



Data Structure Lab Using C/C++

MCA Sem 1

Name : Vaishnavi vidyadhar kothawade

Division : B

Batch : B2

Semester : MCA sem-1

Roll No : 32



Bharati Vidyapeeth's
Institute of Management & Information Technology
C.B.D. Belapur, Navi Mumbai 400614

Vision:

Providing high quality, innovative and value-based education in information technology to build competent professionals.

Mission

M1. Technical Skills:-To provide solid technical foundation theoretically as well as practically capable of providing quality services to industry.

M2. Development: -Department caters to the needs of students through comprehensive educational programs and promotes lifelong learning in the field of computer Applications.

M3. Ethical leadership:-Department develops ethical leadership insight in the students to succeed in industry, government and academia.

CERTIFICATE

This is to certify that the journal is the work of Miss. Vaishnavi Vidyadhar Kothawade, Roll No. 32 of MCA (Sem-1 Div:B) for the academic year 2023 – 2024

Subject Code: MCAE251

Subject Name: Data Structure Lab Using C/C++

Subject-in-charge

Principal

Date:

Bharati Vidyapeeth's Institute of Managment & Information Technology									
MCA Semester I AY 2023-24									
Subject : MCAL11 Data Structure Lab									
	CO1	Implement searching and sorting algorithms							
	CO2	Implement linear and non-linear data structures							
	CO3	Choose the appropriate data structures to solve complex real life problems							
	CO4	Analyze hashing techniques for data storage and retrieval							
Batch B2 INDEX									
Sr No.	Date	Topic	CO	PO	Attentiveness(10)	Presentat ion(10)	Understan ding(15)	Total (35)	Sign
1		Sorting Technique :-	CO1	PO1,					
	26-08-2023	Bubble & Insertion(menu driven) ,		PO2,					
	02-09-2023	Selection & Shell(menu driven) ,		PO3,					
	09-09-2023	Implement Radix sort		PO4, PO5, PO7, PS01					
2		Searching Techniques :-	CO1	PSO1					
	07-10-2023	Linear search , Binary Search(menu driven)		,PSO 2					
3		Hashing	CO4	PO1,					
	14 21-10-2023	To implement hashing technique:Modulo division, digit extraction, fold shift, fold boundary methods for collision resolution use linear probe.		PO2, PO3, PO4, PO5, PO7, PS01 ,PSO 2					
4		Linked lists:-							
		A menu driven program that implements singly linked list for the following operations :- Insert , Display ,Count , Delete , Search.	CO2	PO1,					
		A menu driven program that implements doubly linked list for the following Operations :- Insert , Display ,Count , Delete , Search.		PO2, PO3, PO4, PO5, PO7, PS01 ,PSO 2					
		A menu driven program that implements Singly circular linked list for the following operations :- Insert , Display ,Count , Delete , Search.							
5		Stacks:-	CO2	PO1,					
	28-10-2023	Program that Implements Stack using array		PO2, PO3, PO4, PO5, PO7, PS01 ,PSO 2					
		Program that Implements Stack using Linked list							
	18-11-2023	Implement to check Balancing of Parenthesis							
	11-01-2023	program that Evaluation of postfix Expression							
6		Queues :-							
		program that Implements ordinary Queue using array	CO2	PO1,					
	01-12-2023	program that Implements Circular Queue using Array		PO2, PO3, PO4, PO5, PO7, PS01 ,PSO 2					
		program that Implements priority Queue using Linked list							
		program that Implements Double ended Queue							

7		Binary search trees:-							
		program that implements Binary search tree for the following operation:-	CO2	PO1,					
		Insert, Recursive traversal: preorder, postorder, inorder, search, Largest node, smallest node, count number of nodes.		PO2,					
8		HEAP:-		PO3,					
		program that implements heap tree with methods:Build,Display,Delete	CO2	PO4,					
				PO5,					
9		Graphs:-		PO7,					
		Find a minimum spanning tree using any method	CO2	PSO1					
		Kruskal's Algorithm or Prim's Algorithm		,PSO					
		Create a Graph (Using Adjacency matrix)		2					
		Implementation of Graph traversal (DFS and BFS)							
			CO3	PO1,					
				PO2,					
22-12-2023		Group Project 15 marks		PO3,					
				PO4,					
				PO5,					
				PO7,					
				PSO1					
				,PSO					
				2					

Sorting Techniques:

1. Bubble and Insertion Sort

```
#include<iostream.h>
#include<conio.h>
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        swap(arr[i], arr[minIndex]);
    }
}
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]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}
void displayArray(int arr[], int n) {
    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    clrscr();
    while (1) {
        int n;
```

```

cout << "Enter the number of elements in the array: ";
cin >> n;
int arr[100];
cout << "Enter the elements of the array: ";
for (int i = 0; i < n; i++) {
    cin >> arr[i];
}
int choice;
cout << "\nChoose Sorting Algorithm:\n";
cout << "1. Selection Sort\n";
cout << "2. Bubble Sort\n";
cout << "Enter your choice (1 or 2): ";
cin >> choice;
switch (choice) {
    case 1:
        selectionSort(arr, n);
        break;
    case 2:
        bubbleSort(arr, n);
        break;
    default:
        cout << "Invalid choice!";
        return 1;
}
displayArray(arr, n);
char cont;
cout << "\nDo you want to continue? (y/n): ";
cin >> cont;
if (cont != 'y' && cont != 'Y') {
    break;
}
getch();
return 0;
}

```

```
Enter the number of elements in the array: 5
Enter the elements of the array: 3 8 2 1 9
```

```
Choose Sorting Algorithm:
```

```
1. Selection Sort
```

```
2. Bubble Sort
```

```
Enter your choice (1 or 2): 1
```

```
Sorted Array: 1 2 3 8 9
```

```
Do you want to continue? (y/n): y
```

```
Enter the number of elements in the array: 5
```

```
Enter the elements of the array: 2 9 4 1 6
```

```
Choose Sorting Algorithm:
```

```
1. Selection Sort
```

```
2. Bubble Sort
```

```
Enter your choice (1 or 2): 2
```

```
Sorted Array: 1 2 4 6 9
```

```
Do you want to continue? (y/n):
```

Selection and Shell Sort

```
#include<iostream.h>
```

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

```
void swap(int &a, int &b) {
```

```
    int temp = a;
```

```
    a = b;
```

```
    b = temp;
```

```
}
```

```
void selectionSort(int arr[], int n) {
```

```
    for (int i = 0; i < n - 1; i++) {
```

```
        int minIndex = i;
```

```
        for (int j = i + 1; j < n; j++) {
```

```
            if (arr[j] < arr[minIndex]) {
```

```
                minIndex = j;
```

```
            }}
```

```
        swap(arr[i], arr[minIndex]);
```

```
    }}
```

```
void shellSort(int arr[], int n) {
```

```
    for (int gap = n / 2; gap > 0; gap /= 2) {
```

```
        for (int i = gap; i < n; i++) {
```

```
            int temp = arr[i], j;
```

```
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
```

```
                arr[j] = arr[j - gap];
```

```
            }
```

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

```

    }}
void displayArray(int arr[], int n) {
    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    clrscr();
    while (1) {
        int n;
        cout << "Enter the number of elements in the array: ";
        cin >> n;
        int arr[100];
        cout << "Enter the elements of the array: ";
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }
        int choice;
        cout << "\nChoose Sorting Algorithm:\n";
        cout << "1. Selection Sort\n";
        cout << "2. Shell Sort\n";
        cout << "Enter your choice (1 or 2): ";
        cin >> choice;
        switch (choice) {
            case 1:
                selectionSort(arr, n);
                break;
            case 2:
                shellSort(arr, n);
                break;
            default:
                cout << "Invalid choice!";
                return 1;
        }
        displayArray(arr, n);
    }
}

```



```

char cont;
cout << "\nDo you want to continue? (y/n): ";
cin >> cont;
if (cont != 'y' && cont != 'Y') {
    break;
}
getch();
return 0;
}

```

```

Enter the number of elements in the array: 5
Enter the elements of the array: 3 6 1 9 5

```

```

Choose Sorting Algorithm:

```

```

1. Selection Sort

```

```

2. Shell Sort

```

```

Enter your choice (1 or 2): 1

```

```

Sorted Array: 1 3 5 6 9

```

```

Do you want to continue? (y/n): y

```

```

Enter the number of elements in the array: 5

```

```

Enter the elements of the array: 2 9 6 0 1

```

```

Choose Sorting Algorithm:

```

```

1. Selection Sort

```

```

2. Shell Sort

```

```

Enter your choice (1 or 2): 2

```

```

Sorted Array: 0 1 2 6 9

```

```

Do you want to continue? (y/n):

```

Radix Sort

```

#include<iostream.h>

```

```

#include<conio.h>

```

```

class radix

```

```

{

```

```

    int A[20],n,i;

```

```

    public:

