**Аннотация**

Целью проделанной работы было научиться применять полученные знания для самостоятельного решения на ПК конкретных практических задач.

По разрешению преподавателя выбрать задание, была спроектирована и написана программа, выполняющая требующиеся в задании действия. После этого был написан отчёт о проделанной работе.

В процессе работы были получены как навыки в разработке программ, так и опыт по части подготовки курсовых работ и отчётов по ним.

**Оглавление**

1. Постановка задачи…………………………………………………… 4

2. Формализация задачи .......................................................................... 5

2.1. Математические расчеты........................................................ 6

2.2. Работа программы………………………………………….... 14

3. Блок-схема алгоритма………………………………………………... 16

4. Текст программы …………………………………………………….. 17

5. Руководство пользователя…………………………………………… 23

5.1. Функции и их назначение…………………………………… 24

5.2. Идентификаторы и их назначение………………………….. 28

6. Заключение……………………………………………………………. 30

**Тема: Использование ЭВМ в анализе задачи о ходе коня.**

1. **Постановка задачи.**

Использовать математический аппарат и ЭВМ для моделирования путешествия коня по шахматной доске. Конь - шахматная фигура, которая передвигается по шахматной доске буквой «Г», а именно на два поля в любом из четырёх направлений и на одно поле ортогонально предыдущему направлению. Если условиться называть поля шахматной доски вершинами графа, то задача требует, чтобы конь, начав передвигаться по шахматной доске из заданного поля, совершил гамильтонов путь. Задача была предложена Эйлером на рассмотрение математиков. Были разработаны алгоритмы решения данной задачи.

Конечная цель данной программы – смоделировать путешествие коня и определить количество простых цепей каждой длины для получения числа, равного вероятности события А, где А – событие «конь совершил гамильтонов путь с первого раза».

Todd K. Moon, автор учебного пособия «Теория Кодирования: математические методы и алгоритмы» Университета Штата Юта, пишет: «понимание любого предмета обычно повышается, когда студент имеет шанс обучить кого-то или что-то. Студенту понадобится углубленное знание материала, чтобы «обучить» компьютер выполнять задачи».

Это поможет лучше понять приложения теории графов, применять изученные в университете дисциплины.

1. **Формализация.**

Стратегия, используемая в нижеследующем решении, является правилом Варнсдорфа. Правило Варнсдорфа для данной задачи формулируется так:

*При обходе доски конь следует на то поле, с которого можно пойти на минимальное число ещё не пройденных полей. Если таких полей несколько, то можно пойти на любое из них.*

Долгое время считалось, что правило Варнсдорфа работает безукоризненно. Позднее с помощью компьютеров была установлена неточность во второй его части: если существует несколько подходящих полей, то не все они равноценны, и произвольный выбор поля может завести коня в тупик. На практике, однако, вероятность попадания в тупик невелика даже при вольном пользовании второй частью правила Варнсдорфа. Автор решил модифицировать правило Варнсдорфа на следующее, рекурентное:

*При обходе доски конь следует на то поле, с которого можно пойти на минимальное число ещё не пройденных полей. Если таких полей несколько, то следует применить этот же алгоритм для таких полей.*

В данной задаче используются массивы, указатели, функции. Программа реализуется с помощью конечного числа циклов, условий, итераций и арифметических действий. Практическая часть решения задачи предполагает два этапа:

1. Силовой метод.
2. Стратегический метод.
   1. Правило Варнсдорфа.
   2. Модифицированное правило Варнсдорфа.

Стратегический метод с использованием модифицированного правила Варнсдорфа гарантирует успех, однако он требует больших ресурсных затрат.

Можно описать первый метод следующими действиями:

1. Инициализация переменных
2. Цикл до успешного завершения
3. Запись результатов в файл

Второй метод, пункт б), можно описать следующими действиями:

1. Инициализация переменных
2. Просчитывание ходов до оптимальных условий завершения
3. Успешное завершение
   1. ***Математические расчеты***

В данном разделе будут выполнены пункты 3, 4, 5 раздела постановки задачи.

Для пункта 4 следует рассчитать теоретическую вероятность успешного маршрута вообще. Эта вероятность равна

P(64) = ***количество маршрутов***

***количество всевозможных путей***

Если учесть что количество всех замкнутых маршрутов коня равно

26 534 768 821 064

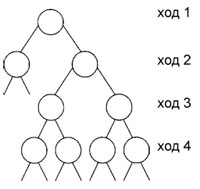
Количество всех незамкнутых маршрутов коня равно

19 591 828 170 979 904

Получается, что общее число всевозможных маршрутов равно 19 618 362 899 800 968

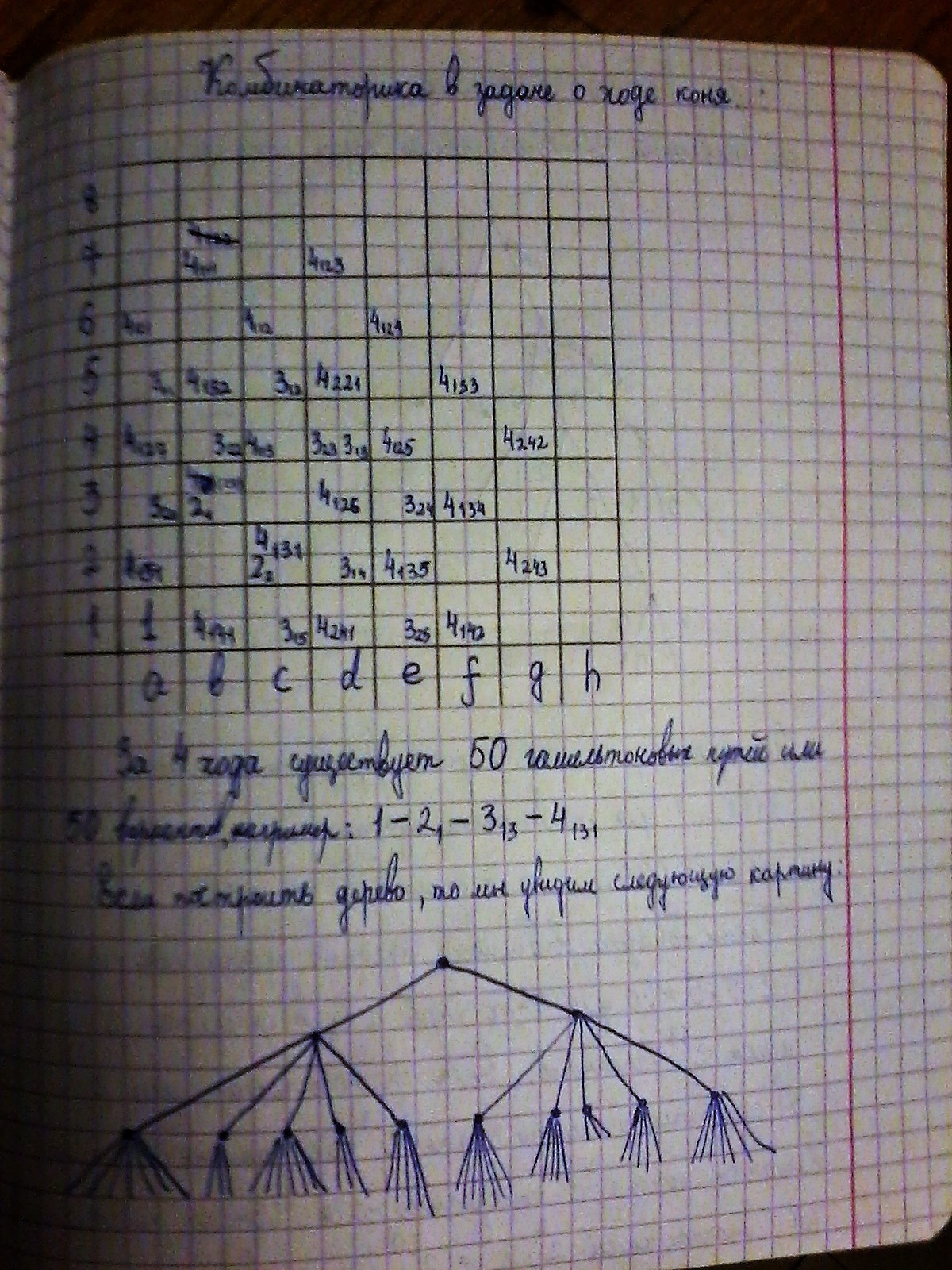
Нам остаётся только посчитать количество всевозможных путей (маршрутов и неполных путешествий). Доктора физико-математических наук Р. БАХТИЗИН, К. ШТУКАТУРОВ пишут следующее:

