**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Стек, очередь, дек**

**Вариант №5в**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 7382 |  | Давкаева В.С. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2018

**Цель работы.**

Ознакомление с парадигмой объектно-ориентированного программирования. Получение навыков реализации рекурсивных функций и использования стека и очереди на языке программирования C++.

**Постановка задачи.**

Разработать программу для выполнения задания с использованием функций языка С++.

*Задание варианта 5в:*

Правильная скобочная конструкция с тремя видами скобок определяется как

<текст> ::= <пусто> | <элемент> <текст>

<элемент> ::= <символ> | (<текст>) | [<текст>] | {<текст>},

где <символ> - любой символ, кроме ( , ) , [ , ] , { , }. Проверить, является ли текст, содержащийся в заданном файле F, правильной скобочной конструкцией; если нет, то указать номер ошибочной позиции.

**Основные теоретические положения.**

Стек — структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. По определению, элементы извлекаются из стека в порядке, обратном их добавлению в эту структуру, т.е. действует принцип "последний пришёл — первый ушёл".

Очередь — структура данных, которая, как и стек, имеет ограничения, по добавлению и удалению элементов. Чтобы понять очереди, можно представить очередь в  магазин: человек в начале очереди (тот, кто пришел первый) будет обслуживаться первым, вновь пришедшие люди будут добавляться в конец очереди. Таким образом, первый человек в очереди обслуживается первым, последний в очереди, обслуживается последним. Сокращенно очереди обозначаются как: FIFO — First In, First Out (первым пришел, первым ушел).

**Спецификация программы.**

Данная программа предназначена для реализации программы, которая анализирует заданную символьную последовательность и определяет, является ли она текстом, с использованием очереди и стека, реализованных на базе вектора(массива).

*Описание функций.*

* void call\_error(Queue & Q3, int num) - функция показывающая место ошибки (Q3 – элемент класса очередь, num – номер ошибки)
* bool bracket\_c\_f(char symb) - проверка закрывающей скобки (symb – проверяемый символ, функция возвращает true если скобка закрывающая, false – если нет)
* bool bracket\_o\_f(char symb) ­- проверка открывающей скобки (symb – проверяемый символ, функция возвращает true если скобка открывающая, false – если нет)
* bool cur\_brckt\_o(char symb) – проверка круглой открывающейся скобки (symb – проверяемый символ, функция возвращает true если скобка открывающаяся типа curcle, false – если нет)
* bool cur\_brckt\_c(char symb) – проверка круглой закрывающейся скобки (symb – проверяемый символ, функция возвращает true если скобка закрывающаяся типа curcle, false – если нет)
* bool sq\_brckt\_o(char symb) – – проверка квадратной открывающейся скобки (symb – проверяемый символ, функция возвращает true если скобка открывающаяся типа square, false – если нет)
* bool sq\_brckt\_c(char symb) – проверка квадратной закрывающейся скобки (symb – проверяемый символ, функция возвращает true если скобка закрывающаяся типа square, false – если нет)
* bool fig\_brckt\_o(char symb) – проверка фигурной открывающейся скобки (symb – проверяемый символ, функция возвращает true если скобка открывающаяся типа figure, false – если нет)
* bool fig\_brckt\_c(char symb) – проверка фигурной закрывающейся скобки (symb – проверяемый символ, функция возвращает true если скобка закрывающаяся типа figure, false – если нет)
* bool symb\_f(char symb) – проверка на символ(symb – проверяемый символ, функция возвращает true если это символ, false – если нет)
* bool is\_it\_txt(Stack Q1, Queue & Q3)– проверка формула или нет (если в стеке Q1 содержится текст, то - true,иначе false; Q3 – элемент класса очереди для хранения удовлетворяющих условию символов)
* void main\_ txt(ifstream & fin, Stack & Q2) - является ли полученная строка текстом, в функции также осуществляется вывод ошибок и части строки, являющейся текстом или вывод текста
* void print\_f(Queue & Q3) – печать элементов очереди Q3

