DATA STRUCTURES USING C++ LAB

MASTER OF COMPUTER APPLICATIONS (MCA)

SEMESTER-I, PAPER-VI

LESSON WRITERS

Dr. Kampa Lavanya

Assistant Professor
Department of CS&E
University College of Sciences
Acharya Nagarjuna University

EDITOR

Dr. Neelima Guntupalli

Assistant Professor
Department of CS&E
University College of Sciences
Acharya Nagarjuna University

DIRECTOR, I/c.

Prof. V. Venkateswarlu

M.A., M.P.S., M.S.W., M.Phil., Ph.D.

Professor
Centre for Distance Education
Acharya Nagarjuna University
Nagarjuna Nagar 522 510

Ph: 0863-2346222, 2346208
0863- 2346259 (Study Material)
Website www.anucde.info
E-mail: anucdedirector@gmail.com

MCA: DATA STRUCTURES USING C++ LAB

First Edition : 2025

No. of Copies :

This book is exclusively prepared for the use of students of Master of Computer Applications (MCA), Centre for Distance Education, Acharya Nagarjuna University and this book is meant for limited circulation only.

Published by:

Prof. V. VENKATESWARLU
Director, I/c
Centre for Distance Education,
Acharya Nagarjuna University

Printed at:

FOREWORD

Since its establishment in 1976, Acharya Nagarjuna University has been forging ahead in the path of progress and dynamism, offering a variety of courses and research contributions. I am extremely happy that by gaining 'A+' grade from the NAAC in the year 2024, Acharya Nagarjuna University is offering educational opportunities at the UG, PG levels apart from research degrees to students from over 221 affiliated colleges spread over the two districts of Guntur and Prakasam.

The University has also started the Centre for Distance Education in 2003-04 with the aim of taking higher education to the door step of all the sectors of the society. The centre will be a great help to those who cannot join in colleges, those who cannot afford the exorbitant fees as regular students, and even to housewives desirous of pursuing higher studies. Acharya Nagarjuna University has started offering B.Sc., B.A., B.B.A., and B.Com courses at the Degree level and M.A., M.Com., M.Sc., M.B.A., and L.L.M., courses at the PG level from the academic year 2003-2004 onwards.

To facilitate easier understanding by students studying through the distance mode, these self-instruction materials have been prepared by eminent and experienced teachers. The lessons have been drafted with great care and expertise in the stipulated time by these teachers. Constructive ideas and scholarly suggestions are welcome from students and teachers involved respectively. Such ideas will be incorporated for the greater efficacy of this distance mode of education. For clarification of doubts and feedback, weekly classes and contact classes will be arranged at the UG and PG levels respectively.

It is my aim that students getting higher education through the Centre for Distance Education should improve their qualification, have better employment opportunities and in turn be part of country's progress. It is my fond desire that in the years to come, the Centre for Distance Education will go from strength to strength in the form of new courses and by catering to larger number of people. My congratulations to all the Directors, Academic Coordinators, Editors and Lesson-writers of the Centre who have helped in these endeavors.

Prof. K. Gangadhara Rao
M.Tech., Ph.D.
Vice-Chancellor I/c
Acharya Nagarjuna University

106MC24: DATA STRUCTURES LAB

1) Write a program for implementing the operations on complex numbers using classes.
2) Program for finding the area of circle, rectangle and room using function overloading.
3) Program for finding the volume of box using constructor overloading.
4) Program for Sorting 'n' elements Using bubble sort technique.
5) Sort given elements using Selection Sort.
6) Sort given elements using Insertion Sort.
7) Sort given elements using Merge Sort.
8) Sort given elements using Quick Sort.
9) Implement the following operations on single linked list. (i) Creation (ii) Insertion (iii) Deletion (iv) Display
10) Implement the following operations on double linked list. (i) Creation (ii) Insertion (iii) Deletion (iv) Display
11) Implement the following operations on circular linked list. (i) Creation (ii) Insertion (iii) Deletion (iv) Display
12) Program for splitting given linked list.
13) Program for traversing the given linked list in reverse order.
14) Merge two given linked lists.
15) Implement Stack Operations Using Arrays.
16) Implement Stack Operations Using Linked List.
17) Implement Queue Operations Using Arrays.
18) Implement Queue Operations Using Linked List.
19) Implement Operations on Circular Queue.
20) Construct and implement operations on Priority Queue.
21) Implement Operations on double ended Queue.
22) Converting infix expression to postfix expression by using stack.
23) Write program to evaluate post fix expression.
24) Add two polynomials using Linked List.
25) Multiply Two polynomials using Linked List.
26) Construct BST and implement traversing techniques recursively.
27) Implement preorder traversal on BST non recursively.
28) Implement inorder traversal on BST non recursively.
29) Implement postorder traversal on BST non recursively.
30) Implement binary search techniques recursively.

DATA STRUCTURES USING C++ LAB

Code: 106MC24

OBJECTIVES:

The objective of this lab is to master the DATA STRUCTURES concepts with C++ programming and to learn how to work with real time applications using DATA STRUCTURES USING C++. These programs are widely used in most real-time scenarios. After the end of lab, students will be able know the complete practical exposure on DATA STRUCTURES USING C++.

STRUCTURE:

1) Write a program for implementing the operations on complex numbers using classes.
2) Program for finding the area of circle, rectangle and room using function overloading.
3) Program for finding the volume of box using constructor overloading.
4) Program for Sorting ‘n’ elements Using bubble sort technique.
5) Sort given elements using Selection Sort.
6) Sort given elements using Insertion Sort.
7) Sort given elements using Merge Sort.
8) Sort given elements using Quick Sort.
9) Implement the following operations on single linked list.
   (i) Creation (ii) Insertion (iii) Deletion (iv) Display
10) Implement the following operations on double linked list.
   (i) Creation (ii) Insertion (iii) Deletion (iv) Display
11) Implement the following operations on circular linked list.
   (i) Creation (ii) Insertion (iii) Deletion (iv) Display
12) Program for splitting given linked list.
13) Program for traversing the given linked list in reverse order.

14) Merge two given linked lists.

15) Implement Stack Operations Using Arrays.

16) Implement Stack Operations Using Linked List.

17) Implement Queue Operations Using Arrays.

18) Implement Queue Operations Using Linked List.

19) Implement Operations on Circular Queue.

20) Construct and implement operations on Priority Queue.

21) Implement Operations on double ended Queue.

22) Converting infix expression to postfix expression by using stack.

23) Write program to evaluate post fix expression.

24) Add two polynomials using Linked List.

25) Multiply Two polynomials using Linked List.

26) Construct BST and implement traversing techniques recursively.

27) Implement preorder traversal on BST non recursively.

28) Implement inorder traversal on BST non recursively.

29) Implement postorder traversal on BST non recursively.

30) Implement binary search techniques recursively.

1) Write a program for implementing the operations on complex numbers using classes.

```
#include<iostream.h>
#include<conio.h>
class complex
{
   public:
   int real, imaginary;
   void read()
   {
cout<< "Enter the real value\n";
cin>>real;
cout<< "Enter the imaginary value\n";
cin>>imaginary;
   }
   void add(complex c2)
   {
cout<< "The result is\n";
cout<< real + c2.real<< "+" << imaginary + c2.imaginary << "i";
   }
};
int main()
{
   complex c1, c2;
   c1.read();
   c2.read();
   c1.add(c2);
getch();
}
```

Output

```
Enter the real value
5
Enter the imaginary value
4
Enter the real value
3
Enter the imaginary value
2
The result is
8+6i
```