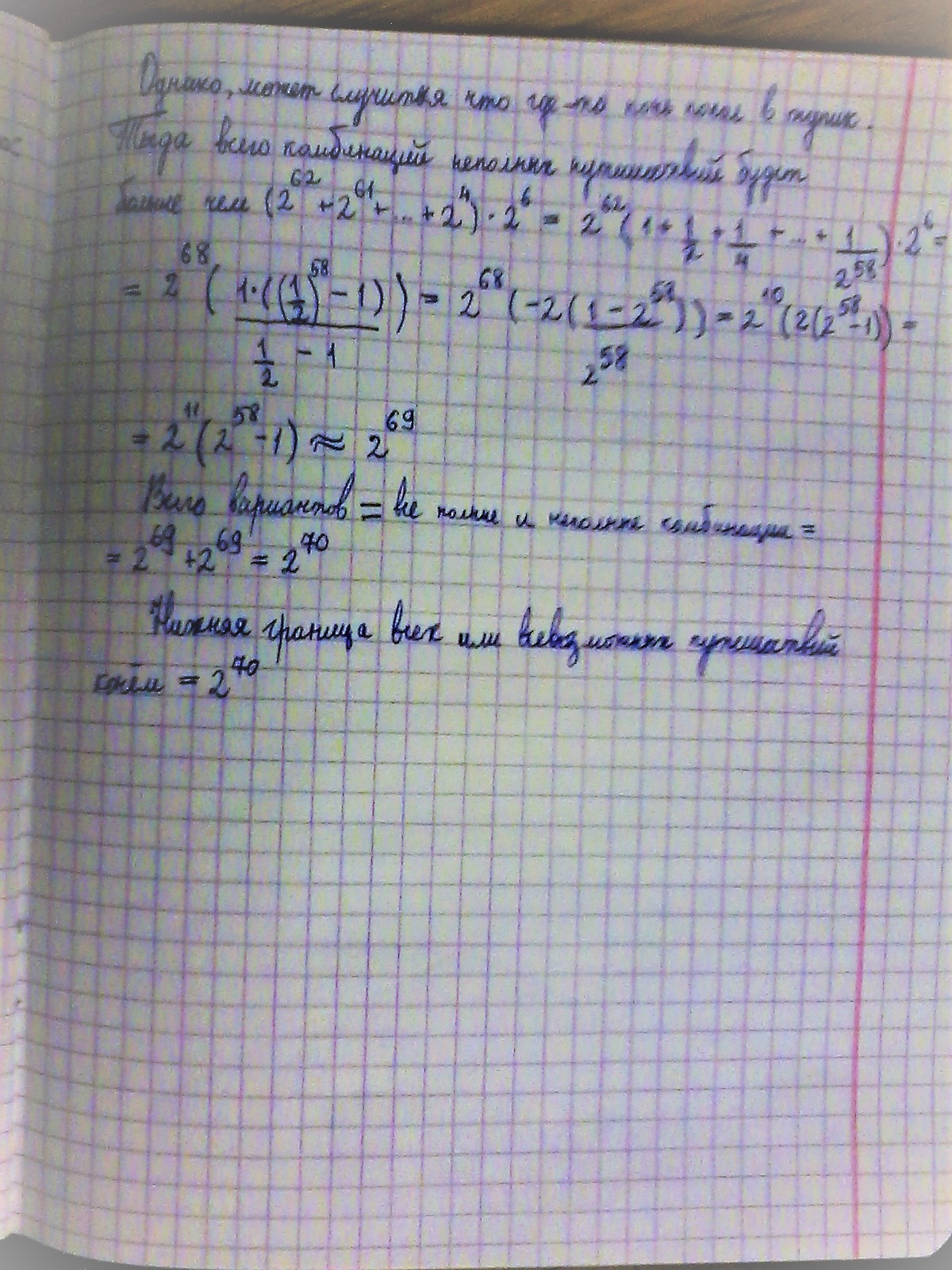
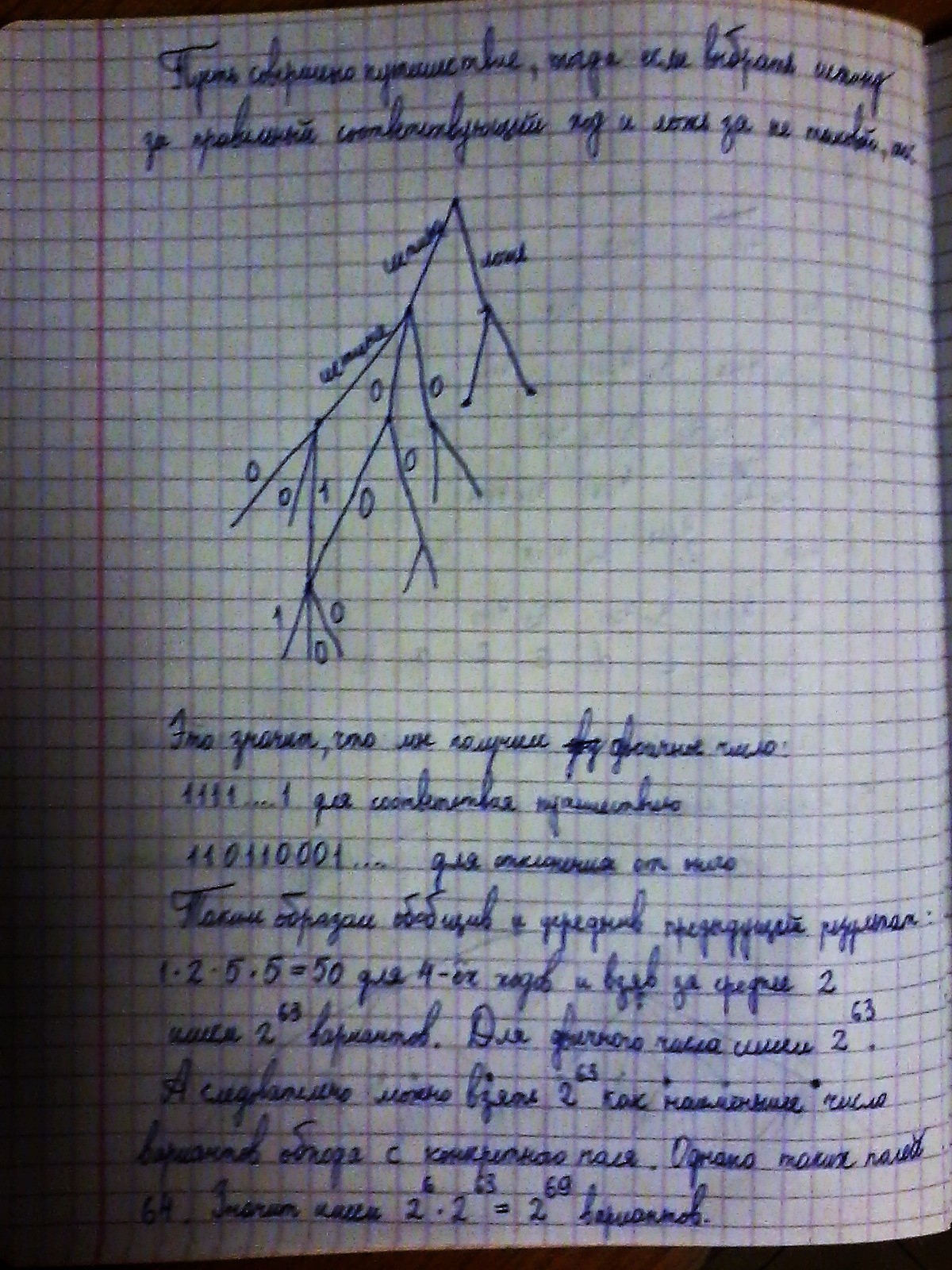
«В научно-популярной литературе нередко встречаются так называемые «шахматные» задачи, связанные с шахматными фигурами, например: можно ли ходом коня обойти доску размером 8x6, побывав на каждой клетке только один раз? Это не такая простая задача, как может показаться на первый взгляд. Нетрудно убедиться, что число возможных ходов из каждой клетки может быть от 0 до 8. Представим их в виде дерева, имеющего иерархическую структуру. Примем, что «среднее» число ходов равно двум:



Получается, что на 64-м ходу число возможных комбинаций достигает 264 …»

Пользуясь этим результатом можно попробовать посчитать общее количество путей.





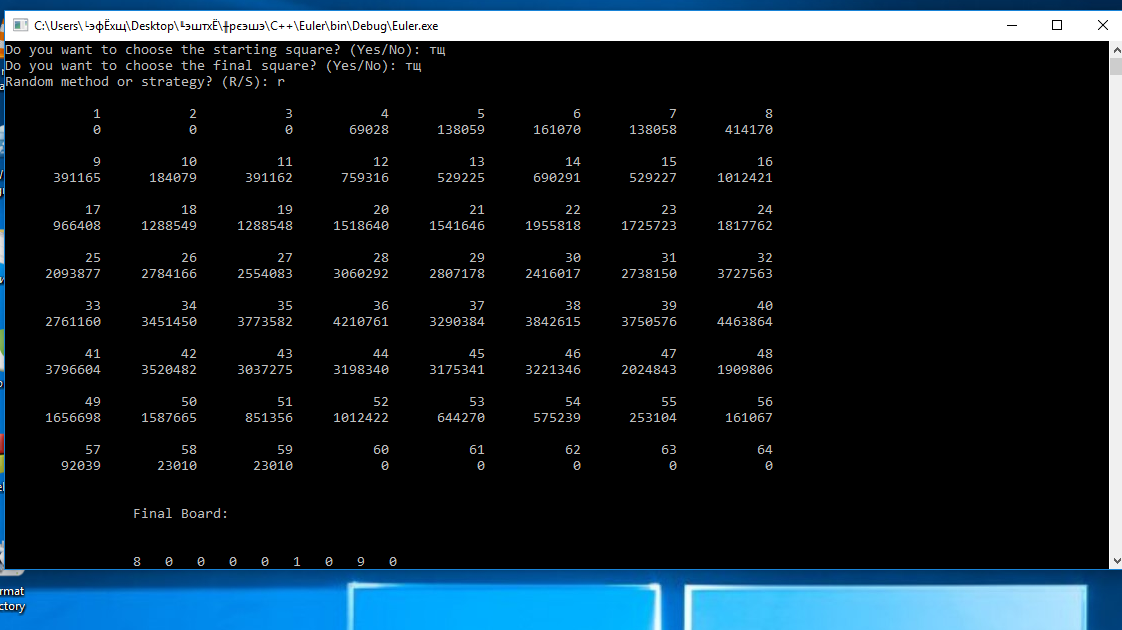
Таким образом, можно взять за нижнюю границу всевозможных путей число 270 ≈ 1021 путей. Значит P(64) 0,00001962

Или, если перейти в проценты, то P(64) 0,002%

На практике, однако, эта вероятность на порядки меньше. Это означает что нужно выполнить на порядки больше 105 попыток путешествий, каждое из которых вызывает функцию **makepossiblemove()** более 104 раз, что означает что нужно выполнить эту функцию на порядки больше 109 раз. Если допустить что все операторы этой функции выполняются за 1 такт процессора, то время выполнения полного путешествия будет более 1 секунды. Однако на практике, 109 пробных путешествий не дали положительного результата, что означает что нижняя граница невероятно далека от истинного значения.

