INDERPRASTHA ENGINEERING COLLEGE

GHAZIABAD

Department of Information Technology

Compiler Design Lab (RCS-652)
(2019-20)

Name	:	**Natasha Sharma**
Roll Number	:	**1703013043**
Course	:	**B.Tech. (I.T.)**
Year	:	**3**
Semester	:	**6**
Section	:	**A**

INDEX

WAP for implementation of LEXICAL ANALYZER for IF statement.

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#include<string.h>charvars[100][100];i
ntvcnt;
charinput[1000],c;chartoken[50],tlen;
intstate=0,pos=0,i=0,id;
char*getAddress(charstr[])
{
for(i=0;i<vcnt;i++)if(strcmp(str,vars[i])==0
)returnvars[i];strcpy(vars[vcnt],str);return
vars[vcnt++];
}
intisrelop(charc)
{
if(c=='>'||c=='<'||c=='|'||c=='=')return1;
elsereturn0;
}
intmain(void)
{
clrscr();
printf("EntertheInputString:");gets(input);
do
{
c=input[pos];putchar(c);switch(state)
{
case0:if(c=='i')state=1;break;
case1:if(c=='f')
{
printf("\t<1,1>\n");state=2;
}
break;case2:
if(isspace(c))printf("\b");if(isalpha(c)
)
{
token[0]=c;tlen=1;state=3;
}
if(isdigit(c))state=4;
if(isrelop(c))state=5;
if(c==';')printf("\t<4,4>\n");
if(c=='(')printf("\t<5,0>\n");
if(c==')')printf("\t<5,1>\n");
if(c=='{')printf("\t<6,1>\n");
if(c=='}')printf("\t<6,2>\n");break;
case3:
if(!isalnum(c))
{
token[tlen]='\o';printf("\b\t<2,%p>\n",getAddress(token));
state=2;
pos--;
```


NATASHA SHARMA
1703013043

```
}
elsetoken[tlen++]=c;break;
case4:if(!isdigit(c))
{
printf("\b\t<3,%p>\n",&input[pos]);state=2;
pos--;
}
break;case5:
id=input[pos-1];if(c=='=')
printf("\t<%d,%d>\n",id*10,id*10);else
{
printf("\b\t<%d,%d>\n",id,id);pos--;
}
state=2;break;
}
pos++;
}
while(c!=0);
getch();
return0;
}
```


NATASHA SHARMA
1703013043

Output of LEXICAL ANALYZER for IF statement.

```
Enter the Input String:if(a>=b)max=a;
if      <1,1>
(       <5,0>
a       <2,00AA>
>=      <620,620>
b       <2,010E>
)       <5,1>
max     <2,0172>
=       <61,61>
a       <2,01D6>
;       <4,4>
```


NATASHA SHARMA
1703013043

WAP for implementation of LEXICAL ANALYZER for Arithmetic Expression.

```
#include<stdio.h>
#include<ctype.h>
#include<conio.h>
#include<string.h>
char vars[100][100];
int vcnt;
char input[1000],c;
char token[50],tlen;
int state=0,pos=0,i=0,id;
char *getAddress(char str[]){
for(i=0;i<vcnt;i++)
if(strcmp(str,vars[i])==0)
return vars[i];
strcpy(vars[vcnt],str);
return vars[vcnt++];
}
int isrelop(char c){
if(c=='+'||c=='-'||c=='*'||c=='/'||c=='%'||c=='^') return 1;
else return 0;
}
int main(void)
{
clrscr();
printf("Enter the Input String:");
gets(input);
do
{
c=input[pos];
putchar(c);
switch(state)
{
case 0: if(isspace(c))
printf("\b");
if(isalpha(c))
{
token[0]=c;
tlen=1;
state=1;
}
if(isdigit(c))
state=2;
if(isrelop(c))
```

WAP for implementation of LEXICAL ANALYZER for Arithmetic Expression.

```
state=3;
if(c==';')
printf("\t<3,3>\n");
if(c=='=')
printf("\t<4,4>\n");
break;
case 1: if(!isalnum(c))
{
token[tlen]='\o';
printf("\b\t<1,%p>\n",getAddress(token));
state=0;
pos--;
}
else token[tlen++]=c;
break;
case 2: if(!isdigit(c))
{
printf("\b\t<2,%p>\n",&input[pos]);
state=0;
pos--;
}
break;
case 3:
id=input[pos-1];
if(c=='=')
printf("\t<%d,%d>\n",id*10,id*10);
else{
printf("\b\t<%d,%d>\n",id,id);
pos--;
}
state=0;
break;
}
pos++;
}
while(c!=0);
getch();
return 0;
}
```

OUTPUT for implementation of LEXICAL ANALYZER for Arithmetic Expression.