```

```

        void getdata();

```

```

        void radsort(int*,int);

```

```

        void display(int*,int);

```

```

};

```

```

void radix::getdata()

```

```

{

```

```

    cout<<"\n\t\tRadix Sorting";

```

```

    cout<<"\n\t Enter the size of array:";

```

```

    cin>>n;

```

```

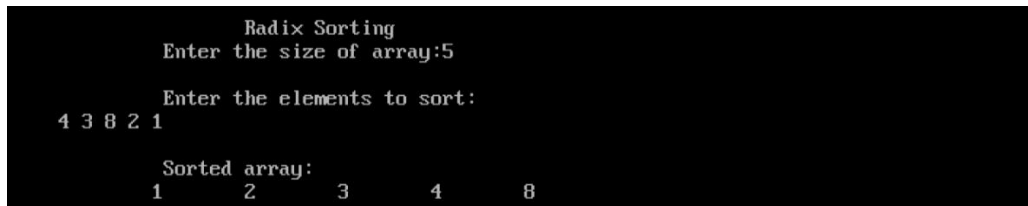
        cout<<"\n\t Enter the elements to sort:\n";
        for(i=0;i<n;i++)
        {
            cout<<"\t:";
            cin>>A[i];
        }
        radsort(A,n);
        display(A,n);
    }
    void radix::radsort(int *A,int n)
    {
        int temp;
        int bucket[10][20];
        int buck_count[10];
        int i,k,j,r,no_of_passes=0,divisor=1;
        int largest,pass_no;
        largest=A[0];
        for(i=1;i<n;i++)//find largest element
        {
            if(A[i]>largest)
                largest=A[i];
        }
        while(largest>0)//find passes find no of digits in number
        {
            no_of_passes++;
            largest=largest/10;
        }
        for(pass_no=0;pass_no<no_of_passes;pass_no++)
        {
            for(k=0;k<10;k++)//initialize the buckets
                buck_count[k]=0;//bucket count
            for(i=0;i<n;i++)
            {
                r=(A[i]/divisor)%10;
                bucket[r][buck_count[r]++]=A[i];
            }
            i=0;//collect elements from buckets

```

```

        for(k=0;k<10;k++)
        {
            for(j=0;j<buck_count[k];j++)
                A[i++]=bucket[k][j];
        }
        divisor=divisor*10;
    }}
void radix::display(int* A,int n)
{
    int i;
    cout<<"\n\t Sorted array:\n\t";
    for(i=0;i<n;i++)
        cout<<A[i]<<"\n\t";
}
void main()
{
    radix r;
    clrscr();
    r.getdata();
    getch();
}

```



```

Radix Sorting
Enter the size of array:5

Enter the elements to sort:
4 3 8 2 1

Sorted array:
1      2      3      4      8

```

2. Searching Techniques:

Linear and Binary Search

```
#include<iostream.h>
#include<conio.h>
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i; // Return the index where the key is found
        }
    }
    return -1; // Return -1 if the key is not found
}
int binarySearch(int arr[], int low, int high, int key) {
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) {
            return mid; // Return the index where the key is found
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1; // Return -1 if the key is not found
}
void displayResult(int index) {
    if (index != -1) {
        cout << "Element found at index: " << index << endl;
    } else {
        cout << "Element not found in the array." << endl;
    }
}
int main() {
    clrscr();
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int arr[100];
    cout << "Enter the elements of the sorted array: ";
```

```

for (int i = 0; i < n; i++) {
    cin >> arr[i];
}
int choice, key;
while (1) {
    cout << "\nChoose Search Algorithm:\n";
    cout << "1. Linear Search\n";
    cout << "2. Binary Search\n";
    cout << "Enter your choice (1 or 2): ";
    cin >> choice;
    switch (choice) {
        case 1:
            cout << "Enter the key to search: ";
            cin >> key;
            displayResult(linearSearch(arr, n, key));
            break;
        case 2:
            cout << "Enter the key to search: ";
            cin >> key;
            displayResult(binarySearch(arr, 0, n - 1, key));
            break;
        default:
            cout << "Invalid choice!";
            return 1;
    }
    char cont;
    cout << "\nDo you want to continue? (y/n): ";
    cin >> cont;
    if (cont != 'y' && cont != 'Y') {
        break;
    }
    getch();
    return 0;
}

```

```
Enter the number of elements in the array: 5
Enter the elements of the sorted array: 1 3 5 7 9
```

```
Choose Search Algorithm:
```

```
1. Linear Search
```

```
2. Binary Search
```

```
Enter your choice (1 or 2): 1
```

```
Enter the key to search: 3
```

```
Element found at index: 1
```

```
Do you want to continue? (y/n): y
```

```
Choose Search Algorithm:
```

```
1. Linear Search
```

```
2. Binary Search
```

```
Enter your choice (1 or 2): 2
```

```
Enter the key to search: 5
```

```
Element found at index: 2
```

```
Do you want to continue? (y/n): y
```

3.Hashing:

```
#include<iostream.h>
```

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

```
const int TABLE_SIZE = 10;
```

```
class HashTable {
```

```
private:
```

```
    int table[TABLE_SIZE];
```

```
    int linearProbe;
```

```
    // Hashing methods
```

```
    int moduloDivision(int key) {
```

```
        return key % TABLE_SIZE;
```

```
    }
```

```
    int digitExtraction(int key) {
```

```
        return key % 10; // Extract the last digit as the hash value
```

```
    }
```

```
    int foldShift(int key) {
```

```
        int sum = 0;
```

```
        while (key > 0) {
```

```
            sum += key % 10;
```

```
            key /= 10;
```

```
        }
```

```
        return sum % TABLE_SIZE;
```

```
    }
```

```
    int foldBoundary(int key) {
```

```
        int sum = 0;
```

```
        int numDigits = 0;
```

```
        while (key > 0) {
```

```
            sum += key % 10;
```

```
            key /= 10;
```

```
            numDigits++;
```

```

    }

    // Ensure that the hash value is within the table size
    return (sum + numDigits) % TABLE_SIZE;
}

int linearProbeHash(int hashValue) {
    return (hashValue + linearProbe) % TABLE_SIZE;
}

public:
    HashTable() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            table[i] = -1; // Initialize table with -1 to indicate empty slots
        }
    }

    void insert(int key) {
        int hashValue;

        cout << "\nEnter linear probe value: ";
        cin >> linearProbe;

        // Choose hashing method
        int choice;
        cout << "\nChoose Hashing Method:\n";
        cout << "1. Modulo Division\n";
        cout << "2. Digit Extraction\n";
        cout << "3. Fold Shift\n";
        cout << "4. Fold Boundary\n";
        cout << "Enter your choice (1-4): ";
        cin >> choice;

        switch (choice) {
            case 1:
                hashValue = moduloDivision(key);
                break;

```



```

        case 2:
            hashValue = digitExtraction(key);
            break;
        case 3:
            hashValue = foldShift(key);
            break;
        case 4:
            hashValue = foldBoundary(key);
            break;
        default:
            cout << "Invalid choice!";
            return;
    }

    // Linear probing if the slot is occupied
    while (table[hashValue] != -1) {
        hashValue = linearProbeHash(hashValue);
    }

    // Insert the key into the table
    table[hashValue] = key;

    cout << "Key " << key << " inserted at index " << hashValue << endl;
}

void display() {
    cout << "\nHash Table:\n";
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (table[i] != -1) {
            cout << "Index " << i << ": " << table[i] << endl;
        } else {
            cout << "Index " << i << ": Empty" << endl;
        }
    }
}

};

```

```

int main() {
    clrscr();

    HashTable hashTable;

    char cont;
    do {
        cout << "\n1. Insert Key\n";
        cout << "2. Display Hash Table\n";
        cout << "3. Exit\n";
        cout << "Enter your choice (1-3): ";
        int choice;
        cin >> choice;

        switch (choice) {
            case 1:
                int key;
                cout << "Enter the key to insert: ";
                cin >> key;
                hashTable.insert(key);
                break;
            case 2:
                hashTable.display();
                break;
            case 3:
                cout << "Exiting program.\n";
                getch();
                return 0;
            default:
                cout << "Invalid choice! Please enter a valid option.\n";
        }

        cout << "Do you want to continue? (y/n): ";
        cin >> cont;

    } while (cont == 'y' || cont == 'Y');

