**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» ИНЖЕНЕРНО- ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчет по лабораторной работе №1**

по курсу «Объектно ориентированное программирование»

**«Классы и объекты в С++»**

Выполнил: студент гр. КТбо2-7

Измайлов В.Н.

Таганрог 2020

**Оглавление**

[Техническое задание 3](#_TOC_250004)

[Выполнение задания 3](#_TOC_250003)

1. [Спецификация классов 3](#_TOC_250002)
2. [Диаграмма классов 4](#_TOC_250001)
3. [Листинг 5](#_TOC_250000)

**Вариант № 15**

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Реализовать класс Стек (Stack для какого-либо типа данных с методами push, pop, isEmpty и back (показывается последний элемент без его извлечения), работающими согласно соответствующей дисциплине обслуживания. Размер стека задается при его создании задается пользователем.

# ВЫПОЛНЕНИЕ ЗАДАНИЯ

# Спецификация классов.

Класс StackMassive в private содержит поля sizeOfstack(размер стека), stackMemory(размер памяти выделенной под стек) и строковый массив array в котором будут храниться элементы нашего стека. Класс StackException предназначен для определения типа выдаваемых исключения. StackConsole класс реализации пользовательского интерфейса.

# Изображение выглядит как текст Автоматически созданное описание

# 2. Диаграмма классов.

# Листинг

1. Main.cpp

#include <ctime>

#include "ConsoleIteractor.h"

#include "Test.h"

#include <vld.h>

#include <cstdio>

int main()

{

double start = clock();

ConsoleIteractor console;

console.InputAction();

unsigned int end\_time = clock();

printf("%.4lf\n", (clock() - start) / CLOCKS\_PER\_SEC);

}

1. Queue.h

#pragma once

#include <string>

#include <sstream>

#include <algorithm>

#include "Container.h"

class Queue : public Container {

public:

Queue(int sizeQueue) {

\_queue = new std::string[sizeQueue];

\_queueMemory = sizeQueue;

}

Queue(const Queue& object) : Queue(object.\_queueMemory) {

if (object.\_queue != nullptr) {

this->\_queue = new std::string[object.\_queueMemory];

this->\_queueMemory = object.\_queueMemory;

this->\_queueSize = object.\_queueSize;

std::copy(object.\_queue, object.\_queue + object.\_queueMemory, \_queue);

}

}

Queue& operator = (const Queue& object) {

if (&object == this) {

return \*this;

}

if (\_queue != nullptr) {

delete[] \_queue;

}

if (object.\_queue != nullptr) {

\_queue = new std::string[object.\_queueMemory];

this->\_queueMemory = object.\_queueMemory;

this->\_queueSize = object.\_queueSize;

std::copy(object.\_queue, object.\_queue + object.\_queueMemory, \_queue);

}

return \*this;

}

void MultiPush(const int count, std::string \*elements) override;

void Push(const std::string element) override;

void MultiPop(const int count) override;

Container\* Clone() const override;

void Pop() override;

std::string ToString() const override;

std::string Front() const;

std::string Back() const override;

std::string GetType() const override {

return \_type;

}

bool Empty() const override;

~Queue() {

delete[] \_queue;

}

private:

std::string \_type = "Queue";

std::string\* \_queue;

int \_queueSize = 0;

int \_queueMemory;

};

1. Queue.cpp

#include "Queue.h"

#include <iostream>

#include "ContainerException.h"

void Queue::Push (const std::string element) {

if (\_queueSize < \_queueMemory) {

\_queue[\_queueSize] = element;

\_queueSize++;

}

else {

throw ContainerException("Переполнение");

}

}

void Queue::MultiPush(const int count, std::string \*elements) {

if (count + \_queueSize <= \_queueMemory) {

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

Push(elements[i]);

}

delete[]elements;

}

else {

throw ContainerException("Невозможно добавить такое количество элементов");

}

}

void Queue::MultiPop(const int count) {

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

if (\_queueSize) {

Pop();

}

else {

throw ContainerException("Все элементы удалены");

}

}

}

Container\* Queue::Clone() const

{

return new Queue(\*this);

}

std::string Queue::ToString() const {

std::ostringstream out;

std::string\* array = \_queue;

for (int begin = 0; begin < \_queueSize; begin++) {

out << "[" << array[begin] << "] ";

}

return out.str();