2) Program for finding the area of circle, rectangle and room using function overloading.

```
#include<iostream.h>
#include<conio.h>
class Area
{
   public:
   float area;
   float calcArea(float radius)
   {
      area = 3.14 * radius * radius;
      return area;
   }
   float calcArea(float length, float breadth)
   {
      area = length * breadth;
      return area;
   }
   float calcArea(float length, float breadth, float height)
   {
      area = length * breadth * height;
      return area;
   }
};
int main()
{
   Area a;
   float r, l, b, h;
cout<< "Enter radius of circle: ";
cin>>r;
```

```
cout<< "Area of circle: " <<a.calcArea(r) <<endl;

cout<< "Enter length and breadth of rectangle: ";
cin>> l >>b;
cout<< "Area of rectangle: " <<a.calcArea(l, b) <<endl;

cout<< "Enter length, breadth and height of room: ";
cin>> l >> b >>h;
cout<< "Volume of room: " <<a.calcArea(l, b, h) <<endl;
getch();
}
```

Output

```
Enter radius of circle: 5
Area of circle: 78.5

Enter length and breadth of rectangle: 4 6
Area of rectangle: 24

Enter length, breadth and height of room: 4 6 3
Volume of room: 72
```

3) Program for finding the volume of box using constructor overloading

```
#include<iostream.h>
#include<conio.h>
class Box
{
   public:
   int length, breadth, height;
Box() { length = 0; breadth = 0; height = 0; }
Box(int l, int b, int h) { length = l; breadth = b; height = h; }
   void volume()
   {
cout<< "Volume of the box is: " << length * breadth * height <<endl;
   }
};
int main()
{
   Box b1;
   Box b2(10, 5, 2);
   b1.volume();
   b2.volume();
getch();
}
```

Output

Volume of the box is: 0

Volume of the box is: 100

4) Program for Sorting ‘n’ elements Using bubble sort technique

```
#include<iostream.h>
#include<conio.h>
void bubbleSort(int arr[], int n)
{
for(int i=0; i<n-1; i++)
   {
for(int j=0; j<n-i-1; j++)
      {
         if(arr[j] >arr[j+1])
         {
            int temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
         }
      }
   }
}
int main()
{
   int arr[] = {64, 34, 25, 12, 22, 11, 90};
   int n = sizeof(arr)/sizeof(arr[0]);
bubbleSort(arr, n);
cout<< "Sorted array: ";
for(int i=0; i< n; i++)
cout<<arr[i] << " ";
getch();
}
```

Output

Sorted array: 11 12 22 25 34 64 90

5) Sort given elements using Selection Sort

```
#include<iostream.h>
#include<conio.h>
void selectionSort(int arr[], int n)
{
for(int i=0; i<n-1; i++)
   {
      int minIdx = i;
for(int j=i+1; j<n; j++)
      {
         if(arr[j] <arr[minIdx])
minIdx = j;
      }
      int temp = arr[i];
arr[i] = arr[minIdx];
arr[minIdx] = temp;
   }
}
int main()
{
   int arr[] = {64, 25, 12, 22, 11};
   int n = sizeof(arr)/sizeof(arr[0]);
selectionSort(arr, n);
cout<< "Sorted array: ";
for(int i=0; i< n; i++)
cout<<arr[i] << " ";
getch();
}
```

Output

Sorted array: 11 12 22 25 64

6) Sort given elements using Insertion Sort

```
#include<iostream.h>
#include<conio.h>
void insertionSort(int arr[], int n)
{
for(int i=1; i<n; i++)
   {
      int key = arr[i];
      int j = i-1;
while(j >= 0 &&arr[j] > key)
      {
arr[j+1] = arr[j];
         j = j-1;
      }
arr[j+1] = key;
   }
}
int main()
{
   int arr[] = {12, 11, 13, 5, 6};
   int n = sizeof(arr)/sizeof(arr[0]);
insertionSort(arr, n);
cout<< "Sorted array: ";
for(int i=0; i< n; i++)
cout<<arr[i] << " ";
getch();
}
```

Output

Sorted array: 5 6 11 12 13

7) Sort given elements using Merge Sort

```
#include<iostream.h>
#include<conio.h>
void merge(int arr[], int l, int m, int r)
{
   int n1 = m - l + 1;
   int n2 = r - m;
   int L[n1], R[n2];
for(int i=0; i<n1; i++)
      L[i] = arr[l + i];
for(int j=0; j<n2; j++)
      R[j] = arr[m + 1 + j];
   int i = 0, j = 0, k = l;
while(i<n1 && j < n2)
   {
      if(L[i] <= R[j])
arr[k++] = L[i++];
      else
arr[k++] = R[j++];
   }
while(i< n1)
arr[k++] = L[i++];
while(j < n2)
arr[k++] = R[j++];
}
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);
    }
}
int main()
{
    int arr[] = {38, 27, 43, 3, 9, 82, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
mergeSort(arr, 0, n-1);
cout<< "Sorted array: ";
for(int i=0; i< n; i++)
cout<<arr[i] << " ";
getch();
}
```

Output

Sorted array: 3 9 10 27 38 43 82

8) Sort given elements using Quick Sort

```
##include<iostream.h>
#include<conio.h>
int partition(int arr[], int low, int high)
{
   int pivot = arr[high];
   int i = low - 1;
for(int j=low; j<high; j++)
   {
      if(arr[j] <= pivot)
      {
i++;
         int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
      }
   }
   int temp = arr[i+1];
arr[i+1] = arr[high];
arr[high] = temp;
   return i+1;
}
void quickSort(int arr[], int low, int high)
{
if(low < high)
   {
      int pi = partition(arr, low, high);
quickSort(arr, low, pi-1);
quickSort(arr, pi+1, high);
   }
```

```
}
int main()
{
   int arr[] = {10, 7, 8, 9, 1, 5};
   int n = sizeof(arr)/sizeof(arr[0]);
quickSort(arr, 0, n-1);
cout<< "Sorted array: ";
for(int i=0; i< n; i++)
cout<<arr[i] << " ";
getch();
}
```

Output

Sorted array: 1 5 7 8 9 10

9) Implement the following operations on single linked list.

(i) Creation (ii) Insertion (iii) Deletion (iv) Display

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
};
class LinkedList
{
   public:
   Node* head;
LinkedList() { head = NULL; }

   void create()
   {
      Node* newNode = new Node;
cout<< "Enter data to insert: ";
cin>>newNode->data;
newNode->next = head;
      head = newNode;
cout<< "Node created with data: " <<newNode->data << "\n";
   }
   void insert()
   {
      Node* newNode = new Node;
cout<< "Enter data to insert at the end: ";
cin>>newNode->data;
newNode->next = NULL;
            if (head == NULL)
      {
```

```
            head = newNode;
        }
        else
        {
            Node* temp = head;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newNode;
        }
cout<< "Node inserted with data: " <<newNode->data << "\n";
    }
    void deleting()
    {
        if (head == NULL)
        {
cout<< "List is empty, nothing to delete.\n";
return;
        }
        int value;
cout<< "Enter value to delete: ";
cin>>value;
         Node* temp = head;
        Node* prev = NULL;

        while (temp != NULL && temp->data != value)
        {
prev = temp;
            temp = temp->next;
        }
        if (temp == NULL)
        {
cout<< "Value not found in the list.\n";
```

```
        }
        else
        {
            if (prev == NULL)  // Deleting the first node
            {
                head = temp->next;
            }
            else
            {
prev->next = temp->next;
            }
cout<< "Deleted node with data: " << temp->data << "\n";
            delete temp;
        }
    }
    void display()
    {
        if (head == NULL)
        {
cout<< "List is empty.\n";
return;
        }

        Node* temp = head;
cout<< "Linked List: ";
        while (temp != NULL)
        {
cout<< temp->data << " ";
            temp = temp->next;
        }
cout<< "\n";
    }
```

```
};
int main()
{
clrscr();
   LinkedList list;
   int ch, nodes;
do
   {
cout<< "Enter your choice\n";
cout<< "1-Create\n";
cout<< "2-Insert\n";
cout<< "3-Delete\n";
cout<< "4-Display\n";
cout<< "5-Exit\n";
cin>>ch;
      switch(ch)
      {
         case 1:
cout<< "Enter the size of Linked List: ";
cin>>nodes;
            for (int i = 0; i< nodes; i++)
list.create();
break;
         case 2:
list.insert();
break;
         case 3:
list.deleting();
break;
         case 4:
list.display();
break;
```

