ARASU ENGINEERING COLLEGE
Chennai Salai, Kumbakonam – 612501
(Approved by AICTE, New Delhi | Affiliated to Anna University, Chennai |NAAC Accredited | ISO 9001:2008 Certified)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
(NBA Accredited)

CS3311 - DATA STRUCTURES LABORATORY

III SEMESTER – R 2021

LABORATORY MANUAL

Prepared By:
Mrs. R. Kalaiselvi,
AP/CSE
Mrs.S.Parvathy
AP/CSE

Approved By:
Dr. P.Ramesh,
Head of the Department,
Department of CSE.

ARASU ENGINEERING COLLEGE, KUMBAKONAM

Vision

To reach the levels of Teaching-Learning process with societal concerns, disseminate technical knowledge, to uplift women with moral values and enhance the role of educands with ethical awareness.

Mission

- ❖ Concentrated attention towards the lagging student Sector
- ❖ Commitment with a stress on ethical and moral values in shaping the individual for the technical needs of the nation.
- ❖ Dissemination of technical knowledge with a stress on pragmatic values.
- ❖ To alleviate the gap between the learner and the teacher.
- ❖ Commitment towards the societal values

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Vision

To be in the forefront of Computer Science and Engineering by producing competing professional with innovative skills, moral values and societal concerns with a commitment towards building a strong nation.

Mission

- ❖ **Problem – Solving Skills :** Equipping students with fundamental computing knowledge and problem - solving skills which are necessary to solve real-world engineering challenges to meet industry and societal needs.
- ❖ **Quality Education :** Imparting quality education through continuous Teaching – Learning process, including interdisciplinary areas that extend the scope of Computer Science.
- ❖ **Societal Commitment :** Inculcating students with analytical ability, innovative spirit and entrepreneur skills with ethical values and societal commitment.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

The PEOs are to facilitate graduating students to

- **Computational Ability:** Acquire knowledge on different domains of Computing Technologies and to have a successful career in industries or as entrepreneurs.
- **Quality Professionals:** Exhibit professionalism, team spirit, problem-solving skills, leadership skills and to adapt with emerging technological changes.
- **Ethical Values:** Practice their profession with consideration of ethical values, societal responsibility, environmental concern, and motivation for life-long learning.

PROGRAM SPECIFIC OUTCOMES (PSOs)

- **PSO1 (Software Skills):**

  Imparting the knowledge for the development of quality software for scientific and business applications by applying fundamental and advanced concepts of Computer Science.
- **PSO2 (Professional Skills):**

  Developing the practical competency to work in industries and to manage different projects effectively by using modern tools with professional behavior and ethics.

CS3311 DATA STRUCTURES LABORATORY

L T P C
0 0 3 1.5

COURSE OBJECTIVES:

- To demonstrate array implementation of linear data structure algorithms.
- To implement the applications using Stack.
- To implement the applications using Linked list
- To implement Binary search tree and AVL tree algorithms.
- To implement the Heap algorithm.
- To implement Dijkstra's algorithm.
- To implement Prim's algorithm
- To implement Sorting, Searching and Hashing algorithms.

LIST OF EXERCISES:

1. Array implementation of Stack, Queue and Circular Queue ADTs
2. Implementation of Singly Linked List
3. Linked list implementation of Stack and Linear Queue ADTs
4. Implementation of Polynomial Manipulation using Linked list
5. Implementation of Evaluating Postfix Expressions, Infix to Postfix conversion
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues
9. Implementation of Dijkstra's Algorithm
10. Implementation of Prim's Algorithm
11. Implementation of Linear Search and Binary Search
12. Implementation of Insertion Sort and Selection Sort
13. Implementation of Merge Sort
14. Implementation of Open Addressing (Linear Probing and Quadratic Probing)

TOTAL:45 PERIODS

COURSE OUTCOMES:

At the end of this course, the students will be able to:
CO1: Implement Linear data structure algorithms.
CO2: Implement applications using Stacks and Linked lists
CO3: Implement Binary Search tree and AVL tree operations.
CO4: Implement graph algorithms.
CO5: Analyze the various searching and sorting algorithms.

Ex.No	PROGRAMS
1)a)	Array implementation of Stack
1)b)	Array implementation of Queue
1)c)	Array Implementation of Circular queue ADTs
2)	Singly Linked List
3)a)	Stack Using Linked List
3)b)	Queue Using Linked List
4)a)	Polynomial addition
5)a)	Evaluating Postfix Expression
5)b)	Infix To Postfix Conversion
6)	Binary Search Tree
7)	AVL Trees
8)	Heaps using Priority Queues
9)	Dijkstra’s Algorithm
10)	Prim’s Algorithm
11)a)	Linear Search
11)b)	Binary Search
12)a)	Insertion Sort
12)b)	Selection Sort
13)	Merge Sort
14)a)	Open Addressing Hashing Technique
14)b)	Quadratic Probing

Ex. No.1.a **Array implementation of Stack**
Date:

Aim

To implement a stack operations using array.

Algorithm

1. Start
2. Define a array *stack* of size *max* = 5
3. Initialize *top* = -1
4. Display a menu listing stack operations
5. Accept choice
6. If choice = 1 then
   If top < max -1
      Increment top
      Store element at current position of top
   Else
      Print Stack overflow
   Else If choice = 2 then If top < 0
   then
      Print Stack underflow
   Else
      Display current top element Decrement
      top
   Else If choice = 3 then
   Display stack elements starting from top
7. Stop

Program

/* Stack Operation using Arrays */

```
#include <stdio.h>
#include<conio.h>
 #define max 5
static intstack[max];
int top = -1;
void push(int x)
{
     stack[++top] = x;
}
int pop()
{
     return (stack[top--]);
}
void view()
{
     int i;
     if (top < 0)
     printf("\n Stack Empty\n");
```

```
        else
        {
            printf("\n Top-->");
            for(i=top; i>=0; i--)
            {
                printf("%4d", stack[i]);
            }
            printf("\n");
        }}
    void main()
    {
        int ch=0, val;
        clrscr();
        while(ch != 4)
        {
            printf("\nSTACK        OPERATION \n");
            printf("1.PUSH          ");
            printf("2.POP  ");
            printf("3.VIEW ");
            printf("4.QUIT \n");
            printf("Enter Choice :");
            scanf("%d", &ch);
            switch(ch)
            {
                case 1:
                    if(top < max-1)
                    {
                        printf("\nEnter Stack element : ");
                        scanf("%d", &val);
                        push(val);
                    }
                    else
                    printf("\n Stack Overflow\n");
                    break;

                case 2:
                    if(top < 0)
                    printf("\n Stack Underflow\n");
                    else
                    {
                        val = pop();
                        printf("\n Popped element is %d\n",val);
                    }
                break; case 3:
                    view(); break;
                case 4:
                exit(0); default:
                    printf("\n Invalid Choice \n");
            } } }
```

Output

```
STACK         OPERATION
1.PUSH            2.POP    3.VIEW       4.QUIT
Enter Choice : 1
Enter Stack element :12
STACK         OPERATION
1.PUSH            2.POP    3.VIEW       4.QUIT
Enter Choice : 1
Enter Stack element : 23
STACK         OPERATION
1.PUSH            2.POP    3.VIEW       4.QUIT
Enter Choice : 1
Enter Stack element :34
STACK         OPERATION
1.PUSH            2.POP    3.VIEW       4.QUIT
Enter Choice : 1
Enter Stack element :45
STACK         OPERATION
1.PUSH            2.POP    3.VIEW       4.QUIT
Enter Choice : 3
Top-->   45   34     23     12
STACK  OPERATION
1.PUSH            2.POP    3.VIEW       4.QUIT
Enter Choice : 2
Popped element is45
STACK         OPERATION
1.PUSH            2.POP    3.VIEW       4.QUIT
Enter Choice : 3
Top-->   34   23     12
STACK OPERATION
  1.    PUSH      2.POP    3.VIEW       4.QUIT
Enter Choice : 4
```

Result

Thus push and pop operations of a stack was demonstrated using arrays.

