**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.

# 

# 2. Диаграмма классов.

# Листинг

1. Main.cpp

#include <ctime>

#include "Containers.h"

#include <vld.h>

#include <cstdio>

int main()

{

double start = clock();

Containers containers;

containers.ChooseContainer();

unsigned int end\_time = clock();

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

}

1. ContainersFactory.h

#pragma once

#include "IConsole.h"

#include "QueueConsole.h"

#include "StackConsole.h"

class ContainersFactory {

public:

void ChooseContainer();

QueueConsole\* CreateQueueContainer() {

return new QueueConsole();

}

StackConsole\* CreateStackContainer() {

return new StackConsole();

}

};

1. Containers.cpp

#include "Containers.h"

#include <iostream>

using namespace std;

void Containers::ChooseContainer()

{

setlocale(LC\_ALL, "Russian");

IConsole\* console = nullptr;

int container;

cout << "1. Queue -> Создать очередь" << endl;

cout << "2. Stack -> Создать стек" << endl;

cin >> container;

switch (container) {

case 1:

{

console = CreateQueueContainer();

break;

}

case 2:

{

console = CreateStackContainer();

break;

}

}

console->Actions();

console->InputAction();

delete console;

}

1. Queue.h

#pragma once

#include <string>

#include <sstream>

#include "Container.h"

class Queue : public Container {

public:

Queue(int sizeQueue) {

queue = new std::string[sizeQueue];

queueMemory = sizeQueue;

}

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

void push(const std::string& element) override;

void MultiPop(const int &count) override;

void pop() override;

std::string ToString() const override;

std::string Front() const;

std::string back() const override;

bool isEmpty() const override;

~Queue() {

delete[] queue;

}

private:

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++;

}

}

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");

}

}

}

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

std::ostringstream out;

std::string\* array = Queue::queue;

out << "\n";

for (int begin = 0; begin < queueSize; begin++) {

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

}

out << "\n";

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. QueueConsole.h

#pragma once

#include "IConsole.h"

class QueueConsole : public IConsole{

public:

virtual void Actions() override;

virtual void InputAction() override;

int InputSize();

};

1. QueueConsole.cpp

#include <iostream>

#include <iterator>

#include <string>

#include "Queue.h"

#include "QueueConsole.h"

#include "ContainerException.h"

using namespace std;

void QueueConsole::Actions() {

setlocale(LC\_ALL, "Russian");

cout << "1. push -> Добавление элемента в конец стека" << endl;

cout << "2. multi\_push -> Добавление нескольких элементов в конец стека" << endl;

cout << "3. pop -> Удаление первого элемента стека" << endl;

cout << "4. multi\_pop -> Добавление элемента в конец стека" << endl;

cout << "5. Empty -> Проверка стека на наличие элементов" << endl;

cout << "6. printQueue -> Вывод стека" << endl;

cout << "7. back -> Получение последнего элемента стека" << endl;

cout << "8. front -> Первый элемент" << endl;

cout << "9. cls -> Очистить командное окно" << endl;

cout << "10. help -> Список команд" << endl;

cout << "0. exit -> Выход из программы" << endl;

}

int QueueConsole::InputSize() {

int size;

cout << "Размер очереди: ";

cin >> size;

cout << endl;

return size;

}

void QueueConsole::InputAction() {

setlocale(LC\_ALL, "Russian");

int count;

int action = -1;

string element;

string type = "";

Queue queue(InputSize());

while (action != 0) {

try {

cout << endl << "Введите команду: ";

cin >> action;

switch (action) {

case 1:

{

cout << "Введите элемент : ";

cin >> element;

cout << endl;

queue.Push(element);

break;

}

case 2:

{

cout << "Введите количество элементов: ";

cin >> count;

std::string\* elements = new std::string[count];

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

cin >> elements[i];

}

queue.MultiPush(count, elements);

break;

}

case 3:

{

cout << endl;

queue.Pop();

cout << endl;

break;

}

case 4:

{

cout << "Введите количество удаляемых элементов: ";

cin >> count;

queue.MultiPop(count);

break;

