RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution

Affiliated to ANNA UNIVERSITY, Chennai

25 years OF ACADEMIC EXCELLENCE

DEPARTMENT OF INFORMATION TECHNOLOGY

Laboratory Manual

REGULATION 2023

CS23231 - DATA STRUCTURES

RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institution, Affiliated to Anna University
Rajalakshmi Nagar, Thandalam – 602 105

DEPARTMENT OF INFORMATION TECHNOLOGY

CS23231 – DATA STRUCTURES

(***Regulation 2023***)

LAB MANUAL

Name : . . . . . .

Register No. : . . . . . .

Year / Branch / Section : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

LESSON PLAN

Course Code	Course Title (Laboratory Integrated Theory Course)	L	T	P	C
CS23231	Data Structures	3	0	4	5

LIST OF EXPERIMENTS	
Sl. No	**Name of the experiment**
Week 1	Implementation of Single Linked List (Insertion, Deletion and Display)
Week 2	Implementation of Doubly Linked List (Insertion, Deletion and Display)
Week 3	Applications of Singly Linked List (Polynomial Manipulation)
Week 4	Implementation of Stack using Array and Linked List implementation
Week 5	Applications of Stack (Infix to Postfix)
Week 6	Applications of Stack (Evaluating Arithmetic Expression)
Week 7	Implementation of Queue using Array and Linked List implementation
Week 8	Implementation of Binary Search Tree
Week 9	Performing Tree Traversal Techniques
Week 10	Implementation of AVL Tree
Week 11	Performing Topological Sorting
Week 12	Implementation of BFS, DFS
Week 13	Implementation of Prim's Algorithm
Week 14	Implementation of Dijkstra's Algorithm
Week 15	Program to perform Sorting
Week 16	Implementation of Open Addressing (Linear Probing and Quadratic Probing)
Week 17	Implementation of Rehashing

INDEX

Note: Students have to write the Algorithms at left side of each problem statements.

Ex. No.:	Implementation of Single Linked List	Date:

Write a C program to implement the following operations on Singly Linked List.

(i) Insert a node in the beginning of a list.

(ii) Insert a node after P

(iii) Insert a node at the end of a list

(iv) Find an element in a list

(v) FindNext

(vi) FindPrevious

(vii) isLast

(viii) isEmpty

(ix) Delete a node in the beginning of a list.

(x) Delete a node after P

(xi) Delete a node at the end of a list

(xii) Delete the List

Algorithm:

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{ int ele;
  struct Node *next;};
typedef struct Node node;
int isempty(node *list);
node *find(int e,node *list);
node *findprev(node *list,int x);
node *delmid(node *list,int x);

int isempty(node *list)
{ if (list==NULL)
    return 1;
  else
   return 0;}
node * insertbeg(node*list,int x)
{ node*newnode=malloc(sizeof(node));
  newnode->ele=x;
  newnode->next=NULL;
  if (isempty(list))
  { list=newnode;
    return list;}
  newnode->next=list;

  return newnode;
}
void insertmid(node *list,int p,int x)
{ node *posn=find(p,list);
  node *newnode=malloc(sizeof(node));
  if(posn==NULL)
```

```
  return;
  newnode->next=posn->next;
  posn->next=newnode;
  newnode->ele=x;}

node * insertlast(node *list,int x)
{ node *newnode=malloc(sizeof(node));
 newnode->ele=x;
 newnode->next=NULL;
 node*temp=list;
 while(temp->next!=NULL)
    temp=temp->next;
 temp->next=newnode;
 return list;
}
node *delbeg(node *list)
{  printf("The deleted elem is %d\n",list->ele);
  return list->next;}

void display(node * list)
{ node *temp=list;
 while(temp!=NULL)
   {printf("%d ",temp->ele);
   temp=temp->next;}
}
node *find(int x,node *list)
{ node *temp=list;
 while(temp!=NULL)
 { if (temp->ele==x)
      return temp;
  temp=temp->next;}
 return temp;}

node *dellast(node *list)
{ node *temp=list;
 while(temp->next!=NULL)
    temp=temp->next;
 if(temp==list)
    return list->next;
 node *pre=findprev(list,temp->ele);
 pre->next=NULL;
 return list;}
node *findprev(node *list,int x)
{ node *temp=list;
 node *prev=NULL;
 while(temp!=NULL)
 {  if(temp->ele==x)
      return prev;
  prev=temp;
  temp=temp->next;}
  return prev;}

 node *delmid(node *list,int x)
 { node *f=find(x,list);
  if(f==NULL)
  {printf("Not found");
```

```
return list;
  }

  node *prev=findprev(list,x);
  if (prev==NULL)
  { list=NULL;
   return list;}
  prev->next=f->next;
  free(f);
  return list;
  //free(f);
 }
int main()
{
   node *list=NULL;


   printf("%d\n",isempty(list));

   list=insertbeg(list,5);

   list=insertbeg(list,15);
   list=insertbeg(list,115);
   list=insertbeg(list,1115);
   list=insertbeg(list,50);
   display(list);
   printf("\n");
   insertmid(list,115,34);
  // list=delbeg(list);
   display(list);
   list=dellast(list);
   printf("\n");
   display(list);
   list=delmid(list,1115);
   printf("\n");
   display(list);
   printf("\n");
   list=insertlast(list,800);
   display(list);
   list=delbeg(list);
   display(list);




   return 0;
}
```

OUTPUT:-

```
50 1115 115 15 5
50 1115 115 34 15 5
50 1115 115 34 15
50 115 34 15
50 115 34 15 800 The deleted elem is 50
115 34 15 800
```

Ex. No.:	Implementation of Doubly Linked List	Date:

Write a C program to implement the following operations on Doubly Linked List.

(i) Insertion

(ii) Deletion

(iii) Search

(iv) Display

Algorithm:

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{ int ele;
  struct Node *prev;
  struct Node *next;};
typedef struct Node node;
//int isempty(node *list);
//node *find(int e,node *list);
int isempty(node *list)
{ if (list==NULL)
    return 1;
  return 0;}
node *find(int x,node *list)
{ node *temp=list;
  while(temp!=NULL)
  { if (temp->ele==x)
      return temp;
    temp=temp->next;}
  return temp;}
node * insertbeg(node*list,int x)
{ node*newnode=malloc(sizeof(node));
  newnode->ele=x;
  newnode->prev=NULL;
  newnode->next=NULL;
  if (isempty(list))
  { list=newnode;
    return list;}
  newnode->next=list;
  list->prev=newnode;
  return newnode;
}

void insertmid(node *list,int p,int x)
{ node *f=find(p,list);
  if (isempty(list))
   {   printf("CANT INSERT\n");
      return;}
  node *newnode=malloc(sizeof(node));
  newnode->ele=x;
```

```
 f->next->prev=newnode;
 newnode->next=f->next;
 newnode->prev=f;
 f->next=newnode;
}

void insertlast(node *list,int x)
{ node *newnode=malloc(sizeof(node));
 newnode->ele=x;
 newnode->next=NULL;
 if (isempty(list))
 { printf("EMPTY\n");
  return;}
 node*temp=list;
 while(temp->next!=NULL)
    temp=temp->next;
 newnode->prev=temp;
 newnode->next=NULL;
 temp->next=newnode;
 //return list;
}

node *delbeg(node *list)
{  if(isempty(list))
  { printf("EMPTY\n");
   return list;}
  if(list->next==NULL)
     return NULL;
  printf("The deleted elem is %d\n",list->ele);
  list->next->prev=NULL;
  return list->next;
}


node *delmid(node *list,int x)
{ node *temp=find(x,list);
  if(temp==NULL)
  { printf("CANT DELETE");
   return list;}
  temp->next->prev=temp->prev;
  temp->prev->next=temp->next;
  free(temp);
  return list;
  //free(f);
}

void dellast(node *list)
{ if(isempty(list))
  { printf("EMPTY\n");
   return;}
  node *temp=list;
  while(temp->next!=NULL)
      temp=temp->next;
  printf("The deleted elem is %d\n",temp->ele);
  temp->prev->next=NULL;
  free(temp);
}

void display(node * list)
```

```
{ node *temp=list;
  while(temp!=NULL)
    {printf("%d ",temp->ele);
    temp=temp->next;}
}

 int main()
{
    node *list=NULL;
    node *posn=NULL;
    printf("%d\n",isempty(list));
    list=insertbeg(list,5);
    insertlast(list,15);
    insertlast(list,115);
    insertlast(list,1115);
    list=insertbeg(list,50);
    display(list);
    printf("\n");
    insertmid(list,115,34);
    display(list);
    printf("\n");
    dellast(list);
    display(list);
    printf("\n");
    list=delmid(list,15);
    display(list);
    printf("\n");
    insertlast(list,800);
    display(list);
    printf("\n");
    list=delbeg(list);
    display(list);
    printf("\n");
    posn=find(800,list);
    if(posn!=NULL)
      printf("1");
    return 0;
}
```