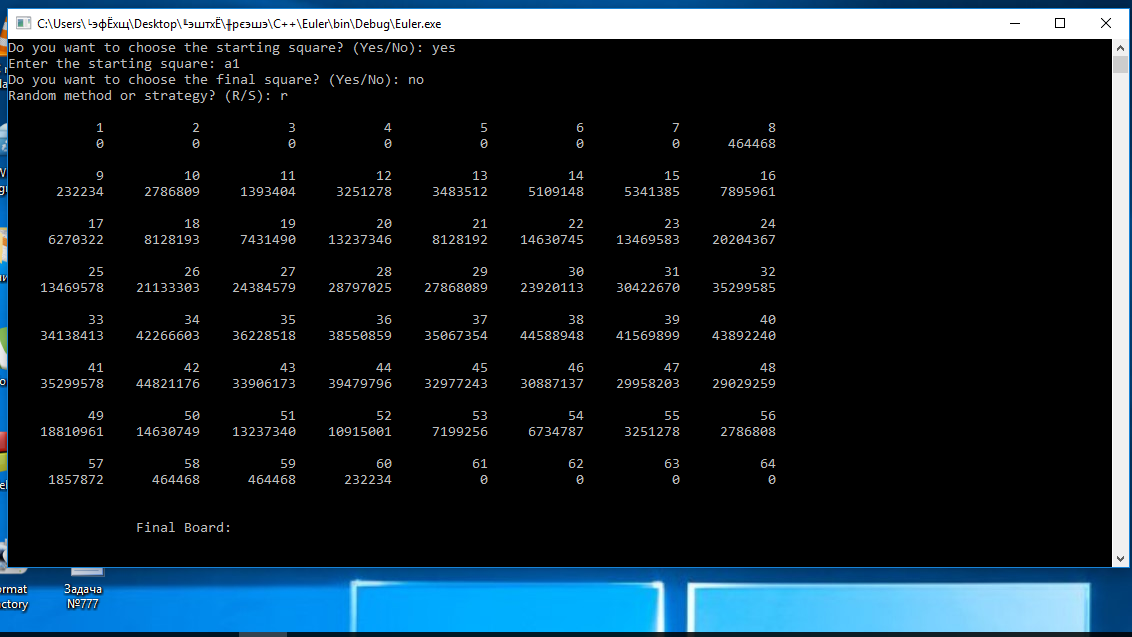
Для пункта 3 требовалось вычислить эмпирическую вероятность успешного маршрута силовым методом и были получены следующие результаты:

*Данные за миллион итераций (путешествий).*



Здесь можно предположить, что дальше 59 ходов конь не ходит, но это ошибочное предположение, поскольку при большем количество итераций мы убеждаемся в обратном.

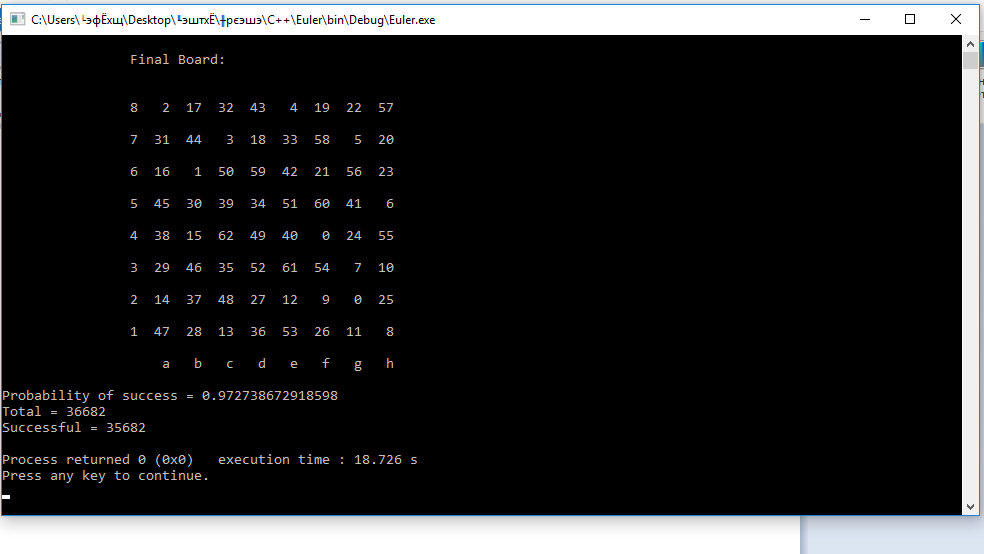
*Данные за миллиард итераций (путешествий).*



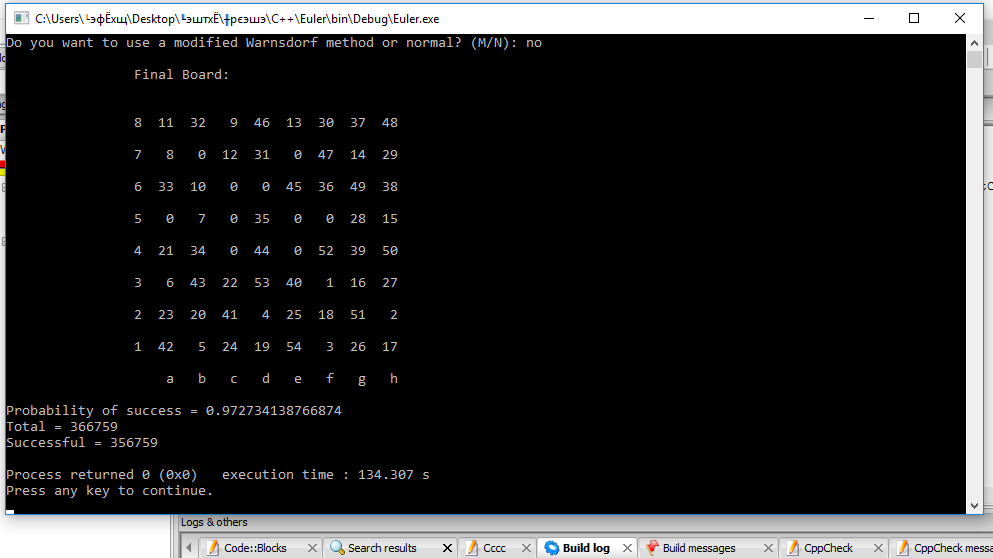
Здесь ясно видно, что вероятность путешествия тем меньше, чем оно длиннее. Автору так и не удалось получить маршрута жёстким силовым методом, что не удивительно, ибо вероятность такого события слишком мала как видно из распределения.

Для пункта 5 требуется получить эмпирическую вероятность стратегического метода. Были получены следующие данные:

*Данные за 36682 итерации.*



*Данные за 356759 итераций.*



Далее было обнаружено, что за миллион итераций, вероятность равнялась 0,977776.

Взяв среднее арифметическое из этих чисел получим:

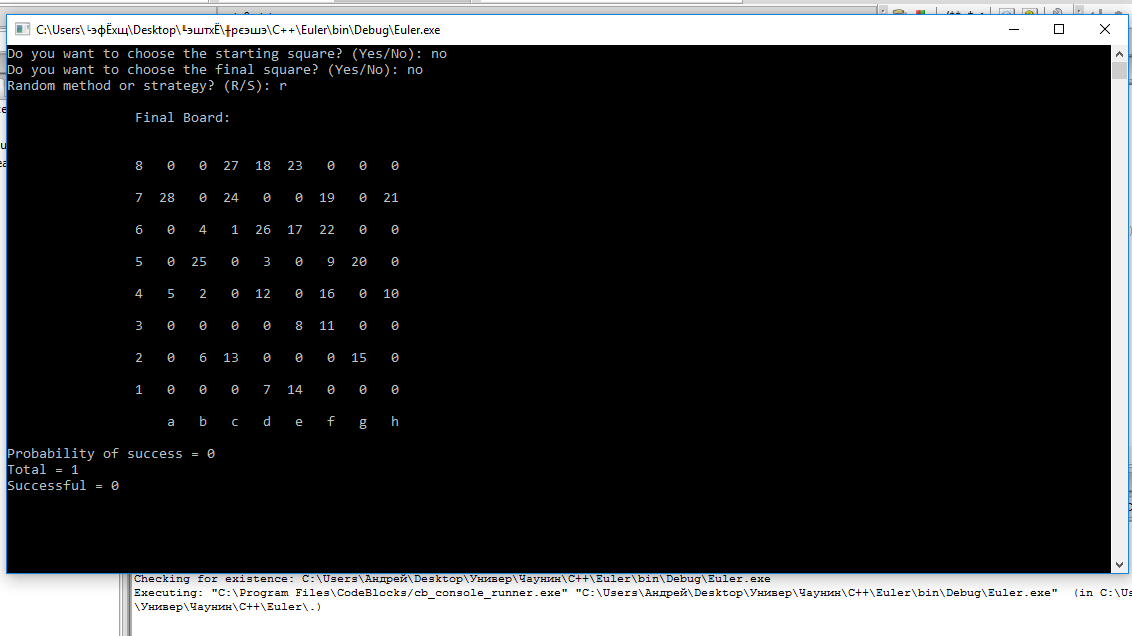
P(64) = = 0,974416

Это говорит о том, что вероятность попасть в тупик, используя вторую часть правила Варнсдорфа произвольно очень мала. Вычислить же теоретическую вероятность такого метода слишком затруднительно, потому что практически невозможно рассчитать количество всех путей, ибо они выбираются по комплексному алгоритму.

* 1. ***Работа программы***

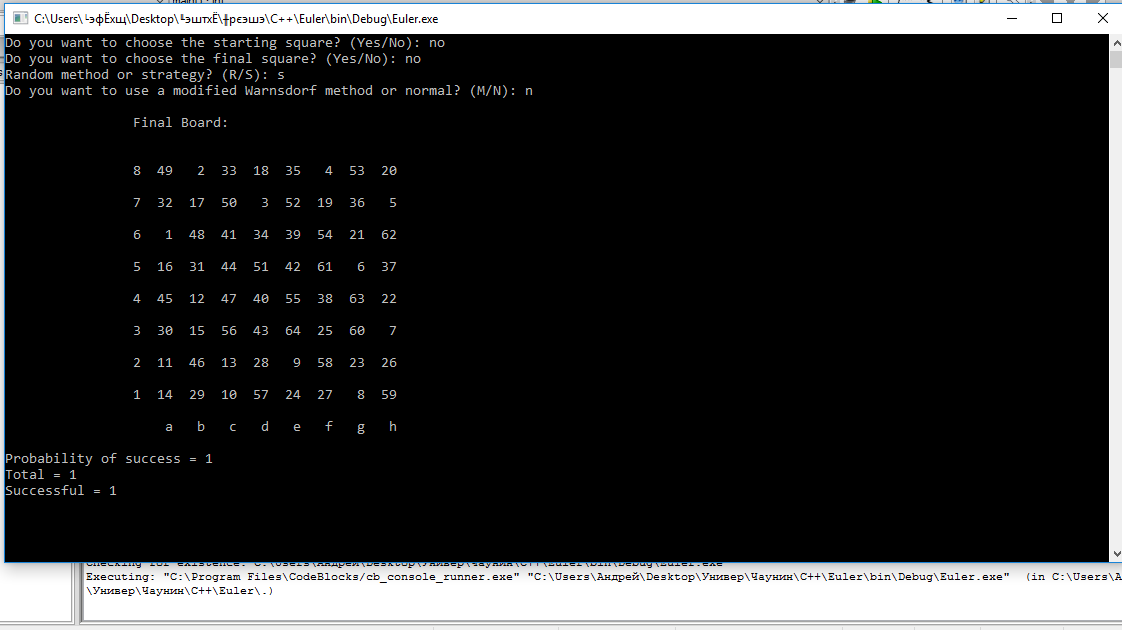
В данном разделе были выполнены пункты 1 и 2 раздела постановки задачи. Были получены следующие результаты:

*Обход силовым методом.*



Здесь отчетливо видно, что никакая стратегия не используется и все ходы делаются случайно. Каждый ход равноправен. Также видно что начальная позиция выбирается случайно, а конечная не учитывается.

