

**Оглавление**

[Введение 3](#_Toc186299935)

[Постановка задачи 1 4](#_Toc186299936)

[Ход решения задачи 1 6](#_Toc186299937)

[Постановка задачи 2 10](#_Toc186299938)

[Ход решения задачи 2 13](#_Toc186299939)

[Постановка задачи 3 19](#_Toc186299940)

[Ход решения задачи 3 21](#_Toc186299941)

[Постановка задачи 4 25](#_Toc186299942)

[Ход решения задачи 4 26](#_Toc186299943)

[Заключение 29](#_Toc186299944)

[Список используемой литературы 30](#_Toc186299945)

# **Введение**

В данной расчетно-графической работе используется язык программирования C#.

Язык программирования C# представляет собой современный, объектно-ориентированный язык, разработанный компанией Microsoft в рамках платформы .NET. Он был создан для упрощения разработки приложений, обеспечивая высокую производительность и надежность. C# сочетает в себе элементы языков C++ и Java, что делает его доступным для программистов с различным опытом.

C# широко используется для создания приложений различного типа, включая настольные приложения, веб-приложения и мобильные приложения. Благодаря поддержке таких технологий, как ASP.NET, WPF и Xamarin, разработчики могут создавать мощные и интерактивные решения, способные работать на различных платформах.

Одной из ключевых особенностей C# является его богатая стандартная библиотека, которая предоставляет множество готовых классов и методов для решения типичных задач, что значительно ускоряет процесс разработки. Кроме того, C# поддерживает такие концепции, как сборка мусора, управление памятью и обработка исключений, что позволяет создавать более устойчивые и безопасные приложения.

# **Постановка задачи 1**

Название задачи: «Крестьянин и черт».

Условие задачи: Идет крестьянин и плачется: "Эхма! Жизнь моя горькая! Заела нужда совсем! Вот в кармане только несколько монет, да и те сейчас нужно отдать. И как это у других бывает, что на всякие свои деньги они еще деньги получают? Хоть бы кто помочь мне захотел".

Только успел это сказать, как глядь, а перед ним черт стоит и говорит: "Вот видишь этот мост через реку. Стоит тебе перейти через мост, и у тебя будет вдвое больше денег, чем есть. Перейдешь опять, и снова станет вдвое больше. Но за то, что я у тебя деньги удваиваю, после каждого перехода ты мне должен отдавать по **K** монет".

"Ой ли," – сказал крестьянин –"ну-ка, попробуем". Перешел мост, и деньги у него удвоились. Отдал он черту **K** монет, перешел мост еще раз, и опять деньги удвоились. Снова отдал крестьянин черту **K** монет.

Однако после **Z** переходов и отдач черту по **K** монет оказалось, что у крестьянина не осталось ни одной монеты.

Требуется определить, сколько комбинаций условий перехода через мост может быть, если известно, что у крестьянина изначально было не более **MaxN** монет. Комбинацией условий перехода является тройка чисел **N**, **K**, **Z**, где **N** – начальное количество монет у крестьянина, **K** – количество монет, отдаваемых черту после каждого перехода, **Z** – количество переходов. Естественно, что для этой тройки должно выполняться условие, что после **Z** циклов у крестьянина не должно остаться монет.

**Входной файл** содержит целое число **MaxN** – максимальное количество, которое может быть изначально у крестьянина (1 ≤ **MaxN** ≤ 2000000000).

**Выходной файл** должен содержать одно целое число – количество комбинаций условий перехода через мост.

***Пример:***

Input.txt: 3

Output.txt: 4

# **Ход решения задачи 1**

Данный алгоритм решает задачу, связанную с определением количества комбинаций условий перехода через мост, которые могут привести к тому, что у крестьянина не останется монет после определенного числа переходов. Код написан на языке C# и выполняет следующие шаги:

