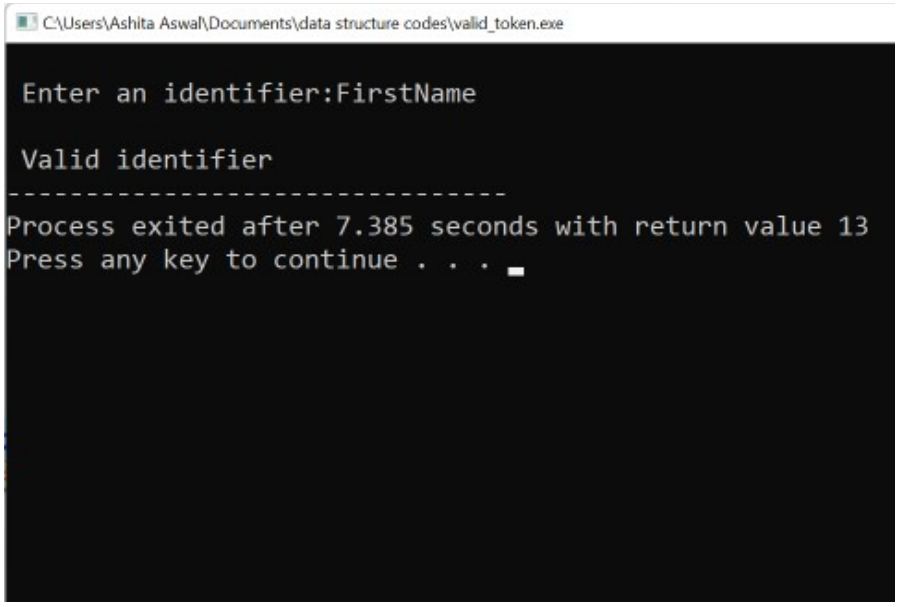
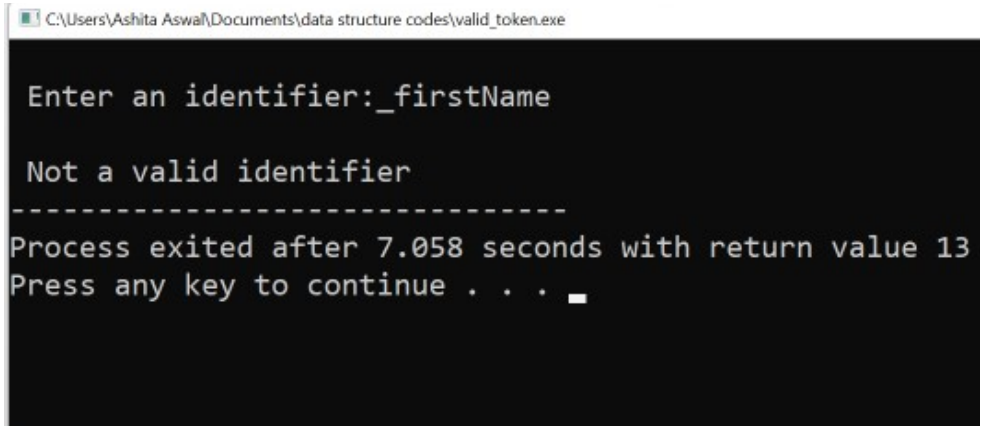
Program 1- Write a program in C to recognize valid identifiers.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
char a[10];
int flag, i=1;
printf("\n Enter an identifier:");
gets(a);
if(isalpha(a[0]))
flag=1;
else
printf("\n Not a valid identifier");
while(a[i]!='\0')
{
if(!isdigit(a[i])&&!isalpha(a[i]))
{
flag=0;
break;
}
i++;
}
if(flag==1)
printf("\n Valid identifier");
getch();
}
```

Output-


C:\Users\Ashita Aswal\Documents\data structure codes\valid_token.exe
Enter an identifier:FirstName
Valid identifier
Process exited after 7.385 seconds with return value 13
Press any key to continue . . .

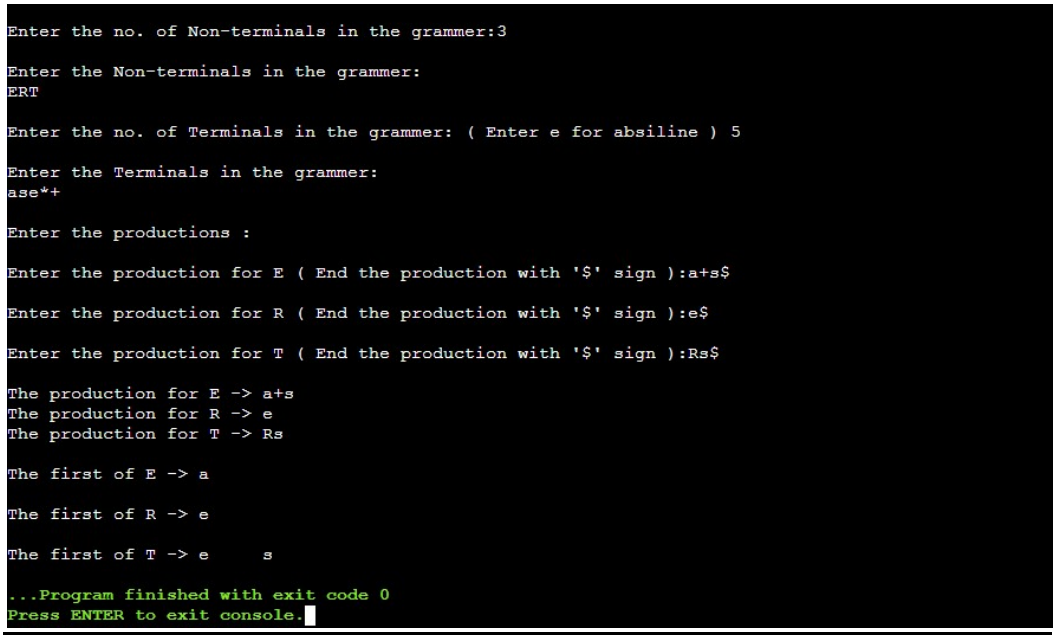

C:\Users\Ashita Aswal\Documents\data structure codes\valid_token.exe
Enter an identifier:_firstName
Not a valid identifier
Process exited after 7.058 seconds with return value 13
Press any key to continue . . .

Program 2- Write a program in C to find FIRST() from a Grammar.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char t[5],nt[10],p[5][5],first[5][5],temp;
int i,j,not,nont,k=0,f=0;
clrscr();
printf("\nEnter the no. of Non-terminals in the grammer:");