*Обход стратегическим методом.*



Здесь же видно, как именно выбираются ходы. Начальная и конечная позиции также не учитываются. Вероятность установления маршрута, будучи велика, демонстрирует своё господство в данном методе: с первого раза выполнен маршрут.

1. **Блок схема алгоритма**

method2 = 0?

**нет**

**да**

Начало

board[8][8], accessibility[8][8], vertical[64], horizontal[64];

жесткий силовой метод

Конец

method1 = 0?

**да**

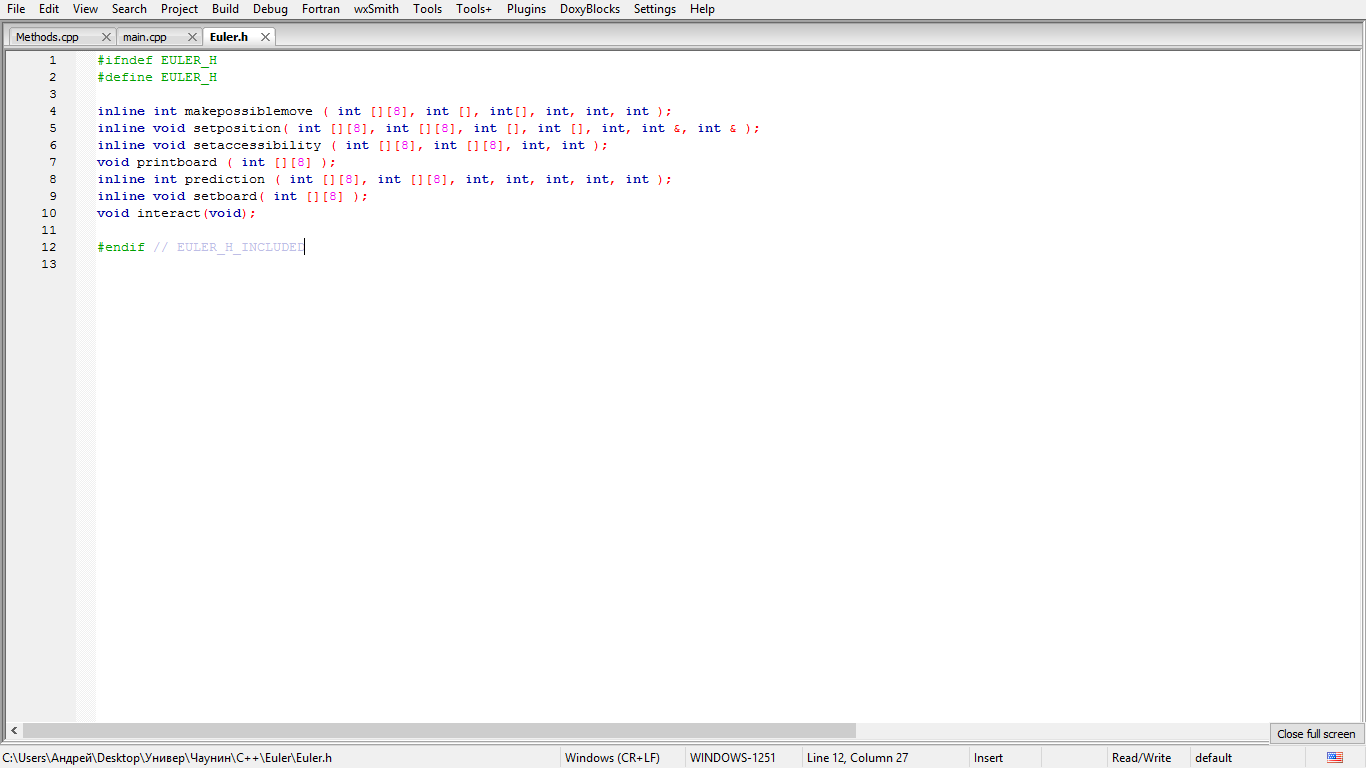
**нет**

алгоритм Варнсдорфа

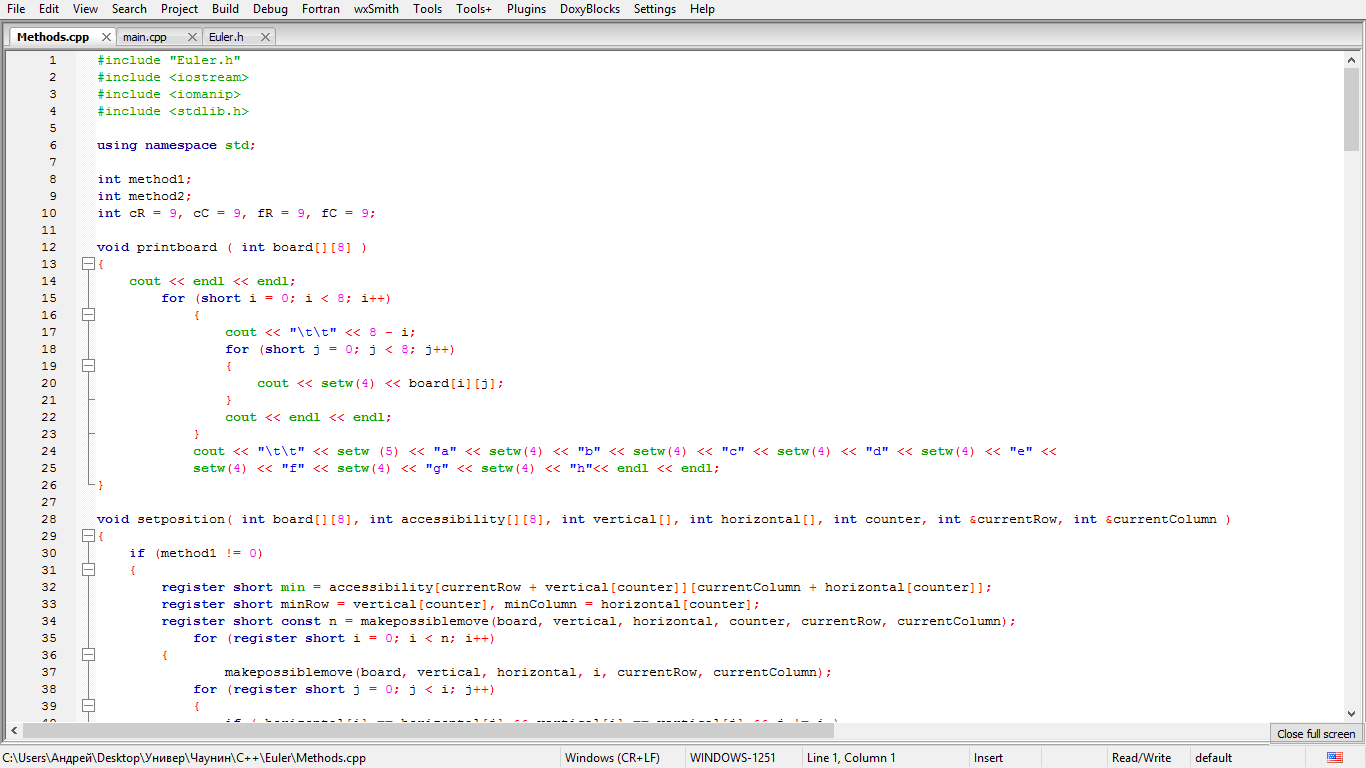
модификация алгоритма Варнсдорфа

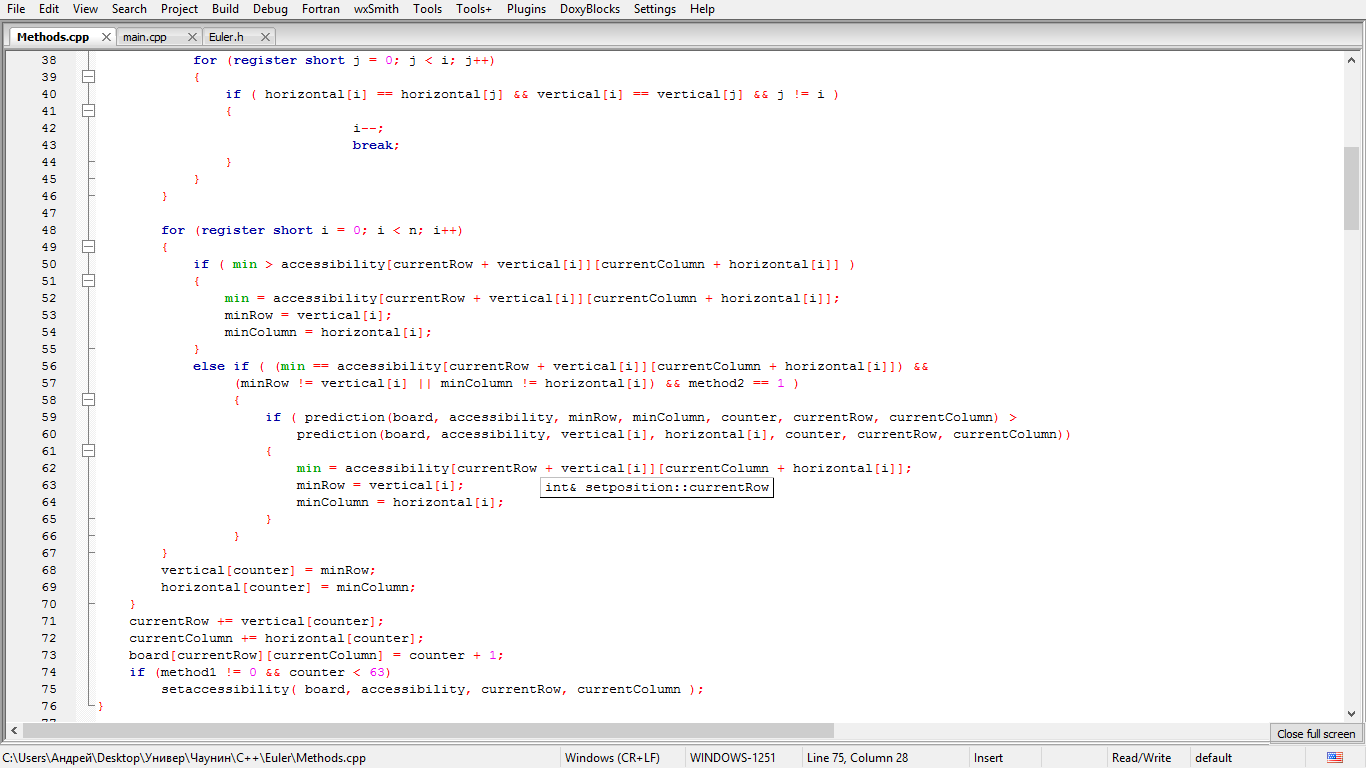
board[8][8];