1. Ввод данных:
   * Программа считывает целое число MaxN, которое представляет максимальное количество монет, которые могут быть у крестьянина изначально. Это значение вводится пользователем через стандартный ввод.
2. Инициализация счетчика:
   * Переменная countCombinations инициализируется нулем. Она будет использоваться для хранения общего количества допустимых комбинаций (тройка чисел *N*, *K*,  *Z*).
3. Цикл по значениям Z:
   * Цикл начинается с Z = 1 и продолжается бесконечно (for (int Z = 1; ; Z++)).
   * В каждой итерации цикла вычисляется значение d, равное 2Z−12*Z*−1. Это достигается с помощью побитового сдвига: 1L << Z, что эквивалентно возведению 2 в степень *Z* и вычитанию 1.
   * Проверка условия завершения: Если d становится больше MaxN, цикл прерывается с помощью оператора break. Это условие гарантирует, что мы не будем рассматривать значения, превышающие максимальное количество монет, которое может быть у крестьянина.
4. Подсчет комбинаций:
   * Для каждого значения d, которое меньше или равно MaxN, вычисляется количество возможных начальных монет *N*. Это количество определяется как целочисленное деление MaxN / d.
   * Это деление представляет собой количество возможных начальных значений *N*, которые удовлетворяют условию, что после *Z* переходов и уплаты *K* монет у крестьянина не останется денег.
   * Результат добавляется к переменной countCombinations.
5. Вывод результата:
   * После завершения цикла программа выводит общее количество комбинаций, которое было подсчитано, на стандартный вывод

Ход программы представлен на рисунке 1:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1 – Ход программы 1

Примеры работы на тестовых данных показан на рисунке 2.

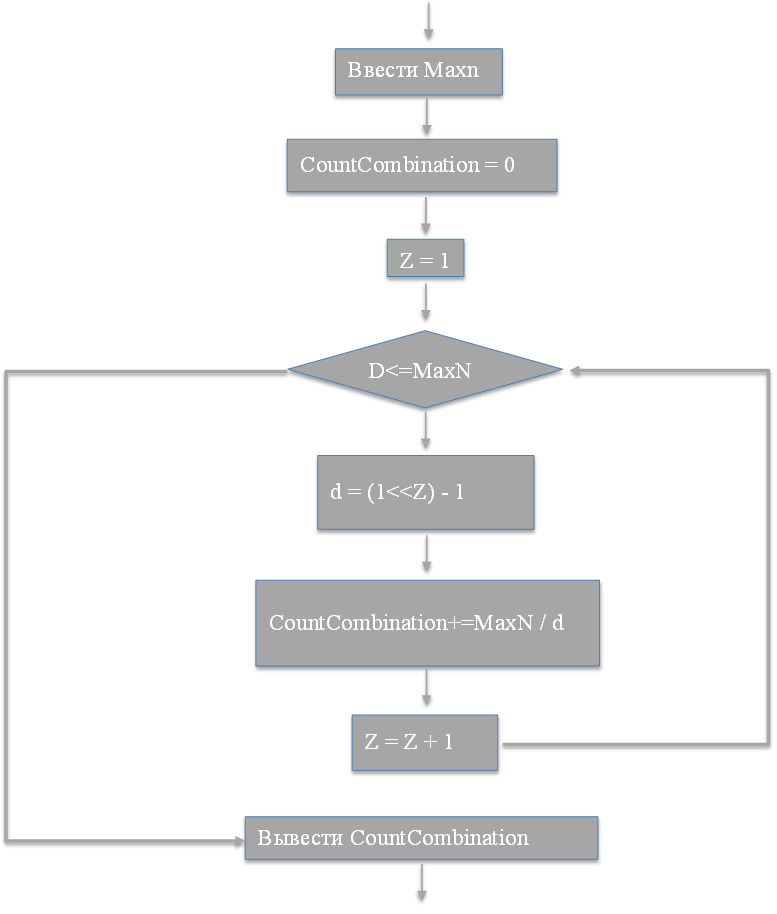


Рисунок 2 – Блок-cхема задачи 1

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Тесты задачи 1

# **Постановка задачи 2**

Название задачи: «Зельеварение».

Условие задачи: Одним из нелюбимых предметов Невилла Долгопупса, товарища Гарри Поттера с факультета Гриффиндор школы чародейства и волшебства “Хогвартс”, было зельеварение.

Чтобы помочь Невиллу в совершенствовании навыков зельеварения, Гарри придумал зельеварочный комбайн, изготавливающий зелья с помощью определенных заклинаний. Каждое заклинание представляет собой одно слово, формируемое в зависимости от последовательности и методов приготовления зелья.