scanf("%d",&nont);
printf("\nEnter the Non-terminals in the grammer:\n");
for(i=0;i<nont;i++)
{
scanf("\n%c",&nt[i]);
}
printf("\nEnter the no. of Terminals in the grammer: ( Enter e for absiline ) ");
scanf("%d",&not);
printf("\nEnter the Terminals in the grammer:\n");
for(i=0;i<not||t[i]=='$';i++)
{
scanf("\n%c",&t[i]);
}
for(i=0;i<nont;i++)
{
p[i][0]=nt[i];
first[i][0]=nt[i];
}
printf("\nEnter the productions :\n");
for(i=0;i<nont;i++)
{
scanf("%c",&temp);
printf("\nEnter the production for %c ( End the production with '$' sign )
:",p[i][0]);
for(j=0;p[i][j]!='$';)
{
j+=1;
scanf("%c",&p[i][j]);
}
}
for(i=0;i<nont;i++)
{
printf("\nThe production for %c -> ",p[i][0]);
for(j=1;p[i][j]!='$';j++)
{
printf("%c",p[i][j]);
}
```

```
}
for(i=0;i<nont;i++)
{
f=0;
for(j=1;p[i][j]!='$';j++)
{
for(k=0;k<not;k++)
{
if(f==1)
break;
if(p[i][j]==t[k])
{
first[i][j]=t[k];
first[i][j+1]='$';
f=1;
break;
}
else if(p[i][j]==nt[k])
{
first[i][j]=first[k][j];
if(first[i][j]=='e')
continue;
first[i][j+1]='$';
f=1;
break;
}
}
}
}
for(i=0;i<nont;i++)
{
printf("\n\nThe first of %c -> ",first[i][0]);
for(j=1;first[i][j]!='$';j++)
{
printf("%c\t",first[i][j]);
}
}
getch();
}
```