```

```
    getch();  
    return 0;  
}
```

```
1. Insert Key  
2. Display Hash Table  
3. Exit  
Enter your choice (1-3): 1  
Enter the key to insert: 3  
  
Enter linear probe value: 1  
  
Choose Hashing Method:  
1. Modulo Division  
2. Digit Extraction  
3. Fold Shift  
4. Fold Boundary  
Enter your choice (1-4): 1  
Key 3 inserted at index 3  
Do you want to continue? (y/n): y
```

```
1. Insert Key  
2. Display Hash Table  
3. Exit  
Enter your choice (1-3): 1  
Enter the key to insert: 3  
  
Enter linear probe value: 1  
  
Choose Hashing Method:  
1. Modulo Division  
2. Digit Extraction  
3. Fold Shift  
4. Fold Boundary  
Enter your choice (1-4): 2  
Key 3 inserted at index 4  
Do you want to continue? (y/n): y
```

```
1. Insert Key  
2. Display Hash Table  
3. Exit  
Enter your choice (1-3): 1  
Enter the key to insert: 5
```

```
1. Modulo Division
2. Digit Extraction
3. Fold Shift
4. Fold Boundary
Enter your choice (1-4): 4
Key 5 inserted at index 6
Do you want to continue? (y/n): y
```

```
1. Insert Key
2. Display Hash Table
3. Exit
Enter your choice (1-3): 2
```

```
Hash Table:
Index 0: Empty
Index 1: Empty
Index 2: Empty
Index 3: 3
Index 4: 3
Index 5: 5
Index 6: 5
Index 7: Empty
Index 8: Empty
Index 9: Empty
Do you want to continue? (y/n):
Enter linear probe value: 1
```

```
Choose Hashing Method:
1. Modulo Division
2. Digit Extraction
3. Fold Shift
4. Fold Boundary
Enter your choice (1-4): 3
Key 5 inserted at index 5
Do you want to continue? (y/n): y
```

```
1. Insert Key
2. Display Hash Table
3. Exit
Enter your choice (1-3): 1
Enter the key to insert: 5
```

```
Enter linear probe value: 1
```

```
Choose Hashing Method:
```

4. Linked List:

Singly Linked List

```
#include<iostream.h>
#include<conio.h>
class node {
    private:
        int data;
        node *address;

    public:
        int count(void);
        void add(int);
        void display(void);
        void sort(void);
        void insert(int,int);
        void remove(int);
        void search(int);
};
node * p;
node * q;
void node :: add(int num)
{
    q=p;
    if(p==NULL)
    {
        p=new node;
        p-> data=num;
        p-> address=NULL;
    }
    else
    {
        while(q->address!=NULL)
        {
            q=q->address;
        }
        q->address=new node;
        q->address->data=num;
        q->address->address=NULL;
    }
}
void node :: display(void)
{
    q=p;
    if( p == NULL)
    {
        cout<< "No Linkedlist";
    }else
    {
        while (q != NULL)
```

```

        {
            cout<<q->data<<endl;
            q = q->address;
        }
    }
}
int node :: count(void)
{
    q=p;
    int i = 0;
    if(p == NULL)
    {
        return 0;
    }
    else
    {
        while (q!= NULL)
        {
            i++;
            q = q-> address;
        }
        return i;
    }
}
void node :: sort()
{
    //node * q = p;
    node *i;
    node *j;
    int temp;
    for(i=p;i!=NULL;i=i->address)
    {
        for(j=i->address;j!=NULL;j=j->address)
        {
            if(i->data>j->data)
            {
                temp=i->data;
                i->data=j->data;
                j->data=temp;
            }
        }
    }
}
void node :: insert(int pos, int num)
{
    node *q = p;
    node *temp;
    int i;
    if(pos == 1)
    {
        p = new node;
    }

```

```

        p->data=num;
        p->address = q;
        return;
    }
    if(pos == 1+count())
    {
        add(num);
        return;
    }
    for(i=1; i<=(pos-2);i++)
    {
        q=q->address;
    }
    temp=q->address;
    q->address = new node;
    q->address->data = num;
    q->address->address=temp;
}
void node :: remove(int pos)
{
    node *temp;
    node *q=p;
    if(pos == 1)
    {
        p=q->address;
        delete(q);
        return;
    }
    for(int i=1; i<=(pos-2);i++)
    {
        q=q->address;
    }
    temp = q->address;
    q->address = q->address->address;
    delete (temp);
}
void node :: search(int num)
{
    node *q=p;
    int flag=0;
    int pos=0;
    for(q=p;q!=NULL;q=q->address)
    {
        if(q->data == num)
        {
            flag=1;
            break;
        }
        pos++;
    }
    if(flag==1)

```

```

        {
            cout<<"Number Found At Position : "<<pos;
        }
        else
        {
            cout<<"Number NA Found";
        }
    }
}

void main()
{
    int num, option,pos;
    node n;
    p=NULL;
    char ch='y';
    clrscr();
    while(ch == 'y')
    {
        cout<<"\n1.Add";
        cout<<"\n2.Display";
        cout<<"\n3.Count";
        cout<<"\n4.Sort";
        cout<<"\n5.Insert";
        cout<<"\n6.Delete";
        cout<<"\n7.Search";
        cout<<"\nEnter an option: ";
        cin>>option;
        switch(option)
        {
            case 1:
            {
                cout<<"\nEnter the value: ";
                cin>>num;
                n.add(num);
                cout<<"\nDo you Want to do again\n";
                break;
            }
            case 2:
            {
                n.display();
                cout<<"\nDo you Want to do again\n";
                break;
            }
            case 3:
            {
                num=n.count();
                cout<<num;
                cout<<"\nDo you Want to do again\n";
                break;
            }
            case 4:
            {
                cout<<"Sorted List";
            }
        }
    }
}

```



```

n.sort();
cout<<"\nDo you want to continue\n";
n.display();
break;
    }
case 5:
{
    cout<<"Enter Position: ";
    cin>>pos;
    cout<<"Enter Number: ";
    cin>>num;
    n.insert(pos,num);
    cout<<"\nDo you Want to do again\n";
    break;
    }
case 6:
{
    cout<<"Enter Position";
    cin>>pos;
    n.remove(pos);
    cout<<"\nDo you Want to do again\n";
    break;
    }
case 7:
{
    cout<<"Enter Number: ";
    cin>>num;
    n.search(num);
    cout<<"\nDo you Want to do again\n";
    break;
    }
    }
ch=getch(); } }

```

Doubly Linked List

```

#include<iostream.h>
#include<conio.h>
class node
{
    private:
    int data;
    node *next;
    node *prev;
    public:
    void add(int);
    void display(void);
    int count(void);
    void sort(void);
    void insert(int,int);
    void remove(int);

```

```

        void search(int);
        void reverse(void);
};
node *p;
void node::add(int num)
{
    node *q=p;

    if(p==NULL)
    {
        p=new node;
        p->data=num;
        p->next=NULL;
        p->prev=NULL;
    }
    else
    {
        while(q->next != NULL)
        {
            q=q->next;
        }
        q->next=new node;
        q->next->prev =q;
        q->next->data=num;
        q->next->next=NULL;
    }
}
void node::display(void)
{
    node *q=p;
    if(p==NULL)
    {
        cout<<"No Elements link list/n";
    }
    else
    {
        while(q!=NULL)
        {
            cout<<q->data;
            q=q->next;
        }
    }
}
int node::count(void)
{
    node *q=p;
    int i=0;
    if(p==NULL)
    {
        return 0;
    }
}

```

```

        else
        {
            while(q!=NULL)
            {
                i=i+1;
                q=q->next;
            }
            return i;
        }
    }
}

void node:: sort(void)
{
    node *i;
    node *j;
    int temp;
    for(i=p;i!=NULL;i=i->next)
    {
        for(j=i->next;j!=NULL;j=j->next)
        {
            if(i->data>j->data)
            {
                temp=i->data;
                i->data=j->data;
                j->data=temp;
            }
        }
    }
}

void node:: insert(int pos,int num)
{
    node*temp;
    node *q=p;
    int i;
    if(pos==1)
    {
        p=new node;
        p->data=num;
        p->next=q;
        p->prev=NULL;
        return;
    }
    if(pos==1+count())
    {
        add(num);
        return;
    }
    temp=q->next;
    for(i=1;i<=(pos-2);i++)
    {
        q=q->next;
    }
}

```

```

        q->next=new node;
        q->next->data=num;
        q->next->prev=q;
        q->next->next=temp;
        q->next->next->prev=q->next;
    }
    void node::remove(int pos)
    {
        int i;
        node*q=p;

        node*temp;
        if(pos==1)
        {
            p=q->next;
            delete (q);
            p->prev=NULL;
            return;
        }
        for(i=1;i<=(pos-2);i++)
        {
            q=q->next;
        }
        temp=q->next;
        q->next=q->next->next;
        q->next->prev=q;
        delete(temp);
    }
    void node:: search(int num)
    {
        node*q= p;
        int flag =0;
        int pos =0;
        for(q=p;q!=NULL;q=q->next)
        {
            if(q->data == num){
                flag =1;
                break;
            }

            pos++;
        }

        if(flag ==1)
        {
            cout<<"Number found at position"<<pos;
        }
        else
        {
            cout<<"number not found";
        }
    }