```
Enter the Input String:a=a*2+b/c;
a       <1,08CE>
=       <4,4>
a       <1,08CE>
*       <42,42>
2       <2,04E9>
+       <43,43>
b       <1,0932>
/       <47,47>
c       <1,0996>
;       <3,3>
```

S.No: 3	Exp. Name: ***C program to identify whether a given line is a comment or not***	Date:

ID: 1703013043

Aim:

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

Source Code:

comment.c

```
#include<stdio.h>
#include<conio.h>
void main(){
   char a[100];
   int i=2,j=0;
   printf("Enter comment: ");
   scanf("%s",a);
   if(a[0]=='/'){
      if(a[1]=='/'){
         printf("It is a comment\n");
      }
      else if(a[1]=='*'){
         for(i=2;i<=100;i++){
            if(a[i]=='*' && a[i+1]=='/'){
               printf("It is a comment");
               j=1;
               break;
            }
            else continue;
         }
         if(j==0){
            printf("It is a comment\n");
         }
         else{
            printf("It is not a comment\n");
         }
      }

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

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter comment: //hello
It is a comment

Test Case - 2
User Output
Enter comment: hello

Test Case - 2
It is not a comment

Test Case - 3
User Output
Enter comment: /* hello how are you */
It is a comment

S.No: 4	Exp. Name: ***C program to recognize strings***	Date:

Aim:

Write a C program to recognize strings under **a*, a*b+, abb**.

Source Code:

recognizeStrings.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void main(){
   char s[20],c;
   int state=0,i=0;
   printf("Enter a string: ");
   scanf("%s",s);
   while(s[i]!='\0'){
      switch(state){
         case 0: c=s[i++];
         if(c=='a')
         state=1;
         else if(c=='b')
         state=2;
         else
         state=6;
         break;
         case 1: c=s[i++];
         if(c=='a')
         state=3;
         else if (c=='b')
         state=4;
         else
         state=6;
         break;
         case 2: c=s[i++];
         if (c=='a')
         state=6;
         else if(c=='b')
         state=2;
         else
         state=6;
         break;
         case 3: c=s[i++];
         if (c=='a')
         state=3;
         else if (c=='b')
         state=2;
         else
         state=6;
         break;
         case 4: c=s[i++];
         if (c=='a')
         state=6;
         else if(c=='b')
```

```
        state=5;
        else
        state=6;
        break;
        case 5: c=s[i++];
        if (c=='a')
        state=6;
        else if(c=='b')
        state=2;
        else
        state=6;
        break;
        case 6:
        printf("%s is not recognised\n",s);
        exit(0);

     }
   }
   if(state==1)
   printf("%s is accepted under rule 'a'\n",s);
   else if((state==2) ||(state==4))
   printf("%s is accepted under rule 'a*b+'\n",s);
   else if(state==5)
   printf("%s is accepted under rule 'abb'\n",s);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter a string: cdgs
cdgs is not recognised

Test Case - 2
User Output
Enter a string: a
a is accepted under rule 'a'

Test Case - 3
User Output
Enter a string: abbbbbb
abbbbbb is accepted under rule 'a*b+'

Test Case - 4
User Output
Enter a string: abb
abb is accepted under rule 'abb'

S.No: 5	Exp. Name: ***C program to test the identifier is valid or not***	Date:

Aim:

Write a C program to test whether a given identifier is **valid** or **not valid**.

Source Code:

identifier.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
   char s[25];
   int count=0,flag;
   printf("Enter an identifier: ");
   scanf("%s",s);
   if((s[0]>='a' && s[0]<='z') || (s[0]>='A' && s[0]<='Z') || s[0]=="_"){
      for(int i=1;i<=strlen(s);i++){
         if((s[i]>='a' && s[i]<='z') || (s[i]>='A' && s[i]<='Z') || (s[i]>='0' && s[i]<
='9') || s[i]<='_'){
            count++;
         }

      }
      if(count==strlen(s)){
         flag=0;
      }
   }
   else flag=1;
   if(flag==1)
   printf("Not a valid identifier\n");
   else
   printf("Valid identifier\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter an identifier: first
Valid identifier

Test Case - 2
User Output
Enter an identifier: 1aqw
Not a valid identifier

Test Case - 3

Test Case - 3
User Output
Enter an identifier: 5447
Not a valid identifier

Test Case - 4
User Output
Enter an identifier: CodeTantra
Valid identifier

S.No: 6	Exp. Name: ***Construction of NFA from Regular Expression***	Date:

ID: 1703013043

Aim:

Write a C program for construction of **NFA** from **Regular Expression**.