```
            case 5:
cout<< "Exiting...\n";
break;
            default:
cout<< "Invalid choice, please try again.\n";
        }
    } while (ch != 5);
getch();
    return 0;
}
```

Output

```
Enter your choice
1-Create
2-Insert
3-Delete
4-Display
5-Exit
1
Enter the size of Linked List: 3
Enter data to insert: 10
Node created with data: 10
Enter data to insert: 20
Node created with data: 20
Enter data to insert: 30
Node created with data: 30

Enter your choice
2-Insert
3-Delete
4-Display
5-Exit
```

2

Enter data to insert at the end: 40

Node inserted with data: 40

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

3

Enter value to delete: 20

Deleted node with data: 20

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

4

Linked List: 30 10 40

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

5

Exiting...

10) Implement the following operations on double linked list.

(i) Creation (ii) Insertion (iii) Deletion (iv) Display

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
   Node* prev;
};

class DoubleLinkedList
{
   public:
   Node* head;
DoubleLinkedList() { head = NULL; }

   void create()
   {
      Node* newNode = new Node;
cout<< "Enter data to insert: ";
cin>>newNode->data;
newNode->next = head;
newNode->prev = NULL;
      if (head != NULL)
         head->prev = newNode;
      head = newNode;
cout<< "Node created with data: " <<newNode->data << "\n";
   }
   void insert()
   {
      Node* newNode = new Node;
```

```
cout<< "Enter data to insert at the end: ";
cin>>newNode->data;
newNode->next = NULL;

    if (head == NULL)
    {
newNode->prev = NULL;
        head = newNode;
    }
    else
    {
        Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
newNode->prev = temp;
    }
cout<< "Node inserted with data: " <<newNode->data << "\n";
  }
  void deleting()
  {
    if (head == NULL)
    {
cout<< "List is empty, nothing to delete.\n";
return;
    }
    int value;
cout<< "Enter value to delete: ";
cin>>value;
    Node* temp = head;
    while (temp != NULL && temp->data != value)
        temp = temp->next;
```

```
        if (temp == NULL)
        {
cout<< "Value not found in the list.\n";
        }
        else
        {
            if (temp->prev != NULL)
                temp->prev->next = temp->next;
            else
                head = temp->next;

            if (temp->next != NULL)
                temp->next->prev = temp->prev;

cout<< "Deleted node with data: " << temp->data << "\n";
            delete temp;
        }
    }
    void display()
    {
        if (head == NULL)
        {
cout<< "List is empty.\n";
return;
        }
        Node* temp = head;
cout<< "Double Linked List: ";
        while (temp != NULL)
        {
cout<< temp->data << " ";
            temp = temp->next;
        }
```

```
cout<< "\n";
   }
};
int main()
{
clrscr();
DoubleLinkedListlist;
   int ch, nodes;

   do
   {
cout<< "Enter your choice\n";
cout<< "1-Create\n";
cout<< "2-Insert\n";
cout<< "3-Delete\n";
cout<< "4-Display\n";
cout<< "5-Exit\n";
cin>>ch;
      switch(ch)
      {
         case 1:
cout<< "Enter the size of Linked List: ";
cin>>nodes;
            for (int i = 0; i< nodes; i++)
list.create();
break;
         case 2:
list.insert();
break;
         case 3:
list.deleting();
break;
```

```
            case 4:
list.display();
break;
            case 5:
cout<< "Exiting...\n";
break;
            default:
cout<< "Invalid choice, please try again.\n";
        }
    } while (ch != 5);
getch();
    return 0;}
```

Output

```
Enter your choice
1-Create
2-Insert
3-Delete
4-Display
5-Exit
1
Enter the size of Linked List: 3
Enter data to insert: 10
Node created with data: 10
Enter data to insert: 20
Node created with data: 20
Enter data to insert: 30
Node created with data: 30
Enter your choice
2-Insert
3-Delete
4-Display
5-Exit
```

2

Enter data to insert at the end: 40

Node inserted with data: 40

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

3

Enter value to delete: 20

Deleted node with data: 20

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

4

Double Linked List: 30 10 40

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

5

Exiting...

11) Implement the following operations on circular linked list.

(i) Creation (ii) Insertion (iii) Deletion (iv) Display

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
};
class CircularLinkedList
{
   public:
   Node* head;
CircularLinkedList() { head = NULL; }
   void create()
   {
      Node* newNode = new Node;
cout<< "Enter data to insert: ";
cin>>newNode->data;
      if (head == NULL)
      {
         head = newNode;
newNode->next = head;
      }
      else
      {
         Node* temp = head;
         while (temp->next != head)
            temp = temp->next;
         temp->next = newNode;
```

```
newNode->next = head;
        }
cout<< "Node created with data: " <<newNode->data << "\n";
    }
    void insert()
    {
        Node* newNode = new Node;
cout<< "Enter data to insert at the end: ";
cin>>newNode->data;
newNode->next = head;
        if (head == NULL)
        {
            head = newNode;
newNode->next = head;
        }
        else
        {
            Node* temp = head;
            while (temp->next != head)
                temp = temp->next;
            temp->next = newNode;
newNode->next = head;
        }
cout<< "Node inserted with data: " <<newNode->data << "\n";
    }

    void deleting()
    {
        if (head == NULL)
        {
```

```
cout<< "List is empty, nothing to delete.\n";
return;
        }
        int value;
cout<< "Enter value to delete: ";
cin>>value;
        Node* temp = head;
        Node* prev = NULL;
        while (temp->next != head && temp->data != value)
        {
prev = temp;
           temp = temp->next;
        }
        if (temp->data != value)
        {
cout<< "Value not found in the list.\n";
        }
        else
        {
           if (prev == NULL)  // Deleting the only node
           {
              head = NULL;
           }
           else
           {
prev->next = temp->next;
              if (temp == head)
                 head = temp->next;
           }
cout<< "Deleted node with data: " << temp->data << "\n";
```

```
            delete temp;
        }
    }
    void display()
    {
        if (head == NULL)
        {
cout<< "List is empty.\n";
return;
        }
        Node* temp = head;
cout<< "Circular Linked List: ";
        do
        {
cout<< temp->data << " ";
            temp = temp->next;
        } while (temp != head);
cout<< "\n";
    }
};
int main()
{
clrscr();
CircularLinkedListlist;
    int ch, nodes;
    do
    {
cout<< "Enter your choice\n";
cout<< "1-Create\n";
cout<< "2-Insert\n";
```

```
cout<< "3-Delete\n";
cout<< "4-Display\n";
cout<< "5-Exit\n";
cin>>ch;
      switch(ch)
      {
         case 1:
cout<< "Enter the size of Linked List: ";
cin>>nodes;
            for (int i = 0; i< nodes; i++)
list.create();
break;
         case 2:
list.insert();
break;
         case 3:
list.deleting();
break;
         case 4:
list.display();
break;
         case 5:
cout<< "Exiting...\n";
break;
         default:
cout<< "Invalid choice, please try again.\n";
      }
   } while (ch != 5);
getch();
   return 0;
```

```
}
```

Output

```
Enter your choice
1-Create
2-Insert
3-Delete
4-Display
5-Exit
1
Enter the size of Linked List: 3
Enter data to insert: 10
Node created with data: 10
Enter data to insert: 20
Node created with data: 20
Enter data to insert: 30
Node created with data: 30

Enter your choice
2-Insert
3-Delete
4-Display
5-Exit
2
Enter data to insert at the end: 40
Node inserted with data: 40