Последовательность приготовления зелья описывается в виде набора действий, каждое из которых указывает на метод обработки заданного списка ингредиентов:

* *смешивание* описывается **MIX < ингредиент1 ингредиент2, …>**;
* *растворение в воде* описывается **WATER < ингредиент1 ингредиент2, …>**;
* *измельчение* описывается **DUST < ингредиент1 ингредиент2, …>**;
* *обжиг* описывается **FIRE < ингредиент1 ингредиент2, …>**,

при этом в качестве любого ингредиента может выступать либо некоторое вещество, задаваемое строковой константой, либо результат выполнения любого из предыдущих действий, задаваемый с помощью номера действия. Название действия и названия ингредиентов разделяются пробелами. В действии участвует как минимум один ингредиент.

Каждое действие переводится в слово по следующему правилу:

* *смешивание* задается в формируемом заклинании словом **MX<список ингредиентов>XM**;
* *растворение в воде* – слово **WT<список ингредиентов>TW**;
* *измельчение* – слово **DT<список ингредиентов>TD**;
* *обжиг* – слово **FR<список ингредиентов>RF**,

где **<список ингредиентов** – единое слово, сформированное путем сложения названий ингредиентов или слов, описывающих предыдущие действия.

Последнее действие явно или неявно использует результаты выполнения всех предыдущих действий и является основой для заклинания.

Помогите Невиллу по заданной последовательности действий сформировать заклинание.

**Примечание:**

* при формировании заклинания **учитывается** регистр названий действий и ингредиентов;
* порядок названий ингредиентов в заклинании должен соответствовать их порядку в действии, т.е. для действия “DUST root tooth” в заклинании правильным считается слово “DTroottoothTD”, а слово “DTtoothrootTD” считается неправильным;
* в названии веществ нет цифр, а используются только английские буквы;
* каждое действие может быть несколько раз использовано в последующих действиях;
* гарантируется, что длина строки, содержащей сформированное заклинание, не превышает 50000 символов.

**Входной файл** содержит набор строк, каждая из которых описывает отдельное действие. Строки расположены в порядке выполнения действий. Длина каждой строки не превышает 255 символов. Количество строк не превышает 100.

**Выходной файл** должен содержать строку, содержащую сформированное заклинание.

***Пример:***

Input.txt

DUST root tooth

WATER 1 tear

Output.txt

WTDTroottoothTDtearTW

# **Ход решения задачи 2**

Вот пошаговое описание алгоритма:

1. **Чтение входных данных**:
   * Программа считывает строки из файла input1.txt, каждая из которых описывает одно действие (например, смешивание, растворение и т.д.).
   * Данные считываются в массив строк actions.
2. **Инициализация структур данных**:
   * Создается двумерный массив ArrayOfAction, который будет хранить разбитые на части действия.
   * Создается одномерный массив ShortAction, который будет хранить результат каждого действия в виде заклинания.
3. **Обработка действий**:
   * Для каждого действия в массиве actions программа разбивает строку на части с помощью метода Split(), сохраняя результат в ArrayOfAction.
   * Затем происходит обработка каждого действия в зависимости от его типа (DUST, WATER, MIX, FIRE):

* **Определение префикса и суффикса**: Для каждого типа действия определяется соответствующий префикс и суффикс (например, для DUST это "DT" и "TD").
* **Обработка ингредиентов**: В цикле по ингредиентам (начиная с индекса 1, так как индекс 0 – это название действия):
* Если ингредиент является числом (номер предыдущего действия), программа извлекает соответствующий результат из массива ShortAction.
* Если ингредиент – строка (название вещества), он добавляется в текущее действие.
* **Формирование слова заклинания**: После обработки всех ингредиентов к строке добавляются суффикс и сохраняется в ShortAction.

1. **Сравнение с выходными данными**:
   * Программа считывает содержимое файла output{p + 1}.txt, чтобы сравнить его с полученным заклинанием, которое хранится в ShortAction по индексу последнего действия.
   * Если заклинание, сформированное программой, совпадает с ожидаемым выводом, выводится сообщение о совпадении.
2. **Цикл обработки нескольких файлов**:
   * Внешний цикл for (int p = 0; p < 10; p++) позволяет обрабатывать до 10 различных файлов, что делает программу гибкой для тестирования с разными входными данными.

