Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования

«Омский государственный технический университет»

|  |  |
| --- | --- |
| Факультет (институт) | *Информационных технологий и компьютерных систем* |
|  |  |
| Кафедра | *Прикладная математика и фундаментальная информатика* |
|  |  |

**Расчетно–графическая работа**

|  |  |
| --- | --- |
| по дисциплине | ***Алгоритмизация и программирование*** |
|  |  |
| на тему | Программная реализация задач |

Пояснительная записка

|  |  |
| --- | --- |
| **Шифр проекта** | 020–РГР–02.03.02–№ 21 – ПЗ |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Студента** | | Завьялов Егор Сергеевич | | | | | |
|  |  |  |  | | фамилия, имя, отчество полностью | | | | | |
|  |  |  | Курс | 1 |  | Группа | | ФИТ-242 | | |
|  |  |  |  |  |  | |  |  | |  |
|  | | | **Направление (специальность)** | | | | | ***02.03.03*** | | |
|  | | | Математическое обеспечение и администрирование информационных систем | | | | | | | |
|  |  |  | код, наименование | | | | | | | |
|  |  |  | Руководитель | | ***ст. преподаватель*** | | | | | |
|  |  |  | ученая степень, звание | | | | | |
|  |  |  | ***Федотова И.В.*** | | | | | | | |
|  |  |  | фамилия, инициалы | | | | | | | |
|  |  |  | Выполнил | |  | | | | | |
|  |  |  | дата, подпись студента | | | | | |
|  |  |  |  | | | | | | | |
|  |  |  | **Работа защищена с количеством баллов** | | | | | |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | дата, подпись руководителя |  |  |  |

Омск 2024

**Содержание**

[Введение 3](#_Toc186558702)

[Задача 1 4](#_Toc186558703)

[Постановка задачи 4](#_Toc186558704)

[Ход решения задачи 5](#_Toc186558705)

[Код программы 7](#_Toc186558706)

[Примеры работы алгоритма 8](#_Toc186558707)

[Задача 2 9](#_Toc186558708)

[Постановка задачи 9](#_Toc186558709)

[Ход решения задачи 10](#_Toc186558710)

[Код программы 11](#_Toc186558711)

[Примеры работы алгоритма 12](#_Toc186558712)

[Задача 3 13](#_Toc186558713)

[Постановка задачи 13](#_Toc186558714)

[Ход решения задачи 14](#_Toc186558715)

[Код программы 15](#_Toc186558716)

[Примеры работы алгоритма 17](#_Toc186558717)

[Задача 4 18](#_Toc186558718)

[Постановка задачи 18](#_Toc186558719)

[Ход решения задачи 19](#_Toc186558720)

[Код программы 20](#_Toc186558721)

[Примеры работы алгоритма 21](#_Toc186558722)

[Заключение 22](#_Toc186558723)

[Список используемой литературы 23](#_Toc186558724)

# Введение

C# (си-шерп) — современный объектно-ориентированный язык программирования, разработанный Microsoft в рамках платформы .NET. С момента своего появления в начале 2000-х годов он стал одним из самых популярных языков, и его использование продолжает расти благодаря широкой области применения: C# используется для создания веб-приложений, мобильных приложений, настольных программ и игр.

Язык поддерживается Microsoft, что обеспечивает его постоянное развитие и обновление. Объектно-ориентированная природа C# позволяет создавать структурированные и поддерживаемые программы, а богатая экосистема библиотек и фреймворков, таких как ASP.NET и Unity, значительно ускоряет процесс разработки.

С выходом .NET Core и .NET 5, C# стал кроссплатформенным, позволяя разработчикам создавать приложения для Windows, macOS и Linux. Его актуальность подтверждается ростом облачных технологий и микросервисной архитектуры, а активное сообщество разработчиков предоставляет множество ресурсов и поддержки.

C# — это мощный инструмент для создания высококачественных и масштабируемых приложений, что делает его отличным выбором для любого проекта.