Ex. No.1.b **Array implementation of Queue**
Date:

Aim

To implement queue operations using array.

Algorithm

1. Start
2. Define a array queue of size max = 5
3. Initialize front = rear = –1
4. Display a menu listing queue operations
5. Accept choice
6. If choice = 1 then
   If rear < max -1
      Increment rear
      Store element at current position of rear
   Else
      Print Queue Full
   Else If choice = 2 then If
   front = –1 then
      Print Queue empty
   Else
      Display current front element
      Increment front
   Else If choice = 3 then
   Display queue elements starting from front to rear.
7. Stop

Program

/* Queue Operation using Arrays */

```
#include <stdio.h>
#include<conio.h>
#define max 5
static intqueue[max];
int front = -1;
int rear= -1;
void insert(int x)
{
     queue[++rear] =x;
     if (front == -1)
          front = 0;
}
```

```
int remove()
{
    int val;
    val = queue[front];
        if (front==rear &&rear==max-1)
         front = rear = -1;
    else
    front ++;
    return (val);
}
void view()
{
    int i;

    if (front == -1)
    printf("\n Queue Empty\n");
    else
    {
        printf("\n Front-->");
        for(i=front; i<=rear;i++)
            printf("%4d", queue[i]);
        printf("<--Rear\n");
    }}
void main()
{
    int ch= 0,val;
    clrscr();
while(ch != 4)
    {
        printf("\n QUEUE OPERATION\n");
    printf("1.INSERT        ");
        printf("2.DELETE ");
        printf("3.VIEW  ");
        printf("4.QUIT\n");
        printf("Enter Choice :");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                if(rear < max-1)
                {
                    printf("\n Enter element to be inserted : ");
                    scanf("%d", &val);
                    insert(val);
                }
                else
                printf("\n Queue Full\n");
                break;
```

```
                case 2:
                    if(front == -1)
                    printf("\n Queue Empty\n");
                    else
                    {
                        val = remove();
                        printf("\n Element deleted : %d \n",val);
                    }break;
                 case 3:
                    view();
               break;
                case 4:
                exit(0); default:
                    printf("\n Invalid Choice \n");
            } } }
```

Output

```
QUEUE OPERATION
1.INSERT   2.DELETE   3.VIEW   4.QUIT
Enter Choice :1
Enter element to be inserted : 12
QUEUE OPERATION
1.INSERT   2.DELETE   3.VIEW   4.QUIT
Enter Choice :1
Enter element to be inserted : 23
 QUEUE OPERATION
1.INSERT   2.DELETE   3.VIEW   4.QUIT
Enter Choice :1
Enter element to be inserted : 34
 QUEUE OPERATION
1.INSERT   2.DELETE   3.VIEW   4.QUIT
Enter Choice :1
Enter element to be inserted : 45
 QUEUE OPERATION
1.INSERT   2.DELETE   3.VIEW   4.QUIT
Enter Choice :1
Enter element to be inserted : 56
 QUEUE OPERATION
1.INSERT   2.DELETE   3.VIEW   4.QUIT
Enter Choice :1 Queue Full
 QUEUE OPERATION
1. INSERT  2.DELETE   3.VIEW   4.QUIT
Enter Choice : 3
Front-->      12    23    34    45    56    <--Rear
```

Result

Thus the insertion and delete operation of a queue was demonstrated using arrays.

Ex. No.1.c **Array Implementation of Circular queue ADTs**
Date:

Aim

To implement Circular queue operations using array.

Algorithm

1. Start
2. Define a array *queue* of size *max* = 5
3. Initialize *front* = *rear* = –1
4. Display a menu listing queue operations
5. Accept choice
6. If choice = 1 then
    If rear < max -1
        Increment rear
        Store element at current position of rear
    Else
        Print Queue Full
   Else If choice = 2 then If
    front = –1 then
        Print Queue empty
    Else
        Display current front element
        Increment front
   Else If choice = 3 then
    Display queue elements starting from front to rear.
7. Stop

Program:

```
#include <stdio.h>
#include<conio.h>
#define size 5
void insertq(int[], int);
void deleteq(int[]);
void display(int[]);
int front =  - 1;
int rear =  - 1;
int main()
{
   int n, ch;
   int queue[size];
   do
   {
      printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0. Exit");
      printf("\nEnter Choice 0-3? : ");
      scanf("%d", &ch);

 switch (ch)
      {
```

```
            case 1:
                printf("\nEnter number: ");
                scanf("%d", &n);
                insertq(queue, n);
                break;
            case 2:
                deleteq(queue);
                break;
            case 3:
                display(queue);
                break;
        }
    }while (ch != 0);
}
void insertq(int queue[], int item)
{
    if ((front == 0 && rear == size - 1) || (front == rear + 1))
    {
        printf("queue is full");
        return;
    }
    else if (rear ==  - 1)
    {
        rear++;
        front++;
    }
    else if (rear == size - 1 && front > 0)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    queue[rear] = item;
}

void display(int queue[])
{
    int i;
    printf("\n");
    if (front > rear)
    {
        for (i = front; i < size; i++)
        {
            printf("%d ", queue[i]);
        }
```

```
        for (i = 0; i <= rear; i++)
            printf("%d ", queue[i]);
    }
    else
    {
        for (i = front; i <= rear; i++)
            printf("%d ", queue[i]);
    }
}
void deleteq(int queue[])
{
    if (front ==  - 1)
    {
        printf("Queue is empty ");
    }
    else if (front == rear)
    {
        printf("\n %d deleted", queue[front]);
        front =  - 1;
        rear =  - 1;
    }
    else
    {
        printf("\n %d deleted", queue[front]);
        front++;
    } }
```

Output:

```
Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 1
Enter number: 12
Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 1
Enter number: 13
Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 1
Enter number: 14
```

```
Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 3
12 13 14
Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 0
```

Result

Thus the insertion and delete operation of a circular queue was demonstrated using arrays.

Ex. No. 2 **Singly Linked List**
Date:

Aim

To implement a singly linked list node and perform operations such as insertions and deletions dynamically.

