**Содержание**

[Введение 3](#_Toc186547886)

[Постановка задачи «Отбор в разведку» 4](#_Toc186547887)

[Ход решения задачи «Отбор в разведку» 5](#_Toc186547888)

[Постановка задачи «Упаковки молока» 8](#_Toc186547889)

[Ход решения задачи «Упаковки молока» 10](#_Toc186547890)

[Постановка задачи «Отгадай число» 13](#_Toc186547891)

[Ход решения задачи «Отгадай число» 14](#_Toc186547892)

[Постановка задачи «Зельеварение» 17](#_Toc186547893)

[Ход решения задачи «Зельеварение» 19](#_Toc186547894)

[Заключение 21](#_Toc186547895)

[Список литературы 22](#_Toc186547896)

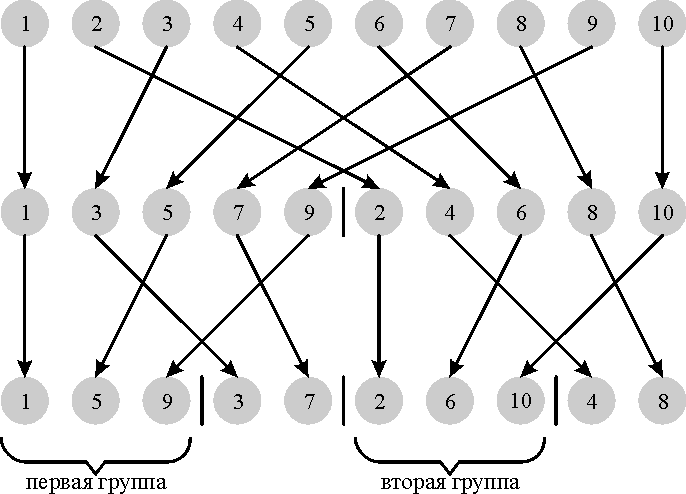
# Введение

В этой работе рассматриваются подходы к решению олимпиадных задач с использованием языка программирования C#. Олимпиадное программирование не только развивает логическое мышление, но и позволяет отточить навыки алгоритмического подхода к решению сложных задач. Язык C# предоставляет обширные возможности для создания эффективных алгоритмов и работы с различными структурами данных. Основная цель работы — изучение ключевых инструментов языка, включая работу с файлами, циклами, строками и массивами, а также углубление понимания возможностей C# в рамках разработки программ для решения задач повышенной сложности.

# Постановка задачи «Отбор в разведку»

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:***

Input.txt

10

Output.txt

2

# **Ход решения задачи «Отбор в разведку»**

Задача решается с помощью рекурсивной функции, которая находит количество групп для заданного числа n. Сначала проверяется базовое условие: если n < 3, то возвращается 0, а если n == 3, то возвращается 1. Для других значений n рекурсивно делятся два подмножества: одно с размером n / 2 (для четных значений), другое — с размером (n + 1) / 2 (для нечетных значений). Результат для каждого из подмножеств складывается и возвращается.

В главной программе значение n считывается из файла, после чего вызывается функция gruppa, которая и вычисляет итоговый результат. Сначала выполняются проверки на базовые условия, а затем рекурсивно вычисляются значения для меньших подмножеств. Итоговый результат выводится на экран.

