#pragma once

template<class T>

class Array

{

public:

Array();

explicit Array(size\_t); //explicit - запрет компилятору на неявное преобразование типа

Array(const Array<T>&);

Array<T>& operator=(const Array<T>&);

~Array();

private:

size\_t size;

size\_t capacity;

T\* data;

};

template <class T>

Array<T>::Array():size(0), capacity(0), data(nullptr) {}

template <class T>

Array<T>::Array(size\_t \_size) : size(0), capacity(0), data(new T[size]) {}

template <class T>

Array<T>::~Array()

{

delete[]data;

};

template <class T>

Array<T>::Array(const Array<T>& a) :size(a.size), capacity(a.capacity), data(new T[a.capacity])

{

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

{

data[i] = a.data[i];

}

};

template<class T>

Array<T>& Array<T>::operator=(const Array<T>& a)

{

if (this == &a) //проверка на совпадение объектов

{

return \*this;

}

delete[]data;

size = a.size;

capacity = a.capacity;

data = new T[capacity];

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

{

data[i] = a.data[i];

}

return \*this;

}

#pragma once

#include <vector>

template <class T>

class Stack

{

public:

Stack() = default;

Stack(const T&);

T getTop();

bool isEmpty();

void push\_back(const T&);

T pop\_back();

~Stack() = default;

private:

std::vector<T> data;

size\_t size;

T\* top;

};

template <class T>

Stack<T>::Stack(const T& t)

{

this->data.push\_back(t);

size++;

}

template <class T>

void Stack<T>::push\_back(const T& t)

{

this->data.push\_back(t);

size++;

this->top = &data[size-1];

}

template <class T>

T Stack<T>::pop\_back()

{

if (!this->isEmpty()) {

T last = this->data[this->size - 1];

data.pop\_back();

size--;

this->top = &data[size - 1];

return last;

}

else {

return 0;

}

}

template <class T>

T Stack<T>::getTop()

{

return \*(this->top);

}

template <class T>

bool Stack<T>::isEmpty()

{

return this->size == 0;

}

#include <iostream>

#include "StackClass.h"

int main()

{

Stack<int> stack(2);

stack.push\_back(5);

stack.push\_back(3);

stack.push\_back(7);

stack.pop\_back();

std::cout << stack.getTop();

return 0;

}

#pragma once

#include <vector>

template <class T>

class Queue

{

public:

Queue();

Queue(const T&);

bool isEmpty();

T front();

T back();

void delFirstEl(std::vector<T>& v);

void push(const T&);

T pop();

~Queue() = default;

private:

size\_t start;

size\_t end;

std::vector<T> data;

};

template <class T>

Queue<T>::Queue()

{

start = 0;

end = 0;

data = { 0 };

}

template <class T>

Queue<T>::Queue(const T& t)

{

start = 0;

end = 0;

data.push\_back(t);

}

template <class T>

void Queue<T>::push(const T& t)

{

data.push\_back(t);

end++;

}

template <class T>

bool Queue<T>::isEmpty()

{

return data.size() == 0;

}

template <class T>

T Queue<T>::pop()

{

if (this->isEmpty()) {

return NULL;

}

T first = data[0];

if (data.size() > 2) {

delFirstEl(data);

}

else {

data.pop\_back();

}

return first;

}

template <class T>

T Queue<T>::front()

{

return data[start];

}

template <class T>

T Queue<T>::back()

{

return data[end];

}

template <class T>

void Queue<T>::delFirstEl(std::vector<T> &v)

{

for (size\_t i = 1; i < v.size() - 1; i++)

{

v[i - 1] = v[i];

}

}

#include <iostream>

#include "QueueClass.h"

int main()

{

Queue<int> q(2);

q.push(5);

q.push(8);

q.push(59);

q.pop();

std::cout << q.front() << '\n' << q.back();

return 0;

}