}

void Queue::Pop() {

for (int i = 1; i < \_queueSize; i++) {

\_queue[i - 1] = \_queue[i];

}

\_queue[\_queueSize - 1] = "";

\_queueSize--;

}

std::string Queue::Front() const {

if (\_queueSize != 0) {

return \_queue[0];

}

throw ContainerException("Стек пуст");

}

std::string Queue::Back() const {

if (\_queueSize != 0) {

return \_queue[\_queueSize - 1];

}

throw ContainerException("Стек пуст");

}

bool Queue::Empty() const {

return \_queueSize == 0;

}}

1. ConsoleIteractor.h

s#pragma once

#include "Container.h"

#include "Factory.h"

class ConsoleIteractor {

public:

Container\* CreateContainer(Container \*container, Factory factory);

void PrintAllContainers(Container\*\* container);

void InputAction();

void Actions();

void TestForConstructors();

void TestForOperators();

int InputSize();

private:

int \_indexOfContainer = 0;

int \_containerSize = 0;

static const int \_CONTAINER\_SIZE = 100;

};

1. Factory.h

#pragma once

#include "StackList.h"

#include "StackMassive.h"

#include "Queue.h"

class Factory {

public:

StackList\* CreateStackList() const {

return new StackList();

}

StackMassive\* CreateStackMassive(int size) const {

return new StackMassive(size);

}

Queue\* CreateQueue(int size) const {

return new Queue(size);

}

};

1. Container.h

#pragma once

#include <string>

class Container {

public:

virtual void MultiPush(const int count, std::string \*elements) = 0;

virtual void Push(const std::string new\_element) = 0;

virtual void MultiPop(const int count) = 0;

virtual std::string ToString() const = 0;

virtual std::string Back() const = 0;

virtual Container\* Clone() const = 0;

virtual std::string GetType() const = 0;

virtual bool Empty() const = 0;

virtual void Pop() = 0;

virtual ~Container() = default;

};

1. StackList.h

#pragma once

#include "Container.h"

#include <string>

struct Node {

std::string element;

Node\* next;

Node\* prev;

Node(std::string new\_el, Node\* nextNode) {

element = new\_el;

next = nextNode;

if (nextNode) {

nextNode->prev = this;

}

}

};

class StackList : public Container {

public:

StackList() {

\_head = NULL;

}

StackList(const StackList& object) : StackList() {

if (object.\_head != nullptr) {

Node\* tmp1 = object.\_head;

Node\* tmp2 = \_head;

while (tmp1->next != nullptr) {

tmp1 = tmp1->next;

}

while (tmp1 != object.\_head) {

this->Push(tmp1->element);

tmp1 = tmp1->prev;

}

this->Push(tmp1->element);

}

}

StackList& operator = (const StackList& object) {

if (&object == this) {

return \*this;

}

if (\_head != nullptr) {

while(\_sizeOfStack) {

Pop();

}

}

if (object.\_head != nullptr) {

Node\* tmp1 = object.\_head;

Node\* tmp2 = \_head;

while (tmp1->next != nullptr) {

tmp1 = tmp1->next;

}

while (tmp1 != object.\_head) {

this->Push(tmp1->element);

tmp1 = tmp1->prev;

}

this->Push(tmp1->element);

}

return \*this;

}

Node\* GetHead() const {

return \_head;

}

void Push(const std::string new\_element) override;

void MultiPush(const int count, std::string \*elements) override;

void MultiPop(const int count) override;

void Pop() override;

std::string ToString() const override;

bool Empty() const override;

std::string Back() const override;

Container\* Clone() const override;

std::string GetType() const override {

return \_type;

}

void SetHead(Node\* head) {

this->\_head = head;

}

int Size() const {

return \_sizeOfStack;

}

~StackList() {

while (Size())

Pop();

}

private:

std::string \_type = "StackList";

int \_sizeOfStack = 0;

Node\* \_head;

};

1. StackList.cpp

#include "StackList.h"

#include "ContainerException.h"

#include <iostream>

#include <sstream>

void StackList::Push(const std::string new\_element) { //O(1)

Node\* node = new Node(new\_element, \_head);

\_sizeOfStack++;

\_head = node;

}

void StackList::MultiPush(const int count, std::string\* elements) {

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

Push(elements[i]);

}

delete[]elements;

}

void StackList::Pop() {

if (\_sizeOfStack) {//O(1)

\_sizeOfStack--;