```

```

void node::reverse(void)
{
    node *q=p;
    if(p==NULL)
    {
        cout<<"NO LinkList";
    }
    else
    {
        while(q->next!=NULL)
        {
            q=q->next;
        }
        while(q!=NULL)
        {
            cout<<q->data<<endl;
            q=q->prev;
        }
    }
}

void main(void)
{
    int num,pos,option;
    node n;
    p=NULL;
    char ch='y';
    clrscr();
    while(ch=='y')
    {
        cout<<"\n 1.Add";
        cout<<"\n 2.Display";
        cout<<"\n 3.Count";
        cout<<"\n 4.Sort";
        cout<<"\n 5.Insert";
        cout<<"\n 6.Remove";
        cout<<"\n 7.Search";
        cout<<"\n 8.Reverse ";
        cout<<"\n Enter Option : " ;
        cin>>option;
        switch(option)
        {
            case 1:
            {
                cout<<"\nEnter the value: ";
                cin>>num;
                n.add(num);
                cout<<"\nDo you Want to do again\n";
                break;
            }
        }
    }
}

```

```

case 2:
{
    n.display();
    cout<<"\nDo you Want to do again\n";
    break;
}

case 3:
{
    num=n.count();
    cout<<num;
    cout<<"\nDo you Want to do again\n";
    break;
}

case 4:
{
    cout<<"Sorted List";
    n.sort();
    n.display();
    cout<<"\nDo you want to continue\n";
    break;
}

case 5:
{
    cout<<"Enter Position: ";
    cin>>pos;
    cout<<"Enter Number: ";
    cin>>num;
    n.insert(pos,num);
    cout<<"\nDo you Want to do again\n";
    break;
}

case 6:
{
    cout<<"\nEnter Position: ";
    cin>>pos;
    n.remove(pos);
    cout<<"\nDo you Want to do again\n";
    break;
}

case 7:
{
    cout<<"Enter Number: ";
    cin>>num;
    n.search(num);
    cout<<"\nDo you Want to do again\n";
    break;
}

case 8:
{
    cout<<"\nReverse List: \n";

```

```

        n.reverse();
        cout<<"\nDO YOU WANT TO CONTINUE \n";
        break;
    }

}

    ch=getch();
}
}

```

Circular Linked List

```

#include<iostream.h>
#include<conio.h>
class node
{
    private:
    int data;
    node *address;
    public:
    void add(int);
    void display(void);
    int count(void);
    void sort(void);
    void insert(int,int);
    void remove(int);
    void search(int);
};
node *p;
void node::add(int num)
{
    node *q=p;

    if(p==NULL)
    {
        p=new node;
        p->data=num;
        p->address=p;
    }
    else
    {
        while(q->address != p)
        {
            q=q->address;
        }
        q->address=new node;
        q->address->data=num;
        q->address->address=p;
    }
}

```

```

}
void node::display(void)
{
    node *q=p;
    if(p==NULL)
    {
        cout<<"No Circular link list/n";
    }
    else
    {
        do{
            cout<<q->data;
            q=q->address;
        }
        while(q!= p);
    }
}
int node::count(void)
{
    node *q=p;
    int i=0;
    if(p==NULL)
    {
        return 0;
    }
    else
    {
        do{
            i=i++;
            q=q->address;
        }while(q!=p);
        return i;
    }
}
void node:: sort(void)
{
    node *i;
    node *j;
    int temp;
    if(p==NULL)
    {
        cout<<"\n No link list";
    }
    else
    {
        for(i=p;i!=NULL;i=i->address)
        {
            for(j=i->address; j!=NULL; j=j->address)
            {
                if(i->data>j->data)

```



```

        {
            temp=i->data;
            i->data=j->data;
            j->data=temp;
        }

    }
}
}
}
void node:: insert(int pos,int num)
{
    node*temp;
    node *q=p;
    int i;
    if(pos==1)
    {
        p=new node;
        p->data=num;
        p->address=q;
        return;
    }
    if(pos==1+count())
    {
        add(num);
        return;
    }
    for(i=1;i<=(pos-2);i++)
    {
        q=q->address;
    }
    temp=q->address;
    q->address=new node;
    q->address->data=num;
    q->address->address=temp;
}
void node::remove(int pos)
{
    int i;
    node*q = p;
    node*temp;
    if(pos==1)
    {
        p=q->address;
        delete (q);
        return;
    }
    for(i=1;i<=(pos-2);i++)
    {
        q=q->address;
    }
}

```

```

        temp=q->address;
        q->address=q->address->address;
        delete(temp);
    }
void node:: search(int num)
{
    node*q= p;
    int flag =0;
    int pos =0;
    do{
        if(q->data == num){
            flag =1;
            break;
        }
        pos++;
        q=q->address;
    }while(q!=p);
    if(flag ==1)
    {
        cout<<"Number found at position"<<pos;
    }
    else
    {
        cout<<"number not found";
    }
}
void main(void)
{
    int num,pos,option;
    node n;
    p=NULL;
    char ch='y';
    clrscr();
    while(ch=='y')
    {
        cout<<"\n 1.Add";
        cout<<"\n 2.Display";
        cout<<"\n 3.Count";
        cout<<"\n 4.Sort";
        cout<<"\n 5.Insert";
        cout<<"\n 6.Remove";
        cout<<"\n 7.Search";

        cin>>option;
        switch(option)
        {
            case 1:
            {
                cout<<"\nEnter Number";
                cin>>num;
                n.add(num);
            }
        }
    }
}

```

```

        cout<<"\nDo you want to Continue";
        break;
    }
    case 2:
    {
        n.display();
        cout<<"\nDo you want to Continue";
        break;
    }
    case 3:
    {

        num=n.count();
        cout<<"\ncount of linklist is= "<<num;
        cout<<"\nDo you want to continue";
        break;
    }
    case 4:
    {
        n.sort();
        cout<<"\n is sorted";
        cout<<"\n Do you want to continue...";
        break;
    }
    case 5:
    {
        cout<<"\nENter position ";
        cin>>pos;
        cout<<"\nEnter the Number ";
        cin>>num;
        n.insert(pos,num);
        cout<<"\n Do you want to continue ";
        break;
    }
    case 6:
    {
        cout<<"\nEnterPosition ";
        cin>>pos;
        n.remove(pos);
        cout<<"Do You Want to continue ";
        break;
    }

    case 7:
    {
        cout<<"\nEnter Number to be searched";
        cin>>num;
        n.search(num);
        cout<<"\nDo you wantto continue";
        break;
    }

```

```

        default:
            cout<<"Not valid";
        }

        ch=getch();
    }
}

```

5.Stack:

Stack using Array

```

#include<stdio.h>
#include<conio.h>
#include<iostream.h>
class stack
{
    int data;
    stack *next;
    stack *pre;
public:
    void push(int);
    void display(void);
    int pop(void);
};
stack *top;
stack *bottom;

void stack::push(int num)
{
    if((top==NULL)&& (bottom==NULL))
    {
        top=bottom=new stack;
        top->data=num;
        top->next=NULL;
        top->pre=NULL;
    }
    else
    {
        top->next=new stack;
        top->next->data=num;
        top->next->pre=top;
        top->next->next=NULL;
        top=top->next;
    }
}

void stack::display()
{

```

```

        stack *q=bottom;
        if((top==NULL)&&(bottom==NULL))
        {
            cout<<"\nNo stack exists.";
        }
        else
        {
            while(q!=NULL)
            {
                cout<<q->data<<" ";
                q=q->next;
            }
        }
    }
}

```

```

int stack::pop()
{
    int num;
    if((top==NULL)&&(bottom==NULL))
    {
        return -1;
    }
    num=top->data;
    top=top->pre;
    if(top!=NULL)
    {
        delete(top->next);
        top->next=NULL;
    }
    else
    {
        delete(bottom);
        bottom=NULL;
    }
    return num;
}

```

```

void main()
{
    clrscr();
    stack obj;
    char ch='y';
    int option,num;
    while(ch=='y')
    {
        cout<<"\n1.Push";
        cout<<"\n2.Display";
        cout<<"\n3.Pop";
        cout<<"\n\tEnter choice";
    }
}

```

```

        cin>>option;
switch(option)
{
    case 1:
        cout<<"\n Enter value:";
        cin>>num;
        obj.push(num);
        cout<<"\tdo you want to continue:";
        break;
    case 2:
        cout <<"\nStack values:";
        obj.display();
        cout<<"\tdo you want to continue:";
        break;
    case 3:
        num= obj.pop();
        cout<<"1 element popped out "<<num;
        cout<<"\tdo you want to continue:";
        break;
}
ch=getch();
}
}

```

Stack using Doubly Linked List

```

#include<conio.h>
#include<stdio.h>
#include<iostream.h>
class stack
{
private:
    int data;
    stack *next;
    stack *pre;
public:
    void push(int);
    void display();
    int pop();
};
stack *top;
stack *bottom;
void stack::push(int num)
{
    if((top==NULL)&&(bottom==NULL))
    {
        top=bottom=new stack;
        top->data=num;
        top->next=NULL;
        top->pre=NULL;
    }
}

