Day 00. Процедурный C#

# Общие требования

* Убедитесь, что на вашем компьютере установлен [SDK для разработки на .NET 5](https://dotnet.microsoft.com/download) и вы используете именно его.
* Помните, ваш код будут читать! Обратите особое внимание на оформление вашего кода и именование переменных. Придерживайтесь общепринятых стандартов [C# Coding Conventions](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/inside-a-program/coding-conventions).
* Самостоятельно выберите удобную для себя IDE.
* Программа должна иметь возможность запуска через командную строку dotnet.
* В каждом из заданий указаны входные параметры и формат ответа на выходе. Необходимо придерживаться их.
* В начале каждого задания приведен список разрешенных языковых конструкций.
* Если затрудняетесь в решении задачи, обратитесь с вопросами к другим участникам бассейна, интернету, Google, посмотрите на StackOverflow.
* Избегайте хардкода и “магических чисел”.
* Вы демонстрируете все решение, верный результат работы программы — лишь один из способов проверки ее корректной работы. Поэтому когда необходимо получить определенный вывод в результате работы ваших программ, запрещено показывать предрассчитанный результат.
* Обращайте особое внимание на термины, выделенные **bold** шрифтом: их изучение пригодится вам как в выполнении текущего задания, так и в вашей дальнейшей карьере .NET разработчика.
* Have fun :)

# Требования к заданиям дня

* Все сборки должны быть в одном решении.
* Каждому из заданий должно соответствовать отдельное консольное приложение, созданное на основе стандартного шаблона .NET SDK.
* Используйте **top-level-statements** и **var**.
* Название проекта (и его отдельного каталога) должно выглядеть как d{*xx*}\_ex{*yy*}, где *xx* - цифры текущего дня, *yy* - цифры текущего задания.
* Название решения (и его отдельного каталога) - d{*xx*}, где *xx* - цифры текущего дня.
* Для форматирования выходных данных используйте **культуру** en-US: N2 для вывода денежных сумм, d для дат.
* В математических расчетах используйте округление до сотых.

Так как в первом занятии мы рассматриваем язык как процедурный:

* Нельзя использовать классы. Да, класс Program тоже!
* Нельзя использовать nuget-пакеты.

# Задание 00. Исправление опечаток

## Разрешенные языковые конструкции

* Локальные функции
* Массивы
* Циклы

## Структура проекта

d00\_ex00

Program.cs

## Задание

Представьте, что вы нашли неплохую работу с графиком, который позволит вам комфортно обучаться в Школе 21, да еще и оттачивать полученные в школе навыки. И вот на новом месте вам приходит первая задача:

“Реализовать автокоррекцию имени пользователя”

Дело в том, что пользователи системы не всегда внимательно относятся к заполнению форм и допускают опечатки. Но есть словарь, содержащий список всех имен, по которому можно проверить верность написания, и в ответ на ошибку в имени пользователя можно предложить исправление. Это вам и нужно реализовать.

У вас есть файл со списком всех англоязычных имен, приложенный к уроку. Из него нужно считать список имен для проверки. Не копируйте текст в код! Есть способы куда удобнее, посмотрите в сторону **File.IO**.

Здесь мы считаем, что словарь полный и содержит все возможные варианты (если пользователь ничего не выбирает или имя не найдено, выводится ошибка).

Для сравнения слов вы решаете использовать [расстояние Левенштейна](https://en.wikipedia.org/wiki/Levenshtein_distance). Считаем имя близким к введенному, если расстояние редактирования до него менее 2. Выберите сами, циклы использовать или рекурсию. Операции со строками, их очистку, например, от лишних пробелов, проще всего выполнять с помощью методов типа **string**.

Итак, вы должны реализовать консольное приложение, которое запрашивает имя пользователя:

>Enter name:

### Формат ответа на выходе

Далее пользователь должен ввести свое имя (оно может содержать только буквы, пробелы и дефисы), на что программа реагирует следующим образом:

|  |  |
| --- | --- |
| Имя найдено в словаре | Hello, {имя}! |
| В словаре найдено близкое имя (расстояние редактирования до которого не более 1) | >Did you mean “{исправленное имя}”? Y/N |
| Y | Hello, {исправленное имя}! |
| N | * >Did you mean “{новый вариант исправления}”? Y/N * Your name was not found. |
| Близкое имя не найдено | Your name was not found. |
| Имя не введено | Your name was not found. |

#### Пользователь указал некорректные данные

Something went wrong. Check your input and retry.

#### Пример запуска приложения из папки проекта и вывода

$ dotnet run

>Enter name:

Mrk

>Did you mean “Mark”? Y/N

Y

>Hello, Mark!

# Задание 01. Кредитный калькулятор

## Разрешенные языковые конструкции

* Локальные функции
* Массивы
* Циклы

## Структура проекта

d00\_ex01

Program.cs

## Задание

Дела на работе продвигаются неплохо, но медленно - вы слишком старательны, и очень уж много времени уходит на учебу. Чтобы не остаться без средств к существованию, вы рассматриваете вариант с кредитом. Что же, думаете вы, решив просчитать все наперед, а почему бы не составить график предстоящих выплат?

