**// 1. Write a program to find the mean and the median of the numbers stored in an array**

#include <iostream>

using namespace std;

void Swap(int \*x, int \*y)

{

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void BubbleSort(int arr[], int n)

{

    int flag;

    for (int i = 0; i < n - 1; i++)

    {

        flag = 0;

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

        {

            if (arr[j] > arr[j + 1])

            {

                Swap(&arr[j], &arr[j + 1]);

                flag++;

            }

        }

        if (flag == 0)

        {

            return;

        }

    }

}

int main()

{

    int n;

    cout << "Enter the number of numbers you wanted in your array" << endl;

    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++)

    {

        cout << "Enter the " << i + 1 << " element in the array" << endl;

        cin >> arr[i];

    }

    int sum = 0;

    for (int i = 0; i < n; i++)

    {

        sum += arr[i];

    }

    float mean = sum / n;

    BubbleSort(arr, n);

    float median;

    if (n % 2 == 0)

    {

        median = (arr[n / 2 - 1] + arr[n / 2]) / 2;

    }

    else if (n % 2 != 0)

    {

        median = arr[(n - 3) / 2];

    }

    cout<<"Mean of the input array is "<<mean<<endl;

    cout<<"Median of the input array is "<<median<<endl;

    return 0;

}

**// 2. Write a program to insert one element in an array and delete an element from an array.**

#include <iostream>

using namespace std;

class Array

{

public:

    int arr[10];

    int length;

    int size;

    Array(int arr1[], int length, int size);

    void Display();

    void Insert(int key, int index);

    int Delete(int index);

};

Array::Array(int arr1[], int length, int size)

{

    this->size = size;

    this->length = length;

    for (int i = 0; i < length; i++)

    {

        arr[i] = arr1[i];

    }

}

void Array::Display()

{

    for (int i = 0; i < length; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

void Array::Insert(int key, int index)

{

    if (length == size)

    {

        cout << "Array is Full" << endl;

        return;

    }

    else if (index >= size || index < 0)

    {

        cout << "Wrong index" << endl;

    }

    else

    {

        for (int i = length - 1; i >= index; i--)

        {

            arr[i + 1] = arr[i];

        }

        length++;

        arr[index] = key;

    }

}

int Array::Delete(int index)

{

    int x = -1;

    if (length == 0)

    {

        cout<<"Array is Empty"<<endl;

    }

    else if (index < 0 || index >= size)

    {

        cout<<"Wrong index"<<endl;

    }

    else  if( -1 < index < length)

    {

        x = arr[index];

        for (int i = index; i < length - 1; i++)

        {

           arr[i] = arr[i +1];

        }

        length --;

    }

    return x;

}

int main()

{

    int arr1[] = {1,3,5,7,9};

    Array a(arr1,5,10);

    cout<<"Intially the array : ";

    a.Display();

    cout<<"After the Insert : ";

    a.Insert(11,5);

    a.Display();

    cout<<"After the delete : ";

    a.Delete(0);

    a.Display();

    return 0;

}

**// 3. Write a program to search for a number in an array.**

#include <iostream>

using namespace std;

int LinearSearch(int arr[], int n, int key)

{

    for (int i = 0; i < n; i++)

    {

        if (key == arr[i])

        {

            return i;

        }

    }

    return -1;

}

int BinarySearch(int arr[], int n, int key)

{

    int l = 0;

    int h = n - 1;

    int mid = (l + h) / 2;

    while (l <= h)

    {

        if (arr[mid] == key)

        {

            return mid;

        }

        else if (key < arr[mid])

        {

            h = mid - 1;

        }

        else if (key > arr[mid])

        {

            l = mid + 1;

        }

    }

    return -1;

}

int main()

{

    int arr[] = {1, 3, 2, 6, 4, 11, 9};

    int n = sizeof(arr) / sizeof(arr[0]);

    int x = LinearSearch(arr, n, 2);

    if (x != -1)

    {

        cout << "Element is found at " << x << " index" << endl;

    }

    else

    {

        cout << "Not found" << endl;

    }

    int y = BinarySearch(arr,n,6);

     if (y != -1)

    {

        cout << "Element is found at " << y << " index" << endl;

    }

    else

    {

        cout << "Not found" << endl;

