Experiment-1:

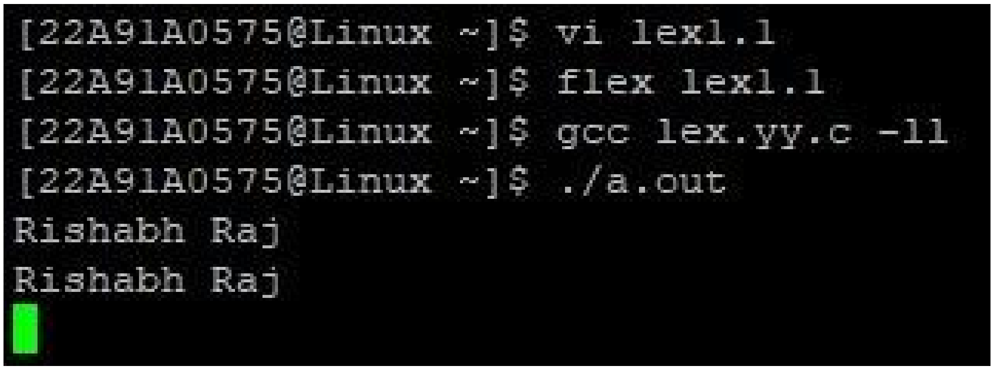
Lexical analysis using lex tool

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

Program:

```
 %%
. ECHO;
%%
int yywrap(void) {
        return 1;
}
int main(void) {
        yylex();
        return 0;
}
```

Output:



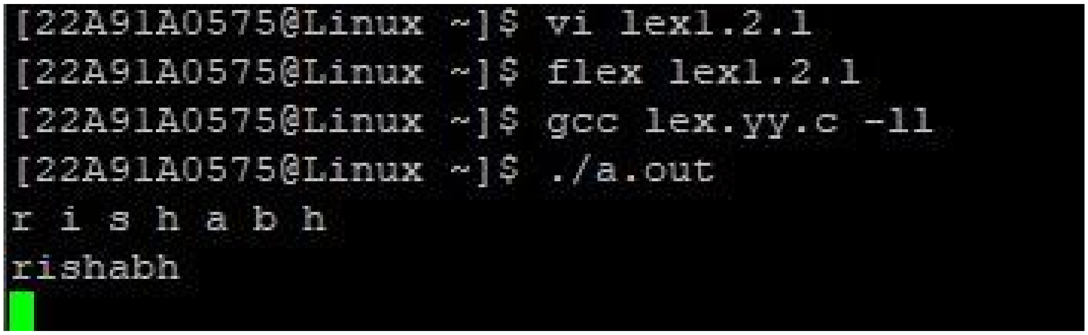
```
[22A91A0575@Linux ~]$ vi lex1.l
[22A91A0575@Linux ~]$ flex lex1.l
[22A91A0575@Linux ~]$ gcc lex.yy.c -ll
[22A91A0575@Linux ~]$ ./a.out
Rishabh Raj
Rishabh Raj
```

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

Program:

```
%{
#include<stdio.h>
%}
%%
[\n\t ' '] {};
%%
main()
{
        yyin=fopen("myfile.txt","r");
        yylex();
}
int yywrap()
{
        return1;
}
```

Output:



```
[22A91A0575@Linux ~]$ vi lex1.2.1
[22A91A0575@Linux ~]$ flex lex1.2.1
[22A91A0575@Linux ~]$ gcc lex.yy.c -ll
[22A91A0575@Linux ~]$ ./a.out
r i s h a b h
rishabh
```