OUTPUT:-

```
50 5 15 115 1115
50 5 15 115 34 1115
The deleted elem is 1115
50 5 15 115 34
50 5 115 34
50 5 115 34 800
The deleted elem is 50
5 115 34 800
```

Ex. No.:	Polynomial Manipulation	Date:

Write a C program to implement the following operations on Singly Linked List.

(i) Polynomial Addition

(ii) Polynomial Subtraction

(iii) Polynomial Multiplication

Algorithm:

Code:

```
//poly add
#include <stdio.h>
#include <stdlib.h>
struct Poly
{ int ele;
 int p;
 struct Poly*next;}*l1=NULL,*l2=NULL,*l3=NULL;
int coeff,power,ch=1;
typedef struct Poly poly;
void display(poly *list);
void create(poly **list)
{  do{
   poly *n1=malloc(sizeof(poly));
   printf("Enter coeff :");
   scanf("%d",&n1->ele);
   printf("Enter power :");
   scanf("%d",&n1->p);
   n1->next=NULL;
   if(*list==NULL)
    *list=n1;
    else{
    poly *temp=*list;
    while(temp->next!=NULL)
      temp=temp->next;
    temp->next=n1;}
    printf("To continue 1 otherwise 0");
    scanf("%d",&ch);
    }while(ch==1);
}

void add()
{ poly *temp1=l1;
  poly *temp2=l2;
  poly *temp=NULL;
  while((temp1!=NULL) && (temp2!=NULL))
  { poly *newnode=(poly*)malloc(sizeof(poly));
    newnode->next=NULL;
    if(temp1->p > temp2->p)
    {  newnode->ele=temp1->ele;
      newnode->p=temp1->p;
      temp1=temp1->next;
    }
```

```
   else if( (temp1->p) < (temp2->p))


   {  newnode->ele=temp2->ele;
      newnode->p=temp2->p;
      temp2=temp2->next;
   }

   else if(temp1->p == temp2->p)
   {  newnode->ele=temp1->ele+temp2->ele;
      newnode->p=temp1->p;
      temp1=temp1->next;
      temp2=temp2->next;
   }

    if (l3 == NULL) {
          l3 = newnode;
          temp = l3;
       } else {
          //temp=l3;
          //while(temp->next!=NULL)
              // temp = temp->next;
          temp->next=newnode;
          temp=newnode;
       }
  }

   while(temp1!=NULL || temp2!=NULL)
 {   poly *newnode=malloc(sizeof(poly));
     newnode->next=NULL;
     if(temp1!=NULL)
     {newnode->ele=temp1->ele;
      newnode->p=temp1->p;
      temp->next=newnode;
      temp=temp->next;
      temp1=temp1->next;}

     else if(temp2!=NULL)
     {newnode->ele=temp2->ele;
      newnode->p=temp2->p;
      temp->next=newnode;
      temp=temp->next;
      temp2=temp2->next;}
 }

}

void display(poly *list)
{ poly *temp=list;
  while(temp->next!=NULL)
  { printf("%dx^%d",temp->ele,temp->p);
    if(temp->next->ele>0)
       printf("+");
    temp=temp->next;
  }
```

```
  printf("%dx^%d",temp->ele,temp->p);
  printf("\n");
}

int main()
{

   printf("ENTER DETAILS OF POLYNOMIAL 1\n");
   create(&l1);
   printf("THE POLY 1 ");
   display(l1);
   printf("ENTER DETAILS OF POLYNOMIAL 2\n");
   create(&l2);
   printf("THE POLY 2 ");
   display(l2);
   add();
   printf("THE RESULT IS ");
   display(l3);


}
```

Output:-

```
ENTER DETAILS OF POLYNOMIAL 1
Enter coeff :3
Enter power :2
To continue 1 otherwise 01
Enter coeff :2
Enter power :1
To continue 1 otherwise 00
THE POLY 1 3x^2+2x^1
ENTER DETAILS OF POLYNOMIAL 2
Enter coeff :3
Enter power :2
To continue 1 otherwise 00
THE POLY 2 3x^2
THE RESULT IS 6x^2+2x^1
```

```
//polysub
#include <stdio.h>
#include <stdlib.h>
struct Poly
{ int ele;
  int p;
  struct Poly*next;}*l1=NULL,*l2=NULL,*l3=NULL;
int coeff,power,ch=1;
typedef struct Poly poly;
void display(poly *list);
void create(poly **list)
{  do{
   poly *n1=malloc(sizeof(poly));
   printf("Enter coeff :");
   scanf("%d",&n1->ele);
   printf("Enter power :");
```

```
   scanf("%d",&n1->p);

   n1->next=NULL;
   if(*list==NULL)
    *list=n1;



    else{
    poly *temp=*list;
    while(temp->next!=NULL)
      temp=temp->next;
    temp->next=n1;}
    printf("To continue 1 otherwise 0");
    scanf("%d",&ch);
    }while(ch==1);
}
void add()
{ poly *temp1=l1;
  poly *temp2=l2;
  poly *temp=NULL;
  while((temp1!=NULL) && (temp2!=NULL))
  { poly *newnode=(poly*)malloc(sizeof(poly));
    newnode->next=NULL;
    if(temp1->p > temp2->p)
    {  newnode->ele=temp1->ele;
       newnode->p=temp1->p;
       temp1=temp1->next;
    }
    else if( (temp1->p) < (temp2->p))
    {  newnode->ele=-1*(temp2->ele);
       newnode->p=temp2->p;
       temp2=temp2->next;
    }

    else if(temp1->p == temp2->p)
    {  newnode->ele=temp1->ele-temp2->ele;
       newnode->p=temp1->p;
       temp1=temp1->next;
       temp2=temp2->next;
    }

     if (l3 == NULL) {
          l3 = newnode;
          temp = l3;
       } else {
          temp=l3;
          while(temp->next!=NULL)
               temp = temp->next;
          temp->next=newnode;
          temp=newnode;
       }
   }
```

```
  while(temp1!=NULL || temp2!=NULL)
 {   poly *newnode=malloc(sizeof(poly));
     newnode->next=NULL;
     if(temp1!=NULL)
     {newnode->ele=temp1->ele;
      newnode->p=temp1->p;
      temp->next=newnode;
      temp=temp->next;
      temp1=temp1->next;}




     else if(temp2!=NULL)
     {newnode->ele=-1*(temp2->ele);
      newnode->p=temp2->p;
      temp->next=newnode;
      temp=temp->next;
      temp2=temp2->next;}
 }

}



void display(poly *list)
{ poly *temp=list;
  while(temp->next!=NULL)
  { printf("%dx^%d",temp->ele,temp->p);
    if(temp->next->ele>0)
       printf("+");
    temp=temp->next;
  }
  printf("%dx^%d",temp->ele,temp->p);
  printf("\n");
}




int main()
{
   printf("ENTER DETAILS OF POLYNOMIAL 1\n");
   create(&l1);
   printf("THE POLY 1 ");
   display(l1);
   printf("ENTER DETAILS OF POLYNOMIAL 2\n");
   create(&l2);
   printf("THE POLY 2 ");
   display(l2);
   add();
   printf("THE RESULT IS ");
   display(l3);


}
```

