Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт прикладных информационных

технологий и коммуникаций

Кафедра Информационная безопасность

автоматизированных систем

Направление подготовки 10.03.01 Информационная безопасность

Расчётно-графическая работа

по дисциплине «Языки программирования»

Тема: «Лабиринт, поиск выхода»

Выполнил: студент 1 курса

Учебной группы с-ИБС11

Очной формы обучения

Сахипов Эльдар Аскарович

Проверил: ассистент каф. ИБС

Романчук С. П.

Саратов 2020

**Аннотация**

Выполнение расчётно-графической работы можно свести к нескольких обобщённым областям, а именно: работой с средой разработки Code::Blocks; изучение алгоритмов нахождения кратчайших путей между двумя точками в пространстве, способном преодолевать между ними преграды;

Содержание

[**Введение** 4](#_Toc41182466)

[**Тема расчётно-графической работы:** 4](#_Toc41182467)

[**Описание задачи:** 4](#_Toc41182468)

[**Цель расчётно-графической работы:** 4](#_Toc41182469)

[**Задачи расчётно-графической работы:** 4](#_Toc41182470)

[**Видение итога выполнения работы:** 4](#_Toc41182471)

[**Теоретическая часть** 5](#_Toc41182472)

[**Выбор языка программирования: С++** 5](#_Toc41182473)

[**Выбор среды разработки:** Code::Blocks 6](#_Toc41182474)

[**Выбор поиска кратчайшего пути:** Алгоритм Ли 6](#_Toc41182475)

[**Практическая часть** 7](#_Toc41182476)

[**Описание алгоритма Ли** 7](#_Toc41182477)

[**Используемые библиотеки:** 8](#_Toc41182478)

[**Ход работы программы:** 9](#_Toc41182479)

[**Заключение** 11](#_Toc41182480)

[**Приложения** 12](#_Toc41182481)

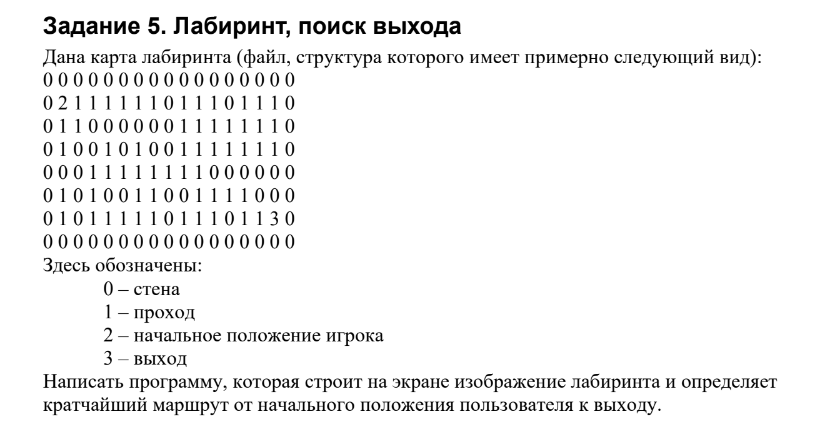
[**Литература** 14](#_Toc41182482)

# **Введение**

## **Тема расчётно-графической работы:**

Программа «Лабиринт, поиск выхода»

## **Описание задачи:**

****

## **Цель расчётно-графической работы:**

Используя среду разработки Code::Blocks создать программу, которая строит на экране изображение лабиринта и определяет кратчайший маршрут от начального положения пользователя к выходу на языке С++.

## **Задачи расчётно-графической работы:**

* Изучение основных возможностей Code::Blocks;
* Изучение алгоритма Ли (волновой алгоритм);
* Создание карты лабиринта;
* Создание лабиринта с кратчайшим путем;
* Реализация алгоритма Ли;
* Вывод в консольную панель лабиринтов;
* Оформление лабиринтов разными цветами;

## **Видение итога выполнения работы:**

В результате выполнения работы ожидается получить программу, которая сможет определять кратчайший путь из начальной точки в конечную, используя алгоритм Ли (волновой алгоритм).

# **Теоретическая часть**

## **Выбор языка программирования: С++**

C++ - компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.