```

```

else
{
    top->next=new stack;
    top->next->data=num;
    top->next->next=NULL;
    top->next->pre=top;
    top=top->next;
}
}
void stack::display()
{
    stack *q=bottom;
    if((top==NULL)&&(bottom==NULL))
    {
        cout<<"\nEmpty stack";
    }
    else
    {
        while(q!=NULL)
        {
            cout<<q->data<<" ";
            q=q->next;
        }
    }
}
int stack::pop()
{
    int num;
    if((bottom==NULL)&&(top==NULL))
    {
        cout<<"\tEmpty stack";
        return -1;
    }
    num=top->data;
    top=top->pre;
    if( top!=NULL)
    {
        delete(top->next);
        top->next=NULL;
    }
    else
    {
        delete(bottom);
        bottom=NULL;
    }
    return num;
}
void main()
{
    clrscr();
    stack s1;

```

```

char ch='y';
int num,option;
while(ch=='y')
{
    cout<<"\n1.Add";
    cout<<"\n2.Display";
    cout<<"\n3.Remove";
    cout<<"\n\tEnter choice: ";
    cin>>option;
    switch(option)
    {
        case 1:
            cout<<"enter value of element: ";
            cin>>num;
            s1.push(num);
            cout<<"\tElement added.";
            cout<<"\nDo you want to continue : ";
            break;
        case 2:
            cout<<"\nStack Elements: ";
            s1.display();
            cout<<"\nDo you want to continue : ";
            break;
        case 3:
            num=s1.pop();
            cout<<num << " Removed";
            cout<<"\nDo you want to continue : ";
            break;
    }
    ch=getch();
}
}

```

Stack implementaion to check balancing of paranthesis

```

#include<iostream.h>
#include<string.h>
#include<conio.h>

```

```

class node {
public:
    char data;
    node* next;
};

```

```

class stack {

```



```

public:
    node* top;

    stack() {
        top = NULL;
    }

    void push(char x) {
        node* p = new node;
        p->data = x;
        p->next = NULL;
        if (top == NULL) {
            top = p;
        } else {
            node* save = top;
            top = p;
            p->next = save;
        }
    }

    char pop() {
        if (top == NULL) {
            cout << "UNDERFLOW";
            return '\0'; // Return null character in case of underflow
        } else {
            node* ptr = top;
            char data = top->data;
            top = top->next;
            delete ptr;
            return data;
        }
    }
};

int main() {
    clrscr();

```

```

int i;
stack s;
char c[30], a, y, z;
cout << "ENTER THE EXPRESSION:\n";
cin.getline(c, sizeof(c));

for (i = 0; i < strlen(c); i++) {
    if ((c[i] == '(') || (c[i] == '{') || (c[i] == '[')) {
        s.push(c[i]);
    } else {
        switch (c[i]) {
            case ')':
                a = s.pop();
                if ((a == '{') || (a == '[')) {
                    cout << " INVALID EXPRESSION!! ";
                    getch();
                    return 0;
                }
                break;
            case '}':
                y = s.pop();
                if ((y == '[') || (y == '(')) {
                    cout << "INVALID EXPRESSION!!!";
                    getch();
                    return 0;
                }
                break;
            case ']':
                z = s.pop();
                if ((z == '{') || (z == '(')) {
                    cout << "INVALID EXPRESSION!!!";
                    getch();
                    return 0;
                }
                break;
        }
    }
}
}

```

```

    }

    if (s.top == NULL) {
        cout << " BALANCED EXPRESSION!! ";
    } else {
        cout << " STRING IS NOT VALID!! ";
    }

    getch();
    return 0;
}

```

```

ENTER THE EXPRESSION:
(1+2)*(3+5)
BALANCED EXPRESSION!! _

ENTER THE EXPRESSION:
[1+2
STRING IS NOT VALID!! _

```

Evaluate the Postfix Expression using stack

```

#include<iostream>
#include<ctype.h>
#include<string.h>
#include<conio.h>
using namespace std;
class node
{
public:
    int data;
    node *next;
};
class stack
{
    node *top;
public:
    stack()
    {
        top=NULL;
    }
    void push(int x)
    {

```

```

node *p = new node();
p->data = x;
p->next = NULL;
if(top==NULL)
{
top = p;
}
else
{
node *save = top;
top = p;
p->next= save;
}}
int pop()
{
if(top==NULL)
{
cout<<&quot;\n UNDERFLOW&quot;;
}
else
{
node *ptr = top;
top = top->next;
return(ptr->data);
delete ptr;
}}
};
int main()
{
char x[30];
int a,b;
stack s;
cout<<&quot;ENTER THE BALANCED EXPRESSION: &quot;;
cin>>x;
for(int i= 0; i< strlen(x); i++)
{
if(x[i]==&#39;\n&#39;|| x[i]==&#39;\t&#39;)
{
continue;
}

```

```

if(isdigit(x[i]))
s.push(x[i]-0);
else
{
a= s.pop();
b= s.pop();
switch(x[i])
{
case '+':
s.push(a+b);
break;
case '-':
s.push(a-b);
break;
case '*':
s.push(a*b);
break;
case '/':
s.push(a/b);
break;
}}}
cout<<<"\n ANSWER IS: " <<s.pop() <<endl;
return 0;
}

```

6.Queue:

Simple Queue

```
#include<iostream.h>
```

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

```
class Queue{
    private:
        int data;
        Queue *next;
    public:
        void add(int);
        void display();
        int remove();
};

Queue *f;
Queue *r;

void Queue::add(int num){

    if((f==NULL) && (r==NULL))
    {
        f = r = new Queue;
        f->data = num;
        f->next = NULL;
    }
    else
    {
        while(r->next!=NULL)
        {
            r = r->next;
        }
        r->next = new Queue;
        //q = q->next;
        r->next->data = num;
        r->next->next = NULL;
        r = r->next;
    }
}

void Queue::display()
{
    Queue *q = f;
    if((f==NULL) && (r==NULL)){
        cout<<"Nothing to display"<<endl;
    }
    else{
```

```

        while(q!=NULL){
            cout<<q->data<<" ";
            q = q->next;
        }
    }
    cout<<endl;
}

int Queue::remove()
{
    int num;
    Queue *temp;
    if((f==NULL) && (r==NULL)){
        return -1;
    }
    else
    {
        temp = f;
        num = f->data;
        f = f->next;
        delete(temp);
        return num;
    }
}

void main(){
    int num, opt, pos;
    char ch = 'y';
    f = NULL;
    r = NULL;
    Queue q;

    clrscr();

    while(ch=='y'){
        cout<<"Choose any option: ";
        cout<<"\n1. Add";
        cout<<"\n2. Display";
        cout<<"\n3. Remove";
        cout<<"/n" + endl;

        cin>>opt;

        switch(opt)
        {
            case 1:
                cout<<"Enter your data(integer): ";
                cin>>num;
                q.add(num);
                break;

```

case 2:

```
cout<<"Elements are: ";
q.display();
break;
```

case 3:

```
num = q.remove();
if(num == -1)
    cout<<"No Queue present\n";
else
    cout<<"Removed element: "<<num<<endl;
break;
```

default:

```
{
    cout<<"Enter correct choice!"<<endl;
    break;
}
```

```
}
```

```
cout<<"DO YOU WANT TO CONTINUE? (y/n): "<<endl;
ch = getch();
```

```
}
```

```
}
```

Doubly Ended Queue

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

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

```
#include<iostream.h>
```

```
class queue
```

```
{
```

```
private:
```

```
    int data;
```

```
    queue *next;
```

```
    queue *prv;
```

```
public:
```

```
    void add(int);
```

```
    void display();
```

```
    int remove();
```

```
    void addf(int);
```

```
    int remover(void);
```

```
};
```

```
queue *front;
```

```
queue *rear;
```

```
void queue::add(int num)
```

```
{
```



```

if((front==NULL)&&(rear==NULL))
{
    front=rear=new queue;
    front->data=num;
    front->next=NULL;
    front->prv=NULL;
}
else
{
    rear->next=new queue;
    rear->next->data=num;
    rear->next->next=NULL;
    rear->next->prv=rear;
    rear=rear->next;
}
}
void queue::display()
{
    queue *q=front;
    if((front==NULL)&&(rear==NULL))
    {
        cout<<"\nStack empty";
    }else
    {
        while(q!=NULL)
        {
            cout<<q->data<<" ";
            q=q->next;
        } } }
int queue::remove()
{
    int num;
    queue *temp;
    if((front==NULL)&&(rear==NULL))
    {
        return -1;
    }
    else
    {
        temp=front;
        num=front->data;
        front=front->next;
        front->prv=NULL;
        delete(temp);
    }
    return num;
}
void queue::addf(int num)
{
    queue *temp;
    if((front==NULL)&&(rear==NULL))

