2;j<=strlen(prod[i]);j++)
                {if(prod[i][j]==c)
                        {if(prod[i][j+1]!='\0')
                                        first(prod[i][j+1]);
                                if(prod[i][j+1]=='\0'&&c!=prod[i][0])
                                        FOLLOW(prod[i][0]);}}}}
void first(char c)
{       int k;
```

```
        if(!(isupper(c)))
                result(c);
        for(k=0;k<nop;k++)
        {if(prod[k][0]==c)
                {if(prod[k][2]=='#')
                                FOLLOW(prod[k][0]);
                        else if(prod[k][2]==c)
                                return ;
                        else if(islower(prod[k][2]))
                                result(prod[k][2]);
                        else    first(prod[k][2]);}}}
void result(char c)
{       int i;
        for( i=0;i<=m;i++)
                if(res[i]==c)
                        return;
        res[m++]=c;
}
```

## Output:



```
Enter the no.of productions: 8
enter the production string like E=E+T
and epsilon as #
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=#
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=#
Enter productions Number 7 : F=(E)
Enter productions Number 8 : a
Find FOLLOW of -->E
FOLLOW(E) = { $ )  }
Do you want to continue(Press 1 to continue....)?X
Find FOLLOW of -->FOLLOW(X) = { $ )  }
Do you want to continue(Press 1 to continue....)?F
Find FOLLOW of -->FOLLOW(F) = { * + $ )  }
Do you want to continue(Press 1 to continue....)?
```

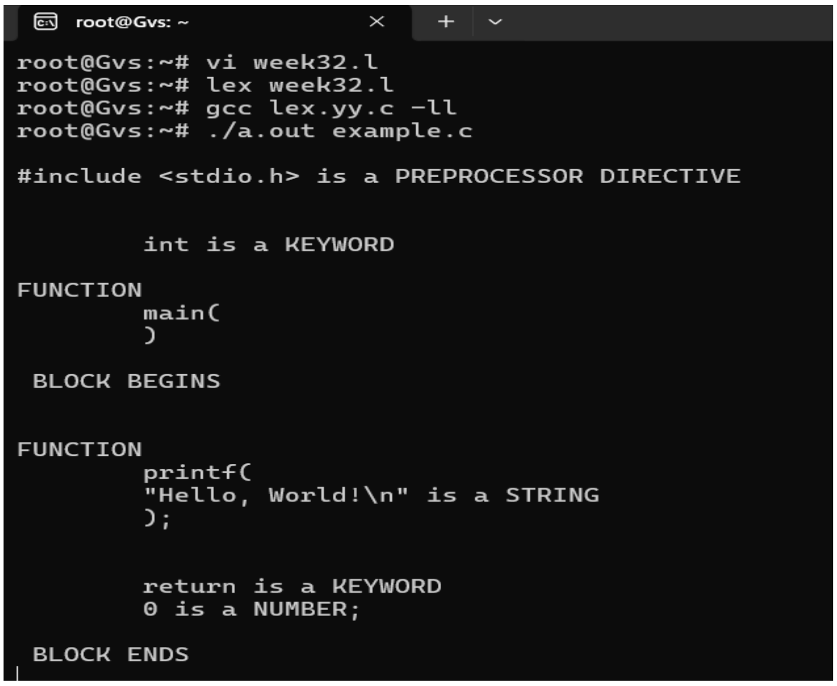
**3.2)**

## Program:

```
%{int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
{ printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int|float|char|double|while|for|do|if|break|continue|void|switch|case|long|struct|const|typedef|re
turn|else|goto {printf("\n\t%s is a KEYWORD",yytext);}
"/*"{COMMENT = 1;} {printf("\n\n\t%s is a COMMENT\n",yytext) ;}
"*/"{COMMENT = 0;} {printf("\n\n\t%s is a COMMENT\n",yytext);}
{identifier}\(   {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}\{
{if(!COMMENT) printf("\n BLOCK BEGINS");}\}{if(!COMMENT) printf("\n BLOCK
ENDS");}{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\"    {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}
[0-9]+  {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)?   {if(!COMMENT) printf("\n\t");ECHO;printf("\n");ECHO;
= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<=|\>=|\<|==|\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%int main(int argc,char **argv)
{if  (argc > 1)
{FILE *file;
file = fopen(argv[1],"r");
if(!file)
{printf("could not open %s \n",argv[1]);
exit(0);}
yyin = file;}
yylex();
printf("\n\n");
return 0;
int yywrap()
```

{return 0;}

## Output:



```
root@Gvs: ~
root@Gvs:~# vi week32.l
root@Gvs:~# lex week32.l
root@Gvs:~# gcc lex.yy.c -ll
root@Gvs:~# ./a.out example.c

#include <stdio.h> is a PREPROCESSOR DIRECTIVE


        int is a KEYWORD

FUNCTION
        main(
        )

 BLOCK BEGINS


FUNCTION
        printf(
        "Hello, World!\n" is a STRING
        );


        return is a KEYWORD
        0 is a NUMBER;