Output-


Enter the no. of Non-terminals in the grammer:3
Enter the Non-terminals in the grammer:
ERT
Enter the no. of Terminals in the grammer: (Enter e for absiline) 5
Enter the Terminals in the grammer:
ase*+
Enter the productions :
Enter the production for E (End the production with '$' sign):a+s$
Enter the production for R (End the production with '$' sign):e$
Enter the production for T (End the production with '$' sign):Rs$
The production for E -> a+s
The production for R -> e
The production for T -> Rs
The first of E -> a
The first of R -> e
The first of T -> e s
...Program finished with exit code 0
Press ENTER to exit console.

Program 3- Write a program in C to find FOLLOW() from a Grammar.

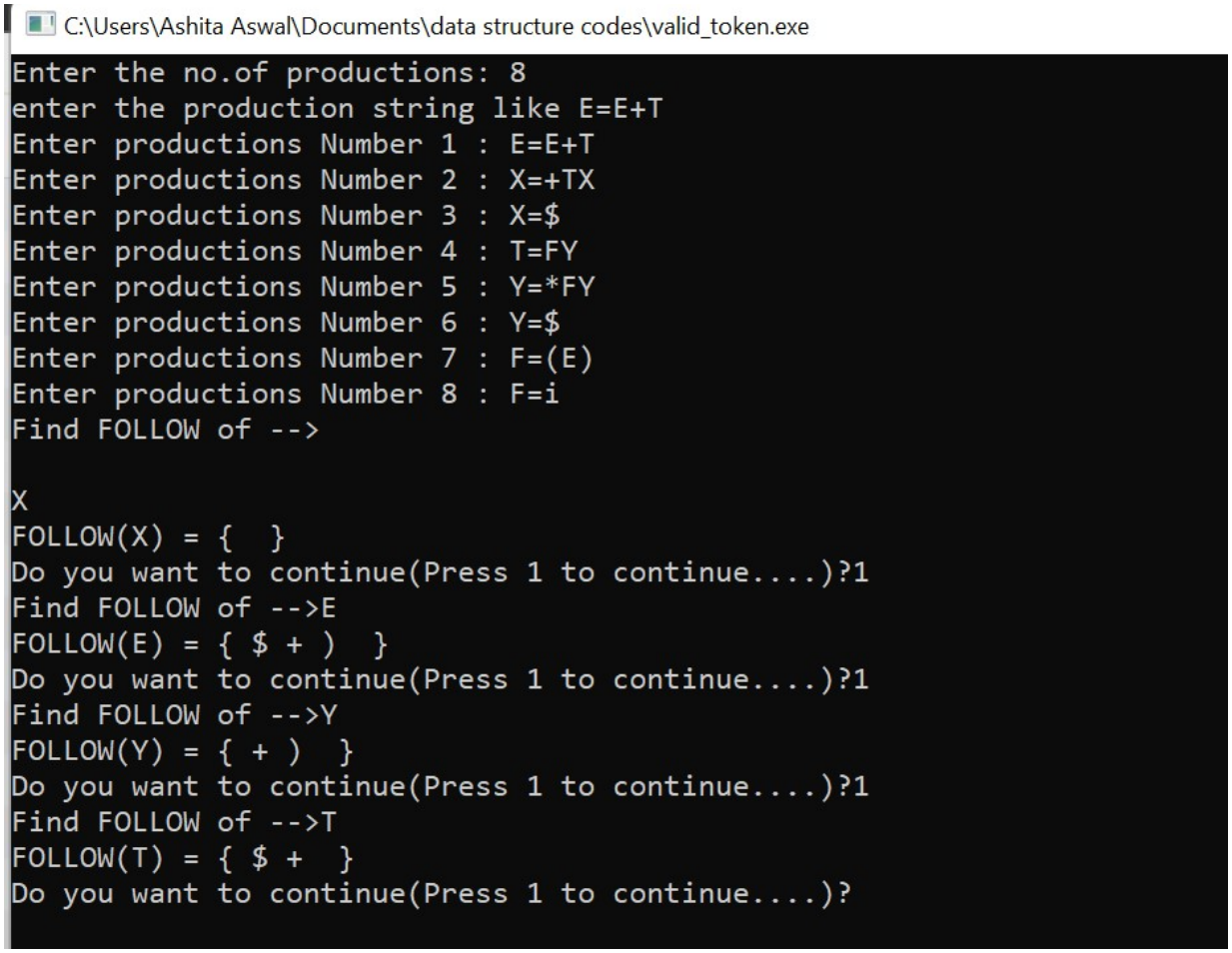
```
#include<stdio.h>
#include<string.h>
int nop,m=0,p,i=0,j=0;
char prod[10][10],res[10];
void FOLLOW(char c);
void first(char c);
void result(char);
void main()
{
int i;
int choice;
char c,ch;
printf("Enter the no.of productions: ");
scanf("%d", &nop);
printf("enter the production string like E=E+T\n");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


C:\Users\Ashita Aswal\Documents\data structure codes\valid_token.exe
Enter the no.of productions: 8
enter the production string like E=E+T
Enter productions Number 1 : E=E+T
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=i
Find FOLLOW of -->
X
FOLLOW(X) = { }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->E
FOLLOW(E) = { $ +) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->Y
FOLLOW(Y) = { +) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->T
FOLLOW(T) = { $ + }
Do you want to continue(Press 1 to continue....)?

Program 4 - Write a program in C to construct LL(1) parsing table.

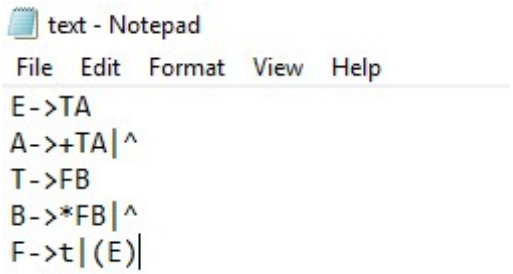
```
#include<stdio.h>
#include<string.h>
#define TSIZE 128
int table[100][TSIZE];
char terminal[TSIZE];
char nonterminal[26];
struct product {
    char str[100];
    int len;
}pro[20];
int no_pro;
char first[26][TSIZE];
char follow[26][TSIZE];
char first_rhs[100][TSIZE];
int isNT(char c) {
    return c >= 'A' && c <= 'Z';
}
void readFromFile() {
    FILE* fptr;
    fptr = fopen("text.txt", "r");
    char buffer[255];
    int i;
    int j;
    while (fgets(buffer, sizeof(buffer), fptr)) {
        printf("%s", buffer);
        j = 0;
        nonterminal[buffer[0] - 'A'] = 1;
        for (i = 0; i < strlen(buffer) - 1; ++i) {
            if (buffer[i] == '|') {
                ++no_pro;
                pro[no_pro - 1].str[j] = '\0';
                pro[no_pro - 1].len = j;
                pro[no_pro].str[0] = pro[no_pro - 1].str[0];
                pro[no_pro].str[1] = pro[no_pro - 1].str[1];
                pro[no_pro].str[2] = pro[no_pro - 1].str[2];
                j = 3;
            }
            else {
                pro[no_pro].str[j] = buffer[i];
                ++j;
                if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') {
                    terminal[buffer[i]] = 1;
                }}}
        pro[no_pro].len = j;
        ++no_pro;
    }}
void add_FIRST_A_to_FOLLOW_B(char A, char B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];
    }}
void add_FOLLOW_A_to_FOLLOW_B(char A, char B) {
```

