#include <iostream>

#include <assert.h>

#include <initializer\_list>

#include <windows.h>

using namespace std;

//template<class T,int size>

//class FixedArray

//{

// T arr[size]{};

// size\_t mysize = 0;

//public:

// FixedArray(initializer\_list<T> list)

// {

// for (auto data : list)

// {

// arr[mysize] = data;

// mysize++;

// }

// }

//

// void Show() const

// {

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

// {

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

// }

// cout << endl;

// }

//

// T& operator[](int index)

// {

// return arr[index];

// }

//};

//

//int main()

//{

// FixedArray<int, 20> myarray{1,3,4,6,2,6,2,6};

// myarray.Show();

//

// return 0;

//}

//Dynamic data structure - datanin ramda ve ya databasede hansi formada saxlanmasi

//Data strukturlari

//Stack

//Queue

//Dequeue

//Circular

//Binary tree

//Linked list

//Double linked test

// 1) Stack - (heap, stakc deyil)

// Custom stack class

template<class T>

class Stack

{

private:

T\* data;

size\_t size = 0;

public:

Stack() : data(NULL), size(NULL){}

void push(const T& value)

{

T\* newarr = new T[size + 1]{};

for (size\_t x = 0; x < size; x++)

{

newarr[x] = data[x];

}

newarr[size] = value;

if (size != 0)

{

delete[]data;

}

data = newarr;

newarr = nullptr;

size++;

}

T pop()

{

assert(size > 0);

T\* newarr = new T[size - 1] {};

for (size\_t x = 0; x < size - 1; x++)

{

newarr[x] = data[x];

}

T last = data[size - 1];

if (size != 0)

{

delete[]data;

}

data = newarr;

newarr = nullptr;

size--;

return last;

}

// returns the last element

T& peek()

{

assert(size > 0);

return data[size - 1];

}

T& operator[](int index)

{

return data[index];

}

size\_t GetSize() const

{

return size;

}

void Clear()

{

if (this->data == NULL && this->size == NULL)

{

return;

}

delete[]data;

this->data = NULL;

this->size = NULL;

}

~Stack()

{

delete[]data;

}

};

//class User

//{

// string name;

// string surname;

//public:

// User()

// {

//

// }

// User(const string& name, const string& surname)

// {

// this->name = name;

// this->surname = surname;

// }

//

// void ShowUser() const

// {

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

// cout << "Name ; " << this->name << endl;

// cout << "Surname : " << this->surname << endl;

// }

//

//};

//

//

//int\* main()

//{

// Stack<int, 10> mystack;

// mystack.push(10);

// mystack.push(11);

// mystack.push(12);

// //int lastint = mystack.pop();

// //cout << " Last int : " << lastint << endl;

//

// Stack<char, 10> mystack2;

// mystack2.push('A');

// mystack2.push('B');

// mystack2.push('C');

// char lastchar = mystack2.pop();

// cout << " Last char : " << lastchar << endl;

//

// Stack<User, 3> mystack3;

// User u1("Tofiq", "Tofiqli");

// User u2("A", "Ali");

// User u3("John", "Johnlu");

//

// mystack3.push(u1);

// mystack3.push(u2);

// mystack3.push(u3);

// User lastuser = mystack3.pop();

// cout << "Last User : " << endl;

// lastuser.ShowUser();

//

// return 0;

//}

template<class T>

class Queue

{

T\* arr;

int capacity;

int front;

int rear;

int count;

public:

Queue(int size)

{

arr = new T[size] {};

capacity = size;

front = 0;

rear = -1;

count = 0;

}

void enqueue(T value)

{

assert(!IsFull() && "Queue is full");

arr[++rear] = value;

++count;

}

void dequeue()

{

assert(!IsEmpty() && "Queue is empty!");

auto temp = new T[count - 1]{};

for (int x = 0; x < count; x++)

{

temp[x] = arr[x + 1];

}

if (!IsEmpty())

{

delete[]arr;

}

arr = temp;

temp = nullptr;

--count;

}

bool IsFull()

{

return size() == capacity;

}

bool IsEmpty() const

{

return size() == 0;

}

T peek()

{

return arr[front];

}

int size() const

{

return count;

}

~Queue()

{

delete[]arr;

}

};

class Person

{

string name;

string surname;

double money;

bool hasVIPcard;

public:

Person() = default;

Person(const string& name, const string& surname, const double& money)

{

SetName(name);

SetSurname(surname);

SetMoney(money);

}

string GetName() const

{

return name;

}

string GetSurname() const

{

return surname;

}

double GetMoney() const

{

return money;

}

void SetName(const string& name)

{

this->name = name;

}

void SetSurname(const string& surname)

{

this->surname = surname;

}

void SetMoney(const double& money)

{

this->money = money;

}

friend ostream& operator <<(ostream& out, const Person& p)

{

out << "======= PERSON INFO ======= " << endl;

out << "Name : " << p.GetName() << endl;

out << "Surname : " << p.GetSurname() << endl;

out << "Money : " << p.GetMoney() << endl;

return out;

}

};

// void main() {

// Person p1("John", "Johnlu", 1000);

// Person p2("Mike", "Eliyev", 200);

// Person p3("Hakuna", "Matata", 100);

// Queue<Person>persons(3);

// persons.enqueue(p1);

// persons.enqueue(p2);

// persons.enqueue(p3);

// while (!persons.IsEmpty())

// {

// auto person = persons.peek();

// cout << person << endl;

// Sleep(1500);

// cout << "his work finished" << endl;

// persons.dequeue();

// }

// /\*Queue myqueue(5);

// int i = 0;

// while (!myqueue.IsFull())

// {

// myqueue.enqueue(++i);

// }

// Sleep(2000);

// while (!myqueue.IsEmpty())

// {

// cout << "Element : " << myqueue.peek() << endl;

// myqueue.dequeue();

// }\*/

//}

//class Bank

//{

// Stack<Person> people;

//public:

// void AddPerson(const Person& p)

// {

// people.push(p);

// }

// void ShowAll()

// {

// int size = people.GetSize();

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

// {

// auto person = people.peek();

// cout << person << endl;

// people.pop();

// }

//

// }

//};

//int main()

//{

// Person p1("John", "Johnlu", 1000);

// Person p2("Mike", "Eliyev", 200);

// Person p3("Hakuna", "Matata", 100);

// Bank bank;

// bank.AddPerson(p1);

// bank.AddPerson(p2);

// bank.AddPerson(p3);

// bank.ShowAll();

//

// return 0;

//}

int main()

{

return 0;

}