}

case 5:

{

if (queue.Empty()) {

cout << endl << "Стек пуст" << endl << endl;

}

else {

cout << endl << "Стек не пуст" << endl << endl;

}

break;

}

case 6:

{

if (!queue.Empty()) {

cout << queue.ToString() << endl;

}

else {

cout << "Стек пуст" << endl;

}

break;

}

case 7:

{

cout << endl << "[" << queue.Back() << "]" << endl << endl;

break;

}

case 8:

{

cout << endl << "[" << queue.Front() << "]" << endl << endl;

break;

}

case 9:

{

system("cls");

break;

}

case 10:

{

Actions();

break;

}

case 0:

{

break;

}

default:

{

cout << "Неверная команда, попробуйте еще раз" << endl;

}

}

}

catch (ContainerException& ex) {

cout << ex.GetError() << endl;

}

}

}

1. Factory.h

#pragma once

#include "StackList.h"

#include "StackMassive.h"

class Factory {

public:

StackList\* CreateStackList() const {

return new StackList();

};

StackMassive\* CreateStackMassive(int size) const {

return new StackMassive(size);

};

};

1. Stack.h

#pragma once

#include <string>

class Stack {

public:

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

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

virtual void MultiPop(int& count) = 0;

virtual void pop() = 0;

virtual bool isEmpty() = 0;

virtual std::string back() = 0;

virtual ~Stack() = default;

};

1. StackList.h

#pragma once

#include "Stack.h"

#include <string>

struct Node {

std::string element;

Node\* next;

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

element = new\_el;

next = nextNode;

}

};

class StackList : public Stack {

public:

StackList() {

head = NULL;

}

Node\* getHead() {

return head;

}

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

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

void MultiPop(int& count) override;

void pop() override;

bool isEmpty() override;

std::string back() override;

void setHead(Node\* head) {

this->head = head;

}

int size() {

return sizeOfStack;

}

~StackList() {

while (size())

pop();

}

private:

int sizeOfStack = 0;

Node\* head;

};

1. StackList.cpp

#include "StackList.h"

#include "StackConsole.h"

#include "StackException.h"

#include <iostream>

using namespace std;

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

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

sizeOfStack++;

head = node;

}

void StackList::MultiPush(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 StackException("Стек пуст");

}

}

void StackList::MultiPop(int& count) {

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

if (sizeOfStack) {

pop();

}

else {

throw StackException("Все элементы удалены");

}

}

}

bool StackList::isEmpty() { //O(1)

if (head) {

return true;

}

else {

return false;

}

}

std::string StackList::back() {

if (sizeOfStack > 0) {

Node\* node = head;

while (node->next != nullptr) {

node = node->next;

}

return node->element;

}

else {

throw StackException("Последний элемент не найден");

}

}

1. StackMassive.h

#pragma once

#include "Container.h"

#include <string>

class StackMassive : public Container {

public:

StackMassive(int massiveSize) {

sizeOfStack = 0;

stackMemory = massiveSize;

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

}

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

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

void pop() override;

void MultiPop(int& count) override;

bool isEmpty() override;

std::string back() override;

int getSize() {

return sizeOfStack;

}

std::string\* getArray() {

return array;

}

int GetSizeArray() {

return stackMemory;

}

~StackMassive() {

delete[] array;

}

private:

int sizeOfStack;

int stackMemory;

std::string\* array;

};

1. StackMassive.cpp

#include "StackMassive.h"

#include "StackException.h"

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

if (sizeOfStack < stackMemory) {

array[sizeOfStack] = new\_element;

sizeOfStack++;

}

else {

throw StackException("Выход за пределы массива");

}

}

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

if (count + sizeOfStack <= stackMemory) {

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

push(elements[i]);

}

delete[]elements;

}

else {

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

}

}

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

if (sizeOfStack > 0) {

sizeOfStack--;

}

else {

throw StackException("Стек пуст");

}

}

void StackMassive::MultiPop(int &count) {

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

if (sizeOfStack) {

pop();

