**Ордена Трудового Красного Знамени**

Федеральное Государственное

бюджетное образовательное учреждение

высшего образования

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**СВЯЗИ И ИНФОРМАТИКИ**

Кафедра Информационной Безопасности

**ЛАБОРАТОРНАЯ РАБОТА**

**№ 6**

по дисциплине «Объектно-ориентированное программирование система защиты информации»

на тему:

**«Классы с динамическими структурами данных. Шаблонные классы»**

Вариант №16

Выполнил: студ. гр. БПЗ1501

Романюк И. Ф.

Проверил: ассисент кафедры ИБ

Барков В. В.

Москва 2018

**Цель работы**

Изучить динамические структуры данных, овладеть навыками создания конструкторов копирования, перемещения, деструкторов, перегрузки операций копирования и перемещения.

**Выполнение**

Задание

Для типа динамической структуры данных, указанного в индивидуальном задании, разработать соответствующий класс, предусмотрев в нем конструкторы инициализации, копирования, перемещения, деструктор, функции вставки и удаления элемента, просмотра доступного элемента и функцию, проверяющую наличие элементов. Перегрузить операции присваивания, присваивание с перемещением и потокового вывода для вывода содержимого динамической структуры на экран.

Исходный код:

Stack.h

#pragma once

#include "iostream"

#include "conio.h"

using namespace std;

class Stack

{

int \*arr;

int count;

int end = 0, start = 0, size = 0;

public:

Stack(int count); //инициализация

Stack(const Stack &other); //копирование

Stack(Stack &&other); //перемещение

Stack &operator=(const Stack &other); //присваивание

Stack &operator=(Stack &&other); //присваивание с перемещением

~Stack(); //деструктор

int GetSize();

void Push(int element); //Вставка (добавление) элемента

int Pop(); //Удаление (взятие) элемента

int Peek(); //Просмотр (взятие без удаления) элемента

bool CheckNoEmptyStack(); //Проверка наличия элементов

friend ostream & operator << (ostream & stream, const Stack &a); //Вывод

};

Stack.cpp

#include "Stack.h"

Stack::Stack(int count)

: count(count)

{

arr = new int[count];

}

Stack::~Stack()

{

delete[] arr;

}

Stack::Stack(const Stack &other)

{

arr = new int[other.count];

count = other.count;

end = other.end;

start = other.start;

size = other.size;

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

{

arr[i] = other.arr[i];

}

}

Stack &Stack::operator=(const Stack &other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = new int[other.count];

count = other.count;

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

{

arr[i] = other.arr[i];

}

}

Stack &Stack::operator=(Stack &&other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

Stack::Stack(Stack &&other)

{

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

int Stack::GetSize()

{

return this->size;

}

void Stack::Push(int element)

{

if (size < count)

{

arr[end] = element;

end = (end + 1) % count;

size++;

}

}

int Stack::Pop()

{

if (size > 0)

{

int element = arr[start];

start = (start + 1) % count;

size--;

return element;

}

return 0;

}

int Stack::Peek()

{

return arr[start];

}

bool Stack::CheckNoEmptyStack()

{

return (size > 0);

}

ostream & operator << (ostream & stream, const Stack &a)

{

Stack b = a;

stream << "=====\n";

for (int i = 0; i < b.count; i++)

{

stream << "| " << b.Pop() << "\n";

}

stream << "=====\n";

return stream;

}

StackTemplate.h

#pragma once

#include "iostream"

#include "conio.h"

using namespace std;

template<typename T>

class StackTemplate

{

T \*arr;

int count;

int end = 0, start = 0, size = 0;

public:

StackTemplate(int count); //инициализация

StackTemplate(const StackTemplate &other); //копирование

StackTemplate(StackTemplate &&other); //перемещение

StackTemplate &operator=(const StackTemplate &other); //присваивание

StackTemplate &operator=(StackTemplate &&other); //присваивание с еремещением

~StackTemplate(); //деструктор

int GetSize();

void Push(T element); //Вставка (добавление) элемента

T Pop(); //Удаление (взятие) элемента

T Peek(); //Просмотр (взятие без удаления) элемента

bool CheckNoEmptyStack(); //Проверка наличия элементов

friend ostream & operator << <T>(ostream & stream, const StackTemplate<T> &a); //Вывод

};

template<typename T>

StackTemplate<T>::StackTemplate(int count)

: count(count)

{

arr = new T[count];

}

template<typename T>

StackTemplate<T>::~StackTemplate()

{

delete[] arr;

}

template<typename T>

StackTemplate<T>::StackTemplate(const StackTemplate &other)

{

arr = new T[other.count];

count = other.count;

end = other.end;

start = other.start;

size = other.size;

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

{

arr[i] = other.arr[i];

}

}

template<typename T>

StackTemplate<T> &StackTemplate<T>::operator=(const StackTemplate &other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = new T[other.count];

count = other.count;

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

{

arr[i] = other.arr[i];

}

}

template<typename T>

StackTemplate<T> &StackTemplate<T>::operator=(StackTemplate &&other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

template<typename T>

StackTemplate<T>::StackTemplate(StackTemplate &&other)

{

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

template<typename T>

int StackTemplate<T>::GetSize()

{

return this->size;

}

template<typename T>

void StackTemplate<T>::Push(T element)

{

if (size < count)

{

arr[end] = element;

end = (end + 1) % count;

size++;

}

}

template<typename T>

T StackTemplate<T>::Pop()

{

if (size > 0)

{

T element = arr[start];

start = (start + 1) % count;

size--;

return element;

}

return 0;

}

template<typename T>

T StackTemplate<T>::Peek()

{

return arr[start];

}

template<typename T>

bool StackTemplate<T>::CheckNoEmptyStack()

{

return (size > 0);

}

template<typename T>

ostream & operator << (ostream & stream, const StackTemplate<T> &a)

{

StackTemplate<T> b = a;

stream << "=====\n";

for (int i = 0; i < b.count; i++)

{

stream << "| " << b.Pop() << "\n";

}

stream << "=====\n";

return stream;

}