Ход программы представлен ниже:

using System;

using System.Diagnostics;

*class* Potion

{

    static void Main(string[] *args*)

    {

        string[] actions = File.ReadAllLines("input1.txt");

        string[][] ArrayOfAction = new string[actions.Length][];

        string[] ShortAction = new string[actions.Length];

        string action;

        int n;

        string output;

        for (int p = 0; p < 10; p++)

        {

            actions = File.ReadAllLines($"input{p + 1}.txt");

            ArrayOfAction = new string[actions.Length][];

            ShortAction = new string[actions.Length];

            for (int i = 0; i < actions.Length; i++)

            {

                action = actions[i];

                ArrayOfAction[i] = action.Split(' ');

            }

            for (int i = 0; i < ArrayOfAction.Length; i++)

            {

                switch (ArrayOfAction[i][0])

                {

                    case "DUST":

                        action = "DT";

                        for (int k = 1; k < ArrayOfAction[i].Length; k++)

                        {

                            if (int.TryParse(ArrayOfAction[i][k], out int res))

                            {

                                n = Convert.ToInt32(ArrayOfAction[i][k]);

                                action += ShortAction[+(n - 1)];

                            }

                            else

                            {

                                action += ArrayOfAction[i][k];

                            }

                        }

                        action += "TD";

                        ShortAction[i] = action;

                        break;

                    case "WATER":

                        action = "WT";

                        for (int k = 1; k < ArrayOfAction[i].Length; k++)

                        {

                            if (int.TryParse(ArrayOfAction[i][k], out int res))

                            {

                                n = Convert.ToInt32(ArrayOfAction[i][k]);

                                action += ShortAction[+(n - 1)];

                            }

                            else

                            {

                                action += ArrayOfAction[i][k];

                            }

                        }

                        action += "TW";

                        ShortAction[i] = action;

                        break;

                    case "MIX":

                        action = "MX";

                        for (int k = 1; k < ArrayOfAction[i].Length; k++)

                        {

                            if (int.TryParse(ArrayOfAction[i][k], out int res))

                            {

                                n = Convert.ToInt32(ArrayOfAction[i][k]);

                                action += ShortAction[+(n - 1)];

                            }

                            else

                            {

                                action += ArrayOfAction[i][k];

                            }

                        }

                        action += "XM";

                        ShortAction[i] = action;

                        break;

                    case "FIRE":

                        action = "FR";

                        for (int k = 1; k < ArrayOfAction[i].Length; k++)

                        {

                            if (int.TryParse(ArrayOfAction[i][k], out int res))

                            {

                                n = Convert.ToInt32(ArrayOfAction[i][k]);

                                action += ShortAction[+(n - 1)];

                            }

                            else

                            {

                                action += ArrayOfAction[i][k];

                            }

                        }

                        action += "RF";

                        ShortAction[i] = action;

                        break;

                }

            }

            output = File.ReadAllText($"output{p + 1}.txt");

            if (output == ShortAction[ShortAction.Length - 1])

            {

                Console.WriteLine($"ввод и вывод файла {p + 1} совпадают");

            }

        }

    }

}

Примеры работы на тестовых данных показан на рисунке 4.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 4 – Тесты задачи 2

# **Постановка задачи 3**

Название задачи: «Отгадай число».

Условие задачи: Известен следующий фокус. Фокусник предлагает выполнить действия следующего характера: задумайте число, прибавьте 2, умножьте на 3, отнимите 5, отнимите задуманное число и т.д. После этого по названному полученному результату фокусник определяет задуманное число.

Пусть задумано некоторое целое число **X**. Требуется после выполнения ряда действий по известному результату **R** определить это число.

Примечание:

* + - * + гарантируется, что имеется только один ответ;
        + гарантируется, что во время выполнения действий какие-либо промежуточные результаты не превышают по модулю 2 000 000 000.

Входной файл:

Первая строка содержит количество действий **N** (0  **N**  100).

Следующие **N** строк содержат описания действий в последовательности их выполнения, причем в каждой строке указывается одно действие в формате **S V**, где:

* + - * + **S** – тип действия, состоящий из одного символа: "\*" – умножить; "-" – отнять; "+" – прибавить;
        + **V** – аргумент действия. Может быть целым числом (|**V**| ≤ 100) либо символом "**x**". Символ "**x**" может применяться только в действиях "-" и "+" и обозначает, что нужно отнять или прибавить задуманное число, соответственно.