Enter your choice
2-Insert
3-Delete
4-Display
```

5-Exit

3

Enter value to delete: 20

Deleted node with data: 20

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

4

Circular Linked List: 30 10 40

Enter your choice

2-Insert

3-Delete

4-Display

5-Exit

5

Exiting...

12) Program for splitting given linked list.

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
};

class LinkedList
{
   public:
   Node* head;
LinkedList() { head = NULL; }

   void create()
   {
      Node* newNode = new Node;
cout<< "Enter data to insert: ";
cin>>newNode->data;
newNode->next = head;
      head = newNode;
cout<< "Node created with data: " <<newNode->data << "\n";
   }

   void split()
   {
      if (head == NULL)
      {
cout<< "List is empty.\n";
return;
      }
```

```
        Node* slow = head;
        Node* fast = head;
        Node* list2 = NULL;

        // Move fast by 2 and slow by 1 to find the middle
        while (fast != NULL && fast->next != NULL)
        {
            fast = fast->next->next;
            slow = slow->next;
        }

        list2 = slow->next;
        slow->next = NULL;

        // Display the split lists
    cout<< "First List: ";
        Node* temp = head;
        while (temp != NULL)
        {
    cout<< temp->data << " ";
            temp = temp->next;
        }

    cout<< "\nSecond List: ";
        temp = list2;
        while (temp != NULL)
        {
    cout<< temp->data << " ";
            temp = temp->next;
        }
    cout<< "\n";
```

```
    }
    void display()
    {
        if (head == NULL)
        {
cout<< "List is empty.\n";
return;
        }
        Node* temp = head;
cout<< "Linked List: ";
        while (temp != NULL)
        {
cout<< temp->data << " ";
            temp = temp->next;
        }
cout<< "\n";
    }
};
int main()
{
clrscr();
    LinkedList list;
    int ch, nodes;
    do
    {
cout<< "Enter your choice\n";
cout<< "1-Create\n";
cout<< "2-Split\n";
cout<< "3-Display\n";
cout<< "4-Exit\n";
cin>>ch;
        switch(ch)
```

```
        {
            case 1:
cout<< "Enter the size of Linked List: ";
cin>>nodes;
                for (int i = 0; i< nodes; i++)
list.create();
break;

            case 2:
list.split();
break;

            case 3:
list.display();
break;

            case 4:
cout<< "Exiting...\n";
break;
            default:
cout<< "Invalid choice, please try again.\n";
        }
    } while (ch != 4);
getch();
    return 0;
}
```

Output

Enter your choice

1-Create

2-Split

3-Display

4-Exit

```
1
Enter the size of Linked List: 5
Enter data to insert: 10
Node created with data: 10
Enter data to insert: 20
Node created with data: 20
Enter data to insert: 30
Node created with data: 30
Enter data to insert: 40
Node created with data: 40
Enter data to insert: 50
Node created with data: 50

Enter your choice
2-Split
3-Display
4-Exit
2
First List: 10 20 30
Second List: 40 50

Enter your choice
2-Split
3-Display
4-Exit
4
Exiting...
```

13) Program for traversing the given linked list in reverse order.

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
};
class LinkedList
{
   public:
   Node* head;
LinkedList() { head = NULL; }
   void create()
   {
      Node* newNode = new Node;
cout<< "Enter data to insert: ";
cin>>newNode->data;
newNode->next = head;
      head = newNode;
cout<< "Node created with data: " <<newNode->data << "\n";
   }
   void reverseTraversal(Node* temp)
   {
      if (temp == NULL)
return;
reverseTraversal(temp->next);
cout<< temp->data << " ";
   }
voiddisplayReverse()
```

```
    {
cout<< "Reverse Traversal: ";
reverseTraversal(head);
cout<< "\n";
    }
    void display()
    {
        if (head == NULL)
        {
cout<< "List is empty.\n";
return;
        }
        Node* temp = head;
cout<< "Linked List: ";
        while (temp != NULL)
        {
cout<< temp->data << " ";
            temp = temp->next;
        }
cout<< "\n";
    }
};
int main()
{
clrscr();
    LinkedList list;
    int ch, nodes;
    do
    {
cout<< "Enter your choice\n";
```

```
cout<< "1-Create\n";
cout<< "2-Display Reverse\n";
cout<< "3-Display\n";
cout<< "4-Exit\n";
cin>>ch;
      switch(ch)
      {
         case 1:
cout<< "Enter the size of Linked List: ";
cin>>nodes;
            for (int i = 0; i< nodes; i++)
list.create();
break;
         case 2:
list.displayReverse();
break;
         case 3:
list.display();
break;
         case 4:
cout<< "Exiting...\n";
break;
         default:
cout<< "Invalid choice, please try again.\n";
      }
   } while (ch != 4);
getch();
   return 0;
}
```

Output

Enter your choice

1-Create

2-Display Reverse

3-Display

4-Exit

1

Enter the size of Linked List: 3

Enter data to insert: 10

Node created with data: 10

Enter data to insert: 20

Node created with data: 20

Enter data to insert: 30

Node created with data: 30

Enter your choice

2-Display Reverse

3-Display

4-Exit

2

Reverse Traversal: 10 20 30

Enter your choice

2-Display Reverse

3-Display

4-Exit

4

Exiting...

14) Merge two given linked lists.

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
};
class LinkedList
{
   public:
   Node* head;
LinkedList() { head = NULL; }
   void create()
   {
      Node* newNode = new Node;
cout<< "Enter data to insert: ";
cin>>newNode->data;
newNode->next = head;
      head = newNode;
cout<< "Node created with data: " <<newNode->data << "\n";
   }
   void merge(LinkedList& list2)
   {
      if (head == NULL)
      {
         head = list2.head;
return;
      }
      Node* temp = head;
```

```
        while (temp->next != NULL)
            temp = temp->next;
            temp->next = list2.head;
  list2.head = NULL;
    }
    void display()
    {
        if (head == NULL)
        {
cout<< "List is empty.\n";
return;
        }
        Node* temp = head;
cout<< "Linked List: ";
        while (temp != NULL)
        {
cout<< temp->data << " ";
            temp = temp->next;
        }
cout<< "\n";
    }
};
int main()
{
clrscr();
    LinkedList list1, list2;
    int ch, nodes;
    do
    {
cout<< "Enter your choice\n";
```

```
cout<< "1-Create List 1\n";
cout<< "2-Create List 2\n";
cout<< "3-Merge Lists\n";
cout<< "4-Display\n";
cout<< "5-Exit\n";
cin>>ch;
      switch(ch)
      {
         case 1:
cout<< "Enter the size of Linked List 1: ";
cin>>nodes;
            for (int i = 0; i< nodes; i++)
               list1.create();
break;
         case 2:
cout<< "Enter the size of Linked List 2: ";
cin>>nodes;
            for (int i = 0; i< nodes; i++)
               list2.create();
break;
         case 3:
            list1.merge(list2);
break;

         case 4:
            list1.display();
break;
         case 5:
cout<< "Exiting...\n";
break;
```

```
            default:
cout<< "Invalid choice, please try again.\n";
        }
    } while (ch != 5);
getch();
    return 0;
}
```

Output

```
Enter your choice
1-Create List 1
2-Create List 2
3-Merge Lists
4-Display
5-Exit
1
Enter the size of Linked List 1: 2
Enter data to insert: 10
Node created with data: 10
Enter data to insert: 20
Node created with data: 20
Enter your choice
2-Create List 2
3-Merge Lists
4-Display
5-Exit
2
Enter the size of Linked List 2: 2
Enter data to insert: 30
Node created with data: 30
Enter data to insert: 40
```

Node created with data: 40

Enter your choice

3-Merge Lists

4-Display

5-Exit

3

Enter your choice

4-Display

5-Exit

4

Linked List: 10 20 30 40

Enter your choice

5-Exit

5

Exiting...