Source Code:

nfa.c

```
#include<stdio.h>
#include<conio.h>
void main(){
   char reg[20];
   int q[20][3],i,j,len,a,b;
   for(a=0;a<20;a++){
      for(b=0;b<3;b++){
         q[a][b]=0;
      }
   }
   printf("Regular expression: ");
   scanf("%s",reg);
   len=strlen(reg);
   i=0,j=1;
   while(i<len){
      if(reg[i]=='a' & reg[i+1]!='/' & reg[i+1]!='*'){
         q[j][0]=j+1;
         j++;
      }
      if(reg[i]=='b' & reg[i+1]!='/' & reg[i+1]!='*'){
         q[j][1]=j+1;
         j++;
      } if(reg[i]=='e'&reg[i+1]!='/' & reg[i+1]!='*'){
         q[j][2]=((j+1));
         j++;
      }if(reg[i]=='a' & reg[i+1]=='/' & reg[i+2]=='b'){
         q[j][2]=((j+1)*10)+(j+3);
         j++;
         q[j][0]=j+1;
         j++;
         q[j][2]=j+3;
         j++;
         q[j][1]=j+1;
         j++;
         q[j][2]=j+1;
         j++;
         i=i+2;
      }if(reg[i]=='b' &reg[i+1]=='/' & reg[i+2]=='a'){
         q[j][2]=((j+1)*10)+(j+3);
         j++;
         q[j][1]=j+1;
         j++;
         q[j][2]=j+3;
         j++;
         q[j][0]=j+1;
         j++;
         q[j][2]=j+1;
```

```
         j++;
         i=i+2;
      }if(reg[i]=='a' & reg[i+1]=='*'){
         q[j][2]=((j+1)*10)+(j+3);
         j++;
         q[j][0]=j+1;
         j++;
         q[j][2]=((j+1)*10)+(j-1);
         j++;
      }if(reg[i]=='b'& reg[i+1]=='*'){
         q[j][2]=((j+1)*10)+(j+3);
         j++;
         q[j][1]=j+1;
         j++;
         q[j][2]=((j+1)*10)+(j-1);
      }if(reg[i]==')' & reg[i+1]=='*'){
         q[0][2]=((j+1)*10)+1;
         q[j][2]=((j+1)*10)+1;
         j++;
      }
      i++;

   }
   printf("Transition function\n");
   for(i=0;i<=j;i++){
      if(q[i][0]!=0){
         printf("q[%d,a]-->%d",i,q[i][0]);
      }
      if(q[i][1]!=0){
         printf("q[%d,b]-->%d",i,q[i][1]);
      }
      if(q[i][2]!=0){
         if(q[i][2]<10)
           printf("q[%d,e]-->%d",i,q[i][2]);
         else
            printf("q[%d,e]-->%d & %d",i,q[i][2]/10,q[i][2]%10);
       }

   }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Regular expression: a+b
Transition function
q[1,a]-->2q[2,b]-->3

Test Case - 2
User Output
Regular expression: a+c/d*a+v
Transition function
q[1,a]-->2q[2,a]-->3

S.No: 7	Exp. Name: ***construction of DFA from NFA***	Date:

Aim:

Write a C program for construction of **DFA** from **NFA**.

Source Code:

DfaFromNfa.c

```
#include<stdio.h>
#include<string.h>
#include<math.h>
int ninputs,dfa[100][2][100]={0},state[10000]={0},ch[10],str[1000],go[10000][2]={0},arr
[10000]={0};
int main(){
   int st,fin,in,f[10],i,j=3,s=0,final=0,flag=0,curr1,curr2,k,l,c;
   printf("Follow the one based indexing\n");
   printf("Enter the number of states: ");
   scanf("%d",&st);
   printf("Give state numbers from 0 to %d\n",st-1);
   for(i=0;i<st;i++){
      state[(int)(pow(2,i))]=1;
   }
   printf("Enter number of final states: ");
   scanf("%d",&fin);
   printf("Enter final states: ");
   for(i=0;i<fin;i++){
      scanf("%d",&f[i]);
   }
   int p,q,r,rel;
   printf("Enter the number of rules according to NFA: ");
   scanf("%d",&rel);
   printf("Define transition rule as \"initial state input symbol final state: \"");
   for(i=0;i<rel;i++){
      scanf("%d%d%d",&p,&q,&r);
      if(q==0)
      dfa[p][0][r]=1;
      else
      dfa[p][1][r]=1;
   }
   printf("\nEnter initial state: ");
   scanf("%d",&in);
   in=pow(2,in);
   i=0;
   printf("Solving according to DFA");
   int x=0;
   for(i=0;i<st;i++){
      //printf("\nfor %d ----",arr[x])
      for(j=0;j<2;j++){
         int stf=0;
         for(k=0;k<st;k++){
            if(dfa[i][j][k]==1)
            stf=stf+pow(2,k);
         }
         go[(int)(pow(2,i))][j]=stf;
         printf("\n%d-%d-->%d",(int)(pow(2,i)),j,stf);
```