Последняя строка содержит результат **R** (|**R**| ≤ 2 000 000 000).

**Выходной файл** должен содержать одно целое число – задуманное число **X**.

***Пример:***

Input.txt

4

+ 2

\* 3

- 5

- x

7

Output.txt

3

# **Ход решения задачи 3**

1. **Чтение входных данных**:
   * Программа считывает данные из входного файла, где первая строка содержит количество действий N*N* (от 0 до 100).
   * Следующие *N* строк описывают действия в формате S V, где S — тип действия (умножение, вычитание, сложение), а V — аргумент действия. Аргумент может быть целым числом или символом x, который обозначает задуманное число.
   * Последняя строка содержит результат R*R*.
2. **Инициализация переменных**:
   * Переменная x инициализируется значением 1. Она будет использоваться для отслеживания влияния задуманного числа на итоговый результат.
   * Переменная a инициализируется значением 0. Она будет использоваться для накопления изменений, которые не зависят от задуманного числа.
   * Переменная r инициализируется значением, считанным из входных данных, и представляет собой конечный результат после выполнения всех действий.
3. **Обработка действий**:
   * Программа проходит через все действия и в зависимости от типа действия (+, -, \*) выполняет соответствующие операции:
     + - **Сложение (+)**:
         * Если аргумент – это x, то увеличиваем x на 1, что отражает добавление задуманного числа.
         * Если аргумент – это число, то добавляем его к a.
       - **Вычитание (-)**:
         * Если аргумент – это x, то уменьшаем x на 1.
         * Если аргумент – это число, то вычитаем его из a.
       - **Умножение (\*)**:
         * Умножаем a на значение аргумента.

* Умножаем x на значение аргумента.

1. **Вычисление задуманного числа**:

После обработки всех действий, задуманное число *X* вычисляется по формуле:

​

* + Это выражение позволяет получить значение *X* на основе конечного результата *R* и накопленных изменений *a* и *x*.

1. **Вывод результата**:
   * Программа выводит значение задуманного числа *X* для каждого набора действий из входных данных.

Ход программы представлен ниже:

using System;

*class* Program

{

    static void Main()

    {

        string[] input;

        string action;

        string argument;

        int x = 1;

        int a = 0;

        int r;

        for (int i = 0; i < 12; i++)

        {

            if(i<9){

                input = File.ReadAllLines($"input\_s1\_0{i+1}.txt");

            }else{

                input = File.ReadAllLines($"input\_s1\_{i+1}.txt");

            }

            int CountACtion = int.Parse(input[0]);

            x = 1 ;

            a = 0;

            r =int.Parse(input[CountACtion+1]);

            for (int j = 1; j <=CountACtion; j++)

            {

                // Console.WriteLine(input[j]+"   "+ CountACtion);

                action = input[j].Split(" ")[0];

                argument = input[j].Split(" ")[1];

                switch (action)

                {

                    case "+":

                        if (argument == "x")

                        {

                            x += 1;

                        }

                        else

                        {

                            a += int.Parse($"{argument}");

                        }

                        break;

                    case "-":

                        if (argument == "x")

                        {

                            x -= 1;

                        }

                        else

                        {

                            a -= int.Parse($"{argument}");

                        }

                        break;

                    case "\*":

                        a \*= +int.Parse($"{argument}");

                        x \*= +int.Parse($"{argument}");

                        break;

                }

            }

                Console.WriteLine($"В файле {i+1} Изначальный x = "+ (r-a)/x);

        }

    }

}

Примеры работы на тестовых данных показан на рисунке 5.

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 5 – Тесты задачи 3

# **Постановка задачи 4**

Название задачи: «Отбор в разведку».

Условие задачи: Из **N** солдат, выстроенных в шеренгу, требуется отобрать троих в разведку. Для того чтобы сделать это, выполняется следующая операция: если солдат в шеренге больше 3, то шеренга разбивается на две, одна из которых состоит из солдат, стоящие на четных позициях, а вторая – стоящих на нечетных позициях. Эта процедура повторяется для всех полученных шеренг до тех пор, пока в каждой из них не останется 3 или менее солдат. Если солдат осталось трое, то данную группу можно послать в разведку.