15) Implement Stack Operations Using Arrays.

```
#include<iostream.h>
#include<conio.h>
#define MAX 5
class Stack
{
   public:
   int arr[MAX];
   int top;
Stack() { top = -1; }
   void push(int val)
   {
      if (top == MAX - 1)
cout<< "Stack Overflow\n";
      else
arr[++top] = val;
   }
   void pop()
   {
      if (top == -1)
cout<< "Stack Underflow\n";
      else
cout<< "Popped: " <<arr[top--] << "\n";
   }
   void display()
   {
      if (top == -1)
cout<< "Stack is empty.\n";
      else
      {
```

```
cout<< "Stack: ";
        for (int i = 0; i<= top; i++)
cout<<arr[i] << " ";
cout<< "\n";
      }
   }
};

int main()
{
clrscr();
   Stack stack;
   int ch, val;
   do
   {
cout<< "Enter your choice\n";
cout<< "1-Push\n";
cout<< "2-Pop\n";
cout<< "3-Display\n";
cout<< "4-Exit\n";
cin>>ch;
      switch(ch)
      {
         case 1:
cout<< "Enter value to push: ";
cin>>val;
stack.push(val);
break;
         case 2:
stack.pop();
```

```
break;
        case 3:
stack.display();
break;
        case 4:
cout<< "Exiting...\n";
break;
        default:
cout<< "Invalid choice, please try again.\n";
      }
   } while (ch != 4);
getch();
   return 0;
}
```

Output

```
Enter your choice
1-Push
2-Pop
3-Display
4-Exit
1
Enter value to push: 10
Enter your choice
1-Push
2-Pop
3-Display
4-Exit
1
Enter value to push: 20
Enter your choice
```

3-Display

4-Exit

3

Stack: 10 20

Enter your choice

2-Pop

3-Display

4-Exit

2

Popped: 20

Enter your choice

3-Display

4-Exit

3

Stack: 10

Enter your choice

4-Exit

4

Exiting...

16) Implement Stack Operations Using Linked List

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
};
class Stack
{
   public:
   Node* top;
Stack() { top = NULL; }
   void push(int val)
   {
      Node* newNode = new Node;
newNode->data = val;
newNode->next = top;
      top = newNode;
   }
   void pop()
   {
      if (top == NULL)
cout<< "Stack Underflow\n";
      else
      {
cout<< "Popped: " << top->data << "\n";
         Node* temp = top;
         top = top->next;
         delete temp;
```

```
        }
    }
    void display()
    {
        if (top == NULL)
cout<< "Stack is empty.\n";
        else
        {
            Node* temp = top;
cout<< "Stack: ";
            while (temp != NULL)
            {
cout<< temp->data << " ";
                temp = temp->next;
            }
cout<< "\n";
        }
    }
};
int main()
{
clrscr();
    Stack stack;
    int ch, val;
    do
    {
cout<< "Enter your choice\n";
cout<< "1-Push\n";
cout<< "2-Pop\n";
cout<< "3-Display\n";
```

```
cout<< "4-Exit\n";
cin>>ch;
        switch(ch)
        {
            case 1:
cout<< "Enter value to push: ";
cin>>val;
stack.push(val);
break;
            case 2:
stack.pop();
break;
            case 3:
stack.display();
break;
            case 4:
cout<< "Exiting...\n";
break;
            default:
cout<< "Invalid choice, please try again.\n";
        }
    } while (ch != 4);
getch();
    return 0;
}
```

Output

```
Enter your choice
1-Push
2-Pop
3-Display
```

```
4-Exit
1
Enter value to push: 10
Enter your choice
1-Push
2-Pop
3-Display
4-Exit
1
Enter value to push: 20
Enter your choice
3-Display
4-Exit
3
Stack: 20 10
Enter your choice
2-Pop
3-Display
4-Exit
2
Popped: 20
Enter your choice
3-Display
4-Exit
3
Stack: 10
Enter your choice
4-Exit
4
Exiting...
```

17) Implement Queue Operations Using Arrays.

```
#include<iostream.h>
#include<conio.h>
#define MAX 5
class Queue
{
   public:
   int arr[MAX];
   int front, rear;
Queue() { front = rear = -1; }
   void enqueue(int val)
   {
      if (rear == MAX - 1)
cout<< "Queue Overflow\n";
      else
      {
         if (front == -1)
            front = 0;
arr[++rear] = val;
      }
   }
   void dequeue()
   {
      if (front == -1)
cout<< "Queue Underflow\n";
      else
      {
cout<< "Dequeued: " <<arr[front++] << "\n";
         if (front > rear)
            front = rear = -1;
```

```
        }
    }
    void display()
    {
        if (front == -1)
cout<< "Queue is empty.\n";
        else
        {
cout<< "Queue: ";
            for (int i = front; i<= rear; i++)
cout<<arr[i] << " ";
cout<< "\n";
        }
    }
};
int main()
{
clrscr();
    Queue q;
    int ch, val;
    do
    {
cout<< "Enter your choice\n";
cout<< "1-Enqueue\n";
cout<< "2-Dequeue\n";
cout<< "3-Display\n";
cout<< "4-Exit\n";
cin>>ch;
        switch(ch)
        {
```

```
        case 1:
cout<< "Enter value to enqueue: ";
cin>>val;
q.enqueue(val);
break;
        case 2:
q.dequeue();
break;
        case 3:
q.display();
break;

        case 4:
cout<< "Exiting...\n";
break;
        default:
cout<< "Invalid choice, please try again.\n";
      }
   } while (ch != 4);
getch();
   return 0;
}
```

Output

```
Enter your choice
1-Enqueue
2-Dequeue
3-Display
4-Exit
1
Enter value to enqueue: 10
```

Enter your choice

1-Enqueue

2-Dequeue

3-Display

4-Exit

1

Enter value to enqueue: 20

Enter your choice

3-Display

4-Exit

3

Queue: 10 20

Enter your choice

2-Dequeue

3-Display

4-Exit

2

Dequeued: 10

Enter your choice

3-Display

4-Exit

3

Queue: 20

Enter your choice

4-Exit

4

Exiting...

18) Implement Queue Operations Using Linked List.

```
#include<iostream.h>
#include<conio.h>
struct Node
{
   int data;
   Node* next;
};
class Queue
{
   public:
   Node *front, *rear;
Queue() { front = rear = NULL; }
   void enqueue(int val)
   {
      Node* newNode = new Node;
newNode->data = val;
newNode->next = NULL;
      if (rear == NULL)
      {
         front = rear = newNode;
return;
      }
      rear->next = newNode;
      rear = newNode;
   }
   void dequeue()
   {
      if (front == NULL)
cout<< "Queue Underflow\n";
      else
      {
cout<< "Dequeued: " << front->data << "\n";
```

```
            Node* temp = front;
            front = front->next;
            if (front == NULL)
               rear = NULL;
            delete temp;
         }
      }
      void display()
      {
         if (front == NULL)
cout<< "Queue is empty.\n";
         else
         {
            Node* temp = front;
cout<< "Queue: ";
            while (temp != NULL)
            {
cout<< temp->data << " ";
               temp = temp->next;
            }
cout<< "\n";
         }
      }
};
int main()
{
clrscr();
   Queue q;
   int ch, val;
   do
   {
cout<< "Enter your choice\n";
cout<< "1-Enqueue\n";
```

```
cout<< "2-Dequeue\n";
cout<< "3-Display\n";
cout<< "4-Exit\n";
cin>>ch;
      switch(ch)
      {
         case 1:
cout<< "Enter value to enqueue: ";
cin>>val;
q.enqueue(val);
break;
         case 2:
q.dequeue();
break;
         case 3:
q.display();
break;
         case 4:
cout<< "Exiting...\n";
break;
         default:
cout<< "Invalid choice, please try again.\n";
      }
   } while (ch != 4);
getch();
   return 0;
}
```

Output

```
Enter your choice
1-Enqueue
2-Dequeue
3-Display
4-Exit
```

1

Enter value to enqueue: 10

Enter your choice

1-Enqueue

2-Dequeue

3-Display

4-Exit

1

Enter value to enqueue: 20

Enter your choice

3-Display

4-Exit

3

Queue: 10 20

Enter your choice

2-Dequeue

3-Display

4-Exit

2

Dequeued: 10

Enter your choice

3-Display

4-Exit

3

Queue: 20

Enter your choice

4-Exit

4

Exiting...

19) Implement Operations on Circular Queue.