    }

    return 0;

}

**// 4. Write a program to sort an array.**

#include <iostream>

using namespace std;

void Swap(int \*x, int \*y)

{

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void BubbleSort(int arr[], int n)

{

    int flag;

    for (int i = 0; i < n - 1; i++)

    {

        flag = 0;

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

        {

            if (arr[j] > arr[j + 1])

            {

                Swap(&arr[j], &arr[j + 1]);

                flag++;

            }

        }

        if (flag == 0)

        {

            return;

        }

    }

}

void Display(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

int main()

{

    int arr[] = {12,3,45,67,32,43,98,76};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout<<"Array befor Sorting : ";

    Display(arr, n);

    cout<<"Array after Sorting : ";

    BubbleSort(arr, n);

    Display(arr, n);

    return 0;

}

**// 5. Write a program to merge two sorted arrays.**

#include <iostream>

using namespace std;

int \*Merge(int arr1[], int n1, int arr2[], int n2)

{

    int \*arr3 = new int[n1 + n2];

    int i, j, k;

    i = j = k = 0;

    while (i != n1 && j != n2)

    {

        if (arr1[i] > arr2[j])

        {

            arr3[k++] = arr2[j++];

        }

        else

        {

            arr3[k++] = arr1[i++];

        }

    }

    for (; i < n1; i++)

    {

        arr3[k++] = arr1[i];

    }

    for (; j < n2; j++)

    {

        arr3[k++] = arr2[j];

    }

    return arr3;

}

void Display(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

int main()

{

    int arr1[] = {23, 65, 121, 340, 875, 2589};

    int arr2[] = {42, 54, 110, 441, 523, 990, 2233};

    int n = sizeof(arr1) / sizeof(arr1[0]);

    int n1 = sizeof(arr2) / sizeof(arr2[0]);

    int \*arr = Merge(arr1, n, arr2, n1);

    cout << "Ist sorted array : ";

    Display(arr1, n);

    cout << "2nd Sorted array : ";

    Display(arr2, n1);

    cout << "Merged array : ";

    Display(arr, n + n1);

    delete [] arr;

    return 0;

}

**// 6. Write a program to store the marks obtained by 10 students in 5 courses in a two-dimensional array**

#include <iostream>

using namespace std;

int main()

{

    int arr[5][10];

    for (int i = 0; i < 5; i++)

    {

        for (int j = 0; j < 10; j++)

        {

            cout << "Enter the marks of course no. " << i + 1 << " by student no. " << j + 1 << " is : ";

            cin >> arr[i][j];

        }

        cout << endl;

    }

    return 0;

}

**// 7. Write a program to implement a linked list.**

#include <iostream>

using namespace std;

class Node

{

public:

    int data;

    Node \*next;

};

int main()

{

    Node \*head = new Node;

    head->data = 4;

    head->next = NULL;

    cout << head->data << endl;

    return 0;

}

**// 8. Write a program to insert a node in a linked list and delete a node from a linked list.**

#include <iostream>

using namespace std;

class Node

{

public:

    int data;

    Node \*next;

};

class LinkedList

{

    Node \*first;

public:

    LinkedList();

    ~LinkedList();

    void Insert(int key);

    int Delete(int index);

    void Display();

    int count();

};

void LinkedList::Display()

{

    Node \*p = first;

    while (p != NULL)

    {

        cout<<p->data<<" ";

        p = p->next;

    }

    cout<<endl;

}

int LinkedList::count()

{

    Node \*p = first;

    int count = 0;

    while (p != NULL)

    {

        p = p->next;

        count++;

    }

    return count;

}

LinkedList::LinkedList()

{

    first = NULL;

}

LinkedList::~LinkedList()

{

    while (first != NULL)

    {

        Node \*p = first;

        first = first->next;

        delete p;

    }

}

void LinkedList::Insert(int key)

{

    if (first == NULL)

    {

        first = new Node;

        first->data = key;

        first->next = NULL;

    }

    else

    {

        Node \*t = new Node;

        t->data = key;

        t->next = NULL;

        Node \*p = first;

        while (p->next != NULL)

        {

            p = p->next;

        }

        p->next = t;

    }

}

int LinkedList::Delete(int index)