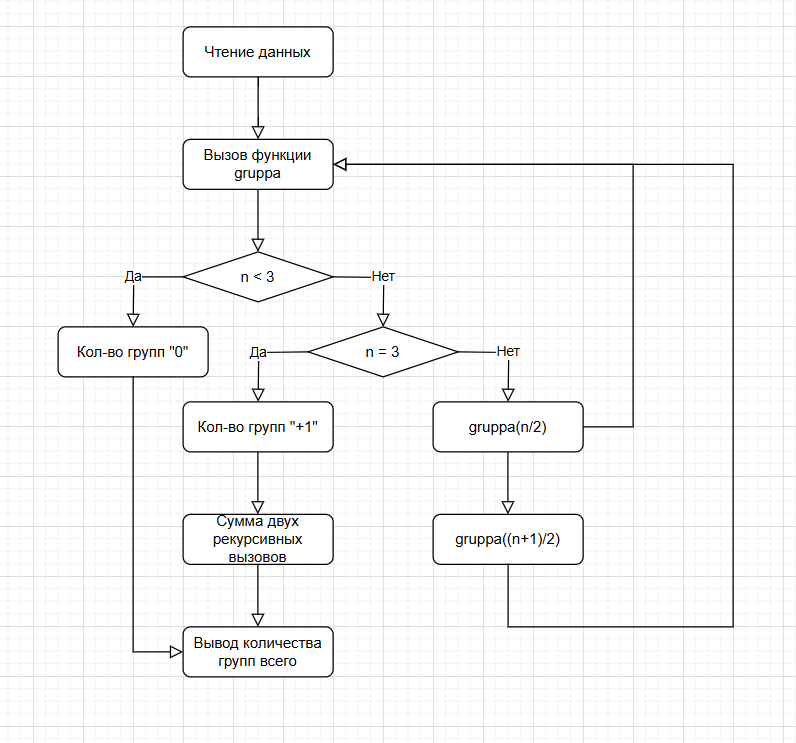


Рисунок 1 – Блок-схема для решения задачи «Отбор в разведку»

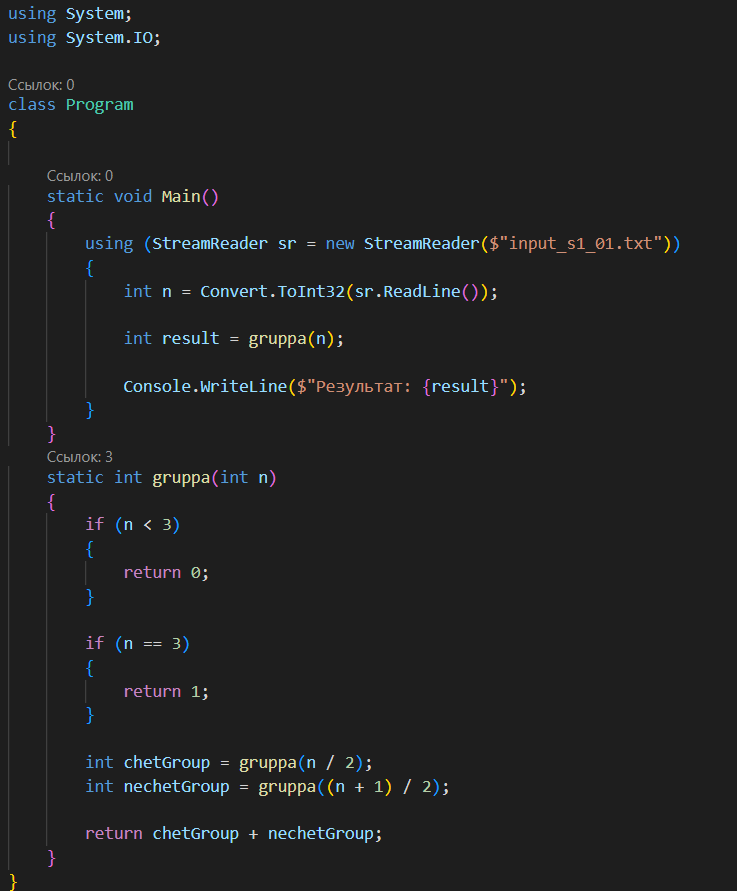


Рисунок 2 – Код программы

Проверка программы на тестах



Рисунок 3 – Результат выполнения программы – слева, входные данные – справа



Рисунок 4 – Результат выполнения программы – слева, входные данные – справа



Рисунок 5 – Результат выполнения программы – слева, входные данные – справа

# Постановка задачи «Упаковки молока»

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.

***Пример:***

Input.txt

2

10 10 5 10 10 10 12.23 20.12

5 15 20 7 8 9 43.28 16.99

Output.txt

2 4.17

# Ход решения задачи «Упаковки молока»

Задача решается путем перебора всех строк данных, представленных в файле. Каждая строка содержит параметры двух объектов: размеры (длина, ширина, высота) и стоимости. Для каждого объекта вычисляются его площадь поверхности и объём. Затем, с использованием данных о стоимости и этих геометрических характеристик, вычисляется цена объекта по заданной формуле. Формула учитывает соотношение между поверхностью и объёмом двух объектов, а также их стоимость.