Algorithm

1. Start
2. Define single linked list *node* as self referential structure
3. Create *Head* node with label = -1 and next = NULL using
4. Display menu on list operation
5. Accept user choice
6. If choice = 1 then
   Locate node after which insertion is to be done
   Create a new node and get data part
   Insert new node at appropriate position by manipulating address
   Else if choice = 2
   Get node's data to be deleted. Locate
   the node and delink the node
   Rearrange the links
   Else
   Traverse the list from Head node to node which points to null
7. Stop

Program

/* Single Linked List */

```
#include <stdio.h>
 #include <conio.h>
 #include<process.h>
#include <alloc.h>
 #include <string.h>
struct node
{
      int label;
      struct node *next;
};
void main()
{
      int ch, fou=0;
      int k;
      struct node *h, *temp, *head, *h1;
      /* Head node construction */
      head = (struct node*) malloc(sizeof(struct node));
      head->label = -1;
      head->next = NULL;
      while(-1)
      {
            clrscr();
            printf("\n\n SINGLY LINKED LIST OPERATIONS\n");
            printf("1->Add  ");
            printf("2->Delete   ");
            printf("3->View     ");
            printf("4->Exit \n");
            printf("Enter your choice :");
            scanf("%d", &ch);
            switch(ch)
            {

                  case 1:
                        printf("\n Enter label after which to add : ");
                        scanf("%d", &k);
                        h = head;
                        fou = 0;
                              if (h->label ==k)
                              fou = 1;
                    while(h->next != NULL)
                        {
                              if (h->label == k)
                              {
                                          fou=1;
                                          break;
                              }
```

```
                    h = h->next;
                }
            if (h->label ==k)
            fou = 1;
            if (fou != 1)
                printf("Node notfound\n");
                else
            {
            temp=(structnode*)(malloc(sizeof(structnode)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);
                temp->next =h->next;
                 h->next = temp;
            }
            break;

        /* Delete any intermediate node */
        case 2:
            printf("Enter label of node to be deleted\n");
            scanf("%d", &k);
            fou = 0;
            h = h1 = head;
            while (h->next != NULL)
            {
                h = h->next;
                if (h->label == k)
                {
                        fou = 1; break;
                }}
            if (fou == 0)
                printf("Sorry Node notfound\n");
                else
                {
                while (h1->next !=h)
                h1 = h1->next;
                h1->next =h->next; free(h);
                printf("Node deleted successfully \n");
            }
            break;
         case 3:
            printf("\n\n HEAD ->");
             h=head;
            while (h->next != NULL)
            {
                h = h->next;
                printf("%d -> ",h->label);
            }
            printf("NULL");
            break;
        case 4:
            exit(0);
}}}
```

Output

SINGLY LINKED LIST OPERATIONS
1->Add 2->Delete 3->View 4->Exit Enter
your choice : 1
Enter label after which new node is to be added : -1
Enter label for new node : 23

SINGLY LINKED LISTOPERATIONS
1->Add 2->Delete 3->View 4->Exit Enter
your choice : 1
Enter label after which new node is to be added : 23
Enter label for new node : 67

SINGLY LINKED LISTOPERATIONS
2- >Add 2->Delete 3->View 4->Exit Enter
your choice : 3
HEAD -> 23 -> 67 -> NULL

Result:

Thus the insertion and deletion operation on single linked list was demonstrated successfully.

Ex. No. 3.a **Stack Using Linked List**
Date:

Aim

To implement stack operations using linked list.

Algorithm

1. Start
2. Define a singly linked list node for stack
3. Create Head node
4. Display a menu listing stack operations
5. Accept choice
6. If choice = 1 then
   Create a new node with data Make
   new node point to first node
   Make head node point to new node
   Else If choice = 2 then
   Make temp node point to first node
   Make head node point to next of temp node
   Release memory
   Else If choice = 3 then
   Display stack elements starting from head node till null
7. Stop

Program

```
/* Stack using Single Linked List */

#include <stdio.h>
#include <conio.h>
 #include<process.h>
#include <alloc.h>
struct node
{
    int label;
    struct node *next;
};
void main()
{
    int ch = 0;
     int k;
    struct node *h, *temp, *head;

    /* Head node construction */
    head = (struct node*) malloc(sizeof(struct node));
    head->next = NULL;
```

```
while(1)
{
    printf("\n Stack using Linked List \n");
    printf("1->Push        ");
    printf("2->Pop         ");
    printf("3->View        ");
     printf("4->Exit\n");
    printf("Enter your choice :");
    scanf("%d", &ch);
    switch(ch)
    {
        case 1:
            /* Create a new node */
            temp=(struct node*)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);
            h = head;
            temp->next =h->next;
            h->next = temp; break;

        case 2:
            /* Delink the first node*/
            h = head->next;
            head->next = h->next;
            printf("Node %s deleted\n", h->label); free(h);
            break;

        case 3:
            printf("\n HEAD ->");
            h = head;
            /* Loop till last node*/
            while(h->next != NULL)
            {
                h = h->next;
                printf("%d -> ",h->label);
            }
            printf("NULL \n"); break;

        case 4:
            exit(0);
    }
    }
    }
```

Output

```
 Stack using Linked List
1->Push    2->Pop    3->View    4->Exit Enter
your choice : 1
Enter label for new node :23
New node added
 Stack using Linked List
1->Push    2->Pop    3->View    4->Exit Enter
your choice : 1
Enter label for new node : 34
Stack using Linked List
1->Push    2->Pop    3->View    4->Exit Enter
your choice : 3
HEAD -> 34 -> 23 -> NULL
```

Result

Thus push and pop operations of a stack was demonstrated using linked list.

Ex. No. 3.b **Queue Using Linked List**
Date:

Aim

To implement queue operations using linked list.

Algorithm

1. Start
2. Define a singly linked list node for stack
3. Create Head node
4. Display a menu listing stack operations
5. Accept choice
6. If choice = 1 then
   Create a new node with data
   Make new node point to first
   node
   Make head node point to new
   node Else If choice = 2 then
   Make temp node point to first node
   Make head node point to next of temp
   node Release memory
   Else If choice = 3 then
   Display stack elements starting from head node till null
7. Stop

Program

```
/* Queue using Linked List */

#include <stdio.h>
#include <conio.h>
 #include<process.h>
#include <alloc.h>

struct node
{
    int label;
    struct node *next;
};

Void main()
{
    int ch=0; int k;
    struct node *h, *temp, *head;

    /* Head node construction */
    head = (struct node*) malloc(sizeof(struct node));
    head->next = NULL;

    while(1)
    {
        printf("\n Queue using Linked List \n");
        printf("1->Insert           ");
        printf("2->Delete       ");
        printf("3->View         ");
        printf("4->Exit \n");
        printf("Enter your choice :");
        scanf("%d", &ch);

        switch(ch)
        {
            case 1:
                /* Create a new node */
                temp=(struct node *)(malloc(sizeof(struct node)));
                printf("Enter label for new node : ");
                scanf("%d", &temp->label);
                h = head;
                while (h->next !=NULL)
                h = h->next;
                h->next = temp;
                temp->next =NULL;
                break;
```

```
                        case 2:
                        h = head->next;
                        head->next = h->next;
                         printf("Node deleted\n");
                         free(h);
                        break;
                        case 3:
                        printf("\n\nHEAD ->");
                         h=head;
                        while (h->next!=NULL)
                        {
                            h = h->next;
                            printf("%d -> ",h->label);
                        }
                        printf("NULL\n");
                        break;
                        case 4:
                        exit(0);
                }
        }
}
```

Output

```
  Queue using Linked List
1->Insert     2->Delete     3->View     4->Exit
 Enter your choice : 1
Enter label for new node : 12

  Queue using Linked List
1->Insert     2->Delete     3->View     4->Exit
 Enter your choice : 1
Enter label for new node : 23

  Queue using Linked List
1->Insert     2->Delete     3->View     4->Exit
2-Enter your choice : 3
HEAD -> 12 -> 23 -> NULL
```

.