{

    if (count() == 0)

    {

        cout << "linkedList is Empty" << endl;

        return -1;

    }

    else if (index > 0 && index <= count())

    {

        if (index == 1)

        {

            Node \*p = first->next;

            int x = first->data;

            delete first;

            first = p;

            return x;

        }

        else

        {

            Node \*p, \*q;

            p = first;

            for (int i = 0; i < index - 1; i++)

            {

                q = p;

                p = p->next;

            }

            int x = p->data;

            q->next = p->next;

            delete p;

            return x;

        }

    }

    return -1;

}

int main()

{

    LinkedList l;

    l.Insert(12);

    l.Display();

    l.Insert(9);

    l.Display();

    l.Insert(5);

    l.Display();

    l.Delete(2);

    l.Display();

    return 0;

}

**// 9. Write a program to print the elements of a linked list in reverse order without disturbing the linked list.**

#include <iostream>

using namespace std;

class Node

{

public:

    Node \*prev;

    int data;

    Node \*next;

};

class LinkedList

{

    Node \*first;

public:

    LinkedList(int arr[], int n);

    ~LinkedList();

    void Display();

    void ReverseDiplay();

};

LinkedList::LinkedList(int arr[], int n)

{

    first = new Node;

    first->data = arr[0];

    first->prev = NULL;

    first->next = NULL;

    Node \*p = first;

    for (int i = 1; i < n; i++)

    {

        Node \*t = new Node;

        t->data = arr[i];

        t->next = NULL;

        p->next = t;

        t->prev = p;

        p = t;

    }

}

LinkedList::~LinkedList()

{

    Node \*p;

    while (first != NULL)

    {

        p = first;

        first = first->next;

        delete p;

    }

}

void LinkedList::Display()

{

    Node \* p = first;

    while(p != NULL)

    {

        cout<<p->data<<" ";

        p = p->next;

    }

    cout<<endl;

}

void LinkedList::ReverseDiplay()

{

    Node \* p = first;

    while (p->next != NULL)

    {

         p = p->next;

    }

    while (p != NULL)

    {

        cout<<p->data<<" ";

        p = p ->prev;

    }

    cout<<endl;

}

int main()

{

    int arr[] = {1,4,7,5,4};

    int n = sizeof(arr)/sizeof(arr[0]);

    LinkedList l(arr,n);

    cout<<"Original order : ";

    l.Display();

    cout<<"Reverse Order : ";

    l.ReverseDiplay();

    return 0;

}

**// 10. Write a program to reverse a linked list.**

#include<iostream>

using namespace std;

class Node

{

    public:

        int data;

        Node\* next;

};

class LinkedList

{

    Node \* first;

    public:

        LinkedList(int arr[],int n);

        ~LinkedList();

        void reverse();

        void Display();

};

LinkedList::LinkedList(int arr[],int n)

{

    first = new Node ;

    first->data = arr[0];

    first->next = NULL;

    Node \* p = first;

    for (int i = 1; i < n; i++)

    {

        Node \* t = new Node;

        t->data = arr[i];

        t->next = NULL;

        p->next = t;

        p = t;

    }

}

LinkedList::~LinkedList()

{

    Node \*p;

    while (first != NULL)

    {

        p = first;

        first = first->next;

        delete p;

    }

}

void LinkedList::reverse()

{

    int count = 0;

    Node \* p = first;

    while (p != NULL)

    {

        p = p->next;

        count++;

    }

    p = first;

    int arr[count];

    for (int i = 0; i < count; i++)

    {

        arr[i] = p->data;

        p = p->next;

    }

    p = first;

    for (int i = 0; i < count; i++)

    {

        p->data = arr[count-i-1];

        p = p->next;

    }

}

void LinkedList::Display()

{

    Node \* p = first;

    while(p != NULL)

    {

        cout<<p->data<<" ";

        p = p->next;

    }

    cout<<endl;

}

int main()

{

    int arr[] = {3,5,4,8,6,1,2};

    int n = sizeof(arr)/sizeof(arr[0]);

    LinkedList l(arr,n);

    cout<<"Original order : ";

    l.Display();

    cout<<"After Reverse : ";

    l.reverse();

    l.Display();

 return 0;

