МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ЛАБОРАТОРНАЯ №3**

**«**Объектно-ориентированное программирование**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБ-320», «АВТФ»  *Мартынова Е.Д.*  «8» декабря 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Ассистент кафедры ЗИ*  *Исаев Г.А.*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

Оглавление

[Цели и задачи работы 4](#_Toc184621780)

[Задание к работе: 4](#_Toc184621781)

[Методика выполнения работы: 4](#_Toc184621782)

[Структура программы 5](#_Toc184621783)

[Язык С++ 5](#_Toc184621784)

[Функции классов: 5](#_Toc184621785)

[Array.h 5](#_Toc184621786)

[AVLtree.h 5](#_Toc184621787)

[HashTable.h 5](#_Toc184621788)

[Header.h 5](#_Toc184621789)

[OneList.h 5](#_Toc184621790)

[Queue.h 5](#_Toc184621791)

[Stack.h 5](#_Toc184621792)

[TwoList.h 5](#_Toc184621793)

[Функции тестирования классов: 5](#_Toc184621794)

[Test\_Array.cpp 5](#_Toc184621795)

[Test\_AVL.cpp 5](#_Toc184621796)

[Test\_HashTable.cpp 5](#_Toc184621797)

[Test\_Header.cpp 5](#_Toc184621798)

[Test\_OneList.cpp 5](#_Toc184621799)

[Test\_TwoList.cpp 5](#_Toc184621800)

[Test\_Stack.cpp 5](#_Toc184621801)

[Test\_Queue.cpp 5](#_Toc184621802)

[Test\_Main.cpp 5](#_Toc184621803)

[Функции бенчмарка для классов: 5](#_Toc184621804)

[Benchmark\_Array.cpp 5](#_Toc184621805)

[Benchmark\_AVL.cpp 5](#_Toc184621806)

[Benchmark\_ HashTable.cpp 5](#_Toc184621807)

[Benchmark\_ Header.cpp 5](#_Toc184621808)

[Benchmark\_ OneList.cpp 5](#_Toc184621809)

[Benchmark\_ TwoList.cpp 5](#_Toc184621810)

[Benchmark\_ Stack.cpp 5](#_Toc184621811)

[Benchmark\_ Queue.cpp 6](#_Toc184621812)

[Benchmark\_ Main.cpp 6](#_Toc184621813)

[Язык GO 6](#_Toc184621814)

[Функции класса Queue: 6](#_Toc184621815)

[main.go 6](#_Toc184621816)

[Функции тестирования и бенчмарка: 6](#_Toc184621817)

[main\_test.go 6](#_Toc184621818)

[Библиотека Testify: 6](#_Toc184621819)

[go.mod 6](#_Toc184621820)

[go.sum 6](#_Toc184621821)

[Примеры работы программы 6](#_Toc184621822)

[UML-диаграммы 8](#_Toc184621823)

[Основные виды тестирования 11](#_Toc184621824)

[Список литературы 16](#_Toc184621825)

Цели и задачи работы: изучение основных принципов объектно- ориентированного программирования и основ юнит-тестирования.

Задание к работе: Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

# Методика выполнения работы:

1. Разработать алгоритмы решения задачи по индивидуальному заданию.
2. Написать и отладить программы решения задачи (С++, Go или Rust).
3. Протестировать работу программ на различных исходных данных.
4. По запросу преподавателя быть готовым модифицировать/добавить алгоритмы/блоки кода в контексте ООП.
5. Ответить на теоретические вопросы к лабораторной работе на выбор преподавателя (не менее двух вопросов).

# Структура программы

## Язык С++

## Функции классов:

### Array.h

#pragma once

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

class Array {

private:

string\* data;

int size;

int maxSize;

public:

Array() : size(0), maxSize(1000) {

data = new string[maxSize];

}

~Array() {

delete[] data;

}

void MPUSHend(string& value) {

data[size++] = value + " ";

}

void MPUSHindex(const int& index, string& value) {

for (int i = size; i > index; i--) {

data[i] = data[i - 1];

}

data[index] = value + " ";

size++;

}

string MGETL(const int& index) {

if (index < 0 || index >= size) {

//cout << endl << "INCORRECT INPUT!!!" << endl;

return "ERROR";

}

return data[index];

}

void MDEL(const int& index) {

if (index < 0 || index >= size) {

//cout << endl << "INCORRECT INPUT!!!" << endl;

return;

}

for (int i = index; i < size - 1; i++) {

data[i] = data[i + 1];

}

size--;

}

void MREPL(const int& index, string& value) {

if (index < 0 || index >= size) {

//cout << endl << "INCORRECT INPUT!!!" << endl;

return;

}

data[index] = " " + value + " ";

}

int MSIZE() {

return size;

}

void MREAD() {

cout << endl;

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

cout << data[i];

}

cout << endl;

}

void SaveToFile(string &fileName) {

ofstream FileWrite(fileName);

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

FileWrite << MGETL(i);

}

FileWrite.close();

}

void WritingFromFileToStructure(string& filename) {

ifstream FileRead(filename);

string tempValue;

while (getline(FileRead, tempValue)) {

stringstream ss(tempValue);

string element;

while (ss >> element) {

MPUSHend(element);

}

}

FileRead.close();

}

void BinarySerialization(string& fileName) {

ofstream File(fileName, ios::binary);

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

string value = MGETL(i);

int valueLen = value.size();

File.write(reinterpret\_cast<char\*>(&valueLen), sizeof(valueLen));

File.write(value.c\_str(), valueLen);

}

File.close();

}

void BinaryDESerialization(string& fileName) {

ifstream File(fileName, ios::binary);

while (File) {

int len;

File.read(reinterpret\_cast<char\*>(&len), sizeof(len));

if (File.eof())

break;

char\* buffer = new char[len + 1];

File.read(buffer, len);

buffer[len] = '\0';

string text(buffer);

data[size++] = text;

delete buffer;

}

File.close();

}

};

### AVLtree.h

#pragma once

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

class AVL {

private:

struct AVLnode {

string key;

AVLnode\* left;

AVLnode\* right;

int height;

AVLnode(string key) : key(key), left(nullptr), right(nullptr), height(1) {};

};

struct SingleNode {

string cell;

SingleNode\* next;

};

AVLnode\* root;

string key;

unsigned char height;

SingleNode\* left;

SingleNode\* right;

public:

AVL() : root(nullptr) {};

~AVL() {

TDestroyAVL(root);

}

void TDestroyAVL(AVLnode\* node) {

if (node == nullptr) {

return;

}

TDestroyAVL(node->left);

TDestroyAVL(node->right);

delete node;

}

AVLnode\* GetRoot() {

return root;

}

int FindSize(AVLnode\* node) {

if (node == nullptr) {

return 0;

}

return 1 + FindSize(node->left) + FindSize(node->right);

}

int Size() {

return FindSize(root);

}

int THeight(AVLnode\* node) {

if (node == nullptr) {

return 0;

}

return node->height;

}

int TBalance(AVLnode\* node) {

if (node == nullptr) return 0;

return THeight(node->left) - THeight(node->right);

}

AVLnode\* TRightRotate(AVLnode\* node) {

AVLnode\* A = node->left; // левый потомок узла становится А

AVLnode\* B = A->right; // правый потомок узла А становится В

A->right = node; // А становится корнем, ноде становится правым потомком

node->left = B; // левый потомок ноде становится В

node->height = max(THeight(node->left), THeight(node->right)) + 1; // ноде и А становятся двумя разными поддеревьями

A->height = max(THeight(A->left), THeight(A->right)) + 1;

return A; // корень

}

AVLnode\* TLeftRotate(AVLnode\* node) { //симметрия

AVLnode\* A = node->right;

AVLnode\* B = A->left;

A->left = node;

node->right = B;

node->height = max(THeight(node->left), THeight(node->right)) + 1;

A->height = max(THeight(A->left), THeight(A->right)) + 1;

return A;

}

AVLnode\* TInsert(AVLnode\* node, string& key) {

// если текущий узел пустой, создаем новый узел с переданным ключом

if (node == nullptr) {

return new AVLnode(key);

}

//рекурсивно вставляем в левое поддерево

if (key < node->key) {

node->left = TInsert(node->left, key);

}

// рекурсивно вставляем в правое поддерево

else if (key > node->key) {

node->right = TInsert(node->right, key);

}

// если ключ уже существует, возвращаем текущий узел

else {

return node; //(без дубликатов)

}

node->height = 1 + max(THeight(node->left), THeight(node->right)); // обновляем высоту текущего узла

// вычисляем баланстекущего узла

int balance = TBalance(node);

// левый-левый

if (balance > 1 && key < node->left->key)

return TRightRotate(node);

// левый-правый

if (balance > 1 && key > node->left->key) {

node->left = TLeftRotate(node->left); //меняет местами

return TRightRotate(node);

}

// правый-правый

if (balance < -1 && key > node->right->key)

return TLeftRotate(node);

// правый-левый

if (balance < -1 && key < node->right->key) {

node->right = TRightRotate(node->right); // меняет местами

return TLeftRotate(node);

}

return node; // возвращаем корень поддерева

}

AVLnode\* TMinValueLeftNode(AVLnode\* node) {

AVLnode\* current = node;

while (current->left != nullptr) // идем по левому поддереву до самого левого узла

current = current->left;

return current;

}

AVLnode\* TDeleteNode(AVLnode\* root, string& key) {

if (root == nullptr) {

return root; // если дерево пустое

}

// рекурсивно удаляем в левом поддереве

if (key < root->key)

root->left = TDeleteNode(root->left, key);

// рекурсивно удаляем в правом поддереве

else if (key > root->key)

root->right = TDeleteNode(root->right, key);

// если ключ совпадает с ключом текущего узла, удаляем текущий узел

else {

// если у текущего узла нет одного из потомков или оба потомка отсутствуют

if (root->left == nullptr || root->right == nullptr) {

// выбираем непустого потомка или nullptr, если оба потомка отсутствуют

AVLnode\* temp = root->left ? root->left : root->right;

// если непустой потомок отсутствует, удаляем текущий узел

if (temp == nullptr) {

temp = root; //переходим к этому потомку

root = nullptr; //обнуляем его

}

// если есть непустой потомок, копируем его данные в текущий узел

else

\*root = \*temp;

delete temp; // освобождаем память удаляемого узла

}

// если у текущего узла есть оба потомка

else {

AVLnode\* temp = TMinValueLeftNode(root->right); // находим узел с минимальным ключом в правом поддереве

root->key = temp->key; // копируем ключ минимального узла в текущий узел

root->right = TDeleteNode(root->right, temp->key); // рекурсивно удаляем минимальный узел из правого поддерева

}

}

// если дерево стало пустым после удаления, возвращаем его

if (root == nullptr) {

return root;

}

// обновляем высоту текущего узла

root->height = 1 + max(THeight(root->left), THeight(root->right));

// вычисляем баланс текущего узла

int balance = TBalance(root);

// левый-левый случай: баланс-фактор > 1 и баланс-фактор левого поддерева >= 0

if (balance > 1 && TBalance(root->left) >= 0)

return TRightRotate(root);

// левый-правый случай: баланс-фактор > 1 и баланс-фактор левого поддерева < 0

if (balance > 1 && TBalance(root->left) < 0) {

root->left = TLeftRotate(root->left);

return TRightRotate(root);

}

// правый-правый случай: баланс-фактор < -1 и баланс-фактор правого поддерева <= 0

if (balance < -1 && TBalance(root->right) <= 0)

return TLeftRotate(root);

// правый-левый случай: баланс-фактор < -1 и баланс-фактор правого поддерева > 0

if (balance < -1 && TBalance(root->right) > 0) {

root->right = TRightRotate(root->right);

return TLeftRotate(root);

}

return root; // возвращаем корень поддерева

}

bool Find(string& element) {

return TSearch(root, element) != nullptr;

}

AVLnode\* TSearch(AVLnode\* node, string& key) {

if (node == nullptr || node->key == key) {// если узел пустой или ключ найден, возвращаем узел

return node;

}

if (key < node->key)

return TSearch(node->left, key);

else

return TSearch(node->right, key);

}

void TDisplay(AVLnode\* node) {

if (node != nullptr) {

cout << node->key << " "; // выводим ключ текущего узла

TDisplay(node->left); // рекурсивно обходим левое поддерево

TDisplay(node->right); // рекурсивно обходим правое поддерево

}

}

void TPUSH(string& key) {

root = TInsert(root, key);

}

void TDEL(string& key) {

root = TDeleteNode(root, key);

}

bool TSEARCH(string& key) {

return TSearch(root, key) != nullptr;

}

void TREAD() {

TDisplay(root);

cout << endl;

}

void WritingFromFileToStructure(string& filename) {

ifstream FileRead(filename);

string tempValue;

while (getline(FileRead, tempValue)) {

stringstream ss(tempValue);

string element;

while (ss >> element) {

TPUSH(element);