Язык С++ возник в начале 1980-х годов, когда сотрудник фирмы Bell Labs Бьёрн Страуструп придумал ряд усовершенствований к языку C под собственные нужды. Когда в конце 1970-х годов Страуструп начал работать в Bell Labs над задачами теории очередей, он обнаружил, что попытки применения существующих в то время языков моделирования оказываются неэффективными, а применение высокоэффективных машинных языков слишком сложно из-за их ограниченной выразительности. Так, язык Симула имеет такие возможности, которые были бы очень полезны для разработки объемного программного обеспечения, но работает слишком медленно, а язык BCPL достаточно быстр, но слишком близок к языкам низкого уровня и не подходит для разработки объемного программного обеспечения. Страуструп дополнил язык C возможностями работы с классами и объектами. В результате практические задачи моделирования оказались доступными для решения как с точки зрения времени разработки (благодаря использованию Симула-подобных классов), так и с точки зрения времени вычислений (благодаря быстродействию C).

C++ — один из самых мощных и востребованных языков программирования. Ежедневно на нём пишут сотни приложений, зачастую использующих GUI. В противовес этому часто ставится скорость написания кода, которая, например, у интерпретируемых языков на порядок выше. Однако очевиден выбор, что лучше для конечного пользователя: время разработки приложения или его скорость работы?

Скорость исполнения кода — пожалуй, главный аргумент в пользу того, почему C++ был, есть и будет востребован в ИТ. В первую же очередь, огромное сообщество — плодородная почва для появления качественной литературы. По C++ есть несколько фундаментальных книг, по которым училось несколько поколений, есть новые, учитывающие все свежие изменения и актуальное ПО, есть масса интернет-ресурсов для обучения.

Компиляторы C++ есть на каждой операционной системе, большинство программ легко переносится с платформы на платформу, со средой разработки и библиотеками у вас точно не возникнет проблем. C++ — это демонстрация идеи классического программирования, когда 90% мыслей связано с кодом и лишь 10% с используемой периферией. Достаточно просто вспомнить, где он используется: микроконтроллеры, IoT, роботы, десктопные и мобильные приложения, веб, игры, системы моделирования, прогнозирования, обработки статистики и в нейронных сетях. Везде. Просто не существует такой области программирования, где C++ был бы бесполезен.

C++ имеет колоссальное сообщество программистов, которые постоянно делятся библиотеками, шаблонами и кодами, приходят на помощь новичкам и опытным коллегам.

С++ полезен в качестве фундамента для обучения. Java, JavaScript, C #, как и огромное количество других популярных языков программирования, содержит в основе принципы C++. Именно поэтому учебные программы многих вузов содержат курс «плюсплюс», который идёт сразу за получением основ на C. К примеру, принцип работы Java, одного из мощнейших языков современности, достаточно сложно понять, если не начать обучение с основ, которые впервые появились именно в C++. Кроме того, популярно мнение, что если вы сможете изучить C++, то любой другой язык не вызовет у вас никаких затруднений. Это что-то вроде того, как легко пытаться пересесть с механической коробки передач на автоматическую и как сложно проделать путь в обратном направлении.

Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие.

## **Выбор среды разработки:** Code::Blocks

Code::Blocks — свободная кроссплатформенная среда разработки. Code::Blocks написана на С++ и использует библиотеку wxWidgets. Имея открытую архитектуру, может масштабироваться за счёт подключаемых модулей. Поддерживает языки программирования С, С++, D (с ограничениями), Fortran.

Code::Blocks разрабатывается для Windows, Linux и Mac OS X. Среду можно собрать из исходников практически под любую Unix-подобную систему, например FreeBSD, PC-BSD.

## **Выбор поиска кратчайшего пути:** Алгоритм Ли

Алгоритм волновой трассировки (волновой алгоритм, алгоритм Ли) — алгоритм поиска пути, алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину.

В основном используется при компьютерной трассировке (разводке) печатных плат, соединительных проводников на поверхности микросхем. Другое применение волнового алгоритма — поиск кратчайшего расстояния на карте в компьютерных стратегических играх.

Волновой алгоритм в контексте поиска пути в лабиринте был предложен Э. Ф. Муром. Ли независимо открыл этот же алгоритм при формализации алгоритмов трассировки печатных плат в 1961 году.

# **Практическая часть**

## **Описание алгоритма Ли**

Алгоритм работает на *дискретном рабочем поле* (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е при поиске пути их можно проходить, «непроходимые» (препятствия), путь через эту ячейку запрещён, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь.

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути.

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки.

Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке.