```
         if(state[stf]==0)
         arr[x++]=stf;
         state[stf]=1;
      }

   }
   for(i=0;i<x;i++){
      printf("\nfor %d ----",arr[x]);
      for(j=0;j<2;j++){
         int new=0;
         for(k=0;k<st;k++){
            if(arr[i] & (1<<k)){
               int h=pow(2,k);
               if(new==0)
               new=go[h][j];
            new=new|(go[h][j]);
            }
         }
         if(state[new]==0){
            arr[x++]=new;
            state[new]=1;

         }
      }
   }
   printf("\nThe total number of distinct states are:\n");
   printf("STATE 0 1\n");
   for(i=0;i<10000;i++){
      if(state[i]==1){
         int y=0;
         if(i==0)
         printf("q0 ");
         else
         for(j=0;j<st;j++){
            int x=1<<j;
            if(x&i){
               printf("q%d ",j);
               y=y+pow(2,j);
            }
         }
         printf(" %d %d",go[y][0],go[y][1]);
         printf("\n");

      }

   }
   j=3;
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

Test Case - 1

User Output

```
Follow the one based indexing 3
Enter the number of states:  3
Give state numbers from 0 to 2 1
Enter number of final states:  1
Enter final states:  4
Enter the number of rules according to NFA:  4
Define transition rule as "initial state input symbol final state: " 1 0 1
1 1 1                                                                1 1 1
1 0 2                                                                1 0 2
2 0 4                                                                2 0 4
Enter initial state:  1
Solving according to DFA
1-0-->0
1-1-->0
2-0-->6
2-1-->2
4-0-->0
4-1-->0
for 0 ----
for 0 ----
The total number of distinct states are:
STATE 0 1
q0  0 0
q0  0 0
q1  6 2
q2  0 0
q1 q2  0 0
```

Test Case - 2

User Output

```
Follow the one based indexing 3
Enter the number of states:  3
Give state numbers from 0 to 2 1
Enter number of final states:  1
Enter final states:  3
Enter the number of rules according to NFA:  3
Define transition rule as "initial state input symbol final state: " 1 2 4
2 1 3                                                                2 1 3
2 4 2                                                                2 4 2
Enter initial state:  1
Solving according to DFA
1-0-->0
1-1-->0
2-0-->0
2-1-->0
4-0-->0
4-1-->4
for 0 ----
```

Test Case - 2

```
The total number of distinct states are:
STATE 0 1
q0  0 0
q0  0 0
q1  0 0
q2  0 4
```

S.No: 8	Exp. Name: ***Implementation of shift reduce parsing algorithm***	Date:

ID: 1703013043

Aim:

Write a C program to implement **Shift Reduce Parsing** algorithm.

Source Code:

shiftReduceParsing.c

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
int main(){
   puts("Grammar is E->E+E \n E->E*E \n E->(E) \n E->id");
   puts("Enter input string: ");
   gets(a);
   c=strlen(a);
   strcpy(act,"Shift->");
   puts("stack \t input \t action");
   for(k=0,i=0;j<c;k++,i++,j++){
      if(a[j]=='i' && a[j+1]=='d'){
         stk[i]=a[j];
         stk[i+1]=a[j+1];
         stk[i+2]='\0';
         a[j]=' ';
         a[j+1]=' ';
         printf("$%s\t%s$\t%sid\n",stk,a,act);
         check();
      }
      else{
         stk[i]=a[j];
         stk[i+1]='\0';
         a[j]=' ';
         printf("$%s\t%s$\t%ssymbols\n",stk,a,act);
         check();
      }
   }
}
void check(){
   strcpy(ac,"Reduce To E");
   for(z=0;z<c;z++)
   if(stk[z]=='i' && stk[z+1]=='d'){
      stk[z]='E';
      stk[z+1]='\0';
      printf("$%s\t%s$\t%s\n",stk,a,ac);
      j++;
   }
   for(z=0;z<c;z++)
   if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E'){
      stk[z]='E';
      stk[z+1]='\0';
      stk[z+2]='\0';
      printf("$%s\t%s$\t%s\n",stk,a,ac);
      i=i-2;
```