Result

Thus insert and delete operations of a Queue was demonstrated using linked list.

Ex. No. 4.a **Polynomial addition**
Date:

Aim

To perform addition of two polynomial using linked list.

Algorithm

1. Start
2. Define a array size *a,b,c* = 10
3. Read the values of m,n,k,k
4. Let a and b be the two polynomials represented by the linked list.
   1. while a and b are not null, repeat step 2.
   2. If powers of the two terms are equal then if the terms do not cancel then insert the sum of the terms into the sum Polynomial
   Advance a
   Advance b
   Else if the power of the first polynomial> power of second
   Then insert the term from first polynomial into sum polynomial
   Advance a
   Else insert the term from second polynomial into sum polynomial
   Advance b
   3. Copy the remaining terms from the non empty polynomial into the sum polynomial.
5. Stop

Program

```
#include<stdio.h>
#include<conio.h>
Int main()
{
int a[10], b[10], c[10],m,n,k,k1,i,j,x;
clrscr();
printf("\n\tPolynomial Addition\n");
printf("\t===================\n");
printf("\n\tEnter the no. of terms of the polynomial:");
scanf("%d", &m);
printf("\n\tEnter the degrees and coefficients:");
for (i=0;i<2*m;i++)
scanf("%d", &a[i]);
printf("\n\tFirst polynomial is:");
k1=0;
if(a[k1+1]==1)
printf("x^%d", a[k1]);
else
printf("%dx^%d", a[k1+1],a[k1]);
k1+=2;
```

```
while (k1<i)
{
printf("+%dx^%d", a[k1+1],a[k1]);
k1+=2;
}
printf("\n\n\n\tEnter the no. of terms of 2nd polynomial:");
scanf("%d", &n);
printf("\n\tEnter the degrees and co-efficients:");
for(j=0;j<2*n;j++)
scanf("%d", &b[j]);
printf("\n\tSecond polynomial is:");
k1=0;
if(b[k1+1]==1)
printf("x^%d", b[k1]);
else
printf("%dx^%d",b[k1+1],b[k1]);
k1+=2;
while (k1<2*n)
{
printf("+%dx^%d", b[k1+1],b[k1]);
k1+=2;
}
i=0;
j=0;
k=0;
while (m>0 && n>0)
{
if (a[i]==b[j])
{
c[k+1]=a[i+1]+b[j+1];
c[k]=a[i];
m--;
n--;
i+=2;
j+=2;
}
else if (a[i]>b[j])
{
c[k+1]=a[i+1];
c[k]=a[i];
m--;
i+=2;
}
else
{
c[k+1]=b[j+1];
c[k]=b[j];
n--;
j+=2;
}
```

```
k+=2;
}
while (m>0)
{
c[k+1]=a[i+1];
c[k]=a[i];
k+=2;
i+=2;
m--;
}
while (n>0)
{
c[k+1]=b[j+1];
c[k]=b[j];
k+=2;
j+=2;
n--;
}
printf("\n\n\n\n\tSum of the two polynomials is:");
k1=0;
if (c[k1+1]==1)
printf("x^%d", c[k1]);
else
printf("%dx^%d", c[k1+1],c[k1]);
k1+=2;
while (k1<k)
{
if (c[k1+1]==1)
printf("+x^%d", c[k1]);
else
printf("+%dx^%d", c[k1+1], c[k1]);
k1+=2;
}
getch();
return 0;
}
```

Output:

Polynomial Addition

Enter the no. of terms of the polynomial:3
Enter the degrees and coefficients:3 5 2 6 1 8
First polynomial is:5x^3+6x^2+8x^1

Enter the no. of terms of 2nd polynomial:2
Enter the degrees and co-efficients:3 6 2 9
Second polynomial is:6x^3+9x^2

Sum of the two polynomials is:11x^3+15x^2+8x^1

Result

Thus the addition of two polynomial was demonstrated using linked list.

Ex. No. 5.a **Evaluating Postfix Expression**
Date:

Aim

To evaluate the given postfix expression using stack operations.

Algorithm

1. Start
2. Define a array *stack* of size *max* = 20
3. Initialize *top* = -1
4. Read the postfix expression character-by-character
   If character is an operand push it onto the
   stack If character is an operator
      Pop topmost two elements from stack.
      Apply operator on the elements and push the result onto the stack,
5. Eventually only result will be in the stack at end of the expression.
6. Pop the result and print it.
7. Stop

Program:

```
#include<stdio.h>
#include<conio.h>
Struct stack
{
 int top,float a[50];
} s;

main()
  {char pf[50];
        float d1,d2,d3;
         int i; clrscr();
        s.top = -1;
        printf("\n\n Enter the postfix expression: ");
        gets(pf);
        for(i=0; pf[i]!='\0'; i++)
        {
              switch(pf[i])
              {
                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
                    case '7':
                    case '8':
                    case '9':
                          s.a[++s.top] =pf[i]-'0';
                          break;

                    case '+':
                          d1 = s.a[s.top--];
                          d2 = s.a[s.top--];
                          s.a[++s.top] = d1 +d2;
                          break;

                    case '-':
                          d2 =s.a[s.top--];
                          d1 =s.a[s.top--];
                          s.a[++s.top] = d1 -d2;
                          break;
          case '*':
                          d2 = s.a[s.top--];
                          d1 = s.a[s.top--];
                s.a[++s.top] =d1*d2;
                          break;
```

```
            case '/':
                d2 = s.a[s.top--];
                d1 = s.a[s.top--];
                s.a[++s.top] = d1 /d2;
                break;
        }}
    printf("\n Expression value is %5.2f", s.a[s.top]);
    getch();
}
```

Output

```
Enter the postfix expression: 6523+8*+3+*
 Expression value is 288.00
```

Result

Thus the given postfix expression was evaluated using stack.

Ex. No.5.b **Infix To Postfix Conversion**
Date:

Aim

To convert infix expression to its postfix form using stack operations.

Algorithm

1. Start
2. Define a array *stack* of size *max* = 20
3. Initialize *top* = -1
4. Read the infix expression character-by-character If character is an operand print it
   If character is an operator
      Compare the operator's priority with the stack[top] operator.
      If the stack [top] has higher/equal priority than the input operator, Pop it from the stack and print it.
      Else
      Push the input operator onto the stack
   If character is a left parenthesis, then push it onto the stack.
   If character is a right parenthesis, pop all operators from stack and print it until a left parenthesis is encountered. Do not print the parenthesis.
   If character = $ then Pop out all operators, Print them and Stop