 BLOCK ENDS
```

## Week-4

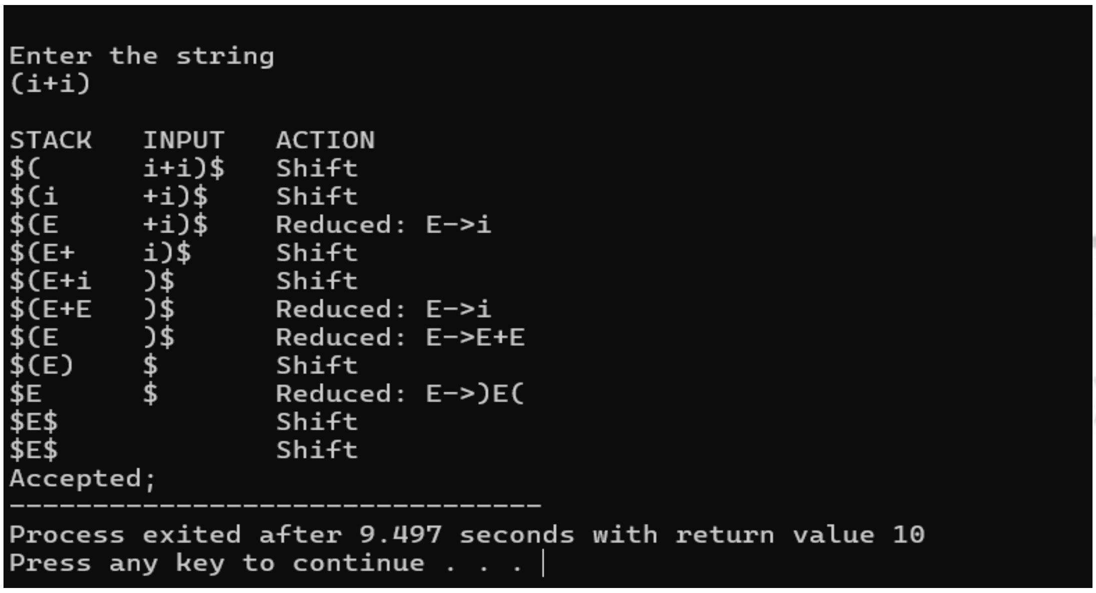
## Program:

```
#include<stdio.h>

#include<string.h>

char *input;

int i=0;

char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};

int top=0,l;

char prec[9][9]={/*input*/

      /*stack   +   -   *   /   ^   i   (   )   $  */

      /*  + */  '>', '>','<','<','<','<','<','>','>',

      /*  - */  '>', '>','<','<','<','<','<','>','>',

      /*  * */  '>', '>','>','>','<','<','<','>','>',

      /*  / */  '>', '>','>','>','<','<','<','>','>',

      /*  ^ */  '>', '>','>','>','<','<','<','>','>',

      /*  i */  '>', '>','>','>','>','e','e','>','>',

      /*  ( */  '<', '<','<','<','<','<','<','>','e',
        /*  ) */  '>', '>','>','>','>','e','e','>','>',
        /*  $ */  '<', '<','<','<','<','<','<','<','>',};
int getindex(char c)
{switch(c)
  {case '+':return 0;
  case '-':return 1;
  case '*':return 2;
  case '/':return 3;
  case '^':return 4;
  case 'i':return 5;
  case '(':return 6;
  case ')':return 7;
  case '$':return 8;}}
int shift(){stack[++top]=*(input+i++);
stack[top+1]='\0';}
int reduce()
{int i,len,found,t;
for(i=0;i<5;i++)//selecting handles
  {len=strlen(handles[i]);
  if(stack[top]==handles[i][0]&&top+1>=len)
    {  found=1;
    for(t=0;t<len;t++)
      {if(stack[top-t]!=handles[i][t])
        {
        found=0;
        break;}          }
    if(found==1)
      {stack[top-t+1]='E';
      top=top-t+1;
      strcpy(lasthandle,handles[i]);
```

```
        stack[top+1]='\0';
        return 1;//successful reduction}}}
return 0;}
void dispstack()
{int j;
for(j=0;j<=top;j++)
  printf("%c",stack[j]);}
void dispinput()
{int j;
for(j=i;j<l;j++)
  printf("%c",*(input+j));}
void main(){
int j;
input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
l=strlen(input);
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
        {shift();
        printf("\n");
        dispstack();
        printf("\t");
        dispinput();
        printf("\tShift");
        if(prec[getindex(stack[top])][getindex(input[i])]=='>')
                {while(reduce())
                        {printf("\n");
```

```
                    dispstack();
                    printf("\t");
                    dispinput();
                    printf("\tReduced: E->%s",lasthandle);}}}
if(strcmp(stack,"$E$")==0)
  printf("\nAccepted;");
else
  printf("\nNot Accepted;");
}
```

**Output:**



```
Enter the string
(i+i)