```
   }
   for(z=0;z<c;z++)
   if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E'){
      stk[z]='E';
      stk[z+1]='\0';
      stk[z+1]='\0';
      printf("$%s\t%s$\t%s\n",stk,a,ac);
      i=i-2;
   }
   for(z=0;z<c;z++){
      if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')'){
         stk[z]='E';
         stk[z+1]=='\0';
         stk[z+1]=='\0';
         printf("$%s\t%s$\t%s\n",stk,a,ac);
         i=i-2;
      }
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1		
User Output		
Grammar is E->E+E id+id*id+id		
E->E*E id+id*id+id		
E->(E) id+id*id+id		
E->id id+id*id+id		
Enter input string: id+id*id+id		
stack	input	action
$id	+id*id+id$	Shift->id
$E	+id*id+id$	Reduce To E
$E+	id*id+id$	Shift->symbols
$E+id	*id+id$	Shift->id
$E+E	*id+id$	Reduce To E
$E	*id+id$	Reduce To E
$E*	id+id$	Shift->symbols
$E*id	+id$	Shift->id
$E*E	+id$	Reduce To E
$E	+id$	Reduce To E
$E+	id$	Shift->symbols
$E+id	$	Shift->id
$E+E	$	Reduce To E
$E	$	Reduce To E

Test Case - 2
User Output
Grammar is E->E+E id*id+id*id
E->E*E id*id+id*id
E->(E) id*id+id*id

Test Case - 2

```
 E->id id*id+id*id
Enter input string:  id*id+id*id
stack    input   action
$id       *id+id*id$    Shift->id
$E        *id+id*id$    Reduce To E
$E*        id+id*id$    Shift->symbols
$E*id        +id*id$    Shift->id
$E*E         +id*id$    Reduce To E
$E           +id*id$    Reduce To E
$E+           id*id$    Shift->symbols
$E+id           *id$    Shift->id
$E+E            *id$    Reduce To E
$E              *id$    Reduce To E
$E*              id$    Shift->symbols
$E*id              $    Shift->id
$E*E               $    Reduce To E
$E                 $    Reduce To E
```

S.No: 9	Exp. Name: ***Operator precedence parser***	Date:

Aim:

Write a C program to implement **Operator Precedence Parser**.

Source Code:

operatorPrecedenceParser.c

```
#include<stdio.h>
#include<string.h>
void f(){
   printf("Not operator grammar\n");
   exit(0);
}
void main(){
   char grm[20][20],c;
   int i,n,j=2,flag=0;
   printf("Enter number of productions: ");
   scanf("%d",&n);
   printf("Enter the grammar:");
   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\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1	
User Output	
Enter number of productions: 3	
Enter the grammar: A=A*A	
B=AA	B=AA
A=$	A=$
Not operator grammar	

Test Case - 2	
User Output	
Enter number of productions: 2	
Enter the grammar: A=A/A	
B=A+A	B=A+A
Operator grammar	

S.No: 10	Exp. Name: ***Implementation of recursive descent parser***	Date:

Aim:

Write a C program to implement **Recursive Descent Parser**.

Source Code:

recursiveDescentParser.c

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
void Tprime();
void Tprime();
void Eprime();
void E();
void check();
void T();
char exp[10];
int count,flag;
int main(){
   count=0;
   flag=0;
   printf("Enter an arithmetic expression: ");
   scanf("%s", exp);
   E();
   if((strlen(exp)==count)&&(flag==0)){
      printf("Accepted\n");
   }
   else{
      printf("Rejected\n");
   }

}
void E(){
   T();
   Eprime();
}
void T(){
   check();
   Tprime();
}
void Tprime(){
   if(exp[count]=='*'){
      count++;
      check();
      Tprime();
   }
}
void check(){
   if(isalnum(exp[count])){
      count++;
   }else if(exp[count]=='('){
      count++;
      E();
      if(exp[count]=')'){
```

```
         count++;
      } else{
         flag=1;
      }
   } else{
      flag=1;
   }
}
void Eprime(){
   if(exp[count]=='+'){
      count++;
      T();
      Eprime();
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter an arithmetic expression: a+a*a
Accepted

Test Case - 2
User Output
Enter an arithmetic expression: a/b
Rejected

S.No: 11	Exp. Name: ***C program for implementing the functionalities of predictive parser***	Date:

Aim:

Write a C program for implementing the functionalities of **predictive parser**.

Source Code:

predictiveParser.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char prol[7][10]={"S","A","A","B","B","C","C"};
char pror[7][10]={"A","Bb","Cd","aB","@","Cc","@"};
char prod[7][10]={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"};
char first[7][10]={"abcd","ab","cd","a@","@","c@","@"};
char follow[7][10]={"$","$","$","a$","b$","c$","d$"};
char table[5][6][10];
numr(char c){
   switch(c){
      case 'S':return 0;
      case 'A':return 1;
      case 'B':return 2;
      case 'C':return 3;
      case 'a':return 0;
      case 'b':return 1;
      case 'c':return 2;
      case 'd':return 3;
      case '$':return 4;
   }
   return(2);

}
void main(){
   int i,j,k;
   for(i=0;i<5;i++)
   for(j=0;j<6;j++)
   strcpy(table[i][j]," ");
   printf("The following is the predictive parsing table for the following grammar:
\n");
   for(i=0;i<7;i++)
   printf("%s\n",prod[i]);
   printf("Predictive parsing table is: \t\n");
   fflush(stdin);
   for(i=0;i<7;i++){
      k=strlen(first[i]);
      for(j=0;j<10;j++)
      if(first[i][j]!='@')
      strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);

   }
   for(i=0;i<7;i++){
      if(strlen(pror[i])==1){
         if(pror[i][0]=='@'){
            k=strlen(follow[i]);
            for(j=0;j<k;j++)
```

```
            strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);

         }
      }
   }
   strcpy(table[0][0]," ");
   strcpy(table[0][1],"a");
   strcpy(table[0][2],"b");
   strcpy(table[0][3],"c");
   strcpy(table[0][4],"d");
   strcpy(table[0][5],"$");
   strcpy(table[1][0],"S");
   strcpy(table[2][0],"A");
   strcpy(table[3][0],"B");
   strcpy(table[4][0],"C");
   printf("");
   for(i=0;i<5;i++)
   for(j=0;j<6;j++){
      printf("%-10s",table[i][j]);
      if(j==5)
      printf("\t\n");
   }
}
```