Experiment-2

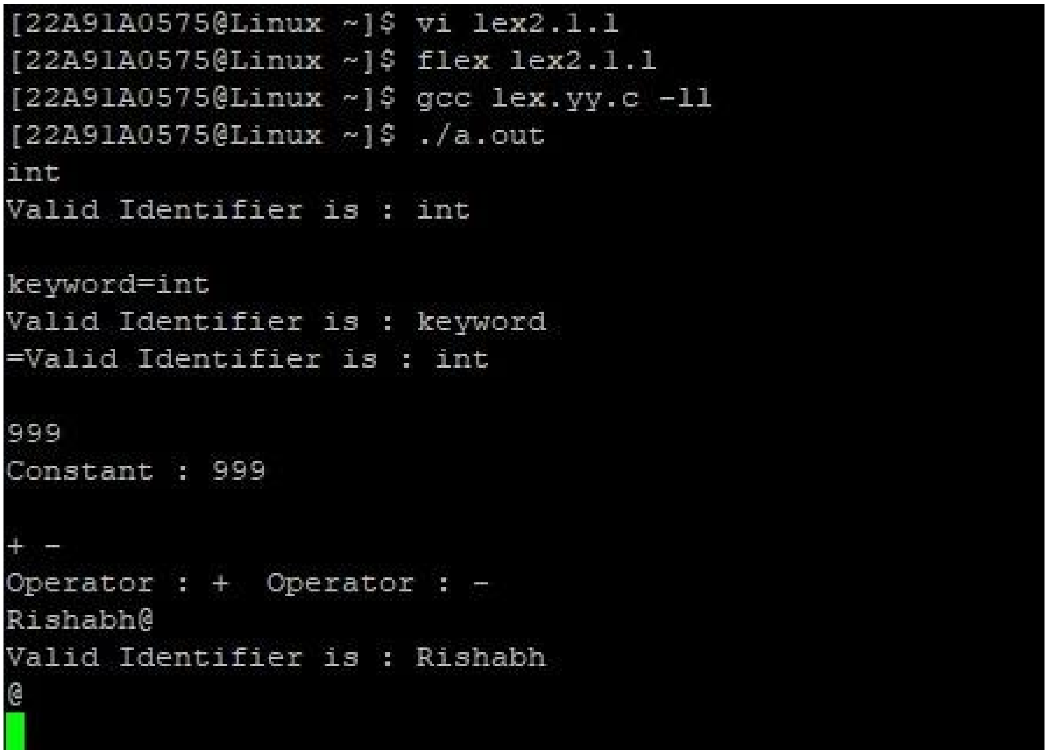
Lexical analysis using lex tool

2.1) Write a lex program to identify the patterns in the input file.

Program:

```
%{
#include<stdio.h>
%}
%%
["int""char""for""if""while""then""return""do"] {printf("keyword : %s\n");}
[*%+\-] {printf("Operator : %s ", yytext);}
[(){};] {printf("Special Character: %s\n", yytext);}
[0-9]+ {printf("Constant : %s\n", yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {printf("Valid Identifier is : %s\n", yytext);}
^[^a-zA-Z_] {printf("Invalid Indentifier \n");}
%%
```

Output:



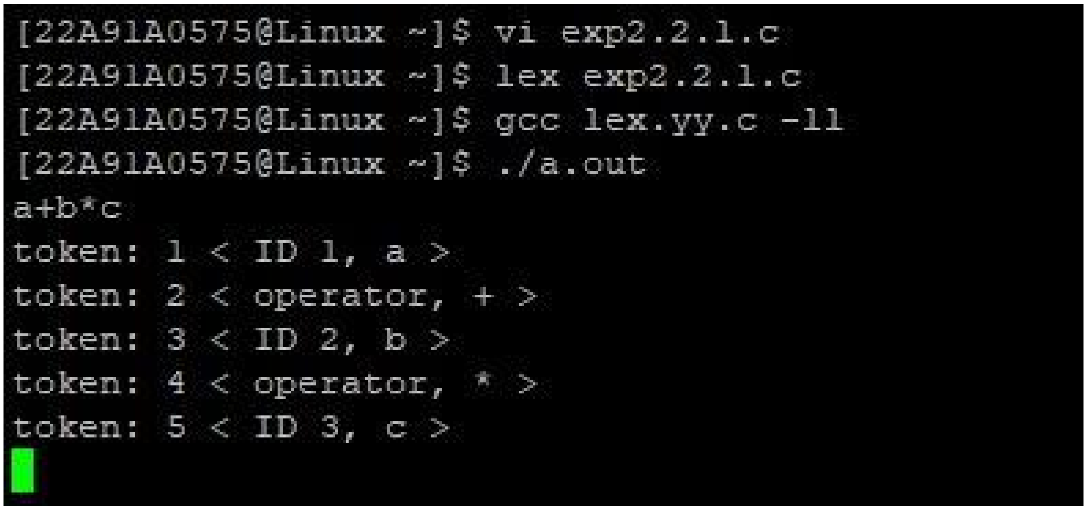
```
[22A91A0575@Linux ~]$ vi lex2.1.l
[22A91A0575@Linux ~]$ flex lex2.1.l
[22A91A0575@Linux ~]$ gcc lex.yy.c -ll
[22A91A0575@Linux ~]$ ./a.out
int
Valid Identifier is : int

keyword=int
Valid Identifier is : keyword
=Valid Identifier is : int

999
Constant : 999

+ -
Operator : +  Operator : -
Rishabh@
Valid Identifier is : Rishabh
@
```

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|while {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:



```
[22A91A0575@Linux ~]$ vi exp2.2.1.c
[22A91A0575@Linux ~]$ lex exp2.2.1.c
[22A91A0575@Linux ~]$ gcc lex.yy.c -ll
[22A91A0575@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 >
```