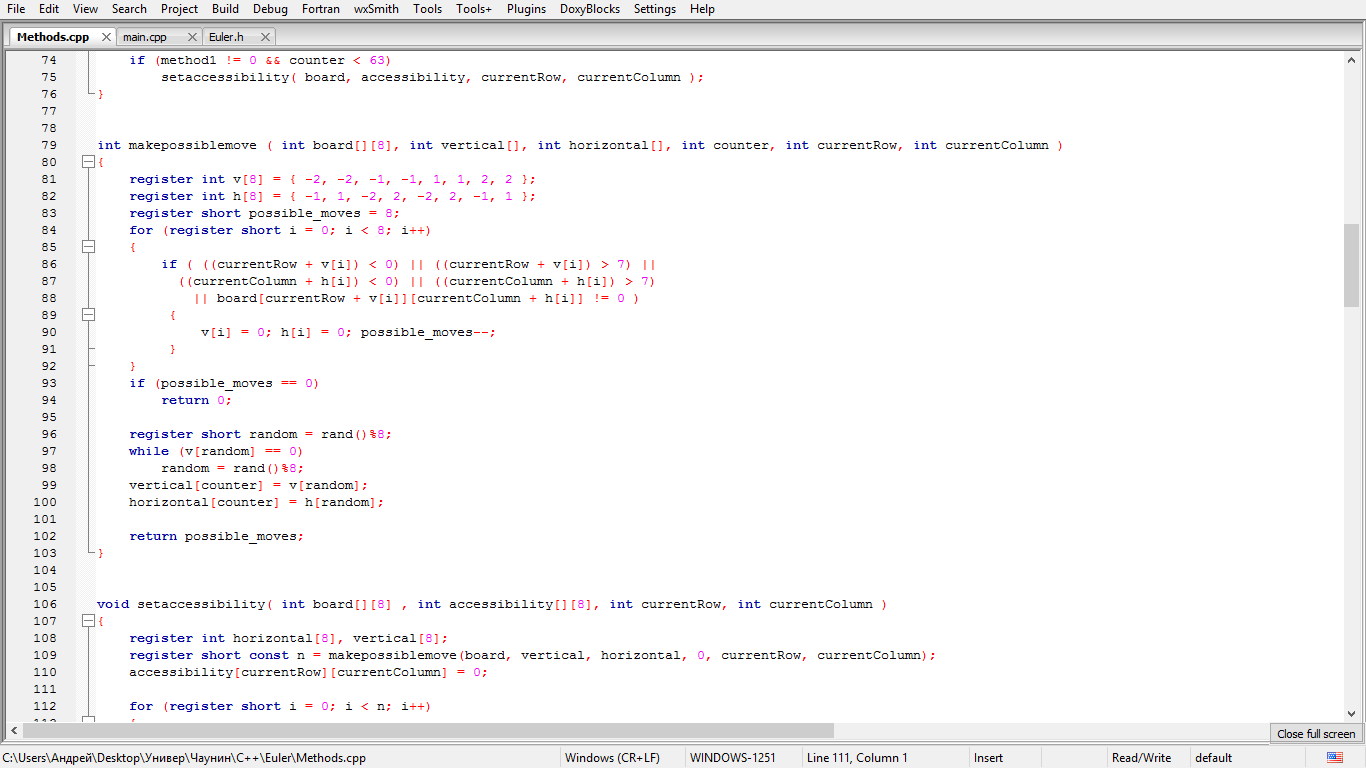
1. **Текст программы**

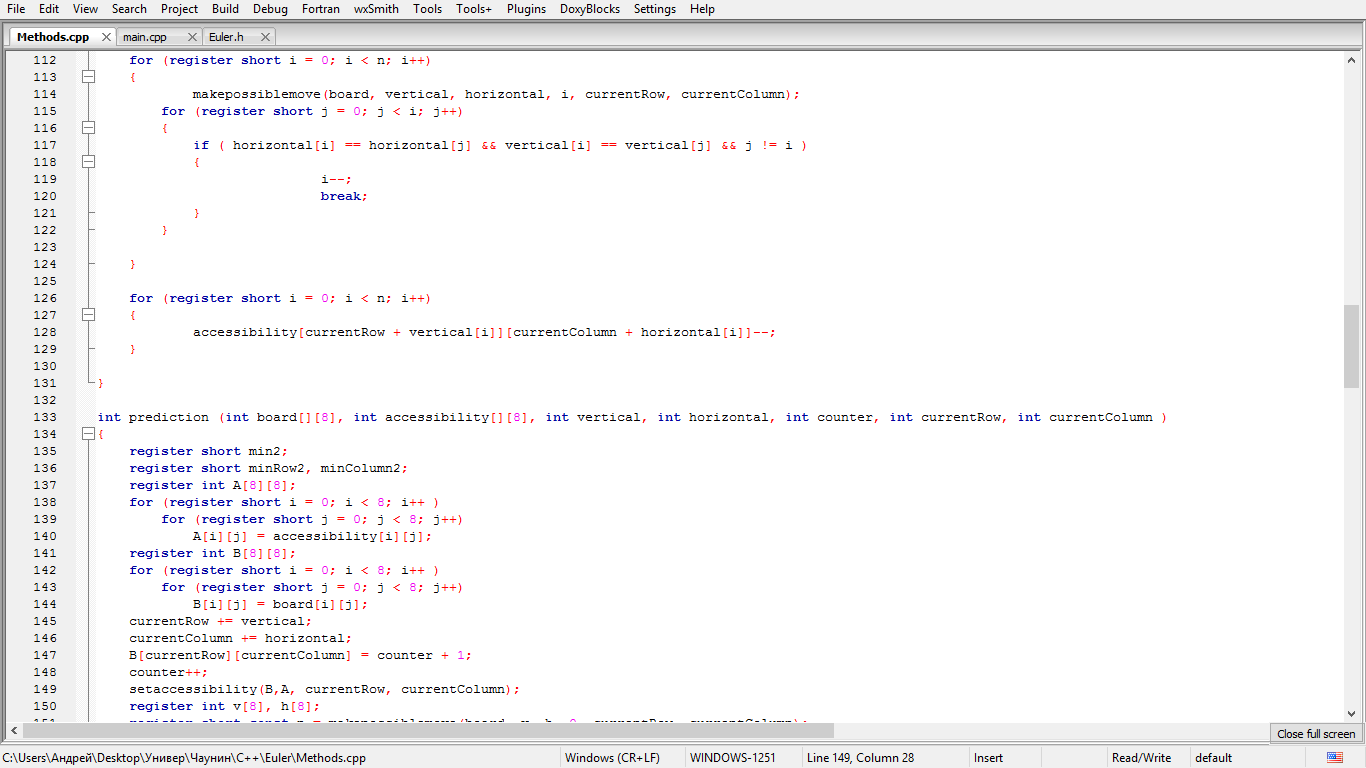


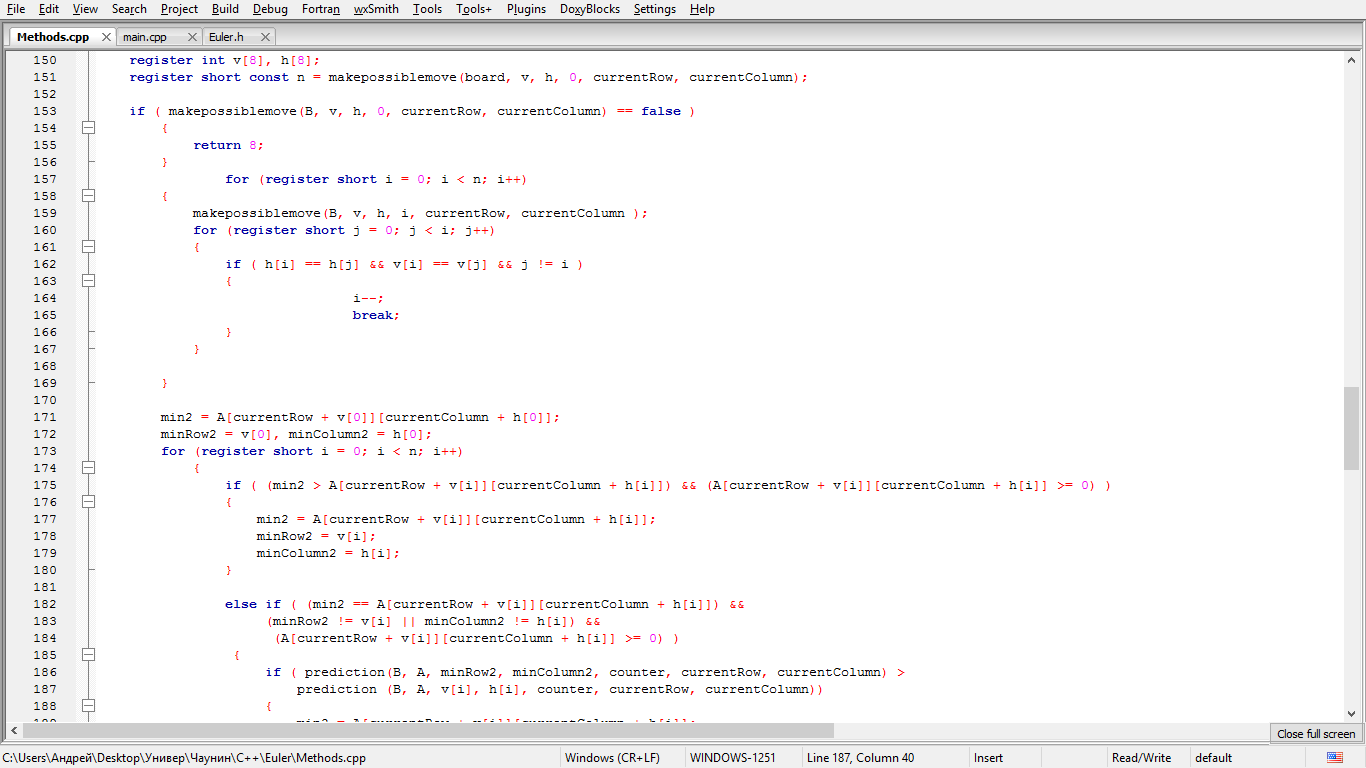
*Заголовочный файл Euler.h*

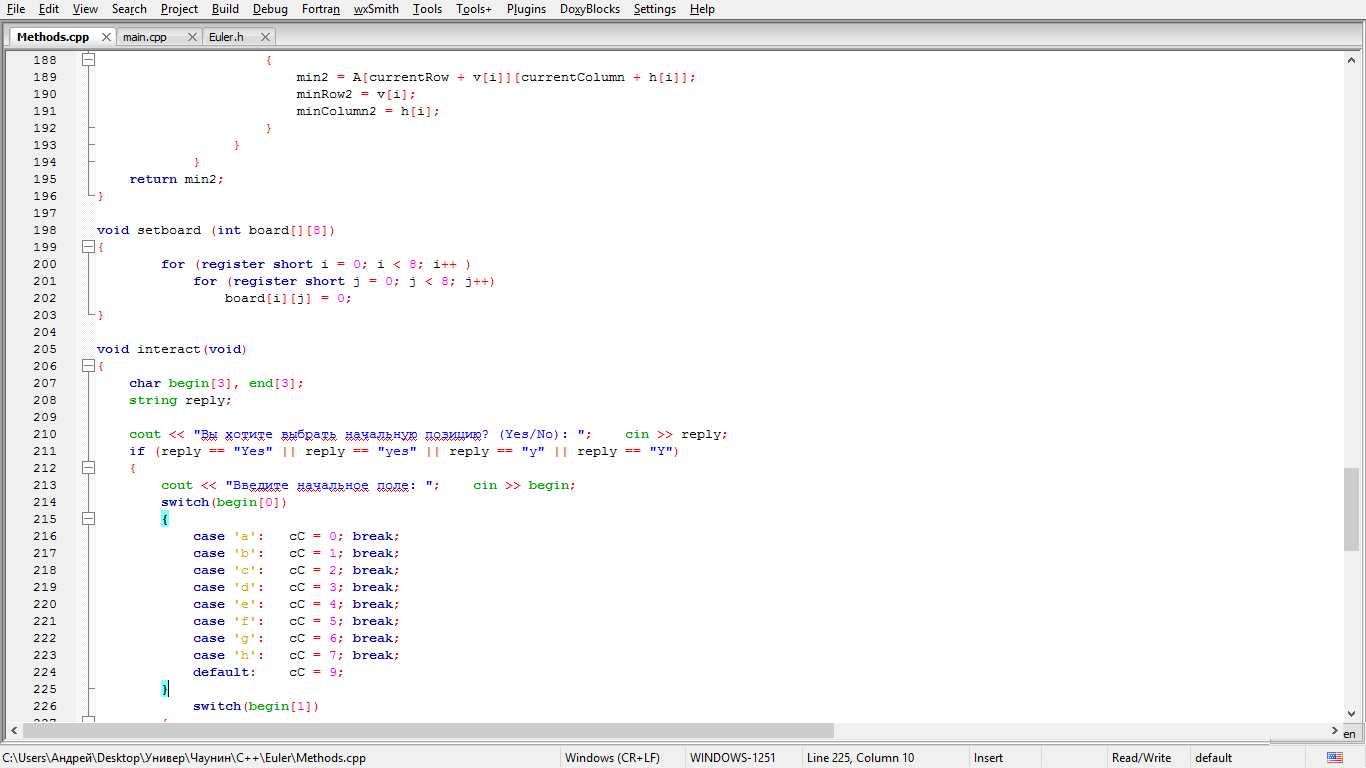


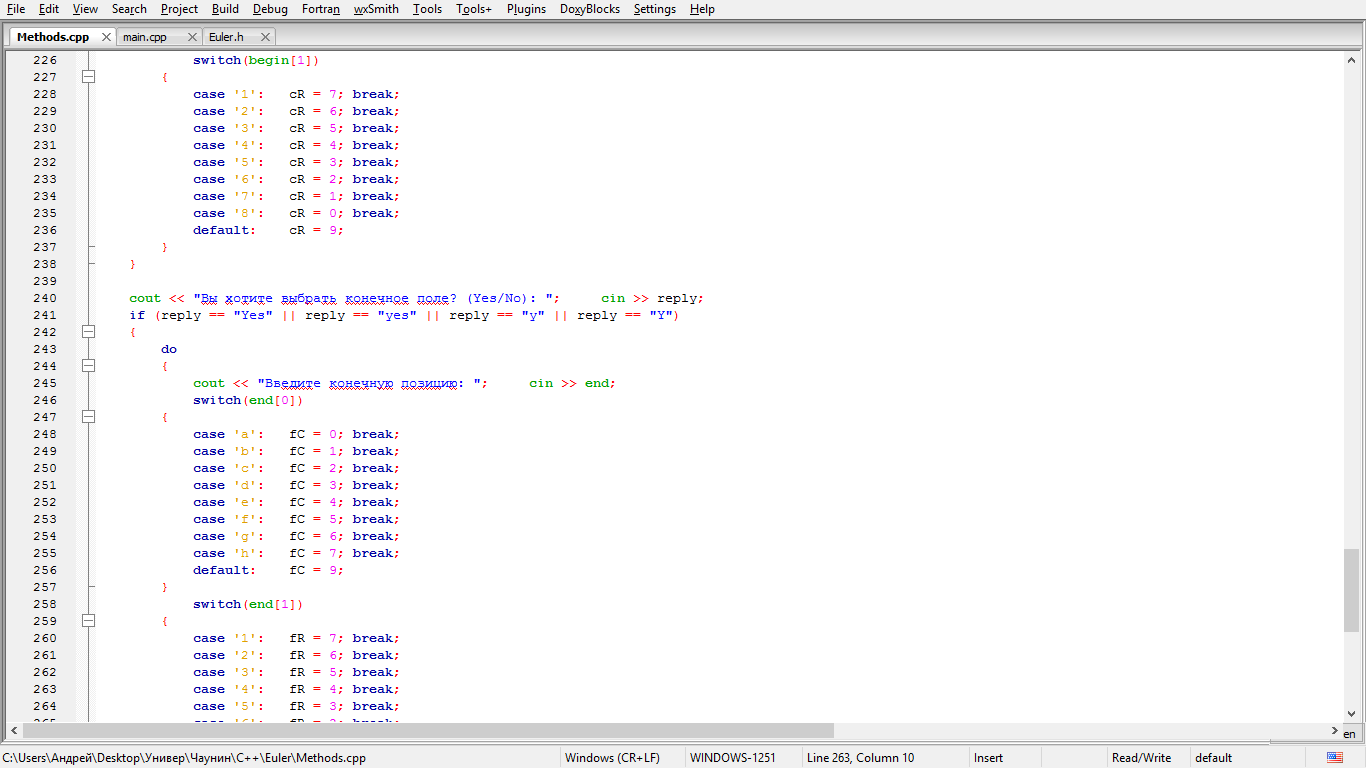


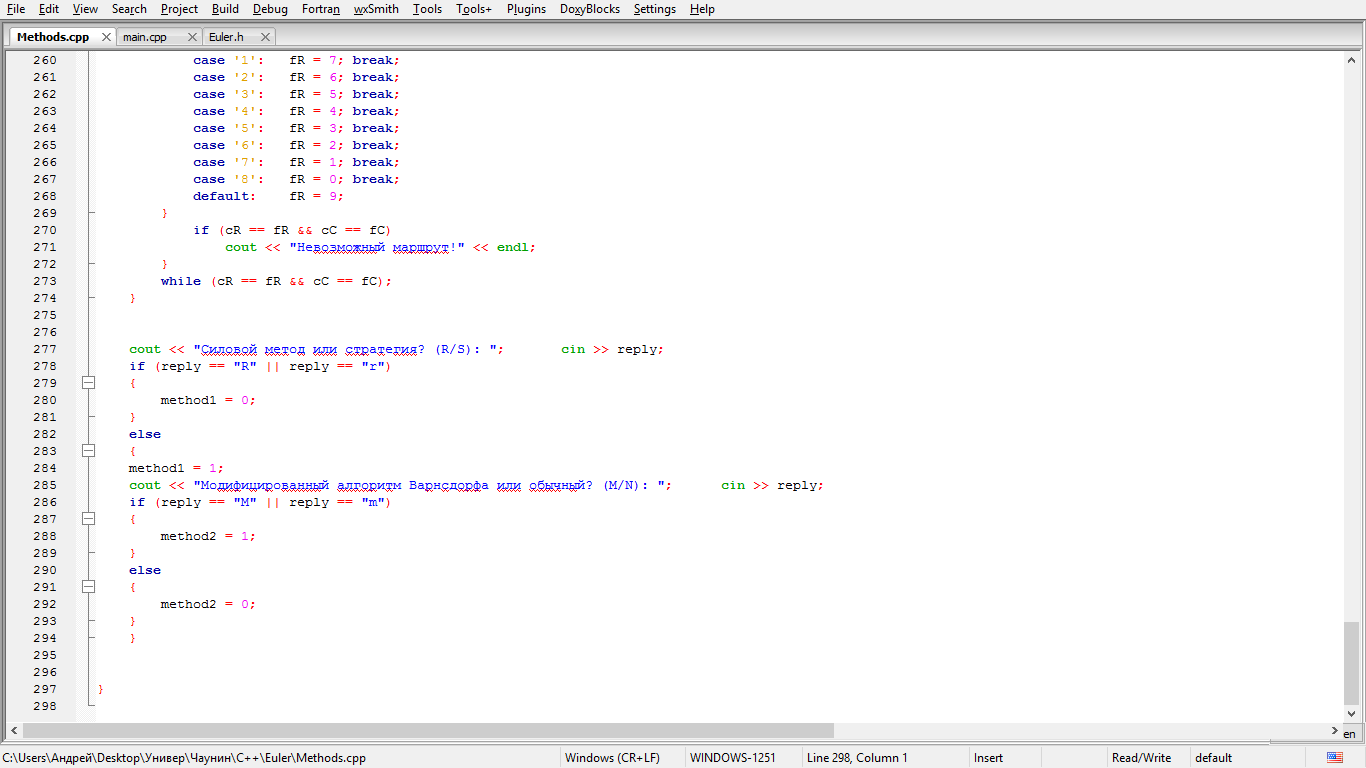




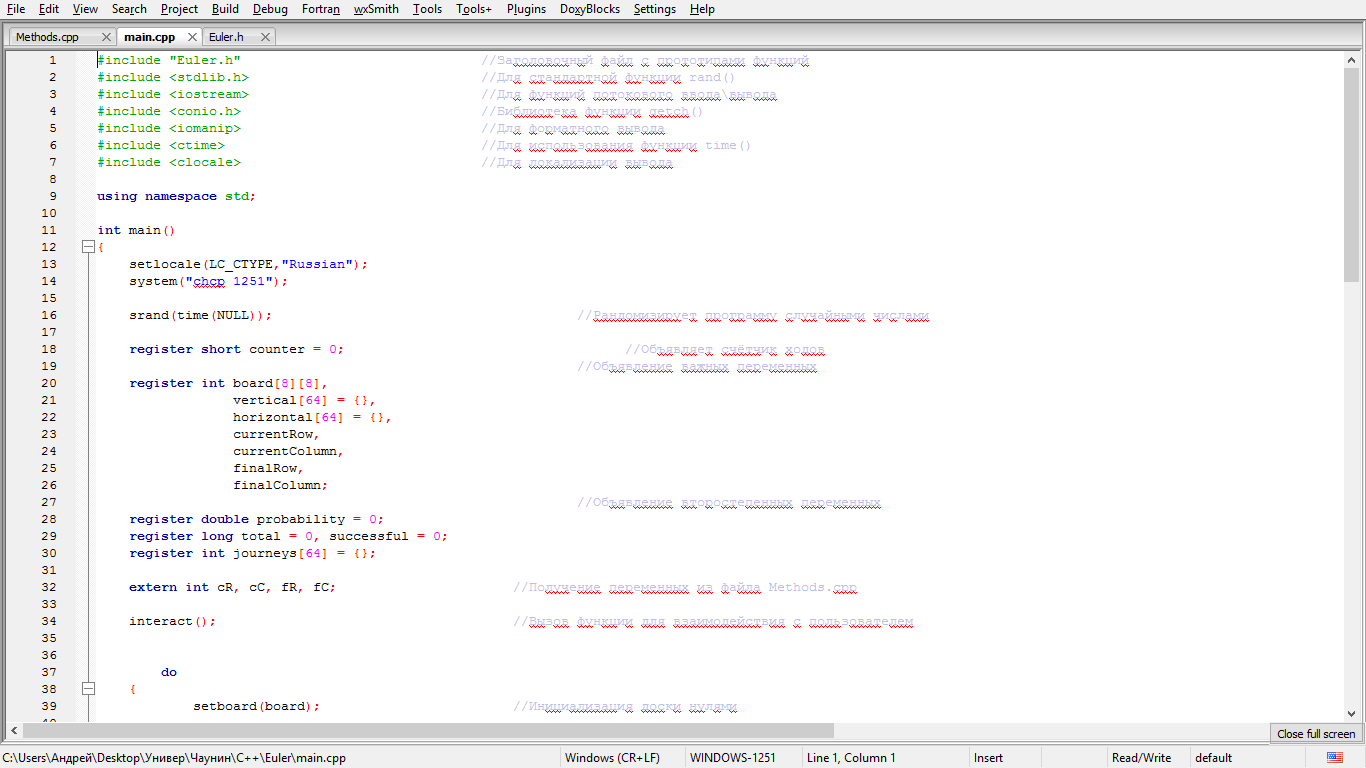