Execution Results - All test cases have succeeded!

Test Case - 1					
User Output					
The following is the predictive parsing table for the following grammar:					
S->A					
A->Bb					
A->Cd					
B->aB					
B->@					
C->Cc					
C->@					
Predictive parsing table is:					
	a	b	c	d	$
S	S->A	S->A	S->A	S->A	
A	A->Bb	A->Bb	A->Cd	A->Cd	
B	B->aB	B->@	B->@		B->@
C			C->@	C->@	C->@

S.No: 12	Exp. Name: ***C program for constructing of LL(1) parsing***	Date:

Aim:

Write a C program for constructing of **LL(1) parsing**.

Source Code:

parsing.c

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
char s[20],stack[20];
int main(){
   char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","f
c"," "," "," ","n","*fc"," a ","n","n","i"," "," ","(e)"," "," "};
   int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0,};
   int i,j,k,n,str1,str2;
   printf("Enter the input string: ");
   scanf("%s",s);
   strcat(s,"$");
   n=strlen(s);
   stack[0]='$';
   stack[1]='e';
   i=1;
   j=0;
   printf("Stack\tInput\n");
   while((stack[i]!='$')&&(s[j]!='$')){
      if(stack[i]==s[j]){
         i--;
         j++;
      }
      switch(stack[i]){
         case 'e':str1=0;
         break;
         case 'b':str1=1;
         break;
         case 't':str1=2;
         break;
         case 'c':str1=3;
         break;
         case 'f':str1=4;
         break;
      }
      switch(s[j]){
         case 'i':str2=0;
         break;
         case '+':str2=1;
         break;
         case '*':str2=2;
         break;
         case '(':str2=3;
         break;
         case ')':str2=4;
         break;
         case '$':str2=5;
```

```
         break;
      }
      if(m[str1][str2][0]=='\0'){
         printf("Error\n");
         printf("\n");
         exit(0);
      }
      else if(m[str1][str2][0]=='n')
      i--;
      else if(m[str1][str2][0]=='i')
      stack[i]='i';
      else{
         for(k=size[str1][str2]-1;k>=0;k--){
            stack[i]=m[str1][str2][k];
            i++;
         }
         i--;
      }
      for(k=0;k<=i;k++)
      printf("%c",stack[k]);
      printf("\t");
      for(k=j;k<n;k++)
      printf("%c",s[k]);
      printf("\n");
   }
   printf("Success\n");
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the input string: a*a+a
Stack Input
Error

Test Case - 2
User Output
Enter the input string: i*i+i
Stack Input
$bt i*i+i$
$bcf i*i+i$
$bci i*i+i$
$bcf* *i+i$
$bci i+i$
$b +i$
$bt+ +i$
$bcf i$
$bci i$
$b $
Success

S.No: 13	Exp. Name: ***C program to design LALR bottom up parser***	Date:

Aim:

Write a program to design **LALR Bottom up Parser**.