Node\* node = \_head;

\_head = \_head->next;

delete node;

}

else {

throw ContainerException("Стек пуст");

}

}

std::string StackList::ToString() const

{

std::ostringstream out;

Node\* var = \_head;

while (var->next != nullptr) {

out << "[" << var->element << "] ";

var = var->next;

}

out << "[" << var->element << "] \n";

return out.str();

}

void StackList::MultiPop(const int count) {

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

if (\_sizeOfStack) {

Pop();

}

else {

throw ContainerException("Все элементы удалены");

}

}

}

bool StackList::Empty() const{ //O(1)

return \_head == NULL;

}

std::string StackList::Back() const{

if (\_sizeOfStack > 0) {

Node\* node = \_head;

while (node->next != nullptr) {

node = node->next;

}

return node->element;

}

else {

throw ContainerException("Последний элемент не найден");

}

}

Container\* StackList::Clone() const

{

return new StackList(\*this);

}

1. StackMassive.h

#pragma once

#include "Container.h"

#include <sstream>

#include <string>

class StackMassive : public Container {

public:

StackMassive(int massiveSize) {

\_sizeOfStack = 0;

\_stackMemory = massiveSize;

this->\_array = new std::string[massiveSize];

}

StackMassive(const StackMassive& object) : StackMassive(object.\_stackMemory) {

if (object.\_array != nullptr) {

this->\_array = new std::string[object.\_stackMemory];

this->\_stackMemory = object.\_stackMemory;

this->\_sizeOfStack = object.\_sizeOfStack;

}

}

StackMassive& operator = (const StackMassive& object) {

if (&object == this) {

return \*this;

}

if (\_array != nullptr) {

delete[] \_array;

}

if (object.\_array != nullptr) {

\_array = new std::string[\_stackMemory];

std::copy(object.\_array, object.\_array + object.\_stackMemory, \_array);

}

return \*this;

}

void Push( const std::string new\_element) override;

void MultiPush(const int count, std::string \*elements) override;

void Pop() override;

void MultiPop(const int count) override;

bool Empty() const override;

std::string ToString() const override;

std::string Back() const override;

Container\* Clone() const override;

std::string GetType() const override {

return \_type;

}

int GetSize() const {

return \_sizeOfStack;

}

std::string\* GetArray() const {

return \_array;

}

int GetSizeArray() {

return \_stackMemory;

}

~StackMassive() {

delete[] \_array;

}

private:

std::string \_type = "StackMassive";

int \_sizeOfStack = 0;

int \_stackMemory;

std::string\* \_array;

};

1. StackMassive.cpp

#include "StackMassive.h"

#include "ContainerException.h"

#include <iostream>

void StackMassive::Push(const std::string new\_element) { // O(1)

if (\_sizeOfStack < \_stackMemory) {

\_array[\_sizeOfStack] = new\_element;

\_sizeOfStack++;

}

else {

throw ContainerException("Выход за пределы массива");

}

}

void StackMassive::MultiPush(const int count, std::string \*elements) {

if (count + \_sizeOfStack <= \_stackMemory) {

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

Push(elements[i]);

}

delete[]elements;

}

else {

throw ContainerException("Невозможно добавить такое количество элементов");

}

}

void StackMassive::Pop() { // O(1)

if (\_sizeOfStack > 0) {

\_sizeOfStack--;

}

else {

throw ContainerException("Стек пуст");

}

}

void StackMassive::MultiPop(const int count) {

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

if (\_sizeOfStack) {

Pop();

}

else {

throw ContainerException("Все элементы удалены");

}

}

}

bool StackMassive::Empty() const { // O(1)

return \_sizeOfStack == 0;

}

std::string StackMassive::ToString() const

{

std::ostringstream out;

std::string\* array = StackMassive::\_array;

for (int begin = 0; begin < \_sizeOfStack; begin++) {

out << "[" << array[begin] << "] ";

}

return out.str();

}

std::string StackMassive::Back() const { // O(1)

if (\_sizeOfStack > 0) {

return \_array[0];

}

else {

throw ContainerException("Последний элемент не найден");

}

}

Container\* StackMassive::Clone() const

{

return new StackMassive(\*this);

}

1. StackException.h

#pragma once

#include <string>

class ContainerException {

public:

ContainerException(std::string error) {

\_error = error;

}

std::string GetError() {

return \_error;

}

private:

std::string \_error;

};