Требуется определить, сколько групп по 3 человека может быть сформировано из исходной шеренги.

**Входной файл** содержит число **N –** количество солдат в исходной шеренге. (0 < **N** ≤ 10000000).

**Выходной файл** должен содержать количество вариантов формирования групп разведки.

***Пример 1:***

Input.txt

10

Output.txt

2

***Пример 2:***

Input.txt

4

Output.txt

0

# **Ход решения задачи 4**

Описание Алгоритма:

1. **Чтение входных данных**:
   * Программа считывает количество солдат *N* из входного файла.
2. **Определение количества групп**:
   * Если количество солдат меньше 3, вернуть 0, так как нельзя сформировать группу.
   * Если количество солдат равно 3, вернуть 1, так как можно сформировать одну группу.
   * Если количество солдат больше 3, рекурсивно разбить шеренгу на две подгруппы:
     1. Подгруппа с четными позициями: *N*/2
     2. Подгруппа с нечетными позициями: *N*−*N*/2
   * Суммировать количество групп, которые можно сформировать из обеих подгрупп.
3. **Вывод результата**:
   * Вывести общее количество групп, которые могут быть сформированы.

Ход программы представлен ниже:

using System;

using System.IO;

*class* Program

{

    static void Main()

    {

        if(i<9){

                N = int.Parse(File.ReadAllText("input\_s1\_01.txt"));

            }else{

                N = int.Parse(File.ReadAllText("input\_s1\_01.txt"));

            }

        int result = CountGroups(N);

        Console.WriteLine(result);

    }

    static int CountGroups(int *N*)

    {

        if (*N* < 3)

        {

            return 0;

        }

        if (*N* == 3)

        {

            return 1;

        }

        int evenCount = CountGroups(*N* / 2);

        int oddCount = CountGroups(*N* - *N* / 2);

        return evenCount + oddCount;

    }

}

Примеры работы на тестовых данных показан на рисунке 6.

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

Рисунок 6 – Тесты задачи 3

# **Заключение**

В ходе выполнения задач были рассмотрены и изучены различные алгоритмические подходы к обработке данных и вычислению значений, а также использованы методы для работы с файлами и обработки строковых данных в языке программирования C#. В частности, в первой задаче реализован алгоритм, который считывает данные из файлов, выполняет арифметические операции и выводит результаты, основанные на определенных действиях. Рассмотренные формулы и методы вычисления позволяют эффективно управлять переменными и производить необходимые расчеты, что является ключевым аспектом программирования.

Во второй задаче акцент сделан на подсчете количества групп солдат, где для решения использованы формулы, позволяющие вычислять количество групп на основе заданного количества солдат  *N*. Учитывая, что для формирования групп необходимо минимум три солдата, алгоритм обеспечивает корректную обработку различных значений  *N* и использует массив для хранения промежуточных результатов, что значительно повышает производительность и снижает время вычислений.

Третья задача сосредоточена на подсчете комбинаций, используя битовые операции для оптимизации вычислений. Этот подход демонстрирует, как можно эффективно работать с большими числами и использовать математические свойства для достижения высокой производительности.

Таким образом, выполненные задачи продемонстрировали важность алгоритмического мышления и применение математических формул для решения практических задач программирования. Использование методов для обработки данных, такие как чтение из файлов и работа с массивами, значительно повышает гибкость и эффективность разработанных решений. Данные подходы могут быть адаптированы и расширены для решения более сложных задач, что открывает новые возможности для дальнейшего изучения и применения алгоритмических методов в программировании.

# **Список используемой литературы**

1. Документация по языку C# – <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 12.12.2024)
2. Васильев А. Н. Программирование на C# для начинающих. Основные сведения. – Москва: Эксмо, 2018. – 592 с. (дата обращения: 12.12.2024)
3. Мюллер, Джон Пол, Семnф, Билл, Сфер, Чак. С# для чайников. – Пер. с англ. – СПб.: ООО "Диалектика", 2019. – 608 с. (дата обращения: 12.12.2024)
4. METANIT – <https://metanit.com/sharp/tutorial/> (дата обращения: 12.12.2024)
5. Skillbox – <https://skillbox.ru/media/code/yazyk-s-gde-ispolzuyut-chto-pishut-kak-poyavilsya-i-chem-khorosh/> (дата обращения: 12.12.2024).