Программа последовательно обрабатывает все строки, вычисляя цену для каждого объекта, и отслеживает минимальное значение цены. После обработки всех строк выводится индекс строки с минимальной ценой и сама минимальная цена, округлённая до двух знаков после запятой. В результате пользователю показывается наиболее выгодное предложение на основе заданных данных.



Рисунок 6 – Код программы, 1 часть



Рисунок 7 – Код программы, 2 часть

Проверка программы на тестах

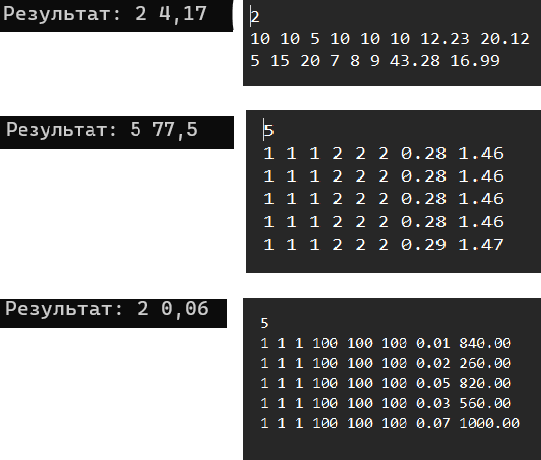


Рисунок 8 – Результат выполнения программы – слева, входные данные – справа

# Постановка задачи «Отгадай число»

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**.

# Ход решения задачи «Отгадай число»

Программа решает задачу, выполняя последовательность арифметических операций, заданных в файле, с числом, которое изначально загадал фокусник. Вначале она считывает все данные и инициализирует переменные для промежуточного результата (calculationResult) и множителя (multiplier). Затем программа обрабатывает каждую операцию по очереди, выполняя соответствующее действие с числом: прибавляет, вычитает или умножает. Если операнд операции — это символ x, то программа изменяет только множитель, а не сам результат.

После выполнения всех операций программа вычисляет исходное число, используя разницу между конечным результатом и полученным промежуточным результатом, делённую на множитель. Это позволяет восстановить задуманное фокусником число. Полученное значение выводится на экран. Таким образом, программа выполняет инвертирование операций, корректно восстанавливая исходное число на основе итогового результата.



Рисунок 9 – Код программы

Проверка программы на тестах

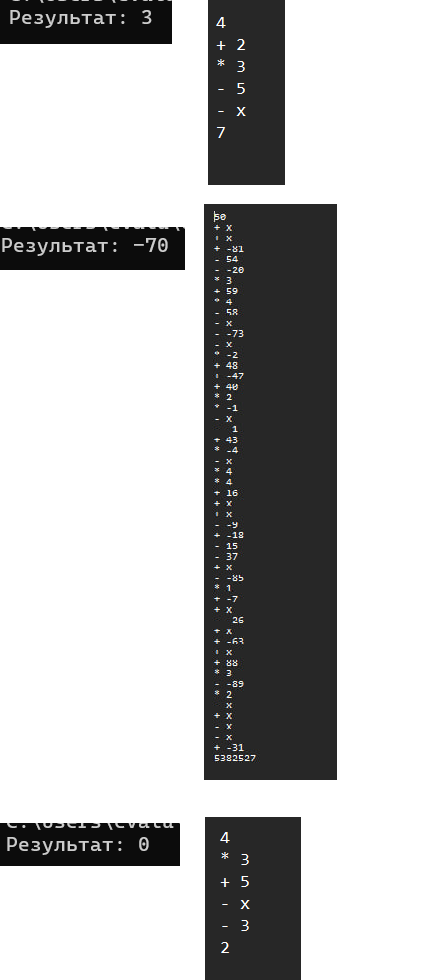


Рисунок 10 – Результат выполнения программы – слева, входные данные – справа

# Постановка задачи «Зельеварение»

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, изго—тавливающий 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.

# **Ход решения задачи «Зельеварение**»

Программа решает задачу преобразования последовательности действий в зельеварении в соответствующее заклинание. Каждый тип действия (MIX, WATER, DUST, FIRE) преобразуется в строку с конкретным форматом в зависимости от указанных ингредиентов. В процессе обработки строк программы используется регулярные выражения для поиска ссылок на предыдущие действия (с помощью номеров), а затем эти ссылки заменяются на результаты выполнения соответствующих действий. Действия могут использовать как ингредиенты, так и результаты предыдущих действий, что требует корректной обработки ссылок на результаты.