Program

/* Conversion of infix to postfix expression */

```
#include <stdio.h>
#include  <conio.h>
#include<string.h>
#define MAX 20
int top = -1;
charstack[MAX];
char pop();
void push(charitem);
int prcd(charsymbol)
{
    switch(symbol)
    {
        case '+':
        case '-':
            return 2;
            break;
```

```
            case '*':
            case '/':
                return 4;
                break;
            case '^':
            case '$':
                return 6;
                break;
            case '(':
            case ')':
            case '#':
                return 1;
          break;
        }
}

int isoperator(char symbol)
{
        switch(symbol)
        {
            case '+':
            case '-':
            case '*':
            case '/':
            case '^':
            case '$':
            case '(':
            case ')':
                return 1;
         break;
          default:
                return 0;
        }
}

void convertip(char infix[],char postfix[])
{
        int i,symbol,j = 0;
        stack[++top] = '#';
        for(i=0;i<strlen(infix);i++)
        {
            symbol = infix[i];
        if(isoperator(symbol) ==0)
            {
                postfix[j] =symbol;
          j++;
            }
```

```
                else
                {
                        if(symbol =='(')
                   push(symbol);
                     else if(symbol == ')')
                     {
                          while(stack[top] != '(')
                          {
                               postfix[j] =pop();
                        j++;
                          }
                          pop();
                          //pop out (.
                     }
                     else
                     {
                               if(prcd(symbol) >prcd(stack[top]))
                        push(symbol);
                          else
                          {
                        while(prcd(symbol) <= prcd(stack[top]))
                        {
                                postfix[j] =pop();
                                 j++;
                        }
                        push(symbol);
                   }}}}

     while(stack[top] != '#')
     {
          postfix[j] = pop();

          j++;
     }
     postfix[j] = '\0';
}

Void main()
{
     charinfix[20],postfix[20]; clrscr();
     printf("Enter the valid infix string: ");
     gets(infix);
     convertip(infix, postfix);
     printf("The corresponding postfix string is: ");
     puts(postfix);
     getch();
}
```

```
void push(char item)
{
	top++;
	stack[top] = item;
}
char pop()
{
	char a;
	a =stack[top]; top-
	-;
	return a;
}
```

Output

Enter the valid infix string: (a+b*c)/(d$e) T
The corresponding postfix string is: abc*+de$/

Enter the valid infix string: a*b+c*d/e
The corresponding postfix string is: ab*cd*e/+

Enter the valid infix string: a+b*c+(d*e+f)*g
The corresponding postfix string is:abc*+de*f+g*+

Result

Thus the given infix expression was converted into postfix form using stack.

Ex. No. 6 **Binary Search Tree**
Date:

Aim

To insert and delete nodes in a binary search tree.

Algorithm

1. Create a structure with key and 2 pointer variable left and right.
2. Read the node to be inserted.
   If (root==NULL)
   root=node
   else if (root->key<node-
   >key) root-
   >right=NULL
   else
   Root->left=node
3. For
   Deletion
   if it is a leaf node
   Remove immediately
   Remove pointer between del node &
   child if it is having one child
   Remove link between del
   node&child Link delnode is child with
   delnodes parent
   If it is a node with a children
   Find min value in right
   subtree Copy min value to
   delnode place Delete the
   duplicate

4. Stop

Program

/* Binary Search Tree */

```
#include <stdio.h>
#include<stdlib.h>
struct node
{
    int key;
    struct node *left;
    struct node*right;
};
struct node *newNode(int item)
{
    struct node *temp = (struct node*)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right =NULL;
    return temp;
}
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ",root->key);
        inorder(root->right);
    }
}
struct node* insert(struct node* node, int key)
{
        if (node == NULL) return
      newNode(key);
    if (key < node->key)
      node->left= insert(node->left,key);
       else
    node->right=insert(node->right, key);
    return node;
}
struct node * minValueNode(struct node* node)
{
    struct node* current = node;
    while (current->left !=NULL)
    current =current->left;
    return current;
}
```

```
struct node* deleteNode(struct node* root, intkey)
{
      struct node
*temp;
       if (root == NULL)
            return root;
      if (key < root->key)
      root->left=deleteNode(root->left,key);
      else if (key > root->key)
      root->right=deleteNode(root->right,key);
      else
      {
            if (root->left == NULL)
            {
                  temp = root->right;
       free(root);
                  return temp;
            }
            else if (root->right == NULL)
            {
                  temp =root->left;
       free(root);
                  return temp;
            }
            temp =minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
      }
      return root;
}

Void main()
{
      struct node *root =NULL;
      root = insert(root, 50);
      root = insert(root, 30);
      root = insert(root, 20);
      root = insert(root, 40);
      root = insert(root, 70);
      root = insert(root, 60);
      root = insert(root,80);
      printf("Inorder traversal of the given tree \n");
       inorder(root);
      printf("\nDelete 20\n"); r
      oot = deleteNode(root,20);
      printf("Inorder traversal of the modified tree \n");
      inorder(root);
      printf("\nDelete 30\n");
       root = deleteNode(root,30);
      printf("Inorder traversal of the modified tree \n");
      inorder(root);
      printf("\nDelete 50\n");
      root = deleteNode(root, 50);
```

```
        printf("Inorder traversal of the modified tree \n");
        inorder(root);
    }
```

Output:

Inorder traversal of the given tree 20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree 30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree 40 50 60 70 80
Delete 50
Inorder traversal of the modified tree 40 60 70 80

Result

Thus nodes were inserted and deleted from a binary search tree.

Ex. No. 7 **Implementation of AVL TREE**
Date:

Aim

To write a C program for implementation of AVL Tree.

Algorithm

1. Start
2. Declare the node with left link, right link, data and height of node.
3. Enter the number of elements to be inserted.
4. While inserting each element the height of each node will be checked.
5. If the height difference of left and right node is equal to 2 for an node then the height is unbalanced at the node.
6. The present node while inserting a new node t left subtree then perform rotation with left child otherwise rotation with right child.
7. Height is unbalanced at grandparent node while inserting a new node at right sub tree of parent node then performs double rotation with left.
8. Height is unbalanced at grandparent node while inserting a new node then perform double rotation with right.
9. Stop.