```
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];
    }}
void FOLLOW() {
    int t = 0;
    int i, j, k, x;
    while (t++ < no_pro) {
        for (k = 0; k < 26; ++k) {
            if (!nonterminal[k])    continue;
            char nt = k + 'A';
            for (i = 0; i < no_pro; ++i) {
                for (j = 3; j < pro[i].len; ++j) {
                    if (nt == pro[i].str[j]) {
                        for (x = j + 1; x < pro[i].len; ++x) {
                            char sc = pro[i].str[x];
                            if (isNT(sc)) {
                                add_FIRST_A_to_FOLLOW_B(sc, nt);
                                if (first[sc - 'A']['^'])
                                    continue;
                            }
                            else {
                                follow[nt - 'A'][sc] = 1;
                            }
                            break;
                        }
                        if (x == pro[i].len)
                            add_FOLLOW_A_to_FOLLOW_B(pro[i].str[0], nt);
                    }  } }   } }    }
void add_FIRST_A_to_FIRST_B(char A, char B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^') {
            first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];
        } } }
void FIRST() {
    int i, j;
    int t = 0;
    while (t < no_pro) {
        for (i = 0; i < no_pro; ++i) {
            for (j = 3; j < pro[i].len; ++j) {
                char sc = pro[i].str[j];
                if (isNT(sc)) {
                    add_FIRST_A_to_FIRST_B(sc, pro[i].str[0]);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else {
                    first[pro[i].str[0] - 'A'][sc] = 1;
                }
                break;
            }
            if (j == pro[i].len)
                first[pro[i].str[0] - 'A']['^'] = 1;
```