Output:-

```
ENTER DETAILS OF POLYNOMIAL 1
Enter coeff :3
Enter power :2
To continue 1 otherwise 01
Enter coeff :2
Enter power :1
To continue 1 otherwise 00
THE POLY 1 3x^2+2x^1
ENTER DETAILS OF POLYNOMIAL 2
Enter coeff :3
Enter power :2



To continue 1 otherwise 00
THE POLY 2 3x^2
THE RESULT IS 2x^1
```

//polymul

```
#include <stdio.h>
#include <stdlib.h>

struct Poly
{ int ele;
  int p;
  struct Poly*next;}*l1=NULL,*l2=NULL,*l3=NULL;
int coeff,power,ch=1;
typedef struct Poly poly;
void display(poly *list);
void  arrange();
void create(poly **list)

{  do{
   poly *n1=malloc(sizeof(poly));
   printf("Enter coeff :");
   scanf("%d",&n1->ele);
   printf("Enter power :");
   scanf("%d",&n1->p);
   n1->next=NULL;
   if(*list==NULL)
    *list=n1;
    else{
    poly *temp=*list;
    while(temp->next!=NULL)
      temp=temp->next;
    temp->next=n1;}
    printf("To continue 1 otherwise 0");
    scanf("%d",&ch);
    }while(ch==1);
}
```

```
poly *findprev(poly *current)
{ poly *temp=l3;
  poly *prev=NULL;
  while(temp!=NULL)
  {  if(temp==current)
       return prev;
   prev=temp;
   temp=temp->next;}
   return prev;}

void mul()
{ poly*temp1=l1;
  poly*temp2=l2;
  poly*temp=l3;
  int c=0;
  while(temp1!=NULL)
  {   c++;
     temp1=temp1->next;}
  temp1=l1;
  for(int i=0;i<c;i++)


  { while(temp2!=NULL)
    { poly*newnode=malloc(sizeof(poly));
      newnode->ele=(temp1->ele )* (temp2->ele);
      newnode->p=(temp1->p)+(temp2->p);
      if(l3==NULL)
      {    l3=newnode;
           temp=l3;}
      else{
      temp->next=newnode;
      newnode->next=NULL;
      temp=newnode;}
      temp2=temp2->next;
    }
    temp1=temp1->next;
    temp2=l2;
  }
  arrange();

}

void arrange()
{
   poly* t1 = l3;
   while (t1->next != NULL)
   {
      poly* t = l3;
      while (t != NULL)
      {
         if (t->p == t1->p && t != t1)
         {
```

```
poly* pre = findprev(t);
            t1->ele = t1->ele + t->ele;
            pre->next = t->next;
            free(t);
            t = pre->next;
          }
          else
          {
             t = t->next;
             }
       }
       t1 = t1->next;
    }
}


void display(poly *list)
{ poly *temp=list;
  while(temp->next!=NULL)
  { printf("%dx^%d",temp->ele,temp->p);
    if(temp->next->ele>0)
       printf("+");
    temp=temp->next;
  }
  printf("%dx^%d",temp->ele,temp->p);
  printf("\n");
}




int main()
{
    printf("ENTER DETAILS OF POLYNOMIAL 1\n");
    create(&l1);
    printf("THE POLY 1 ");
    display(l1);
    printf("ENTER DETAILS OF POLYNOMIAL 2\n");
    create(&l2);
    printf("THE POLY 2 ");
    display(l2);
    mul();
    printf("THE RESULT IS ");
    display(l3);

}
```

Output:-

```
ENTER DETAILS OF POLYNOMIAL 1
Enter coeff :3
Enter power :2
To continue 1 otherwise 01
Enter coeff :2
Enter power :1
To continue 1 otherwise 00
THE POLY 1 3x^2+2x^1
ENTER DETAILS OF POLYNOMIAL 2
Enter coeff :3
Enter power :2
To continue 1 otherwise 00
THE POLY 2 3x^2
THE RESULT IS 9x^4+6x^3
```

Ex. No.:	Implementation of Stack using Array and Linked List Implementation	Date:

Write a C program to implement a stack using Array and linked List implementation and execute the following operation on stack.

(i) Push an element into a stack

(ii) Pop an element from a stack

(iii) Return the Top most element from a stack

(iv) Display the elements in a stack

Algorithm:
Code:

Array Implementation:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 10

struct ArrayStack {
   int top;
   int array[MAX_SIZE];
};

struct ArrayStack* createArrayStack() {
   struct ArrayStack* stack = (struct ArrayStack*)malloc(sizeof(struct ArrayStack));
   stack->top = -1;
   return stack;
}

int isArrayStackEmpty(struct ArrayStack* stack) {
   return (stack->top == -1);
}

void pushArray(struct ArrayStack* stack, int data) {
   if (stack->top == MAX_SIZE - 1) {
      printf("Stack Overflow\n");
      return;
   }
   stack->array[++stack->top] = data;
}

int popArray(struct ArrayStack* stack) {
   if (isArrayStackEmpty(stack)) {
      printf("Stack Underflow\n");
      return -1;
   }
   return stack->array[stack->top--];
}

int main() {
   struct ArrayStack* arrayStack = createArrayStack();

   // Pushing elements onto the stack
   pushArray(arrayStack, 10);
```

```
    pushArray(arrayStack, 20);
    pushArray(arrayStack, 30);