Source Code:

bottomUpParser.c

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

};
struct gotol{
   char r[3][4];
};
const struct gotol G[12]={
   {"b","c","d"},
   {"emp","emp","emp"},
   {"emp","emp","emp"},
   {"emp","emp","emp"},
   {"i","c","d"},
   {"emp","emp","emp"},
   {"emp","j","d"},
   {"emp","emp","k"},
   {"emp","emp","emp"},
   {"emp","emp","emp"},
```

```
};
char ter[6]={'i','+','*',')','(','$'};
char nter[3]={'E','T','F'};
char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};
char stack[100];
int top=-1;
char temp[10];
struct grammar{
   char left;
   char right[5];
};
const struct grammar rl[6]={
   {'E',"e+T"},
   {'E',"T"},
   {'T',"T*F"},
   {'T',"F"},
   {'F',"(E)"},
   {'F',"i"},
};
void main(){
   char inp[80],x,p,dl[80],y,bl='a';
   int i=0,j,k,l,n,m,c,len;
   printf("Enter the input: ");
   scanf("%s",inp);
   len=strlen(inp);
   inp[len]='$';
   inp[len+1]='\0';
   push(stack,&top,bl);
   printf("Stack\tInput");
   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("\nAccept the input\n");
    else
    printf("\nDo not accept the input\n");
}
void push(char *s,int *sp,char item){
   if(*sp==100)
   printf("\nStack is full\n");
   else{
      *sp=*sp+1;
      s[*sp]=item;
   }
}
char stacktop(char *s){
   char i;
   i=s[top];
   return i;
}
void isproduct(char x,char p){
   int k,l;
   k=ister(x);
   l=isstate(p);
   strcpy(temp,A[l-1].row[k-1]);
}
int ister(char x){
   int i;
   for(i=0;i<6;i++)
   if(x==ter[i])
   return i+1;
   return 0;
}
int isnter(char x){
   int i;
   for(i=0;i<3;i++)
   if(x==nter[i])
   return i+1;
   return 0;
}
int isstate(char p){
   int i;
   for(i=0;i<12;i++)
   if(p==states[i])
   return i+1;
   return 0;
}
void error(){
   printf("\nError in the input\n");
   exit(0);
```

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

```
    }
  }
```


Execution Results - All test cases have succeeded!

Test Case - 1
User 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*7F10 +i$
0E1 +i$
0E1+6 i$
0E1+6i5 $
0E1+6F3 $
0E1+6T9 $
0E1 $
Accept the input

Test Case - 2
User Output
Enter the input: i*i*i+i
Stack Input
0 i*i*i+i$
0i5 *i*i+i$
0F3 *i*i+i$
0T2 *i*i+i$
0T2*7 i*i+i$
0T2*7i5 *i+i$
0T2*7F10 *i+i$
0T2 *i+i$
0T2*7 i+i$
0T2*7i5 +i$
0T2*7F10 +i$
0E1 +i$
0E1+6 i$
0E1+6i5 $
0E1+6F3 $
0E1+6T9 $
0E1 $
Accept the input

Test Case - 3

Test Case - 3

User Output

```
Enter the input:  i*i
Stack   Input
0       i*i$
0i5     *i$
0F3     *i$
0T2     *i$
0T2*7   i$
0T2*7i5 $
0T2*7F10        $
0T2     $
0E1     $
Accept the input
```

Test Case - 4

User Output

```
Enter the input:  i/i
Stack   Input
0       i/i$
0i5     /i$
Error in the input
```

S.No: 14	Exp. Name: ***Implementation of code optimization techniques***	Date:

ID: 1703013043

Aim:

Write a C program to implement **Code Optimization Techniques**.

Source Code:

codeOptimizationTechniques.c

```
#include<stdio.h>
#include<string.h>
struct op{
   char l;
   char r[20];

}op[10],pr[10];
int main(){
   int a,i,k,j,n,z=0,m,q;
   char *p,*l,temp,t,*tem;
   printf("Enter the number of values: ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Left: ");
      scanf(" %c",&op[i].l);
      printf("Right: ");
      scanf(" %s",&op[i].r);
      }
      printf("Intermediate Code\n");
      for(i=0;i<n;i++)
      {
         printf("%c=",op[i].l);
         printf("%s\n",op[i].r);

      }
      for(i=0;i<n-1;i++)
      {
         temp=op[i].l;
         for(j=0;j<n;j++)
         {
            p=strchr(op[j].r,temp);
            if(p)
            {
               pr[z].l=op[i].l;
                strcpy(pr[z].r,op[i].r);
                z++;
                }