Program

```
# include<stdio.h>
#include<conio.h>
Typedef struct node
{
int data;
struct node *left,*right;
int ht;
}node;

node *insert(node *,int);
node *Delete(node *,int);
void preorder(node *);
void inorder(node *);
int height( node *);
node *rotateright(node *);
node *rotateleft(node *);
node *RR(node *);
node *LL(node *);
node *LR(node *);
node *RL(node *);
int BF(node *);
int main()
{
node *root=NULL;
int x,n,i,op;
do
{
printf("\n1)Create:");
printf("\n2)Insert:");
printf("\n3)Delete:");
```

```
printf("\n4)Print:");
printf("\n5)Quit:");
printf("\n\nEnter Your Choice:");
scanf("%d",&op);
switch(op)
{
case 1: printf("\nEnter no. of elements:");
scanf("%d",&n);
printf("\nEnter tree data:");
root=NULL;
for(i=0;i<n;i++)
{
scanf("%d",&x);
root=insert(root,x);
}
break;
case 2: printf("\nEnter a data:");
scanf("%d",&x);
root=insert(root,x);
break;
case 3: printf("\nEnter a data:");
scanf("%d",&x);
root=Delete(root,x);
break;
case 4: printf("\nPreorder sequence:\n");
preorder(root);
printf("\n\nInorder sequence:\n");
inorder(root);
printf("\n");
break;
}
}while(op!=5);
return 0;
}

node * insert(node *T,int x)
{
if(T==NULL)
{
T=(node*)malloc(sizeof(node));
T->data=x;
T->left=NULL;
T->right=NULL;
}
else
if(x > T->data) // insert in right subtree
{
T->right=insert(T->right,x);
if(BF(T)==-2)
if(x>T->right->data)
T=RR(T);
```

```
else
T=RL(T);
}
else
if(x<T->data)
{
T->left=insert(T->left,x);
if(BF(T)==2)
if(x < T->left->data)
T=LL(T);
else
T=LR(T);
}
T->ht=height(T);
return(T);
}

node * Delete(node *T,int x)
{
node *p;
if(T==NULL)
{
return NULL;
}
else
if(x > T->data) // insert in right subtree
{
T->right=Delete(T->right,x);
if(BF(T)==2)
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);
}
else
if(x<T->data)
{
T->left=Delete(T->left,x);
if(BF(T)==-2) //Rebalance during windup
if(BF(T->right)<=0)
T=RR(T);
else
T=RL(T);
}
else
{
//data to be deleted is found
if(T->right!=NULL)
{ //delete its inorder succesor
p=T->right;
```

```
while(p->left!= NULL)
p=p->left;
T->data=p->data;
T->right=Delete(T->right,p->data);
if(BF(T)==2)//Rebalance during windup
if(BF(T->left)>=0)
T=LL(T);
else
T=LR(T);\
}
else
return(T->left);
}
T->ht=height(T);
return(T);
}

int height(node *T)
{
int lh,rh;
if(T==NULL)
return(0);
if(T->left==NULL)
lh=0;
else
lh=1+T->left->ht;
if(T->right==NULL)
rh=0;
else
rh=1+T->right->ht;
if(lh>rh)
return(lh);
return(rh);
}
node * rotateright(node *x)
{
node *y;
y=x->left;
x->left=y->right;
y->right=x;
x->ht=height(x);
y->ht=height(y);
return(y);
}
node * rotateleft(node *x)
{
node *y;
y=x->right;
x->right=y->left;
y->left=x;
x->ht=height(x);
```

```
y->ht=height(y);
return(y);
}

node * RR(node *T)
{
T=rotateleft(T);
return(T);
}
node * LL(node *T)
{
T=rotateright(T);
return(T);
}

node * LR(node *T)
{
T->left=rotateleft(T->left);
T=rotateright(T);
return(T);
}

node * RL(node *T)
{
T->right=rotateright(T->right);
T=rotateleft(T);
return(T);
}
int BF(node *T)
{
int lh,rh;
if(T==NULL)
return(0);

if(T->left==NULL)
lh=0;
else
lh=1+T->left->ht;

if(T->right==NULL)
rh=0;
else
rh=1+T->right->ht;

return(lh-rh);
}
```

```
void preorder(node *T)
{
if(T!=NULL)
{

printf("%d(Bf=%d)",T->data,BF(T));
preorder(T->left);
preorder(T->right);
}
}
void inorder(node *T)
{
if(T!=NULL)
{
inorder(T->left);
printf("%d(Bf=%d)",T->data,BF(T));
inorder(T->right);
}
}
```

Output

```
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:1
Enter no. of elements:4
Enter tree data:7 12 4 9
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:4
Preorder sequence:
7(Bf=-1)4(Bf=0)12(Bf=1)9(Bf=0)
Inorder sequence:
4(Bf=0)7(Bf=-1)9(Bf=0)12(Bf=1)
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
```

```
Enter Your Choice:3
Enter a data:7
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:4
Preorder sequence:9
(Bf=0)4(Bf=0)12(Bf=0)
Inorder sequence:
4(Bf=0)9(Bf=0)12(Bf=0)
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:
Enter Your Choice:5
```

Result:

Thus the implementation of AVL tree was executed successfully.

Ex. No. 8 **Heap using Priority Queue**
Date:

Aim
To implement priority queue to add and delete elements.

Program

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
void insert_by_priority(int);
void delete_by_priority(int);
void create();
void check(int);
void display_pqueue();
int pri_que[MAX];
int front, rear;
void main()
 {
 int n, ch;
 printf("\n1 - Insert an element into queue");
printf("\n2 - Delete an element from queue");
 printf("\n3 - Display queue elements");
 printf("\n4 - Exit");
  create();
  while (1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nEnter value to be inserted : ");
            scanf("%d",&n);
        insert_by_priority(n);
            break;
        case 2:
            printf("\nEnter value to delete : ");
            scanf("%d",&n);
            delete_by_priority(n);
            break;
```

```
        case 3:
            display_pqueue();
            break;
        case 4:
            exit(0);
        default:
            printf("\nChoice is incorrect, Enter a correct choice");
        }
    }
}

/* Function to create an empty priority
queue */
void create()
{
    front = rear = -1;
}

/* Function to insert value into priority
queue */
void insert_by_priority(int data)
{
    if (rear >= MAX - 1)
    {
        printf("\nQueue overflow no more elements can be inserted");
        return;
    }
    if ((front == -1) && (rear == -1))
    {
        front++;
        rear++;
        pri_que[rear] = data;
        return;
    }
    Else
check(data);
                        rear++;
}

/* Function to check priority and place
element */
void check(int data)
{
    int i,j;
```

```
    for (i = 0; i <= rear; i++)
    {
        if (data >= pri_que[i])
        {
            for (j = rear + 1; j > i; j--)
            {
                pri_que[j] = pri_que[j - 1];
            }
            pri_que[i] = data;
            return;
        }
    }
    pri_que[i] = data;
}

/* Function to delete an element from
queue */
void delete_by_priority(int data)
{
    int i;

    if ((front==-1) && (rear==-1))
    {
        printf("\nQueue is empty no
elements to delete");
        return;
    }

    for (i = 0; i <= rear; i++)
    {
        if (data == pri_que[i])
        {
            for (; i < rear; i++)
            {
                pri_que[i] = pri_que[i + 1];
            }

        pri_que[i] = -99;
        rear--;

        if (rear == -1)
            front = -1;
        return;
        }
    }
    printf("\n%d not found in queue to
delete", data);
}
```

```
/* Function to display queue elements
*/
void display_pqueue()
{
    if ((front == -1) && (rear == -1))
    {
        printf("\nQueue is empty");
        return;
    }

    for (; front <= rear; front++)
    {
        printf(" %d ", pri_que[front]);
    }