    // Popping elements from the stack
    printf("Popped from arrayStack: %d\n", popArray(arrayStack));
    printf("Popped from arrayStack: %d\n", popArray(arrayStack));
    printf("Popped from arrayStack: %d\n", popArray(arrayStack));

    return 0;
}
```

Linked List Implementation:

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

    int data;

    struct Node* next;

};


struct LinkedListStack {

    struct Node* top;

};


struct LinkedListStack* createLinkedListStack() {

    struct LinkedListStack* stack = (struct LinkedListStack*)malloc(sizeof(struct LinkedListStack));

    stack->top = NULL;

    return stack;

}


int isLinkedListStackEmpty(struct LinkedListStack* stack) {

    return (stack->top == NULL);

}


void pushLinkedList(struct LinkedListStack* stack, int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
  newNode->data = data;
    newNode->next = stack->top;
    stack->top = newNode;
}

int popLinkedList(struct LinkedListStack* stack) {
    if (isLinkedListStackEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node* temp = stack->top;
    int data = temp->data;
    stack->top = temp->next;
    free(temp);
    return data;
}

int main() {
    struct LinkedListStack* linkedListStack = createLinkedListStack();

    // Pushing elements onto the stack
    pushLinkedList(linkedListStack, 40);
    pushLinkedList(linkedListStack, 50);
    pushLinkedList(linkedListStack, 60);

    // Popping elements from the stack
    printf("Popped from linkedListStack: %d\n", popLinkedList(linkedListStack));
    printf("Popped from linkedListStack: %d\n", popLinkedList(linkedListStack));
    printf("Popped from linkedListStack: %d\n", popLinkedList(linkedListStack));

    return 0;
}
```

OUTPUT :

Stack Menu

1. Push

2. Pop

3. Display

4. Exit

Enter your choice: 1

Enter data to push: 10

10 pushed to stack

Enter your choice: 1

Enter data to push: 20

20 pushed to stack

Enter your choice: 1

Enter data to push: 30

30 pushed to stack

Enter your choice: 3

Elements in stack: 30 20 10

Enter your choice: 2

Popped element: 30

Enter your choice: 3

Elements in stack: 20 10

Enter your choice: 2

Popped element: 20

Enter your choice: 2

Popped element: 10

Enter your choice: 2

Stack Underflow

Enter your choice: 3

Stack is empty

Enter your choice: 4

Exiting program

Ex. No.:	Infix to Postfix Conversion	Date:

Write a C program to perform infix to postfix conversion using stack.

Algorithm:
Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 20

char stk[SIZE];
char post[SIZE];
int top=-1,p=-1;

void push(char c)
{ top++;
  stk[top]=c;
}

void pop()
{ //top--;
  p++;
  post[p]=stk[top];
  top--;
  //return top;
}

void postfix(char c)
{ if(isalpha(c)!=0)
  { p++;
    post[p]=c;
    return;
  }

  if(top==-1 || stk[top]=='(' || c=='(')
  { push(c);
    return;}

  if(c==')')
  { while(stk[top]!='(')
    { pop();
    }
    top--;
    stk[top]=c;
    return;
  }

  if(c=='*' || c=='/')
  { while(top!=-1)
    { if(stk[top]=='+' || stk[top]=='-' || stk[top]=='('
      { break;
      }
```

```
      else
      { pop();
      }
    }
    push(c);
    return;
  }

  if(c=='+' || c=='-')
  { while(top!=-1)
    { if(stk[top]=='(')
          break;
      else
      { pop();
      }
    }
    push(c);
    return;
  }
}
int main()
{
   char s[200];
   scanf("%s",s);
   for(int i=0;i<strlen(s);i++)
      postfix(s[i]);
   while(top!=-1)
   { pop();
   }
   printf("%s",post);
}
```

Output:-

```
a+b*c/d
abc*d/+
```

Ex. No.:	Evaluating Arithmetic Expression	Date:

Write a C program to evaluate Arithmetic expression using stack.

Algorithm:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define SIZE 20
int stk[SIZE];
int top=-1;
void push(int c);
void pop(char c);
void eval(char c[])
{
  if(isdigit(c[0])!=0)
  { push(atoi(c));
    return;
  }
  pop(c[0]);
}

void push(int c)
{ top++;
  stk[top]=c;
  return;
}

void pop(char c)
{
  int d;
  if(c=='+')
     d=stk[top-1]+stk[top];
  if(c=='-')
     d=stk[top-1]-stk[top];
  if(c=='*')
     d=stk[top-1]*stk[top];
  if(c=='/')
     d=stk[top-1]/stk[top];
  top=top-2;
  push(d);
}
```

```
int main()
{
   char c[100];
   char s[100];
   int j=0;
   fgets(c,25,stdin);
   c[strlen(c)-1]='\0';
   for(int i=0;i<=strlen(c);i++)
   {



    if(c[i]!=32  && c[i]!='\0')
    { s[j]=c[i];
      j++;}
    else
    { s[j]='\0';
      eval(s);
      j=0;
    }
   }
   printf("%d",stk[0]);
}
```

OUTPUT:

5 3 + 4 * 6 2 / -

29

Ex. No.:	Implementation of Queue using Array and Linked List Implementation	Date:

Write a C program to implement a Queue using Array and linked List implementation and execute the following operation on stack.

(i) Enqueue

(ii) Dequeue

(iii) Display the elements in a Queue

Algorithm:

CODE:-

```
//Queue linked list
#include <stdio.h>
#include <stdlib.h>
struct Queue
{ int ele;
 struct Queue *next;};
typedef struct Queue q;
q *f=NULL;
q *r=NULL;

void enqueue(int x)
{ q *newnode=malloc(sizeof(q));
 newnode->ele=x;
 if(f==NULL && r==NULL)
 { f=r=newnode;
  newnode->next=NULL;
  return;
 }
 r->next=newnode;
 r=newnode;
 newnode->next=NULL;}
 //f=newnode;}

void dequeue()
{ if(f==NULL && r==NULL)
```

```
  { printf("UNDERFLOW\n");
    return;}
  if(f==r)
  { printf("THE DELETED ELE IS %d\n",f->ele);
    f=r=NULL;
    return;}
   q *temp=f;
   printf("DELETED ELEMENT IS %d\n",temp->ele);
   f=f->next;
   free(temp);
}

void display()
{  q *temp=f;
  while(temp!=NULL)
  { printf("%d ",temp->ele);
    temp=temp->next;
  }
  printf("\n");
}

int main()
{
   int ch;
   printf("1 TO ENQUEUE\n2 TO DEQUEUE\n3 TO DISPLAY\n");
   do
   { printf("ENTER YOUR CHOICE ");
     scanf("%d",&ch);
     switch(ch)
     { case 1:
         int x;
         printf("ELEMENT TO BE ADDED");
         scanf("%d",&x);
```

```
          enqueue(x);
          break;
        case 2:
          dequeue();
          break;

       case 3:
          display();
          break;
       default:
          break;
     } } while(ch<=3);

   printf("THANK YOU");
}
//QUEUE USING ARRAY IMPLEMENTATION
//queue array
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100

int q[SIZE];
int f=-1,r=-1;

void enqueue(int x)
{ if(f==-1 && r==-1)
  { f++;
    r++;
    q[f]=x;
    return;
  }
  if(r==SIZE-1)
  { printf("OVERFLOW\n");
```

```
   return;}
  r++;
  q[r]=x;
}

void dequeue()
{ if(f==-1 && r==-1)
  { printf("UNDERFLOW\n");
    return;}
  if(f==r)
  { printf("THE DELETED ELE %d\n",q[f]);
    f=r=-1;
    return;}
  printf("The deleted element is %d\n",q[f]);
  f++;
}

void display()
{ for(int i=f;i<=r;i++)
  { printf("%d ",q[i]);
  }
  printf("\n");}


int main()
{
    int ch;
    printf("1 TO ENQUEUE\n2 TO DEQUEUE\n3 TO DISPLAY\n");
    do
    { printf("ENTER YOUR CHOICE ");
      scanf("%d",&ch);
      switch(ch)
      { case 1:
```

```
        int x;
        printf("ELEMENT TO BE ADDED");
        scanf("%d",&x);
        enqueue(x);
        break;
      case 2:
       dequeue();
       break;

     case 3:
       display();
       break;
     default:
      break;
    } } while(ch<=3);

  printf("THANK YOU");
}
```