Пусть он будет выглядеть таблицей следующего вида:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № платежа | Дата платежа | Платеж | Основной долг | Проценты | Остаток долга |

Рассчитываться все будет помесячно. И для этих расчетов достаточно, чтобы ваш калькулятор принимал на входе: Сумму кредита, Годовую процентную ставку, Количество месяцев кредита.

Вы открываете кредитный договор и видите формулы, по которым банк производит вычисления:

|  |
| --- |
| Общий ежемесячный платеж:  n — количество месяцев, в которые выплачивается кредит.  i — процентная ставка по займу в месяц.  Процентная ставка по займу в месяц:  Проценты ежемесячного платежа:  Остаток общего долга - сумма основного долга на дату расчета.  Число дней периода - разность в днях между датами «Дата текущего платежа» и датой предыдущего платежа. |

Кажется, сложно, но давайте разберемся.

Будем считать, что вы выплачиваете кредит 1 числа каждого месяца, начиная со следующего. Важно учесть високосные года! В этом, а также в прочих операциях с датами и временем вам помогут методы **System.DateTime**.

Всего таких месяцев будет столько, на сколько вы взяли кредит (входной аргумент).

Ежемесячный платеж (*Платеж*) состоит из части, идущей на погашение основного долга (*Основной долг*), и части, идущей в счет оплаты процентов по кредиту (*Проценты*).

Формулы для расчета Суммы ежемесячного платежа (*Платеж*) и Процентов ежемесячного платежа (*Проценты*) приведены в кредитном договоре выше. Для математических расчетов используйте средства библиотеки **System.Math**.

Теперь, зная их, можно рассчитать и *Основной долг*.

Остаток долга на каждый месяц определить так же просто: мы просто постепенно вычитаем из общей суммы кредита ежемесячную выплату. Здесь вам помогут **циклы** и **операторы инкремента**.

Обратите внимание, если это последний месяц кредита и остаток долга ненулевой, в графике нужно просто увеличить ежемесячный платеж и сумму, идущую в уплату долга. Так мы расквитаемся с кредитом вовремя.

Осталось свести все в красивую и удобную таблицу. С выводом вам помогут **интерполяция строк** и **символы табуляции** и **переноса строки**.

Кредиты еще никогда не были такими веселыми! Правда?..

### Входные параметры

Не забудьте о [преобразовании строк в числа](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/types/how-to-convert-a-string-to-a-number)!

**Parse/TryParse**

|  |  |  |
| --- | --- | --- |
| sum | double | Сумма кредита, р |
| rate | double | Годовая процентная ставка, % |
| term | int | Количество месяцев кредита |

### Формат ответа на выходе

#### Пользователь указал некорректные данные

Something went wrong. Check your input and retry.

#### Примеры запуска приложения из папки проекта и вывода

$ dotnet run 1000000 12 10

1 06/01/2021 105,582.08 95,390.30 10,191.78 904,609.70

2 07/01/2021 105,582.08 96,659.90 8,922.18 807,949.81

3 08/01/2021 105,582.08 97,347.63 8,234.45 710,602.18

4 09/01/2021 105,582.08 98,339.77 7,242.30 612,262.40

5 10/01/2021 105,582.08 99,543.32 6,038.75 512,719.08

6 11/01/2021 105,582.08 100,356.56 5,225.52 412,362.52

7 12/01/2021 105,582.08 101,514.94 4,067.14 310,847.58

8 01/01/2022 105,582.08 102,413.99 3,168.09 208,433.60

9 02/01/2022 105,582.08 103,457.77 2,124.31 104,975.83

10 03/01/2022 105,942.18 104,975.83 966.35 0.00

$ dotnet run 55000 10 20

1 06/01/2021 2,996.95 2,529.82 467.12 52,470.18

2 07/01/2021 2,996.95 2,565.68 431.26 49,904.49

3 08/01/2021 2,996.95 2,573.10 423.85 47,331.39

4 09/01/2021 2,996.95 2,594.95 401.99 44,736.44

5 10/01/2021 2,996.95 2,629.25 367.70 42,107.19

6 11/01/2021 2,996.95 2,639.32 357.62 39,467.87

7 12/01/2021 2,996.95 2,672.55 324.39 36,795.32

8 01/01/2022 2,996.95 2,684.44 312.51 34,110.88

9 02/01/2022 2,996.95 2,707.24 289.71 31,403.64

10 03/01/2022 2,996.95 2,756.04 240.90 28,647.60

11 04/01/2022 2,996.95 2,753.64 243.31 25,893.97

12 05/01/2022 2,996.95 2,784.12 212.83 23,109.85

13 06/01/2022 2,996.95 2,800.67 196.28 20,309.18

14 07/01/2022 2,996.95 2,830.02 166.92 17,479.16

15 08/01/2022 2,996.95 2,848.49 148.45 14,630.66

16 09/01/2022 2,996.95 2,872.69 124.26 11,757.98

17 10/01/2022 2,996.95 2,900.30 96.64 8,857.67

18 11/01/2022 2,996.95 2,921.72 75.23 5,935.96

19 12/01/2022 2,996.95 2,948.16 48.79 2,987.80

20 01/01/2023 3,013.18 2,987.80 25.38 0.00