Experiment-3

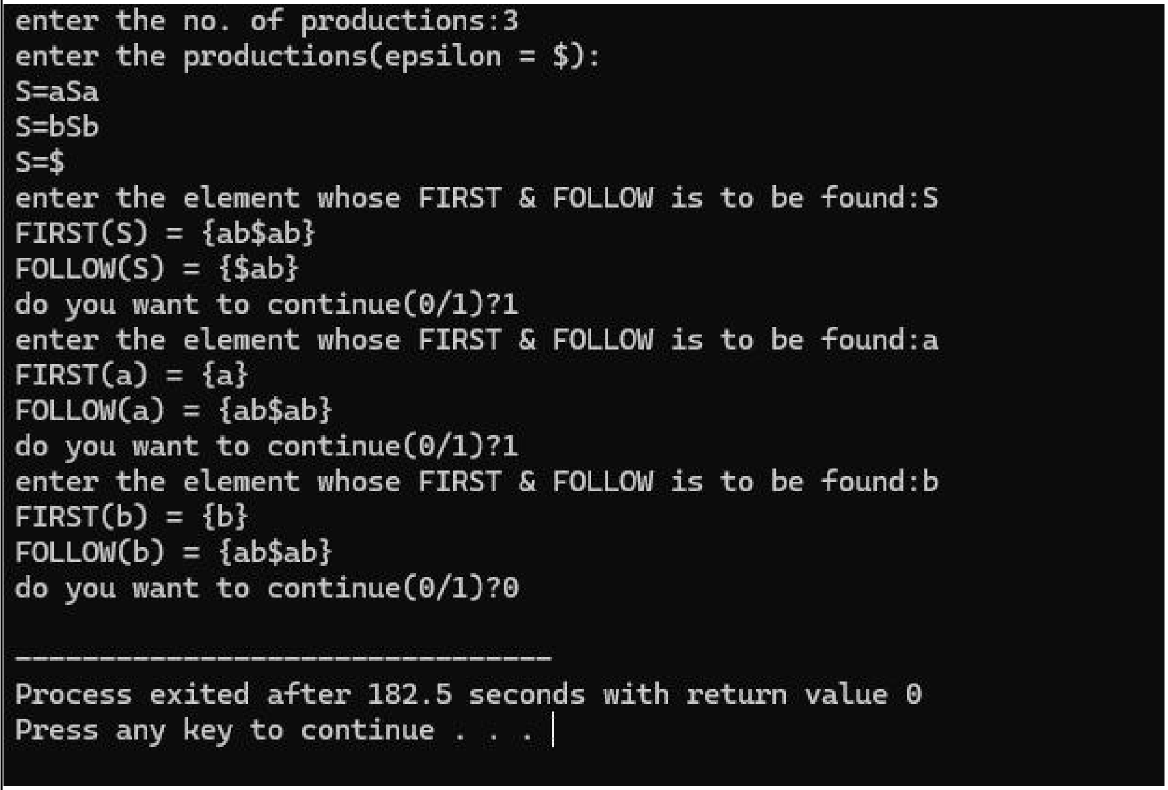
First and Follow

3.1) Simulate First and Follow of a Grammar

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int n, m = 0, p, i = 0, j = 0;
char a[10][10], f[10];
void follow(char c){
    if(a[0][0] == c)
    f[m++] = '$';
    for(i = 0;i<n;i++){
    for(j = 2;j<strlen(a[i]);j++){
        if(a[i][j] == c){
        if(a[i][j+1] != '\0') first(a[i][j+1]);
        if(a[i][j+1] == '\0' && c != a[i][0]) follow(a[i][0]);
        }
    }
    }
}
void first(char c){
    int k;
    if(!(isupper(c))) f[m++] = c;
    for(k = 0;k<n;k++){
    if(a[k][0] == c){
        if(a[k][2] == '$') follow(a[i][0]);
        else if(islower(a[k][2])) f[m++] = a[k][2];
        else first(a[k][2]);
    }
    }
}
int main(){
    int i, z;
    char c, ch;
    printf("enter the no. of productions:");
    scanf("%d", &n);
    printf("enter the productions(epsilon = $):\n");
    for(i = 0; i < n; i++) scanf("%s%c", a[i], &ch);
do{
        m = 0;
        printf("enter the element whose FIRST & FOLLOW is to be found:");
        scanf("%c", &c);
        first(c);
        printf("FIRST(%c) = {", c);
        for(i = 0;i<m;i++)
         printf("%c", f[i]);
        printf("}\n");
        follow(c);
        printf("FOLLOW(%c) = {", c);
        for(;i<m;i++) printf("%c", f[i]);
        printf("}\n");
        printf("do you want to continue(0/1)?");
        scanf("%d%c", &z, &ch);
        }while(z == 1);
}
```

Output:



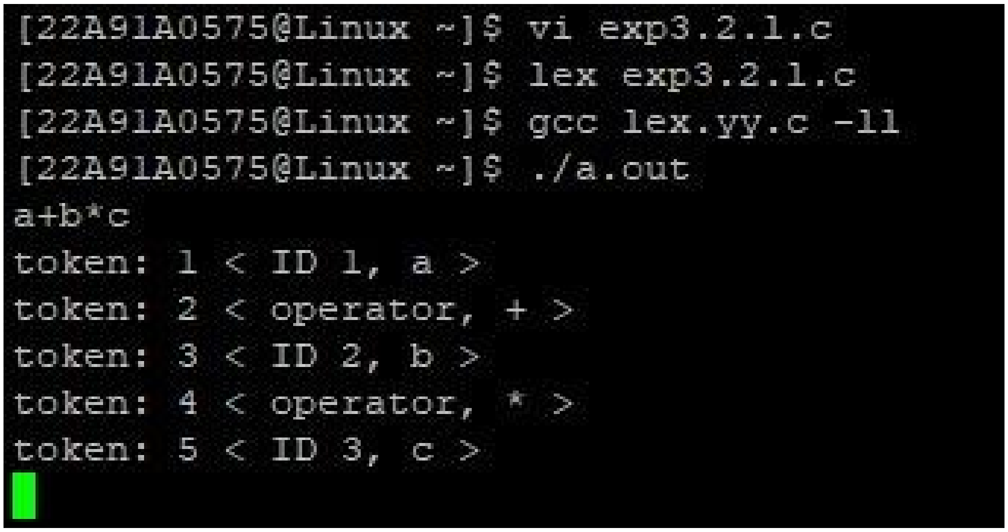
```
enter the no. of productions:3
enter the productions(epsilon = $):
S=aSa
S=bSb
S=$
enter the element whose FIRST & FOLLOW is to be found:S
FIRST(S) = {ab$ab}
FOLLOW(S) = {$ab}
do you want to continue(0/1)?1
enter the element whose FIRST & FOLLOW is to be found:a
FIRST(a) = {a}
FOLLOW(a) = {ab$ab}
do you want to continue(0/1)?1
enter the element whose FIRST & FOLLOW is to be found:b
FIRST(b) = {b}
FOLLOW(b) = {ab$ab}
do you want to continue(0/1)?0

