**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №5**

**по дисциплине «Программирование»**

Тема: Рекуррентные соотношения в программировании на примере задачи с кузнечиком

|  |  |  |
| --- | --- | --- |
| Студент гр. 9893 |  | Росихин А.Ю. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2019

**Цель работы.**

Изучить рекуррентные соотношения. Научиться создавать и применять рекурсивные функции для достижения поставленных целей. Научиться вычислять количество путей за определенное количество ходов. Научиться подсчитывать вероятность удачных путей.

**Основные теоретические положения.**

Рекуррентная формула — формула вида

, (1)

выражающая каждый член последовательности *an* через *p* предыдущих членов и возможно номер члена последовательности *n*. Общая проблематика вычислений с использованием рекуррентных формул является предметом теории рекурсивных функций. Рекуррентным уравнением называется уравнение, связывающее несколько подряд идущих членов некоторой числовой последовательности. Последовательность, удовлетворяющая такому уравнению, называется рекуррентной последовательностью [1].

Рекурсия — это процесс, в котором функция вызывает себя прямо или косвенно, называется рекурсией. Эта функция называется рекурсивной функцией. Используя рекурсивный алгоритм, некоторые задачи могут быть решены довольно легко [2].

**Постановка задачи.**

Написать программу, которая решает задачу про кузнечика, который перемещается по вершинам многоугольника на одну вершину (шаг) за один ход. Задания следующие:

1. Определить количество путей за *N* прыжков из вершины *Х* в вершину *Y*. Вывести на экран все варианты и выделить подходящие;
2. Определить количество путей за *N* прыжков из вершиы *Х* в вершину *Y*, при условии, что в вершине *Z* находится препятствие — камень, который не позволяет перемещаться в эту вершину;
3. В вершине *Z* находится бомба, то есть при прохождении этой вершины кузнечик взрывается. Найти вероятность гибели кузнечика за *N* прыжков из вершиы *Х* в вершину *Y.*

**Выполнение работы.**

Разобьем нашу программу на несколько функцию и блоков, для облегчения понимания кода программы:

1. int CircleWay(int start, int way) — функция, которая возвращает номер вершины после перемещения. Позволяет не выйти за рамки допустимых минимальных и максимальных значений.
2. void go(int start, int now, int all) — рекурсивная функция. Заполняет массив номерами вершин всех возможных путей. Если оставшихся ходов больше одного, то эта функция вызывает сама себя.
3. void grasshopper(int n) — основная функция этой программы. Запускает рекурсивную функцию go(), форматирует данные перед выводом на экран, выводит результат.
4. int main() — внутри этой функции реализована проверка вводимых данных на корректность и реализован бесконечный цикл ввода данных.

Исходный код программы представлен в приложении А.

**Тестирование программы.**

Результаты тестирования программы представлены в виде скриншотов экрана на рис. 1-4.