}

**// 11. Write a program to add two polynomials using linked lists.**

#include <iostream>

using namespace std;

class Node

{

public:

    int coff;

    int pow;

    Node \*next;

};

class Poly

{

    Node \*root;

public:

    Poly();

    ~Poly();

    void Create(int coff, int pow);

    // Node \* add(Node \* p, Node \* q);

    Node \*getroot() { return root; }

    void Display();

};

void Poly::Display()

{

    Node \*p = root;

    while (p != NULL)

    {

        cout << p->coff << "x^" << p->pow;

        if (p->next != NULL)

        {

            if (p->next->coff > 0)

            {

                cout << " + ";

            }

            else if (p->next->coff < 0)

            {

                cout << " - ";

            }

        }

        p = p->next;

    }

    cout << endl;

}

void Poly::Create(int coff1, int pow1)

{

    if (root == NULL)

    {

        root = new Node;

        root->coff = coff1;

        root->pow = pow1;

        root->next = NULL;

        return;

    }

    else

    {

        Node \*p = root;

        while (p->next != NULL)

        {

            p = p->next;

        }

        Node \*t = new Node;

        t->coff = coff1;

        t->pow = pow1;

        t->next = NULL;

        p->next = t;

    }

}

Poly::Poly()

{

    root = NULL;

}

Poly::~Poly()

{

    Node \*p;

    while (root != NULL)

    {

        p = root;

        root = root->next;

        delete p;

    }

}

Poly add1(Node \*p, Node \*q)

{

    Poly result;

    while (p != NULL && q != NULL)

    {

        if (p->pow > q->pow)

        {

            result.Create(p->coff, p->pow);

            p = p->next;

        }

        else if (p->pow < q->pow)

        {

            result.Create(q->coff, q->pow);

            q = q->next;

        }

        else if (p->pow == q->pow)

        {

            result.Create(p->coff + q->coff, p->pow);

            p = p->next;

            q = q->next;

        }

    }

    while (p != NULL)

    {

        result.Create(p->coff, p->pow);

        p = p->next;

    }

    while (q != NULL)

    {

        result.Create(q->coff, q->pow);

        q = q->next;

    }

    return result;

}

int main()

{

    Poly p1;

    Poly p2;

    for (int i = 0; i < 2; i++)

    {

        int n;

        cout << "Enter the number of terms in polynomial " << i + 1 << endl;

        cin >> n;

        for (int j = 0; j < n; j++)

        {

            int x, y;

            cout << "Enter the cofficient and power of term no. " << j + 1 << " of Polynomail No. " << i + 1 << endl;

            cin >> x >> y;

            if (i == 0)

            {

                p1.Create(x, y);

            }

            else if (i == 1)

            {

                p2.Create(x, y);

            }

        }

    }

    cout<<"Polynomial No. 1 : ";

    p1.Display();

    cout<<"Polynomial No. 2 : ";

    p2.Display();

    Poly p3;

    p3 = add1(p1.getroot(), p2.getroot());

    cout<<"Resultant Polynomail : ";

    p3.Display();

    return 0;

}

**// 12. Write a program to implement a doubly-linked list.**

#include<iostream>

using namespace std;

class Node

{

    public:

        Node \*prev;

        int data;

        Node \* next;

};

class LinkedList

{

    Node \* first;

    public:

        LinkedList();

        ~LinkedList();

        void Display();

        void Insert(int key);

        int Delete();

};

LinkedList::LinkedList()

{

    first = NULL;

}

LinkedList::~LinkedList()

{

    Node \* p ;

    while(first != NULL)

    {

        p = first;

        first = first->next;

        delete p;

    }

}

void LinkedList:: Display()

{

    Node \* p = first;

    while (p != NULL)

    {

        cout<<p->data<<" ";

        p = p->next;

    }

    cout<<endl;

}

void LinkedList::Insert(int key)

{

    if (first == NULL)

    {

        first = new Node ;

        first->data = key;

        first->next = NULL;

        first->prev  = NULL;

        return ;

    }

    else

    {

        Node \* p = first;

        while(p->next != NULL)

        {

            p = p->next;

        }

        Node \* t = new Node;

        t->data = key;

        t->next  =NULL;

        p->next = t;

        t->prev = p;