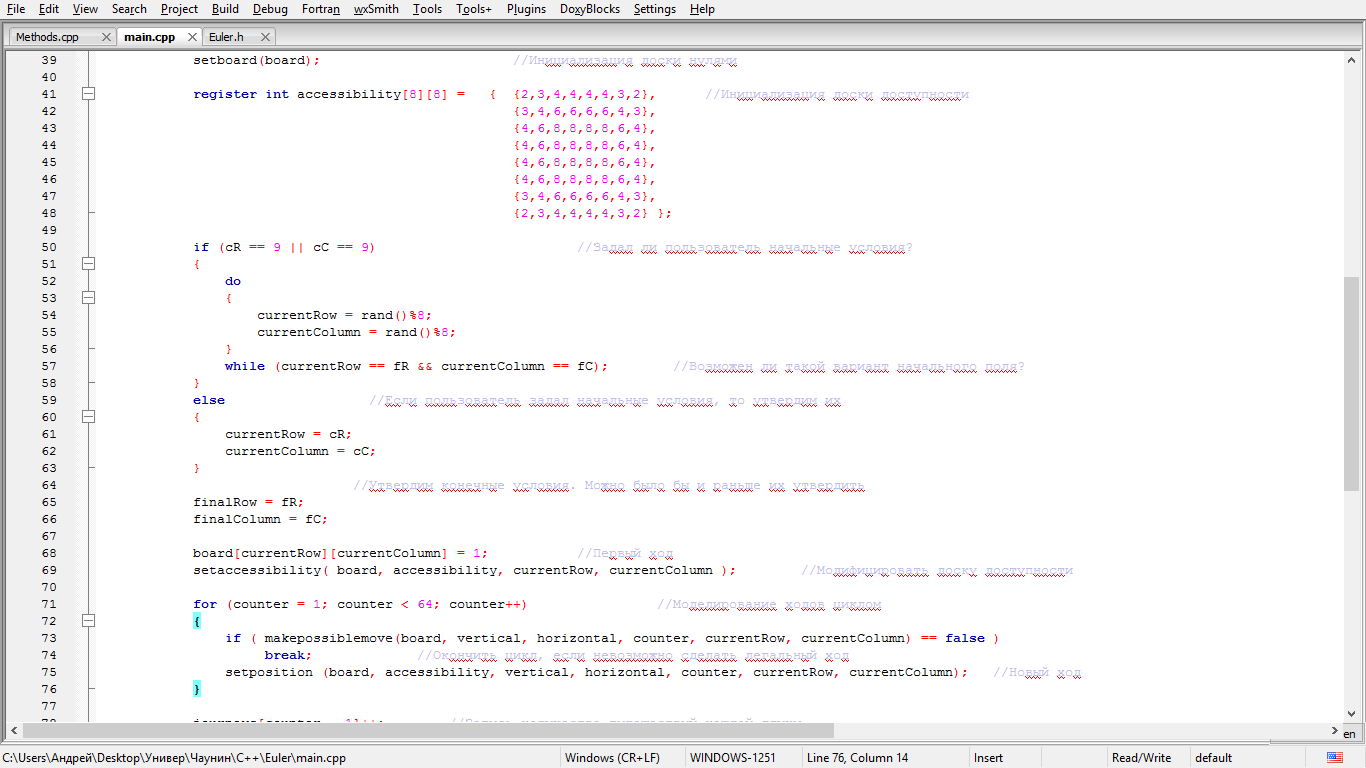


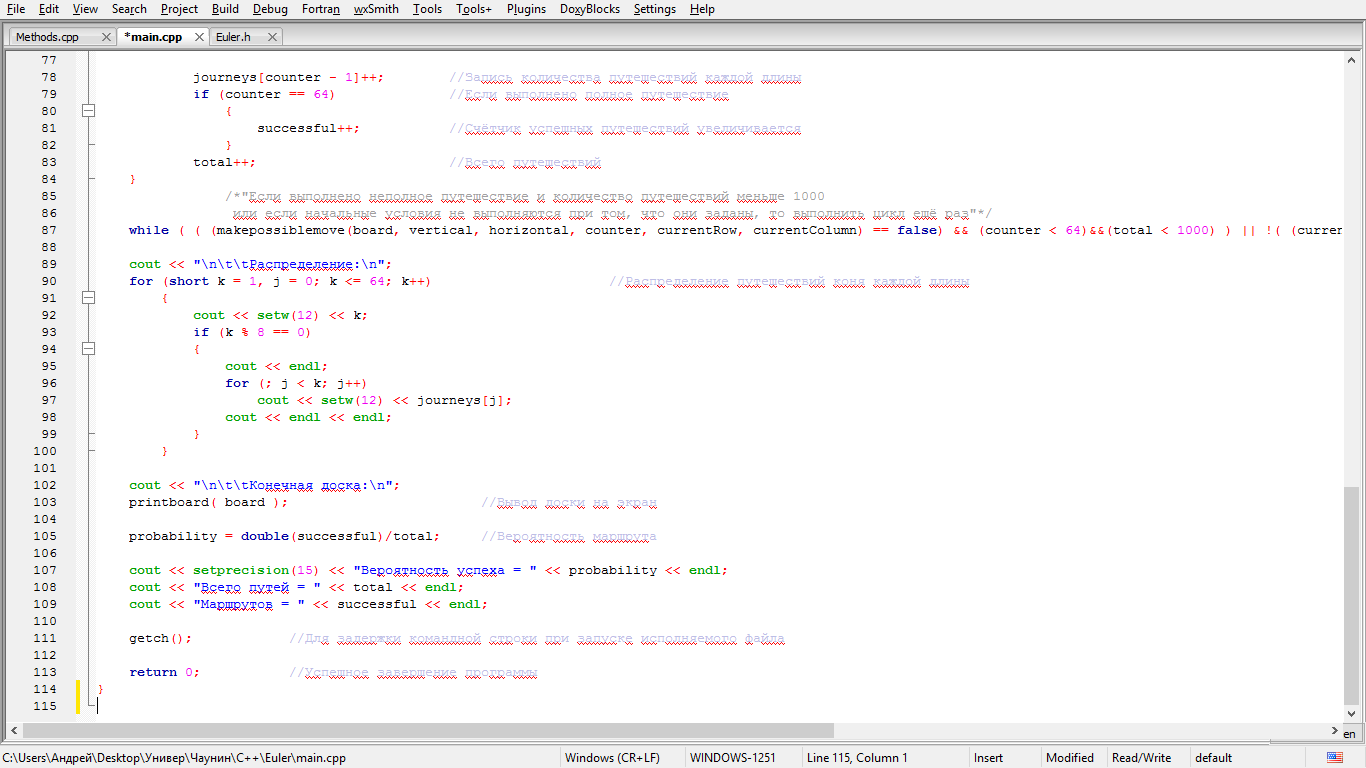




*Исполняемый файл Methods.cpp*







*Исполняемый файл main.cpp*

1. **Руководство пользователя.**

В данном пункте содержатся наиболее полные сведения о работе программы. Начнём с пояснения исходного кода программы. Реализация программы отделена от интерфейса. Есть три файла: заголовочный Euler.h, исходный main.cpp, исходный Methods.cpp. В заголовочном файле Euler.h содержатся прототипы всех функций исходного файла Methods.cpp, а файл main.cpp является реализацией всех функций файла Methods.cpp. Далее рассмотрим все функции по порядку вызова и их назначение.

