**ЛАБОРАТОРНАЯ РАБОТА No4**

**«ООПC++.ШАБЛОНЫ»**

**Задание:**

Лабораторная работа №4 состоит из двух независимых частей: **Шаблоны функций** и **Шаблоны классов**.

Для выполнения и защиты Лабораторной работы №4 достаточно выполнить только одну часть (**Шаблоны функций**): на оценку «удовлетворительно» - 1ое задание; на оценку «хорошо» - два первых задания (1ое и 2ое); на оценку «отлично» - все три задания. Подготовить несколько наборов характерных тестов!

**Шаблоны классов** – это дополнительное задание (за него ставится дополнительная оценку, которая будет учитываться на экзамене).

Выполнить задание на тему «Одномерные массивы», оформив каждый пункт задания в виде шаблона функции. Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается. Привести примеры программ, использующих эти шаблоны для типов int, float и double.

**Шаблоны функций:**

**Вариант 16**

В одномерном массиве, состоящем из п вещественных элементов, вычислить:

1) количество отрицательных элементов массива;

2) Сумму модулей элементов массива, расположенных после минимального по модулю элемента.

3) Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.

## Вариант 2

Создать шаблон класса «стек». Использовать его при решении следующей задачи:

Составить программу, отыскивающую проход по лабиринту. Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квадрат определяется его координатами в матрице. Программа находит проход через лабиринт, двигаясь от заданного входа. После отыскания прохода программа выводит найденный путь в виде координат квадратов. Для хранения пути использовать стек.

**1.Постановка задачи**

Данный отчет будет сделан для второй части на тему «**Шаблоны классов**».

Для реализации данной задачи я создам классы: «**LabirintElem**», «**Labirint**», «**Robot**», «**StackElem**» и «**Stack**».

В качестве входных данных у нас будет текстовый документ, в котором будут указаны координаты входа в лабиринт, а также сам лабиринт в матричном виде, состоящий из «**0**» и «**1**», где «**0**» - закрытый квадрат лабиринта, а «**1**» - открытый квадрат лабиринта.

Табл. 1

Входные данные

|  |  |
| --- | --- |
| Входные данные | Смысл |
| Текстовый файл | Содержит координаты входа и сам лабиринт в виде матрицы |

В качестве выходных данных будем использовать сообщение в консоли и текстовый файл, в который будут записаны координаты прохода лабиринта (взятые из стека), если выход из лабиринта был найден.

Табл. 2

Выходные данные

|  |  |
| --- | --- |
| Выходные данные | Тип данных |
| Текстовый файл | Путь прохода лабиринта в виде координат |

Выходом будет считаться любой крайний элемент матрицы, за исключением входа.

**Ограничения:**

1. Не указан вход
2. Нету выхода из лабиринта

**Обработка ограничений**

1. Будет выведено сообщение «*Не правильно записаны данные в текстовом файле!»* и программа закончит работу.
2. Будет выведено сообщение «*Нет выхода, либо неправильно указан вход!*» и программа закончит работу.

**Среда разработки**

Необходимо разработать консольное приложение в среде разработки Visual Studio Community 2019 версии 16.3.0 C++ в соответствии с заданием. Продемонстрировать работу классов и выявить проблемы возникшие при разработки программы.

**2. Разработка программы**

**2.1 Разработка структуры программы**

В соответствии с заданием разработаем программу, я думаю разбить программу на несколько частей: Заголовочные файлы классов, которые мы будем создавать, реализация их методов в .cpp файлах, а также главный .cpp файл с функцией **main** (он будет называться «**Source.cpp**»), в которой мы реализуем пользовательский интерфейс. Создадим эти файлы. Всего у нас будет 5 различных классов, со своими полями и методами.

Классы:

1. «**LabirintElem**» (Табл. 3)
2. «**Labirint**» (Табл. 4)
3. «**Robot**» (Табл. 5)
4. «**StackElem**» (Табл. 6)
5. «**Stack**» (Табл. 7)

Класс «**StackElem**» будет вложенным в класс «**Stack**».

Распишем приблизительные поля и методы классов.

Табл. 3

Класс LabirintElem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | Тип данных | Смысл поля | Методы | Смысл методов |
| open (private) | bool | Открыт ли квадрат | isOpen | Возвращает значение open |
| x (private) | int | Номер столбца | getX | Возвращает значение x |
| y (private) | int | Номер строки | getY | Возвращает значение y |

Табл. 4

Класс Labirint

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | Тип данных | Смысл поля | Методы | Смысл методов |
| labirbint (public) | vector<vector  <LabirintElem>> | Лабиринт, записанный в виде матрицы | setStart | Устанавливает значение поля start |
| start (private) | LabirintElem | Вход в лабиринт | getStart | Возвращает поле start |
|  |  |  | nextIsOpen | Проверяет, открыт ли следующий квадрат лабиринта, с учетом поворота робота |
|  |  |  | readLabirint | Считывает лабиринт из файла «labirint.txt» |
|  |  |  | thatIsExit | Проверяет, является ли данный квадрат выходом |

Класс Robot

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | Тип данных | Смысл поля | Методы | Смысл методов |
| storona (private) | int | Поворот робота 0-вверх, 1-вправо 2-вниз, 3-влево | goNext | Перейти в следующую ячейку, с учетом поворота робота |
| thatLabirintElem (private) | LabirintElem | Текущее местоположение робота в лабиринте | NaitiVihod | Найти выход в лабиринте, если он существует |
|  |  |  | povernutLeft | Повернуть влево на 90 ° |
|  |  |  | povernutRight | Повернуть вправо на 90 ° |
|  |  |  | WriteInStack | Записать в стек текущую клетку |

Класс StackElem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | Тип данных | Смысл поля | Методы | Смысл методов |
| info (public) | Указывается программистом | Поле для хранение информации | goNext | Возвращает значение следующего элемента стека |
| next (private) | StackElem\* | Указатель на следующий элемент стека |  |  |

Класс Stack

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Поля | Тип данных | Смысл поля | Методы | Смысл методов |
| head (private) | StackElem\* | Вершина стека | getSize | Возвращает значение следующего элемента стека |
| size (private) | int | Размер стека | push | Добавить элемент в вершину стека |
|  |  |  | pop | Удалить элемент из вершины стека |
|  |  |  | top | Вернуть значение вершины стека |

Ниже показана диаграмма классов в разработанной нами программе (Рис. 1)

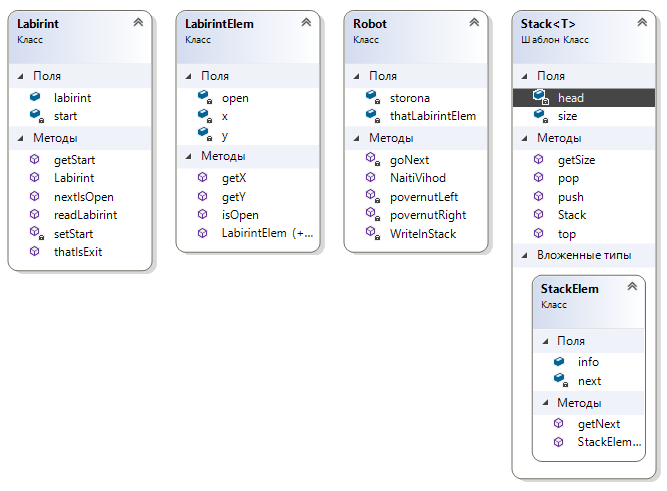


Рис. 1 Диаграмма классов

**2.2 Разработка алгоритма**

Разработаем алгоритм для нахождения выхода из лабиринта, он указан на Рис. 2

**2.3 Разработка пользовательского интерфейса**

Пользователем может являться человек, которому необходимо найти проход через лабиринт, когда перед ним есть его схема.

Программа разрабатывается в консольном виде, основное взаимодействие с программой идет через файлы «labirint.txt» и «LabirintPath.txt», файл «labirint.txt» должен лежать рядом с программой. В нем должен содержаться вход в лабиринт и сам лабиринт в матричной форме.

Пример текстового файла приведен ниже (Рис 3)

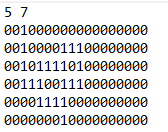


Рис. 3 Пример данных в текстовом файле

В качестве ответа, программа создает текстовый файл «LabirintPath.txt» в той же папке, где и находится программа. Пример выходных данных представлен ниже на Рис. 4.



Рис. 4 Пример выходных данных

**3. Реализация и тестирование приложения**

***Реализация***

Программа была реализована в Visual Studio Community 2019 версии 16.3.0 в среде разработки C++. На устройстве характеристиками, указанными чуть ниже. Реализация кода программы будет расписана ниже в пункте «**Листинг кода**».

**Описание компьютера:**

Операционная система: Windows 10 (x64)

ОЗУ: 16 ГБ

Процессор: Intel Core i5-2500

Видеокарта: Radeon R7 200 Series

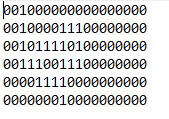
***Тестирование***

**Объект тестирования:** Lab4.exe

**Цель тестирования:** Проверка общей работоспособности программы.

**Метод тестирование:** функциональное тестирование и проход отладчиком для каждой функции по-отдельности и в общем для всей программы.

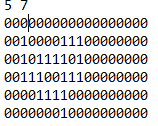
Тестирование 1 (Рис 5):



1. б)

Рис. 5. Тестирование и проверка ограничения №1 где а) Входной файл, б) Диалог в консоли, Выходной файл отсутствует, так как не указан вход

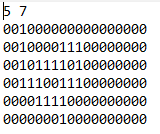
Тестирование 2 (Рис 6):

а) б)

Рис. 6. Тестирование и проверка ограничения №2 где а) Входной файл, б )Диалог в консоли, Выходной файл отсутствует, так как выход отсутствует

Тестирование 3 (Рис 7):



а)



б)



в)

Рис. 7. Тестирование в нормальных условиях, где а) Входной файл, б) Диалог в консоли, в) Выходной файл

**Вывод:** Во время выполнения данной работы я научился создавать и использовать шаблонные классы. Программа работает корректно. Исключения выбрасываются корректно.

**Листинг кода**

**Файл «LabirintElem.h»**

#pragma once

#include <vector>

#include <fstream>

#include <iostream>

#include <string>

using namespace std;

class LabirintElem

{

int x;

int y;

bool open;

public:

LabirintElem();

LabirintElem(int x, int y, bool open);

int getX();

int getY();

bool isOpen();

};

**Файл «LabirintElem.cpp»**

#include "LabirintElem.h"

LabirintElem::LabirintElem()

{

}

LabirintElem::LabirintElem(int y, int x, bool open)

{

this->y = y;

this->x = x;

this->open = open;

}

int LabirintElem::getX() { return x; }

int LabirintElem::getY() { return y; }

bool LabirintElem::isOpen() { return open; }

**Файл «Labirint.h»**

#pragma once

#include "LabirintElem.h"

class Labirint

{

LabirintElem start;

void setStart(LabirintElem start);

public:

vector<vector<LabirintElem>> labirint;

Labirint();

LabirintElem getStart();

bool nextIsOpen(LabirintElem a, int storona);

bool thatIsExit(LabirintElem that);

bool readLabirint(string path = "labirint.txt");

};

**Файл «Labirint.cpp»**

#include "Labirint.h"

void Labirint::setStart(LabirintElem start)

{this->start = start;}

Labirint::Labirint(){}

LabirintElem Labirint::getStart() { return start; }

bool Labirint::nextIsOpen(LabirintElem a, int storona)

{

switch (storona)

{

case 0:

if (a.getY() == 0)

return false;

if (labirint[a.getY() - 1][a.getX()].isOpen()) { return true; }

break;

case 1:

if (a.getX() == labirint[0].size() - 1)

return false;

if (labirint[a.getY()][a.getX() + 1].isOpen())

{ return true; }

break;

case 2:

if (a.getY() == labirint.size() - 1)

return false;

if (labirint[a.getY() + 1][a.getX()].isOpen()) { return true; }

break;

case 3:

if (a.getX() == 0)

return false;

if (labirint[a.getY()][a.getX() - 1].isOpen()) { return true; }

break;

default:

return false;

break;

}

return false;

}

bool Labirint::thatIsExit(LabirintElem that)

{

if (that.getX() == 0 || that.getX() == labirint[0].size() - 1 || that.getY() == 0 || that.getY() == labirint.size() - 1) {

if (that.getX() == start.getX() && that.getY() == start.getY())

return false;

return true;

}

return false;

}

bool Labirint::readLabirint(string path)

{

ifstream fin(path);

int x, y;

fin >> y >> x;

int row = 0;

labirint.clear();

bool flag;

while (!fin.eof()) {

string labirintRow = "";

fin >> labirintRow;

if (labirintRow == "" && row == 0)

return false;

vector<LabirintElem> a1;

labirint.push\_back(a1);

for (int col = 0; col < labirintRow.length(); ++col)

{

if (labirintRow[col] == '0') { flag = false; }

else { flag = true; }

LabirintElem buff(row, col, flag);

labirint[row].push\_back(buff);

}

++row;

}

start = labirint[y][x];

return true;

}

**Файл «Robot.h»**

#pragma once

#include "Labirint.h"

#include "Stack.h"

class Robot

{

LabirintElem thatLabirintElem;

int storona;//0-вперед, 1-вправо, 2-назад, 3-влево

void povernutLeft();

void povernutRight();

void goNext(Labirint a, int storona);

void WriteInStack(Stack<LabirintElem>& stack, LabirintElem& elem);

public:

Robot()

{

//thatLabirintElem = NULL;

storona = 0;

};

bool NaitiVihod(Labirint a, Stack<LabirintElem>& stack);

};

**Файл «Robot.cpp»**

#include "Robot.h"

void Robot::povernutLeft()

{

if (storona-- == 0)

{

storona = 3;

}

}

void Robot::povernutRight()

{

if (storona++ == 3)

{

storona = 0;

}

}

void Robot::goNext(Labirint a, int storona)

{

switch (storona)

{

case 0:

thatLabirintElem = a.labirint[thatLabirintElem.getY() - 1][thatLabirintElem.getX()];

break;

case 1:

thatLabirintElem = a.labirint[thatLabirintElem.getY()][thatLabirintElem.getX() + 1];

break;

case 2:

thatLabirintElem = a.labirint[thatLabirintElem.getY() + 1][thatLabirintElem.getX()];

break;

case 3:

thatLabirintElem = a.labirint[thatLabirintElem.getY()][thatLabirintElem.getX() - 1];

break;

default:

break;

}

}

void Robot::WriteInStack(Stack<LabirintElem>& stack, LabirintElem& elem)

{

stack.push(elem);

}

bool Robot::NaitiVihod(Labirint a, Stack<LabirintElem>& stack)

{

int loop = 0;

thatLabirintElem = a.getStart();

WriteInStack(stack, thatLabirintElem);

if (!thatLabirintElem.isOpen()) {

return false;

}

while (!a.thatIsExit(thatLabirintElem)) {

if (thatLabirintElem.getX() == a.getStart().getX() && thatLabirintElem.getY() == a.getStart().getY()) {

if (loop++ == 2) {

return false;

}

}

povernutRight();

if (a.nextIsOpen(thatLabirintElem, storona))

{

goNext(a, storona);

WriteInStack(stack, thatLabirintElem);

}

else {

povernutLeft();

if (a.nextIsOpen(thatLabirintElem, storona))

{

goNext(a, storona);

WriteInStack(stack, thatLabirintElem);

}

else

{

povernutLeft();

}

}

}

return true;

}

**Файл «Stack.h»**

#pragma once

template <class T>

class Stack

{

class StackElem

{

StackElem\* next;

public:

T info;

StackElem()

{

}

StackElem(T info, StackElem\* next = nullptr)

{

this->info = info;

this->next = next;

}

StackElem\* getNext()

{

return next;

}

};

StackElem\* head;

int size;

public:

Stack()

{

head = nullptr;

size = 0;

}

void push(T info)

{

++size;

StackElem\* buff = head;

head = new StackElem(info, buff);

}

void pop()

{

--size;

StackElem\* buff = head;

head = head->getNext();

delete buff;

}

T top()

{

return head->info;

}

int getSize()

{

return size;

}

};

**Файл «Source.cpp»**

#include "Robot.h"

using namespace std;

void printStack(Stack<LabirintElem> &stack) {

Stack<LabirintElem> buffstack;

ofstream fout("LabirintPath.txt");

int size = stack.getSize();

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

{

buffstack.push(stack.top());

stack.pop();

}

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

{

fout << "(" << buffstack.top().getY() << ", " << buffstack.top().getX() << ")\n";

stack.push(buffstack.top());

buffstack.pop();

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Labirint lab;

if (!lab.readLabirint()) {

cout << "Не правильно записаны данные в текстовом файле!\n";

return -1;

}

Robot robocop;

Stack<LabirintElem> stack;

if (robocop.NaitiVihod(lab, stack))

{

printStack(stack);

cout << "Выход найден, путь прохождения записан в файл \"LabirintPath.txt\"!\n";

}

else

{

cout << "Нет выхода, либо неправильно указан вход!\n";

}

int buff = stack.getSize();

for (int i = 0; i < buff; i++)

{

stack.pop();

}

system("pause");

return 0;

}