# Задача 1

## Постановка задачи

На оптовой базе имеется молоко, выпущенное несколькими фирмами.

Молоко каждой фирмы расфасовано в два вида упаковок, представляющих собой параллелепипеды. Для каждого вида упаковки каждой из фирм известна стоимость, которая включает как стоимость материала тары, так и стоимость собственно молока.

Требуется определить фирму, у которой стоимость одного литра собственно молока минимальна, а также эту стоимость.

**Примечание**

Считать что материал тары абсолютно тонкий и все плоскости параллелепипеда состоят из одного слоя материала.

Считать, что у двух упаковок одной фирмы стоимость единицы площади материала одинакова.

Считать, что у двух упаковок одной фирмы стоимость одного литра собственно молока одинакова.

**Входной файл**

Первая строка содержит целое число **N** - количество фирм (1 £ **N** £ 100).

Следующие **N** строк содержат шесть целых чисел **Xi1**, **Yi1**, **Zi1**, **Xi2**, **Yi2**, **Zi2** - размеры двух видов упаковок **i**-ой фирмы в сантиметрах (0 < **Xi1**, **Yi1**, **Zi1**, **Xi2**, **Yi2**, **Zi2** £ 100; 1 £ **i** £ **N**), а также два вещественных числа **Ci1** и **Ci2** - стоимости первой и второй упаковок соответственно у **i**-ой фирмы в рублях (0 < **Ci1**, **Ci2** £ 1000.0). В стоимости упаковок включаются как стоимость материала тары, так и стоимость собственно молока.

**Выходной файл**

Должен содержать одну строку, состоящую из целого и вещественного чисел, разделенных пробелом - номер фирмы, у которой стоимость одного литра собственно молока минимальна, а также эту стоимость в рублях (стоимость выводить с двумя знаками после запятой).

Если имеется несколько фирм с одинаковой минимальной стоимостью собственно молока, то вывести ту из них, номер которой минимален.

## Ход решения задачи

Изображение выглядит как текст, чек, документ, снимок экрана

Автоматически созданное описание

* Сохраняем список имён входных файлов в массив, чтобы перебрать циклом.
* В цикле работаем с каждым отдельным файлом, сохраняя все значения в переменные, используя условие задачи.
* Подставляем значения в выведенную формулу, чтобы посчитать стоимость упаковки.
* Сравниваем с предыдущим значением, если надо – перезаписываем его.
* Выводим значение и переходим к следующей итерации.

## Код программы

using System;

class program

{

const string path = "./../../milk";

static void Main()

{

System.IO.FileInfo[] a = new System.IO.DirectoryInfo(path).GetFiles();

for (int i = 0; i < a.Length / 2; i++)

{

Console.WriteLine("Обработка файла " + a[i]);

string[] file = System.IO.File.ReadAllLines(path + "/" + a[i]);

double Min = 1000;

int num = 0;

for (int j = 1; j <= int.Parse(file[0]); j++)

{

string[] comp = file[j].Split(' ');

float z1 = float.Parse(comp[0]), x1 = float.Parse(comp[1]), y1 = float.Parse(comp[2]);

float z2 = float.Parse(comp[3]), x2 = float.Parse(comp[4]), y2 = float.Parse(comp[5]);

float c1 = float.Parse(comp[6].Replace('.', ',')), c2 = float.Parse(comp[7].Replace('.', ','));

float res = 1000 \* ((c2 \* (x1 \* y1 + z1 \* y1 + z1 \* x1) - c1 \* (x2 \* y2 + z2 \* y2 + z2 \* x2)) / ((z2 \* x2 \* y2) \* (x1 \* y1 + z1 \* y1 + z1 \* x1) - (x2 \* y2 + z2 \* y2 + z2 \* x2) \* (z1 \* x1 \* y1)));

if (Min > res)

{

Min = Math.Round(res, 2);

num = j;

}

}

Console.WriteLine("Ответ: " + num +" " + Min.ToString().Replace(',', '.'));