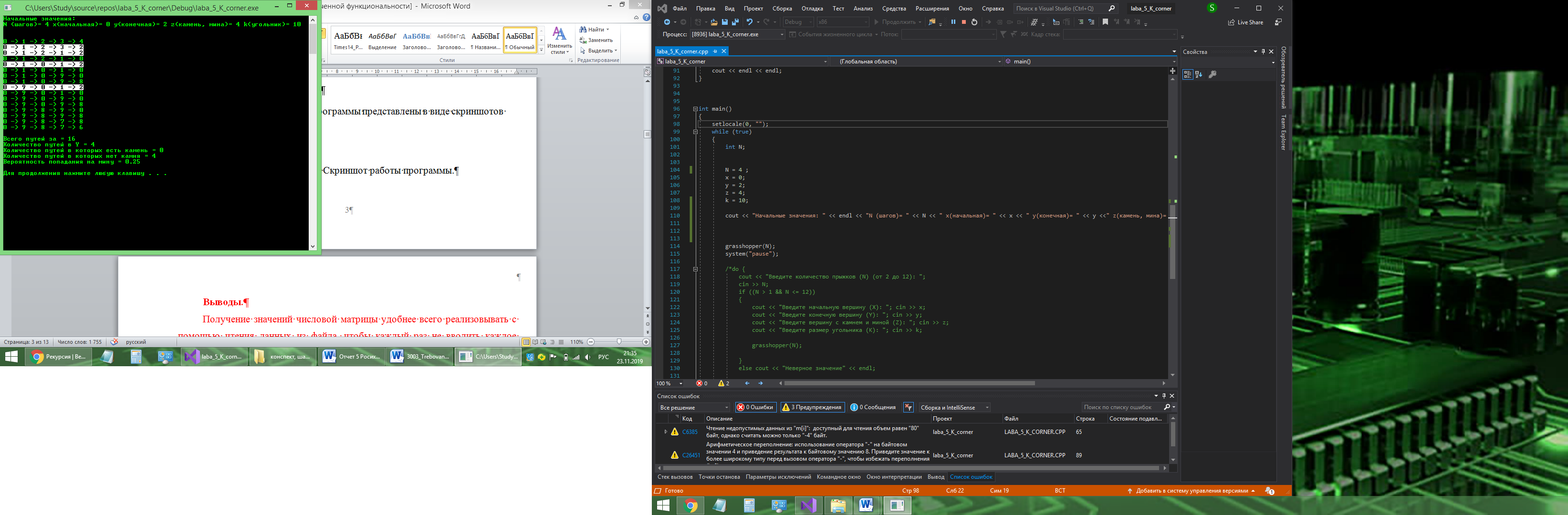
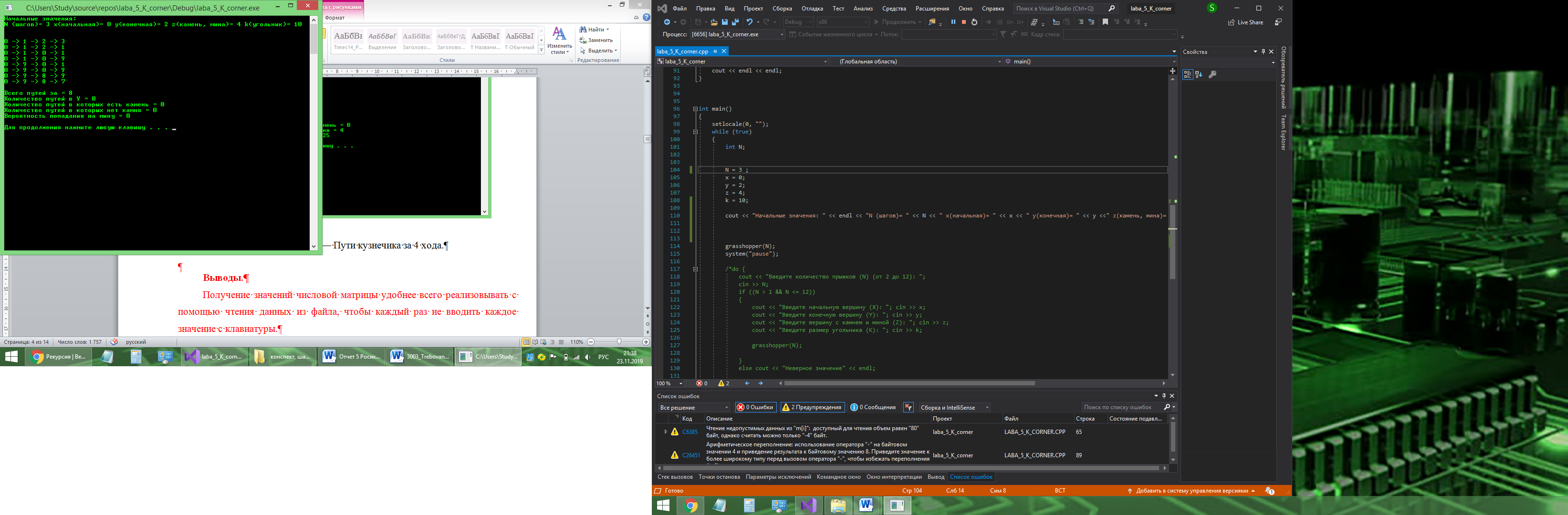
Рисунок 1 — Пути кузнечика за 4 хода в 10 угольнике