}

else {

throw StackException("Все элементы удалены");

}

}

}

bool StackMassive::isEmpty() { // O(1)

if (sizeOfStack != 0) {

return true;

}

else {

return false;

}

}

std::string StackMassive::back() { // O(1)

if (sizeOfStack > 0) {

return array[0];

}

else {

throw StackException("Последний элемент не найден");

}

}

1. StackConsole.h

#pragma once

#include "StackList.h"

#include "StackMassive.h"

class StackConsole {

public:

void Actions();

void InputAction();

void PrintStackList(StackList \*stack);

void PrintStackMassive(StackMassive \*stack);

int InputSize();

};

1. StackConsole.cpp

#include "StackConsole.h"

#include "StackList.h"

#include "StackException.h"

#include <iostream>

#include <string>

using namespace std;

void StackConsole::Actions() {

setlocale(LC\_ALL, "Russian");

cout << "push -> Добавление элемента в конец стека" << endl;

cout << "multi\_push -> Добавление нескольких элементов в конец стека" << endl;

cout << "pop -> Удаление первого элемента стека" << endl;

cout << "multi\_pop -> Добавление элемента в конец стека" << endl;

cout << "isEmpty -> Проверка стека на наличие элементов" << endl;

cout << "back -> Получение последнего элемента стека" << endl;

}

//

void StackConsole::PrintStackList(StackList \*stack) {

Node\* var = stack->getHead();

cout << endl;

while (var->next != nullptr) {

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

var = var->next;

}

cout << "[" << var->element << "] " << endl << endl;

}

void StackConsole::PrintStackMassive(StackMassive \*stack) {

std::string \*array = stack->getArray();

cout << endl;

for (int begin = 0; begin < stack->getSize(); begin++) {

cout << "[" << array[begin] << "] ";

}

cout << endl;

}

int StackConsole::InputSize() {

int size;

cout << "Размер стека: ";

cin >> size;

cout << endl;

return size;

}

void StackConsole::InputAction() {

setlocale(LC\_ALL, "Russian");

int count;

string action = "";

string element;

string type = "";

Container\* stack;

cout << endl << "List -> Стек на основе списка" << endl;

cout << "Massive -> Стек на основе массива" << endl;

cin >> type;

if (type == "List") {

stack = factory.CreateStackList();

}

else {

stack = factory.CreateStackMassive(InputSize());

}

while (action != "exit") {

try {

cout << endl << "Введите команду: ";

cin >> action;

if (action == "push") {

cout << "Введите элемент : ";

cin >> element;

cout << endl;

stack->push(element);

}

else if (action == "multi\_push") {

cout << "Введите количество элементов: ";

cin >> count;

std::string\* elements = new std::string[count];

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

cin >> elements[i];

}

stack->MultiPush(count, elements);

}

else if (action == "pop") {

cout << endl;

stack->pop();

cout << endl;

}

else if (action == "multi\_pop") {

cout << "Введите количество удаляемых элементов: ";

cin >> count;

stack->MultiPop(count);

}

else if (action == "isEmpty") {

if (stack->isEmpty()) {

cout << endl << "Стек не пуст" << endl << endl;

}

else {

cout << endl << "Стек пуст" << endl << endl;

}

}

else if (action == "printStack") {

if (stack->isEmpty()) {

if (type == "List") {

PrintStackList(static\_cast<StackList\*>(stack));

}

else {

PrintStackMassive(static\_cast<StackMassive\*>(stack));

}

}

else {

cout << "Стек пуст" << endl;

}

}

else if (action == "back") {

cout << endl << "[" << stack->back() << "]" << endl << endl;

}

}

catch (StackException& ex) {

cout << ex.GetError() << endl;

}

}

delete stack;

}

1. StackException.h

#pragma once

#include <string>

class StackException {

public:

StackException(std::string error) {

\_error = error;

}

std::string GetError() {

return \_error;

}

private:

std::string \_error;

};

1. IConsole.h

#pragma once

class IConsole {

public:

virtual void Actions() = 0;

virtual void InputAction() = 0;

};