         }

      }
      pr[z].l=op[n-1].l;
      strcpy(pr[z].r,op[n-1].r);
      z++;
      printf("After Dead Code Elimination\n");
      for(k=0;k<z;k++)
```

```
      {
         printf("%c\t=",pr[k].l);
         printf("%s\n",pr[k].r);
         }
         for(m=0;m<z;m++)
         {
            tem=pr[m].r;
            for(j=m+1;j<z;j++)
            {
               p=strstr(tem,pr[j].r);
               if(p)
               {
                  t=pr[j].l;
                  pr[j].l=pr[m].l;
                  for(i=0;i<z;i++)
                  {
                     l=strchr(pr[i].r,t);
                     if(l)
                     {
                        a=l-pr[i].r;
                        printf("pos: %d\n",a);
                        pr[i].r[a]=pr[m].l;
                        }
                        }
                        }
                        }
                        }
                        printf("Eliminate Common Expression\n");
                        for(i=0;i<z;i++)
                        {
                           printf("%c\t=",pr[i].l);
                           printf("%s\n",pr[i].r); }
                           for(i=0;i<z;i++)
                           {
                              for(j=i+1;j<z;j++)
                           {
                              q=strcmp(pr[i].r,pr[j].r);
                              if((pr[i].l==pr[j].l) && !q)
                              pr[i].l='\0';
                              }
                              }
                              printf("Optimized Code\n");
                              for(i=0;i<z;i++)
                              {
                                 if(pr[i].l!='\0')
                                 {
                                    printf("%c=",pr[i].l);
                                    printf("%s\n",pr[i].r);
                                    }

                                 }

}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of values: 5
Left: a
Right: 9
Left: b
Right: c+d
Left: e
Right: c+d
Left: f
Right: b+e
Left: r
Right: f
Intermediate Code
a=9
b=c+d
e=c+d
f=b+e
r=f
After Dead Code Elimination
b =c+d
e =c+d
f =b+e
r =f
pos: 2
Eliminate Common Expression
b =c+d
b =c+d
f =b+b
r =f
Optimized Code
b=c+d
f=b+b
r=f

Test Case - 2
User Output
Enter the number of values: 4
Left: c
Right: e+d
Left: a
Right: c+d
Left: g
Right: e+f
Left: v
Right: s+t
Intermediate Code
c=e+d
a=c+d
g=e+f

Test Case - 2
v=s+t
After Dead Code Elimination
c =e+d
v =s+t
Eliminate Common Expression
c =e+d
v =s+t
Optimized Code
c=e+d
v=s+t

S.No: 15	Exp. Name: ***Implementation of code generator***	Date:

Aim:

Write a C program to implement **Code Generator**.
The name of the input file is supplied as a command-line argument. Use `printf` to print the generated code to the standard output.

Source Code:

codeGenerator.c

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
char op[2],arg1[5],arg2[5],result[5];
int main(int argc,char *argv[]){
   FILE *fp1;  fp1=fopen(argv[1],"r");
   while(!feof(fp1)) {
      fscanf(fp1,"%s%s%s%s",op,arg1,arg2,result);
      if(strcmp(op,"+")==0)
      {        printf("MOV R0,%s\n",arg1);
      printf("ADD R0,%s\n",arg2);
      printf("MOV %s,R0\n",result);
      }     if(strcmp(op,"*")==0)
      {        printf("MOV R0,%s\n",arg1);
      printf("MUL R0,%s\n",arg2);
      printf("MOV %s,R0\n",result);
      }     if(strcmp(op,"-")==0)
      {        printf("MOV R0,%s\n",arg1);
      printf("SUB R0,%s\n",arg2);
      printf("MOV %s,R0\n",result);
      }
      if(strcmp(op,"/")==0)
      {
         printf("MOV R0,%s\n",arg1);
         printf("DIV R0,%s\n",arg2);
         printf("MOV %s,R0\n",result);
      }
      if(strcmp(op,"=")==0)
      {
            printf("MOV R0,%s\n",arg1);
            printf("MOV %s,R0\n",result);
            }
         }
   fclose(fp1);

}
```

i1.txt

```
+ a b t1
* c d t2
```

```
- t1 t2 t
= t ? x
```

i2.txt

```
* c d t2
/ a b t1
+ t1 t2 t3
= t3 ? x
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
MOV R0,a
ADD R0,b
MOV t1,R0
MOV R0,c
MUL R0,d
MOV t2,R0
MOV R0,t1
SUB R0,t2
MOV t,R0
MOV R0,t
MOV x,R0