* 1. ***Функции и их назначение***

1. Стандартная функция библиотеки <stdlib.h> **srand()** (строка 16 файла main.cpp). Она требуется для *рандомизации* при каждом вызове программы. Для её работы нужно передать ей целочисленный аргумент. Запись srand(time(NULL)) полностью рандомизирует квазислучайные числа при каждом запуске функции rand().

2. Функция **interact()** (строки 205-297 файла). Не возвращает значения и не принимает аргументов, но необходима для взаимодействия с пользователем. Первым делом она объявляет локальные переменные begin[3], end[3], reply (строки 207,208 файла); первые две относятся к типу char, а последняя к стандартному типу string. Далее, выполняется запрос о выборе начального положения коня на доске (строка 210 файла), и если пользователь выбирает не указывать начального поля или указывает его не правильно (строки 211-238 файла), то глобальные целочисленные переменные cR и cC, объявленные в этом же файле (строка 10 файла) и имеющие начальные значения cR=cC=9 либо не изменяют своих значений, либо устанавливаются равными 9 соответственно. Однако значение 9 для этих переменных означает рандомизацию выбора начального положения (строки 50-56 файла main.cpp). Иными словами, если пользователь не задаст правильное начальное положение, то начальное положение будет устанавливаться случайно для каждого путешествия. То же самое касается запроса о выборе конечного поля (строки 240-274 файла), с одной поправкой. Если пользователь не задаёт допустимое конечное поле, то конечное поле совсем не участвует в путешествии (строки 10, 256, 268, 57, 65, 66, 87 файла). Далее предлагается выбрать между жестким силовым методом и стратегическим (строка 277 файла), причём, выбрав силовой метод, значение глобальное целочисленной переменной method1 (строка 8 файла) устанавливается равным 0 (строки 278-281 файла). В случае выбора стратегического метода, значение этой переменной устанавливается равным 1 (строка 284 файла). Далее предлагается выбор между модифицированным правилом Варнсдорфа и обычным (строка 285 файла). Выбор первого устанавливает значение целочисленной глобальной переменной method2 равным 1 (строки 286-289 файла), иначе – 0 (строка 292 файла).

Функция interact() не связана с другими функциями.

3. Функция **setboard()** (строки 198-203 файла) получает аргумент в виде двумерного целочисленного массива [8x8] и не возвращает значения. Единственная цель данной функции – инициализировать элементы массива board[8x8] нулями. Эта функция вызывается 1 раз за каждое путешествие и поскольку её код занимает всего 3 строчки, она объявляется как inline, чтобы оптимизировать программу.

Функция setboard() не связана с другими функциями.

4. Функция **setaccessibility()** (строки 106-131 файла) не возвращает значения и принимает 4 аргумента: шахматную доску, доску доступности, текущую позицию строки и текущую позицию столбца. Первые два аргумента выражены целочисленными двумерными массивами (строки 20, 41-48 файла main.cpp), два последних – целочисленными типами. Данная функция объявляет два вектора целочисленного типа (строка 108 файла), вызывает функцию **makepossiblemove()** для получения числа возможных ходов из данной позиции шахматной доски (строка 109 файла). Далее цикл с предусловием for инициализирует векторы всеми уникальными возможными ходами (строки 112-124 файла). После этого все доступные поля на расстоянии одного хода от текущей позиции коня в доске доступности уменьшают свои значения на 1 (строки 126-129 файла). Иначе говоря, доступность любого поля уменьшается, когда конь ходит на поле, из которого можно пойти на первое. Чем больше полей занято, тем меньше доступность полей с ними связанных. Эта функция запускается не больше 63 раз за каждое путешествие и поскольку её код занимает всего 23 строчки, она объявляется как inline, чтобы оптимизировать программу. Функция setaccessibility() вызывает функцию makepossiblemove() и вызывается другими функциями, поэтому она связана с другими функциями.

5. Функция **makepossiblemove()** (строки 79-103 файла) возвращает значение и принимает 6 аргументов: шахматную доску, вектор вертикальных перемещений, вектор горизонтальных перемещений, счетчик ходов, текущую позицию строки и текущую позицию столбца. В теле функции объявляются два вектора, содержащие всевозможные ходы (строки 81-82 файла). Далее все нелегальные ходы, обнуляются, а целочисленная переменная possible\_moves уменьшает своё значение на 1 при каждом обнулении (строка 90 файла). Если возможных ходов ноль, то функция возвращает значение 0 в вызвавшую её функцию (строки 93,94 файла), иначе выполняется цикл while до тех пор, пока случайным образом не будет выбран (из оставшихся легальных ходов и нулей) легальный ход (строки 96-98 файла). Далее в векторы перемещений присваиваются соответствующие ходы (строки 99,100 файла) и функция возвращает число возможных ходов possible\_moves. Функция вызывается другими функциями файла и поэтому она связана с другими функциями.

6. Функция **setposition()** (строки 28-76 файла) не возвращает значения и принимает 7 аргументов: шахматную доску, доску доступности, вектор вертикальных перемещений, вектор горизонтальных перемещений, счётчик ходов, ссылку на текущую позицию строки и ссылку на текущую позицию столбца. В теле функции будет пропущен составной оператор (строки 31-70, 74,75 файла), если был выбран жёсткий силовой метод. Далее, объявленной целочисленной переменной min присваивается значение доступности текущего кандидата в ходы конём; так как другой ход может быть выбран впоследствии. Далее, объявляются переменные minRow и minColumn и инициализируются значениями текущего хода. Далее, целочисленной константе n присваивается количество доступных ходов, вызывая функцию **makepossiblemove()**. Эта функция также будет вызываться n раз в цикле (строки 35-46 файла) для получения отличных друг от друга ходов, коих ровно n штук. Далее, в цикле (строка 48 файла) будет выполнено n сравнений доступности хода с переменной min до тех пор, пока в последней не будет минимального числа доступности текущего хода (строки 50-55 файла). При выборе модифицированного правила Варнсдорфа, при равенстве доступностей двух различных ходов, будет выбран тот ход, который приводит к меньшей доступности в конечном итоге (строки 56-66 файла); это и является модификацией алгоритма Варнсдорфа. В таком случае вызывается функция **prediction()**. Однако, эта модификация на порядок замедляет программу. Далее, оптимальные ходы присваиваются обратно в позиции текущего счетчика векторам ходов (строки 68,69 файла) и этим заканчивается сложный оператор. Далее, к текущим позициям коня на доске прибавляются полученные ходы (строки 71,72 файла) и конь делает ход (строка 73 файла). Функция setposition() вызывает другие функции и поэтому она связана с другими функциями.

7. Функция **prediction()** (строки 133-196 файла) возвращает целочисленный аргумент и принимает 7 аргументов: шахматную доску, доску доступности, значение вертикального перемещения, значение горизонтального перемещения, счётчик ходов, текущую позицию строки, текущую позицию столбца. Далее моделируется следующий ход коня через полученные вертикальное и горизонтальное перемещения (строки 137-149); это делается для того, чтобы оценить доступность на следующем ходе (ибо функция вызывалась из-за равенства доступностей (строки 59-60 файла) текущего хода). Но может статься что сделанный ход привёл в тупик; для этого выполняется оператор (строки 153-156 файла), который возвращает число 8 – максимальное число доступности. В противном случае выполняется та же последовательность действий что и в функции **setboard()**, которая определяла минимальное число доступности. Но эта функция вызывает саму себя (строки 186-187 файла) и поэтому является рекурсивной. Рекурсия будет конечной. В конечном итоге возвращается минимальное значение доступности (строка 195 файла). Функция вызывает другие функции и вызывается ими и поэтому она связана с другими функциями.

8. Функция **printboard()** не возвращает значения и принимает один аргумент – шахматную доску. Эта функция имеет одну цель – выводить на экран доску размером [8x8] в форматированном виде. Она не связана с другими функциями.

* 1. ***Идентификаторы и их назначение***

Файла main.cpp:

NULL – именная константа, ноль.

register short counter - счетчик ходов конём.

register int board[8][8] – шахматная доска.

register int accessibility[8][8] – доска доступности (эвристики).

register int vertical[64] – вектор вертикальных перемещений.

register int horizontal[64] – вектор горизонтальных перемещений.

*(каждый ход конём сам по себе является вектором, состоящим из двух компонент: вертикальной и горизонтальной)*

register int currentRow – текущая позиция строки (горизонталь в шахм. терминологии)

register int currentColumn – текущая позиция столбца (вертикаль в шахм. терминологии)

register int finalRow – конечная позиция строки (после окончания путешествия)

register int finalColumn – конечная позиция столбца (после окончания путешествия)

register double probability – вероятность маршрута.

register long total – общее количество путей.

register long successful – общее количество маршрутов.

register int journeys[64] – вектор-счетчик путей (нужен для счета количества путей каждой длины).

extern short cR, cC, fR, fC – начальные условия из внешнего файла.

short k, j – переменные-счётчики циклов.

Можно заметить, что почти все переменные объявлены с регистровым доступом памяти. Это сделано из соображений оптимизации производительности. Памяти второго-третьего уровня вполне хватит для внесения этих переменных.

Файла Methods.cpp:

short method1 – выбор между силовым методом и стратегией.

short method2 – выбор между модифицированным методом Варнсдорфа и обычным.

short cR, cC, fR, fC – глобальные переменные для задания начальных условий на доске.

register short min, min2 – переменная для хранения наименьшего числа доступности.

register short minRow, minRow2, minColumn, minColumn2 – переменные для хранения строки и столбца с наименьшей доступностью.

register short const n – верхняя граница для цикла.

register int v[8], h[8] – векторы со всевозможными ходами.

register short possible\_moves – колличество возможных ходов.

register short rand – для случайного выбора хода.

register int horizontal[8], vertical[8] – для записи доступных ходов.

register int A[8][8] – дубликат доски доступности.

register int B[8][8] – дубликат доски.

char begin[3], end[3] – для записи полей.

string reply – ответ пользователя.

1. **Заключение**

Целью данной задачи было выяснить является ли ЭВМ достаточным средством для вычисления сложных математических задач. В данной программе выполняете большинство из изученных разделов языка С. Наиболее сложной частью задачи было написание рекурсивной функции, а также вычисления средних величин и оптимизация кода. Были получены более глубокие знания в работе с языками высокого уровня.

* [Файл Euler.h](Euler/Euler.h)
* [Файл Methods.cpp](Euler/Methods.cpp)
* [Файл main.cpp](Euler/main.cpp)
* [Исполняемый файл.](Euler/bin/Debug/Euler.exe)

Используемая литература:

Харви Дейтел, Пол Дейтел – «Как программировать на С++».