--------------------------------
Process exited after 182.5 seconds with return value 0
Press any key to continue . . .
```

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

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|g
oto|if|int|long|register|return|short|signed|sizeof|static|struct|switch|typedef|union|
unsigned|void|volatile|while {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:



```
[22A91A0575@Linux ~]$ vi exp3.2.1.c
[22A91A0575@Linux ~]$ lex exp3.2.1.c
[22A91A0575@Linux ~]$ gcc lex.yy.c -ll
[22A91A0575@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 >
```

Experiment-4

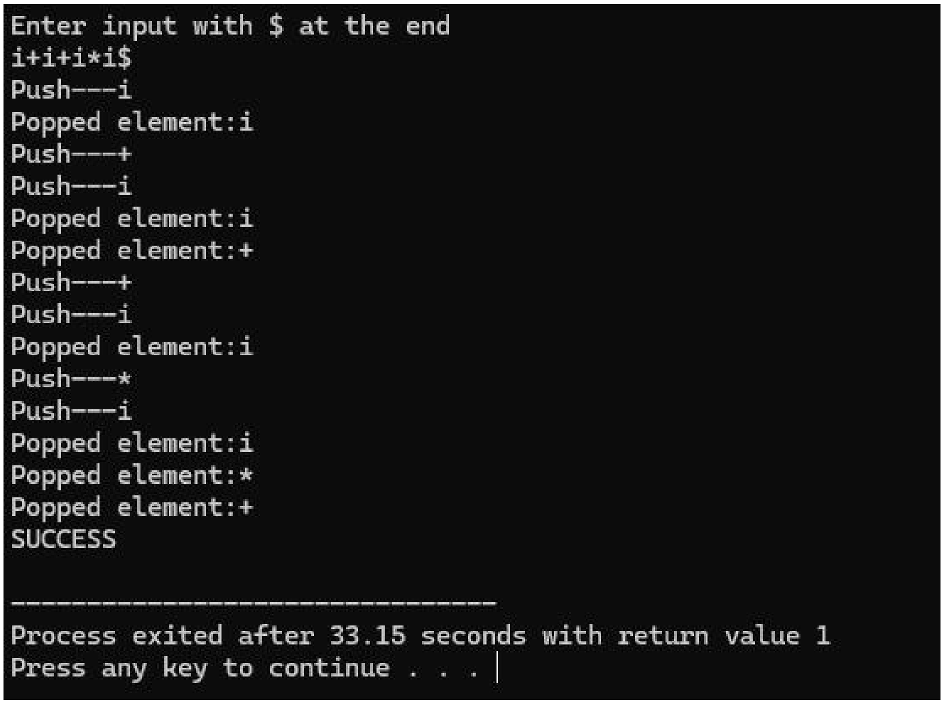
Top-Down Parsing

4.1) Develop an operator precedence parser for a given language.

Program:

```
#include<stdio.h>
#include<string.h>
char stack[20],temp;
int top=-1;
void push(char item){
        if(top>=20){
        printf("STACK OVERFLOW");
        return;
        }
        stack[++top]=item;
}
char pop(){
        if(top<=-1){
        printf("STACK UNDERFLOW");
        return;
        }
        char c;
        c=stack[top--];
        printf("Popped element:%c\n",c);
        return c;
}
char TOS(){
  return stack[top];
}
int convert(char item){
        switch(item){
        case 'i':return 0;
        case '+':return 1;
        case '*':return 2;
        case '$':return 3;
        }
}
int main(){
        char pt[4][4]={
{'-','>','>','>'},
{'<','>','<','>'},
 {'<','>','>','>'},
 {'<','<','<','1'}};
 char input[20];
 int lkh=0;
 printf("Enter input with $ at the end\n");
 scanf("%s",input);
 push('$');
 while(lkh<=strlen(input)){
        if(TOS()=='$'&&input[lkh]=='$'){
        printf("SUCCESS\n");
        return 1;
        }
        else if(pt[convert(TOS())][convert(input[lkh])]=='<'){
                push(input[lkh]);
                printf("Push---%c\n",input[lkh]);
                lkh++;
        }
        else    pop();
        }
        return 0;
}
```

Output:



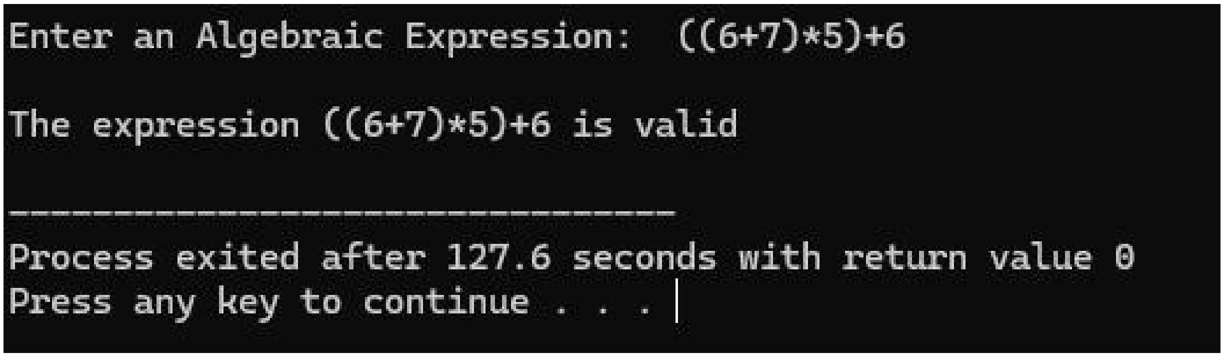
```
Enter input with $ at the end
i+i+i*i$
Push---i
Popped element:i
Push---+
Push---i
Popped element:i
Popped element:+
Push---+
Push---i
Popped element:i
Push---*
Push---i
Popped element:i
Popped element:*
Popped element:+
SUCCESS

--------------------------------
Process exited after 33.15 seconds with return value 1
Press any key to continue . . .
```

4.2) Construct a recursive descent parser for an expression.

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
void Tp();
void Ep();
void E();
void T();
void check();
int count,flag;
char expr[10];
int main(){
        count=0;
        flag=0;
        printf("\nEnter an Algebraic Expression:\t");
        scanf("%s",expr);
        E();
        if((strlen(expr)==count)&&(flag==0))
        printf("\nThe expression %s is valid\n",expr);
        else
        printf("\nThe expression %s is invalid\n",expr);
        return 0;

}
void E()
{
        T();
        Ep();
}

void T()
{
        check();
        Tp();
}
void Tp(){
        if(expr[count]=='*'){
        count++;
        check();
}
        Tp();
        }
}
void check(){
        if(isalnum(expr[count]))
        count++;
        else if(expr[count]=='('){
        count++;
        E();
        if(expr[count]==')')        count++;
        else    flag=1;
        }
        else    flag=1;
}
void Ep(){
        if(expr[count]=='+'){
        count++;
        T();
        Ep();
        }
}
```

Output:



```
Enter an Algebraic Expression:  ((6+7)*5)+6

The expression ((6+7)*5)+6 is valid

--------------------------------
Process exited after 127.6 seconds with return value 0
Press any key to continue . . .
```

Experiment-5

Bottom-up Parsing

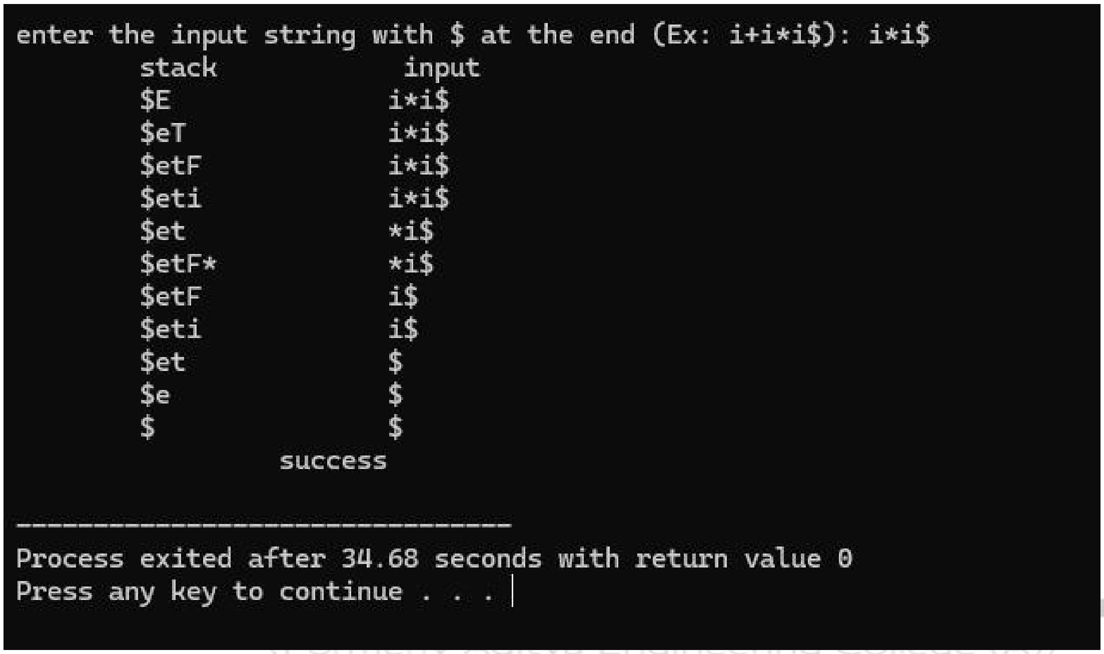
5.1) Construct an LL(1) parser for an expression.