Рисунок 2 — Пути кузнечика за 3 хода в 10 угольнике

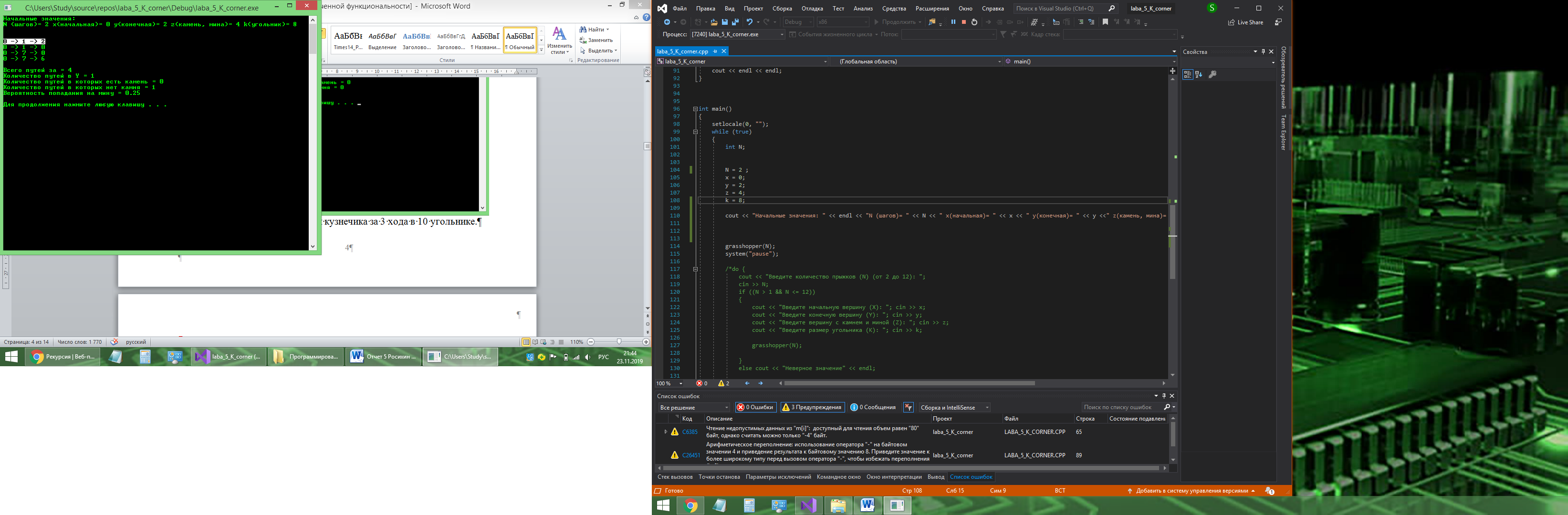
Рисунок 3 — Пути кузнечика за 2 хода в 8 угольнике

Рисунок 4 представлен в приложении Б.

**Выводы.**

Если количество шагов выражается нечетным числом, то решений, удовлетворяющих заданию нет.

В этой практической работе я научился создавать и успешно применять рекурсивные функции. Спектр применения рекурсивных функций в программировании очень широк. Поставленные задачи можно решать как рекурсивно, так и итеративно. Некоторые задачи являются неотъемлемо рекурсивными. Для них предпочтительнее использовать рекурсивный код.

При применении рекурсивной функции надо обращать самое пристальное внимание к возможности переполнения стека и возникновения ошибки. Также необходимо следить за тем, чтобы память не закончилась.

Рекурсия обеспечивает простой и понятный способ написания кода. Всё решение задачи сводится к решению основного или, как ещё его называют, базового случая. В случае, когда рекурсивная функция вызывается для решения сложной задачи, выполняется некоторое количество рекурсивных вызовов или как их еще называют шагов, с целью сведения сложной задачи к более простой. Это значительно упрощает как понимание, так и решение задачи.

Приложение А

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

#include <iostream>

#include <string>

#include <Windows.h>

#include <math.h>

using namespace std;

int countway = 0;

int m[1048576][20];

int x = 0, y = 0, z = 0, k = 0;

int CircleWay(int start, int way)

{

int a = start + way;

int res = k-1;

if (a > res) return 0;

if (a < 0) return res;

}

void go(int start, int now, int all)

{

if (now == 1)

{

m[countway][all - now] = CircleWay(start, +1);

for (int i = 0; i < all - now; i++) m[countway + 1][i] = m[countway][i];

countway++;

m[countway][all - now] = CircleWay(start, -1);

for (int i = 0; i < all - now; i++) m[countway + 1][i] = m[countway][i];

countway++;

}

else

{

m[countway][all - now] = CircleWay(start, +1);

go(CircleWay(start, +1), now - 1, all);

m[countway][all - now] = CircleWay(start, -1);

go(CircleWay(start, -1), now - 1, all);

}

}

void grasshopper(int n) // n max 18

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), (WORD)((0 << 4) | 10));

int goodway = 0;

int stone\_bomb = 0;

go(x, n, n);

cout << endl << endl;

for (int i = 0; i < pow(2, n); i++)

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), (WORD)((0 << 4) | 10));

if (m[i][n - 1] == y)

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), (WORD)((15 << 4) | 0));

goodway++;

}

for (int ii = 0; ii < n \* n; ii++) if (m[i][ii] == z && (m[i][n - 1] == y)) //|| m[i][n - 1] == z))

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), (WORD)((6 << 4) | 0));

stone\_bomb++;

break;

}

cout << x;

for (int ii = 0; ii < n; ii++) cout << " -> " + to\_string(m[i][ii]);

cout << endl;