Console.WriteLine("output file: " + System.IO.File.ReadAllLines(path + "/" + a[i + 10])[0] + "\n");

}

}

}

## Примеры работы алгоритма

**Входной файл**

5

1 1 1 2 2 2 0.28 1.46

1 1 1 2 2 2 0.28 1.46

1 1 1 2 2 2 0.28 1.46

1 1 1 2 2 2 0.28 1.46

1 1 1 2 2 2 0.29 1.47

**Выходные данные**

Обработка файла <Здесь имя файла>

Ответ: 5 77.5

output file: 5 77.50

# Задача 2

## Постановка 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.

## Ход решения задачи

Данная задача решается с помощью рекурсии. Выход из рекурсии – число солдат равное трём и тогда возвращаем единицу или меньше трёх, и тогда функция возвращает ноль.

## Код программы

using System;

using System.IO;

class Program

{

static void Main()

{

System.IO.FileInfo[] a = new System.IO.DirectoryInfo("./../../sold").GetFiles();

for (int i = 0; i < a.Length/2; i++) {

Console.WriteLine("Читаемый файл "+ a[i]);

string[] file = System.IO.File.ReadAllLines("./../../sold/" + a[i]);

int N = int.Parse(file[0]);

Console.WriteLine("Ответ: " + CountGroups(N));

Console.WriteLine("output file: " + System.IO.File.ReadAllLines("./../../sold" + "/" + a[i + 10])[0] + "\n");

}

}

static long CountGroups(int N)

{

if (N < 3) return 0;

return CountGroupsRecursive(N);

}

static long CountGroupsRecursive(int N)

{

if (N == 3) return 1;

if (N < 3) return 0;

return CountGroupsRecursive(N / 2) + CountGroupsRecursive((N + 1) / 2);

}

}

## Примеры работы алгоритма

**Входной файл**

9999999

**Выходные данные**

Обработка файла <Здесь имя файла>

Ответ: 48 0.27

output file: 48 0.27

# Задача 3

## Постановка задачи

Имеется кубик Рубика, у которого количество элементов вдоль каждого ребра равно **N**.

Требуется определить местоположение заданного элемента кубика после **M** вращений блоков кубика. Каждое вращение выполняется на 90 градусов по часовой стрелке или против часовой стрелки.

**Входной файл**

Первая строка содержит два целых числа **N** и **M**, разделенных пробелом - количество элементов вдоль ребра кубика и количество вращений (1 £ **N**, **M** £ 100000).

Вторая строка содержит три целых числа **XN**, **YN** и **ZN**, разделенных пробелами - начальные координаты элемента кубика (1 £ **XN, YN, ZN** £ **N**).

Следующие **M** строк содержат по данные очередного вращения типа **“A K S”**, где **A** - ось, вокруг которой проводится вращение (**A** Î ['**X**', '**Y**', '**Z**']), **K** - номер блока кубиков по этой оси, который подвергается вращению (**1** £ **K** £ **N**), **S** - направление вращения (**-1** - против часовой, **1** - по часовой).

**Выходной файл** должен содержать три целых числа **XK**, **YK** и **ZK**, разделенных пробелами - конечное положение элемента кубика.

## Ход решения задачи

* Поочерёдно загружаем файлы, чтобы отработать с каждым отдельно.
* Из каждого файла получаем и соответственно сохраняем данные.
* Вызываем функцию поворота кубика столько раз, сколько нам нужно, и передаем в неё все необходимое.
* Функция, используя формулу, полученную путём поиска зависимости, рассчитывает и возвращает новые координаты.
* Выводим результат

## Код программы

using System;

class thr

{

static int[] Calc(int cord1, int cord2, int N, int dir)

{

int num1, num2;

if (dir == 1)

{

num1 = cord2;

num2 = N - cord1 + 1;

} else

{

num1 = N - cord2 + 1;

num2 = cord1;

}

return new int[] { num1, num2 };