OUTPUT:-

1 TO ENQUEUE

2 TO DEQUEUE

3 TO DISPLAY

ENTER YOUR CHOICE 1

ELEMENT TO BE ADDED20

ENTER YOUR CHOICE 1

ELEMENT TO BE ADDED30

ENTER YOUR CHOICE 1

ELEMENT TO BE ADDED40

ENTER YOUR CHOICE 3

20 30 40

ENTER YOUR CHOICE 2

DELETED ELEMENT IS 20

ENTER YOUR CHOICE 3

30 40

ENTER YOUR CHOICE 4

THANK YOU

Ex. No.:	Tree Traversal	Date:

Write a C program to implement a Binary tree and perform the following tree traversal operation.

(i) Inorder Traversal

(ii) Preorder Traversal

(iii) Postorder Traversal

Algorithm:

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
  struct Tree
  { int ele;
    struct Tree *left;
    struct Tree *right;};
  typedef struct Tree tree;
  //tree *root=NULL;

  tree *create(tree *root,int x)
  { if(root==NULL)
    { tree *newnode=malloc(sizeof(tree));
      newnode->ele=x;
      newnode->left=NULL;
      newnode->right=NULL;
      root=newnode;}
    else if(x<root->ele)
    { root->left=create(root->left,x);
    }
    else if(x>root->ele)
    { root->right=create(root->right,x);
    }
    return root;
  }
```

```
void inorder(tree *root)
{ if(root!=NULL)
  { inorder(root->left);
    printf("%d ",root->ele);
    inorder(root->right);
  }

}
void preorder(tree *root)
{ if(root!=NULL)
  {
    printf("%d ",root->ele);
    preorder(root->left);
    preorder(root->right);
  }

}
void postorder(tree *root)
{ if(root!=NULL)
  {
    postorder(root->left);
    postorder(root->right);
    printf("%d ",root->ele);
  }

}

int main()
{   tree *root=NULL;
    int n,x;
    printf("ENTER NO OF ELEMENTS");
    scanf("%d",&n);
```

```
    printf("ENTER THE ELEMENTS ");
    for(int i=0;i<n;i++)
    {  scanf("%d",&x);
       root=create(root,x);
    }
    printf("INORDER TRAVERSAL IS ");
    inorder(root);
    printf("\nPOSTORDER TRAVERSAL IS ");
    postorder(root);
    printf("\nPREORDER TRAVERSAL IS ");
    preorder(root);


    return 0;
}
```

OUTPUT:-

ENTER NO OF ELEMENTS7

ENTER THE ELEMENTS 100 90 110 80 95 105 111

INORDER TRAVERSAL IS 80 90 95 100 105 110 111

POSTORDER TRAVERSAL IS 80 95 90 105 111 110 100

PREORDER TRAVERSAL IS 100 90 80 95 110 105 111

Ex. No.:	Implementation of Binary Search tree	Date:

Write a C program to implement a Binary Search Tree and perform the following operations.

(i) Insert

(ii) Delete

(iii) Search

(iv) Display

Algorithm:

CODE:-

```
#include <stdio.h>
#include <stdlib.h>

// Define a structure for the BST node
struct Node {
   int data;
   struct Node *left, *right;
};

// Function to create a new BST node
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->left = newNode->right = NULL;
   return newNode;
}

// Function to insert a node into the BST
struct Node* insertNode(struct Node* root, int data) {
   if (root == NULL) return createNode(data);

   if (data < root->data)
```

```
root->left = insertNode(root->left, data);
else if (data > root->data)
        root->right = insertNode(root->right, data);

    return root;
}

// Function to find the minimum value node in a tree
struct Node* findMin(struct Node* node) {
    struct Node* current = node;

    while (current && current->left != NULL)
        current = current->left;

    return current;
}

// Function to delete a node from the BST
struct Node* deleteNode(struct Node* root, int data) {
    if (root == NULL) return root;

    if (data < root->data)
        root->left = deleteNode(root->left, data);
    else if (data > root->data)
        root->right = deleteNode(root->right, data);
    else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
```

```
            return temp;
        }

        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}

// Function to search a node in the BST
struct Node* searchNode(struct Node* root, int data) {
    if (root == NULL || root->data == data)
        return root;

    if (root->data < data)
        return searchNode(root->right, data);

    return searchNode(root->left, data);
}

// Function to perform in-order traversal and display the tree
void inOrder(struct Node* root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}

int main() {
    struct Node* root = NULL;
```

```
int choice, data,n;
printf("Enter the no of elements to be inserted");
scanf("%d",&n);
printf("Enter elements");
for(int i=0;i<n;i++)
{ scanf("%d",&data);
 root=insertNode(root, data);}
while (1) {
   printf("\nBinary Search Tree Operations Menu\n");
   printf("1. Insert\n");
   printf("2. Delete\n");
   printf("3. Search\n");
   printf("4. Display\n");
   printf("5. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);

   switch (choice) {
      case 1:
         printf("Enter data to insert: ");
         scanf("%d", &data);
         root = insertNode(root, data);
         printf("%d inserted.\n", data);
         break;

      case 2:
         printf("Enter data to delete: ");
         scanf("%d", &data);
         root = deleteNode(root, data);
         printf("%d deleted.\n", data);
         break;

      case 3:
```

```
            printf("Enter data to search: ");
            scanf("%d", &data);
            struct Node* foundNode = searchNode(root, data);
            if (foundNode != NULL)
               printf("%d found in the tree.\n", data);
            else
               printf("%d not found in the tree.\n", data);
            break;

         case 4:
            printf("In-order display of the BST: ");
            inOrder(root);
            printf("\n");
            break;

         case 5:
            exit(0);
            break;

         default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

OUTPUT:-

```
Enter the no of elements to be inserted6
Enter elements100 90 110 80 95 105

Binary Search Tree Operations Menu
1. Insert
```

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 4

In-order display of the BST: 80 90 95 100 105 110

Binary Search Tree Operations Menu

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 2

Enter data to delete: 90

90 deleted.

Binary Search Tree Operations Menu

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 4

In-order display of the BST: 80 95 100 105 110

Binary Search Tree Operations Menu

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 3

Enter data to search: 80

80 found in the tree.

Binary Search Tree Operations Menu

1. Insert

2. Delete

3. Search

4. Display

5. Exit

Enter your choice: 5

Ex. No.:	Implementation of AVL Tree	Date:

Write a function in C program to insert a new node with a given value into an AVL tree. Ensure that the tree remains balanced after insertion by performing rotations if necessary. Repeat the above operation to delete a node from AVL tree.