}

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), (WORD)((0 << 4) | 10));

cout << endl;

cout << "Всего путей за = " << pow(2, n) << endl;

cout << "Количество путей в Y = " << goodway << endl;

cout << "Количество путей в которых есть камень = " << stone\_bomb << endl;

cout << "Количество путей в которых нет камня = " << goodway - stone\_bomb << endl;

cout << "Вероятность попадания на мину = " << (goodway - stone\_bomb) / pow(2, n);

cout << endl << endl;

}

int main()

{

setlocale(0, "");

while (true)

{

int N;

N = 6 ;

x = 0;

y = 2;

z = 4;

k = 12;

cout << "Начальные значения: " << endl << "N (шагов)= " << N << " x(начальная)= " << x << " y(конечная)= " << y <<" z(камень, мина)= " << z << " k(угольник)= " << k << endl;

grasshopper(N);

system("pause");

/\*do {

cout << "Введите количество прыжков (N) (от 2 до 12): ";

cin >> N;

if ((N > 1 && N <= 12))

{

cout << "Введите начальную вершину (X): "; cin >> x;

cout << "Введите конечную вершину (Y): "; cin >> y;

cout << "Введите вершину c камнем и миной (Z): "; cin >> z;

cout << "Введите размер угольника (K): "; cin >> k;

grasshopper(N);

}

else cout << "Неверное значение" << endl;

cin.clear(); // очистка потока, чтобы не зациклилось

cin.ignore(cin.rdbuf()->in\_avail()); // очистка буфера длины in\_avail

} while (!(N > 1 && N <= 12)); // То же самое условие

\*/

}

}

Приложение Б

**РИСУНОК 4**

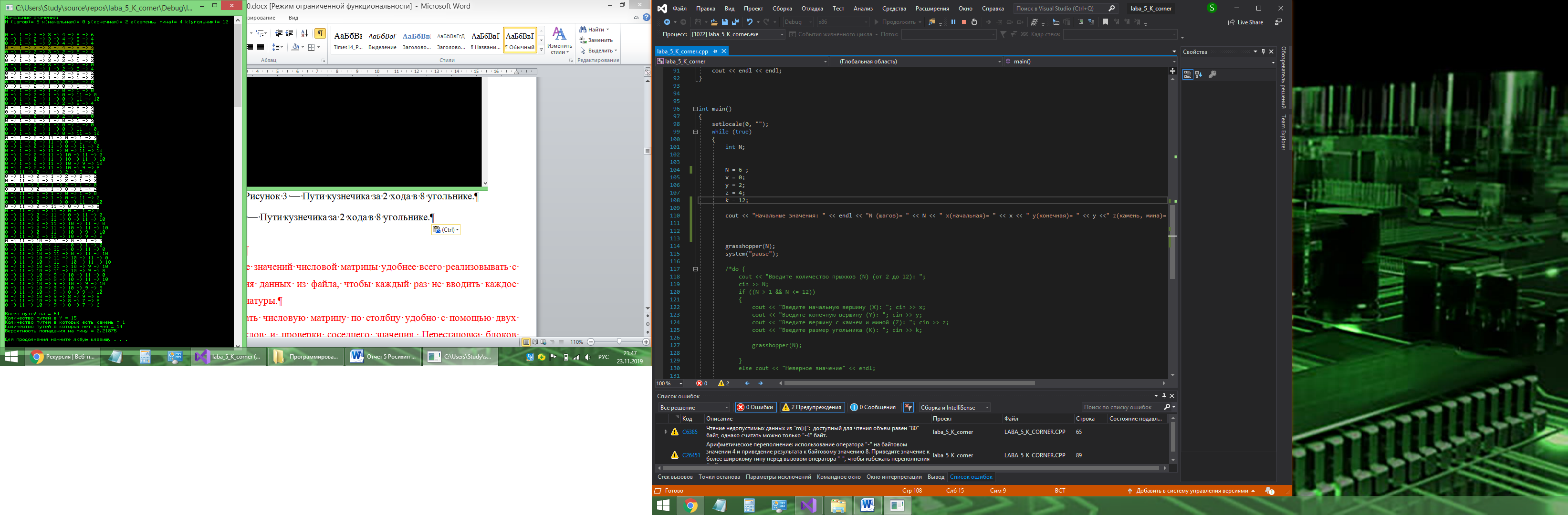


Рисунок 4 — Пути кузнечика за 6 хода в 12 угольнике

**Список использованных источников.**

1. Интернет-энциклопедия // Сайт Википедия. URL: https://ru.wikipedia.org/wiki/Рекуррентная\_формула.html (дата обращения: 06.12.2019).
2. Полезные статьи по программированию // Интернет технологии. ру. URL: https://www.internet-technologies.ru/articles/rekursiya.html (дата обращения: 06.12.2019).