DEPARTMENT OF COMPUTER SCIENCE & TECHNOLOGY

LAB FILE
FOR

SUBJECT: COMPILER DESIGN LAB
SUBJECT CODE: CSP353

SUBMITTED BY: ADITI CHANDRA
SYSTEM ID: 2018002388
ROLL NUMBER: 180101020

DR. LATHA BANDA
ASSISTANT PROFESSOR (CSE)

DEPARTMENT OF COMPUTER SCIENCE & TECHNOLOGY
SHARDA UNIVERSITY, GREATER NOIDA

S. No	Date	Topic	Page No
1	19/01/21	**Practical 01:** Design a DFA which will accept all the strings containing even number of 0's over an alphabet {0, 1} and write a program to implement the DFA.	02
2	19/01/21	**Practical 02:** Design a DFA which will accept all the strings containing odd number of 1's over an alphabet {0, 1} and write a program to implement the DFA.	03
3	02/02/21	**Practical 03:** Design a DFA which will accept all the strings ending with 00 over an alphabet {0, 1} and write a program to implement the DFA.	04
4	09/02/21	**Practical 04:** Design a DFA which will accept all the strings containing mod 3 of 0's over an alphabet {0, 1} and write a program to implement the DFA.	05
5	09/02/21	**Practical 05:** To convert regular expression into NFA.	07
6	23/02/21	**Practical 06:** To find string is keyword, identifier and constant or not.	09
7	23/02/21	**Practical 07:** To check the balanced parenthesis of an expression.	10
8	09/03/21	**Practical 08:** To calculate leading for all non-terminals.	12
9	09/03/21	**Practical 09:** To calculate trailing for all non-terminals.	14
10	16/03/21	**Practical 10:** Write an algorithm and program to compute FIRST function.	16
11	23/03/21	**Practical 11:** Write an algorithm and program to compute FOLLOW function.	18
12	30/03/21	**Practical 12:** Write an algorithm and program on Recursive Descent parser.	20
13	06/04/21	**Practical 13:** To write a C program to construct of DAG (Directed Acyclic Graph).	22
14	13/04/21	**Practical 14:** To write a C program to implement simple code optimization technique.	23
15	20/04/21	**Practical 15:** Write a C program to recognize strings under 'a*', 'a*b+', 'abb'.	25

Practical No 1: Design a DFA which will accept all the strings containing even number of 0's over an alphabet {0, 1} and write a program to implement the DFA.
Date: 19 January 2021

Code:

```
#include<stdio.h>
#define max 100
int main()
{
 char str[max],f='x',c;
 printf("do you want to check for epsilon string's case? (y/n) : ");
 scanf("%c",&c);
 if(c=='y'|| c=='Y')
 goto flag;

 printf("enter the string to be checked: ");
 scanf("%s",str);
 int i;
 for(i=0;i<strlen(str);i++)
 {
  switch(f)
  {
   case 'x': if(str[i]=='0') f='y';
    else if(str[i]=='1') f='x';
   break;
   case 'y': if(str[i]=='0') f='x';
    else if(str[i]=='1') f='y';
   break;
  }
 }
flag: if(f=='x') printf("\nString is accepted as it reaches the final state that is %c",f);
 else printf("\nString is not accepted as it reaches a state %c which is not the final state",f);

return 0;
}
```

Output:

```
do you want to check for epsilon string's case? (y/n) : n
enter the string to be checked: 00100

String is accepted as it reaches the final state that is x
```

Practical No 2: Design a DFA which will accept all the strings containing odd number of 1's over an alphabet {0, 1} and write a program to implement the DFA.
Date: 19 January 2021