Соседние ячейки принято классифицировать двояко: в смысле [окрестности Мура](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%9C%D1%83%D1%80%D0%B0) и [окрестности фон Неймана](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D1%80%D0%B5%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D1%84%D0%BE%D0%BD_%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD%D0%B0), отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченная числом шагов от стартовой ячейки, становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек. Трасс с минимальной числовой длиной пути, как при поиске пути в окрестностях Мура, так и фон Неймана может существовать несколько. Выбор окончательного пути в приложениях диктуется другими соображениями, находящимися вне этого алгоритма. Например, при трассировке печатных плат — минимумом линейной длины проложенного проводника.

## **Используемые библиотеки:**

**<iostream>** — заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++. Он включён в стандартную библиотеку C++.

**<fstream>** (сокращение от «FileStream») — заголовочный файл из стандартной библиотеки C++, включающий набор классов, методов и функций, которые предоставляют интерфейс для чтения/записи данных из/в файл.

**<sstream>** — заголовочный файл с классами, функциями и переменными для организации работы со строками, через интерфейс потоков, в языке программирования C++. Он включён в стандартную библиотеку C++.

**<vector>** – это библиотека, в которой находится шаблон класса для контейнеров последовательности. Вектор хранит элементы заданного типа в линейном упорядочении и обеспечивает быстрый произвольный доступ к любому элементу. Вектор является предпочтительным контейнером для последовательности, когда производительность произвольного доступа имеет уровень "Премиум".

**<math.h>** — заголовочный файл стандартной библиотеки языка программирования С, разработанный для выполнения простых математических операций. Большинство функций привлекают использование чисел с плавающей точкой. C++ также реализует данные функции для обеспечения совместимости, все они содержатся в заголовочном файле cmath.

**<algorithm>** — заголовочный файл в стандартной библиотеке языка программирования C++, включающий набор функций для выполнения алгоритмических операций над контейнерами и над другими последовательностями.

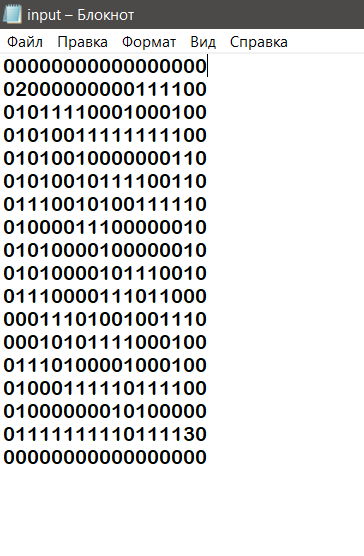
**<queue>** – это библиотека, в которой существует абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO, англ. first in, first out). Добавление элемента (принято обозначать словом enqueue — поставить в очередь) возможно лишь в конец очереди, выборка — только из начала очереди (что принято называть словом dequeue — убрать из очереди), при этом выбранный элемент из очереди удаляется.

**<iomanip>** - реализует инструменты для работы с форматированием вывода, например базу, используемую при форматировании целых и точных значений чисел с плавающей запятой.

**<windows.h>** - специфический файл заголовка для C и C ++ языки программирования, который содержит объявления для всех функций в Windows API, все общие макросы используются программистами для Windows, а также все типы данных, используемые различными функциями и подсистемы. Он определяет очень большое количество специфических функций Windows, которые могут быть использованы в С.

## **Ход работы программы:**

1) Есть файл «input.txt», заполненный числами, где 0 – стена, 1 – проход, 2 – начальное положение игрока, 3 – выход. Файл выглядит следующим образом:



2) Далее, программа считывает этот файл, оформляя красным цветом стенки и зеленым проходы. Программа запоминает начальную точку и конечную. Таким образом, 2 и 3 заменяется на 1. Это нужно для дальнейшей обработки. Вывод на экран карты полученного лабиринта:



3) Следующий шаг – это поиск кратчайшего пути. Программа обрабатывает карту, используя алгоритм Ли и выводит на экран кратчайший путь: 

Таким образом, короткий путь состоит из 46 шагов

# **Заключение**

Таким образом, удалось создать программу, которая умеет находить кратчайший путь в лабиринте, полученного из файла, с помощью реализации алгоритма Ли.