}

}

FileRead.close();

}

void SaveToFile(string& filename) {

ofstream FileWrite(filename);

SavingProcess(root, FileWrite);

FileWrite.close();

}

void SavingProcess(AVLnode\* node, ofstream& FileWrite) {

if (node == nullptr) {

return;

}

FileWrite << node->key << " "; // записываем ключ текущего узла

// рекурсивно записываем левое и правое поддеревья

SavingProcess(node->left, FileWrite);

SavingProcess(node->right, FileWrite);

}

/// //////////

void BinarySerialization(string& filename) {

ofstream File(filename, ios::binary);

SavingProcessBionary(root, File);

File.close();

}

void SavingProcessBionary(AVLnode\* node, ofstream& File) {

if (node == nullptr) { //если узел пустой

int markerZero = 0;

File.write(reinterpret\_cast<char\*>(&markerZero), sizeof(markerZero));

}

else {

int markerOne = 1;

File.write(reinterpret\_cast<char\*>(&markerOne), sizeof(markerOne));

int keyLen = node->key.length();

File.write(reinterpret\_cast<char\*>(&keyLen), sizeof(keyLen));

File.write(node->key.c\_str(), keyLen);

SavingProcessBionary(node->left, File);

SavingProcessBionary(node->right, File);

}

}

void BinaryDESerialization(string& filename) {

ifstream File(filename, ios::binary);

if (File.peek() == ifstream::traits\_type::eof()) { // проверка на пустоту файла с помощью peek

return;

}

LoadFromBinaryFileRecursive(root, File);

File.close();

}

void LoadFromBinaryFileRecursive(AVLnode\*& node, ifstream& File) {

int marker;

File.read(reinterpret\_cast<char\*>(&marker), sizeof(marker));

if (marker == 0) {

node = nullptr;

}

else {

int keyLen;

File.read(reinterpret\_cast<char\*>(&keyLen), sizeof(keyLen));

char\* buffer = new char[keyLen + 1];

File.read(buffer, keyLen);

buffer[keyLen] = '\0';

node = new AVLnode(buffer); //обновляем данные на нормальный формат

delete[] buffer;

LoadFromBinaryFileRecursive(node->left, File);

LoadFromBinaryFileRecursive(node->right, File);

}

}

};

### HashTable.h

#pragma once

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

class HashTable {

private:

struct NodeHash {

string key;

string value;

NodeHash\* next;

NodeHash(string& k, string& v) : key(k), value(v), next(nullptr) {}

};

NodeHash\*\* table; // массив указателей на узлы хеш-таблицы

int size;

int count;

public:

int Size() {

return count;

}

HashTable() {

size = 1000;

count = 0;

table = new NodeHash \* [size];

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

table[i] = nullptr;

}

}

~HashTable() {

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

HDestroyTable(table[i]);

}

delete[] table;

}

void HDestroyTable(NodeHash\* node) {

while (node != nullptr) {

NodeHash\* temp = node;

node = node->next;

delete temp;

}

}

HashTable(int initialSize) {

size = initialSize;

count = 0;

table = new NodeHash \* [size];

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

table[i] = nullptr;

}

}

int HashFunc(string& str) {

int hash = 0;

for (char c : str) {

hash += c; //ASCII

}

return hash % size;

}

void HPUSH(string& key, string& value) {

int index = HashFunc(key);

NodeHash\* current = table[index];

while (current != nullptr) {

if (current->key == key) {

current->value = value;

return;

}

current = current->next;

}

NodeHash\* newNode = new NodeHash(key, value); // создание нового узла

newNode->next = table[index]; // новый узел указывает на текущий узел по индексу

table[index] = newNode; // новый узел становится текущим узлом по индексу

count++;

}

void HDEL(string& key) {

int index = HashFunc(key);

NodeHash\* current = table[index];

NodeHash\* prev = nullptr;

while (current != nullptr) {

if (current->key == key) {

if (prev == nullptr) { // если это первый узел в списке

table[index] = current->next; // следующий узел становится текущим

}

else {

prev->next = current->next; // предыдущий узел указывает на следующий узел

}

delete current;

count--;

return;

}

prev = current; // переход к следующему узлу

current = current->next;

}

}

string HGET(string& key) {

int index = HashFunc(key);

NodeHash\* current = table[index];

while (current != nullptr) {

if (current->key == key) { // если ключ найден

return current->value;

}

current = current->next;

}

return "\n\nThe key was not found\n\n";

}

void HREAD() {

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

cout << "[" << i << "]: ";

NodeHash\* current = table[i];

if (current == nullptr) {

cout << "empty" << endl;

continue;

}

while (current != nullptr) {

cout << "(" << current->key << ", " << current->value << ") ";

current = current->next;

}

cout << endl;

}

}

void SaveToFile(string &filename){

ofstream FileWrite(filename);

if (FileWrite.is\_open()) {

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

NodeHash\* current = table[i];

while (current != nullptr) {

FileWrite << current->key << " " << current->value << endl;

current = current->next;

}

}

FileWrite.close();

}

}

void WriteFromFile(string& filename) {

ifstream FileRead(filename);

if (FileRead.is\_open()) {

string key, value;

while (FileRead >> key >> value) {

HPUSH(key, value);

}

FileRead.close();

}

}

void BinarySerialization(string& filename) {

ofstream FileWrite(filename, ios::binary);

if (!FileWrite.is\_open()) return;

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

NodeHash\* current = table[i];

while (current != nullptr) {

int keyLen = current->key.size();

int valueLen = current->value.size();

FileWrite.write(reinterpret\_cast<char\*>(&keyLen), sizeof(keyLen));

FileWrite.write(current->key.c\_str(), keyLen);

FileWrite.write(reinterpret\_cast<char\*>(&valueLen), sizeof(valueLen));

FileWrite.write(current->value.c\_str(), valueLen);

current = current->next;

}

}

FileWrite.close();

}

void BinaryDEserialization(const string& filename) {

ifstream fileRead(filename, ios::binary);

while (fileRead) {

int keyLen, valueLen;

fileRead.read(reinterpret\_cast<char\*>(&keyLen), sizeof(keyLen));

if (fileRead.eof()) break;

char\* keyBuffer = new char[keyLen + 1];

fileRead.read(keyBuffer, keyLen);

keyBuffer[keyLen] = '\0';

string key(keyBuffer);

fileRead.read(reinterpret\_cast<char\*>(&valueLen), sizeof(valueLen));

char\* valueBuffer = new char[valueLen + 1];

fileRead.read(valueBuffer, valueLen);

valueBuffer[valueLen] = '\0';

string value(valueBuffer);

HPUSH(key, value);

delete valueBuffer;

delete keyBuffer;

}

fileRead.close();

}

};

### Header.h

#pragma once

#include <sstream>

#include <iostream>

#include <fstream>

#include <algorithm>

#include <cstring>

using namespace std;

string elementA = "hehe";

string elementB = "meme";

string elementC = "lol";

struct DoubleNode {

string cell;

DoubleNode\* next;

DoubleNode\* prev;

};

struct SingleNode {

string cell;

SingleNode\* next;

};

//const char\* args[] = {"./program", "--file", "file.txt", "--query", "CREATE TABLE meme"};

void inputArguments(int argc, char\* argv[], string& filename, string& query) {

bool fileFound = false;

bool queryFound = false;

for (int i = 1; i < argc; i++) {

string part = argv[i];

if (part.find("--file") != string::npos ) {

if (i + 1 < argc && argv[i+1] != "--query") {

filename = argv[i + 1];

fileFound = true;

i++;

} else {

//cout << "ERROR!!!!! Missing file" << endl;

return;

}

}

else if (part.find("--query") != string::npos) {

if (i + 1 < argc) {

query = argv[i + 1];

queryFound = true;

i++;

} else {

//cout << "ERROR!!!!! Missing query" << endl;

return;

}

}

}

if (!fileFound) {

//cout << "ERROR!!!!! Missing --file" << endl;

}

if (!queryFound) {

//cout << "ERROR!!!!! Missing --query" << endl;

}

}

### OneList.h

#pragma once

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

class OneList {

private:

SingleNode\* head = nullptr;

SingleNode\* tail = nullptr;

SingleNode\* predTail = nullptr; //добавили ссылку на пред хвост, чтобы убрать while

public:

~OneList() {

Clear();

}

int SearchForTest(string& value) {

SingleNode\* node = head;

if (head == nullptr) {

return -2;

}

int index = 0;

if (head->cell == value) {

return index;

}

while (node != nullptr && node->cell != value) {

node = node->next;

index++;

}

if (node == nullptr) {

return -1;

}

return index;

}

int Size(){

int count = 0;

SingleNode\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

void Clear() {

while (head != nullptr) {

SingleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void LONEPUSHhead(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (head == nullptr) {

head = tail = node;

predTail = nullptr;/////////////

}

else {

node->next = head;

head = node;

predTail = head->next;

}

}

void LONEPUSHtail(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (tail == nullptr) {

head = tail = node;

predTail = nullptr;//////

}

else {

tail->next = node;

predTail = tail; ////////

tail = node;

}

}

void LONEDELhead() {

if (head == nullptr) {

//cout << "\nThe list is empty!" << endl;

return;

}

SingleNode\* node = head;

head = head->next;

delete node;

if (head == nullptr) {

tail = nullptr;

predTail = nullptr;///////////////////

}

}

void LONEDELtail() {

if (head == nullptr) {

//cout << "\nThe list is empty!" << endl;

return;

}

if (head == tail) {

delete head;

head = tail = nullptr;

predTail = nullptr;////////////////////

return;

}

SingleNode\* current = predTail; /////убрали while

predTail->next = tail->next;

delete tail;

tail = current;

tail->next = nullptr;

}

void LONEDELvalue(string& value) {

if (head == nullptr) {

//cout << "\nThe list is empty!" << endl;

return;

}

SingleNode\* node = head;

if (head->cell == value) {

LONEDELhead();

return;

}

while (node != nullptr && node->next != nullptr && node->next->cell != value) {

if (node == nullptr) {

//cout << "\nNo value found!" << endl;

return;

}

node = node->next;

}

SingleNode\* toDelete = node->next;

node->next = toDelete->next;

if (toDelete == tail) {

tail = toDelete;

predTail->next = tail->next;/////////////

}

delete toDelete;

tail = predTail;

}

void LONESEARCH(string& value) {

SingleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

int index = 0;

if (head->cell == value) {

cout << "\nThe element is found! Index: " << index << endl;

return;

}

while (node != nullptr && node->cell != value) {

node = node->next;

index++;

}

if (node == nullptr) {

cout << "\nNo value found!" << endl;

return;

}

cout << "\nThe element is found! Index: " << index << endl;

}

void LONEREAD() {

SingleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

cout << endl;

while (node != nullptr) {

cout << node->cell << " ";

node = node->next;

}

cout << endl;

}

void WritingFromFileToStructure(string& filename) {

ifstream FileRead(filename);

string tempValue;

while (getline(FileRead, tempValue)) {

stringstream ss(tempValue);

string element;

while (ss >> element) {

LONEPUSHtail(element);

}

}

FileRead.close();

}

void WritingFromStructureToFile(string& filename) {

ofstream FileWrite(filename);

SingleNode\* current = head;

while (current != nullptr) {

FileWrite << current->cell << " ";

current = current->next;

}

FileWrite.close();

}

void BinarySerialization(string& filename) {

ofstream FileWrite(filename, ios::binary);

SingleNode\* current = head;

while (current != nullptr) {

int len = current->cell.size();

FileWrite.write(reinterpret\_cast<char\*>(&len), sizeof(len)); //сначала записываем длины строк

FileWrite.write(current->cell.c\_str(), len); //затем записываем данные с начала туда до /0

current = current->next;

}

FileWrite.close();

}

void BinaryDEserialization(string& filename) {

ifstream fileRead(filename, ios::binary);

while (fileRead) {

int len;

fileRead.read(reinterpret\_cast<char\*>(&len), sizeof(len)); //чтение длины строки

if (fileRead.eof())

break;

char\* buffer = new char[len + 1];

fileRead.read(buffer, len);

buffer[len] = '\0';

string text(buffer);

LONEPUSHtail(text);

delete buffer;

}

fileRead.close();

}

};

### Queue.h

#pragma once

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

class Queue {

private:

SingleNode\* head = nullptr;

SingleNode\* tail = nullptr;

public:

~Queue() {

Clear();

}

int Size(){

int count = 0;

SingleNode\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

void Clear() {

while (head != nullptr) {

SingleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void QPUSH(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = nullptr;

if (head == nullptr) {

head = node;

tail = node;

}

else {

tail->next = node;

tail = node;

}

}

void QPOP() {

SingleNode\* node = head;

head = head->next;

delete node;

if (head == nullptr) {

tail = nullptr;

}

}

void QREAD() {

SingleNode\* current = head;

cout << endl;

while (current != nullptr) {

cout << current->cell << " ";

current = current->next;

}

cout << endl;

}

void WritingFromFileToStructure(string& filename) {

ifstream FileRead(filename);

string tempValue;

while (getline(FileRead, tempValue)) {

stringstream ss(tempValue);

string element;

while (ss >> element) {

QPUSH(element);

}

}

FileRead.close();

}

void WritingFromStructureToFile( string& filename) {

ofstream FileWrite(filename);

SingleNode\* current = head;

while (current != nullptr) {

FileWrite << current->cell << " ";

current = current->next;

}

FileWrite.close();

}

void BinarySerialization(string& filename) {

ofstream FileWrite(filename, ios::binary);

SingleNode\* current = head;

while (current != nullptr) {

int len = current->cell.size();

FileWrite.write(reinterpret\_cast<char\*>(&len), sizeof(len));

FileWrite.write(current->cell.c\_str(), len);

current = current->next;

}

FileWrite.close();

}

void BinaryDEserialization(string& filename) {

ifstream fileRead(filename, ios::binary);

while (fileRead) {

int len;

fileRead.read(reinterpret\_cast<char\*>(&len), sizeof(len));

if (fileRead.eof())

break;

char\* buffer = new char[len + 1];

fileRead.read(buffer, len);

buffer[len] = '\0';

string text(buffer);

QPUSH(text);

delete buffer;

}

fileRead.close();

}

};

### Stack.h

#pragma once

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

class Stack {

private:

SingleNode\* head = nullptr;

SingleNode\* tail = nullptr;

public:

~Stack() {

Clear();

}

int Size(){

int count = 0;

SingleNode\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

void Clear() {

while (head != nullptr) {

SingleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void SPUSH(string& cell) {

SingleNode\* node = new SingleNode;

node->cell = cell;

node->next = head;

head = node;

if (tail == nullptr) {

tail = node;

}

}

void WriteFromFile(string& cell) { ///////////////

SingleNode\* node = new SingleNode;

node->cell = cell;

if (head == nullptr) {

head = node;

tail = node;

}

else {

tail->next = node;

tail = node;

}

}

void SPOP() {

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

SingleNode\* current = head; ///////////

head = head->next;

delete current;

}

void SREAD() {

SingleNode\* current = head;

cout << endl;

while (current != nullptr) {

cout << current->cell << " ";

current = current->next;

}

cout << endl;

}

void WritingFromFileToStructure(string& filename) {

ifstream FileRead(filename);

string tempValue;

while (getline(FileRead, tempValue)) {

stringstream ss(tempValue);

string element;

while (ss >> element) {

WriteFromFile(element);

}

}

FileRead.close();

}

void WritingFromStructureToFile(string& filename) {

ofstream FileWrite(filename);

if (head == nullptr) {

return;

}

SingleNode\* current = head;

while (current != nullptr) {

FileWrite << current->cell << " ";

current = current->next;

}

FileWrite.close();

}

void BinarySerialization(string& filename) {

ofstream FileWrite(filename, ios::binary);

SingleNode\* current = head;

while (current != nullptr) {

int len = current->cell.size();

FileWrite.write(reinterpret\_cast<char\*>(&len), sizeof(len));

FileWrite.write(current->cell.c\_str(), len);

current = current->next;

}

FileWrite.close();

}

void BinaryDEserialization(string& filename) {

ifstream fileRead(filename, ios::binary);

while (fileRead) {

int len;

fileRead.read(reinterpret\_cast<char\*>(&len), sizeof(len));

if (fileRead.eof())

break;

char\* buffer = new char[len + 1];

fileRead.read(buffer, len);

buffer[len] = '\0';

string text(buffer);

SingleNode\* node = new SingleNode;

node->cell = text;

node->next = nullptr;

if (head == nullptr) {

head = node;

tail = node;

}

else {

tail->next = node;

tail = node;

}

delete buffer;

}

fileRead.close();

}

};

### TwoList.h

#pragma once

#include "header.h"

#include <iostream>

#include <fstream>

using namespace std;

class TwoList {

private:

DoubleNode\* head = nullptr;

DoubleNode\* tail = nullptr;

public:

~TwoList() {

Clear();

}

int SearchForTest(string& value) {

DoubleNode\* node = head;

if (head == nullptr) {

return -2;

}

int index = 0;

if (head->cell == value) {

return index;

}

while (node != nullptr && node->cell != value) {

node = node->next;

index++;

}

if (node == nullptr) {

return -1;

}

return index;

}

int Size(){

int count = 0;

DoubleNode\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

void Clear() {/////

while (head != nullptr) {

DoubleNode\* temp = head;

head = head->next;

delete temp;

}

tail = nullptr;

}

void LTWOPUSHhead(string& element) {

DoubleNode\* node = new DoubleNode;

node->cell = element;

node->next = nullptr;

node->prev = nullptr;

if (head == nullptr) {

head = tail = node;

}

else {

node->next = head;

head->prev = node;

head = node;

}

}

void LTWOPUSHtail(string& element) {

DoubleNode\* node = new DoubleNode;

node->cell = element;

node->next = nullptr;

node->prev = nullptr;

if (tail == nullptr) {

head = tail = node;

}

else {

tail->next = node;

node->prev = tail;

tail = node;

}

}

void LTWODELhead() {

if (head == nullptr) {

//cout << "\nThe list is empty!" << endl;

return;

}

DoubleNode\* node = head;

head = head->next;

delete node;

if (head == nullptr) {

tail = nullptr;

}

}

void LTWODELtail() {

if (head == nullptr) {

//cout << "\nThe list is empty!" << endl;

return;

}

if (head == tail) {

delete head;

head = tail = nullptr;

return;

}

DoubleNode\* current = tail->prev; //убрали while

current->next = tail->next;

delete tail;

tail = current;

tail->next = nullptr;

tail->prev = current->prev;

}

void LTWODELvalue(string& value) {

DoubleNode\* node = head;

if (head == nullptr) {

//cout << "\nThe list is empty!" << endl;

return;

}

if (head->cell == value) {

LTWODELhead();

return;

}

while (node != nullptr && node->cell != value) {

node = node->next;

}

if (node == nullptr) {

cout << "\nNo value found!" << endl;

return;

}

if (node->prev) {

node->prev->next = node->next;

}

else {

head = node->prev;

}

if (node->next) {

node->next->prev = node->prev;

}

else {

tail = node->next;

}

delete node;

}

void LTWOSEARCH(string& value) {

DoubleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

int index = 0;

if (head->cell == value) {

cout << "\nThe element is found! Index: " << index << endl;

return;

}

while (node != nullptr && node->cell != value) {

node = node->next;

index++;

}

if (node == nullptr) {

cout << "\nNo value found!" << endl;

return;

}

cout << "\nThe element is found! Index: " << index << endl;

}

void LTWOREAD() {

DoubleNode\* node = head;

if (head == nullptr) {

cout << "\nThe list is empty!" << endl;

return;

}

cout << endl;

while (node != nullptr) {

cout << node->cell << " ";

node = node->next;

}

cout << endl;

}

void SaveToFile(string&name) {

ofstream FileWrite(name);

DoubleNode\* current = head;

while (current != nullptr) {

FileWrite << current->cell << " ";

current = current->next;

}

FileWrite.close();

}

void WritingFromFileToStructure(string& filename) {

ifstream FileRead(filename);

string tempValue;

while (getline(FileRead, tempValue)) {

stringstream ss(tempValue);

string element;

while (ss >> element) {

LTWOPUSHtail(element);

}

}

FileRead.close();

}

void BinarySerialization(string& filename) {

ofstream FileWrite(filename, ios::binary);

DoubleNode\* current = head;

while (current != nullptr) {

int len = current->cell.size();

FileWrite.write(reinterpret\_cast<char\*>(&len), sizeof(len)); //сначала записываем длины строк

FileWrite.write(current->cell.c\_str(), len); //затем записываем данные с начала туда до /0

current = current->next;

}

FileWrite.close();

}

void BinaryDEserialization(string& filename) {

ifstream fileRead(filename, ios::binary);

DoubleNode\* current = nullptr;

DoubleNode\* tail = nullptr;

while (fileRead) {

int len;

fileRead.read(reinterpret\_cast<char\*>(&len), sizeof(len)); //чтение длины строки

if (fileRead.eof())

break;

char\* buffer = new char[len + 1];

fileRead.read(buffer, len);

buffer[len] = '\0';

string text(buffer);

LTWOPUSHtail(text);

delete buffer;

}

fileRead.close();

}

};

## Функции тестирования классов:

### Test\_Array.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../Array.h"

#include "../header.h"

BOOST\_AUTO\_TEST\_SUITE(TestArray)

BOOST\_AUTO\_TEST\_CASE(TestArray\_MPUSHend) {

Array array;

array.MPUSHend(elementA);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 1);

array.MPUSHend(elementB);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 2);

array.MPUSHend(elementC);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 3);

string A = array.MGETL(0);

string B = array.MGETL(1);

string C = array.MGETL(2);

A.erase(remove(A.begin(), A.end(), ' '), A.end());

B.erase(remove(B.begin(), B.end(), ' '), B.end());

C.erase(remove(C.begin(), C.end(), ' '), C.end());

BOOST\_CHECK\_EQUAL(A, elementA);

BOOST\_CHECK\_EQUAL(B, elementB);

BOOST\_CHECK\_EQUAL(C, elementC);

}

BOOST\_AUTO\_TEST\_CASE(TestArray\_MPUSHindex) {

Array array;

array.MPUSHindex(0, elementA);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 1);

array.MPUSHindex(1, elementB);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 2);

array.MPUSHindex(2, elementC);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 3);

string A = array.MGETL(0);

string B = array.MGETL(1);

string C = array.MGETL(2);

A.erase(remove(A.begin(), A.end(), ' '), A.end());

B.erase(remove(B.begin(), B.end(), ' '), B.end());

C.erase(remove(C.begin(), C.end(), ' '), C.end());

BOOST\_CHECK\_EQUAL(A, elementA);

BOOST\_CHECK\_EQUAL(B, elementB);

BOOST\_CHECK\_EQUAL(C, elementC);

string temp = "hehe";

array.MPUSHindex(1, temp);

string temp1 = array.MGETL(1);

string temp2 = array.MGETL(2);

string temp3 = array.MGETL(3);

temp1.erase(remove(temp1.begin(), temp1.end(), ' '), temp1.end());

temp2.erase(remove(temp2.begin(), temp2.end(), ' '), temp2.end());

temp3.erase(remove(temp3.begin(), temp3.end(), ' '), temp3.end());

BOOST\_CHECK\_EQUAL(temp1, temp);

BOOST\_CHECK\_EQUAL(temp2, elementB);

BOOST\_CHECK\_EQUAL(temp3, elementC);

}

BOOST\_AUTO\_TEST\_CASE(TestArray\_MDEL) {

Array array;

array.MPUSHend(elementA);

array.MPUSHend(elementB);

array.MPUSHend(elementC);

array.MDEL(0);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 2);

array.MDEL(0);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 1);

string temp = array.MGETL(0);

temp.erase(remove(temp.begin(), temp.end(), ' '), temp.end());

BOOST\_CHECK\_EQUAL(temp, elementC);

array.MPUSHend(elementA);

array.MPUSHend(elementB);

array.MPUSHend(elementC);

array.MDEL(0);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 3);

array.MDEL(0);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 2);

array.MDEL(0);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 1);

array.MDEL(0);

int size = array.MSIZE();

BOOST\_CHECK\_EQUAL(size, 0);

array.MPUSHend(elementA);

array.MPUSHend(elementB);

array.MPUSHend(elementC);

array.MDEL(-1);

array.MDEL(1001);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestArray\_MREPL) {

Array array;

array.MPUSHend(elementA);

array.MPUSHend(elementB);

array.MPUSHend(elementC);

string temp = "GGG";

array.MREPL(0, temp);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 3);

array.MREPL(1, temp);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 3);

array.MREPL(2, temp);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 3);

string A = array.MGETL(0);

string B = array.MGETL(1);

string C = array.MGETL(2);

A.erase(remove(A.begin(), A.end(), ' '), A.end());

B.erase(remove(B.begin(), B.end(), ' '), B.end());

C.erase(remove(C.begin(), C.end(), ' '), C.end());

BOOST\_CHECK\_EQUAL(A, temp);

BOOST\_CHECK\_EQUAL(B, temp);

BOOST\_CHECK\_EQUAL(C, temp);

string temp1 = "value";

array.MREPL(-1, temp1);

array.MREPL(1001, temp1);

A = array.MGETL(0);

B = array.MGETL(1);

C = array.MGETL(2);

A.erase(remove(A.begin(), A.end(), ' '), A.end());

B.erase(remove(B.begin(), B.end(), ' '), B.end());

C.erase(remove(C.begin(), C.end(), ' '), C.end());

BOOST\_CHECK\_EQUAL(A, temp);

BOOST\_CHECK\_EQUAL(B, temp);

BOOST\_CHECK\_EQUAL(C, temp);

}

BOOST\_AUTO\_TEST\_CASE(TestArray\_MGETL) {

Array array;

array.MPUSHend(elementA);

array.MPUSHend(elementB);

array.MPUSHend(elementC);

string A = array.MGETL(0);

string B = array.MGETL(1);

string C = array.MGETL(2);

A.erase(remove(A.begin(), A.end(), ' '), A.end());

B.erase(remove(B.begin(), B.end(), ' '), B.end());

C.erase(remove(C.begin(), C.end(), ' '), C.end());

BOOST\_CHECK\_EQUAL(A, elementA);

BOOST\_CHECK\_EQUAL(B, elementB);

BOOST\_CHECK\_EQUAL(C, elementC);

BOOST\_CHECK\_EQUAL(array.MGETL(-1), "ERROR");

BOOST\_CHECK\_EQUAL(array.MGETL(1001), "ERROR");

}

BOOST\_AUTO\_TEST\_CASE(TestArray\_MSIZE) {

Array array;

BOOST\_CHECK\_EQUAL(array.MSIZE(), 0);

array.MPUSHend(elementA);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 1);

array.MPUSHend(elementB);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 2);

array.MDEL(1);

BOOST\_CHECK\_EQUAL(array.MSIZE(), 1);

}

BOOST\_AUTO\_TEST\_CASE(TestArray\_DUPLICATES) {

Array array;

array.MPUSHend(elementA);

array.MPUSHend(elementA);

string temp1 = array.MGETL(0);

string temp2 = array.MGETL(1);

temp1.erase(remove(temp1.begin(), temp1.end(), ' '), temp1.end());

temp2.erase(remove(temp2.begin(), temp2.end(), ' '), temp2.end());

BOOST\_CHECK\_EQUAL(temp1, elementA);

BOOST\_CHECK\_EQUAL(temp2, elementA);

}

BOOST\_AUTO\_TEST\_CASE(TestArray\_EMPTY) {

Array array;

string temp1 = "";

array.MPUSHend(temp1);

string temp2 = array.MGETL(0);

temp2.erase(remove(temp2.begin(), temp2.end(), ' '), temp2.end());

BOOST\_CHECK\_EQUAL(temp2, temp1);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_AVL.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../AVLtree.h"

#include "../header.h"

BOOST\_AUTO\_TEST\_SUITE(TestAVLtree)

BOOST\_AUTO\_TEST\_CASE(TestAVLtree\_TPUSH) {

AVL AVLtree;

AVLtree.TPUSH(elementA);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 1);

AVLtree.TPUSH(elementB);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 2);

AVLtree.TPUSH(elementC);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestAVLtree\_TDEL) {

AVL AVLtree;

AVLtree.TPUSH(elementA);

AVLtree.TPUSH(elementB);

AVLtree.TPUSH(elementC);

AVLtree.TDEL(elementA);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 2);

AVLtree.TDEL(elementB);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 1);

AVLtree.TDEL(elementC);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 0);

string temp1 = "5";

string temp2 = "10";

string temp3 = "15";

string temp4 = "20";

string temp5 = "30";

AVLtree.TPUSH(temp1);

AVLtree.TPUSH(temp2);

AVLtree.TPUSH(temp3);

AVLtree.TPUSH(temp4);

AVLtree.TPUSH(temp5);

AVLtree.TDEL(temp4);

AVLtree.TDEL(temp3);

AVLtree.TDEL(temp2);

AVLtree.TDEL(temp1);

AVLtree.TDEL(temp5);

}

BOOST\_AUTO\_TEST\_CASE(TestAVLtree\_TSEARCH) {

AVL AVLtree;

AVLtree.TPUSH(elementA);

AVLtree.TPUSH(elementB);

AVLtree.TPUSH(elementC);

AVLtree.TDEL(elementA);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementA), false);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementB), true);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementC), true);

AVLtree.TDEL(elementB);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementA), false);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementB), false);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementC), true);

AVLtree.TDEL(elementC);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementA), false);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementB), false);

BOOST\_CHECK\_EQUAL(AVLtree.Find(elementC), false);

}

BOOST\_AUTO\_TEST\_CASE(TestAVLtree\_DUPLICATES) {

AVL AVLtree;

string temp = "50";

AVLtree.TPUSH(temp);

AVLtree.TPUSH(temp);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 1);

BOOST\_CHECK\_EQUAL(AVLtree.GetRoot()->key, temp);

}

BOOST\_AUTO\_TEST\_CASE(TestAVLtree\_BALANCE) {

AVL AVLtree;

string temp1 = "5";

string temp2 = "10";

string temp3 = "15";

string temp4 = "20";

string temp5 = "30";

AVLtree.TPUSH(temp1);

AVLtree.TPUSH(temp2);

AVLtree.TPUSH(temp3);

AVLtree.TPUSH(temp4);

AVLtree.TPUSH(temp5);

BOOST\_CHECK\_EQUAL(AVLtree.Size(), 5);

BOOST\_CHECK\_EQUAL(AVLtree.THeight(AVLtree.GetRoot()), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestAVLtree\_MINVALUE) {

AVL AVLtree;

string temp1 = "10";

string temp2 = "5";

string temp3 = "15";

AVLtree.TPUSH(temp1);

AVLtree.TPUSH(temp2);

AVLtree.TPUSH(temp3);

BOOST\_CHECK\_EQUAL(AVLtree.TMinValueLeftNode(AVLtree.GetRoot())->key,temp1);

}

BOOST\_AUTO\_TEST\_CASE(TestAVLtree\_CHECKROOT) {

AVL AVLtree;

string element1 = "30";

string element2 = "20";

string element3 = "40";

AVLtree.TPUSH(element1);

AVLtree.TPUSH(element2);

AVLtree.TPUSH(element3);

BOOST\_CHECK\_EQUAL(AVLtree.GetRoot()->key, "30");

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_HashTable.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../HashTable.h"

#include "../header.h"

string A = "a";

string B = "b";

string C = "c";

BOOST\_AUTO\_TEST\_SUITE(TestHashTable)

BOOST\_AUTO\_TEST\_CASE(TestHashTable\_HPUSH) {

HashTable hashtable;

hashtable.HPUSH(A, elementA);

hashtable.HPUSH(B, elementB);

hashtable.HPUSH(C, elementC);

string temp1 = hashtable.HGET(A);

string temp2 = hashtable.HGET(B);

string temp3 = hashtable.HGET(C);

BOOST\_CHECK\_EQUAL(temp1, elementA);

BOOST\_CHECK\_EQUAL(temp2, elementB);

BOOST\_CHECK\_EQUAL(temp3, elementC);

}

BOOST\_AUTO\_TEST\_CASE(TestHashTable\_HDEL) {

HashTable hashtable;

hashtable.HPUSH(A, elementA);

hashtable.HPUSH(B, elementB);

hashtable.HPUSH(C, elementC);

hashtable.HDEL(A);

hashtable.HDEL(B);

hashtable.HDEL(C);

BOOST\_CHECK\_EQUAL(hashtable.Size(), 0);

}

BOOST\_AUTO\_TEST\_CASE(TestHashTable\_HGET) {

HashTable hashtable;

hashtable.HPUSH(A, elementA);

hashtable.HPUSH(B, elementB);

hashtable.HPUSH(C, elementC);

string temp1 = hashtable.HGET(A);

string temp2 = hashtable.HGET(B);

string temp3 = hashtable.HGET(C);

BOOST\_CHECK\_EQUAL(temp1, elementA);

BOOST\_CHECK\_EQUAL(temp2, elementB);

BOOST\_CHECK\_EQUAL(temp3, elementC);

hashtable.HDEL(A);

BOOST\_CHECK\_EQUAL(hashtable.HGET(A), "\n\nThe key was not found\n\n");

}

BOOST\_AUTO\_TEST\_CASE(TestHashTable\_SIZE) {

HashTable hashtable;

hashtable.HPUSH(A, elementA);

hashtable.HPUSH(B, elementB);

hashtable.HPUSH(C, elementC);

BOOST\_CHECK\_EQUAL(hashtable.Size(), 3);

hashtable.HDEL(A);

BOOST\_CHECK\_EQUAL(hashtable.Size(), 2);

hashtable.HDEL(B);

hashtable.HDEL(C);

BOOST\_CHECK\_EQUAL(hashtable.Size(), 0);

}

BOOST\_AUTO\_TEST\_CASE(TestHashTable\_HASHFUNC) {

HashTable hashtable;

int index = hashtable.HashFunc(elementA);

BOOST\_CHECK(index >= 0);

BOOST\_CHECK(index < 1000);

}

BOOST\_AUTO\_TEST\_CASE(TestHashTable\_COLLISION) {

HashTable hashtable;

hashtable.HPUSH(elementA, elementB);

hashtable.HPUSH(elementA, elementC);

hashtable.HPUSH(elementC, elementA);

string temp1 = hashtable.HGET(elementA);

string temp2 = hashtable.HGET(elementC);

BOOST\_CHECK\_EQUAL(temp1, elementC);

BOOST\_CHECK\_EQUAL(temp2, elementA);

}

BOOST\_AUTO\_TEST\_CASE(TestHashTable\_EMPTY) {

HashTable hashtable;

BOOST\_CHECK\_EQUAL(hashtable.Size(), 0);

hashtable.HDEL(A);

BOOST\_CHECK\_EQUAL(hashtable.Size(), 0);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_Header.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../header.h"

BOOST\_AUTO\_TEST\_SUITE(TestHeader)

BOOST\_AUTO\_TEST\_CASE(TestHeader\_ARGUMENTS) {

string filename;

string query;

const char\* args[] = {"./program", "--file", "file.txt", "--query", "CREATE TABLE meme"};

int argc = 5;

inputArguments(argc, const\_cast<char\*\*>(args), filename, query); //const\_cast убирает const

BOOST\_CHECK\_EQUAL(filename, "file.txt");

BOOST\_CHECK\_EQUAL(query, "CREATE TABLE meme");

}

BOOST\_AUTO\_TEST\_CASE(TestHeader\_EMPTY) {

string filename;

string query;

const char\* args[] = {"./program"};

int argc = sizeof(args) / sizeof(args[0]);

inputArguments(argc, const\_cast<char\*\*>(args), filename, query);

BOOST\_CHECK(filename.empty());

BOOST\_CHECK(query.empty());

}

BOOST\_AUTO\_TEST\_CASE(TestHeader\_MISSINGS) {

string filename1;

string query1;

string filename2;

string query2;

int argc = 4;

// --file

const char\* args1[] = {"./program", "--file", "--query", "CREATE TABLE meme"};

inputArguments(argc, const\_cast<char\*\*>(args1), filename1, query1);

BOOST\_CHECK(query1.empty());

// --query

const char\* args2[] = {"./program", "--file", "file.txt", "--query"};

inputArguments(argc, const\_cast<char\*\*>(args2), filename2, query2);

BOOST\_CHECK(query2.empty());

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_OneList.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../OneList.h"

#include "../header.h"

BOOST\_AUTO\_TEST\_SUITE(TestOneList)

BOOST\_AUTO\_TEST\_CASE(TestOneList\_LONEPUSHhead) {

OneList onelist;

onelist.LONEPUSHhead(elementA);

BOOST\_CHECK\_EQUAL(onelist.Size(), 1);

onelist.LONEPUSHhead(elementB);

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

onelist.LONEPUSHhead(elementC);

BOOST\_CHECK\_EQUAL(onelist.Size(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestOneList\_LONEPUSHtail) {

OneList onelist;

onelist.LONEPUSHtail(elementA);

BOOST\_CHECK\_EQUAL(onelist.Size(), 1);

onelist.LONEPUSHtail(elementB);

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

onelist.LONEPUSHtail(elementC);

BOOST\_CHECK\_EQUAL(onelist.Size(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestOneList\_LONEDELhead) {

OneList onelist;

onelist.LONEDELhead();

BOOST\_CHECK\_EQUAL(onelist.Size(), 0);

onelist.LONEPUSHhead(elementA);

onelist.LONEPUSHhead(elementB);

onelist.LONEPUSHhead(elementC);

onelist.LONEDELhead();

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

onelist.LONEDELhead();

BOOST\_CHECK\_EQUAL(onelist.Size(), 1);

onelist.LONEDELhead();

BOOST\_CHECK\_EQUAL(onelist.Size(), 0);

}

BOOST\_AUTO\_TEST\_CASE(TestOneList\_LONEDELtail) {

OneList onelist;

onelist.LONEDELtail();

BOOST\_CHECK\_EQUAL(onelist.Size(), 0);

onelist.LONEPUSHtail(elementA);

onelist.LONEDELtail();

BOOST\_CHECK\_EQUAL(onelist.Size(), 0);

onelist.LONEPUSHtail(elementA);

onelist.LONEPUSHtail(elementB);

onelist.LONEPUSHtail(elementC);

onelist.LONEDELtail();

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

}

BOOST\_AUTO\_TEST\_CASE(TestOneList\_LONEDELvalue) {

OneList onelist;

onelist.LONEPUSHtail(elementA);

onelist.LONEPUSHtail(elementB);

onelist.LONEPUSHtail(elementC);

onelist.LONEDELvalue(elementA);

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

BOOST\_CHECK\_EQUAL(onelist.SearchForTest(elementA),-1);

onelist.LONEDELvalue(elementB);

BOOST\_CHECK\_EQUAL(onelist.Size(), 1);

BOOST\_CHECK\_EQUAL(onelist.SearchForTest(elementB),-1);

onelist.LONEDELvalue(elementC);

BOOST\_CHECK\_EQUAL(onelist.Size(), 0);

BOOST\_CHECK\_EQUAL(onelist.SearchForTest(elementC),-2);

onelist.LONEPUSHtail(elementA);

onelist.LONEPUSHtail(elementB);

onelist.LONEPUSHtail(elementC);

onelist.LONEDELvalue(elementB);

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

}

BOOST\_AUTO\_TEST\_CASE(TestOneList\_LONESEARCH) {

OneList onelist;

onelist.LONEPUSHtail(elementA);

onelist.LONEPUSHtail(elementB);

onelist.LONEPUSHtail(elementC);

string notExist = "notExist";

int temp1 = onelist.SearchForTest(elementA);

int temp2 = onelist.SearchForTest(elementB);

int temp3 = onelist.SearchForTest(elementC);

int test = onelist.SearchForTest(notExist);

BOOST\_CHECK\_EQUAL(temp1, 0);

BOOST\_CHECK\_EQUAL(temp2, 1);

BOOST\_CHECK\_EQUAL(temp3, 2);

BOOST\_CHECK\_EQUAL(test, -1);

}

BOOST\_AUTO\_TEST\_CASE(TestOneList\_SIZE) {

OneList onelist;

onelist.LONEPUSHtail(elementA);

BOOST\_CHECK\_EQUAL(onelist.Size(), 1);

onelist.LONEPUSHtail(elementB);

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

onelist.LONEPUSHtail(elementC);

BOOST\_CHECK\_EQUAL(onelist.Size(), 3);

onelist.LONEDELhead();

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

onelist.LONEDELhead();

BOOST\_CHECK\_EQUAL(onelist.Size(), 1);

onelist.LONEDELhead();

BOOST\_CHECK\_EQUAL(onelist.Size(), 0);

}

BOOST\_AUTO\_TEST\_CASE(TestOneList\_DUPLICATES) {

OneList onelist;

onelist.LONEPUSHtail(elementA);

onelist.LONEPUSHtail(elementA);

BOOST\_CHECK\_EQUAL(onelist.Size(), 2);

onelist.LONEDELvalue(elementA);

BOOST\_CHECK\_EQUAL(onelist.Size(), 1);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_TwoList.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../TwoList.h"

#include "../header.h"

BOOST\_AUTO\_TEST\_SUITE(TestTwoList)

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_LTWOPUSHhead) {

TwoList twolist;

twolist.LTWOPUSHhead(elementA);

BOOST\_CHECK\_EQUAL(twolist.Size(), 1);

twolist.LTWOPUSHhead(elementB);

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

twolist.LTWOPUSHhead(elementC);

BOOST\_CHECK\_EQUAL(twolist.Size(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_LTWOPUSHtail) {

TwoList twolist;

twolist.LTWOPUSHtail(elementA);

BOOST\_CHECK\_EQUAL(twolist.Size(), 1);

twolist.LTWOPUSHtail(elementB);

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

twolist.LTWOPUSHtail(elementC);

BOOST\_CHECK\_EQUAL(twolist.Size(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_LTWODELhead) {

TwoList twolist;

twolist.LTWODELhead();

BOOST\_CHECK\_EQUAL(twolist.Size(), 0);

twolist.LTWOPUSHhead(elementA);

twolist.LTWOPUSHhead(elementB);

twolist.LTWOPUSHhead(elementC);

twolist.LTWODELhead();

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

twolist.LTWODELhead();

BOOST\_CHECK\_EQUAL(twolist.Size(), 1);

twolist.LTWODELhead();

BOOST\_CHECK\_EQUAL(twolist.Size(), 0);

}

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_LTWODELtail) {

TwoList twolist;

twolist.LTWODELtail();

BOOST\_CHECK\_EQUAL(twolist.Size(), 0);

twolist.LTWOPUSHtail(elementA);

twolist.LTWODELtail();

BOOST\_CHECK\_EQUAL(twolist.Size(), 0);

twolist.LTWOPUSHtail(elementA);

twolist.LTWOPUSHtail(elementB);

twolist.LTWOPUSHtail(elementC);

twolist.LTWODELtail();

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

}

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_LTWODELvalue) {

TwoList twolist;

twolist.LTWOPUSHtail(elementA);

twolist.LTWOPUSHtail(elementB);

twolist.LTWOPUSHtail(elementC);

twolist.LTWODELvalue(elementA);

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

BOOST\_CHECK\_EQUAL(twolist.SearchForTest(elementA),-1);

twolist.LTWODELvalue(elementB);

BOOST\_CHECK\_EQUAL(twolist.Size(), 1);

BOOST\_CHECK\_EQUAL(twolist.SearchForTest(elementB),-1);

twolist.LTWODELvalue(elementC);

BOOST\_CHECK\_EQUAL(twolist.Size(), 0);

BOOST\_CHECK\_EQUAL(twolist.SearchForTest(elementC),-2);

twolist.LTWOPUSHtail(elementA);

twolist.LTWOPUSHtail(elementB);

twolist.LTWOPUSHtail(elementC);

twolist.LTWODELvalue(elementB);

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

}

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_LTWOSEARCH) {

TwoList twolist;

twolist.LTWOPUSHtail(elementA);

twolist.LTWOPUSHtail(elementB);

twolist.LTWOPUSHtail(elementC);

string notExist = "notExist";

int temp1 = twolist.SearchForTest(elementA);

int temp2 = twolist.SearchForTest(elementB);

int temp3 = twolist.SearchForTest(elementC);

int test = twolist.SearchForTest(notExist);

BOOST\_CHECK\_EQUAL(temp1, 0);

BOOST\_CHECK\_EQUAL(temp2, 1);

BOOST\_CHECK\_EQUAL(temp3, 2);

BOOST\_CHECK\_EQUAL(test, -1);

}

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_SIZE) {

TwoList twolist;

twolist.LTWOPUSHtail(elementA);

BOOST\_CHECK\_EQUAL(twolist.Size(), 1);

twolist.LTWOPUSHtail(elementB);

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

twolist.LTWOPUSHtail(elementC);

BOOST\_CHECK\_EQUAL(twolist.Size(), 3);

twolist.LTWODELhead();

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

twolist.LTWODELhead();

BOOST\_CHECK\_EQUAL(twolist.Size(), 1);

twolist.LTWODELhead();

BOOST\_CHECK\_EQUAL(twolist.Size(), 0);

}

BOOST\_AUTO\_TEST\_CASE(TestTwoList\_DUPLICATES) {

TwoList twolist;

twolist.LTWOPUSHtail(elementA);

twolist.LTWOPUSHtail(elementA);

BOOST\_CHECK\_EQUAL(twolist.Size(), 2);

twolist.LTWODELvalue(elementA);

BOOST\_CHECK\_EQUAL(twolist.Size(), 1);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_Stack.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../Stack.h"

#include "../header.h"

BOOST\_AUTO\_TEST\_SUITE(TestStack)

BOOST\_AUTO\_TEST\_CASE(TestStack\_SPUSH) {

Stack stack;

stack.SPUSH(elementA);

stack.SPUSH(elementB);

stack.SPUSH(elementC);

BOOST\_CHECK\_EQUAL(stack.Size(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestStack\_SPOP) {

Stack stack;

stack.SPUSH(elementA);

stack.SPUSH(elementB);

stack.SPUSH(elementC);

stack.SPOP();

stack.SPOP();

stack.SPOP();

BOOST\_CHECK\_EQUAL(stack.Size(), 0);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_Queue.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../Queue.h"

#include "../header.h"

BOOST\_AUTO\_TEST\_SUITE(TestQueue)

BOOST\_AUTO\_TEST\_CASE(TestQueue\_QPUSH) {

Queue queue;

queue.QPUSH(elementA);

queue.QPUSH(elementB);

queue.QPUSH(elementC);

BOOST\_CHECK\_EQUAL(queue.Size(), 3);

}

BOOST\_AUTO\_TEST\_CASE(TestQueue\_QPOP) {

Queue queue;

queue.QPUSH(elementA);

queue.QPUSH(elementB);

queue.QPUSH(elementC);

queue.QPOP();

queue.QPOP();

queue.QPOP();

BOOST\_CHECK\_EQUAL(queue.Size(), 0);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Test\_Main.cpp

#define BOOST\_TEST\_MODULE Main

#include <boost/test/included/unit\_test.hpp>

#include "../Test\_Array.cpp"

#include "../Test\_AVL.cpp"

#include "../Test\_HashTable.cpp"

#include "../Test\_OneList.cpp"

#include "../Test\_Queue.cpp"

#include "../Test\_Stack.cpp"

#include "../Test\_TwoList.cpp"

#include "../Test\_header.cpp"

## Функции бенчмарка для классов:

### Benchmark\_Array.cpp

#include <boost/test/included/unit\_test.hpp>

#include <boost/timer/timer.hpp>

#include "../Array.h"

#include "Benchmark\_header.h"

string Arr = "element";

BOOST\_AUTO\_TEST\_SUITE(BenchmarkArray)

BOOST\_AUTO\_TEST\_CASE(BenchmarkArray\_MPUSHend) {

Array array;

boost::timer::cpu\_timer timer;

timer.start();

array.MPUSHend(Arr);

timer.stop();

string name = "BenchmarkArray\_MPUSHend";

string result = "time: " + timer.format(9);

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkArray\_MPUSHindex) {

Array array;

boost::timer::cpu\_timer timer;

timer.start();

array.MPUSHindex(0,Arr);

timer.stop();

string name = "BenchmarkArray\_MPUSHindex";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkArray\_MDEL) {

Array array;

boost::timer::cpu\_timer timer;

array.MPUSHend(Arr);

timer.start();

array.MDEL(0);

timer.stop();

string name = "BenchmarkArray\_MDEL";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkArray\_MREPL) {

Array array;

boost::timer::cpu\_timer timer;

string temp = "value";

array.MPUSHend(Arr);

timer.start();

array.MREPL(0,temp);

timer.stop();

string name = "BenchmarkArray\_MREPL";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkArray\_MGETL) {

Array array;

boost::timer::cpu\_timer timer;

string temp = "value";

array.MPUSHend(Arr);

timer.start();

array.MREPL(0,temp);

timer.stop();

string name = "BenchmarkArray\_MGETL";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Benchmark\_AVL.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../AVLtree.h"

#include "Benchmark\_header.h"

#include <boost/timer/timer.hpp>

string avl = "element";

BOOST\_AUTO\_TEST\_SUITE(BenchmarkAVLtree)

BOOST\_AUTO\_TEST\_CASE(BenchmarkAVLtree\_TPUSH) {

AVL AVLtree;

boost::timer::cpu\_timer timer;

timer.start();

AVLtree.TPUSH(avl);

timer.stop();

string name = "BenchmarkAVLtree\_TPUSH";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkAVLtree\_TDEL) {

AVL AVLtree;

boost::timer::cpu\_timer timer;

AVLtree.TPUSH(avl);

timer.start();

AVLtree.TDEL(avl);

timer.stop();

string name = "BenchmarkAVLtree\_TDEL";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkAVLtree\_TSEARCH) {

AVL AVLtree;

boost::timer::cpu\_timer timer;

AVLtree.TPUSH(avl);

timer.start();

AVLtree.TSEARCH(avl);

timer.stop();

string name = "BenchmarkAVLtree\_TSEARCH";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Benchmark\_ HashTable.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../HashTable.h"

#include "Benchmark\_header.h"

#include <boost/timer/timer.hpp>

string hashKey = "key";

string hashValue= "value";

BOOST\_AUTO\_TEST\_SUITE(BenchmarkHashTable)

BOOST\_AUTO\_TEST\_CASE(BenchmarkHashTable\_HPUSH) {

HashTable hashtable;

boost::timer::cpu\_timer timer;

timer.start();

hashtable.HPUSH(hashKey,hashValue);

timer.stop();

string name = "BenchmarkHashTable\_HPUSH";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkHashTable\_HDEL) {

HashTable hashtable;

boost::timer::cpu\_timer timer;

hashtable.HPUSH(hashKey,hashValue);

timer.start();

hashtable.HDEL(hashKey);

timer.stop();

string name = "BenchmarkHashTable\_HDEL";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkHashTable\_HGET) {

HashTable hashtable;

boost::timer::cpu\_timer timer;

hashtable.HPUSH(hashKey,hashValue);

timer.start();

hashtable.HGET(hashKey);

timer.stop();

string name = "BenchmarkHashTable\_HGET";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Benchmark\_ Header.cpp

#pragma once

#include <fstream>

#include <iostream>

#include <string>

#include <boost/timer/timer.hpp>

#include <boost/regex.hpp>

using namespace std;

void SaveBenchmarkReport(string& name, string& result) {

ofstream benchmark\_report("Benchmarks.html", ios::app);

if (benchmark\_report.is\_open()) {

benchmark\_report << "<html><head><title>Benchmark Report</title></head><body>";

benchmark\_report << "<h3>" << name << "</h3>";

benchmark\_report << "<p>" << result << "</p>";

benchmark\_report.close();

}

else {

cout << endl<< "ERROR!!!" << endl;

}

}

### Benchmark\_ OneList.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../OneList.h"

#include "Benchmark\_header.h"

#include <boost/timer/timer.hpp>

string one = "element";

BOOST\_AUTO\_TEST\_SUITE(BenchmarkOneList)

BOOST\_AUTO\_TEST\_CASE(BenchmarkOneList\_LONEPUSHhead) {

OneList onelist;

boost::timer::cpu\_timer timer;

timer.start();

onelist.LONEPUSHhead(one);

timer.stop();

string name = "BenchmarkOneList\_LONEPUSHhead";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkOneList\_LONEPUSHtail) {

OneList onelist;

boost::timer::cpu\_timer timer;

timer.start();

onelist.LONEPUSHtail(one);

timer.stop();

string name = "BenchmarkOneList\_LONEPUSHtail";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkOneList\_LONEDELhead) {

OneList onelist;

boost::timer::cpu\_timer timer;

onelist.LONEPUSHtail(one);

timer.start();

onelist.LONEDELhead();

timer.stop();

string name = "BenchmarkOneList\_LONEDELhead";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkOneList\_LONEDELtail) {

OneList onelist;

boost::timer::cpu\_timer timer;

onelist.LONEPUSHtail(one);

timer.start();

onelist.LONEDELtail();

timer.stop();

string name = "BenchmarkOneList\_LONEDELtail";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkOneList\_LONEDELvalue) {

OneList onelist;

boost::timer::cpu\_timer timer;

onelist.LONEPUSHtail(one);

timer.start();

onelist.LONEDELvalue(one);

timer.stop();

string name = "BenchmarkOneList\_LONEDELvalue";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Benchmark\_ TwoList.cpp

#include <boost/test/included/unit\_test.hpp>

#include "../TwoList.h"

#include "Benchmark\_header.h"

#include <boost/timer/timer.hpp>

string two = "element";

BOOST\_AUTO\_TEST\_SUITE(BenchmarkTwoList)

BOOST\_AUTO\_TEST\_CASE(BenchmarkTwoList\_LTWOPUSHhead) {

TwoList twolist;

boost::timer::cpu\_timer timer;

timer.start();

twolist.LTWOPUSHhead(two);

timer.stop();

string name = "BenchmarkTwoList\_LTWOPUSHhead";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkTwoList\_LTWOPUSHtail) {

TwoList twolist;

boost::timer::cpu\_timer timer;

timer.start();

twolist.LTWOPUSHtail(two);

timer.stop();

string name = "BenchmarkTwoList\_LTWOPUSHtail";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkTwoList\_LTWODELhead) {

TwoList twolist;

boost::timer::cpu\_timer timer;

twolist.LTWOPUSHtail(two);

timer.start();

twolist.LTWODELhead();

timer.stop();

string name = "BenchmarkTwoList\_LTWODELhead";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkTwoList\_LTWODELtail) {

TwoList twolist;

boost::timer::cpu\_timer timer;

twolist.LTWOPUSHtail(two);

timer.start();

twolist.LTWODELtail();

timer.stop();

string name = "BenchmarkTwoList\_LTWODELtai";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkTwoList\_LTWODELvalue) {

TwoList twolist;

boost::timer::cpu\_timer timer;

twolist.LTWOPUSHtail(two);

timer.start();

twolist.LTWODELvalue(two);

timer.stop();

string name = "BenchmarkTwoList\_LTWODELvalue";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Benchmark\_ Stack.cpp

#include "Benchmark\_header.h"

#include <boost/timer/timer.hpp>

#include <boost/test/included/unit\_test.hpp>

#include "../Stack.h"

string st = "element";

BOOST\_AUTO\_TEST\_SUITE(BenchmarkStack)

BOOST\_AUTO\_TEST\_CASE(BenchmarkStack\_SPUSH) {

Stack stack;

boost::timer::cpu\_timer timer;

timer.start();

stack.SPUSH(st);

timer.stop();

string name = "BenchmarkStack\_SPUSH";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkStack\_SPOP) {

Stack stack;

boost::timer::cpu\_timer timer;

stack.SPUSH(st);

timer.start();

stack.SPOP();

timer.stop();

string name = "BenchmarkStack\_SPOP";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Benchmark\_ Queue.cpp

#include "Benchmark\_header.h"

#include <boost/timer/timer.hpp>

#include <boost/test/included/unit\_test.hpp>

#include "../Queue.h"

string que = "element";

BOOST\_AUTO\_TEST\_SUITE(BenchmarkQueue)

BOOST\_AUTO\_TEST\_CASE(BenchmarkQueue\_QPUSH) {

Queue queue;

boost::timer::cpu\_timer timer;

timer.start();

queue.QPUSH(que);

timer.stop();

string name = "BenchmarkQueue\_QPUSH";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_CASE(BenchmarkQueue\_QPOP) {

Queue queue;

boost::timer::cpu\_timer timer;

queue.QPUSH(que);

timer.start();

queue.QPOP();

timer.stop();

string name = "BenchmarkQueue\_QPOP";

string result = "time: " + timer.format(9) + "s";

SaveBenchmarkReport(name, result);

}

BOOST\_AUTO\_TEST\_SUITE\_END()

### Benchmark\_ Main.cpp

#define BOOST\_TEST\_MODULE Main

#include <boost/test/included/unit\_test.hpp>

#include <boost/timer/timer.hpp>

#include "../Benchmark\_Array.cpp"

#include "../Benchmark\_Queue.cpp"

#include "../Benchmark\_Stack.cpp"

#include "../Benchmark\_OneList.cpp"

#include "../Benchmark\_TwoList.cpp"

#include "../Benchmark\_AVL.cpp"

#include "../Benchmark\_HashTable.cpp"

# Язык GO

## Функции класса Queue:

### main.go

package main //принадлежит пакету мэйн

import (

"bufio" // для ввод/вывод

"fmt" // для форматированного вывода

"os" // для операцимй системы, такими как открытие файлов

"strings"

"encoding/binary"

)

type SingleNode struct {

cell string

next \*SingleNode

}

type Queue struct {

head \*SingleNode

tail \*SingleNode

}

func newQueue() \*Queue {

return &Queue{}

}

func (queue \*Queue) QPUSH(cell string) {

node := &SingleNode{cell: cell, next: nil}

if queue.head == nil {

queue.head = node

queue.tail = node

} else {

queue.tail.next = node

queue.tail = node

}

}

func (queue \*Queue) QPOP() {

if queue.head == nil {

return

}

queue.head = queue.head.next

if queue.head == nil {

queue.tail = nil

}

}

func (queue \*Queue) QREAD() {

current := queue.head

if current == nil {

fmt.Println("QUEUE IS EMPTY!!!")

} else {

for current != nil {

fmt.Print(current.cell, " ")

current = current.next

}

fmt.Println()

}

}

func (queue \*Queue) WritingFromFileToStructure(filename string) {

file, err := os.Open(filename)

if err != nil {

fmt.Println("ERROR OPENING FILE:", err)

return

}

defer file.Close()

scanner := bufio.NewScanner(file)

for scanner.Scan() {

line := scanner.Text()

elements := strings.Fields(line)

for \_, element := range elements {

queue.QPUSH(element)

}

}

}

func (queue \*Queue) WritingFromStructureToFile(filename string) {

file, err := os.Create(filename)

if err != nil {

fmt.Println("ERROR CREATING FILE", err)

return

}

defer file.Close()

current := queue.head

for current != nil {

\_, err := fmt.Fprint(file, current.cell+" ")

if err != nil {

fmt.Println("ERROR WRITING TO FILE", err)

return

}

current = current.next

}

}

func (queue \*Queue)BinarySerialization(filename string) error {

file, err := os.Create(filename)

if err != nil {

return err

}

defer file.Close()

current := queue.head

for current != nil {//запись в файл. littleEndian записывает младший байт раньше

len := int32(len(current.cell)) //находим длину и записываем в 32бит = 4байта

if err := binary.Write(file, binary.LittleEndian, len); err != nil {

return err

}

\_, err := file.Write([]byte(current.cell)) //записывает байтовый срез

if err != nil {

return err

}

current = current.next

}

return nil

}

func (queue \*Queue)BinaryDEserialization(filename string) ([]string, error) { //аргументы и возвращаемый результат

file, err := os.Open(filename)

if err != nil {

return nil, err

}

defer file.Close()

var result []string

for {

var len int32

err := binary.Read(file, binary.LittleEndian, &len)

if err != nil {

if err.Error() == "EOF" {

break

}

return nil, err

}

buffer := make([]byte, len) //мэйк это выделение памяти

\_, err = file.Read(buffer)

if err != nil {

return nil, err

}

text := string(buffer)

result = append(result, text)

}

return result, nil

}

/\*

func main() {

scanner := bufio.NewScanner(os.Stdin)

queue := newQueue()

var filename string

for {

fmt.Println()

fmt.Print("Enter command ==>> ")

scanner.Scan()

command := scanner.Text()

parts := strings.Fields(command)

if parts[0] == "EXIT" {

return

}

if len(parts) > 1 {

filename = parts[1] + ".txt"

}

if \_, err := os.Stat(filename); os.IsNotExist(err) {

fmt.Println("FILE DOESNT EXIST!!!")

return

}

fmt.Print("Choose serialization:")

fmt.Println()

fmt.Print("1 - binary")

fmt.Println()

fmt.Print("2 - text")

fmt.Println()

fmt.Print("==>> ")

scanner.Scan()

serialization := scanner.Text()

if serialization == "2" {

queue.WritingFromFileToStructure(filename)

} else {

queue.BinaryDEserialization(filename)

}

switch parts[0] {

case "QPUSH":

if len(parts) == 3 {

queue.QPUSH(parts[2])

if serialization == "2" {

queue.WritingFromStructureToFile(filename)

} else {

queue.BinarySerialization(filename)

}

} else {

fmt.Println("ERROR INPUT!!")

}

case "QREAD":

if len(parts) == 2 {

queue.QREAD()

} else {

fmt.Println("ERROR INPUT!!")

}

case "QPOP":

if len(parts) == 2 {

queue.QPOP()

} else {

fmt.Println("ERROR INPUT!!")

}

if serialization == "2" {

queue.WritingFromStructureToFile(filename)

} else {

queue.BinarySerialization(filename)

}

default:

fmt.Println("Unknown command:")

}

}

}

\*/

## Функции тестирования и бенчмарка:

### main\_test.go

package main

import (

"testing"

"github.com/stretchr/testify/assert"

)

func BenchmarkQPUSH(bench \*testing.B) {

queue := newQueue()

for i := 0; i < 1000; i++ {

queue.QPUSH("temp")

}

}

func BenchmarkQPOP(bench \*testing.B) {

queue := newQueue()

for i := 0; i < 1000; i++ {

queue.QPUSH("temp")

}

bench.ResetTimer()

for i := 0; i < 1000; i++ {

queue.QPOP()

}

}

func TestQPUSH(test \*testing.T) {

queue := newQueue()

queue.QREAD()

queue.QPUSH("temp1")

queue.QPUSH("temp2")

queue.QPUSH("temp3")

assert.Equal(test, "temp1", queue.head.cell)

assert.Equal(test, "temp2", queue.head.next.cell)

assert.Equal(test, "temp3", queue.tail.cell)

queue.QREAD()

}

func TestQPOP(test \*testing.T) {

queue := newQueue()

queue.QPUSH("temp1")

queue.QPUSH("temp2")

queue.QPUSH("temp3")

assert.Equal(test, "temp1", queue.head.cell)

queue.QPOP()

assert.Equal(test, "temp2", queue.head.cell)

assert.Equal(test, "temp3", queue.tail.cell)

queue.QPOP()

queue.QPOP()

assert.Nil(test, queue.head)

assert.Nil(test, queue.tail)

}

func TestQPOP\_EMPTY(test \*testing.T) {

queue := newQueue()

queue.QPOP()

assert.Nil(test, queue.head)

assert.Nil(test, queue.tail)

}

func TestBinarySerialization(test \*testing.T) {

queue := newQueue()

queue.QPUSH("temp1")

queue.QPUSH("temp2")

queue.QPUSH("temp3")

queue.BinarySerialization("/root/notExist.txt")

err := queue.BinarySerialization("file.bin")

assert.Nil(test, err)

queue.BinaryDEserialization("notExist.bin")

result, err := queue.BinaryDEserialization("file.bin")

assert.Nil(test, err)

assert.Equal(test, []string{"temp1", "temp2", "temp3"}, result)

}

func TestFileOperations(test \*testing.T) {

queue := newQueue()

queue.QPUSH("temp1")

queue.QPUSH("temp2")

queue.WritingFromStructureToFile("/root/notExist.txt")

queue.WritingFromStructureToFile("file.txt")

newQueue := newQueue()

newQueue.WritingFromFileToStructure("notExist.txt")

newQueue.WritingFromFileToStructure("file.txt")

assert.Equal(test, "temp1", newQueue.head.cell)

assert.Equal(test, "temp2", newQueue.head.next.cell)

}

## Библиотека Testify:

### go.mod

module project\_name

go 1.18

require (

github.com/davecgh/go-spew v1.1.1 // indirect

github.com/pmezard/go-difflib v1.0.0 // indirect

github.com/stretchr/testify v1.10.0 // indirect

gopkg.in/yaml.v3 v3.0.1 // indirect

)

### go.sum

github.com/davecgh/go-spew v1.1.1 h1:vj9j/u1bqnvCEfJOwUhtlOARqs3+rkHYY13jYWTU97c=

github.com/davecgh/go-spew v1.1.1/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=

github.com/pmezard/go-difflib v1.0.0 h1:4DBwDE0NGyQoBHbLQYPwSUPoCMWR5BEzIk/f1lZbAQM=

github.com/pmezard/go-difflib v1.0.0/go.mod h1:iKH77koFhYxTK1pcRnkKkqfTogsbg7gZNVY4sRDYZ/4=

github.com/stretchr/testify v1.10.0 h1:Xv5erBjTwe/5IxqUQTdXv5kgmIvbHo3QQyRwhJsOfJA=

github.com/stretchr/testify v1.10.0/go.mod h1:r2ic/lqez/lEtzL7wO/rwa5dbSLXVDPFyf8C91i36aY=

gopkg.in/check.v1 v0.0.0-20161208181325-20d25e280405/go.mod h1:Co6ibVJAznAaIkqp8huTwlJQCZ016jof/cbN4VW5Yz0=

gopkg.in/yaml.v3 v3.0.1 h1:fxVm/GzAzEWqLHuvctI91KS9hhNmmWOoWu0XTYJS7CA=

gopkg.in/yaml.v3 v3.0.1/go.mod h1:K4uyk7z7BCEPqu6E+C64Yfv1cQ7kz7rIZviUmN+EgEM=

# Примеры работы программы

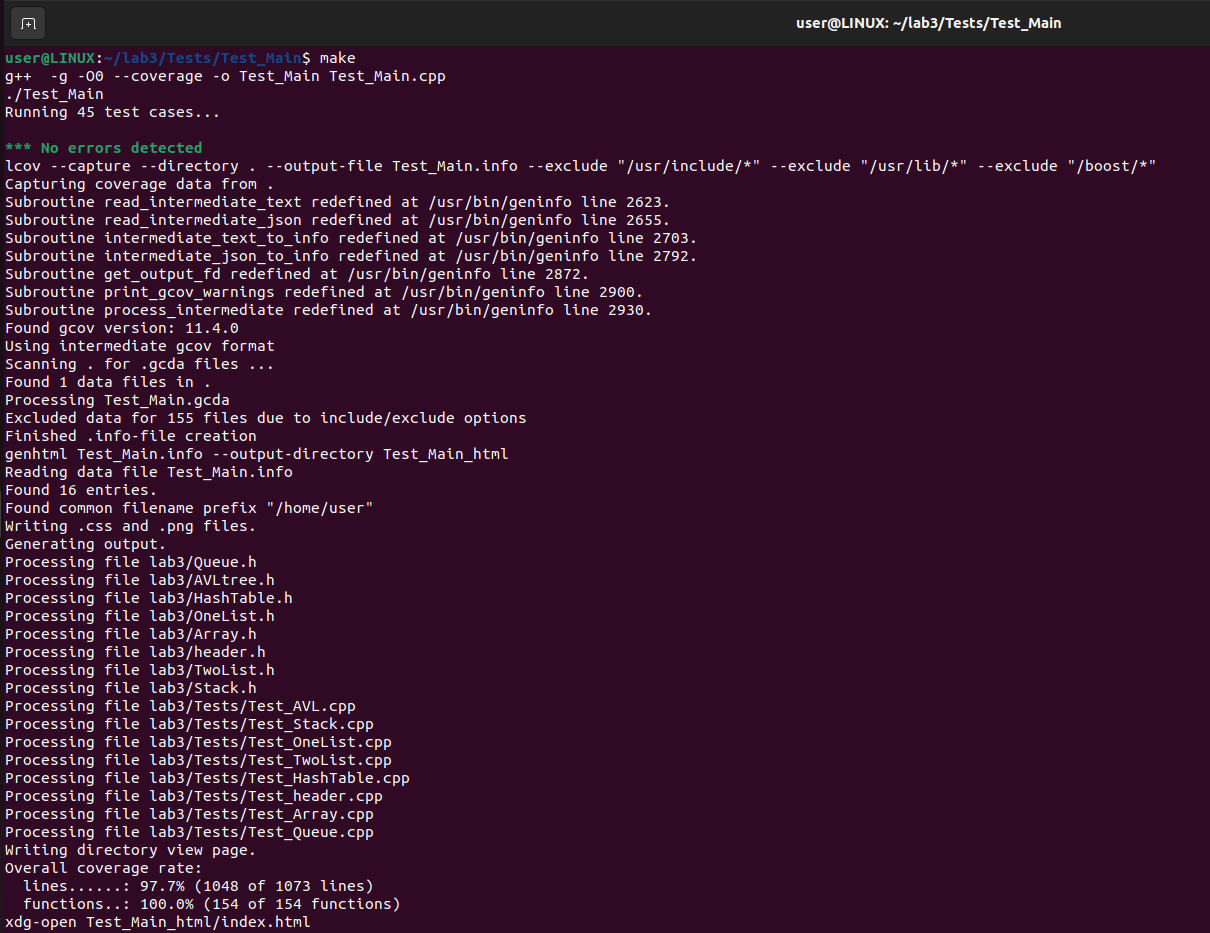


Рисунок 1. Запуск команд для тестирования С++.

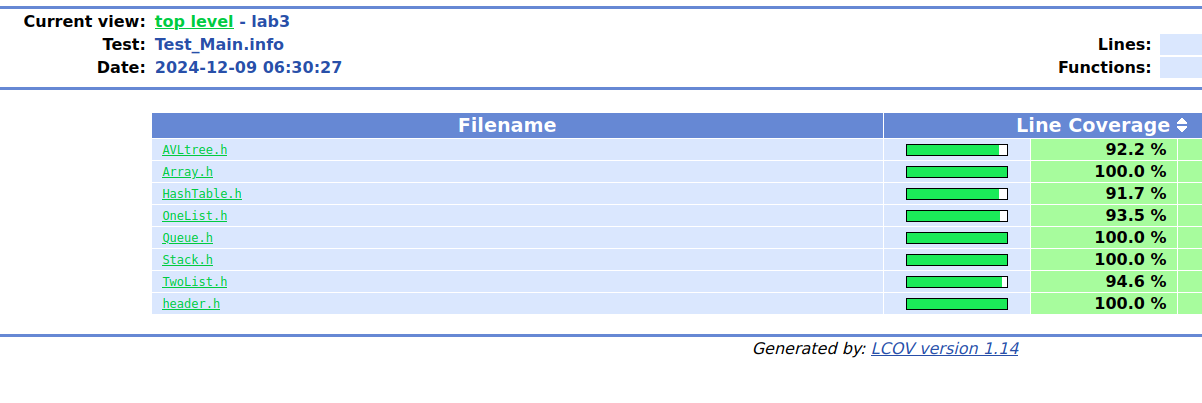


Рисунок 2. Результат покрытия тестами классов С++.

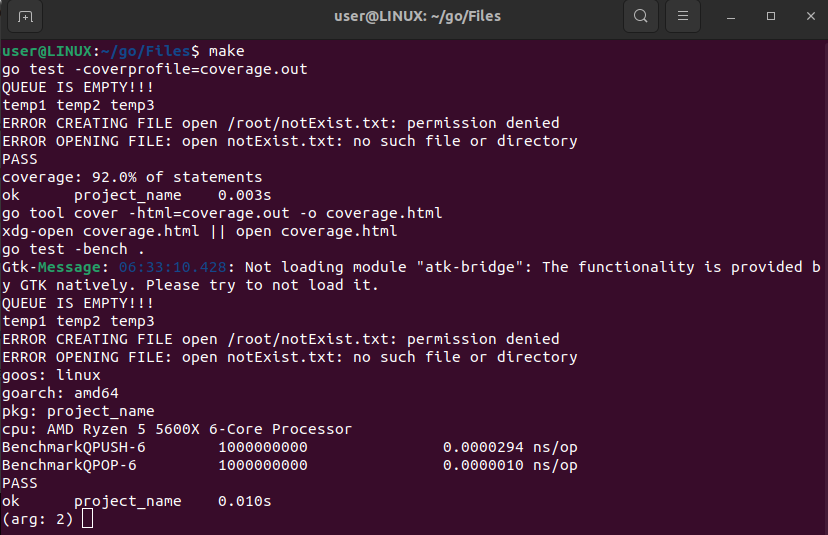


Рисунок 3. Запуск команд для тестирования и бенчмарка Go.

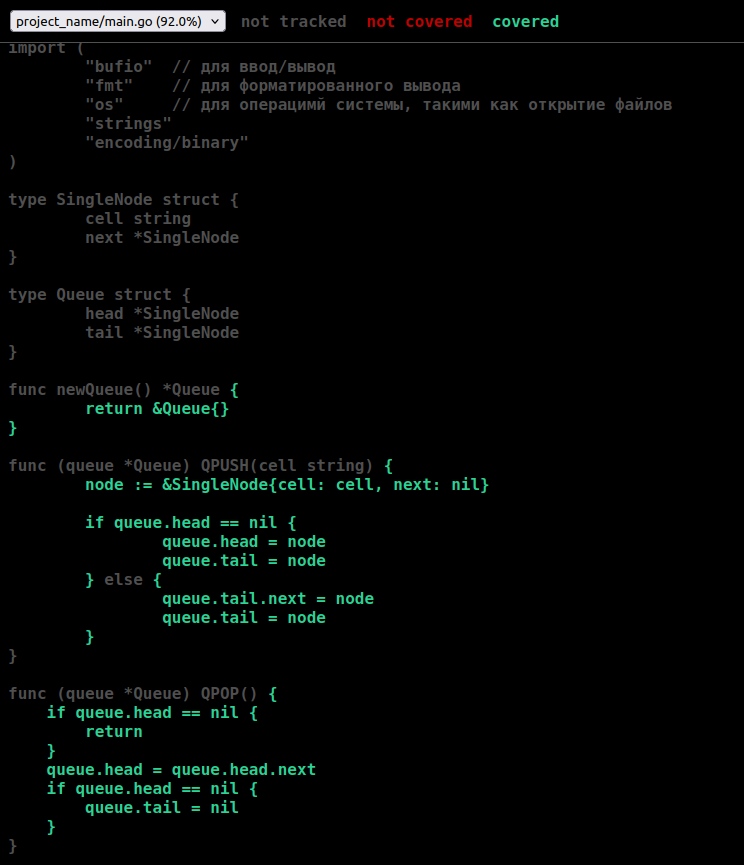


Рисунок 4. Результат покрытия тестами структуры Go.

# UML-диаграммы

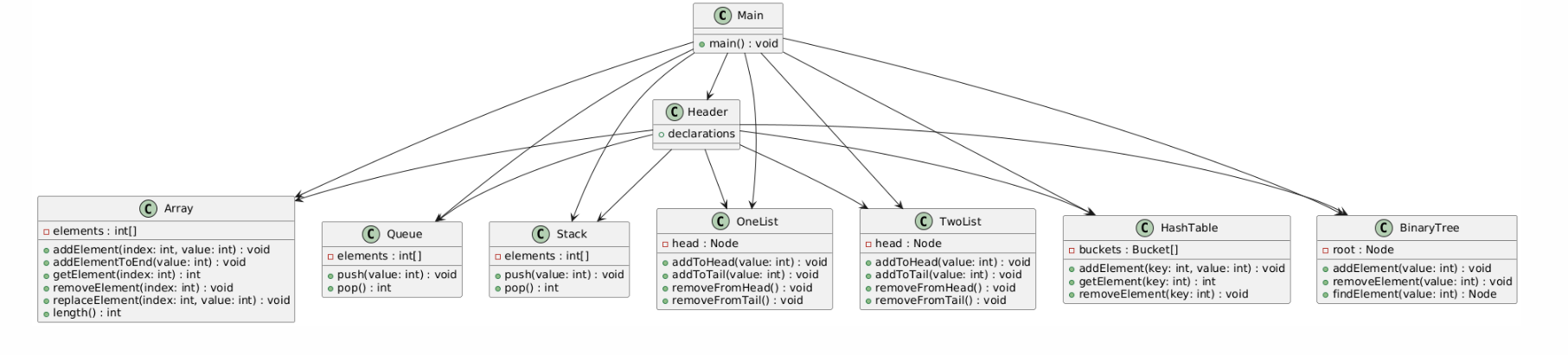
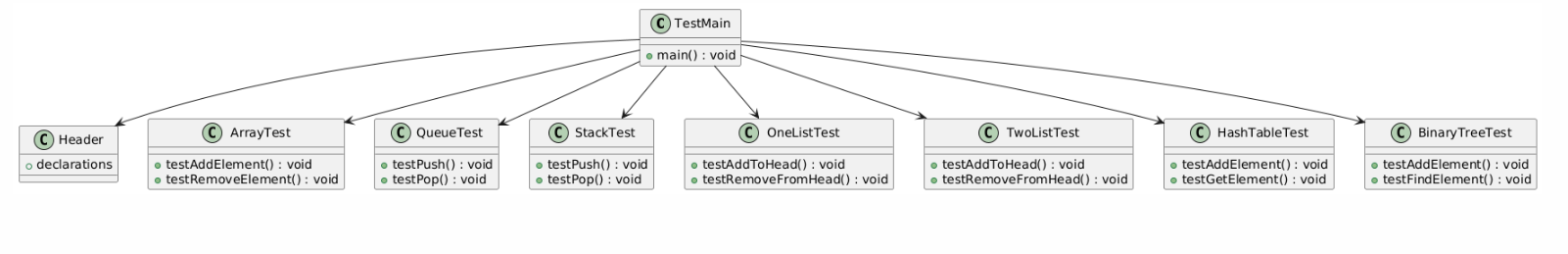


Рисунок 1. UML-диаграмма для классов С++.

 Рисунок 2. UML-диаграмма для тестирования С++.

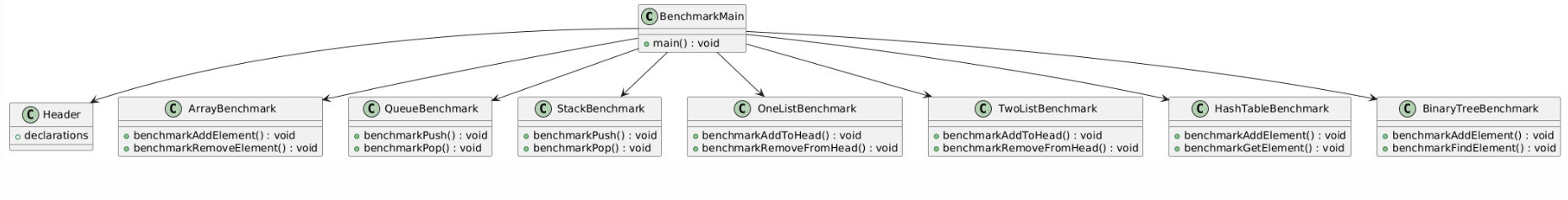


Рисунок 3. UML-диаграмма для бенчмарка С++.

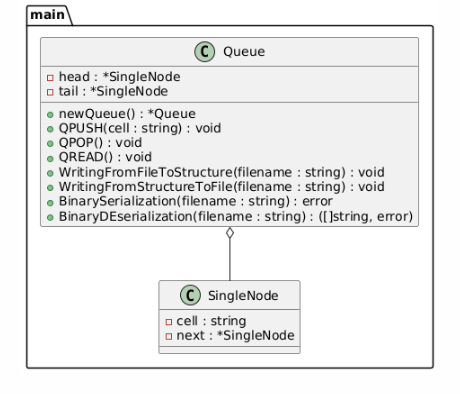


Рисунок 4. UML-диаграмма для класса Go.

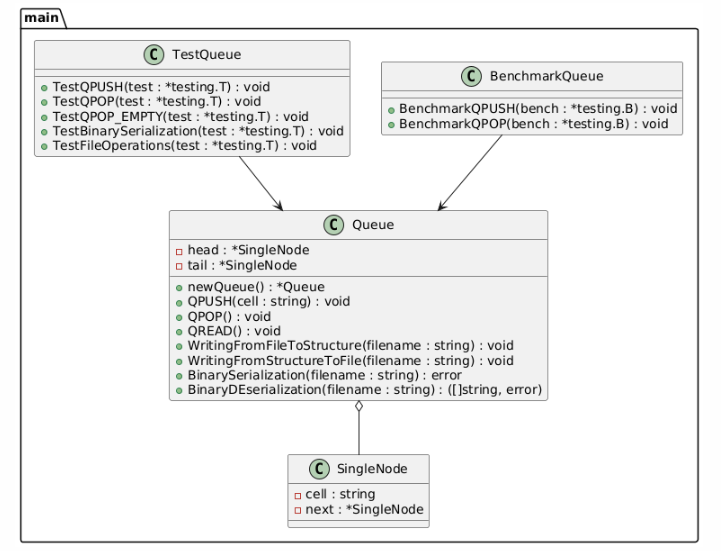


Рисунок 5. UML-диаграмма для тестирования и бенчмарка Go.

# Основные виды тестирования

Тестирование — это проверка программного обеспечения, которая показывает, соответствует ли оно ожиданиям разработчиков и правильно ли работает. Тестирование проводят тестировщики — они отвечают за обеспечение качества, контролируют его и проверяют, что продукт соответствует всем заданным требованиям. Процесс работы над продуктом включает в себя множество этапов: от проработки идеи и расчета эффективности до самой разработки и выпуска. И в этом процессе участвует множество людей: аналитики, руководители проекта, разработчики, дизайнеры.

Представьте, что все эти люди объединяются, чтобы создать какой-то продукт. Они разрабатывают его, выкатывают на прод. А позже пользователи вдруг выясняют, что где-то в продукте есть баги. В результате команде придется заново его прорабатывать, что стоит немалых денег и времени, да и репутация продукта на рынке будет испорчена.

Устранить ошибки можно заранее, доверив эту работу тестировщикам. Они должны участвовать во всем цикле создания программного обеспечения: от появления требований к проекту до момента сопровождения самого ПО.

**Этапы тестирования**

Как правило, в большинстве проектов этапы тестирования схожи. Давайте по очереди их разберем.

**Проработка требований к продукту**

На этом этапе тестировщики внимательно изучают требования продукта — это могут быть документы, спецификации, описание того, как пользователь взаимодействует с продуктом (по-другому это называют пользовательскими сценариями). Четкое понимание требований помогает определить области, которые нужно протестировать.

**Анализ требований**

Анализ требований позволяет выяснить, какие возможные риски или сложности могут возникнуть при тестировании. Также на этом этапе можно выявить возможные несоответствия или недостаточно ясные требования, которые требуют уточнения у разработчиков или заказчика.

**Разработка стратегии и плана тестирования**

Когда все требования к продукту понятны, остается разработать план тестирования. В него входит:

* Выбор методов тестирования — ручное, автоматизированное, тестирование на реальных устройствах и другие.
* Анализ потенциальных рисков, которые могут повлиять на качество и успешность тестирования, и планирование мер по их минимизации.
* Планирование ресурсов — кто будет тестировать продукт, каким оборудованием и инструментами можно при этом пользоваться и сколько времени займет тестирование, к какому сроку оно должно быть закончено.

**Создание тестовой документации**

На этом этапе на основе требований и анализа тестировщики создают тестовые случаи, тест-планы, отчетность и другую документацию, которая будет использоваться во время тестирования. Тестовая документация определяет, какие тесты будут проведены, как будут собраны результаты и как будет оценено качество ПО.

**Тестирование**

После того как команда утверждает стратегию тестирования и тестовую документацию, проводится тестирование. Тестирование программного обеспечения — это длительный и обширный процесс. По ходу составляются отчеты о выявленных недостатках, проводится набор тестовых сценариев, создается тестовая среда и выполняется тестирование согласно заранее задокументированным видам тестов, описанным в тестовой документации.

**Эксплуатация и поддержка**

После того как разработчики устраняют дефекты и выпускают продукт, тестировщик переходит к тестированию продукта в рабочей среде. Важно отметить, что на этом этапе не только происходит релиз продукта, но и начинается пост-релизовая поддержка. Невозможно предусмотреть все особенности использования и окружение, в котором будет работать продукт. Поэтому на данном этапе акцент делается на обратной связи пользователей. Теперь они становятся главными тестировщиками, а продукт становится частью их повседневной жизни. Устранение дефектов и поиск ошибок проводится быстро, но тщательно.

**Виды тестирования**

В своей работе тестировщики используют различные виды и методы тестирования, а также прорабатывают сценарии, в которых продукт может оказаться. Есть много способов тестирования, по разным оценкам в среднем их больше 30. Выбор способов зависит от программного продукта, требований заказчиков и самой команды, которая разрабатывает продукт. Происходит это на этапе разработки стратегии и планов тестирования: заказчик или его представитель — владелец продукта или проджект-менеджер — включает в план тот или иной вид тестирования.

Далее к проекту привлекают тестировщиков, которые специализируются на выбранном методе тестирования. Существуют фулстек-тестировщики, которые умеют применять в проекте все виды тестирования. Но чаще всего компании выбирают более узкоспециализированных специалистов — как правило, их знания глубже в каком-то одном из способов. И также компании выбирают тестировщиков под сами требования проекта.

**По характеру сценариев**

Сценарий в тестировании — это описание того, как пользователь будет взаимодействовать с готовым продуктом. В эту группу входят два вида тестирования: позитивных сценариев и негативных.

* **Тестирование позитивных сценариев** проверяет, как должна работать программа в нормальных условиях.
* **Тестирование негативных сценариев** проверяет, как программа ведет себя в необычных или некорректных ситуациях.

**По критериям запуска программы или кода**

Критерии запуска программы или кода означают условия, которые необходимо выполнить для запуска тестов. Под эту группу подходит статическое тестирование и динамическое тестирование.

* **Статическое тестирование** — это вид проверки программного обеспечения, который выполняется без запуска программы.
* **Динамическое тестирование** — это вид проверки программного обеспечения, который выполняется во время работы программы.

**По степени автоматизации тестирования**

* **Ручное тестирование** — это проверка программного обеспечения вручную, без использования автоматизированных инструментов.
* **Автоматизированное тестирование** — это проверка программного обеспечения с использованием специальных программных инструментов.

**По объектам тестирования**

* **Функциональное тестирование** проверяет соответствие программы или системы заранее определенным функциональным требованиям и ожиданиям. Оно делится на:
  + Unit-тестирование
  + Интеграционное тестирование
  + Системное тестирование
  + Приемочное тестирование
* **Нефункциональное тестирование** проверяет нефункциональные аспекты программы: производительность, безопасность, надежность, масштабируемость и совместимость.

**По степени знания системы**

* **Тестирование «черного ящика»** — это способ проверки программного обеспечения, когда тестировщик не знает внутренней структуры или деталей работы программы.
* **Тестирование «белого ящика»** — предполагает, что тестировщик имеет доступ к внутренней структуре и коду программы.
* **Тестирование «серого ящика»** — это комбинация тестирования «черного ящика» и «белого ящика».

**По времени проведения тестирования**

* **Альфа-тестирование** — это этап тестирования перед официальным выпуском.
* **Дымовое тестирование** — быстрая проверка программного обеспечения после внесения изменений.
* **Регрессионное тестирование** — проверка новых изменений и функционала.
* **Приемочное тестирование** — проверка заказчиком или пользователем.
* **Бета-тестирование** — предоставление программы ограниченной группе реальных пользователей.

**Вывод**

Тестировщики играют важную роль в разработке программного обеспечения, проверяя его на ошибки и убеждаясь, что оно работает правильно. Чтобы протестировать продукт, сначала нужно изучить его требования, проанализировать их, разработать стратегию тестирования и выбрать методы. В зависимости от типа продукта тестировщики используют различные подходы и виды тестирования.

# Список литературы

1. Яковлева Е. Фундаментальная теория тестирования / Яковлева Е. [Электронный ресурс] // Хабр : [сайт]. — URL: https://habr.com/ru/articles/549054/ (дата обращения: 09.12.2024).
2. Тестирование программного обеспечения / [Электронный ресурс] // Википедия : [сайт]. — URL: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5\_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE\_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F (дата обращения: 09.12.2024).