Code:

```
#include<stdio.h>
#define max 100
main()
{
 char str[max],f='x';
 int i;
 printf("enter the string to be checked: ");
 scanf("%s",str);
 for(i=0;i<strlen(str);i++)
 {
  switch(f)
  {
   case 'x': if(str[i]=='0') f='x';
    else if(str[i]=='1') f='y';
   break;
   case 'y': if(str[i]=='0') f='y';
    else if(str[i]=='1') f='x';
   break;
  }
 }
 if(f=='y') printf("\nString accepted. State reached is: %c",f);
 else printf("\nstring not accepted. State reached is: %c",f);
}
```

Output:

```
enter the string to be checked: 1100110
string not accepted. State reached is: x
```

Practical No 3: Design a DFA which will accept all the strings ending with 00 over an alphabet {0, 1} and write a program to implement the DFA.
Date: 02 February 2021

Code:

```
#include<stdio.h>
#define max 100
main()
{
 char str[max],f='a';
 int i;
 printf("enter the string to be checked: ");
 scanf("%s",str);
 for(i=0;str[i]!='\0';i++)
 {
  switch(f)
  {
   case 'a': if(str[i]=='0') f='b';
    else if(str[i]=='1') f='a';
   break;
   case 'b': if(str[i]=='0') f='c';
    else if(str[i]=='1') f='d';
   break;
   case 'c': if(str[i]=='0') f='c';
    else if(str[i]=='1') f='d';
   break;
   case 'd': if(str[i]=='0') f='b';
    else if(str[i]=='1') f='d';
   break;
  }
 }
 if(f=='c') printf("\nString is accepted as it reached the final state %c at the end.",f);
 else printf("\nString is not accepted as it reached %c which is not the final state.",f);
}
```

Output:

```
enter the string to be checked: 1100
String is accepted as it reached the final state c at the end.
```

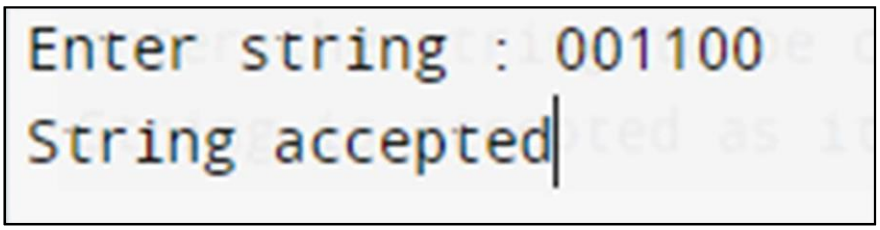
Practical No 4: Design a DFA which will accept all the strings containing mod 3 of 0's over an alphabet {0, 1} and write a program to implement the DFA.
Date: 09 February 2021

Code:

```
#include <stdio.h>
#include <string.h>
char sub[100];
void  funA(char*);
void  funB(char*);
void funC(char*);
char* substring(char* str ,int pos)
{
  int length=strlen(str), c = 0;
  while (c < length)
  {
    sub[c] = str[pos+c];
    c++;
  }
  sub[c] = '\0';
  return sub;
}
void funA(char* a)
{
   if(strlen(a)==0)
  {
      printf("String accepted");
   }
   else
   {
      if (a[0] == '1')
         funA(substring(a,1));
      else
         funB(substring(a,1));
   }
}
void funB(char* a)
{
   if(strlen(a)==0)
   {
      printf("String not accepted");
   }
   else
   {
      if (a[0] == '1')
         funB(substring(a,1));
      else
         funC(substring(a,1));
   }
}
void funC(char* a)
```

```
{
   if(strlen(a)==0)
  {
      printf("String not accepted");
   }
   else
   {
      if (a[0] == '1')
         funC(substring(a,1));
      else
         funA(substring(a,1));
   }
}
int main()
{
   char s[100];
   printf("Enter string : ");
   for(int i=0;i<100;i++)
  {
      scanf("%c",&s[i]);
      if(s[i]='\n')
    {
         break;
      }
   }
   funA(s);
   return 0;
}
```

Output:



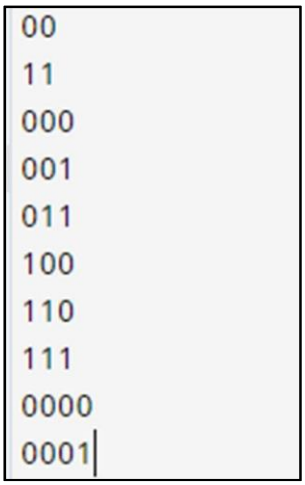
Practical No 5: To convert regular expression into NFA.
Date: 09 February 2021

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include  <math.h>
int row = 0;
{
int data; struct node* next; char edgetype;
}
typedef node;
{
node* new_node = (node*)malloc(sizeof(node)); new_node->edgetype = edgetype;
new_node->data = data; new_node->next = NULL; if (first==NULL)
{
first = new_node; return new_node;
}
first->next = push(first->next,edgetype,data); return first;
}
{
if (start==(int)strlen(input)) return accept[current];
node* temp = graph[current]; while (temp != NULL)
{
if (input[start]==temp->edgetype) if (nfa(graph,temp->data,input,accept,start+1==1)
return 1; temp=temp->next;
}
return 0;
}
{
if (size==0)
{
strcpy(arr[row], a); row++; return;
}
char b0[20] = {'\0'}; char b1[20] = {'\0'}; b0[0] = '0'; b1[0] = '1';
generate((char**)arr, size-1, strcat(b0,a));
}
{
int n; int i, j;
scanf("%d", &n);
for (i=0;i<n+1;i++) graph[i]=NULL;
int accept[n+1];
for (i=0; i<n; i++)
{
for (j=0;j<number_nodes;j++)
{
int node_add; int edge;
scanf("%d%d",&edge,&node_add); graph[index] = push(graph[index],'0'+edge,node_add);
}
```

```
}
int size = 1;
if (accept[1]==1)
{
printf("e\n");
count++;
}
while (count < 11)
{
char** arr; int power = pow(2,size); arr = (char**)malloc(power*sizeof(char*));
for (i=0;i<power;i++) arr[i] = (char*)malloc(size*sizeof(char));
char a[20] = {'\0'};
generate((char**)arr,size,a);
for (i=0; i<power; i++)
{
char input[20] = {'\0'};
for (j=0; j<size; j++)
{
char foo[2]; foo[0] = arr[i][size-1-j]; foo[1] = '\0'; strcat(input,foo);
}
int result = nfa(graph,1,input,accept,0);
if (result==1)
{
printf("%s\n",input);
}
if (count==10) return 0;
}
size++;
}
return 0;
}
```

Output:



Practical No 6: To find string is keyword, identifier and constant or not.
Date: 23 February 2021

Code:

```
#include<stdio.h>
#include<string.h>
void main()
{
   int i,flag=0,m;
   char s[5][10]={"if","else","goto","continue","return"},st[100];
   printf("\n enter the string :");
   gets(st);
   for(i=0;i<5;i++)
  {
      m=strcmp(st,s[i]);
      if(m==0)
      flag=1;
      if (isdigit(st[a]))
      flag=2;
      if(flag==0)
      printf("\n it is an identifier");
      else
      {
         if(flag==1)
         {
            printf("\n it is a keyword");
         }
         else
         {
            printf("\n it is a constant");
         }
      }
   }
}
```

Output:

```
enter the string : else
 it is a keyword
```

Practical No 7: To check the balanced parenthesis of an expression.
Date: 23 February 2021

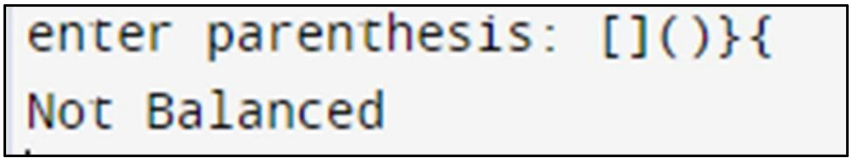
Code:

```
#include<stdio.h>
#include<stdlib.h>
#define bool int
struct sNode
{ char data; struct sNode *next;
};
void push(struct sNode** top_ref, int new_data);
int pop(struct sNode** top_ref);
bool isMatchingPair(char character1, char character2)
{
if (character1 == '(' && character2 == ')') return 1;
else if (character1 == '{' && character2 == '}') return 1;
else if (character1 == '[' && character2 == ']') return 1; else return 0;
}
bool areParenthesisBalanced(char exp[])
{ int i = 0;
struct sNode *stack = NULL;
while (exp[i])
{
if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[') push(&stack, exp[i]);
if (exp[i] == '}' || exp[i] == ')' || exp[i] == ']')
{
if (stack == NULL) return 0;
else if ( !isMatchingPair(pop(&stack), exp[i]) ) return 0;
} i++;
}
if (stack == NULL)
return 1; else
return 0;
}
int main()
{
char exp[100] = "{()}[]";
printf("\nenter parenthesis: ");
gets(exp);
if (areParenthesisBalanced(exp)) printf("Balanced \n"); else
printf("Not Balanced \n"); return 0;
}
void push(struct sNode** top_ref, int new_data)
{
struct sNode* new_node =
(struct sNode*) malloc(sizeof(struct sNode));
if (new_node == NULL)
{
printf("Stack overflow n"); getchar(); exit(0);
}
new_node->data = new_data;
```

```
new_node->next = (*top_ref);

(*top_ref) = new_node;
}
int pop(struct sNode** top_ref)
{ char res; struct sNode *top;
if (*top_ref == NULL)
{
printf("Stack overflow n"); getchar();
exit(0); } else
top = *top_ref; res = top->data; *top_ref = top->next; free(top); return res;
}
```

Output:



Practical No 8: To calculate leading for all non-terminals.
Date: 09 March 2021

Code:

```
#include<conio.h>
#include<stdio.h>
char arr[18][3] ={{'E', '+', 'F'},{'E', '*', 'F'},{'E', '(', 'F'}, {'E', ')', 'F'},{'E', 'i', 'F'},{'E', '$',
'F'},
{'F', '+', 'F'},{'F', '*', 'F'},{'F', '(', 'F'},{'F', ')', 'F'},{'F', 'i', 'F'},{'F', '$', 'F'}, {'T', '+', 'F'},
{'T', '*', 'F'}, {'T', '(', 'F'},{'T', ')', 'F'},{'T', 'i', 'F'},{'T', '$', 'F'}};
char prod[6] = "EETTFF";
char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0'}, {'T', '*', 'F'}, {'F', '\0'}, {'(', 'E', ')'}, {'i', '\0'}};
char stack [5][2];
int top = -1;
void install(char pro, char re) {
int i;
for (i = 0; i < 18; ++i) { if (arr[i][0] == pro && arr[i][1] == re) {
arr[i][2] = 'T'; break;
}
}
++top; stack[top][0] = pro; stack[top][1] = re;
}
void main() {
int i = 0, j; char pro, re, pri = ' '; clrscr();
for (i = 0; i < 6; ++i) { for (j = 0; j < 3 && res[i][j] != '\0'; ++j) {
if (res[i][j] == '+' || res[i][j] == '*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' ||
res[i][j] ==
'$') { install(prod[i], res[i][j]); break;
}
}
}
while (top >= 0) { pro = stack[top][0]; re = stack[top][1]; --top; for (i = 0; i < 6; ++i) { if
(res[i][0] == pro && res[i][0] != prod[i]) {
install(prod[i], re);
}
}
}
for (i = 0; i < 18; ++i) { printf("\n\t"); for (j = 0; j < 3; ++j) printf("%c\t", arr[i][j]);
}
getch(); clrscr();
printf("\n\n"); for (i = 0; i < 18; ++i) { if (pri != arr[i][0]) { pri = arr[i][0]; printf("\n\t%c ->
", pri);
}
if (arr[i][2] == 'T') printf("%c ", arr[i][1]);
}
getch();
}
```

Output:

```
E        +        T
E        *        T
E        (        T
E        )        F
E        i        T
E        $        F
F        +        F
F        *        F
F        (        T
F        )        F
F        i        T
F        $        F
T        +        F
T        *        T
T        (        T
T        )        F
T        i        T
T        $        F

E -> + * ( i
F -> ( i
T -> * ( i
```

Practical No 9: To calculate trailing for all non-terminals.
Date: 09 March 2021

Code:

```
#include<conio.h>
#include<stdio.h>
char arr[18][3] ={{'E', '+', 'F'}, {'E', '*', 'F'}, {'E', '(', 'F'}, {'E', ')', 'F'}, {'E', 'i', 'F'},
{'E', '$', 'F'}, {'F', '+', 'F'}, {'F', '*', 'F'}, {'F', '(', 'F'}, {'F', ')', 'F'}, {'F', 'i', 'F'},
{'F', '$', 'F'}, {'T', '+', 'F'}, {'T', '*', 'F'}, {'T', '(', 'F'}, {'T', ')', 'F'}, {'T', 'i', 'F'},
{'T', '$', 'F'},
};
char prod[6] = "EETTFF";
char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0', '\0'}, {'T', '*', 'F'}, {'F', '\0', '\0'}, {'(', 'E', ')'}, {'i', '\0',
'\0'},};
char stack [5][2]; int top = -1;
void install(char pro, char re)
{
int i;
for (i = 0; i < 18; ++i) { if (arr[i][0] == pro && arr[i][1] == re)
{
}
}
++top; arr[i][2] = 'T'; break;
stack[top][0] = pro; stack[top][1] = re;
}
void main()
{
int i = 0, j; char pro, re, pri = ' ';
clrscr(); for (i = 0; i < 6; ++i) { for (j = 2; j >= 0; --j)
{
if (res[i][j] == '+' || res[i][j] == '*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' ||
res[i][j] =='$') { install(prod[i], res[i][j]);
break;
}
else if (res[i][j] == 'E' || res[i][j] == 'F' || res[i][j] == 'T') { if (res[i][j - 1] == '+' ||
res[i][j - 1] == '*' || res[i][j - 1] == '(' || res[i][j -1] == ')' || res[i][j - 1] == 'i' || res[i][j -
1] == '$') { install(prod[i], res[i][j - 1]);
break;
}
}
}
}
while (top >= 0) { pro = stack[top][0]; re = stack[top][1];
--top;
for (i = 0; i < 6; ++i) { for (j = 2; j >= 0; --j) { if (res[i][0] == pro && res[i][0] !=prod[i])
{
install(prod[i], re); break;
}
else if (res[i][0] != '\0');
break;
}
```

```
}
}
for (i = 0; i < 18; ++i) { printf("\n\t");
for (j = 0; j < 3; ++j)
printf("%c\t", arr[i][j]);
}
getch();
clrscr();
printf("\n\n");
for (i = 0; i < 18; ++i) { if (pri != arr[i][0]) { pri = arr[i][0]; printf("\n\t%c -> ", pri);
}
if (arr[i][2] == 'T') printf("%c ", arr[i][1]);}
getch();
}
```

Output:

```
E       +       T
E       *       T
E       (       F
E       )       T
E       i       T
E       $       F
F       +       F
F       *       F
F       (       F

F       )       T
F       i       T
F       $       F
T       +       F
T       *       T
T       (       F
T       )       T
T       i       T
T       $       F