```

```

{
front=rear=new queue;
front->data=num;
front->next=NULL;
front->prv=NULL;
}
else{
temp=front;
front=new queue;
front->data=num;
front->prv=NULL;
front->next=temp;
front->next->prv=front;
}
}
int queue::remover(void)
{
queue *temp;
int num;
if((front==NULL)&&(rear==NULL))
{
return -1;
}else
{
temp=rear;
num=rear->data;
rear=rear->prv;
rear->next=NULL;
delete(temp);
return num;
}}
void main()
{
queue z;
clrscr();
char c='y';
int num,op;
while(c=='y')
{
cout<<"\n1.Add";
cout<<"\n2.Display";
cout<<"\n3.Remove";
cout<<"\n4.Add from Front";
cout<<"\n5. Remove from Rear";
cout<<"\n Enter an option";
cin>>op;
switch(op)
{case 1:
cout<<"\nEnter value: ";
cin>>num;
z.add(num);

```

```

        cout<<"\n\tDo you want to continue: ";
        break;
    case 2:
        cout<<"Queue values:\t";
        z.display();
        cout<<"\n\tDo you want to continue: ";
        break;
    case 3:
        num=z.remove();
        cout<< num <<" removed";
        cout<< "Do you want to continue ";
        break;
    case 4:
        cout<<"Enter number";
        cin>>num;
        z.addf(num);
        cout<<"Do you want to continue";
        break;
    case 5:
        num=z.remover();
        cout<< num <<" removed";
        cout<< "Do you want to continue ";
        break;
    }
    c=getch();
}
}
}

```

Circular Queue

```

#include<iostream.h>
#include<conio.h>

```

```

class Queue {
private:
    int data;
    Queue* next;
public:
    void add(int);
    void display();
    int remove();
};

```

```

Queue* front;
Queue* rear;

```

```

void Queue::add(int num) {
    Queue* newNode = new Queue;
    newNode->data = num;
    newNode->next = NULL;
}

```

```

    if (front == NULL && rear == NULL) {
        front = rear = newNode;
        rear->next = front;
    }
    else {
        rear->next = newNode;
        rear = newNode;
        rear->next = front;
    }
}

```

```

void Queue::display() {
    if (front == NULL && rear == NULL) {
        cout << "Nothing to display" << endl;
    }
    else {
        Queue* temp = front;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != front);
    }
    cout << endl;
}

```

```

int Queue::remove() {
    int num;
    if (front == NULL && rear == NULL) {
        return -1;
    }
    else {
        Queue* temp = front;
        num = front->data;

        if (front == rear) {
            front = rear = NULL;
        }
        else {
            front = front->next;
            rear->next = front;
        }

        delete(temp);
        return num;
    }
}

```

```

void main() {
    int num, opt;

```

```

char ch = 'y';
front = NULL;
rear = NULL;
Queue q;

clrscr();

while (ch == 'y') {
    cout << "Choose any option: ";
    cout << "\n1. Add";
    cout << "\n2. Display";
    cout << "\n3. Remove";
    cout << endl;

    cin >> opt;

    switch (opt) {
    case 1:
        cout << "Enter your data (integer): ";
        cin >> num;
        q.add(num);
        break;

    case 2:
        cout << "Elements are: ";
        q.display();
        break;

    case 3:
        num = q.remove();
        if (num == -1)
            cout << "No Queue present\n";
        else
            cout << "Removed element: " << num << endl;
        break;

    default: {
        cout << "Enter the correct choice!" << endl;
        break;
    }
    }
    cout << "DO YOU WANT TO CONTINUE? (y/n): ";
    ch = getch();
}
}

```

Priority Queue

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
class queue
{ private:
    int data;
    queue *next;
public:
    void add(int);
    void display();
    int remove();
    void priority(void);
};
queue *front;
queue *rear;
void queue::add(int num)
{
    if((front==NULL)&&(rear==NULL))
    {
        front=rear=new queue;
        front->data=num;
        front->next=NULL;
    }
    else
    {
        rear->next=new queue;
        rear->next->data=num;
        rear->next->next=NULL;
        rear=rear->next;
    }
}
void queue::display()
{
    queue *q=front;
    if((front==NULL)&&(rear==NULL))
    {
        cout<<"\nStack empty";
    }
    else
    {
        while(q!=NULL)
        {
            cout<<q->data<<" ";
            q=q->next;
        }
    }
}
int queue::remove()
{
```

```

int num;
queue *temp;
if((front==NULL)&&(rear==NULL))
{
    return -1;
}
else
{
    temp=front;
    num=front->data;
    front=front->next;
    delete(temp);
}
return num;
}
void queue::priority(void)
{
int num;
if((front==NULL)&&(rear==NULL))
{
    cout<<"Queue is Empty";
}
else
{
    if(front->data>rear->data)
    {
        num=remove();
        cout<<num;
    }
    else
    {
        cout<<"Enter Number";
        cin>>num;
        add(num);
    }
}
}
void main()
{
    queue z;
    clrscr();
    char c='y';
    int num,op;
    while(c=='y')
    {
        cout<<"\n1.Add";
        cout<<"\n2.Display";
        cout<<"\n3.Remove";
        cout<<"\n4.Priority";
        cout<<"\n Enter an Option";
        cin>>op;
    }
}

```

```

switch(op)
{
    case 1:
        cout<<"\nEnter value: ";
        cin>>num;
        z.add(num);
        cout<<"\n\tDo you want to continue: ";
        break;
    case 2:
        cout<<"Queue values:\t";
        z.display();
        cout<<"\n\tDo you want to continue: ";
        break;
    case 3:
        num=z.remove();
        cout<< num <<" removed";
        cout<< "Do you want to continue ";
        break;
    case 4:
        z.priority();
        cout<<"Do you want to continue";
        break;
}
c=getch();
}
}

```


7. Binary Search Tree:

```
#include<iostream.h>
```

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

```
class bst{
    private:
        int data;
        bst *left;
        bst *right;
    public:
        void add(int,bst**);
        void preorder(bst*);
        void postorder(bst*);
        void inorder(bst*);
        void max(bst *q);
        void min(bst *q);
        void count(bst **);
        int search (int,bst **);
};
bst *root;
void bst ::add(int num,bst **q){
    if(*q==NULL){
        (*q)=new bst;
        (*q)->data=num;
        (*q)->left=NULL;
        (*q)->right=NULL;
    }
    else{
        if(num<(*q)->data){
            add(num,&((*q)->left));
        }
        else{
            add(num,&((*q)->right));
        }
    }
}
void bst :: preorder(bst *q){
    if(q!=NULL){
        cout<<q->data<<" ";
        preorder(q->left);
        preorder(q->right);
    }
}
void bst :: postorder(bst *q){
    if(q!=NULL){
        postorder(q->left);
        postorder(q->right);
        cout<<q->data<<" ";
    }
}
```

```

    }
}

void bst :: inorder(bst *q){
    if(q!=NULL){
        inorder(q->left);
        cout<<q->data<<" ";
        inorder(q->right);
    }
}

void bst::max(bst *q){
    if(q==NULL)
        cout<<"Tree doesnt exit\n";
    else{
        while(q->right != NULL)
        {
            q = q->right;
        }
        cout<<q->data<<endl;
    }
}

void bst::min(bst *q){
    if(q==NULL)
        cout<<"Tree doesnt exit\n";
    else{
        while(q->left != NULL)
        {
            q = q->left;
        }
        cout<<q->data<<endl;
    }
}

void bst::count(bst **q)
{
    int c=0;
    bst *temp=*q;
    if(root==NULL)
    {
        cout<<c;
    }
    else
    {
        c=1;

        while(temp->left!=NULL)
        {
            c++;

```

```

        temp=temp->left;
    }

    temp=(*q);

    while(temp->right!=NULL)
    {
        c++;
        temp=temp->right;
    }
    cout<<"Number of Nodes"<<c;
}
}

```

```

void main()
{
    clrscr();
    int num,option;
    char ch='y';
    root=NULL;
    bst b;
    while(ch=='y'){
        cout<<"\n1.add:";
        cout<<"\n2.preorder:";
        cout<<"\n3. postorder";
        cout<<"\n4. in order";
        cout<<"\n5. Max element";
        cout<<"\n6. Min element";
        cout<<"\n7. Count elements";
        cout<<"\n7. Search elements";
        cout<<"\n Enter an option:";

        cout<<"\n";

        cin>>option;

        switch(option)
        {
            case 1:
                cout<<"Enter number:";
                cin>>num;
                b.add(num,&root);
                cout<<" ";
                break;
            case 2:
                b.preorder(root);
                cout<<" ";
                break;
            case 3:
                b.postorder(root);

```