*Структура данных.*

Class Stack/Queue{

public:

char arr[];//массив хранения данных

int work[2];//массив состояния первого и последнего элемента стека или очереди

void init(); //инициализация массива

void push(char a); //помещение элемента в стек

bool can\_pop();//можно ли взять элемент

char pop();//взятие элемента из стека

Stack();

};

**Описание алгоритма.**

Для решения этой задачи используется стек, а вывод исходной позиции осуществляется с помощью очереди. Сначала программа посимвольно считывает из файла и заносит полученные символы в стек. Далее используется промежуточный стек, чтобы разместить считанные символы в таком порядке, чтобы при их прочтении из стека получилась исходная символьная последовательность. На следующем этапе программа считывает символ из последнего стека, и, согласно условию задачи, ожидает либо открытую скобку любого типа, либо символ, либо пустой символ. Следующей операцией, в зависимости от предыдущего считанного символа, ожидает либо открытую/закрытую скобку определенного типа, либо символ, либо пустой символ. Анализирование продолжается до тех пор, пока в стеке не закончатся данные, или пока программа получит символ, который не удовлетворяет понятию текста. Параллельно с этим процессом символы, удовлетворяющие этому понятию, заносятся в очередь и, если символьная последовательность не является текстом, то, исходя из последнего элемента очереди, выводится соответствующая информация об ожидаемом символе и удовлетворяющая тексту часть введенной строки.

**Тестирование.**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Результат | Оценка |
| 1 | Hello | This is the text! |  |
| 2 | (hi(hey}buy) | This isn’t the text! | Expected symbol or curcle close bracket. |
| 3 | (({])) | This isn't a text! | Expected symbol or figure close bracket. |
| 4 | {dd[fdddg)fdf}hi | This isn't a text! | Expected symbol or square close bracket. |
| 5 | heloo((hj))) | This isn't a text! | Expected symbol or open bracket of any tipe. |
| 6 | (fgh(hg)hj())(()hn | This isn't a text! | Not enough for the text. Expected symbol or circle close bracket. |
| 7 | ()() | This is the text! |  |
| 8 | {vc{cvvc} | This isn't a text! | Not enough for the text. Expected symbol or figure close bracket. |
| 9 | (([]))[cur[vv] | This isn't a text! | Not enough for the text. Expected symbol or square close bracket. |
| 10 | (вав)(апап)(впвп) | This is the text! |  |
| 11 | п(пропр(рпр)ропрп) | This is the text! |  |

*Пример работы программы.*

Put the elements of the string on the stack:

An item is placed on the stack [д].

An item is placed on the stack [}].

An item is placed on the stack [л].

An item is placed on the stack []].

An item is placed on the stack [в].

An item is placed on the stack [[].

An item is placed on the stack [}].

An item is placed on the stack [с].

An item is placed on the stack [{].

An item is placed on the stack [1].

An item is placed on the stack [{].

An item is placed on the stack [ы].

An item is placed on the stack [)].

An item is placed on the stack [с].

An item is placed on the stack [(].

The string read from the file: (с)ы{1{с}[в]л}д

From the stack, the element is extracted [(] ,which is a curcle open bracket

From the stack, the element is extracted [с] ,which is the symbol

From the stack, the element is extracted [)] ,which is a curcle close bracket

From the stack, the element is extracted [ы] ,which is the symbol

From the stack, the element is extracted [{] ,which is a figure open bracket

From the stack, the element is extracted [1] ,which is the symbol

From the stack, the element is extracted [{] ,which is a figure open bracket

From the stack, the element is extracted [с] ,which is the symbol

From the stack, the element is extracted [}] ,which is a figure close bracket

From the stack, the element is extracted [[] ,which is a square open bracket

From the stack, the element is extracted [в] ,which is the symbol

From the stack, the element is extracted []] ,which is a square close bracket

From the stack, the element is extracted [л] ,which is the symbol

From the stack, the element is extracted [}] ,which is a figure close bracket

From the stack, the element is extracted [д] ,which is the symbol

This is the text!