Algorithm:

CODE:-

```
#include <stdio.h>
#include <stdlib.h>

struct Node
{
   int key;
   struct Node *left;
   struct Node *right;
   int height;
};
typedef struct Node node;
int height(node *n)
{
   if (n==NULL)
      return 0;
   return n->height;
}
node *findmin(node *tree)
{ if(tree==NULL)
  return NULL;
 else if(tree->left==NULL)
    return tree;
 else
    return findmin(tree->left);
```

```
}

int max(int a,int b)
{ return (a>b)?a:b;
}
node *rightrotate(node *y)
{ node *x=y->left;
  node *t2=x->right;

  x->right=y;
  y->left=t2;
  y->height=1+max(height(y->left),height(y->right));
  x->height=1+max(height(x->left),height(x->right));
  return x;
}

node *leftrotate(node *x)
{ node *y=x->right;
  node *t2=y->left;

  y->left=x;
  x->right=t2;
  x->height=1+max(height(x->left),height(x->right));
  y->height=1+max(height(y->left),height(y->right));
  return y;
}
int getbalance(struct Node *n)
{
    if (n == NULL)
        return 0;
    return height(n->left) - height(n->right);
}
```

```
node *insert(node *tree,int k)
{ if(tree==NULL)
  { node *newnode=malloc(sizeof(node));
    newnode->key=k;
    newnode->left=NULL;
    newnode->right=NULL;
    newnode->height=1;
    tree=newnode;
  }

  else if(k<tree->key)
      tree->left=insert(tree->left,k);
  else if(k>tree->key)
      tree->right=insert(tree->right,k);
  //else
     //return tree;
  tree->height=1+max(height(tree->left),height(tree->right));
  int bal=getbalance(tree);
  if(bal>1 && k<tree->left->key)
       return rightrotate(tree);
  if(bal<-1 && k>tree->right->key)
       return leftrotate(tree);
  if(bal>1 && k>tree->left->key)
  {    tree->left=leftrotate(tree->left);
       return rightrotate(tree);  }
  if(bal<-1 && k<tree->right->key)
  {    tree->right=rightrotate(tree->right);
       return leftrotate(tree);  }
  return tree;
}

node *delete(node *tree,int e)
```

```
{  node *temp=malloc(sizeof(node));
  if(e<tree->key)
    tree->left=delete(tree->left,e);
  else if(e>tree->key)
    tree->right=delete(tree->right,e);
  else if(tree->left && tree->right)
   { temp=findmin(tree->right);
     tree->key=temp->key;
     tree->right=delete(tree->right,temp->key);
   }
  else
  { temp=tree;
   if(tree->left==NULL)
       tree=tree->right;
   else if(tree->right==NULL)
      tree=tree->left;
    free(temp);
  }
  if (tree == NULL)
      return tree;

  tree->height = 1 + max(height(tree->left),
                         height(tree->right));

  int balance = getbalance(tree);

   // If this node becomes unbalanced,
   // then there are 4 cases

   // Left Left Case
   if (balance > 1 &&
      getbalance(tree->left) >= 0)
      return rightrotate(tree);
```

```
    // Left Right Case
    if (balance > 1 &&
        getbalance(tree->left) < 0)
    {
        tree->left = leftrotate(tree->left);
        return rightrotate(tree);
    }

    // Right Right Case
    if (balance < -1 &&
        getbalance(tree->right) <= 0)
        return leftrotate(tree);

    // Right Left Case
    if (balance < -1 &&
        getbalance(tree->right) > 0)
    {
        tree->right = rightrotate(tree->right);
        return leftrotate(tree);
    }

    return tree;
}
void inorder(node *tree)
{ if(tree!=NULL)
  { //printf("%d ",tree->key);
    inorder(tree->left);
    printf("%d ",tree->key);
    inorder(tree->right);}
}

int main()
{
```

```
    node *tree=NULL;
    int n;
    printf("ENTER TOT NO OF ELEMENTS");
    scanf("%d",&n);
    int e;
    printf("ENETR ELEMENTS");
    for(int i=0;i<n;i++)
    {   scanf("%d",&e);
       tree=insert(tree,e);}
    //inorder(tree);
    printf("ENETR ELE TO BE DELETED");
    scanf("%d",&e);
    tree = delete(tree,e);
    inorder(tree);



    return 0;
}
```

OUTPUT:-

```
 ENTER TOT NO OF ELEMENTS9
ENETR ELEMENTS9 5 10 0 6 11 -1 1 2
ENETR ELE TO BE DELETED10
-1 0 1 2 5 6 9 11
```

Ex. No.:	Graph Traversal	Date:

Write a C program to create a graph and perform a Breadth First Search and Depth First Search.

Algorithm:

CODE:-

```
//bfs
#include <stdio.h>
#include <stdlib.h>
void enqueue(int ele);
int dequeue();
struct Node
{ int ele;
  struct Node *next;};
typedef struct Node node;

struct Graph
{ int numver;
  node **adjlists;
  int *visited;};

typedef struct Graph graph;
node *createnode(int key)
{ node *nn=malloc(sizeof(node));
  nn->next=NULL;
  nn->ele=key;
  return nn;
}

graph *creategraph(int v)
{ graph *g=malloc(sizeof(graph));
  g->numver=v;
  g->adjlists=malloc(v *sizeof(node *));
  g->visited=malloc(v *sizeof(int));
  for(int i=0;i<v;i++)
  { g->adjlists[i]=NULL;
    g->visited[i]=0;}
  return g;
}

void addedge(graph *g,int sv,int dv)
{
   node *nn=createnode(dv);
   nn->next=g->adjlists[sv];
   g->adjlists[sv]=nn;

   node *n1=createnode(sv);  //undirected graph
   n1->next=g->adjlists[dv];
   g->adjlists[dv]=n1;
}
```

```
int q[10];
int f=-1,r=-1;

void bfs(graph *g,int sv)
{
  enqueue(sv);
  g->visited[sv]=1;
  while (f!=-1)
  { int d=dequeue();
    printf("%d ",d);
    node *temp=g->adjlists[d];
    while(temp!=NULL)
    { if(g->visited[temp->ele] !=1)
      { enqueue(temp->ele);
        g->visited[temp->ele]=1;}
       temp=temp->next;
    }
  }
}



void enqueue(int ele)
{
 if(f==-1)
 {   f++;
 }
 r++;
 q[r]=ele;
}

int dequeue()
{ int t=q[f];
 if(f==r)
 { f=r=-1;
   return t;
 }
 f++;
 return t;
}



int main()
{
   graph *graph=creategraph(5);
   addedge(graph,0,4);
   addedge(graph,0,1);
   addedge(graph,1,2);
   addedge(graph,1,4);
   addedge(graph,1,3);
   addedge(graph,2,3);
   addedge(graph,3,4);
   bfs(graph,0);

   return 0;
}
```

Output:-

0 1 4 3 2

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{ int ele;
  struct Node *next;};
typedef struct Node node;

struct Graph
{ int numver;
  node **adjlists;
  int *visited;};

typedef struct Graph graph;
node *createnode(int key)
{ node *nn=malloc(sizeof(node));
  nn->next=NULL;
  nn->ele=key;
  return nn;
}

graph *creategraph(int v)
{ graph *g=malloc(sizeof(graph));
  g->numver=v;
  g->adjlists=malloc(v *sizeof(node *));
  g->visited=malloc(v *sizeof(int));
  for(int i=0;i<v;i++)
  { g->adjlists[i]=NULL;
    g->visited[i]=0;}
  return g;
}
```