```
        }
        ++t;
    } }
void add_FIRST_A_to_FIRST_RHS__B(char A, int B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            first_rhs[B][i] = first[A - 'A'][i] || first_rhs[B][i];
    } }
void FIRST_RHS() {
    int i, j;
    int t = 0;
    while (t < no_pro) {
        for (i = 0; i < no_pro; ++i) {
            for (j = 3; j < pro[i].len; ++j) {
                char sc = pro[i].str[j];
                if (isNT(sc)) {
                    add_FIRST_A_to_FIRST_RHS__B(sc, i);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else {
                    first_rhs[i][sc] = 1;
                }
                break;
            }
            if (j == pro[i].len)
                first_rhs[i]['^'] = 1;
        }
        ++t;
    } }
int main() {
    readFromFile();
    follow[pro[0].str[0] - 'A']['$'] = 1;
    FIRST();
    FOLLOW();
    FIRST_RHS();
    int i, j, k;
    printf("\n");
    for (i = 0; i < no_pro; ++i) {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
            char c = pro[i].str[0];
            printf("FIRST OF %c: ", c);
            for (j = 0; j < TSIZE; ++j) {
                if (first[c - 'A'][j]) {
                    printf("%c ", j);
                } }
            printf("\n");
        } }
    printf("\n");
    for (i = 0; i < no_pro; ++i) {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
            char c = pro[i].str[0];
            printf("FOLLOW OF %c: ", c);
            for (j = 0; j < TSIZE; ++j) {
```

```
            if (follow[c - 'A'][j]) {
                printf("%c ", j);
            }
        }
        printf("\n");
    }    }
printf("\n");
for (i = 0; i < no_pro; ++i) {
    printf("FIRST OF %s: ", pro[i].str);
    for (j = 0; j < TSIZE; ++j) {
        if (first_rhs[i][j]) {
            printf("%c ", j);
        } }
    printf("\n");
}
terminal['$'] = 1;
terminal['^'] = 0;
printf("\n");
printf("\n\t********** LL(1) PARSING TABLE **********\n");
printf("\t-------------------------------------------------\n");
printf("%-10s", "");
for (i = 0; i < TSIZE; ++i) {
    if (terminal[i])    printf("%-10c", i);
}
printf("\n");
int p = 0;
for (i = 0; i < no_pro; ++i) {
    if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0]))
        p = p + 1;
    for (j = 0; j < TSIZE; ++j) {
        if (first_rhs[i][j] && j != '^') {
            table[p][j] = i + 1;
        }
        else if (first_rhs[i]['^']) {
            for (k = 0; k < TSIZE; ++k) {
                if (follow[pro[i].str[0] - 'A'][k]) {
                    table[p][k] = i + 1;
                }       }
        }       }
}
k = 0;
for (i = 0; i < no_pro; ++i) {
    if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
        printf("%-10c", pro[i].str[0]);
        for (j = 0; j < TSIZE; ++j) {
            if (table[k][j]) {
                printf("%-10s", pro[table[k][j] - 1].str);
            }
            else if (terminal[j]) {
                printf("%-10s", "");
            }    }
        ++k;
        printf("\n");
    } } }
```

Input File -



Output-

```
E->TA
A->+TA|^
T->FB
B->*FB|^
F->t|(E)