# **Приложения**

**#include <iostream>**

**#include <fstream>**

**#include <sstream>**

**#include <vector>**

**#include <cmath>**

**#include <algorithm>**

**#include <queue>**

**#include <iomanip>**

**#include <Windows.h>**

**using namespace std;**

**struct Point {**

**int i;**

**int j;**

**Point() {}**

**Point(int i, int j) {**

**this->i = i;**

**this->j = j;**

**}**

**};**

**enum ConsoleColor {**

**Black = 0,**

**Blue = 1,**

**Green = 2,**

**Cyan = 3,**

**Red = 4,**

**Magenta = 5,**

**Brown = 6,**

**LightGray = 7,**

**DarkGray = 8,**

**LightBlue = 9,**

**LightGreen = 10,**

**LightCyan = 11,**

**LightRed = 12,**

**LightMagenta = 13,**

**Yellow = 14,**

**White = 15**

**};**

**HANDLE hConsoleHandle = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**void SetColor(ConsoleColor text, ConsoleColor background) {**

**SetConsoleTextAttribute(hConsoleHandle, (WORD)((background << 4) | text));**

**}**

**int main() {**

**fstream input("input.txt", fstream::in);**

**Point start\_point, end\_point, boundary\_point;**

**vector<vector<int>> map;**

**vector<int> row;**

**int row\_len;**

**string line;**

**int i = 0, j;**

**while (getline(input, line)) {**

**row.resize(line.length());**

**for (j = 0; j < line.length(); j++) {**

**row[j] = line[j] - '0';**

**if (row[j] == 2) {**

**start\_point = Point(i, j);**

**row[j] = 1}**

**else if (row[j] == 3) {**

**end\_point = Point(i, j);**

**row[j] = 1;**

**}**

**}**

**i++;**

**map.push\_back(row);**

**}**

**boundary\_point.i = map.size() - 1;**

**boundary\_point.j = map[0].size() - 1;**

**input.close();**

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

**for (int j = 0; j < map[0].size(); j++) {**

**if (map[i][j] == 0)**

**SetColor(White, Red);**

**else**

**SetColor(Black, Green);**

**cout << setw(4) << map[i][j];**

**}**

**cout << endl;**

**}**

**cout << endl;**

**queue<Point> free\_points;**

**free\_points.push(start\_point);**

**Point current;**

**int d = 2;**

**while (free\_points.size() != 0) {**

**int run\_length = free\_points.size();**

**for (int k = 0; k < run\_length; k++) {**

**current = free\_points.front();**

**free\_points.pop();**

**i = current.i;**

**j = current.j;**

**if (j - 1 >= 0 && map[i][j - 1] == 1) {**

**map[i][j - 1] = d;**

**free\_points.push(Point(i, j - 1));}**

**if (j + 1 <= boundary\_point.j && map[i][j + 1] == 1) {**

**map[i][j + 1] = d;**

**free\_points.push(Point(i, j + 1));**

**}**

**if (i - 1 >= 0 && map[i - 1][j] == 1) {**

**map[i - 1][j] = d;**

**free\_points.push(Point(i - 1, j));**

**}**

**if (i + 1 <= boundary\_point.i && map[i + 1][j] == 1) {**

**map[i + 1][j] = d;**

**free\_points.push(Point(i + 1, j));**

**}**

**}**

**d += 1;}**

**map[start\_point.i][start\_point.j] = 1;**

**vector<Point> path;**

**i = end\_point.i, j = end\_point.j;**

**current = Point(end\_point.i, end\_point.j);**

**vector<vector<int>> viewing\_map(map);**

**viewing\_map[end\_point.i][end\_point.j] = d + 1;**

**while (map[current.i][current.j] != 1) {**

**if (current.j + 1 <= boundary\_point.j && map[current.i][current.j + 1] == map[current.i][current.j] - 1) current.j += 1;**

**else if (current.i + 1 <= boundary\_point.i && map[current.i + 1][current.j] == map[current.i][current.j] - 1) current.i += 1;**

**else if (current.i - 1 >= 0 && map[current.i - 1][current.j] == map[current.i][current.j] - 1) current.i -= 1;**

**else if (current.j - 1 >= 0 && map[current.i][current.j - 1] == map[current.i][current.j] - 1) current.j -= 1;**

**else**

**cout << "Путь отсутствует!";**

**path.push\_back(current);**

**viewing\_map[current.i][current.j] = d + 1;}**

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

**for (int j = 0; j < map[0].size(); j++) {**

**if (viewing\_map[i][j] == d + 1)**

**SetColor(Black, Cyan);**

**else if (viewing\_map[i][j] == 0)**

**SetColor(White, Red);**

**else**

**SetColor(Black, Green);**

**cout << setw(4) << map[i][j];**

**}**

**cout << endl;**

**}**

**return 0;}**

# **Литература**

<https://ru.wikipedia.org>

<https://ravesli.com>

<https://www.youtube.com>

<https://github.com>

<https://habr.com>

<https://drive.google.com/file/d/13Q3Zm0XOiBtji6wH1O8MJ1wC9E0W8Zb0/view>