    }

}

int LinkedList::Delete()

{

    if (first ==NULL)

    {

      cout<<"LinkedList is empty"<<endl;

      return -1;

    }

    else{

        Node \* p = first;

        Node \*q;

        while (p->next != NULL)

        {

            q = p;

            p = p->next;

        }

        q->next = NULL;

        int x = p->data;

        delete p;

        return x;

    }

}

int main()

{

    LinkedList l;

    l.Insert(12);

    l.Display();

    l.Insert(13);

    l.Display();

    l.Insert(14);

    l.Display();

    l.Delete();

    l.Display();

  return 0;

}

**// 13. Write a program to implement a stack using an array.**

#include <iostream>

using namespace std;

class Stack

{

    int size;

    int \*arr;

    int top;

public:

    Stack(int size);

    ~Stack();

    void push(int key);

    int pop();

    int topvalue();

    bool empty();

    bool full();

    void Display();

};

Stack::Stack(int size)

{

    this->size = size;

    top = -1;

    arr = new int[size];

}

Stack::~Stack()

{

    delete [] arr;

}

void Stack::push(int key)

{

    if (full() == 1)

    {

        cout << "Stack Overflow" << endl;

        return;

    }

    else

    {

        top++;

        arr[top]=  key;

    }

}

int Stack::pop()

{

    if (empty() == 1)

    {

        cout<<"Stack Underflow"<<endl;

        return -1;

    }

    else

    {

        int x = arr[top];

        top--;

        return x;

    }

}

int Stack::topvalue()

{

    return arr[top];

}

bool Stack::full()

{

    if (top == size -1)

    {

        return true;

    }

    else

    {

        return false;

    }

}

bool Stack::empty()

{

    if (top == -1)

    {

        return true;

    }

    else

    {

        return false;

    }

}

void Stack::Display()

{

    for (int i = 0; i <= top; i++)

    {

        cout<<arr[i]<<" ";

    }

    cout<<endl;

}

int main()

{

    Stack stk(12);

    stk.push(12);

    stk.Display();

    stk.push(15);

    stk.Display();

    stk.push(18);

    stk.Display();

    stk.pop();

    stk.Display();

    cout<<stk.empty()<<endl;

    cout<<stk.full()<<endl;

    cout<<stk.topvalue()<<endl;

    return 0;

}

**// 14. Write a program to implement a stack using a linked list.**

#include<iostream>

using namespace std;

class Node

{

    public:

        int data;

        Node \* next;

};

class Stack

{

    Node \* first;

    public:

        Stack();

        ~Stack();

        void push(int key);

        int pop();

        bool IsEmpty();

        bool IsFull();

        int top();

        void Display();

};

void Stack:: Display()

{

    Node \* p = first;

    while (p != NULL)

    {

        cout<<p->data<<" ";

        p = p->next;

    }

    cout<<endl;

}

Stack::Stack()

{

    first = NULL;

}

Stack::~Stack()

{

    Node \* p ;

    while (first != NULL)

    {

        p = first;

        first = first->next;

        delete p;

    }

}

bool Stack::IsEmpty()

{

    if (first == NULL)

    {

        return true;

    }

    else

    {

        return false;

    }

}

bool Stack ::IsFull()

{

    Node \* t = new  Node ;

    if (t == NULL)

    {

       return 1;

    }

    else

    {

        return false;

    }

}

void Stack::push(int key)

{

    if (IsFull() ==  1)

    {

        cout<<"Stack Overflow"<<endl;

        return;

    }

    if (first == NULL)

    {

        first = new Node;

        first->data = key;

        first->next = NULL;

        return ;

    }

    else

    {

        Node \* p = first;

        Node \* t = new Node ;

        t->data =key;

        t->next = p;

        first = t;

    }

}

int Stack::pop()

{

    if (IsEmpty() == 1)

    {

        cout<<"Stack Underflow"<<endl;

        return -1;

    }

    else

    {

        Node \* p = first->next;

        int x = first->data;

        delete first;

        first = p;

        return x;

    }

}

int Stack::top()

{

    if (IsEmpty() == 1)

    {

        cout<<"Stack Underflow"<<endl;

        return -1;

    }

    else

    {

        return first->data;