STACK   INPUT   ACTION
$(      i+i)$   Shift
$(i     +i)$    Shift
$(E     +i)$    Reduced: E->i
$(E+    i)$     Shift
$(E+i   )$      Shift
$(E+E   )$      Reduced: E->i
$(E     )$      Reduced: E->E+E
$(E)    $       Shift
$E      $       Reduced: E->)E(
$E$             Shift
$E$             Shift
Accepted;
--------------------------------
Process exited after 9.497 seconds with return value 10
Press any key to continue . . .
```

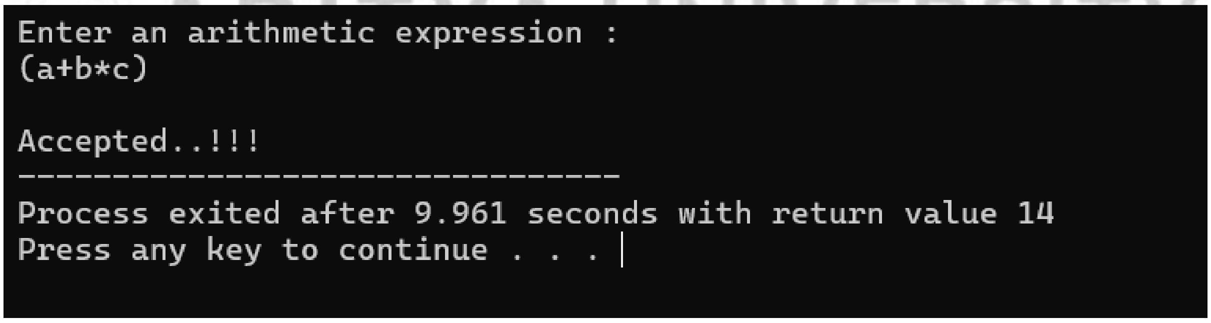
**4.2)**

**Program:**

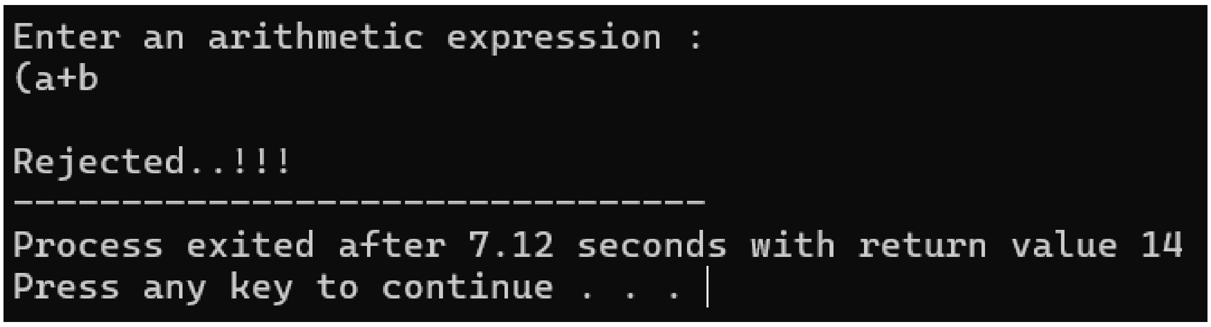
```
#include<stdio.h>
#include<string.h>
char input[10];
int i=0,error=0;
void E();
void T();
void Eprime();
void Tprime();
void F();
void main()
{  printf("Enter an arithmetic expression :\n");
   gets(input);
   E();
   if(strlen(input)==i&&error==0)
   printf("\nAccepted..!!!");
   else
   printf("\nRejected..!!!");}
void E()
{  T();
   Eprime();
}
void Eprime()
{  if(input[i]=='+')
   {  i++;
      T();
      Eprime();}
}
void T()
{  F();
   Tprime();
}
void Tprime()
{ if(input[i]=='*')
   { i++;
      F();
      Tprime();}
}
void F()
```

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

**Output:**



```
Enter an arithmetic expression :
(a+b*c)

Accepted..!!!
--------------------------------
Process exited after 9.961 seconds with return value 14
Press any key to continue . . .
```



```
Enter an arithmetic expression :
(a+b

Rejected..!!!
--------------------------------
Process exited after 7.12 seconds with return value 14
Press any key to continue . . .
```

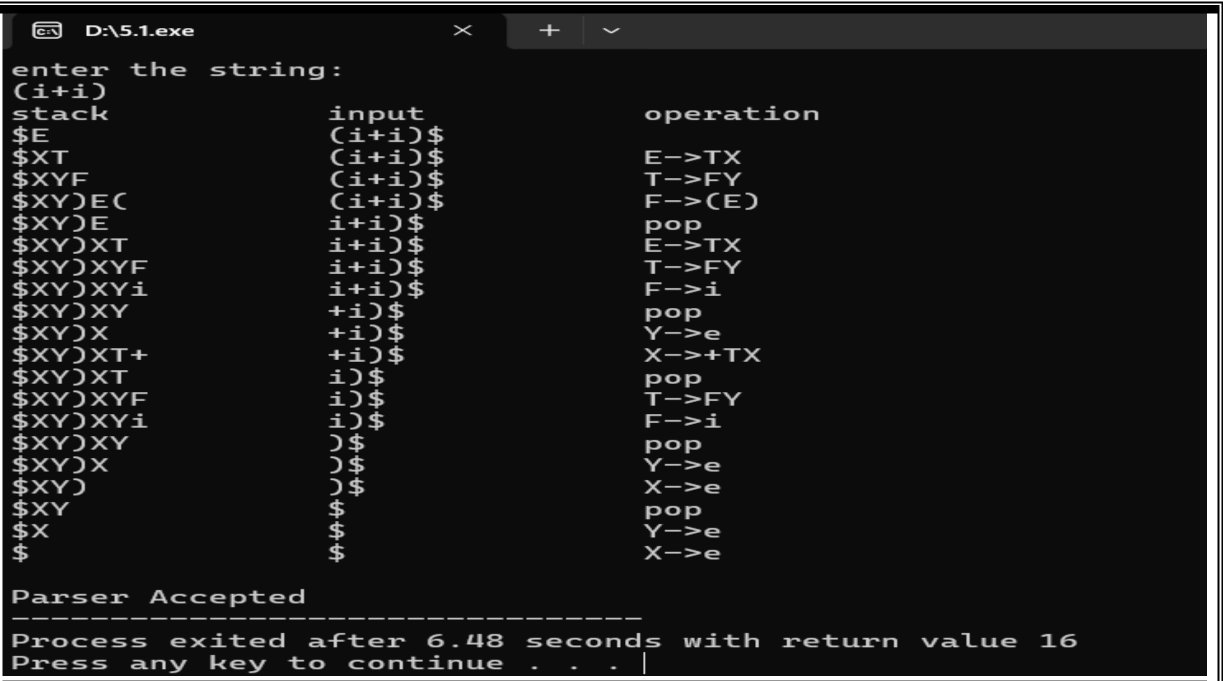
## Week-5

## 5.1

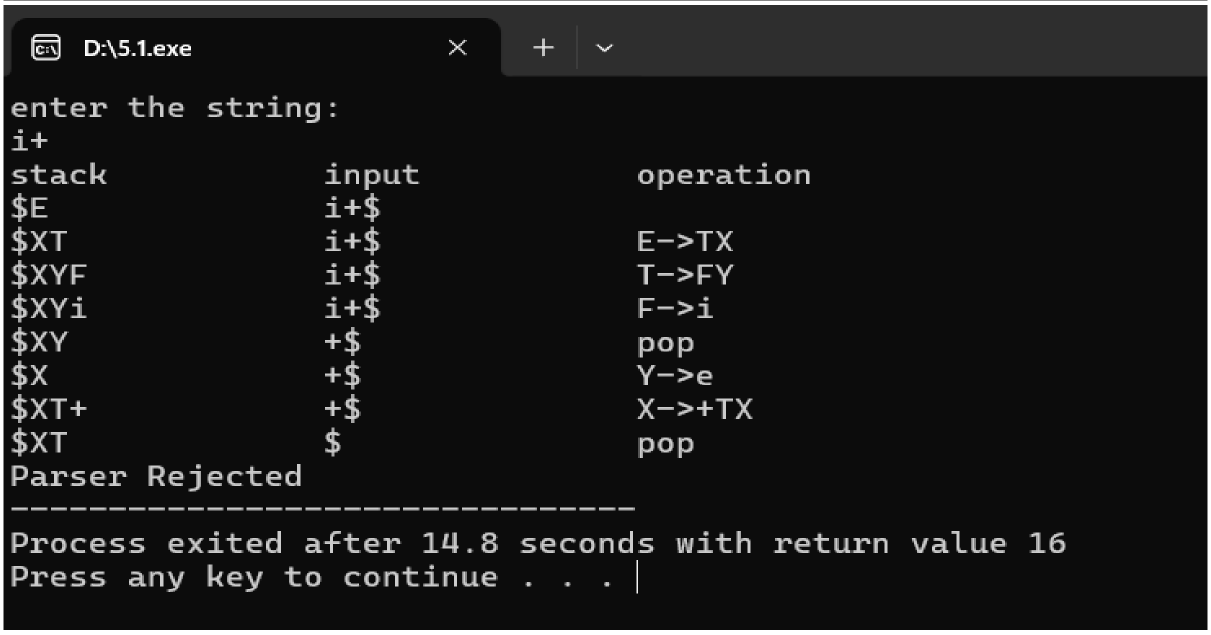
## Program:

```
#include<stdio.h>
#include<string.h>
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('$');
push('E');
printf("stack\t\tinput\t\toperation\n");
while(i<n){
for(k=0;k<=top;k++)
printf("%c",st[k]);
printf("\t\t");
for(k=i;k<n;k++)
printf("%c",str[k]);
printf("\t\t");
if(flag==1)
printf("pop");
if(flag==2)
printf("%c->%s",ch,t[r][c]);
if(str[i]==st[top]){
flag=1;
ch=pop();
i++;}
else
{flag=2;
if(st[top]=='E')
r=0;
else if(st[top]=='X')
r=1;
else if(st[top]=='T')
r=2;
else if(st[top]=='Y')
r=3;
else if(st[top]=='F')
r=4;
else
break;
if(str[i]=='+')
c=0;
else if(str[i]=='*')
c=1;
else if(str[i]=='i')
c=2;
else if(str[i]=='(')
c=3;
else if(str[i]==')')
c=4;
else if(str[i]=='$')
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("\nParser Accepted");
else{printf("\nParser Rejected");}}
```

**Output:**



```
D:\5.1.exe
enter the string:
(i+i)
stack               input               operation
$E                  (i+i)$
$XT                 (i+i)$              E->TX
$XYF                (i+i)$              T->FY
$XY)E(              (i+i)$              F->(E)
$XY)E               i+i)$               pop
$XY)XT              i+i)$               E->TX
$XY)XYF             i+i)$               T->FY
$XY)XYi             i+i)$               F->i
$XY)XY              +i)$                pop
$XY)X               +i)$                Y->e
$XY)XT+             +i)$                X->+TX
$XY)XT              i)$                 pop
$XY)XYF             i)$                 T->FY
$XY)XYi             i)$                 F->i
$XY)XY              )$                  pop
$XY)X               )$                  Y->e
$XY)                )$                  X->e
$XY                 $                   pop
$X                  $                   Y->e
$                   $                   X->e

Parser Accepted
--------------------------------
Process exited after 6.48 seconds with return value 16
Press any key to continue . . .
```



```
D:\5.1.exe
enter the string:
i+
stack               input               operation
$E                  i+$
$XT                 i+$                 E->TX
$XYF                i+$                 T->FY
$XYi                i+$                 F->i
$XY                 +$                  pop
$X                  +$                  Y->e
$XT+                +$                  X->+TX
$XT                 $                   pop
Parser Rejected
--------------------------------
Process exited after 14.8 seconds with return value 16
Press any key to continue . . .
```

**5.2)**

## Program:

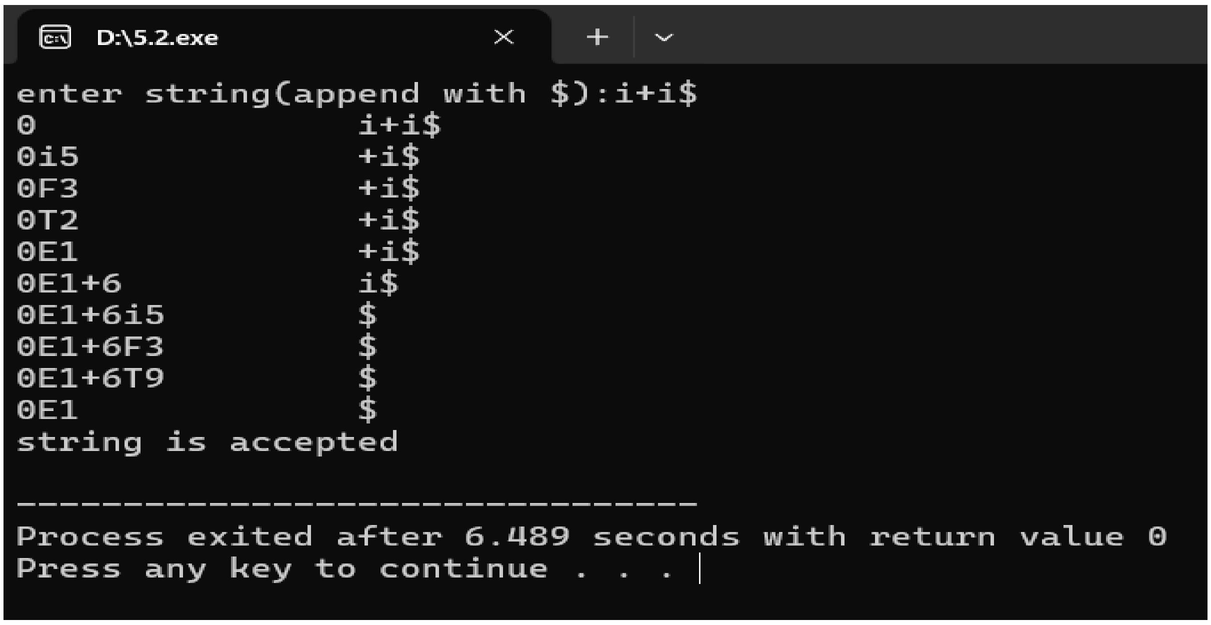
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int st[20], top = -1;
char input[20];
```

```
int encode(char ch) {
    switch (ch) {
    case 'i':
        return 0;
    case '+':
        return 1;
    case '*':
        return 2;
    case '(':
        return 3;
    case ')':
        return 4;
    case '$':
        return 5;
    case 'E':
        return 6;
    case 'T':
        return 7;
    case 'F':
        return 8;    }
    return -1;}
char decode(int n) {
    switch (n) {
    case 0:
        return ('i');
    case 1:
        return ('+');
    case 2:
        return ('*');
    case 3:
        return ('(');
    case 4:
        return (')');
    case 5:
        return ('$');
    case 6:
        return ('E');
    case 7:
        return ('T');
    case 8:
        return ('F');    }
    return 'z';}
void push(int n) {
    st[++top] = n;}
int pop() {
    return (st[top--]);}
void display(int p, char * ptr) {
    int l;
    for (l = 0; l <= top; l++) {
        if (l % 2 == 1)
            printf("%c", decode(st[l]));
        else
            printf("%d", st[l]);    }
    printf("\t\t");
    for (l = p; ptr[l]; l++)
        printf("%c", ptr[l]);
    printf("\n");}
void main() {
    char t1[20][20], pr[20][20], xy;
    int inp[20], t2[20][20], gt[20][20];
    int i, k, x, y, tx = 0, ty = 0, len;
    strcpy(pr[1], "E E+T");
    strcpy(pr[2], "E T");
    strcpy(pr[3], "T T*F");
    strcpy(pr[4], "T F");
    strcpy(pr[5], "F (E)");
    strcpy(pr[6], "F i");
    t2[2][1] = t2[2][4] = t2[2][5] = 2;
    t2[3][1] = t2[3][2] = t2[3][4] = t2[3][5] = 4;
    t2[5][1] = t2[5][2] = t2[5][4] = t2[5][5] = 6;
    t2[9][1] = t2[9][4] = t2[9][5] = 1;
    t2[10][1] = t2[10][2] = t2[10][4] = t2[10][5] = 3;
    t2[11][2] = t2[11][1] = t2[11][4] = t2[11][5] = 5;
    t1[2][1] = t1[2][4] = t1[2][5] = 'r';
    t1[3][1] = t1[3][2] = t1[3][4] = 'r';
    t1[3][5] = t1[5][1] = t1[5][2] = 'r';
    t1[5][4] = t1[5][5] = t1[9][1] = t1[9][4] = 'r';
    t1[9][5] = t1[10][1] = t1[10][2] = t1[10][4] = t1[10][5] = 'r';
    t1[11][1] = t1[11][4] = t1[11][2] = t1[11][5] = 'r';
    t1[0][0] = t1[4][0] = t1[6][0] = t1[7][0] = t1[0][3] = t1[4][3] = t1[6][3] = 's';
    t1[2][2] = t1[9][2] = t1[8][4] = t1[1][1] = t1[8][1] = t1[7][3] = 's';
    t1[1][5] = 'a';
    t2[0][0] = t2[4][0] = t2[6][0] = t2[7][0] = 5;
    t2[0][3] = t2[4][3] = t2[6][3] = t2[7][3] = 4;
    t2[2][2] = t2[9][2] = 7;
    t2[8][4] = 11;
    t2[1][1] = t2[8][1] = 6;
    gt[0][6] = 1;
    gt[0][7] = gt[4][7] = 2;
    gt[0][8] = gt[4][8] = gt[6][8] = 3;
```

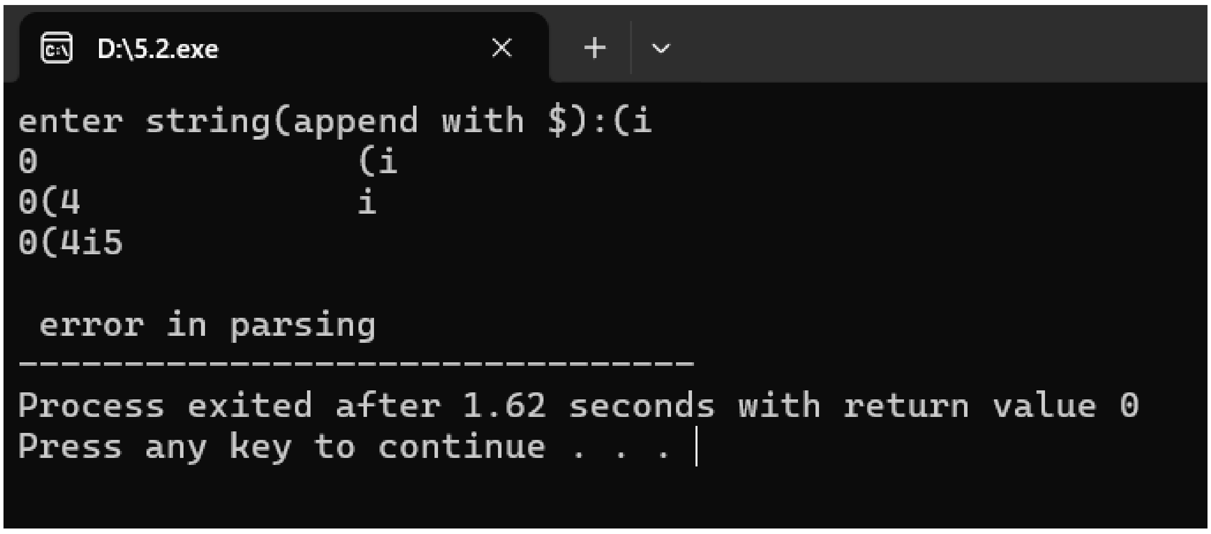
```
gt[4][6] = 8;
gt[6][7] = 9;
gt[7][8] = 10;
printf("enter string(append with $):");
scanf("%s", input);
for (k = 0; input[k]; k++) {
    inp[k] = encode(input[k]);
    if (input[k] < 0 || inp[k] > 5) {
        printf("\n error in input");
        exit(0);}}
push(0);
i = 0;
while (1) {
    x = st[top];
    y = inp[i];
    display(i, input);
    if (t1[x][y] == 'a') {
        printf("string is accepted \n");
        exit(0);
    } else if (t1[x][y] == 's') {
        push(inp[i]);
        push(t2[x][y]);
        i++;
    } else if (t1[x][y] == 'r') {
        len = strlen(pr[t2[x][y]]) - 2;
        xy = pr[t2[x][y]][0];
        ty = encode(xy);
        for (k = 1; k <= 2 * len; k++)
            pop();
        tx = st[top];
```

```
          push(ty);
          push(gt[tx][ty]);
      } else {
          printf("\n error in parsing");
          exit(0);
      }
   }
}
```

**Output:**



```
D:\5.2.exe
enter string(append with $):i+i$
0                i+i$
0i5              +i$
0F3              +i$
0T2              +i$
0E1              +i$
0E1+6            i$
0E1+6i5          $
0E1+6F3          $
0E1+6T9          $
0E1              $
string is accepted

--------------------------------
Process exited after 6.489 seconds with return value 0
Press any key to continue . . .
```



```
D:\5.2.exe
enter string(append with $):(i
0                (i
0(4              i
0(4i5

 error in parsing
--------------------------------
Process exited after 1.62 seconds with return value 0
Press any key to continue . . .
```

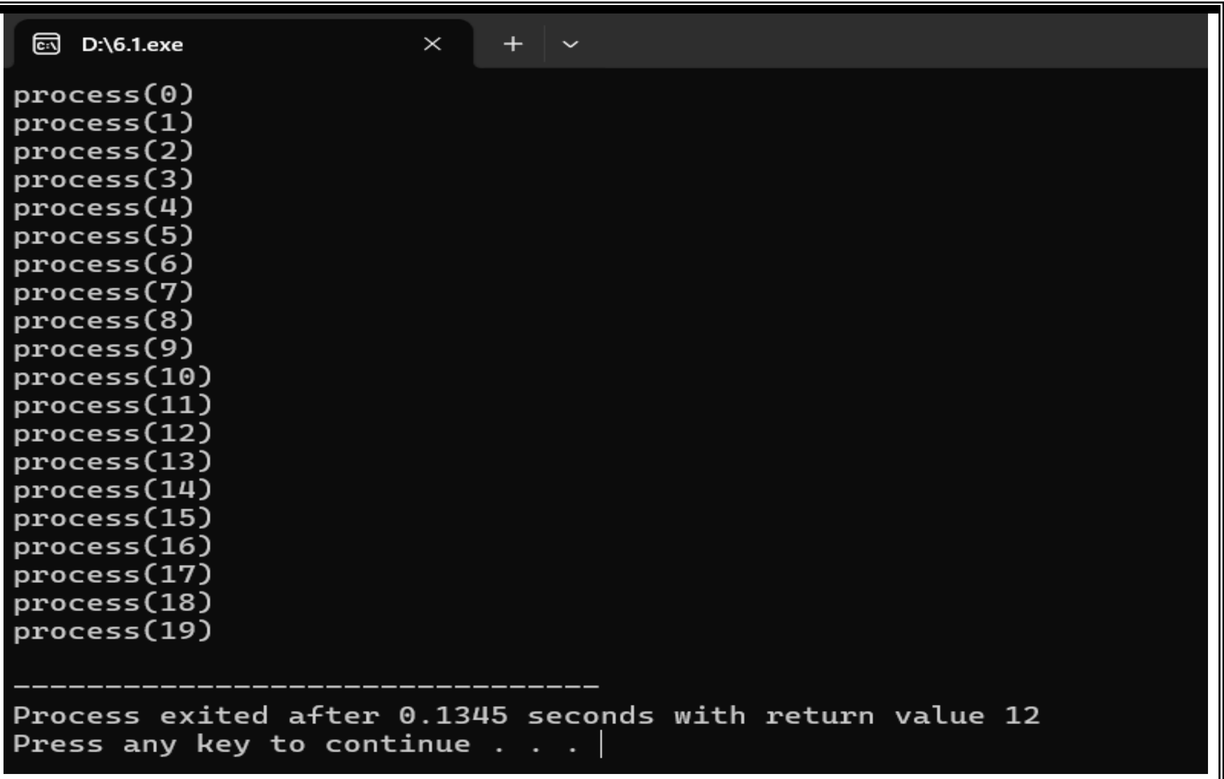
## Week-6

## 6.1)

## Program:

```
#include<stdio.h>
#define TOGETHER 8
int main(){
int i = 0;
int entries = 20,repeat,left;
repeat = (entries/TOGETHER);
left   = (entries%TOGETHER);
while (repeat--){
printf("process(%d)\n", i    );
printf("process(%d)\n", i + 1);
printf("process(%d)\n", i + 2);
printf("process(%d)\n", i + 3);
printf("process(%d)\n", i + 4);
printf("process(%d)\n", i + 5);
printf("process(%d)\n", i + 6);
printf("process(%d)\n", i + 7);
i += TOGETHER;}
switch (left){
case 7 : printf("process(%d)\n", i + (left-7));
case 6 : printf("process(%d)\n", i + (left-6));
case 5 : printf("process(%d)\n", i + (left-5));
case 4 : printf("process(%d)\n", i + (left-4));
case 3 : printf("process(%d)\n", i + (left-3));
case 2 : printf("process(%d)\n", i + (left-2));
case 1 : printf("process(%d)\n", i + (left-1));
case 0 : ;}}
```

## Output:



```
process(0)
process(1)
process(2)
process(3)
process(4)
process(5)
process(6)
process(7)
process(8)
process(9)
process(10)
process(11)
process(12)
process(13)
process(14)
process(15)
process(16)
process(17)
process(18)
process(19)

--------------------------------
Process exited after 0.1345 seconds with return value 12
Press any key to continue . . .
```

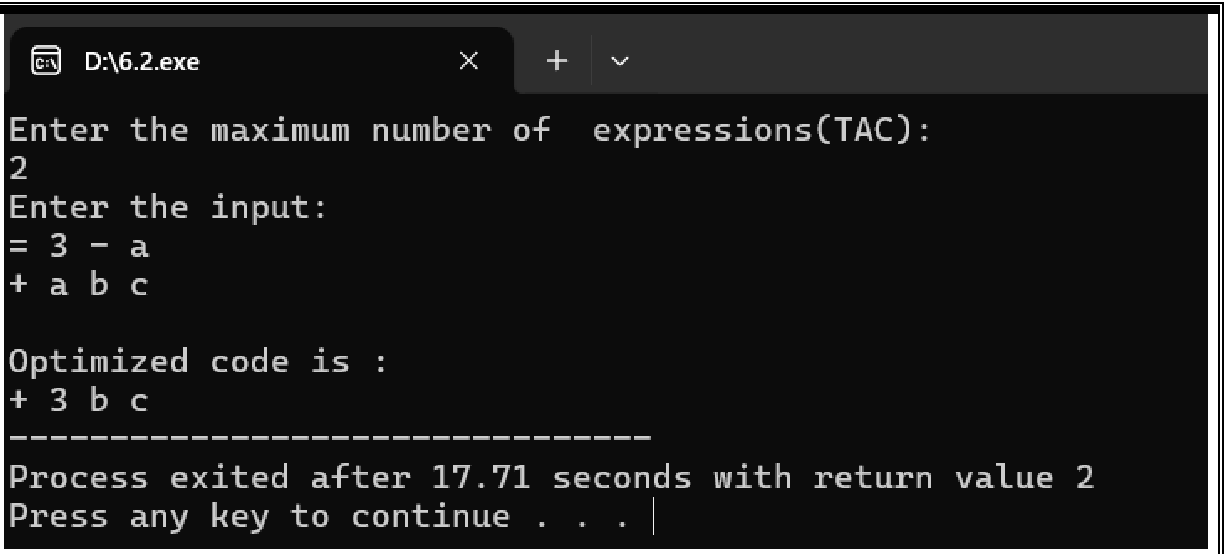
**6.2)**

**Program:**

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
void input();
void output();
void change(int p, char * res);
void constant();
struct expr {
   char op[2], op1[5], op2[5], res[5];
   int flag;}
arr[10];
int n;
void main() {
   input();
   constant();
   output();}
void input() {
   int i;
   printf("Enter the maximum number of  expressions(TAC):\n");
   scanf("%d", & n);
   printf("Enter the input: \n");
   for (i = 0; i < n; i++) {
       scanf("%s", arr[i].op);
       scanf("%s", arr[i].op1);
       scanf("%s", arr[i].op2);
       scanf("%s", arr[i].res);
       arr[i].flag = 0;  }}
void constant() {
   int i,op1, op2, res;
   char op, res1[5];
   for (i = 0; i < n; i++) {
       if (isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op, "=") == 0) /*if both
digits, store them in variables*/ {
            op1 = atoi(arr[i].op1);
            op2 = atoi(arr[i].op2);
            op = arr[i].op[0];
            switch (op) {
            case '+':
                res = op1 + op2;
                break;
            case '-':
                res = op1 - op2;
                break;
            case '*':
```

```
            res = op1 * op2;
            break;
          case '/':
            res = op1 / op2;
            break;
          case '=':
            res = op1;
            break;          }
          sprintf(res1, "%d", res);
          arr[i].flag = 1;
          change(i, res1);}}}}
void output() {
  int i = 0;
  printf("\nOptimized code is : ");
  for (i = 0; i < n; i++) {
    if (!arr[i].flag) {
      printf("\n%s %s %s %s", arr[i].op, arr[i].op1, arr[i].op2, arr[i].res);}
  }
}
void change(int p, char * res) {
  int i;
  for (i = p + 1; i < n; i++) {
    if (strcmp(arr[p].res, arr[i].op1) == 0)
      strcpy(arr[i].op1, res);
    else if (strcmp(arr[p].res, arr[i].op2) == 0)
      strcpy(arr[i].op2, res);
  }
}
```

**Output:**



D:\6.2.exe

```
Enter the maximum number of  expressions(TAC):
2
Enter the input:
= 3 - a
+ a b c

Optimized code is :
+ 3 b c
--------------------------------
Process exited after 17.71 seconds with return value 2
Press any key to continue . . .
```

**13.)**

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

**Procedure:**

1. **Uses a predefined set of valid operators like +, -, *, /, %, =, ==, !=, <, >, <=, >=, &&,**

**||, !, &, |, ^, <<, >>**

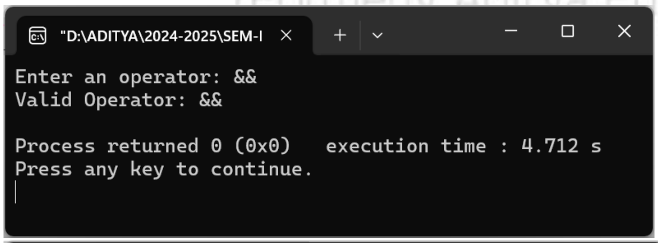
2. **Reads an operator from the user and verifies whether it is valid.**
3. **Outputs whether the input is a valid operator or not.**

**Program:**

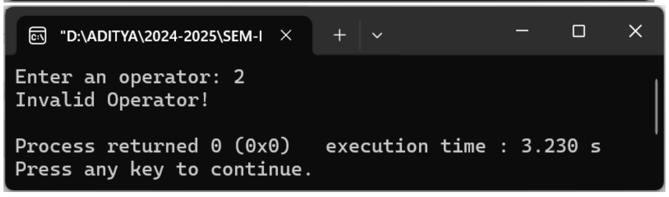
```
#include <stdio.h> #include <string.h>
const char *operators[] = {
"+", "-", "*", "/", "%", "=", "==", "!=", "<", ">", "<=", ">=", "&&",
"||", "!", "&", "|", "^", "<<", ">>"
};
int isValidOperator(char *input) {
int numOperators = sizeof(operators) / sizeof(operators[0]);
for (int i = 0; i < numOperators; i++) { if (strcmp(input, operators[i]) == 0)
{
```

```
return 1;
}
}
return 0;
}
int main() {
char input[10];
printf("Enter an operator: "); scanf("%s", input);
if (isValidOperator(input)) { printf("Valid Operator: %s\n", input);
} else {
printf("Invalid Operator!\n");
}
return 0;
}
```

**Actual Input and Output:**



```
Enter an operator: &&
Valid Operator: &&

Process returned 0 (0x0)   execution time : 4.712 s
Press any key to continue.
```



```
Enter an operator: 2
Invalid Operator!

Process returned 0 (0x0)   execution time : 3.230 s
Press any key to continue.
```

14.)

**AIM: Write a C program to identify whether a given line is a comment or not?**

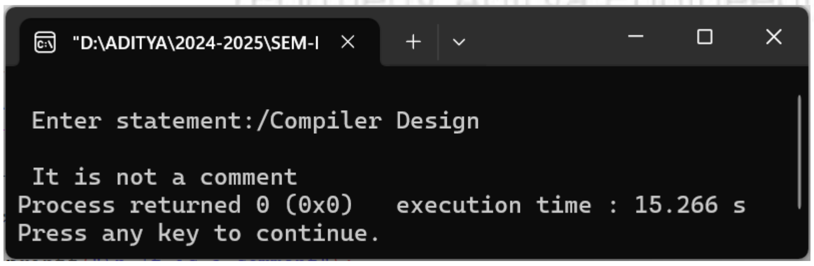
**Procedure:**

1. The program takes a line of input from the user.
2. It checks if the line starts with // (single-line comment).
3. It checks if the line starts with /* and also contains */ (multi-line comment).
4. If neither condition is met, it prints that the input is not a comment.

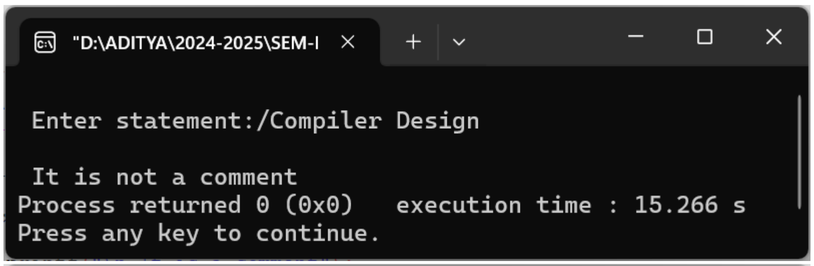
**Program:**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char com[30]; int i=2,a=0,n;
printf("\n Enter statement:"); gets(com);
n=strlen(com); if(com[0]=='/')
{
if(com[1]=='/')
printf("\n It is a comment"); else if(com[1]=='*')
{
for(i=2;i<n;i++)
{
if(com[i]=='*'&&com[i+1]=='/')
{
 printf("\n It is a comment");
a=1;
break;
}
else
continue;
 }


if(a==0)
 printf("\n It is not a comment");
}
else
 printf("\n It is not a comment");
}
else
  printf("\n It is not a comment");
}
```

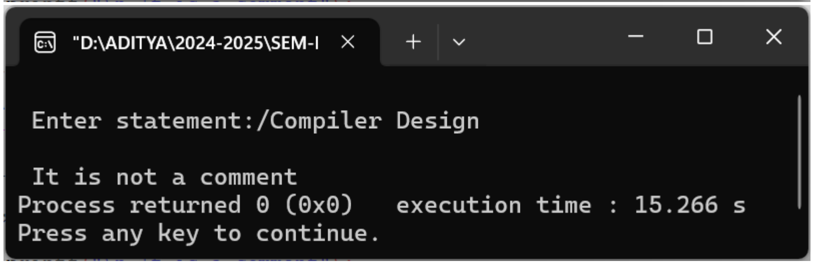
**Outp**



```
"D:\ADITYA\2024-2025\SEM-I

 Enter statement:/Compiler Design

 It is not a comment
Process returned 0 (0x0)   execution time : 15.266 s
Press any key to continue.
```



```
"D:\ADITYA\2024-2025\SEM-I

 Enter statement:/Compiler Design

 It is not a comment
Process returned 0 (0x0)   execution time : 15.266 s
Press any key to continue.
```



```
"D:\ADITYA\2024-2025\SEM-I

 Enter statement:/Compiler Design

 It is not a comment
Process returned 0 (0x0)   execution time : 15.266 s
Press any key to continue.
```