```
void addedge(graph *g,int sv,int dv)
{
   node *nn=createnode(dv);
   nn->next=g->adjlists[sv];
   g->adjlists[sv]=nn;

   node *n1=createnode(sv);  //undirected graph
   n1->next=g->adjlists[dv];
   g->adjlists[dv]=n1;
}


void dfs(graph *g,int sv)
{  g->visited[sv]=1;
  node *temp=g->adjlists[sv];
  printf("%d ",sv);
  while(temp!=NULL)
  { if(g->visited[temp->ele]!=1)
    { //g->visited[temp->ele]=1;
      dfs(g,temp->ele);
    }
    temp=temp->next;
  }

}

int main()
{
   graph *graph=creategraph(5);
   addedge(graph,0,4);
   addedge(graph,0,1);
   addedge(graph,1,2);
```

```
    addedge(graph,1,4);
    addedge(graph,1,3);
    addedge(graph,2,3);
    addedge(graph,3,4);
    dfs(graph,0);

    return 0;
}
```

Output:-

0 1 3 4 2

Ex. No.:	Topological Sorting	Date:

Write a C program to create a graph and display the ordering of vertices.

Algorithm:

CODE:-

```
#include <stdio.h>
#include <stdlib.h>

// Node structure for adjacency list
struct Node {
   int vertex;
   struct Node* next;
};

// Graph structure
struct Graph {
   int numVertices;
   struct Node** adjLists;
   int* visited;
};

int stack[10];
int top=-1;
struct Node* createNode(int v);
struct Graph* createGraph(int vertices);
void addEdge(struct Graph* graph, int src, int dest);
void topologicalSort(struct Graph* graph);
void topologicalSortUtil(struct Graph* graph, int vertex);
//struct Stack* createStack(int capacity);
void push(int value);
int pop();
int isEmpty();

int main() {
   struct Graph* graph = createGraph(6);
   addEdge(graph, 5, 2);
   addEdge(graph, 5, 0);
   addEdge(graph, 4, 0);
   addEdge(graph, 4, 1);
   addEdge(graph, 2, 3);
   addEdge(graph, 3, 1);

   printf("Topological Sort: ");
   topologicalSort(graph);

   return 0;
}
```

```
struct Node* createNode(int v) {
   struct Node* newNode = malloc(sizeof(struct Node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}

struct Graph* createGraph(int vertices) {
   struct Graph* graph = malloc(sizeof(struct Graph));
   graph->numVertices = vertices;

   graph->adjLists = malloc(vertices * sizeof(struct Node*));
   graph->visited = malloc(vertices * sizeof(int));

   for (int i = 0; i < vertices; i++) {
      graph->adjLists[i] = NULL;
      graph->visited[i] = 0;
   }
   return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
   struct Node* newNode = createNode(dest);
   newNode->next = graph->adjLists[src];
   graph->adjLists[src] = newNode;
}

void topologicalSort(struct Graph* graph) {
   //struct Stack* stack = createStack(graph->numVertices);

   for (int i = 0; i < graph->numVertices; i++) {
      if (graph->visited[i] == 0) {
         topologicalSortUtil(graph, i);
      }
   }

   while (!isEmpty()) {
      printf("%d ", pop());
   }
}

void topologicalSortUtil(struct Graph* graph, int vertex) {
   graph->visited[vertex] = 1;

   struct Node* temp = graph->adjLists[vertex];
   while (temp != NULL) {
      int adjVertex = temp->vertex;
      if (graph->visited[adjVertex] == 0) {
         topologicalSortUtil(graph, adjVertex);
      }
      temp = temp->next;
   }

   push(vertex);
}
```

```
void push(int value) {
   top++;
   stack[top]= value;
}



int pop() {
   if (isEmpty()) {
      return -1;
   }
   int ele=stack[top];
   top--;
   return ele;
}

int isEmpty() {
    if(top==-1)
       return 1;
     return 0;
}
```

OUTPUT:-

Topological Sorting Order:5 4 2 3 1 0

Ex. No.:	Graph Traversal	Date:

Write a C program to create a graph and find a minimum spanning tree using prims algorithm.

Algorithm:

CODE:

```
//prims algorithm
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>
#define V 5
void display(int key[V],int p[V]);
int mstmin(int key[],bool mst[])

{
   int min=INT_MAX,minindex;
   for(int i=0;i<V;i++)
   { if(mst[i]!=true && min>key[i])
    {
       min=key[i];
       minindex=i;
    }
   }
   return minindex;
}

void primst(int graph[V][V])
{  int p[V];
   int key[V];
   bool mst[V];
   for(int i=0;i<V;i++)
```

```
    {
        key[i]=INT_MAX;
        mst[i]=false;
    }
    p[0]=-1;
    key[0]=0;

    for(int i=0;i<(V-1);i++)
    { int u=mstmin(key,mst);
      mst[u]=true;
      for(int v=0;v<V;v++)
      {
          if(graph[u][v] && mst[v]==false && key[v]>graph[u][v])
          {
              key[v]=graph[u][v];
              p[v]=u;
          }

      }
    }
    display(key,p);
}

void display(int key[V],int p[V])
{
    for(int i=1;i<V;i++)
     {  printf("%d-%d %d",p[i],i,key[i]);
        printf("\n");}
}
```

```
int main() {
   int graph[V][V] = { { 0, 2, 0, 6, 0 },
                       { 2, 0, 3, 8, 5 },
                       { 0, 3, 0, 0, 7 },
                       { 6, 8, 0, 0, 9 },
                       { 0, 5, 7, 9, 0 } };

   primst(graph);

   return 0;
}
```

Output:-

0-1 2

1-2 3

0-3 6

1-4 5

Ex. No.:	Graph Traversal	Date:

Write a C program to create a graph and find the shortest path using Dijikstra's Algorithm.

Algorithm:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <stdbool.h>

#define V 5

int minDistance(int dist[], bool sptSet[]) {
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;
   return min_index;
}

void printSolution(int dist[]) {
   printf("Vertex \tDistance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t%d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src) {
   int dist[V];
   bool sptSet[V];

   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
```

```
    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist);
}

int main() {
                    { 0, 2, 0, 6, 0 },
                    { 2, 0, 3, 8, 5 },
                    { 0, 3, 0, 0, 7 },
                    { 6, 8, 0, 0, 9 },
                    { 0, 5, 7, 9, 0 } };
    dijkstra(graph, 0);

    return 0;
}
```

OUTPUT:-

4-0 7

4-1 5

4-2 7

4-3 9

Ex. No.:	Sorting	Date:

Write a C program to take n numbers and sort the numbers in ascending order. Try to implement the same using following sorting techniques.