Program:

```
#include<stdio.h>
#include<string.h>
int stack[20],top=-1;
void push(int item){
        if(top>=20){
                printf("stack overflow");
                return;
        }
        stack[++top]=item;
}
int pop(){
        int ch;
        if(top<=-1){
                printf("underflow");
                return;
        }
        ch=stack[top--];
        return ch;
}
char convert(int item){
        char ch;
        switch(item){
                case 0:return('E');
                case 1:return('e');
                case 2:return('T');
                case 3:return('t');
                case 4:return('F');
                case 5:return('i');
                case 6:return('+');
                case 7:return('*');
                case 8:return('(');
                case 9:return(')');
                case 10:return('$');
        }
}
```

```
void main(){
        int m[10][10],i,j,k;
        char ips[20];
        int ip[10],a,b,t;
        m[0][0]=m[0][3]=21;
        m[1][1]=621;
        m[1][4]=m[1][5]=-2;
        m[2][0]=m[2][3]=43;
        m[3][1]=m[3][4]=m[3][5]=-2;
        m[3][2]=743;
        m[4][0]=5;
        m[4][3]=809;
printf("\nenter the input string with $ at the end (Ex: i+i*i$): ");
scanf("%s",ips);
for(i=0;i<strlen(ips);i++){
switch(ips[i]){
 case 'E':k=0;break;
 case 'e':k=1;break;
 case 'T':k=2;break;
 case 't':k=3;break;
 case 'F':k=4;break;
 case 'i':k=5;break;
 case '+':k=6;break;
 case '*':k=7;break;
 case '(':k=8;break;
 case ')':k=9;break;
 case '$':k=10;break;
 }
 ip[i]=k;
 }
 ip[i]=-1;
 push(10);
 push(0);
 i=0;
 printf("\tstack\t\t input \n");
 while(1){
 printf("\t");
 for(j=0;j<=top;j++)
 printf("%c",convert(stack[j]));
 printf("\t\t");
for(k=i;ip[k]!=-1;k++)
printf("%c",convert(ip[k]));
  printf("\n");
  if(stack[top]==ip[i]){
  if(ip[i]==10){
  printf("\t\t success\n");
  return;
}
else{
top--;
i++;
        }
        }
        else if(stack[top]<=4&&stack[top]>=0){
                a=stack[top];
                b=ip[i]-5;
                t=m[a][b];
                top--;
                while(t>0){
                push(t%10);
                t=t/10;
                        }
                }
                else{
                        printf("error\n");
                        return;
                }
        }
}
```

Output:



```
enter the input string with $ at the end (Ex: i+i*i$): i*i$
        stack            input
        $E              i*i$
        $eT             i*i$
        $etF            i*i$
        $eti            i*i$
        $et             *i$
        $etF*           *i$
        $etF            i$
        $eti            i$
        $et             $
        $e              $
        $               $
                 success

--------------------------------
Process exited after 34.68 seconds with return value 0
Press any key to continue . . .
```

5.2) Design a LALR bottom up parser for the given language.

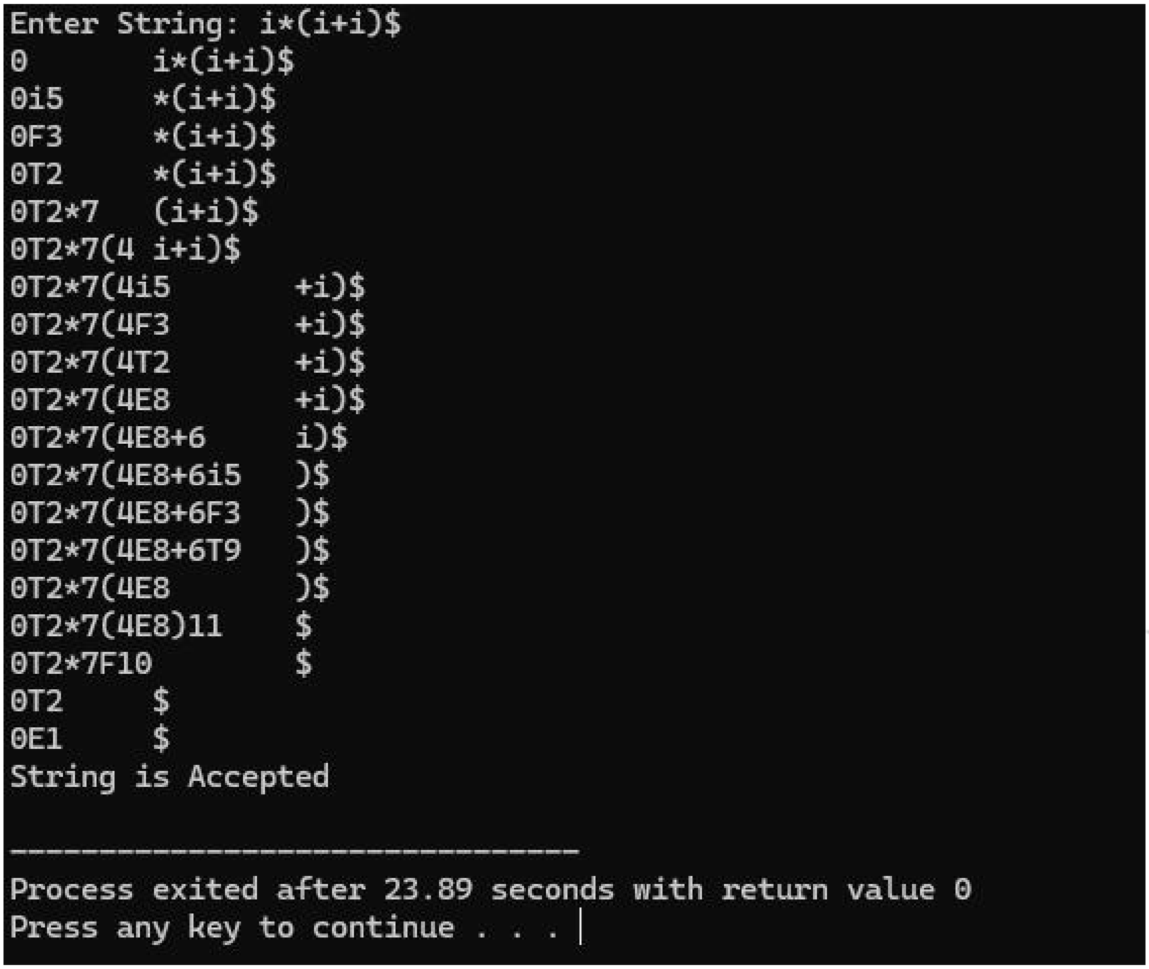
Program:

```
#include<stdio.h>
#include<string.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");
        for(l=p;ptr[l];l++)
                printf("%c",ptr[l]);
        printf("\n");
}
int main(){
        char t1[20][20],pr[20][20],xy;
        int inp[20],t2[20][20],gt[20][20],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: ");
         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");
         }
         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");
 return 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");
         }
}
```

Output:



```
Enter String: i*(i+i)$
0       i*(i+i)$
0i5     *(i+i)$
0F3     *(i+i)$
0T2     *(i+i)$
0T2*7   (i+i)$
0T2*7(4 i+i)$
0T2*7(4i5       +i)$
0T2*7(4F3       +i)$
0T2*7(4T2       +i)$
0T2*7(4E8       +i)$
0T2*7(4E8+6     i)$
0T2*7(4E8+6i5   )$
0T2*7(4E8+6F3   )$
0T2*7(4E8+6T9   )$
0T2*7(4E8       )$
0T2*7(4E8)11    $
0T2*7F10        $
0T2     $
0E1     $
String is Accepted

--------------------------------
Process exited after 23.89 seconds with return value 0
Press any key to continue . . .
```