    front = 0;
}
```

Result

Thus heap using priority queue operations performed successfully.

Ex. No. 9 **Dijkstra's Algorithm**

Date:

Aim

To perform single source shortest path using Dijkstra algorithm

Algorithm

1. Start.
2. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.
3. Array visited[ ] is initialized to zero.If the vertex 0 is the source vertex then visited[0] is marked as 1.
4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.
   Initially, distance of source vertex is taken as 0.
5. Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.Recalculate the shortest distance of remaining vertices from the source. Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v.

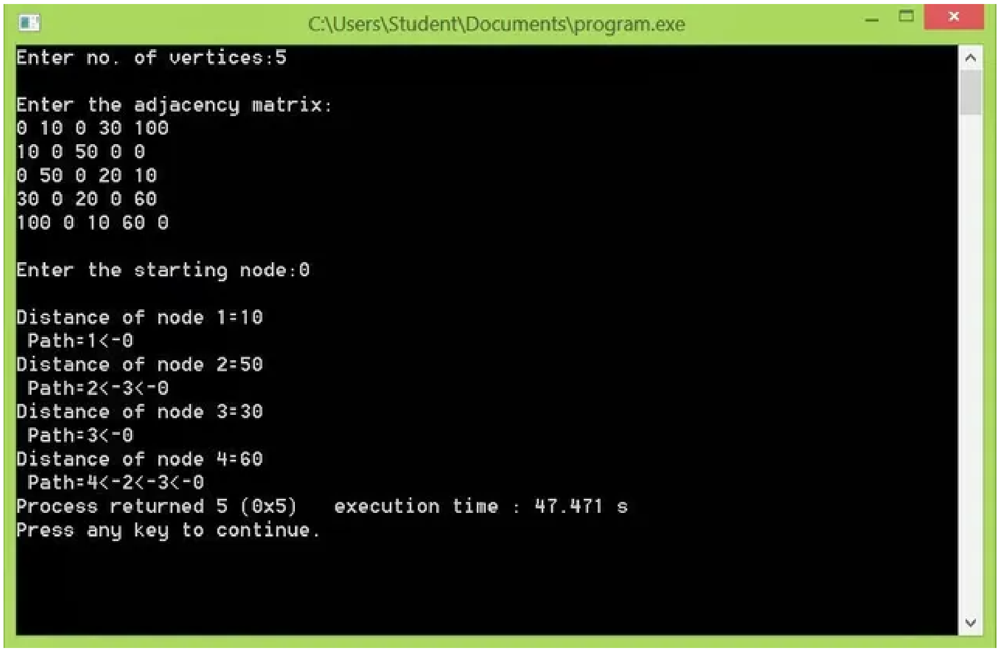
Program

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
 void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{
```

```
 int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
//initialize pred[],distance[] and visited[]
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}

//print the path and distance of each node
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
```

```
}while(j!=startnode);
}
}
```



Result:

Thus the program is to find the shortest path using dijikstra's algorithm was executed successfully.

Ex. No. 10 Prim's Algorithm

Date:

Aim

To create spanning tree with minimum weight from a given weighted graph.

Algorithm

1. Start
2. Create edge list of given graph, with their weights.
3. Draw all nodes to create skeleton for spanning tree.
4. Select an edge with lowest weight and add it to skeleton and delete edge from edge list.
5. Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.
6. Repeat step 5 until n-1 edges are added.
7. Stop

Program

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20
 int G[MAX][MAX],spanning[MAX][MAX],n;
 int prims();
 int main()
{
int i,j,total_cost;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
total_cost=prims();
printf("\nspanning tree matrix:\n");
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
printf("%d\t",spanning[i][j]);
}
printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}
```

```
int prims()
{
int cost[MAX][MAX];
int u,v,min_distance,distance[MAX],from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;
//create cost[][] matrix,spanning[][]
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
}
//initialise visited[],distance[] and from[]
distance[0]=0;
visited[0]=1;
for(i=1;i<n;i++)
{
distance[i]=cost[0][i];
from[i]=0;
visited[i]=0;
}
min_cost=0; //cost of spanning tree
no_of_edges=n-1; //no. of edges to be added
while(no_of_edges>0)
{
//find the vertex at minimum distance from the tree
min_distance=infinity;
for(i=1;i<n;i++)
if(visited[i]==0&&distance[i]<min_distance)
{
v=i;
min_distance=distance[i];
}
u=from[v];
//insert the edge in spanning tree
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;
//updated the distance[] array
for(i=1;i<n;i++)
if(visited[i]==0&&cost[i][v]<distance[i])
{
distance[i]=cost[i][v];
from[i]=v;
}
min_cost=min_cost+cost[u][v];
}
return(min_cost);
}
```

Output:

Enter no. of vertices:6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

spanning tree matrix:

0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0

Result

Thus the spanning tree with minimum weight from a given weighted graph was executed successfully.

Ex. No. 11.a **Linear Search**
Date:

Aim

To perform linear search of an element on the given array.

Algorithm

1. Start
2. Read number of array elements *n*
3. Read array elements A_i, $i = 0,1,2,\ldots n-1$
4. Read *search* value
5. Assign 0 to *found*
6. Check each array element against *search*
   If A_i = *search* then
   found = 1
   Print "Element
   found" Print position
   i
   Stop
7. If *found* = 0 then
   print "Element not found"
8. Stop

Program

```
/* Linear search on a sorted array */

#include <stdio.h>
#include<conio.h>
Void main()
{
    int a[50],i, n, val,found;
    clrscr();
    printf("Enter number of elements : ");
    scanf("%d", &n);
    printf("Enter Array Elements : \n");
     for(i=0; i<n; i++)
     scanf("%d", &a[i]);
    printf("Enter element to locate : ");
    scanf("%d", &val);
    found = 0;
     for(i=0; i<n;i++)
    {
        if (a[i] == val)
        {
            printf("Element found at position %d", i);
            found = 1;
            break;
        }
    }
    if (found == 0)
        printf("\n Element notfound");
        getch();
}
```

Output :

```
Enter number of elements :7
Enter Array Elements :
23   6   12   5   0   32   10
Enter element to locate :5
Element found at position3
```

Result

Thus an array was linearly searched for an element's existence.

Ex. No. 11.b **Binary Search**
Date:

Aim

To locate an element in a sorted array using Binary search method

Algorithm

1. Start
2. Read number of array elements, say *n*
3. Create an array *arr* consisting *n* sorted elements
4. Get element, say *key* to be located
5. Assign 0 to *lower* and *n* to *upper*
6. While (*lower* < *upper*)

    Determine middle element *mid* =
    (upper+lower)/2 If key = arr[mid] then

        Print mid
        Stop

    Else if key > arr[mid]

        then lower = mid
        + 1

    else

        upper = mid – 1

7. Print "Element not found"
8. Stop

Program

/* Binary Search on a sorted array */

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

Void main()
{
    int a[50],i, n, upper, lower, mid, val, found;
    clrscr();
    printf("Enter array size            : ");
    scanf("%d", &n);
    for(i=0; i<n;i++)
     a[i] = 2 * i;
    printf("\n Elements in Sorted Order \n");
    for(i=0; i<n; i++)
        printf("%4d", a[i]);
    printf("\n Enter element to locate : ");
    scanf("%d", &val);
    upper = n;
    lower = 0;
    found = -1;
    while (lower <= upper)
    {
        mid = (upper +lower)/2;
        if (a[mid] == val)
        {
            printf("Located at position %d", mid);
            found = 1;
            break;
        }
        else if(a[mid] >val)
        upper = mid - 1;
        else
            lower = mid + 1;
    }

    if (found == -1) printf("Element notfound");
    getch();
}
```

Output

```
Enter array size        : 9
Elements in SortedOrder
    0     2     4     6     8    10    12    14    16
Enter element to locate :12
Located at position 6


Enter array size        : 10
  Elements in SortedOrder
    0     2     4     6     8    10    12    14    16    18
  Enter element to locate :13
   Element not found
```

Result

Thus an element is located quickly using binary search method.

Ex. No. 12.a **Insertion Sort**
Date:

Aim

To sort an array of N numbers using Insertion sort.