```

        cout<<" ";
        break;
    case 4:
        b.inorder(root);
        break;

    case 5:
        b.max(root);
        break;

    case 6:
        b.min(root);
        break;

    case 7:
        b.count(&root);
        break;

    case 8:
        cout<<"\n enter the number";
        cin>>num;
        pos = b.search(num,&root);
        if(pos!=-1)
        {
            cout<<"\n number not found";
        }
        else
        {
            cout<<"\n number found";
        }
        cout<<"\n do you want to continue ";
        break;
    }
    cout<<"Do you want to continue";
    ch=getch();

}

}

```

Heap:

```
#include<iostream>
```

```
#include<vector>
```

```
using namespace std;
```

```
class MinHeap {
```

```
private:
```

```
vector<int> heap;
```

```
// Helper function to heapify a subtree with the root at given index
```

```
void heapify(int index) {
```

```
    int leftChild = 2 * index + 1;
```

```
    int rightChild = 2 * index + 2;
```

```
    int smallest = index;
```

```
    // Compare with left child
```

```
    if (leftChild < heap.size() && heap[leftChild] < heap[smallest]) {
```

```
        smallest = leftChild;
```

```
    }
```

```
    // Compare with right child
```

```
    if (rightChild < heap.size() && heap[rightChild] < heap[smallest]) {
```

```
        smallest = rightChild;
```

```
    }
```

```
    // If the smallest is not the root, swap and recursively heapify the affected subtree
```

```
    if (smallest != index) {
```

```
        swap(heap[index], heap[smallest]);
```

```
        heapify(smallest);
```

```
    }
```

```
}
```

```
public:
```

```
    // Function to build a heap from a given array
```

```
    void buildHeap(const vector<int>& arr) {
```

```
        heap = arr;
```

```
        // Start from the last non-leaf node and heapify all nodes in reverse order
```

```
        for (int i = (heap.size() / 2) - 1; i >= 0; i--) {
```

```
            heapify(i);
```

```
        }
```

```
}
```

```
    // Function to display the elements of the heap
```

```
    void displayHeap() {
```

```
        cout << "Heap: ";
```

```
        for (int num : heap) {
```

```

        cout << num << " ";
    }
    cout << endl;
}

// Function to delete the minimum element from the heap (root)
void deleteMin() {
    if (heap.empty()) {
        cout << "Heap is empty. Cannot delete." << endl;
        return;
    }

    // Replace the root with the last element
    heap[0] = heap.back();
    heap.pop_back();

    // Heapify the root
    heapify(0);
}

};

int main() {
    MinHeap minHeap;

    // Example usage
    vector<int> inputArray = {4, 10, 3, 5, 1};

    minHeap.buildHeap(inputArray);
    minHeap.displayHeap();

    // Deleting the minimum element
    minHeap.deleteMin();
    minHeap.displayHeap();

    return 0;
}

```

8. Graph:

Adjacency Matrix - Directed

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

class DAdjMatrix
{
    private:
        int adjMatrix[20][20], n;
    public:
        void createGraph();
        void display();
        void insertV();
        void deleteV();
        void insertE(int, int);
        void deleteE(int ,int);

        DAdjMatrix()
        {
            for(int i=1; i<=20; i++){
                for(int j=1; j<=20; j++){
                    adjMatrix[i][j] = 0;
                }
            }
        }
};

void DAdjMatrix :: createGraph()
{
    int i, maxEdge, or, ds;

    cout<<"Enter no of vertices: ";
    cin>>n;
    cout<<endl;

    maxEdge = (n*(n-1)) / 2;

    cout<<"Enter edges:\n";

    for(i=1; i<=maxEdge; i++)
    {
        cout<<"Enter origin=0 and des=0 to exit\n";

        cout<<"Enter origin: ";
        cin>>or;
```

```

        cout<<"Enter des: ";
        cin>>ds;

        if(or==0 || ds==0)
            break;

        if(or>n || or<0 || ds>n || ds<0)
        {
            cout<<"Invalid Edge\n";
            i--;
            continue;
        }
        adjMatrix[or][ds] = 1;
    }
}

void DAdjMatrix :: display()
{
    int i, j;
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++){
            cout<<adjMatrix[i][j]<<"\t";
        }
        cout<<"\n";
    }
}

void DAdjMatrix::insertV()
{
    int i;
    n++;
    cout<<"\nNo. of vertices: "<<n<<endl;

    for(i=1; i<n; i++)
    {
        adjMatrix[i][n] = 0;
        adjMatrix[n][i] = 0;
    }
}

void DAdjMatrix::deleteV()
{
    int i;
    for(i=1; i<n; i++)
    {
        adjMatrix[i][n] = 0;
        adjMatrix[n][i] = 0;
    }
    n--;
    cout<<"\nNo. of vertices: "<<n<<endl;
}

void DAdjMatrix::insertE(int or, int ds)

```



```

{
    if(or>n || or<0 || ds>n || ds<0)
    {
        cout<<"Invalid Edge\n";
    }
    else{
        adjMatrix[or][ds] = 1;
    }
}

void DAdjMatrix::deleteE(int or, int ds)
{
    if(or>n || or<0 || ds>n || ds<0)
    {
        cout<<"Invalid Edge\n";
    }
    else{
        adjMatrix[or][ds] = 0;
    }
}

```

```

void main()
{
    DAdjMatrix m;
    int opt, or, ds;
    char ch='y';

    clrscr();

    m.createGraph();

    while(ch=='y'){
        cout<<"Choose any option: ";
        cout<<"\n0. Display";
        cout<<"\n1. Insert Vertex";
        cout<<"\n2. Delete Vertex";
        cout<<"\n3. Insert Edge";
        cout<<"\n4. Delete Edge";

        cout<<endl;

        cin>>opt;

        switch(opt)
        {
            case 0:
                m.display();
                break;

            case 1:

```

```

        m.insertV();
        break;

    case 2:
        m.deleteV();
        break;

    case 3:
        cout<<"Enter origin: ";
        cin>>or;

        cout<<"Enter des: ";
        cin>>ds;

        m.insertE(or, ds);
        break;

    case 4:
        cout<<"Enter origin: ";
        cin>>or;

        cout<<"Enter des: ";
        cin>>ds;

        m.deleteE(or, ds);
        break;

    default:
    {
        cout<<"Enter correct choice!"<<endl;
        break;
    }

}
cout<<"DO YOU WANT TO CONTINUE? (y/n): "<<endl;
ch = getch();
}

```

Adjacency Matrix - Undirected

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

class UAdjMatrix
{
    private:
        int adjMatrix[20][20], n;
    public:
        void createGraph();
        void display();
        void insertV();
        void deleteV();
        void insertE(int, int);
        void deleteE(int ,int);

        UAdjMatrix()
        {
            for(int i=1; i<=20; i++){
                for(int j=1; j<=20; j++){
                    adjMatrix[i][j] = 0;
                }
            }
        }

        void UAdjMatrix :: createGraph()
        {
            int i, maxEdge, or, ds;
            cout<<"Enter no of vertices: ";
            cin>>n;
            cout<<endl;

            maxEdge = (n*(n-1)) / 2;

            cout<<"Enter edges:\n";

            for(i=1; i<=maxEdge; i++)
            {
                cout<<"Enter origin=0 and des=0 to exit\n";

                cout<<"Enter origin: ";
                cin>>or;

                cout<<"Enter des: ";
                cin>>ds;

                if(or==0 || ds==0)
                    break;

                if(or>n || or<0 || ds>n || ds<0)
                {
```

```

        cout<<"Invalid Edge\n";
        i--;
        continue;
    }
    adjMatrix[or][ds] = 1;
    adjMatrix[ds][or] = 1;
}
}

void UAdjMatrix :: display()
{
    int i, j;
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++){
            cout<<adjMatrix[i][j]<<"\t";
        }
        cout<<"\n";
    }
}

void UAdjMatrix::insertV()
{
    int i;
    n++;
    cout<<"\nNo. of vertices: "<<n<<endl;

    for(i=1; i<n; i++)
    {
        adjMatrix[i][n] = 0;
        adjMatrix[n][i] = 0;
    }
}

void UAdjMatrix::deleteV()
{
    int i;

    for(i=1; i<n; i++)
    {
        adjMatrix[i][n] = 0;
        adjMatrix[n][i] = 0;
    }
    n--;
    cout<<"\nNo. of vertices: "<<n<<endl;
}

void UAdjMatrix::insertE(int or, int ds)
{
    if(or>n || or<0 || ds>n || ds<0)
    {
        cout<<"Invalid Edge\n";
    }
}

```

```

        }
        else{
            adjMatrix[or][ds] = 1;
            adjMatrix[ds][or] = 1;
        }
    }

void UAdjMatrix::deleteE(int or, int ds)
{
    if(or>n || or<0 || ds>n || ds<0)
    {
        cout<<"Invalid Edge\n";
    }
    else{
        adjMatrix[or][ds] = 0;
        adjMatrix[ds][or] = 0;
    }
}

```