1. **Quick Sort**
2. **Merge Sort**

Algorithm:

CODE:

1. QUICK SORT:

```
#include <stdio.h>


void print(int arr[])
{  for(int i=0;i<6;i++)
     printf("%d ",arr[i]);
}
void Quicksort(int arr[],int left,int right)
{  int i,j,temp,pivot;


  if(left<right)
  {   pivot=left;
      i=left;
      j=right;
      while(i<j)
     { while(arr[i]<arr[pivot])
          i++;
       while(arr[pivot]<arr[j])
          j--;


       if (i<j)
      { temp=arr[i];
        arr[i]=arr[j];
```

```
      arr[j]=temp;}
    }

    temp=arr[j];
    arr[j]=arr[pivot];
    arr[pivot]=temp;
    Quicksort(arr,left,j-1);
    Quicksort(arr,j+1,right);

  }
}

int main()
{
   int arr[6]={10,9,8,7,6,5};
   Quicksort(arr,0,5);
   print(arr);
}
```

2. MERGE SORT

```
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   int L[n1], R[n2];

   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];
```

```
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
      if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
      } else {
        arr[k] = R[j];
        j++;
      }
      k++;
    }

    while (i < n1) {
      arr[k] = L[i];
      i++;
      k++;
    }

    while (j < n2) {
      arr[k] = R[j];
      j++;
      k++;
    }
  }

  void mergeSort(int arr[], int l, int r) {
    if (l < r) {
      int m = l + (r - l) / 2;
      mergeSort(arr, l, m);
      mergeSort(arr, m + 1, r);
      merge(arr, l, m, r);
```

```
    }
}

void printArray(int A[], int size) {
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d numbers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    mergeSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:-

```
5 6 7 8 9 10
```

Ex. No.:	Hashing	Date:

Write a C program to create a hash table and perform collision resolution using the following techniques.

(i) Open addressing

(ii) Closed Addressing

(iii) Rehashing

Algorithm:

CODE:

i)Open Addressing:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct HashNode
{ int key;
  int count;
  bool op;};
typedef struct HashNode hn;
hn **create()
{ hn**ht=malloc(10 * sizeof(hn *));
  for(int i=0;i<10;i++)
      ht[i]=NULL;
  return ht;
}

hn *createnode(int key,int count)
{ hn*nn=malloc(sizeof(hn));
  nn->key=key;
  nn->count=count;
  nn->op=true;
  return nn;
}
```

```
void openadd(hn **ht,int key,int count)
{  int index=key%10;
   while(ht[index]!=NULL && ht[index]->op)
   { index=(index+1)%10;

   }
   ht[index]=createnode(key,count);
}

void display(hn** ht)
{   for (int i = 0; i < 10; i++)
   {
    if (ht[i] != NULL)
        printf("%d %d - %d\n", i,ht[i]->key, ht[i]->count);
   else

        printf("NULL\n");

    }
}

int main()
{  hn** hashTable = create();

    openadd(hashTable, 5, 10);
    openadd(hashTable, 15, 20);
    openadd(hashTable, 25, 30);
    openadd(hashTable, 35, 40);
    openadd(hashTable, 45, 50);
    openadd(hashTable, 55, 60);
    openadd(hashTable, 65, 70);
    openadd(hashTable, 75, 80);
    openadd(hashTable, 85, 90);
```

```
    openadd(hashTable, 95, 100);
    display(hashTable);
}
```

2. Closed Addressing:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct HashNode
{ int key;
  int count;
  struct HashNode *next;};
typedef struct HashNode hn;
hn **create()
{ hn**ht=malloc(10 * sizeof(hn *));
  for(int i=0;i<10;i++)
      ht[i]=NULL;
  return ht;
}

hn *createnode(int key,int count)
{ hn*nn=malloc(sizeof(hn));
  nn->key=key;
  nn->count=count;
  nn->next=NULL;
  return nn;
}

void closeadd(hn **ht,int key,int count)
{  int index=key%10;
   if(ht[index]==NULL)
   {
```

```
    ht[index]=createnode(key,count);
    return;
   }
   hn*temp=createnode(key,count);
   temp->next=ht[index];
   ht[index]=temp;

}




void display(hn** ht)
{   for (int i = 0; i < 10; i++)
   {
    if (ht[i] != NULL)
    { hn *temp=ht[i];
      printf("%d ",i);
      while (temp!=NULL)
      { printf("%d - %d\n",temp->key, temp->count);
        temp=temp->next;}
    }
   else

       printf("%d NULL\n",i);


    }
}

int main()
{  hn** hashTable = create();

    closeadd(hashTable, 5, 10);
```

```
    closeadd(hashTable, 15, 20);
    closeadd(hashTable, 25, 30);
    closeadd(hashTable, 35, 40);
    closeadd(hashTable, 45, 50);
    closeadd(hashTable, 55, 60);
    closeadd(hashTable, 65, 70);
    closeadd(hashTable, 75, 80);
    closeadd(hashTable, 85, 90);
    closeadd(hashTable, 95, 100);
    display(hashTable);
}
```

3. Rehashing:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define SIZE 10

struct HashNode
{ int key;
  int count;
  bool op;};
typedef struct HashNode hn;

hn **create(int s)
{ hn**ht=malloc(s * sizeof(hn *));
  for(int i=0;i<10;i++)
      ht[i]=NULL;
  return ht;
}

hn *createnode(int key,int count)
{ hn*nn=malloc(sizeof(hn));
```

```
 nn->key=key;
 nn->count=count;
 nn->op=true;
 return nn;
}

void rehash(hn **ot, hn **nt, int os, int ns) {
   for (int i = 0; i < os; i++) {
      if (ot[i] != NULL && ot[i]->op) {
         int index = ot[i]->key % ns;
         while (nt[index] != NULL && nt[index]->op) {
            index = (index + 1) % ns;
         }
         nt[index] = createnode(ot[i]->key, ot[i]->count);
      }
   }
   // Free old table
   free(ot);
}

// Add a node

void openadd(hn **ht,int key,int count)
{  int index=key%10;
   while(ht[index]!=NULL && ht[index]->op)
   { index=(index+1)%10;

   }
   ht[index]=createnode(key,count);
}

void display(hn** ht)
{   for (int i = 0; i < 20; i++)
```

```
  {
   if (ht[i] != NULL)
      printf("%d %d - %d\n", i,ht[i]->key, ht[i]->count);
  else

      printf("%d NULL\n",i);

   }
}
double load(int c,int s)
{ return (double)c/s;
}
int main()
{  hn** ot = create(SIZE);
  int os=10;
   openadd(ot, 5, 10);
   openadd(ot, 15, 20);
   openadd(ot, 25, 30);
   openadd(ot, 35, 40);
   openadd(ot, 45, 50);
   openadd(ot, 55, 60);
   openadd(ot, 65, 70);
   openadd(ot, 75, 80);
   openadd(ot, 85, 90);
   openadd(ot, 95, 100);
   //display(ot);

   double LF=load(10,SIZE);
   //printf("%f",LF);
   if(LF >0.7)
   { int ns=SIZE *2;
     //printf("hi");
     hn **nt=create(ns);
```

```
    rehash(ot,nt,os,ns);
    ot=nt;}
  display(ot);
}
```

OUTPUT:-

(i) OPEN ADDRESSING

0 55 - 60

1 65 - 70

2 75 - 80

3 85 - 90

4 95 - 100

5 5 - 10

6 15 - 20

7 25 - 30

8 35 - 40

9 45 – 50

(ii)CLOSED ADDRESSING

0 55 - 60

1 65 - 70

2 75 - 80

3 85 - 90

4 95 - 100

5 5 - 10

6 15 - 20

7 25 - 30

8 35 - 40

9 45 - 50

(iii)REHASHING

0 NULL

1 NULL

2 NULL

3 NULL

4 NULL

5 65 - 70

6 85 - 90

7 5 - 10

8 25 - 30

9 45 - 50

10 NULL

11 NULL

12 NULL

13 NULL

14 NULL

15 55 - 60

16 75 - 80

17 95 - 100

18 15 - 20

19 35 - 40

Rajalakshmi Institutions

Rajalakshmi Engineering College

Rajalakshmi Nagar Thandalam, Chennai - 602 105.

Phone : +91-44-67181111, 67181112

Website : www.rajalakshmi.org