E -> + * ) i
F -> ) i
T -> * ) i
```

Practical No 10: Write an algorithm and program to compute FIRST function.
Date: 16 March 2021

Code:

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[10][10];
main()
{
int i;
char choice; char c; char result[20];
printf("How many number of productions ? :");
scanf(" %d",&numOfProductions);
for(i=0;i<numOfProductions;i++)
{
   printf("Enter productions Number %d : ",i+1);
   scanf(" %s",productionSet[i]);
} do {
printf("\n Find the FIRST of :"); scanf(" %c",&c);
FIRST(result,c);
printf("\n FIRST(%c)= { ",c); for(i=0;result[i]!='\0';i++) printf(" %c ",result[i]); /*Display
result*/ printf("}\n"); printf("press 'y' to continue : "); scanf(" %c",&choice);
}
while(choice=='y'||choice =='Y');
}
void FIRST(char* Result,char c)
{ int i,j,k;
char subResult[20]; int foundEpsilon;
subResult[0]='\0';
Result[0]='\0';
if(!(isupper(c)))
{
addToResultSet(Result,c);
return ;
}
for(i=0;i<numOfProductions;i++)
{
 if(productionSet[i][0]==c)
{
if(productionSet[i][2]=='#') addToResultSet(Result,'#');
else { j=2;
while(productionSet[i][j]!='\0')
{
foundEpsilon=0;
FIRST(subResult,productionSet[i][j]);
for(k=0;subResult[k]!='\0';k++) addToResultSet(Result,subResult[k]);
for(k=0;subResult[k]!='\0';k++) if(subResult[k]=='#')
{
```

```
foundEpsilon=1; break;
}
if(!foundEpsilon) break; j++; }
}
} } return ;
}
void addToResultSet(char Result[],char val)
{ int k;
for(k=0 ;Result[k]!='\0';k++) if(Result[k]==val)
return;
Result[k]=val;
Result[k+1]='\0';
}
```

Output:

```
How many number of productions ? :8
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=i

Find the FIRST of :E

FIRST(E)= {  <  i }
press 'y' to continue : y

Find the FIRST of :T

FIRST(T)= {  <  i }
press 'y' to continue : y

Find the FIRST of :F

FIRST(F)= {  <  i }
press 'y' to continue : y

Find the FIRST of :X

FIRST(X)= {  +  # }
press 'y' to continue : y

Find the FIRST of :Y

FIRST(Y)= {  *  # }
press 'y' to continue :
```

Practical No 11: Write an algorithm and program to compute FOLLOW function.
Date: 23 March 2021

Algorithm:

1) FOLLOW(S) = { $ } // where S is the starting Non-Terminal

2) If A -> pBq is a production, where p, B and q are any grammar symbols, then everything in FIRST(q) except Є is in FOLLOW(B).

3) If A -> pB is a production, then everything in FOLLOW(A) is in FOLLOW(B).

4) If A -> pBq is a production and FIRST(q) contains Є, then FOLLOW(B) contains { FIRST(q) – Є } U FOLLOW(A)

Code:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int n,m=0,p,i=0,j=0;
char a[10][10],followResult[10];
void follow(char c);
void first(char c);
void addToResult(char);
int main()
{
int i; int choice; char c,ch;
printf("Enter the no.of productions: "); scanf("%d", &n);
printf(" Enter %d productions\nProduction with multiple terms should be give as separate
productions \n", n); for(i=0;i<n;i++) scanf("%s%c",a[i],&ch); do
{
m=0;
printf("Find FOLLOW of -->"); scanf(" %c",&c); follow(c);
printf("FOLLOW(%c) = { ",c); for(i=0;i<m;i++)
printf("%c ",followResult[i]);
printf(" }\n");
printf("Do you want to continue(Press 1 to continue....)?"); scanf("%d",&choice);
}
while(choice==1);
}
void follow(char c)
{
   if(a[0][0]==c)addToResult('$'); for(i=0;i<n;i++)
{
for(j=2;j<strlen(a[i]);j++)
{
if(a[i][j]==c)
{
if(a[i][j+1]!='\0')first(a[i][j+1]); if(a[i][j+1]=='\0'&&c!=a[i][0]) follow(a[i][0]);
}
}
```