```
#include<iostream.h>
#include<conio.h>
#define MAX 5
class CircularQueue
{
   public:
   int arr[MAX];
   int front, rear;
CircularQueue() { front = rear = -1; }
   void enqueue(int val)
   {
if ((rear + 1) % MAX == front)
cout<< "Queue Overflow\n";
       else
       {
          if (front == -1)
             front = 0;
          rear = (rear + 1) % MAX;
arr[rear] = val;
       }
   }
   void dequeue()
   {
       if (front == -1)
cout<< "Queue Underflow\n";
       else
       {
cout<< "Dequeued: " <<arr[front] << "\n";
          if (front == rear)
             front = rear = -1;
          else
             front = (front + 1) % MAX;
       }
   }
```

```
    void display()
    {
        if (front == -1)
cout<< "Queue is empty.\n";
        else
        {
            int i = front;
cout<< "Queue: ";
            while (i != rear)
            {
cout<<arr[i] << " ";
i = (i + 1) % MAX;
            }
cout<<arr[rear] << "\n";
        }
    }
};
int main()
{
clrscr();
CircularQueueq;
    int ch, val;
    do
    {
cout<< "Enter your choice\n";
cout<< "1-Enqueue\n";
cout<< "2-Dequeue\n";
cout<< "3-Display\n";
cout<< "4-Exit\n";
cin>>ch;
        switch(ch)
        {
            case 1:
cout<< "Enter value to enqueue: ";
cin>>val;
```

```
q.enqueue(val);
break;

        case 2:
q.dequeue();
break;
        case 3:
q.display();
break;
        case 4:
cout<< "Exiting...\n";
break;
        default:
cout<< "Invalid choice, please try again.\n";
      }
   } while (ch != 4);
getch();
   return 0;
}
```

Output

```
Enter your choice
1-Enqueue
2-Dequeue
3-Display
4-Exit
1
Enter value to enqueue: 10
Enter your choice
1-Enqueue
2-Dequeue
3-Display
4-Exit
1
Enter value to enqueue: 20
Enter your choice
```

3-Display

4-Exit

3

Queue: 10 20

Enter your choice

2-Dequeue

3-Display

4-Exit

2

Dequeued: 10

Enter your choice

3-Display

4-Exit

3

Queue: 20

Enter your choice

4-Exit

4

Exiting...

20) Construct and Implement Operations on Priority Queue.

```
#include<iostream.h>
#include<conio.h>
#define MAX 5

class PriorityQueue
{
   public:
   int arr[MAX];
   int size;

PriorityQueue() { size = 0; }

   void insert(int val)
   {
      if (size == MAX)
cout<< "Queue Overflow\n";
      else
      {
         int i = size - 1;
         while (i>= 0 &&arr[i] <val)
         {
arr[i + 1] = arr[i];
i--;
         }
arr[i + 1] = val;
         size++;
      }
   }
```

```
    void remove()
    {
        if (size == 0)
cout<< "Queue Underflow\n";
        else
        {
cout<< "Removed: " <<arr[0] << "\n";
            for (int i = 1; i< size; i++)
arr[i - 1] = arr[i];
            size--;
        }
    }
    void display()
    {
        if (size == 0)
cout<< "Queue is empty.\n";
        else
        {
cout<< "Priority Queue: ";
            for (int i = 0; i< size; i++)
cout<<arr[i] << " ";
cout<< "\n";
        }
    }
};
int main()
{
clrscr();
PriorityQueuepq;
    int ch, val;
```

```
   do
   {
cout<< "Enter your choice\n";
cout<< "1-Insert\n";
cout<< "2-Remove\n";
cout<< "3-Display\n";
cout<< "4-Exit\n";
cin>>ch;
       switch(ch)
       {
          case 1:
cout<< "Enter value to insert: ";
cin>>val;
pq.insert(val);
break;

          case 2:
pq.remove();
break;

          case 3:
pq.display();
break;

          case 4:
cout<< "Exiting...\n";
break;

          default:
cout<< "Invalid choice, please try again.\n";
```

```
        }
    } while (ch != 4);

getch();
    return 0;
}
```

Output

```
Enter your choice
1-Insert
2-Remove
3-Display
4-Exit
1
Enter value to insert: 10

Enter your choice
1-Insert
2-Remove
3-Display
4-Exit
1
Enter value to insert: 20

Enter your choice
3-Display
4-Exit
3
Priority Queue: 20 10

Enter your choice
```

2-Remove

3-Display

4-Exit

2

Removed: 20

Enter your choice

3-Display

4-Exit

3

Priority Queue: 10

Enter your choice

4-Exit

4

Exiting...

21) Implement Operations on Double Ended Queue (Deque).

```
#include<iostream.h>
#include<conio.h>
#define MAX 5
class Deque
{
   public:
   int arr[MAX];
   int front, rear;
Deque() { front = rear = -1; }
   void insertFront(int val)
   {
      if (front == 0 && rear == MAX - 1 || front == rear + 1)
cout<< "Deque Overflow\n";
      else
      {
         if (front == -1) { front = rear = 0; }
         else if (front == 0) front = MAX - 1;
         else front--;
arr[front] = val;
      }
   }
   void insertRear(int val)
   {
      if (front == 0 && rear == MAX - 1 || front == rear + 1)
cout<< "Deque Overflow\n";
      else
      {
         if (front == -1) { front = rear = 0; }
         else if (rear == MAX - 1) rear = 0;
         else rear++;
arr[rear] = val;
      }
```

```
    }
    void deleteFront()
    {
        if (front == -1)
cout<< "Deque Underflow\n";
        else
        {
cout<< "Deleted from front: " <<arr[front] << "\n";
            if (front == rear)
                front = rear = -1;
            else if (front == MAX - 1) front = 0;
            else front++;
        }
    }

    void deleteRear()
    {
        if (front == -1)
cout<< "Deque Underflow\n";
        else
        {
cout<< "Deleted from rear: " <<arr[rear] << "\n";
            if (front == rear)
                front = rear = -1;
            else if (rear == 0) rear = MAX - 1;
            else rear--;
        }
    }
    void display()
    {
        if (front == -1)
cout<< "Deque is empty.\n";
        else
```

```
        {
            int i = front;
cout<< "Deque: ";
            while (i != rear)
            {
cout<<arr[i] << " ";
i = (i + 1) % MAX;
            }
cout<<arr[rear] << "\n";
        }
    }
};
int main()
{
clrscr();
    Deque dq;
    int ch, val;
    do
    {
cout<< "Enter your choice\n";
cout<< "1-Insert Front\n";
cout<< "2-Insert Rear\n";
cout<< "3-Delete Front\n";
cout<< "4-Delete Rear\n";
cout<< "5-Display\n";
cout<< "6-Exit\n";
cin>>ch;
        switch(ch)
        {
            case 1:
cout<< "Enter value to insert at front: ";
cin>>val;
dq.insertFront(val);
```

```
break;
        case 2:
cout<< "Enter value to insert at rear: ";
cin>>val;
dq.insertRear(val);
break;
        case 3:
dq.deleteFront();
break;
        case 4:
dq.deleteRear();
break;
        case 5:
dq.display();
break;
        case 6:
cout<< "Exiting...\n";
break;
        default:
cout<< "Invalid choice, please try again.\n";
      }
   } while (ch != 6);
getch();
   return 0;
}
```

Output