    }

}

int main()

{

    Stack stk;

    stk.push(12);

    stk.Display();

    stk.push(15);

    stk.Display();

    stk.push(18);

    stk.Display();

    stk.pop();

    stk.Display();

  return 0;

}

**// 15. Write a program to implement a queue using an array.**

#include<iostream>

using namespace std;

class Queue

{

    int front;

    int rear ;

    int size;

    int\* arr;

    public:

        Queue(int size);

        ~Queue();

        void enqueue(int key);

        int Dequeue();

        int top();

        int back();

        bool IsEmpty();

        bool IsFull();

        void Display();

};

void Queue::Display()

{

    for (int i = front +1; i <= rear; i++)

    {

      cout<<arr[i]<<" ";

    }

    cout<<endl;

}

Queue::Queue(int size)

{

    this->size = size;

    front = rear = -1;

    arr  = new int[size];

}

Queue::~Queue()

{

    delete [] arr;

}

bool Queue::IsEmpty()

{

    if (front == rear)

    {

        return true;

    }

    else

    {

        return false;

    }

}

bool Queue::IsFull()

{

    if (rear == size -1)

    {

        return true;

    }

    else

    {

        return false;

    }

}

void Queue::enqueue(int key)

{

    if (IsFull() == 1)

    {

        cout<<"Queue is full"<<endl;

        return;

    }

    else

    {

        rear ++;

        arr[rear] = key;

    }

}

int Queue::Dequeue()

{

    if (IsEmpty() == 1)

    {

        cout<<"Queue is Empty";

        return -1;

    }

    else

    {

        front ++;

        return arr[front];

    }

}

int Queue::back()

{

    if (IsEmpty() == 1)

    {

        cout<<"Queue is Empty"<<endl;

        return -1;

    }

    else

    {

        return arr[rear];

    }

}

int Queue::top()

{

    if (IsEmpty() == 1)

    {

        cout<<"Queue is Empty";

        return -1;

    }

    else

    {

        return arr[front +1];

    }

}

int main()

{

  Queue q(10);

  q.enqueue(12);

  q.Display();

  q.enqueue(15);

  q.Display();

  q.enqueue(18);

  q.Display();

  q.Dequeue();

  q.Display();

  return 0;

}

**// 16. Write a program to implement a queue using a linked list.**

#include <iostream>

using namespace std;

class Node

{

public:

    int data;

    Node \*next;

};

class Queue

{

    Node \*front = NULL;

    Node \*rear = NULL;

public:

    ~Queue();

    void enqueue(int);

    int dequeue(void);

    bool IsEmpty(void);

    bool IsFull(void);

    void Display(void);

};

bool Queue::IsEmpty()

{

    if (front == NULL)

    {

        return true;

    }

    else

    {

        return false;

    }

}

bool Queue::IsFull()

{

    Node \*t = new Node;

    if (t == NULL)

    {

        return true;

    }

    else

    {

        return false;

    }

}

void Queue::enqueue(int key)

{

    if (IsFull() == 1)

    {

        cout << "Queue is Full" << endl;

    }

    else

    {

        Node \*t = new Node;

        t->data = key;

        t->next = NULL;

        if (front == NULL)

        {

            front = rear = t;

        }

        else

        {

            rear->next = t;

            rear = t;

        }

    }

}

int Queue::dequeue()

{

    int x = -1;

    if (IsEmpty() == 1)

    {

        cout << "Queue is Empty" << endl;

    }

    else

    {

        Node \*p = front;

        front = front->next;

        x = p->data;

        delete p;

    }

    return x;

}

Queue::~Queue()

{

    Node \*p = front;

    Node \*t = NULL;

    while (p != NULL)

    {

        t = p;

        p = p->next;

        delete t;

    }

}

void Queue::Display()

{

    Node \*p = front;

    while (p != NULL)

    {

        cout<<p->data<<" ";

        p = p->next;

    }

    cout<<endl;

}

int main()

{

    Queue l;

    l.enqueue(9);

    l.enqueue(8);

    l.enqueue(7);

    l.Display();

    l.dequeue();

    l.Display();

    cout<<l.IsEmpty()<<endl;

    cout<<l.IsFull()<<endl;

    return 0;

}