#include <iostream>

#include<Windows.h>

using namespace std;

//// Circular Queue

//

//class CircularQueue

//{

// int\* data;

// int capacity;

// int front = 0;

// int rear = -1; // cari index

// int count = 0;

//public:

// CircularQueue() = default;

//

// CircularQueue(int size)

// {

// data = new int[size] {};

// capacity = size;

// front = 0;

// rear = 0;

// count = 0;

// }

//

// void enqueue(int value)

// {

// data[rear] = value;

// ++rear;

// ++count;

// }

//

// void dequeue()

// {

// auto newarray = new int[size()] {};

// for (int x = 0; x < size() - 1; x++)

// {

// newarray[x] = data[x + 1];

// }

// newarray[size() - 1] = peek();

// if (size() != 0)

// {

// delete[]data;

// }

// data = newarray;

// newarray = nullptr;

// }

//

// int peek() const

// {

// return data[front];

// }

//

// int size() const

// {

// return count;

// }

//

// bool IsEmpty() const

// {

// return size() == 0;

// }

//

// bool IsFull() const

// {

// return capacity == size();

// }

//

// void Show() const

// {

// for (int x = 0; x < size(); x++)

// {

// cout << data[x] << " ";

// }

// cout << endl;

// }

//

// ~CircularQueue()

// {

// delete[]data;

// }

//};

//void main()

//{

// CircularQueue queue(3);

// for (int x = 0; x < 3; x++)

// {

// queue.enqueue(x + 1);

// }

// queue.Show();

// cout << "Start" << endl;

// for (int x = 0; x < 3; x++)

// {

// queue.dequeue();

// queue.Show();

// }

//}

//// PriorityQueue

//

//template<typename T>

//class PriorityQueue {

// T\* data;

// int capacity;

// int front = 0;

// int rear = -1;

// int count = 0;

//public:

// PriorityQueue() = default;

// PriorityQueue(int size)

// {

// data = new T[size]{};

// capacity = size;

// front = 0;

// rear = 0;

// count = 0;

// }

// void enqueue(const T& value) {

// data[rear] = value;

// ++rear;

// ++count;

// }

// void dequeue() {

// int index = getIndexOfMax();

// auto temp = new T[capacity]{};

// for (size\_t i = 0; i < index; i++)

// {

// temp[i] = data[i];

// }

// for (size\_t i = index, i2 = index + 1; i < size() - 1; i++, i2++)

// {

// temp[i] = data[i2];

// }

// if (size() != 0) {

// delete[]data;

// }

// data = temp;

// temp = nullptr;

// --count;

// --rear;

// }

// T peek()const {

// return data[front];

// }

// int size()const {

// return count;

// }

// bool isEmpty()const {

// return size() == 0;

// }

// bool isFull()const {

// return capacity == size();

// }

// void show()const {

// for (size\_t i = 0; i < size(); i++)

// {

// cout << data[i] << " ";

// }

// cout << endl << endl;;

// }

// int getIndexOfMax() {

// T max = data[0];

// int maxindex = -1;

// for (size\_t i = 0; i < size(); i++)

// {

// if (data[i] >= max) {

// max = data[i];

// maxindex = i;

// }

// }

// return maxindex;

// }

// ~PriorityQueue()

// {

// delete[]data;

// }

//};

//

//class Player

//{

// string username;

// double score;

//public:

// Player() = default;

// Player(const string& username, const double& score)

// {

// this->username = score;

// this->score = score;

// }

//

// bool operator >=(const Player& other)

// {

// return this->score >= other.score;

// }

// friend ostream& operator<< (ostream& out, const Player& player);

//

//};

//

//ostream& operator<< (ostream& out, const Player& player)

//{

// cout << "==============" << endl;

// out << "Player Info " << endl;

// out << "Username : " << player.username << endl;

// out << "Score : " << player.score << endl;

// return out;

//}

//

//void main() {

// PriorityQueue<Player> queue(5);

// queue.enqueue(Player("Jordan",78));

// queue.enqueue(Player("Michael",90));

// queue.enqueue(Player("Lebron",67));

// queue.enqueue(Player("Kyle Andreson",78));

// while (!queue.isEmpty()) {

// queue.show();

// queue.dequeue();

// Sleep(2000);

// }

//}

//// (Singly-)Linked list

//

//struct Node

//{

// int data;

// Node\* next;

//};

//

//class LinkedList

//{

//public:

// void PrintList(const Node\* n) const

// {

// cout << "Printing List" << endl;

// while (n != nullptr)

// {

// cout << n->data << endl;

// n = n->next;

// }

// }

// void PushFirst(Node\*\* head, int newdata) // \*\* or \*& // pointerin ozu

// {

// Node\* newnode = new Node;

// newnode->data = newdata;

// newnode->next = \*head;

// \*head = newnode;

// }

// void PushLast(Node\*\* head, int newdata) // \*\* or \*&

// {

// Node\* newnode = new Node;

// newnode->data = newdata;

// newnode->next = nullptr;

// Node\* last = \*head;

// while (last->next != nullptr)

// {

// last = last->next;

// }

// last->next = newnode;

// }

// void InsertAfter(Node\* prenode, int newdata)

// {

// if (prenode->next != nullptr)

// {

// Node\* newnode = new Node;

// newnode->data = newdata;

// newnode->next = prenode->next;

// prenode->next = newnode;

// }

// else

// {

// cout << "The Given Node Is Empty!" << endl;

// }

// }

//};

//

//void main()

//{

// Node\* head = new Node;

// Node\* second = new Node;

// Node\* third = new Node;

//

// head->data = 10;

// head->next = second;

//

// second->data = 20;

// second->next = third;

//

// third->data = 30;

// third->next = nullptr;

//

// LinkedList ll;

// ll.PrintList(head);

//

// ll.PushLast(&head, 55);

// ll.PrintList(head);

//

// ll.PushFirst(&head, 11);

// ll.PrintList(head);

//

// ll.InsertAfter(second, 33);

// ll.PrintList(head);

//}