```
Enter your choice
1-Insert Front
2-Insert Rear
3-Delete Front
4-Delete Rear
5-Display
6-Exit
```

1

Enter value to insert at front: 10

Enter your choice

2-Insert Rear

3-Delete Front

4-Delete Rear

5-Display

6-Exit

2

Enter value to insert at rear: 20

Enter your choice

5-Display

6-Exit

5

Deque: 10 20

Enter your choice

3-Delete Front

4-Delete Rear

5-Display

6-Exit

3

Deleted from front: 10

Enter your choice

5-Display

6-Exit

5

Deque: 20

Enter your choice

6-Exit

6

Exiting...

22) Converting Infix Expression to Postfix Expression Using Stack.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
class Stack
{
   public:
   char arr[50];
   int top;
Stack() { top = -1; }
   void push(char c) { arr[++top] = c; }
   char pop() { return arr[top--]; }
   bool isEmpty() { return top == -1; }
   char peek() { return arr[top]; }
};
int precedence(char c)
{
   if (c == '+' || c == '-') return 1;
   if (c == '*' || c == '/') return 2;
   return 0;
}
void infixToPostfix(char *exp)
{
   Stack s;
   for (int i = 0; exp[i] != '\0'; i++)
   {
      if (isalnum(exp[i]))
cout<< exp[i];
      else if (exp[i] == '(')
s.push(exp[i]);
      else if (exp[i] == ')')
```

```
        {
            while (!s.isEmpty() &&s.peek() != '(')
cout<<s.pop();
s.pop();
        }
        else
        {
            while (!s.isEmpty() && precedence(s.peek()) >= precedence(exp[i]))
cout<<s.pop();
s.push(exp[i]);
        }
    }
    while (!s.isEmpty())
cout<<s.pop();
}

int main()
{
clrscr();
    char exp[50];
cout<< "Enter infix expression: ";
cin>>exp;
cout<< "Postfix expression: ";
infixToPostfix(exp);

getch();
    return 0;
}
```

Output

```
Enter infix expression: A+B*(C-D)
Postfix expression: ABCD-*+
```

23) Write Program to Evaluate Postfix Expression.

```
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
#include<stdlib.h>
class Stack
{
   public:
   int arr[50];
   int top;

Stack() { top = -1; }
   void push(int x) { arr[++top] = x; }
   int pop() { return arr[top--]; }
   bool isEmpty() { return top == -1; }
};

int evaluatePostfix(char *exp)
{
   Stack s;
   for (int i = 0; exp[i] != '\0'; i++)
   {
      if (isdigit(exp[i]))
s.push(exp[i] - '0');
      else
      {
         int val2 = s.pop();
         int val1 = s.pop();
         switch (exp[i])
         {
            case '+': s.push(val1 + val2); break;
```

```
            case '-': s.push(val1 - val2); break;
            case '*': s.push(val1 * val2); break;
            case '/': s.push(val1 / val2); break;
          }
       }
   }
   return s.pop();
}
int main()
{
clrscr();
   char exp[50];
cout<< "Enter postfix expression: ";
cin>>exp;
cout<< "Result of evaluation: " <<evaluatePostfix(exp) << "\n";

getch();
   return 0;
}
```

Output

```
Enter postfix expression: 23*45*+
Result of evaluation: 35
```

24) Add Two Polynomials Using Linked List.

```
#include<iostream.h>
#include<conio.h>
class Polynomial
{
   public:
   int coef;
   int exp;
   Polynomial *next;
Polynomial() { next = NULL; }
};
class PolyList
{
   public:
   Polynomial *head;
PolyList() { head = NULL; }

   void insertTerm(int coef, int exp)
   {
      Polynomial *newTerm = new Polynomial;
newTerm->coef = coef;
newTerm->exp = exp;
      if (head == NULL || head->exp < exp)
      {
newTerm->next = head;
         head = newTerm;
      }
      else
      {
         Polynomial *temp = head;
         while (temp->next != NULL && temp->next->exp >= exp)
            temp = temp->next;
```

```
newTerm->next = temp->next;
            temp->next = newTerm;
        }
    }
    void display()
    {
        Polynomial *temp = head;
        while (temp != NULL)
        {
cout<< temp->coef<< "x^" << temp->exp;
            if (temp->next != NULL)
cout<< " + ";
            temp = temp->next;
        }
cout<< "\n";
    }
    void addPolynomials(PolyList&p2)
    {
        Polynomial *temp1 = head;
        Polynomial *temp2 = p2.head;
PolyListresult;
        while (temp1 != NULL && temp2 != NULL)
        {
            if (temp1->exp > temp2->exp)
            {
result.insertTerm(temp1->coef, temp1->exp);
                temp1 = temp1->next;
            }
            else if (temp1->exp < temp2->exp)
            {
result.insertTerm(temp2->coef, temp2->exp);
                temp2 = temp2->next;
```

```
            }
            else
            {
result.insertTerm(temp1->coef + temp2->coef, temp1->exp);
               temp1 = temp1->next;
               temp2 = temp2->next;
            }
         }
         while (temp1 != NULL)
         {
result.insertTerm(temp1->coef, temp1->exp);
            temp1 = temp1->next;
         }
         while (temp2 != NULL)
         {
result.insertTerm(temp2->coef, temp2->exp);
            temp2 = temp2->next;
         }
result.display();
   }
};
int main()
{
clrscr();
PolyList p1, p2;
   int coef, exp;
cout<< "Enter terms for first polynomial\n";
   for (int i = 0; i< 3; i++)
   {
cout<< "Enter coefficient and exponent: ";
cin>>coef>>exp;
         p1.insertTerm(coef, exp);
```

```
    }
cout<< "Enter terms for second polynomial\n";
    for (int i = 0; i< 3; i++)
    {
cout<< "Enter coefficient and exponent: ";
cin>>coef>>exp;
        p2.insertTerm(coef, exp);
    }
cout<< "First Polynomial: ";
    p1.display();
cout<< "Second Polynomial: ";
    p2.display();
cout<< "Sum of Polynomials: ";
    p1.addPolynomials(p2);
getch();
    return 0;
}
```

Output

```
Enter terms for first polynomial
Enter coefficient and exponent: 3 2
Enter coefficient and exponent: 5 1
Enter coefficient and exponent: 6 0
Enter terms for second polynomial
Enter coefficient and exponent: 4 2
Enter coefficient and exponent: 3 1
Enter coefficient and exponent: 7 0
First Polynomial: 3x^2 + 5x^1 + 6x^0
Second Polynomial: 4x^2 + 3x^1 + 7x^0
Sum of Polynomials: 7x^2 + 8x^1 + 13x^0
```

25) Multiply Two Polynomials Using Linked List.

```
#include<iostream.h>
#include<conio.h>
class Polynomial
{
   public:
   int coef;
   int exp;
   Polynomial *next;
Polynomial() { next = NULL; }
};
class PolyList
{
   public:
   Polynomial *head;
PolyList() { head = NULL; }
   void insertTerm(int coef, int exp)
   {
      Polynomial *newTerm = new Polynomial;
newTerm->coef = coef;
newTerm->exp = exp;
      if (head == NULL || head->exp < exp)
      {
newTerm->next = head;
         head = newTerm;
      }
      else
      {
         Polynomial *temp = head;
         while (temp->next != NULL && temp->next->exp >= exp)
```

```
            temp = temp->next;
newTerm->next = temp->next;
            temp->next = newTerm;
        }
    }
    void display()
    {
        Polynomial *temp = head;
        while (temp != NULL)
        {
cout<< temp->coef<< "x^" << temp->exp;
            if (temp->next != NULL)
cout<< " + ";
            temp = temp->next;
        }
cout<< "\n";
    }

    void multiplyPolynomials(PolyList&p2)
    {
        Polynomial *temp1 = head;
        Polynomial *temp2;
PolyListresult;

        while (temp1 != NULL)
        {
            temp2 = p2.head;
            while (temp2 != NULL)
            {
result.insertTerm(temp1->coef * temp2->coef, temp1->exp + temp2->exp);
```

```
            temp2 = temp2->next;
          }
          temp1 = temp1->next;
       }
result.display();
   }
};
int main()
{
clrscr();
PolyList p1, p2;
   int coef, exp;

cout<< "Enter terms for first polynomial\n";
   for (int i = 0; i< 3; i++)
   {
cout<< "Enter coefficient and exponent: ";
cin>>coef>>exp;
      p1.insertTerm(coef, exp);
   }
cout<< "Enter terms for second polynomial\n";
   for (int i = 0; i< 3; i++)
   {
cout<< "Enter coefficient and exponent: ";
cin>>coef>>exp;
      p2.insertTerm(coef, exp);
   }
cout<< "First Polynomial: ";
   p1.display();
cout<< "Second Polynomial: ";
```

```
    p2.display();
cout<< "Product of Polynomials: ";
    p1.multiplyPolynomials(p2);
getch();
    return 0;
}
```

Output

```
Enter terms for first polynomial
Enter coefficient and exponent: 3 2
Enter coefficient and exponent: 5 1
Enter coefficient and exponent: 6 0
Enter terms for second polynomial
Enter coefficient and exponent: 4 2
Enter coefficient and exponent: 3 1
Enter coefficient and exponent: 7 0
First Polynomial: 3x^2 + 5x^1 + 6x^0
Second Polynomial: 4x^2 + 3x^1 + 7x^0
Product of Polynomials: 12x^4 + 29x^3 + 67x^2 + 41x^1 + 42x^0
```

26) Construct BST and Implement Traversing Techniques Recursively.

```
#include<iostream.h>
#include<conio.h>
class Node
{
   public:
   int data;
   Node *left, *right;
Node(int value)
   {
      data = value;
      left = right = NULL;
   }
};
class BST
{
   public:
   Node *root;
BST() { root = NULL; }
   void insert(int value)
   {
      root = insertRec(root, value);
   }

   Node* insertRec(Node* node, int value)
   {
      if (node == NULL)
         return new Node(value);
      if (value < node->data)
         node->left = insertRec(node->left, value);
```

```
        else if (value > node->data)
            node->right = insertRec(node->right, value);
        return node;
    }
    void inorder(Node *node)
    {
        if (node != NULL)
        {
inorder(node->left);
cout<< node->data << " ";
inorder(node->right);
        }
    }
    void preorder(Node *node)
    {
        if (node != NULL)
        {
cout<< node->data << " ";
            preorder(node->left);
            preorder(node->right);
        }
    }
    void postorder(Node *node)
    {
        if (node != NULL)
        {
postorder(node->left);
postorder(node->right);
cout<< node->data << " ";
        }
```

```
    }
};
int main()
{
clrscr();
    BST tree;
    int values[] = {50, 30, 20, 40, 70, 60, 80};
    for (int i = 0; i< 7; i++)
tree.insert(values[i]);
cout<< "Inorder Traversal: ";
tree.inorder(tree.root);
cout<< "\n";
cout<< "Preorder Traversal: ";
tree.preorder(tree.root);
cout<< "\n";
cout<< "Postorder Traversal: ";
tree.postorder(tree.root);
cout<< "\n";
getch();
    return 0;
}
```

Output

Inorder Traversal: 20 30 40 50 60 70 80

Preorder Traversal: 50 30 20 40 70 60 80

Postorder Traversal: 20 40 30 60 80 70 50

27) Implement Preorder Traversal on BST Non-Recursively.

```
#include<iostream.h>
#include<conio.h>
#include<stack.h>
class Node
{
   public:
   int data;
   Node *left, *right;
Node(int value)
   {
      data = value;
      left = right = NULL;
   }
};
class BST
{
   public:
   Node *root;
BST() { root = NULL; }
   void insert(int value)
   {
      root = insertRec(root, value);
   }
   Node* insertRec(Node* node, int value)
   {
      if (node == NULL)
         return new Node(value);
      if (value < node->data)
         node->left = insertRec(node->left, value);
      else if (value > node->data)
         node->right = insertRec(node->right, value);
      return node;
   }
   void preorder()
```

```
   {
      if (root == NULL)
return;
      stack<Node*>s;
s.push(root);
      while (!s.empty())
      {
         Node* node = s.top();
cout<< node->data << " ";
s.pop();

         if (node->right != NULL)
s.push(node->right);
         if (node->left != NULL)
s.push(node->left);
      }
   }
};
int main()
{
clrscr();
   BST tree;
   int values[] = {50, 30, 20, 40, 70, 60, 80};

   for (int i = 0; i< 7; i++)
tree.insert(values[i]);

cout<< "Preorder Traversal (Non-Recursive): ";
tree.preorder();
cout<< "\n";
getch();
   return 0;
}
```

Output

Preorder Traversal (Non-Recursive): 50 30 20 40 70 60 80

28) Implement Inorder Traversal on BST Non-Recursively.

```
#include<iostream.h>
#include<conio.h>
#include<stack.h>
class Node
{
   public:
   int data;
   Node *left, *right;

Node(int value)
   {
      data = value;
      left = right = NULL;
   }
};
class BST
{
   public:
   Node *root;
BST() { root = NULL; }
   void insert(int value)
   {
      root = insertRec(root, value);
   }
   Node* insertRec(Node* node, int value)
   {
      if (node == NULL)
         return new Node(value);
      if (value < node->data)
```

```
            node->left = insertRec(node->left, value);
        else if (value > node->data)
            node->right = insertRec(node->right, value);
        return node;
    }
    void inorder()
    {
        if (root == NULL)
return;
        stack<Node*>s;
        Node* curr = root;
        while (curr != NULL || !s.empty())
        {
            while (curr != NULL)
            {
s.push(curr);
curr = curr->left;
            }

curr = s.top();
s.pop();
cout<<curr->data << " ";
curr = curr->right;
        }
    }
};
int main()
{
clrscr();
    BST tree;
```

```
    int values[] = {50, 30, 20, 40, 70, 60, 80};
    for (int i = 0; i< 7; i++)
tree.insert(values[i]);
cout<< "Inorder Traversal (Non-Recursive): ";
tree.inorder();
cout<< "\n";
getch();
    return 0;
}
```

Output

```
Inorder Traversal (Non-Recursive): 20 30 40 50 60 70 80
```

29) Implement Postorder Traversal on BST Non-Recursively.

```
#include<iostream.h>
#include<conio.h>
#include<stack.h>
class Node
{
   public:
   int data;
   Node *left, *right;
Node(int value)
   {
      data = value;
      left = right = NULL;
   }
};
class BST
{
   public:
   Node *root;
BST() { root = NULL; }
   void insert(int value)
   {
      root = insertRec(root, value);
   }

   Node* insertRec(Node* node, int value)
   {
      if (node == NULL)
         return new Node(value);
      if (value < node->data)
         node->left = insertRec(node->left, value);
      else if (value > node->data)
         node->right = insertRec(node->right, value);
      return node;
   }
   void postorder()
   {
```

```
        if (root == NULL)
return;

        stack<Node*> s1, s2;
        s1.push(root);
        while (!s1.empty())
        {
           Node* node = s1.top();
s1.pop();
           s2.push(node);

           if (node->left != NULL)
              s1.push(node->left);
           if (node->right != NULL)
              s1.push(node->right);
        }
        while (!s2.empty())
        {
cout<<s2.top()->data << " ";
s2.pop();
        }
 }};
int main()
{
clrscr();
   BST tree;
   int values[] = {50, 30, 20, 40, 70, 60, 80};
   for (int i = 0; i< 7; i++)
tree.insert(values[i]);
cout<< "Postorder Traversal (Non-Recursive): ";
tree.postorder();
cout<< "\n";
getch();
   return 0;
}
```

Output

Postorder Traversal (Non-Recursive): 20 40 30 60 80 70 50

30) Implement Binary Search Techniques Recursively.

```
#include<iostream.h>
#include<conio.h>
class BinarySearch
{
   public:
   int binarySearch(int arr[], int left, int right, int target)
   {
      if (right >= left)
      {
         int mid = left + (right - left) / 2;
         if (arr[mid] == target)
            return mid;
         if (arr[mid] > target)
            return binarySearch(arr, left, mid - 1, target);
         return binarySearch(arr, mid + 1, right, target);
      }
      return -1;
   }
};
int main()
{
clrscr();
BinarySearchbs;
   int arr[] = {2, 5, 7, 10, 13, 17, 19, 23, 25, 30};
   int n = sizeof(arr) / sizeof(arr[0]);
   int target;
cout<< "Enter target value to search: ";
cin>>target;
   int result = bs.binarySearch(arr, 0, n - 1, target);
   if (result == -1)
cout<< "Element not found!\n";
   else
cout<< "Element found at index " << result << "\n";
getch();
   return 0;
}
```

Output

```
Enter target value to search: 17
Element found at index 5
```