```
}
}
void first(char c)
{ int k;
if(!(isupper(c))) /*f[m++]=c; */ addToResult(c);
for(k=0;k<n;k++)
{ if(a[k][0]==c)
{
if(a[k][2]=='#') follow(a[i][0]); else if(islower(a[k][2])) /*f[m++]=a[k][2]; */
addToResult(a[k][2]);
else first(a[k][2]);
}
}
}
void addToResult(char c)
{
int i;
for( i=0;i<=m;i++) if(followResult[i]==c)
return;
followResult[m++]=c;
}
```

Output:

```
Enter the no.of productions: 3
Enter 3 productions
Production with multiple terms should be give as separate productions
S=CC
C=aC
C=d
Find FOLLOW of -->S
FOLLOW(S) = { $  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->C
FOLLOW(C) = { a d $  }
Do you want to continue(Press 1 to continue....)?0
```

Practical No 12: Write an algorithm and program on Recursive Descent parser.
Date: 30 March 2021

Algorithm:

```
for a = 1 to limit
    for b = 1 to Array - 1 do begin
        Increase Production: Array[a] ->Array[b]
    end for
    Remove Immediate Left recursion from A[a]
End for
```

Code:

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

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

void T()
{
    check();
    Tprime();
}
```

```
void Tprime()
{
    if(expression[count] == '*')
    {
        count++;
        check();
        Tprime();
    }
}

void check()
{
    if(isalnum(expression[count]))
    {
        count++;
    }
    else if(expression[count] == '(')
    {
        count++;
        E();
        if(expression[count] == ')')
        {
            count++;
        }
        else
        {
            flag = 1;
        }
    }
    else
    {
        flag = 1;
    }
}
void Eprime()
{
    if(expression[count] == '+')
    {
        count++;
        T();
        Eprime();
    }
}
```

Output:

```
Enter an Algebraic Expression:  (a+b)*c
The Expression (a+b)*c is Valid
```

Practical No 13: To write a C program to construct of DAG (Directed Acyclic Graph).
Date: 06 April 2021

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MIN_PER_RANK 1
#define MAX_PER_RANK 5
#define MIN_RANKS 3
#define MAX_RANKS 5
#define PERCENT 30
void main()
{
int i,j,k,nodes=0;
srand(time(NULL));
int ranks=MIN_RANKS+(rand()%(MAX_RANKS-MIN_RANKS+1)); printf("DIRECTED
ACYCLIC GRAPH\n"); for(i=1;i<ranks;i++)
{
   int new_nodes=MIN_PER_RANK+(rand()%(MAX_PER_RANK-MIN_PER_RANK+1));
for(j=0;j<nodes;j++) for(k=0;k<new_nodes;k++) if((rand()%100)<PERCENT) printf("%d-
>%d;\n",j,k+nodes); nodes+=new_nodes;
}
}
```

Output:

```
DIRECTED ACYCLIC GRAPH
0->3;
0->4;
1->3;
1->5;
2->3;
2->4;
4->6;
5->6;
```

Practical No 14: To write a C program to implement simple code optimization technique.
Date: 13 April 2021

Code:

```
#include<stdio.h>
#include<string.h>
struct op { char l; char r[20];
}
op[10],pr[10];
void main()
{
int a,i,k,j,n,z=0,m,q; char *p,*l; char temp,t; char *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("\nAfter 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);
}
}
}
```

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
```

Practical No 15: Write a C program to recognize strings under 'a*', 'a*b+', 'abb'.
Date: 20 April 2021

Code:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
char  s[20],c;
int state=0,i=0;
printf("\n Enter a string:");
gets(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("\n %s is not recognised.",s);
exit(0);
}
}
if(state==1)
printf("\n %s is accepted under rule 'a'",s);
else if((state==2)||(state==4))
printf("\n %s is accepted under rule 'a*b+'",s);
else if(state==5)
printf("\n %s is accepted under rule 'abb'",s);
}
```

Output: