Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский технический университет связи и информатики»

Кафедра информатики

Отчёт по лабораторной работе №4

на тему «Методы поиска подстроки в строке»

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент группы БВТ1903

Щитов В.М.

Руководитель: Павликов А.Е.

Содержание

[1. Задание 3](#_Toc72401080)

[2. Ход работы 6](#_Toc72401081)

[3. Вывод 28](#_Toc72401082)

# 1. Задание

Реализовать следующие структуры данных:

– **стек (stack)**: операции *для стека*: инициализация, проверка на пустоту, добавление нового элемента в начало, извлечение элемента из начала;

– **дек (двусторонняя очередь, deque)**: операции *для дека*: инициализация, проверка на пустоту, добавление нового элемента в начало, добавление нового элемента в конец, извлечение элемента из начала, извлечение элемента из конца.

Разработать программу обработки данных, содержащихся в заранее подготовленном txt-файле, в соответствии с заданиями, применив указанную в задании структуру данных. Результат работы программы вывести на экран и сохранить в отдельном txt-файле.

Задания:

1. Отсортировать строки файла, содержащие названия книг, в алфавитном порядке с использованием двух ***деков***.
2. ***Дек*** содержит последовательность символов для шифровки сообщений. Дан текстовый файл, содержащий зашифрованное сообщение. Пользуясь ***деком***, расшифровать текст. Известно, что при шифровке каждый символ сообщения заменялся следующим за ним в ***деке*** по часовой стрелке через один.
3. Даны три стержня и *n* дисков различного размера. Диски можно надевать на стержни, образуя из них башни. Перенести *n* дисков со стержня *А* на стержень *С*, сохранив их первоначальный порядок. При переносе дисков необходимо соблюдать следующие правила:

– на каждом шаге со стержня на стержень переносить только один диск;

– диск нельзя помещать на диск меньшего размера;

– для промежуточного хранения можно использовать стержень *В*.

Реализовать алгоритм, используя три ***стека*** вместо стержней *А*, *В*, *С*. Информация о дисках хранится в исходном файле.

1. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс круглых скобок в тексте, используя ***стек***.
2. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс квадратных скобок в тексте, используя ***дек***.
3. Дан файл из символов. Используя ***стек***, за один просмотр файла напечатать сначала все цифры, затем все буквы, и, наконец, все остальные символы, сохраняя исходный порядок в каждой группе символов.
4. Дан файл из целых чисел. Используя ***дек***, за один просмотр файла напечатать сначала все отрицательные числа, затем все положительные числа, сохраняя исходный порядок в каждой группе.
5. Дан текстовый файл. Используя ***стек***, сформировать новый текстовый файл, содержащий строки исходного файла, записанные в обратном порядке: первая строка становится последней, вторая – предпоследней и т.д.
6. Дан текстовый файл. Используя ***стек***, вычислить значение логического выражения, записанного в текстовом файле в следующей форме:

<ЛВ> ::= T | F | (N<ЛВ>) | (<ЛВ>A<ЛВ>) | (<ЛВ>X<ЛВ>) | (<ЛВ>O<ЛВ>), где буквами обозначены логические константы и операции:

T – True, F – False, N – Not, A – And, X – Xor, O – Or.

1. Дан текстовый файл. В текстовом файле записана формула следующего вида:

<Формула> ::= <Цифра> | M(<Формула>, <Формула>) | N(Формула>, <Формула>)

<Цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

где буквами обозначены функции: M – определение максимума, N – определение минимума. Используя ***стек***, вычислить значение заданного выражения.

1. Дан текстовый файл. Используя ***стек***, проверить, является ли содержимое текстового файла правильной записью формулы вида:

<Формула> ::= <Терм> | <Терм> + <Формула> | <Терм> - <Формула>

<Терм> ::= <Имя> | (<Формула>)

<Имя> ::= x | y | z

# 2. Ход работы

Язык программирования, используемый для выполнения работы: C++ (используется стандарт C++14). Для выполнения поставленных задач было создано решение в среде разработки MVS2015, включающее проект «structures-and-algos», исполняемый код которого представлен в файлах Stack.h, Deque.h, Lab4Tasks.h и Lab4.h, листинг которых представлен ниже.

Листинг файла Stack.h:

#ifndef STACK\_H

#define STACK\_H

#include <iostream>

#include <vector>

// Класс для стека

template <class Type> class MyStack {

// Защищённые члены классы

protected:

unsigned int size;

std::vector<Type> stack;

// Публичные члены класса

public:

// Конструктор класса

MyStack() {

this->size = 0;

}

// Метод для проверки содержимого стека

bool isEmpty() {

return (stack.size() == 0);

}

// Метод для очистки стека

void clear() {

stack.clear();

this->size = 0;

}

// Метод возвращает количество элементов в стеке

unsigned int getSize() {

return this->size;

}

// Метод добавления элемента в стек

void push(Type element) {

stack.push\_back(element);

this->size += 1;

}

// Метод возвращает значение с вершины стека

Type peek() {

if (this->isEmpty()) {

std::cout << "Stack is empty! Error!" << std::endl;

return (Type)0;

}

else return stack.back();

}

// Метод считывания и удаления элемента из стека

Type pop() {

if (this->isEmpty()) {

std::cout << "Stack is empty! Error!" << std::endl;

return (Type)0;

}

else {

Type ret = stack.back();

stack.pop\_back();

this->size -= 1;

return ret;

}

}

// Метод для вывода элементов стека в консоль

void print() {

if (this->isEmpty()) std::cout << "Stack is empty!" << std::endl;

else {

for (size\_t i = 0; i < this->size; i++) {

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

}

std::cout << std::endl;

}

}

};

#endif STACK\_H

Листинг файла Deque.h:

#ifndef DEQUE\_H

#define DEQUE\_H

#include <iostream>

#include <vector>

// Класс для дека

template <class Type> class MyDeque {

// Защищённые члены классы

protected:

unsigned int size;

std::vector<Type> deque;

// Публичные члены класса

public:

// Конструктор класса

MyDeque() {

this->size = 0;

}

// Метод для проверки содержимого дека

bool isEmpty() {

return (deque.size() == 0);

}

// Метод для очистки дека

void clear() {

deque.clear();

this->size = 0;

}

// Метод возвращает количество элементов в деке

unsigned int getSize() {

return this->size;

}

// Метод добавления элемента в начало дека

void pushFront(Type element) {

if (this->size == 0) this->deque.push\_back(element);

else this->deque.insert(this->deque.begin(), element);

this->size += 1;

}

// Метод удаления эелеента из начала дека

Type popFront() {

if (this->isEmpty()) {

std::cout << "Deque is empty! Error!" << std::endl;

return (Type)0;

}

else {

Type ret = deque[0];

deque.erase(deque.begin());

std::vector<Type>(deque).swap(deque);

this->size -= 1;

return ret;

}

}

// Метод добавления элемента в конец дека

void pushBack(Type element) {

deque.push\_back(element);

this->size += 1;

}

// Метод считывания и удаления элемента из конца дека

Type popBack() {

if (this->isEmpty()) {

std::cout << "Deque is empty! Error!" << std::endl;

return (Type)0;

}

else {

Type ret = deque.back();

deque.pop\_back();

this->size -= 1;

return ret;

}

}

// Метод для вывода элементов дека в консоль

void print() {

if (this->isEmpty()) std::cout << "Deque is empty!" << std::endl;

else {

for (size\_t i = 0; i < this->size; i++) {

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

}

std::cout << std::endl;

}

}

};

#endif DEQUE\_H

Листинг файла Lab4Tasks.h:

#ifndef LAB4TASKS\_H

#define LAB4TASKS\_H

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include "GetTime.h"

#include "Deque.h"

#include "Stack.h"

using namespace std;

// Пространство имён 4 лабораторной работы

namespace lab4 {

// Объявления функций

vector<string> readFile(char\* pathFile);

void Task01(char\* pathfile);

bool compareString(string in1, string in2);

void Task02();

string encode(string input, MyDeque<char>\* cipher);

string decode(string input, MyDeque<char>\* cipher);

void Task03();

void HanoiSolver(MyStack<size\_t>\* from, MyStack<size\_t>\* buf, MyStack<size\_t>\* to, size\_t count);

void Task04();

bool Task04Calc(char\* pathfile);

void Task05();

bool Task05Calc(char\* pathfile);

void Task06();

bool Task06Calc(char\* pathfile, MyStack<char>\* arr);

bool isDigit(char sym);

bool isCharacter(char sym);

void Task07();

vector<string> separate(string str, string separator);

void Task08();

void Task09();

bool Task09Calc(string input);

bool isBoolean(char sym);

int getPriority(char sym);

char operationAnd(MyStack<char>\* stack);

char operationOr(MyStack<char>\* stack);

char operationXor(MyStack<char>\* stack);

char operationDeny(MyStack<char>\* stack);

void Task10();

int Task10Calc(string input);

int max(MyStack<char>\* stack);

int min(MyStack<char>\* stack);

void Task11();

bool Task11Calc(string input);

// Чтение из файла массива строк

vector<string> readFile(char\* pathFile) {

string temp;

vector <string> result;

ifstream in(pathFile);

while (getline(in, temp)) result.push\_back(temp);

in.close();

return result;

}

// Функция сравнения двух строк, возвращает true, если сначала должна быть первая строка, false - если вторая

bool compareString(string in1, string in2) {

// in1 - сверяемая строка, in2 - temp sorted строка

size\_t size = 0;

if (in1.length() <= in2.length()) size = in1.length();

else size = in2.length();

// Основной цикл. Посимвольно сравниваем строки с использованием переменной-флага

bool flag = true;

size\_t counter = 0;

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

// Если первая строка по индексу должна быть перед второй

if ((int)in1[i] < (int)in2[i]) break;

// Если первая строка должна быть после второй

else if ((int)in1[i] > (int)in2[i]) {

flag = false;

return flag;

}

// Если символы совпадают

else {

counter += 1;

continue;

}

}

// Если длина первой строки больше второй, но символы совпали, то она должна идти второй, flag = false

if (flag && (counter == size) && (in1.length() > in2.length())) flag = false;

// После прохождения предыдущих проверок возвращаем флаг

return flag;

}

// Задание 1

void Task01(char\* pathfile) {

// Считывание файла и его проверка

vector<string> file = readFile(pathfile);

if (file.size() == 0) {

cout << "Введён пустой файл!\n\n";

return;

}

// Основной цикл

clock\_t start = clock(), end;

MyDeque<string> result, temp\_field;

result.pushFront(file[0]);

size\_t file\_size = file.size();

for (size\_t i = 1; i < file\_size; i++) {

// Вытаскиваем из файла i-ую строку

string temp = file[i];

string sorted = result.popFront();

// Если строка должен быть в начале дека

if (compareString(temp, sorted)) {

// Возвращаем элементы в нужной последовательности

result.pushFront(sorted);

result.pushFront(temp);

}

else {

temp\_field.pushBack(sorted);

// Если дек непустой

if (result.getSize() > 0) {

// Пока в деке есть элементы или не найден нужный элемент

while (result.getSize() > 0) {

sorted = result.popFront();

if (!compareString(temp, sorted)) temp\_field.pushBack(sorted);

else {

temp\_field.pushBack(sorted);

break;

}

}

// Строка найдена или подошли к концу дека

result.pushFront(temp);

// Возвращаем всё на место из временного дека

while (temp\_field.getSize() > 0) {

result.pushFront(temp\_field.popBack());

}

}

// Если дек оказался пустым, возвращаем все элементы в нужном порядке

else {

result.pushFront(temp);

result.pushFront(temp\_field.popBack());

}

}

}

end = clock();

// Вывод времени выполнения

size\_t result\_size = result.getSize();

cout << "Время сортировки двумя деками " << result\_size << " элементов: " << getTime(start, end) << " sec.\n";

// Вывод отсортированного дека

cout << "Отсортированный файл:\n";

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

cout << result.popFront() << "\n";

}

cout << endl;

}

// Функция для зашифровки

string encode(string input, MyDeque<char>\* cipher) {

string result = "";

for (size\_t i = 0; i < input.length(); i++) {

// Если символ пробел или перенос строки

if (input[i] == ' ' || input[i] == '\n') result = result + input[i];

else {

char sym = cipher->popFront();

// Пока не найден символ

while (sym != input[i]) {

cipher->pushBack(sym);

sym = cipher->popFront();

}

// Символ найден, возвращаем его в дек

cipher->pushBack(sym);

// Пропускаем один символ

cipher->pushBack(cipher->popFront());

// Получаем шифруемый символ, используем его и возвращаем в дек

sym = cipher->popFront();

result = result + sym;

cipher->pushBack(sym);

}

}

return result;

}

// Функция для расширофвки

string decode(string input, MyDeque<char>\* cipher) {

string result = "";

for (size\_t i = 0; i < input.length(); i++) {

// Если символ пробел или перенос строки

if (input[i] == ' ' || input[i] == '\n') result = result + input[i];

else {

char sym = cipher->popFront();

// Пока не найден символ

while (sym != input[i]) {

cipher->pushBack(sym);

sym = cipher->popFront();

}

// Символ найден, возвращаем его в дек

cipher->pushFront(sym);

// Пропускаем один символ

cipher->pushFront(cipher->popBack());

// Получаем шифруемый символ, используем его и возвращаем в дек

sym = cipher->popBack();

result = result + sym;

cipher->pushFront(sym);

}

}

return result;

}

// Задание 2

void Task02() {

// Считывание файла и его проверка, создание результирующего текста

vector<string> file = readFile("Lab4/Task01.txt");

if (file.size() == 0) {

cout << "Введён пустой файл!\n\n";

return;

}

string text = "", encoded, decoded;

for (size\_t i = 0; i < file.size(); i++) text = text + (file[i] + "\n");

// Формируем дек для шифрования

MyDeque<char> cipher;

for (size\_t i = 0; i < 256; i++) cipher.pushBack((char)i);

// Шифруем и замеряем время

clock\_t start, end;

start = clock();

encoded = encode(text, &cipher);

end = clock();

cout << "Время шифрования текста деком: " << getTime(start, end) << " sec.\n";

cout << "Результат шифрования:\n" << encoded << "\n";

// Расшифровываем и замеряем время

start = clock();

decoded = decode(encoded, &cipher);

end = clock();

cout << "Время расшифрования текста деком: " << getTime(start, end) << " sec.\n";

cout << "Результат расшифрования:\n" << decoded << "\n\n";

}

// Функция для решения загадки о Ханойских башнях

void HanoiSolver(MyStack<size\_t>\* from, MyStack<size\_t>\* buf, MyStack<size\_t>\* to, size\_t count) {

if (count != 0) {

HanoiSolver(from, to, buf, count - 1);

to->push(from->pop());

HanoiSolver(buf, from, to, count - 1);

}

}

// Задание 3

void Task03() {

// Ввод количества колец на первом стержне, заполнение первого стека

size\_t num;

cout << "Введите количество колец на первом стержне: ";

cin >> num;

MyStack<size\_t> first, second, third;

for (size\_t i = 1; i <= num; i++) first.push(i);

// Вывод содержимого стеков перед операцией перемещения

cout << "\nИзначальное содержимое стеков:\n";

cout << "First: "; first.print(); cout << "Second: "; second.print(); cout << "Third: "; third.print();

// Выполнение операции по решению задачи, замеры времени

clock\_t start = clock(), end;

HanoiSolver(&first, &second, &third, num);

end = clock();

// Вывод содержимого стеков после операции перемещения и времени выполнения

cout << "\nСодержимое стеков после перемещения:\n";

cout << "First: "; first.print(); cout << "Second: "; second.print(); cout << "Third: "; third.print();

cout << "Время выполнения операции перемещения: " << getTime(start, end) << " sec.\n\n";

}

// Основная функция для решения 4 задачи

bool Task04Calc(char\* pathfile) {

// Считывание файла и его проверка, создание результирующей строки с текстом

vector<string> file = readFile(pathfile);

if (file.size() == 0) {

cout << "Введён пустой файл!\n";

return false;

}

string text = "";

for (size\_t i = 0; i < file.size(); i++) text = text + (file[i] + "\n");

// Добавляем и удаляем элементы из стека, соблюдая баланс

MyStack<char> stack;

for (size\_t i = 0; i < text.length(); i++) {

if (text[i] == '(') stack.push('(');

else if (text[i] == ')') {

if (stack.getSize() > 0) stack.pop();

else return false;

}

}

// Финальная проверка

if (stack.getSize() > 0) return false;

else return true;

}

// Задание 4

void Task04() {

// Выполнение функции и вывод результатов

bool flag = Task04Calc("Lab4/Deque.h");

cout << "Результат для файла \"Lab4/Deque.h\": ";

if (flag) cout << "true.\n\n";

else cout << "false.\n\n";

}

// Основная функция для решения 5 задачи

bool Task05Calc(char\* pathfile) {

// Считывание файла и его проверка, создание результирующей строки с текстом

vector<string> file = readFile(pathfile);

if (file.size() == 0) {

cout << "Введён пустой файл!\n";

return false;

}

string text = "";

for (size\_t i = 0; i < file.size(); i++) text = text + (file[i] + "\n");

// Добавляем элементы в дек

MyDeque<char> deque;

for (size\_t i = 0; i < text.length(); i++) {

if (text[i] == '[') deque.pushFront('[');

else if (text[i] == ']') deque.pushBack(']');

}

// Проверка элементов в деке

while (deque.getSize() > 1) {

if (!(deque.popFront() == '[' && deque.popBack() == ']')) return false;

}

if (deque.isEmpty()) return true;

else return false;

}

// Задание 5

void Task05() {

// Выполнение функции и вывод результатов

bool flag = Task05Calc("Lab4/Deque.h");

cout << "Результат для файла \"Lab4/Deque.h\": ";

if (flag) cout << "true.\n\n";

else cout << "false.\n\n";

}

// Проверка на то, является ли символ цифрой

bool isDigit(char sym) {

return ((int)sym >= 48 && (int)sym <= 57);

}

// Проверка на то, является ли символ буквой английского алфавита

bool isCharacter(char sym) {

return (((int)sym >= 65 && (int)sym <= 90) || ((int)sym >= 97 && (int)sym <= 122));

}

// Основная функция для решения 6 задачи

bool Task06Calc(char\* pathfile, MyStack<char>\* arr) {

// Считывание файла и его проверка, создание результирующей строки с текстом

vector<string> file = readFile(pathfile);

if (file.size() == 0) {

cout << "Введён пустой файл!\n\n";

return false;

}

string text = "";

for (size\_t i = 0; i < file.size(); i++) text = text + (file[i] + "\n");

// Заполнение 3 стеков

for (int i = text.length() - 1; i >= 0; i--) {

if (isDigit(text[i])) arr[0].push(text[i]);

else if (isCharacter(text[i])) arr[1].push(text[i]);

else arr[2].push(text[i]);

}

return true;

}

// Задание 6

void Task06() {

// Выполнение функции и вывод результатов

const size\_t arr\_size = 3;

MyStack<char>\* syms = new MyStack<char>[arr\_size];

bool flag = Task06Calc("Lab4/Deque.h", syms);

// Если функция успешно выполнилась, выводим поочерёдно содержимое стеков

if (flag) {

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

while (!syms[i].isEmpty()) cout << syms[i].pop();

}

cout << endl;

delete[] syms;

}

// В противном случае выходим из функции

else {

delete[] syms;

return;

}

}

// Функция для получения массива подстрок из строки через указанный разделитель

vector<string> separate(string str, string separator) {

vector<string> arr;

size\_t prev = 0;

size\_t next;

size\_t delta = separator.length();

while ((next = str.find(separator, prev)) != string::npos) {

arr.push\_back(str.substr(prev, next - prev));

prev = next + delta;

}

arr.push\_back(str.substr(prev));

return arr;

}

// Задание 7

void Task07() {

// Считывание файла и его проверка, создание результирующей строки с текстом

vector<string> file = readFile("Lab4/Task07.txt");

if (file.size() == 0) {

cout << "Введён пустой файл!\n\n";

return;

}

string text = "";

for (size\_t i = 0; i < file.size(); i++) text = text + (file[i] + " ");

// Повторно используем массив file, пишем туда подстроки через разделитель " "

file = separate(text, " ");

// Циклично конвертируем значения в числовой тип, затем кладём их в дек

MyDeque<int> deque;

// Сначала все отрицательные числа

for (size\_t i = 0; i < file.size() - 1; i++) {

int num = std::stoi(file[i]);

if (num < 0) deque.pushBack(num);

}

// Затем все положительные

for (size\_t i = 0; i < file.size() - 1; i++) {

int num = std::stoi(file[i]);

if (num >= 0) deque.pushBack(num);

}

// Вывод элементов

while (!deque.isEmpty()) cout << deque.popFront() << " ";

cout << "\n\n";

}

// Задание 8

void Task08() {

// Считывание файла и его проверка

vector<string> file = readFile("Lab4/Deque.h");

if (file.size() == 0) {

cout << "Введён пустой файл!\n\n";

return;

}

// Создаём стек и пушим туда все строки из файла

MyStack<string> stack;

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

stack.push(file[i]);

}

// Затем выводим все строки из стека

while (!stack.isEmpty()) cout << stack.pop() << "\n";

cout << endl;

}

// Проверка на то, является ли символ булевым значением

bool isBoolean(char sym) {

return (sym == 'T' || sym == 'F');

}

// Функция для получения приоритета операции

int getPriority(char sym) {

switch (sym) {

case '(':

return 3;

case 'N':

case 'A':

return 2;

case 'O':

case 'X':

return 1;

default:

break;

}

return 0;

}

// Функция для операции "И"

char operationAnd(MyStack<char>\* stack) {

char first = stack->pop();

char second = stack->pop();

if (first == 'F' || second == 'F') return 'F';

else return 'T';

}

// Функция для операции "ИЛИ"

char operationOr(MyStack<char>\* stack) {

char first = stack->pop();

char second = stack->pop();

if (first == 'F' && second == 'F') return 'F';

else return 'T';

}

// Функция для операции "ИСКЛЮЧАЮЩЕЕ ИЛИ"

char operationXor(MyStack<char>\* stack) {

char first = stack->pop();

char second = stack->pop();

if (first == 'F' && second == 'F' || second == 'T' && first == 'T') return 'F';

else return 'T';

}

// Функция для операции "НЕ"

char operationDeny(MyStack<char>\* stack) {

char first = stack->pop();

if (first == 'F') return 'T';

else return 'F';

}

// Основная функция для решения логической последовательности в задании 9

bool Task09Calc(string input) {

// Начальная проверка

if (input.length() == 0) {

cout << "Введена пустая строка!\n";

return false;

}

string output = "";

MyStack<char> stack;

// Заносим операции в стек

for (size\_t i = 0; i < input.length(); i++) {

char sym = input[i];

if (isBoolean(sym)) output += sym;

else {

switch (sym) {

case '(':

break;

case ')':

break;

case 'A':

case 'O':

case 'X':

case 'N':

if (stack.isEmpty()) stack.push(sym);

else {

while (stack.getSize() > 0 && (getPriority(sym) <= stack.peek())) {

if ('(' == stack.peek()) break;

else output += stack.pop();

}

stack.push(sym);

}

break;

default:

cout << "Некорректный синтаксис выражения.\n";

return false;

break;

}

}

}

// Выполнение операций из стека

while (!stack.isEmpty()) output += stack.pop();

while (!isBoolean(output[output.length() - 1])) {

if (isBoolean(output[0])) {

stack.push(output[0]);

output = output.substr(1);

}

else {

char oper = output[0];

output = output.substr(1);

switch (oper) {

case 'A':

output = operationAnd(&stack) + output;

break;

case 'O':

output = operationOr(&stack) + output;

break;

case 'X':

output = operationXor(&stack) + output;

break;

case 'N':

output = operationDeny(&stack) + output;

break;

default:

break;

}

}

}

// Финальная проверка и возврат результатов

if (stack.getSize() != 0) {

cout << "Некорректный синтаксис выражения.\n";

return false;

}

if (output[0] == 'T') return true;

else return false;

}

// Задание 9

void Task09() {

string str = readFile("Lab4/Task09.txt")[0];

bool result = Task09Calc(str);

cout << "Результат для выражения " << str << ": ";

if (result) cout << "true.\n";

else cout << "false.\n";

}

// Функция для операции нахождения максимума

int max(MyStack<char>\* stack) {

int first = stack->pop() - '0';

int second = stack->pop() - '0';

if (first >= second) return first;

else return second;

}

// Функция для операция нахождения минимума

int min(MyStack<char>\* stack) {

int first = stack->pop() - '0';

int second = stack->pop() - '0';

if (first < second) return first;

else return second;

}

// Основная функция для решения записи в задании 10

int Task10Calc(string input) {

// Начальная проверка

if (input.length() == 0) {

cout << "Введена пустая строка!\n";

return -1;

}

MyStack<char> stack;

string output = "";

// Заносим операции в стек

for (size\_t i = 0; i < input.length(); i++) {

char sym = input[i];

if (isDigit(sym)) output += sym;

else {

switch (sym) {

case '(':

case ',':

case 'M':

case 'N':

stack.push(sym);

break;

case ')':

if (stack.peek() != ',') {

cout << "Некорректный синтаксис выражения.\n";

return -1;

}

stack.pop();

stack.pop();

output += stack.pop();

break;

default:

cout << "Некорректный синтаксис выражения.\n";

return -1;

break;

}

}

}

// Выполнение операций из стека

while (!stack.isEmpty()) output += stack.pop();

while (!isDigit(output[output.length() - 1])) {

if (isDigit(output[0])) {

stack.push(output[0]);

output = output.substr(1);

}

else {

char operation = output[0];

output = output.substr(1);

switch (operation)

{

case 'M':

output = to\_string(max(&stack)) + output;

break;

case 'N':

output = to\_string(min(&stack)) + output;

break;

default:

cout << "Некорректный синтаксис выражения.\n";

return -1;

break;

}

}

}

// Финальная проверка и возврат результатов

if (stack.getSize() != 0) {

cout << "Некорректный синтаксис выражения.\n";

return -1;

}

int result = output[0] - '0';

return result;

}

// Задание 10

void Task10() {

string str = readFile("Lab4/Task10.txt")[0];

int result = Task10Calc(str);

cout << "Результат для выражения " << str << ": " << result << endl;

}

// Функция для проверки, является ли символ переменной

bool isName(char sym) {

if (sym == 'X' || sym == 'Y' || sym == 'Z') return true;

else return false;

}

// Функция для проверки операций

char PMoper(MyStack<char>\* stack) {

char first = stack->pop();

char second = stack->pop();

if (first == 'X' || first == 'Y' || first == 'Z') {

if (second == 'X' || second == 'Y' || second == 'Z') return 'X';

}

return 'T';

}

// Функция для проверки, является ли введённое выражение правильным

bool Task11Calc(string input) {

// Начальная проверка

if (input.length() == 0) {

cout << "Введена пустая строка!\n";

return false;

}

MyStack<char> stack;

string output = "";

// Заносим операции в стек

for (size\_t i = 0; i < input.length(); i++) {

char sym = input[i];

if (isName(sym)) output += sym;

else {

switch (sym) {

case '(':

case '+':

case '-':

stack.push(sym);

break;

case ')':

while (stack.peek() != '(') output += stack.pop();

stack.pop();

break;

default:

cout << "Некорректный синтаксис выражения.\n";

return false;

break;

}

}

}

//

while (!stack.isEmpty()) output += stack.pop();

while (!isName(output[output.length() - 1])) {

if (isName(output[0])) {

stack.push(output[0]);

output = output.substr(1);

}

else {

char operation = output[0];

output = output.substr(1);

switch (operation) {

case '+':

case '-':

output = PMoper(&stack) + output;

break;

default:

cout << "Некорректный синтаксис выражения.\n";

return false;

break;

}

}

}

// Финальная проверка и возврат результатов

if (stack.getSize() != 0) {

cout << "Некорректный синтаксис выражения.\n";

return false;

}

else return true;

}

// Задание 11

void Task11() {

string str = readFile("Lab4/Task11.txt")[0];

bool result = Task11Calc(str);

cout << "Выражение " << str << " является ";

if (result) cout << "корректным.\n";

else cout << "некорректным.\n";

}

}

#endif LAB4TASKS\_H

Листинг файла Lab4.h:

#ifndef LAB4\_H

#define LAB4\_H

#include <iostream>

#include "Lab4Tasks.h"

using namespace std;

// Пространство имён 4 лабораторной работы

namespace lab4 {

// Функция для выбора задания из лабораторной работы

void Lab4Start() {

int taskNum = 0;

cout << "Выберите задание (от 1 до 11, любое другое значение - выход): ";

cin >> taskNum;

switch (taskNum) {

case 1:

cout << "Задача №1.\n";

Task01("Lab4/Task01.txt");

break;

case 2:

cout << "Задача №2.\n";

Task02();

break;

case 3:

cout << "Задача №3.\n";

Task03();

break;

case 4:

cout << "Задача №4.\n";

Task04();

break;

case 5:

cout << "Задача №5.\n";

Task05();

break;

case 6:

cout << "Задача №6.\n";

Task06();

break;

case 7:

cout << "Задача №7.\n";

Task07();

break;

case 8:

cout << "Задача №8.\n";

Task08();

break;

case 9:

cout << "Задача №9.\n";

Task09();

break;

case 10:

cout << "Задача №10.\n";

Task10();

break;

case 11:

cout << "Задача №11.\n";

Task11();

break;

default:

break;

}

}

}

#endif LAB4\_H

Результаты выполнения программы представлены на рисунках ниже.

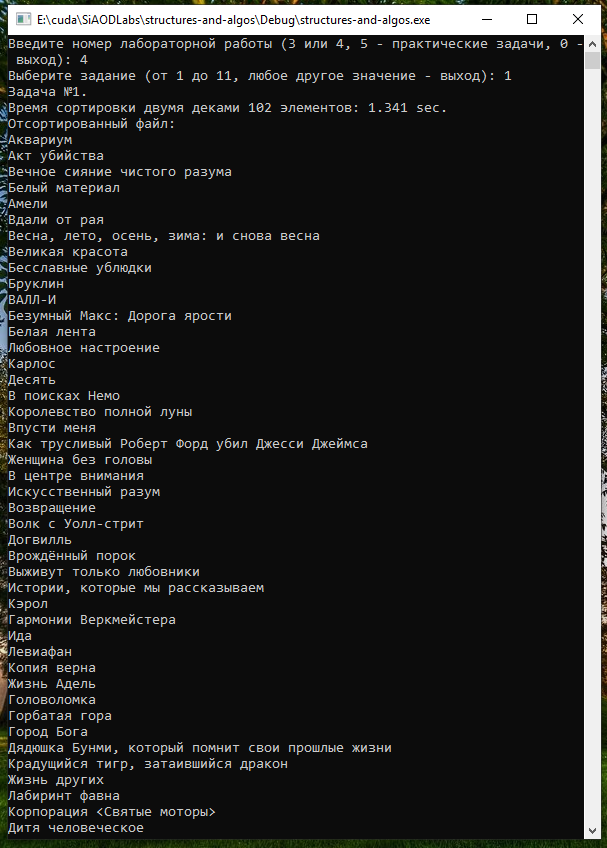


Рисунок 1 – Результат выполнения первой задачи

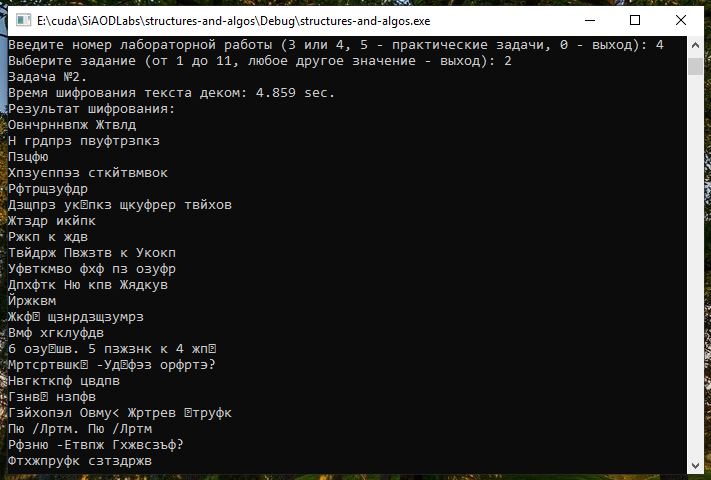
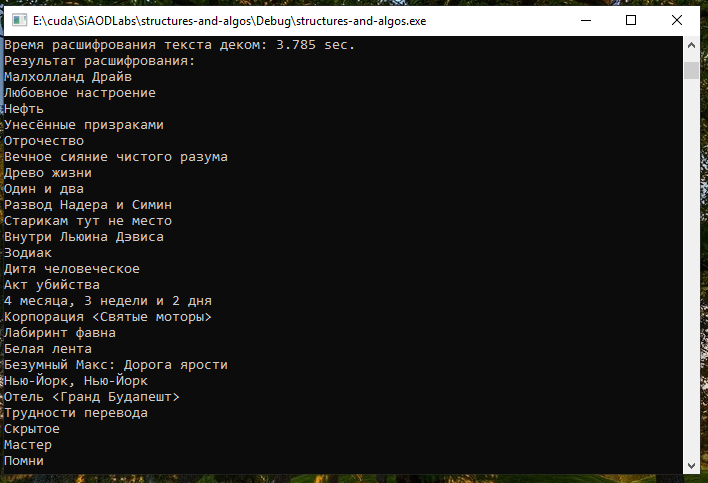
 

Рисунок 2 – Результат выполнения второй задачи

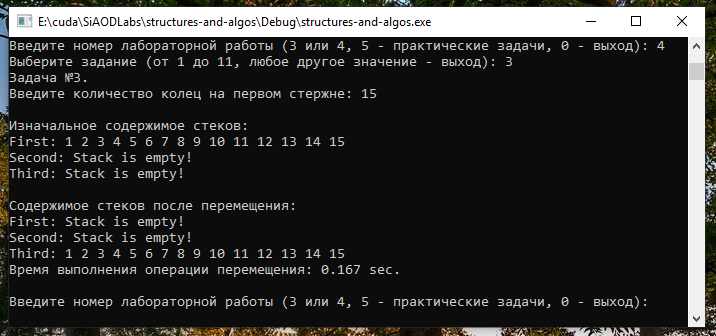


Рисунок 3 – Результат выполнения третьей задачи

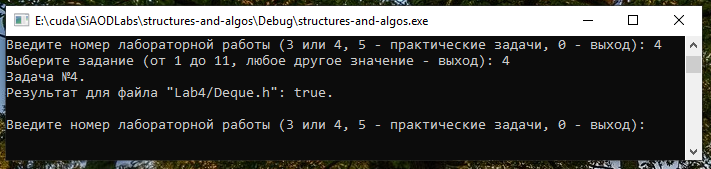


Рисунок 4 – Результат выполнения четвёртой задачи

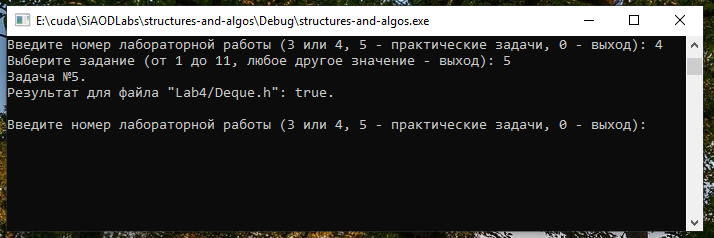


Рисунок 5 – Результат выполнения пятой задачи

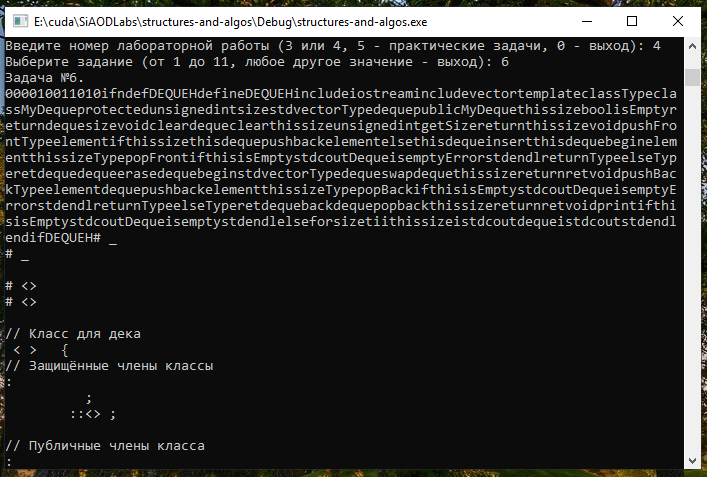


Рисунок 6 – Результат выполнения шестой задачи

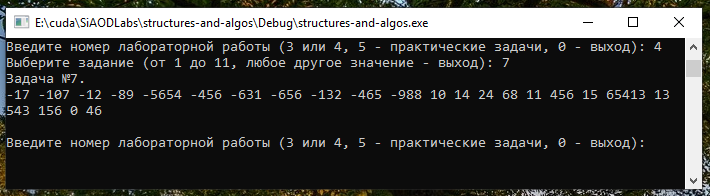


Рисунок 7 – Результат выполнения седьмой задачи

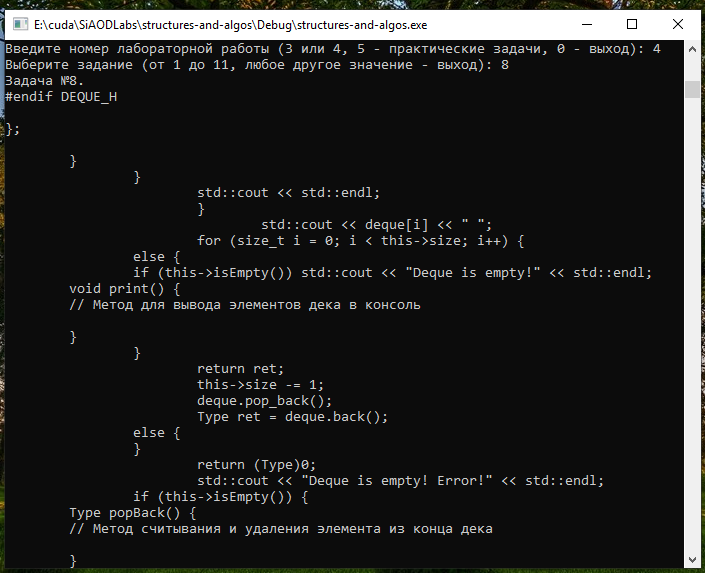


Рисунок 8 – Результат выполнения восьмой задачи

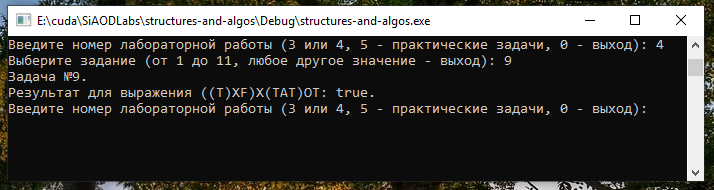


Рисунок 9 – Результат выполнения девятой задачи

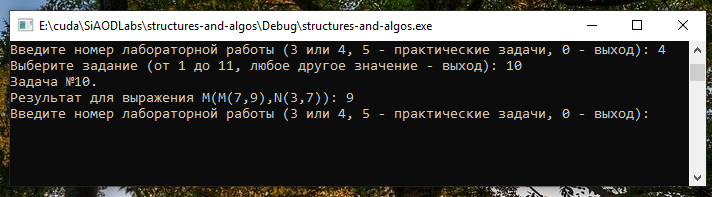


Рисунок 10 – Результат выполнения десятой задачи

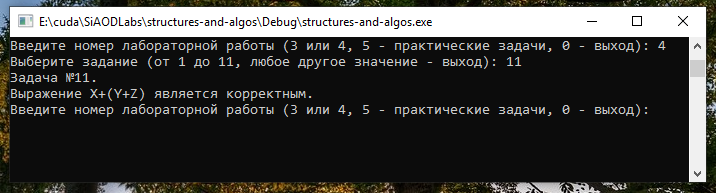


Рисунок 11 – Результат выполнения одиннадцатой задачи

# 3. Вывод

В ходе данной работы были реализованы структуры данных Stack и Deque, а также выполнены прилагающиеся к ним задания. Каждое из 11 заданий является широко известной прикладной задачей, которая эффективно решается именно с помощью этих структур данных. Полученные навыки при реализации этих структур данных, и решении задач с помощью них являются необходимыми в работе программиста.