Программа поэтапно анализирует каждое действие, формируя строку заклинания, и если в нем есть ссылки на предыдущие действия, заменяет их на соответствующие результаты. Для этого используется вспомогательная функция Replace, которая находит такие ссылки (номера) и заменяет их на строки, которые были сформированы на предыдущих шагах. После завершения обработки всех действий программа выводит финальный результат в виде заклинания.



Рисунок 11 – Код программы, 1 часть

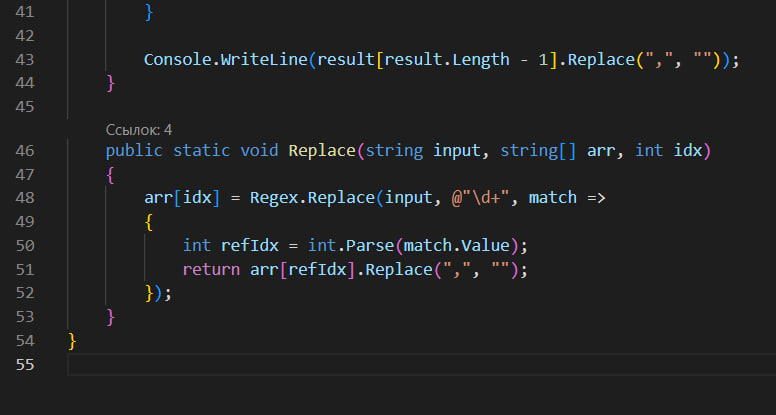


Рисунок 12 – Код программы, 2 часть

Проверка программы на тестах

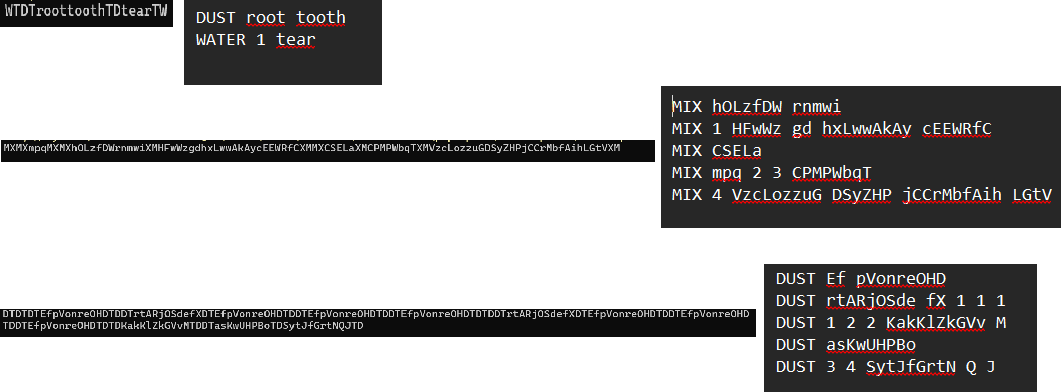


Рисунок 13 – Результат выполнения программы – слева, входные данные – справа

# Заключение

В процессе выполнения данной работы успешно решены олимпиадные задачи с применением языка программирования C#, что внесло значительный вклад в развитие логического мышления и усовершенствование навыков алгоритмического подхода к решению задач. Были изучены и применены на практике ключевые принципы работы с файлами, циклами, строками и массивами, что способствовало более глубокому освоению возможностей языка C#. Применение C# в качестве инструмента для реализации разнообразных алгоритмов и структур данных обеспечило эффективное и точное выполнение поставленных задач.

# Список литературы

1. METANIT <https://metanit.com/sharp/tutorial/2.11.php> (Дата обращения: 01.11.2024)
2. Хабр <https://habr.com/ru/companies/otus/articles/826776/> (Дата обращения: 10.11.2024)
3. Microsoft Learn <https://learn.microsoft.com/ru-ru/troubleshoot/developer/visualstudio/csharp/language-compilers/read-write-text-file> (Дата обращения: 10.11.2024)
4. METANIT <https://metanit.com/sharp/tutorial/7.2.php> (Дата обращения: 11.11.2024)
5. Хабр <https://habr.com/ru/articles/337030/> (Дата обращения: 12.11.2024)