}

static void Main()

{

const string PATH = "../../rubik";

System.IO.FileInfo[] FileNames = new System.IO.DirectoryInfo(PATH).GetFiles();

for (int j = 0; j < FileNames.Length / 2; j++)

{

string[] file = System.IO.File.ReadAllLines(PATH + "/" + FileNames[j].ToString());

string[] NM = file[0].Split(' ');

string[] xyz = file[1].Split(' ');

int x = int.Parse(xyz[0]), y = int.Parse(xyz[1]), z = int.Parse(xyz[2]);

int N = int.Parse(NM[0]), M = int.Parse(NM[1]);

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

{

string[] actions = file[i + 2].Split(' ');

if (actions[0] == "Z" && int.Parse(actions[1]) == z)

{

int[] a = Calc(x, y, N, int.Parse(actions[2]));

x = a[0];

y = a[1];

}

else if (actions[0] == "X" && int.Parse(actions[1]) == x)

{

int[] a = Calc(y, z, N, int.Parse(actions[2]));

y = a[0];

z = a[1];

}

else if (actions[0] == "Y" && int.Parse(actions[1]) == y)

{

int[] a = Calc(z, x, N, -1 \* int.Parse(actions[2]));

z = a[0];

x = a[1];

}

}

Console.Write($"Получившийся ответ в {FileNames[j].ToString()}: {x} {y} {z} \n");

Console.WriteLine($"Ответ из решения({FileNames[j + 20].ToString()}) : " + System.IO.File.ReadAllLines(PATH + "/" + FileNames[j + 20].ToString())[0] + "\n");

}

}

}

## Примеры работы алгоритма

**Входной файл**

3 4

1 1 3

Z 3 1

Z 3 -1

Z 3 1

Z 3 -1

**Выходные данные**

Получившийся ответ в < Здесь имя файла >: 1 1 3

Ответ из решения(<Здесь имя файла >) : 1 1 3

# Задача 4

## Постановка 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.

## Ход решения задачи

Для нахождения всех комбинаций (N, K, Z) при заданном максимальном количестве монет MaxN следуем таким шагам. Сначала считываем MaxN и инициализируем счетчик count для допустимых комбинаций. Затем вычисляем максимальное значение Z с помощью логарифма по основанию 2, чтобы понять, сколько раз можно удвоить количество монет, не превышая MaxN. Запускаем цикл по Z от 1 до максимального значения. Для каждого Z вычисляем сумму 2^Z - 1и проверяем, не превышает ли она MaxN. Если превышает, выходим из цикла. Если нет, определяем максимальное K и увеличиваем count на KMax. В конце выводим общее количество комбинаций. Такой подход позволяет эффективно решить задачу, минимизируя вычисления.

## Код программы

using System;

class f

{

static void Main()

{

long MaxN = long.Parse(Console.ReadLine());

long count = 0;

int maxZ = (int)Math.Log(MaxN + 1, 2);

for (int Z = 1; Z <= maxZ; Z++)

{

long powerOf2Z = 1L << Z;

long sum = powerOf2Z - 1;

if (sum > MaxN) break;

long KMax = MaxN / sum;

count += KMax;

}

Console.WriteLine(count);

}

}

## Примеры работы алгоритма

**Входной файл**

32

**Выходные данные**

49

# Заключение

В ходе выполнения РГР мною были изучены основы языка программирования C# и основы алгоритмизации, а также были получены основы поиска информации, связанной с вышеперечисленными темами. В ходе работы я также освоил основы оформления Расчётно-Графических работ.

# Список используемой литературы

1. ru.wikipedia.org/wiki (дата обращения: 29.12.2024)
2. <https://support.microsoft.com/ru-ru/office/word-для-новичков-cace0fd8-eed9-4aa2-b3c6-07d39895886c> (дата обращения 30.12.2024)
3. Васильев А.Н «Программирование для новичков на C#». — Москва: Эксмо, 2018. — 592с.