Algorithm

1. Start
2. Read number of array elements *n*
3. Read array elements A_i
4. Sort the elements using insertion sort
   In pass p, move the element in position p left until its correct place is found among the first p + 1 elements.
   Element at position p is saved in temp, and all larger elements (prior to position p) are moved one spot to the right. Then temp is placed in the correct spot.
5. Stop

Program

/* Insertion Sort*/

```
Void  main()
{
     int i, j, k, n, temp, a[20], p=0;

     printf("Enter total elements:");
     scanf("%d",&n);
     printf("Enter array elements:");
      for(i=0; i<n; i++)
          scanf("%d", &a[i]);
     for(i=1; i<n; i++)
     {
          temp = a[i]; j = i - 1;
          while((temp<a[j]) && (j>=0))
          {
               a[j+1] = a[j];
               j = j - 1;
          }
          a[j+1] = temp;
          p++;
          printf("\n After Pass %d: ", p);
           for(k=0; k<n; k++)
               printf("    %d", a[k]);
     }

     printf("\n Sorted List :");
     for(i=0; i<n; i++)
```

```
                printf(" %d", a[i]);
        }
```

Output

```
Enter total elements: 6
Enter array elements: 34 8 64 51 32 21
  After Pass 1:     8   34   64   51   32   21
  After Pass 2:     8   34   64   51   32   21
  After Pass 3:     8   34   51   64   32   21
  After Pass 4:     8   32   34   51   64   21
  After Pass 5:     8   21   32   34   51   64
  Sorted List :     8   21   32   34   51   64
```

Result

Thus an array element was sorted using insertion sort.

Ex. No. 12.b **Selection Sort**

Date:

Aim

To sort an array of N numbers using Insertion sort.

Algorithm

1. Start
2. Read number of array elements *n*
3. Read array elements
4. Sort the elements using selection sort
5. Set the first element of the array as minimum.
6. Compare the minimum with the next element, if it is smaller than minimum assign this element as minimum. Do this till the end of the array.
7. Place the minimum at the first position(index 0) of the array.for the next iteration, start sorting from the first unsorted element
8. Stop

Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int arr[10]={6,12,0,18,11,99,55,45,34,2};
   int n=10;
   int i, j, position, swap;
   for (i = 0; i < (n - 1); i++) {
      position = i;
      for (j = i + 1; j < n; j++) {
         if (arr[position] > arr[j])
            position = j;
      }
      if (position != i) {
         swap = arr[i];
         arr[i] = arr[position];
         arr[position] = swap;
      }    }
   for (i = 0; i < n; i++)
      printf("%d\t", arr[i]);
   return 0;
}
```

Output:

```
0   2   6   11   12   18   34   45   55   99
```

Result

Thus an array element was sorted using selection sort.

Ex. No. 13 **Merge Sort**
Date:

Aim

To sort an array of N numbers using Merge sort.

Algorithm

1. Start
2. Read number of array elements *n*
3. Read array elements A_i
4. Divide the array into sub-arrays with a set of elements
5. Recursively sort the sub-arrays
6. Merge the sorted sub-arrays onto a single sorted array.
7. Stop

Program

```
/* Merge sort */

#include<stdio.h>
 #include<conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int);
int size;
void main()
{
      int i, arr[30];
      printf("Enter total no. of elements : ");
      scanf("%d", &size);
      printf("Enter array elements : ");
       for(i=0; i<size; i++)
            scanf("%d", &arr[i]);
      part(arr, 0,size-1);
      printf("\n Merge sorted list : ");
      for(i=0; i<size; i++)
            printf("%d",arr[i]);
            getch();
}
```

```
void part(int arr[], int min, int max)
{
      int mid;
      if(min < max)
      {
            mid = (min + max) / 2;
            part(arr, min, mid);
            part(arr, mid+1, max);
            merge(arr, min, mid,max);
      }
      if (max-min ==(size/2)-1)
      {
            printf("\n Half sorted list : ");
             for(i=min; i<=max; i++)
                  printf("%d ", arr[i]);
      }}

void merge(int arr[],int min,int mid,int max)
{
      int tmp[30];
       int i, j, k, m;
       j = min;
      m = mid + 1;
for(i=min; j<=mid && m<=max; i++)
      {
            if(arr[j] <= arr[m])
            {
                  tmp[i] =arr[j]; j++;
            }
            else
            {
                  tmp[i] =arr[m]; m++;
            }
      }
      if(j > mid)
      {
            for(k=m; k<=max; k++)
            {
                  tmp[i] =arr[k]; i++;
            }
      }
      else
      {
            for(k=j; k<=mid; k++)
            {
                  tmp[i] =arr[k]; i++;
```

```
		}
	}
	for(k=min; k<=max;k++)
	 arr[k] = tmp[k];
}
```

Output

Enter total no. of elements : 8
Enter array elements : 24 13 26 1 2 27 3815

  Half sorted list : 1 13 2426
  Half sorted list : 2 15 2738
  Merge sorted list : 1 2 13 15 24 26 2738

Result

Thus an array element was sorted using merge sort.

Ex. No. 14 **Open Addressing Hashing Technique**
Date:

Aim

To implement hash table using a C program.

Algorithm

1. Create a structure, data (hash table item) with key and value as data.
2. Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).
3. A menu is displayed on the screen.
4. User must choose one option from four choices given in the menu
5. Perform all the operations
6. Stop

Program

/* Open hashing */

```
 #include <stdio.h>
 #include<stdlib.h>
#define MAX 10
void main()
{
    int a[MAX], num, key,i;
    char ans;
    int create(int);
    void linearprobing(int[], int, int);
    void display(int[]);
    printf("\nCollision handling by linear probing\n\n");
    for(i=0; i<MAX; i++)
        a[i] = -1;
    do
    {
        printf("\n Enternumber:");
         scanf("%d", &num);
        key = create(num);
        linearprobing(a, key,num);
        printf("\nwish tocontinue?(y/n):");
        ans = getch();
    }
    while( ans =='y');
    display(a);
```

```
}

int create(int num)
{
    int key;
    key = num % 10;
    return key;
}
void linearprobing(int a[MAX], int key, intnum)
{
    int flag, i, count =0; void
    display(int a[]); flag = 0;
    if(a[key] == -1)
     a[key] = num;
    else
    {
        i=0;
 while(i < MAX)
        {
            if(a[i] != -1)
            count++;
            i++;
        }
        if(count == MAX)
        {
            printf("hash table isfull");
            display(a);
            getch();
            exit(1);
        }
        for(i=key+1; i<MAX;i++)
        if(a[i] == -1)
            {
                a[i] = num;
                flag = 1;
                break;
            }
        for(i=0; i<key && flag==0; i++ )
        if(a[i] == -1)
        {
            a[i] = num; flag = 1;
            break;
        }
    }
}
```

```
void display(int a[MAX])
{
      int i;
      printf("\n Hash tableis:");
      for(i=0; i<MAX; i++)
            printf("\n %d\t\t%d",i,a[i]);
}
```

Output

```
Collision handling by linear probing
Enter number:1   wish to continue?(y/n):
  Enter number:26 wish to continue?(y/n):
  Enter number:62 wish to continue?(y/n):
  Enter number:93 wish to continue?(y/n):
Enter number:84 wish to continue?(y/n):
  Enter number:15 wish to continue?(y/n):
    Enter number:76 wish to continue?(y/n):
  Enter number:98 wish to continue?(y/n):
  Enter number:26 wish to continue?(y/n):
  Enter number:199 wish to continue?(y/n):
  Enter number:1234 wish to continue?(y/n):
  Enternumber:5678 hash
table isfull

  Hash table is:
  0                1234
  1                1

  2                62
  3                93
  4                84
  5                15
  6                26
  7                76
  8                98
  9                199
```

Result

Thus hashing has been performed successfully.