Source text: (с)ы{1{с}[в]л}д

**Вывод.**

В ходе лабораторной работы была написана, отлажена и протестирована программа с использованием стека и очереди.

**Приложение А. Исходный код**

1.Queue.h

#pragma once

#include <iostream>

#include "Stack.h"

using namespace std;

class Queue

{

public:

char arr[size\_1];//массив хранения данных

int work[2];//массив состояния

void init()//инициализация массива

{

for (int i = 0; i < size\_1; i++)

arr[i] = 0;

work[0] = -1;

work[1] = -1;

}

void push(char a)//помещение элемента в стек

{

if (work[1] != size\_1 - 1)

{

if (work[0] == -1)

{

arr[0] = a;

work[1] = 0;

work[0] = 0;

}

else

{

arr[work[1] + 1] = a;

work[1]++;

}

}

else

cout << "Очередь переполнена." << endl;

}

bool can\_pop()//возможно ли взять

{

if (work[0] != -1)

return 1;

else

return 0;

}

char pop()//взятие элемента из очереди

{

if (work[0] != -1)

{

if (work[0] != work[1])

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0]++;

return tmp;

}

else

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0] = -1;

work[1] = -1;

return tmp;

}

}

else

{

cout << "Не удалось извлечь элемент, очередь пуста." << endl;

return 0;

}

}

Queue()

{

init();

}

};

2.Stack.h

#pragma once

#include <iostream>

using namespace std;

const size\_t size\_1 = 100;

class Stack

{

public:

char arr[size\_1];//массив хранения данных

int work[2];//массив состояния

void init()//инициализация массива

{

for (int i = 0; i < size\_1; i++)

arr[i] = 0;

work[0] = -1;

work[1] = -1;

}

void push(char a)//помещение элемента в стек

{

if (work[1] != size\_1 - 1)

{

if (work[0] == -1)

{

arr[0] = a;

work[1] = 0;

work[0] = 0;

}

else

{

arr[work[0] + 1] = a;

work[0]++;

}

}

else

cout << "Стек переполнен." << endl;

}

bool can\_pop()//можно ли поместить

{

if (work[0] != -1)

return 1;

else

return 0;

}

char pop()//взятие элемента из стека

{

if (work[0] != -1)

{

if (work[0] != work[1])

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0]--;

return tmp;

}

else

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0] = -1;

work[1] = -1;

return tmp;

}

}

else

{

cout << "Не удалось извлечь элемент, стек пуст." << endl;

return 0;

}

}

Stack()

{

init();

}

};

3.Source.cpp

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <locale.h>

#include "Queue.h"

using namespace std;

void menu()//меню

{

cout << "============================================" << endl;

cout << "Davkaeva Valentina 7382 variant 5v" << endl;

cout << "====================MENU====================" << endl;

cout << "============================================" << endl;

cout << "1.Read the string from file." << endl;

cout << "2.End the program." << endl;

cout << "============================================" << endl;

cout << "Your choice:";

}

void call\_error(Queue & Q3, int num) // функция,показывающая место ошибки

{

switch (num)

{

case 1:

{ cout << "------------------------------------Error-----------------------------------------" << endl;

cout << "The last read element doesn't satisfy the definition of the text.Expected symbol or curcle close bracket." << endl;

break;

}

case 2:

{ cout << "------------------------------------Error-----------------------------------------" << endl;

cout << "The last read element doesn't satisfy the definition of the text.Expected symbol or square close bracket." << endl;

break;

}

case 3:

{

cout << "------------------------------------Error-----------------------------------------" << endl;

cout << "The last read element doesn't satisfy the definition of the text.Expected symbol or figure close bracket." << endl;

break;

}

case 4:

{

cout << "------------------------------------Error-----------------------------------------" << endl;

cout << "The last read element doesn't satisfy the definition of the text.Expected symbol or open bracket of any tipe." << endl;

break;

}

case 5:

{

cout << "------------------------------------Error-----------------------------------------" << endl;

cout << "Not enough for the text.Expected symbol or circle close bracket." << endl;

break;

}

case 6:

{

cout << "------------------------------------Error-----------------------------------------" << endl;

cout << "Not enough for the text.Expected symbol or figure close bracket." << endl;

break;

}

case 7:

{

cout << "------------------------------------Error-----------------------------------------" << endl;

cout << "Not enough for the text.Expected symbol or square close bracket." << endl;

break;

}

}

cout << "This isn't a text!" << endl;

cout << "Part of the string, which can be completed to the text: ";//Часть строки, которая является формулой и может быть завершена.

while (Q3.can\_pop())//если элемент можно взять

cout << Q3.pop() << " ";//печать посимвольно элементов очереди

cout << "\nEnd of analyzing! " << endl;//Программа будет завершена.

system("pause");

exit(1);

}

bool cur\_brckt\_o(char symb) // проверка круглой открытой скобки

{

if (symb == '(')

return true;

else

return false;

}

bool cur\_brckt\_c(char symb) // проверка круглой закрытой скобки

{

if (symb == ')')

return true;

else

return false;

}

bool sq\_brckt\_o(char symb) // проверка квадратной открытой скобки

{ if (symb == '[')

return true;

else

return false;

}

bool sq\_brckt\_c(char symb) // проверка квадратной закрытой скобки

{

if (symb == ']')

return true;

else

return false;

}

bool fig\_brckt\_o(char symb) // проверка фигурной открытой скобки

{

if (symb == '{')

return true;

else

return false;

}

bool fig\_brckt\_c(char symb) // проверка фигурной закрытой скобки

{

if (symb == '}')

return true;

else

return false;

}

bool symb\_f(char symb) //является ли элемент именем

{

if (symb == '(' || symb == '{' || symb == '[' || symb == ')' || symb == '}' || symb == ']')

return false;

else

return true;

}

void is\_it\_txt(Stack Q1, Queue & Q3) //анализирующая функция

{

char symb;

Stack pos;//стек,служащий для хранения позиции и типа скобки

bool bracket\_c\_cur,bracket\_c\_sq, bracket\_c\_fig;

int n;

bracket\_c\_cur = bracket\_c\_sq = bracket\_c\_fig = false;//вначале не ожидается закрытая скобка любого типа

while(Q1.can\_pop())//пока стек не пуст

{

symb = Q1.pop();//взятие элемента

cout << " From the stack, the element is extracted [" << symb << "] ";//Из стека извлечен элемент [

if (symb\_f(symb))// если символ

{

cout << ",which is the symbol " << endl;//который является элементом.

if (pos.can\_pop()) {//если уже есть открытая скобка любого типа

n = pos.pop();//узнаем,какого типа

if (n == 1) {//круглая скобка

bracket\_c\_cur = true;//тогда ожидается закрывающаяся круглая скобка

bracket\_c\_sq = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 2) {//фигурная скобка

bracket\_c\_fig = true;//тогда ожидается закрывающаяся фигурная скобка

bracket\_c\_sq = bracket\_c\_cur = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 3) {//квадратная скобка

bracket\_c\_sq = true;//тогда ожидается закрывающаяся квадратная скобка

bracket\_c\_cur = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

pos.push(n);//возвращаем проверенный элемент в стек

}else bracket\_c\_cur = bracket\_c\_sq = bracket\_c\_fig = false;//если нет,то запрещаем закрывающиеся

Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}

if (cur\_brckt\_o(symb))//круглая открывающая скобка

{

cout << ",which is a curcle open bracket " << endl;

bracket\_c\_cur = true;//тогда ожидается закрывающаяся круглая скобка

bracket\_c\_sq = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

pos.push(1);//заносим в стек номер скобки,которая на следующих шагах должна быть закрыта

Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}

if (cur\_brckt\_c(symb))//круглая закрывающая скобка

{

cout << ",which is a curcle close bracket" << endl;

if (!bracket\_c\_cur&&bracket\_c\_fig) call\_error(Q3, 3);//если на данном шаге не должна стоять закрывающаяся круглая скобка,а должна стоять фигурная,то выдаем соответсвующую ошибку

if (!bracket\_c\_cur&&bracket\_c\_sq) call\_error(Q3, 2);//если на данном шаге не должна стоять закрывающаяся круглая скобка,а должна стоять квадратная,то выдаем соответсвующую ошибку

if (pos.can\_pop()) {//если уже есть открытая скобка любого типа

n = pos.pop();//узнаем,какого типа

if (n == 1) {//если это круглая

if (pos.can\_pop()) {//проверяем,если еще открытые скобки

n = pos.pop();//узнаем какого типа ранее идущая скобка

if (n == 1) {//круглая скобка

bracket\_c\_cur = true;//тогда ожидается закрывающаяся круглая скобка

bracket\_c\_sq = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 2) {//фигурная скобка

bracket\_c\_fig = true;//тогда ожидается закрывающаяся фигурная скобка

bracket\_c\_sq = bracket\_c\_cur = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 3) {//квадратная скобка

bracket\_c\_sq = true;//тогда ожидается закрывающаяся квадратная скобка

bracket\_c\_cur = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

pos.push(n);//возвращаем проверенный элемент в стек

Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}else Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}

}else call\_error(Q3, 4);//если нет,то вызываем соответствующую ошибку

}

if (fig\_brckt\_o(symb))//фигурная открывающая скобка

{

cout << ",which is a figure open bracket " << endl;

bracket\_c\_fig = true;//тогда ожидается закрывающаяся фигурная скобка

bracket\_c\_sq = bracket\_c\_cur = false;//остальные типы закрывающихся скобок запрещены

pos.push(2);//заносим в стек номер скобки,которая на следующих шагах должна быть закрыта

Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}

if (fig\_brckt\_c(symb))//фигурная закрывающая скобка

{

cout << ",which is a figure close bracket" << endl;

if (!bracket\_c\_fig&&bracket\_c\_cur) call\_error(Q3, 1);//если на данном шаге не должна стоять закрывающаяся фигурная скобка,а должна стоять круглая,то выдаем соответсвующую ошибку

if (!bracket\_c\_fig&&bracket\_c\_sq) call\_error(Q3, 2);//если на данном шаге не должна стоять закрывающаяся фигурная скобка,а должна стоять квадратная,то выдаем соответсвующую ошибк

if (pos.can\_pop()) {//если уже есть открытая скобка любого типа

n = pos.pop();//узнаем,какого типа

if (n == 2) {//если фигурная

if (pos.can\_pop()) {//проверяем,если еще открытые скобки

n = pos.pop();//узнаем какого типа ранее идущая скобка

if (n == 1) {//круглая скобка

bracket\_c\_cur = true;//тогда ожидается закрывающаяся круглая скобка

bracket\_c\_sq = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 2) {//фигурная скобка

bracket\_c\_fig = true;//тогда ожидается закрывающаяся фигурная скобка

bracket\_c\_sq = bracket\_c\_cur = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 3) {//квадратная скобка

bracket\_c\_sq = true;//тогда ожидается закрывающаяся квадратная скобка

bracket\_c\_cur = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

pos.push(n);//возвращаем проверенный элемент в стек

Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}else Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}

}else call\_error(Q3, 4);//если нет,то вызываем соответствующую ошибку

}

if (sq\_brckt\_o(symb))//квадратная открывающая скобка

{

cout << ",which is a square open bracket " << endl;

bracket\_c\_sq = true;//тогда ожидается закрывающаяся квадратная скобка

bracket\_c\_cur = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

pos.push(3);//заносим в стек номер скобки,которая на следующих шагах должна быть закрыта

Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}

if (sq\_brckt\_c(symb))//квадратная закрывающая скобка

{

cout << ",which is a square close bracket" << endl;

if (!bracket\_c\_sq&&bracket\_c\_fig) call\_error(Q3, 3);//если на данном шаге не должна стоять закрывающаяся квадратная скобка,а должна стоять фигурная,то выдаем соответсвующую ошибку

if (!bracket\_c\_sq&&bracket\_c\_cur) call\_error(Q3, 1);//если на данном шаге не должна стоять закрывающаяся квадратная скобка,а должна стоять круглая,то выдаем соответсвующую ошибк

if (pos.can\_pop()) {//если уже есть открытая скобка любого типа

n = pos.pop();//узнаем,какого типа

if (n == 3) {//если квадратная

if (pos.can\_pop()) {//проверяем,если еще открытые скобки

n = pos.pop();//узнаем какого типа ранее идущая скобка

if (n == 1) {//круглая скобка

bracket\_c\_cur = true;//тогда ожидается закрывающаяся круглая скобка

bracket\_c\_sq = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 2) {//фигурная скобка

bracket\_c\_fig = true;//тогда ожидается закрывающаяся фигурная скобка

bracket\_c\_sq = bracket\_c\_cur = false;//остальные типы закрывающихся скобок запрещены

}

if (n == 3) {//квадратная скобка

bracket\_c\_sq = true;//тогда ожидается закрывающаяся квадратная скобка

bracket\_c\_cur = bracket\_c\_fig = false;//остальные типы закрывающихся скобок запрещены

}

pos.push(n);//возвращаем проверенный элемент в стек

Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}else Q3.push(symb);//заносим в очередь удовлетворяющий элемент

}

}else call\_error(Q3, 4);//если нет,то вызываем соответствующую ошибку

}

}

if (pos.can\_pop()) {//если есть еще открывающиеся скобки

n = pos.pop();//достаем элемент

if (n == 1) call\_error(Q3, 5);//если круглая,то выводим соответствующую ошибку

if (n == 2) call\_error(Q3, 6);//если фигурная,то выводим соответствующую ошибку

if (n == 3) call\_error(Q3, 7);//если квадратная,то выводим соответствующую ошибку

}

}

void print\_f(Queue & Q3) // вывести текст

{

while (Q3.can\_pop())

{

cout << Q3.pop();

}

}

void main\_txt(istream &fin,Stack & Q2) // функция,в которой проверятся условие, является ли текстом введенные символы

{

Queue Q3;//очередь для выхода текста

char symb;

fin >> symb;

while (!(fin.eof())) {//пока не достиг конец файла

Q2.push(symb);//добавляем в стек

Q3.push(symb);//добавляем в очередь

fin >> symb;

}

Stack Q1;//промежуточный стек для вывода в нужном порядке

cout << "Put the elements of the string on the stack:" << endl;//Заносим элементы строки в стек:

while (Q2.can\_pop())//пока можно взять из стека

{

char a = Q2.pop();//берем элемент

cout << " An item is placed on the stack [" << a << "]." << endl;//В стек помещен элемент [" << a << "].

Q1.push(a);//перезаписываем в другой стек

}

cout << "The string read from the file: ";//считанная из файла строка

print\_f(Q3);//выводим строку

cout << endl;

is\_it\_txt(Q1, Q3);//проверяем на текст

cout << "This is the text!" << endl;//Это текст

cout << "Source text: ";//Ее вид

print\_f(Q3);//выводим строку

}

int main()

{

setlocale(0, "RUS");

int answer;

Stack Q2;

menu();

cin >> answer;

switch (answer)

{

case 1:

{

ifstream fin("text.txt");

system("cls");

main\_txt(fin,Q2);

fin.close();

break;

}

case 2:

{

system("cls");

cout << "End of program!" << endl;

system("pause");

return 0;

}

default:

system("cls");

cout << "Incorrect input.Try again.";

break;

}

cout << endl;

system("pause");

}