```

void main()
{
    UAdjMatrix m;
    int opt, or, ds;
    char ch='y';

    clrscr();

    m.createGraph();

    while(ch=='y'){
        cout<<"Choose any option: ";
        cout<<"\n0. Display";
        cout<<"\n1. Insert Vertex";
        cout<<"\n2. Delete Vertex";
        cout<<"\n3. Insert Edge";
        cout<<"\n4. Delete Edge";

        cout<<endl;

        cin>>opt;

        switch(opt)
        {
            case 0:
                m.display();
                break;

            case 1:
                m.insertV();
                break;

```

```

        case 2:
            m.deleteV();
            break;

        case 3:
            cout<<"Enter origin: ";
            cin>>or;

            cout<<"Enter des: ";
            cin>>ds;

            m.insertE(or, ds);
            break;

        case 4:
            cout<<"Enter origin: ";
            cin>>or;

            cout<<"Enter des: ";
            cin>>ds;

            m.deleteE(or, ds);
            break;

        default:
        {
            cout<<"Enter correct choice!"<<endl;
            break;
        }

    }
    cout<<"DO YOU WANT TO CONTINUE? (y/n): "<<endl;
    ch = getch();
}

```

Graph traversal

```
#include<iostream.h>
```

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

```
class gdfs
```

```
{
```

```
    private:
```

```
    int adj[10][10];
```

```
    int visitedarr[10];
```

```
    int n;
```

```
    public:
```

```
    void buildadj();
```

```
    void dfs(int);
```

```
    gdfs()
```

```
    {
```

```
        int i,j;
```

```
        for(i=0;i<10;i++)
```

```
        {
```

```
            for(j=0;j<10;j++)
```

```
            {
```

```
                adj[i][j]=0;
```

```
            }
```

```
            visitedarr[i]=0;
```

```
        }
```

```
    }
```

```
};
```

```
void gdfs :: buildadj()
```

```
{
```

```
    int i,j;
```

```
    cout<<"Enter number of vertices: ";
```

```
    cin>>n;
```

```
    for(i=0;i<n;i++)
```

```
    {
```

```
        for(j=0;j<n;j++)
```

```
        {
```

```
            cin>>adj[i][j];
```

```
        }
```

```
    }
```

```
};
```

```
void gdfs :: dfs(int x)
```

```
{
```

```
    int j;
```

```
    visitedarr[x] = 1;
```

```
    cout<< x << " is visited."<< endl;
```

```
        for(j=0;j<n;j++)
        {
            if((adj[x][j] == 1)&&(visitedarr[j] == 0))
            {
                dfs(j);
            }
        }
    };

void main()
{
    clrscr();
    gdfs d;
    d.buildadj();
    d.dfs(0);
    getch();
}
```


9.Project

TOWER OF HANOI

Code:

```
#include<stdio.h>

#include<iostream>

#include<math.h>

using namespace std;

//Abhijeet Ramdas Khadsare


// Define structures for the three stacks
struct node1
{
    int data1;
    node1 *next1;
}*top1 = NULL, *p1 = NULL, *np1 = NULL;

struct node2
{
    int data2;
    node2 *next2;
}*top2 = NULL, *p2 = NULL, *np2 = NULL;

struct node3
{
    int data3;
    node3 *next3;
}*top3 = NULL, *p3 = NULL, *np3 = NULL;

// Function to push an element onto the first stack
void push1(int data)
{
    np1 = new node1;
    np1->data1 = data;
    np1->next1 = NULL;
```

```

    if (top1 == NULL)
    {
        top1 = np1;
    }
    else
    {
        np1->next1 = top1;
        top1 = np1;
    }
} // Function to pop an element from the first stack

int pop1()
{
    int b = 999;
    if (top1 == NULL)
    {
        return b;
    }
    else
    {
        p1 = top1;
        top1 = top1->next1;
        return(p1->data1);
        delete(p1);
    }
}

void push2(int data)
{
    np2 = new node2;
    np2->data2 = data;
    np2->next2 = NULL;
    if (top2 == NULL)

```

```

{
    top2 = np2;
}
else
{
    np2->next2 = top2;
    top2 = np2;
}
}
int pop2()
{
    int b = 999;
    if (top2 == NULL)
    {
        return b;
    }
    else
    {
        p2 = top2;
        top2 = top2->next2;
        return(p2->data2);
        delete(p2);
    }
}
void push3(int data)
{
    np3 = new node3;
    np3->data3 = data;
    np3->next3 = NULL;
    if (top3 == NULL)
    {

```

```

        top3 = np3;
    }
    else
    {
        np3->next3 = top3;
        top3 = np3;
    }
}

int pop3()
{
    int b = 999;
    if (top3 == NULL)
    {
        return b;
    }
    else
    {
        p3 = top3;
        top3 = top3->next3;
        return(p3->data3);
        delete(p3);
    }
} // Function to get the top of the smallest non-empty stack

int top_of_stack()
{
    if (top1 != NULL && top1->data1 == 1 )
    {
        return 1;
    }
    else if (top2 != NULL && top2->data2 == 1)
    {

```

```

        return 2;
    }
    else if (top3 != NULL && top3->data3 == 1)
    {
        return 3;
    }
}

```

// Display functions for each stack

```

void display1()
{
    cout<<endl;
    node1 *p1;
    p1 = top1;
    cout<<"Tower1-> "<<"t";
    while (p1 != NULL)
    {
        cout<<p1->data1<<"t";
        p1 = p1->next1;
    }
    cout<<endl;
}

```

```

void display2()
{
    node2 *p2;
    p2 = top2;
    cout<<"Tower2-> "<<"t";
    while (p2 != NULL)
    {
        cout<<p2->data2<<"t";
        p2 = p2->next2;
    }
}

```

```

        cout<<endl;
    }
    void display3()
    {
        node3 *p3;
        p3 = top3;
        cout<<"Tower3-> "<<"t";
        while (p3 != NULL)
        {
            cout<<p3->data3<<"t";
            p3 = p3->next3;
        }
        cout<<endl;
        cout<<endl;
    } // Tower of Hanoi function(toh- tower of hanoi)
    void toh(int n)
    {
        int i, x, a, b;
        // Loop through the moves of the Tower of Hanoi
        for (i = 0; i < (pow(2,n)); i++)
        {
            display1();
            display2();
            display3();
            x = top_of_stack();
            if (i % 2 == 0)
            {
                if (x == 1)
                {
                    push3(pop1());
                }
            }
        }
    }

```

```
else if (x == 2)
{
    push1(pop2());
}
else if (x == 3)
{
    push2(pop3());
}
}
else
{
    if (x == 1)
    {
        a = pop2();
        b = pop3();
        if (a < b && b != 999)
        {
            push3(b);
            push3(a);
        }
        else if (a > b && a != 999)
        {
            push2(a);
            push2(b);
        }
        else if (b == 999)
        {
            push3(a);
        }
        else if (a == 999)
        {

```

```
        push2(b);
    }
}
else if (x == 2)
{
    a = pop1();
    b = pop3();
    if (a < b && b != 999)
    {
        push3(b);
        push3(a);
    }
    else if (a > b && a != 999)
    {
        push1(a);
        push1(b);
    }
    else if (b == 999)
    {
        push3(a);
    }
    else if (a == 999)
    {
        push1(b);
    }
}
else if (x == 3)
{
    a = pop1();
    b = pop2();
    if (a < b && b != 999)
```



```

        {
            push2(b);
            push2(a);
        }
        else if (a > b && a != 999)
        {
            push1(a);
            push1(b);
        }
        else if (b == 999)
        {
            push2(a);
        }
        else if (a == 999)
        {
            push1(b);
        } // Handling moves for odd steps in Tower of Hanoi
    }
}
}

// Main function
int main()
{
    int n, i;
    cout<<"enter the number of disks: ";
    cin>>n;

    // Initialize the first tower with disks
    for (i = n; i >= 1; i--)
    {
        push1(i);
    }
}

```

```

    }    // Solve the Tower of Hanoi

    toh(n);

    return 0;
}

```

Output:

```
enter the number of disks: 3
```

```
Tower1-> t1t2t3t
Tower2-> t
Tower3-> t
```

```
Tower1-> t2t3t
Tower2-> t
Tower3-> t1t
```

```
Tower1-> t3t
Tower2-> t2t
Tower3-> t1t
```

```
Tower1-> t3t
Tower2-> t1t2t
Tower3-> t
```

```
Tower1-> t3t
Tower2-> t1t2t
Tower3-> t
```

```
Tower1-> t
Tower2-> t1t2t
Tower3-> t3t
```

```
Tower1-> t1t
Tower2-> t2t
Tower3-> t3t
```

```
Tower1-> t1t
Tower2-> t
Tower3-> t2t3t
```

```
Tower1-> t
Tower2-> t
Tower3-> t1t2t3t
```