Experiment-6

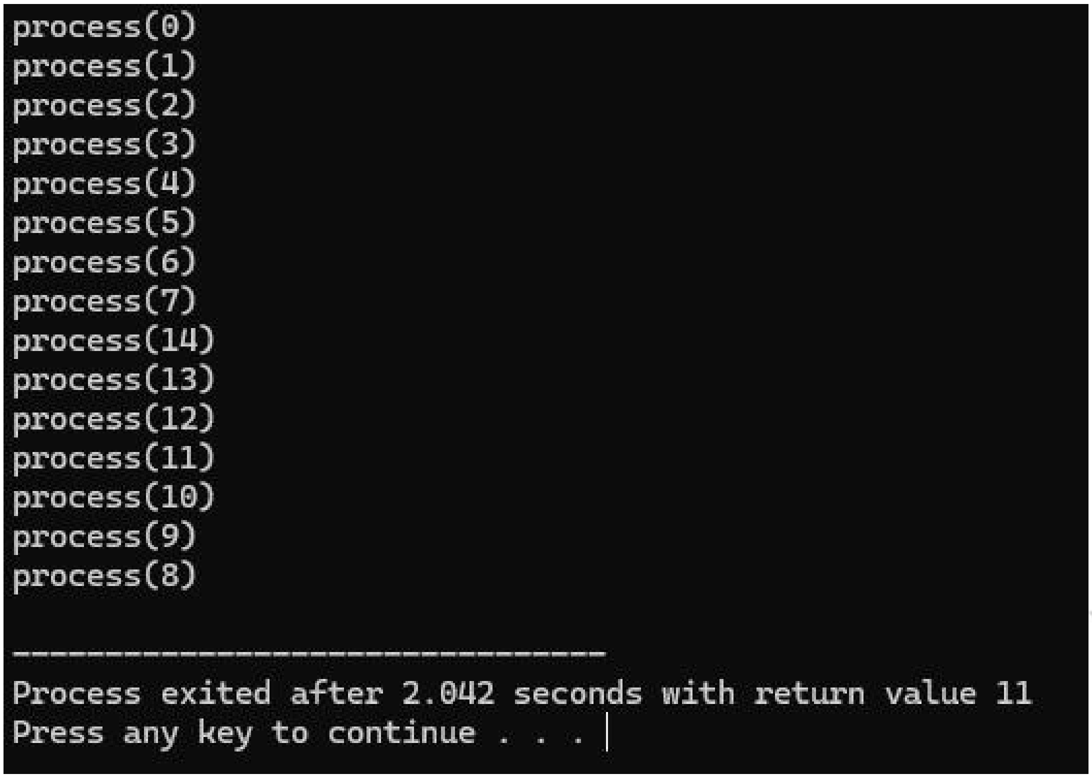
Optimization Phase

6.1) Write a program to perform loop unrolling.

Program:

```
#include<stdio.h>
#define TOGETHER (8)
int main(void){

        int i = 0,entries = 15,repeat,left = 0;
        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 + 6);
                case 6 : printf("process(%d)\n", i + 5);
                case 5 : printf("process(%d)\n", i + 4);
                case 4 : printf("process(%d)\n", i + 3);
                case 3 : printf("process(%d)\n", i + 2);
                case 2 : printf("process(%d)\n", i + 1);
                case 1 : printf("process(%d)\n", i);
                case 0 : ;
        }

}
```

Output:



```
process(0)
process(1)
process(2)
process(3)
process(4)
process(5)
process(6)
process(7)
process(14)
process(13)
process(12)
process(11)
process(10)
process(9)
process(8)

--------------------------------
Process exited after 2.042 seconds with return value 11
Press any key to continue . . .
```

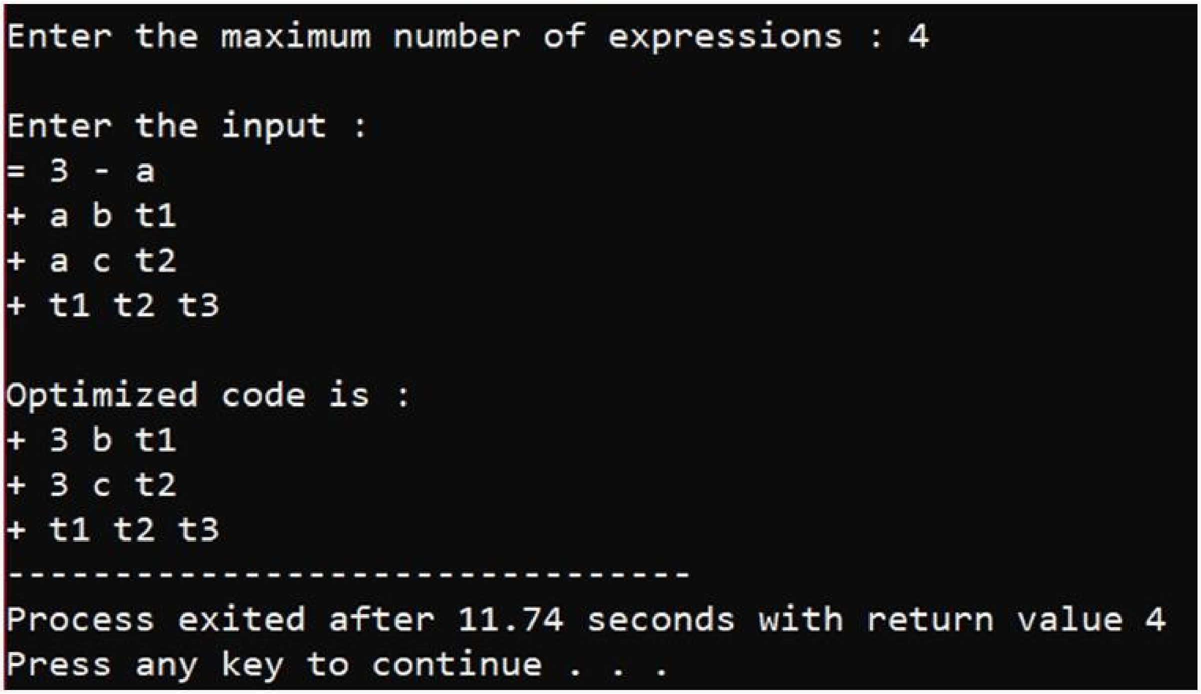
6.2) Write a program for constant propagation.

Program:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.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("\n\nEnter the maximum number of expressions : ");
    scanf("%d",&n);
    printf("\nEnter 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;
    int 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){
```

```
 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:



```
Enter the maximum number of expressions : 4

Enter the input :
= 3 - a
+ a b t1
+ a c t2
+ t1 t2 t3

Optimized code is :
+ 3 b t1
+ 3 c t2
+ t1 t2 t3
--------------------------------
Process exited after 11.74 seconds with return value 4
Press any key to continue . . .
```