FIRST OF E: ( t
FIRST OF A: + ^
FIRST OF T: ( t
FIRST OF B: * ^
FIRST OF F: ( t

FOLLOW OF E: $ )
FOLLOW OF A: $ )
FOLLOW OF T: $ ) +
FOLLOW OF B: $ ) +
FOLLOW OF F: $ ) * +

FIRST OF E->TA: ( t
FIRST OF A->+TA: +
FIRST OF A->^: ^
FIRST OF T->FB: ( t
FIRST OF B->*FB: *
FIRST OF B->^: ^
FIRST OF F->t: t
FIRST OF F->(E): (


        *************** LL(1) PARSING TABLE ******************
        ------------------------------------------------------
          $         (         )         *         +         t
E                   E->TA                                     E->TA
A         A->^                A->^                A->+TA
T                   T->FB                                     T->FB
B         B->^                B->^      B->*FB    B->^
F                   F->(E)                                    F->t
```

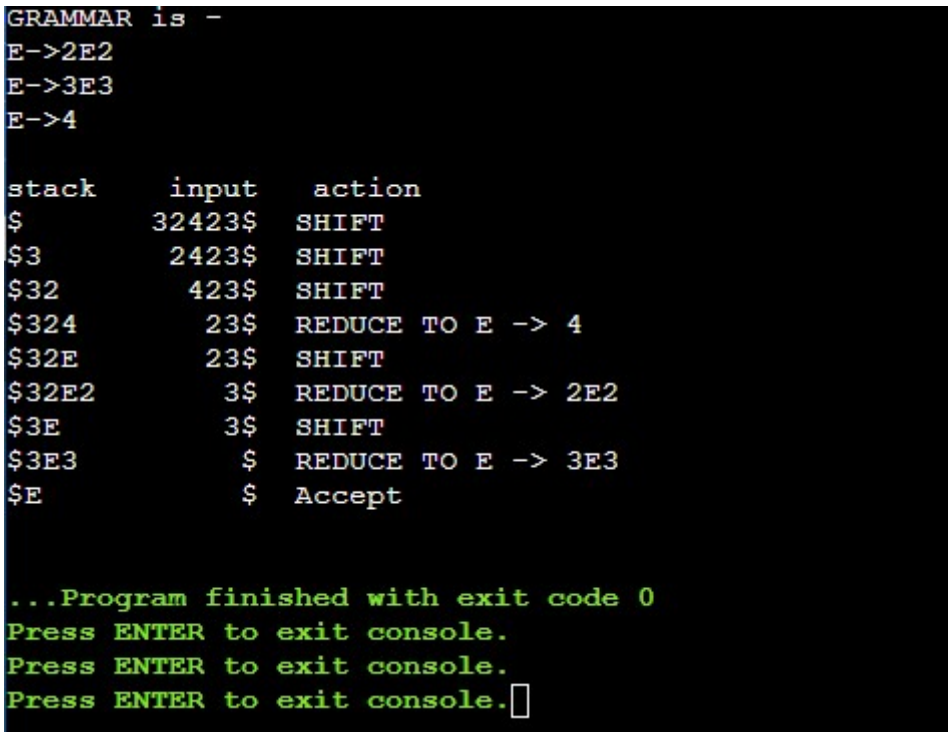
Program 5 - Write a program in C to implement shift reduce parser.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];
void check() {
        strcpy(ac,"REDUCE TO E -> ");
        for(z = 0; z < c; z++) {
                if(stk[z] == '4')
                {
                        printf("%s4", ac);
                        stk[z] = 'E';
                        stk[z + 1] = '\0';
                        printf("\n$%s\t%s$\t", stk, a);
                } }
        for(z = 0; z < c - 2; z++) {
                if(stk[z] == '2' && stk[z + 1] == 'E' && stk[z + 2] == '2') {
                        printf("%s2E2", ac);
                        stk[z] = 'E';
                        stk[z + 1] = '\0';
                        stk[z + 2] = '\0';
                        printf("\n$%s\t%s$\t", stk, a);
                        i = i - 2;
                }  }

        for(z=0; z<c-2; z++) {
                if(stk[z] == '3' && stk[z + 1] == 'E' && stk[z + 2] == '3') {
                        printf("%s3E3", ac);
                        stk[z]='E';
                        stk[z + 1]='\0';
                        stk[z + 1]='\0';
                        printf("\n$%s\t%s$\t", stk, a);
                        i = i - 2;
                } }
        return ;
}
int main() {
        printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
        strcpy(a,"32423");
        c=strlen(a);
        strcpy(act,"SHIFT");
        printf("\nstack \t input \t action");
        printf("\n$\t%s$\t", a);
        for(i = 0; j < c; i++, j++) {
                printf("%s", act);
                stk[i] = a[j];
                stk[i + 1] = '\0';
                a[j]=' ';
                printf("\n$%s\t%s$\t", stk, a);
                check();
        }
        check();
```

```
        if(stk[0] == 'E' && stk[1] == '\0')
                printf("Accept\n");
        else //else reject
                printf("Reject\n");
}
```

Output-



Program 6 - Write a program in C to implement operator precedence parser.

```
#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+i)*i

STACK   INPUT   ACTION
$i      *(i+i)*i$       Shift
$E      *(i+i)*i$       Reduced: E->i
$E*     (i+i)*i$        Shift
$E*(    i+i)*i$ Shift
$E*(i   +i)*i$  Shift
$E*(E   +i)*i$  Reduced: E->i
$E*(E+  i)*i$   Shift
$E*(E+i )*i$    Shift
$E*(E+E )*i$    Reduced: E->i
$E*(E   )*i$    Reduced: E->E+E
$E*(E)  *i$     Shift
$E*E    *i$     Reduced: E->)E(
$E      *i$     Reduced: E->E*E
$E*     i$      Shift
$E*i    $       Shift
$E*E    $       Reduced: E->i
$E      $       Reduced: E->E*E
$E$             Shift
$E$             Shift
Accepted;

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

Program 7 - Write a program in C to remove left recursion from a grammar.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()  {
   char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];
int i=0,j=0,flag=0,consumed=0;
printf("Enter the productions: ");
scanf("%1s->%s",l,r);
printf("%s",r);
while(sscanf(r+consumed,"%[^|]s",temp) == 1 && consumed <= strlen(r))  {
if(temp[0] == l[0])  {
flag = 1;
printf(productions[i++],"%s->%s%s'\0",l,temp+1,l);
      }
else
printf(productions[i++],"%s'->%s%s'\0",l,temp,l);
consumed += strlen(temp)+1;
   }
if(flag == 1)  {
printf(productions[i++],"%s->ε\0",l);
printf("\nThe productions after eliminating Left Recursion are:\n");
for(j=0;j<i;j++)
printf("%s\n",productions[j]);
   }
else
printf("\nThe Given Grammar has no Left Recursion");
getch();
}
```

Output-

```
Enter the productions: A->Ab|D
Ab|D
The productions after eliminating Left Recursion are:
A->bA'
A'->DA'
A->ε
```

Program 8 - Write a program in C to design LALR Bottom up parser.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
void push(char *,int *,char);
charstacktop(char *);
voidisproduct(char,char);
intister(char);
intisnter(char);
intisstate(char);
void error();
voidisreduce(char,char);
char pop(char *,int *);
voidprintt(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"}
};
structgotol {
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"},
 };
charter[6]={'i','+','*',')','(','$'};
charnter[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() {
charinp[80],x,p,dl[80],y,bl='a';
int i=0,j,k,l,n,m,c,len;
clrscr();
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;
} }
charstacktop(char *s) {
char i;
i=s[top];
return i; }
voidisproduct(char x,char p) {
intk,l;
k=ister(x);
l=isstate(p);
strcpy(temp,A[l-1].row[k-1]); }
intister(char x) {
int i;
for(i=0;i<6;i++)
if(x==ter[i])
return i+1;
return 0; }
intisnter(char x) {
int i;
for(i=0;i<3;i++)
if(x==nter[i])
return i+1;
return 0;
}
intisstate(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);
}
voidisreduce(char x,char p) {
intk,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;
}
voidprintt(char *t,int *p,charinp[],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$
0T2*7	i+i$
0T2*7i5	+i$
0T2*7i5F10	+i$
0T2	+i$
0E1	+i$
0E1+6	i$
0E1+6i5	$
0E1+6F3	$
0E1+6T9	$
0E1	$

accept the input*/