ФГАОУВО «ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
ПО ДИСЦИПЛИНЕ
ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

09.03.03 Прикладная информатика

Профиль: Разработка информационных систем бизнеса

форма обучения очная

Тема 1. Основы ООП. Объектная технология

Справка

Расширяемость, возможность повторного использования и надежность - наши главные цели - требуют выполнения ряда условий. Для их достижения требуется систематический метод декомпозиции системы на модули. В этой теме представлены основные элементы такого метода, основанного идее: строить каждый модуль на базе некоторого типа объектов.

Тема 2. Первое знакомство с C#

Справка

- В C#, как и в большинстве других языков программирования, регистр символов имеет значение! console.writeline(42) просто не скомпилируется.
- В программах нужно в точности соблюдать синтаксис языка, иначе программа не скомпилируется. В частности, каждый оператор должен заканчиваться точкой с запятой.
- Вывод на консоль производится методом Console.WriteLine
- 5.5e-2 — это так называемая «Экспоненциальная запись» действительного числа

Тема 3. Ошибки

Лабораторная «Angry Birds»

Справка

- Придется записать пару уравнений движения и решить их.
- Вероятно пригодится формула синуса двойного угла из тригонометрии: 2 sin(a) cos(a) = sin(2a)
- Для ускорения свободного падения используйте константу 9.8.

Задание

Скачайте архив с проектом AngryBirds. Откройте файл с расширением .csproj в IDE (Visual Studio или Rider). Это простой симулятор системы прицеливания. В файле AngryBirdsTask реализуйте функцию расчета угла прицеливания, в зависимости от начальной скорости снаряда и дальности до цели. Если решения не существует, метод должен возвращать double.NaN.

Проверьте корректность своего решения, запустив проект.

Вы можете изучить устройство проекта — это будет полезно, но для выполнения этого задания это совсем не обязательно. Более того, будьте готовы к тому, что в проекте активно используются ещё не пройденные темы.

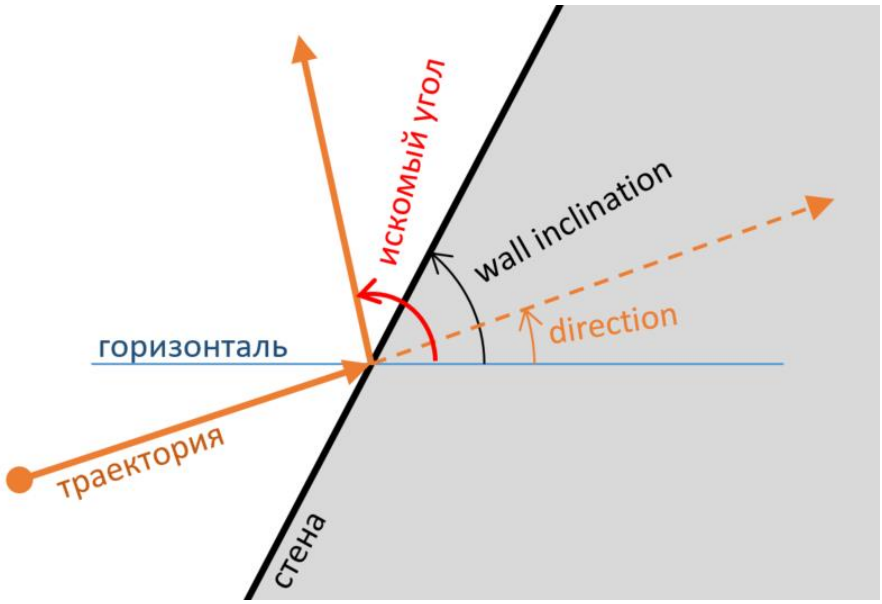
Лабораторная Бильярд

Задание

Скачайте архив с проектом Billard.

Реализуйте метод для расчета угла отскока шарика от стены. Считайте, что угол падения равен углу отражения, то есть можно пренебречь всеми физическими эффектами, связанными с кручением шаров, трением шара об стенку и т.п.

Смысл всех, используемых в задаче углов проиллюстрирован на схеме:



Лабораторная Проценты

Справка

- `Строку можно разбить на части по пробелам с помощью метода `Split` вот так `var parts = str.Split()`.`
  `К частям строки после разбиения можно обращаться вот так `var sum = parts[0]; var rate = parts[1]; ...``
- Напоминаем, что преобразовать строку в число можно одним из методов `int.Parse`, `double.Parse`
- Вспомните, что такое геометрическая прогрессия и как считается её сумма.
- Изучите как работает капитализация и подумайте, причём тут геометрическая прогрессия.
- Ещё раз обратите внимание, что процентная ставка годовая, а срок вклада указан в месяцах

Задание

В этой задаче вам нужно самостоятельно создать с нуля консольное приложение, которое рассчитывает банковские проценты.

Пользователь должен ввести исходные данные с консоли — три числа через пробел: исходную сумму, процентную ставку (в процентах) и срок вклада в месяцах.

Программа должна вывести накопившуюся сумму на момент окончания вклада.

Вот порядок действий:

1. Создайте новый проект с типом Console Application.
2. В методе Main напишите ввод с помощью Console.ReadLine() и вывод с помощью Console.WriteLine().
3. Все вычисление вынесите во вспомогательный метод Calculate. Код этого метода и нужно сдать в этой задаче.

Лабораторная Рефакторинг

Справка

Задание

Скачайте архив с проектом Risovatel. Работайте в файле DrawingProgram.cs.

Ваша задача привести код в этом файле в порядок. Для начала запустите эту программу. Должно появиться окно с изображением невозможного квадрата.

1. Переименуйте всё, что называется неправильно. Это можно делать двойным нажатием комбинации клавиш Ctrl+R (работает для VS и Rider).
2. Исправьте форматирование кода. Частично с этим поможет комбинация клавиш: Ctrl+K, Ctrl+D для VS, Ctrl+Alt+L для Rider, Ctrl+Alt+Enter для Rider, использующий схему клавиш VS.
3. Повторяющиеся части кода вынесите во вспомогательные методы. Это можно сделать, выделив несколько строк кода и нажав: Ctrl+R, Ctrl+M для VS, Ctrl+Alt+M для Rider.
4. Избавьтесь от всех зашитых в коде числовых констант — положите их в переменные с понятными именами.

После того как вы закончите, Ulearn bot не должен выдавать ни одного замечания.

Тема 4. Основы ООП. Абстрактные типы данных.

Справка

Абстрактный тип данных (АТД) — это математическая модель для типов данных, где тип данных определяется поведением (семантикой) с точки зрения пользователя данных, а именно в терминах возможных значений, возможных операций над данными этого типа и поведения этих операций.

Тема 5. Ветвления

Лабораторная Рубль -лей -ля

Задание

Напишите метод, склоняющий существительное «рублей» следующее за указанным числительным.

Например, для аргумента 10, метод должен вернуть «рублей», для 1 — вернуть «рубль», для 2 — «рубля».

Лабораторная Два прямоугольника

Справка

1. Прямоугольники пересекаются, если пересекаются их проекции.
2. Площадь пересечения — это произведение длин пересечения проекций прямоугольников на оси OX и OY.
3. Первый прямоугольник находится во втором, если во втором находятся две противоположные вершины первого.
4. Задачу пересечения прямоугольников легко свести к задаче пересечения отрезков на числовой прямой

Задание

Вам даны два прямоугольника на плоскости, со сторонами параллельными осям координат с целочисленными координатами.

Реализуйте в классе RectanglesTask.cs три метода для работы с прямоугольниками:

- определение, есть ли у двух прямоугольников хотя бы одна общая точка (и граница и внутренность считаются частью прямоугольника);
- вычисление площади пересечения;
- определение, вложен ли один в другой.

Решите задание без использования библиотечных методов, кроме Min и Max.

Обратите внимание, что ваше решение должно корректно работать с вырожденными прямоугольниками: у которых длина или ширина равны 0.

Для проверки своего решения запустите скачанный проект.

В мире компьютерной графики принято, что верхний левый угол экрана имеет координаты (0, 0), а ось Y направлена вниз, а не вверх, как принято в математике. Поэтому в этой задаче нижний край прямоугольника имеет большую координату, чем верхний. Учитывайте это при решении задачи!

Лабораторная Расстояние до отрезка

Справка

- Сначала нужно определить пересекает ли отрезок, высота, опущенная из точки на отрезок. В этом могут помочь скалярные произведения векторов.
- Длину высоты, опущенной на отрезок можно определить через площадь треугольника.

Задание

Напишите метод вычисления расстояния от отрезка до точки.

Для проверки своего решения запустите скачанный проект.

Расстоянием от отрезка до точки называется расстояние от ближайшей точки отрезка до точки. Это либо расстояние до точки от прямой, содержащей отрезок, либо расстояние до точки от одного из концов отрезка.

Тема 6. Циклы

Лабораторная «Пустой лабирит»

Справка

Важно уметь создавать такие методы, чтобы они были понятны без необходимости заглядывать внутрь метода. При работе над большим проектом это позволит вам и вашим коллегам быстрее ориентироваться в коде.

Задание

Скачайте проект Mazes и изучите его. Там заготовлены несколько фиксированных лабиринтов. В каждом лабиринте вам нужно довести робота до выхода — клетки, помеченной зеленым кружком.

Сам лабиринт не известен и у робота нет сенсоров, чтобы его исследовать, но известен тип лабиринта и его размеры.

Этого достаточно, чтобы построить маршрут и отдать нужные команды роботу.

В этой задаче лабиринт — это пустая комната, окружённая стеной по периметру, в которой из верхнего левого угла с координатами (1, 1) нужно пройти в правый нижний с координатами (width-2, height-2).

Дополнительные ограничения:

1. Запрещено использовать более одного цикла в одном методе.
2. Запрещено иметь методы длиннее 6 строк кода.

3. Запрещено использовать ключевое слово `catch`
4. Разрешено создавать вспомогательные методы, но только понятными именами, в том числе именами аргументов.

Лабораторная «Лабиринт змейка»

Задание

В том же проекте выведите робота из лабиринта вида «змейка».
Запустите проект и самостоятельно изучите этот тип лабиринтов.
Дополнительные ограничения:

1. Запрещено использовать более одного цикла в одном методе.
2. Запрещено иметь методы длиннее 12 строк кода.
3. Запрещено использовать ключевое слово `catch`
4. Разрешено создавать вспомогательные методы, но только понятными именами, в том числе именами аргументов.

Обратите внимание, чтобы в вашем коде не было дублирующихся почти одинаковых методов.

Лабораторная «Лабиринт диагональ»

Задание

В том же проекте выведите робота из лабиринта вида «диагональ».
Запустите проект и самостоятельно изучите этот тип лабиринтов.
Дополнительные ограничения:

1. Запрещено использовать более одного цикла в одном методе.
2. Запрещено иметь методы длиннее 12 строк кода.
3. Запрещено использовать ключевое слово `catch`
4. Разрешено создавать вспомогательные методы, но только понятными именами, в том числе именами аргументов.

Лабораторная Dragon curve

Справка

Как генерировать случайные числа?

Для этого в пространстве имен System есть класс Random. Работать с ним нужно так:

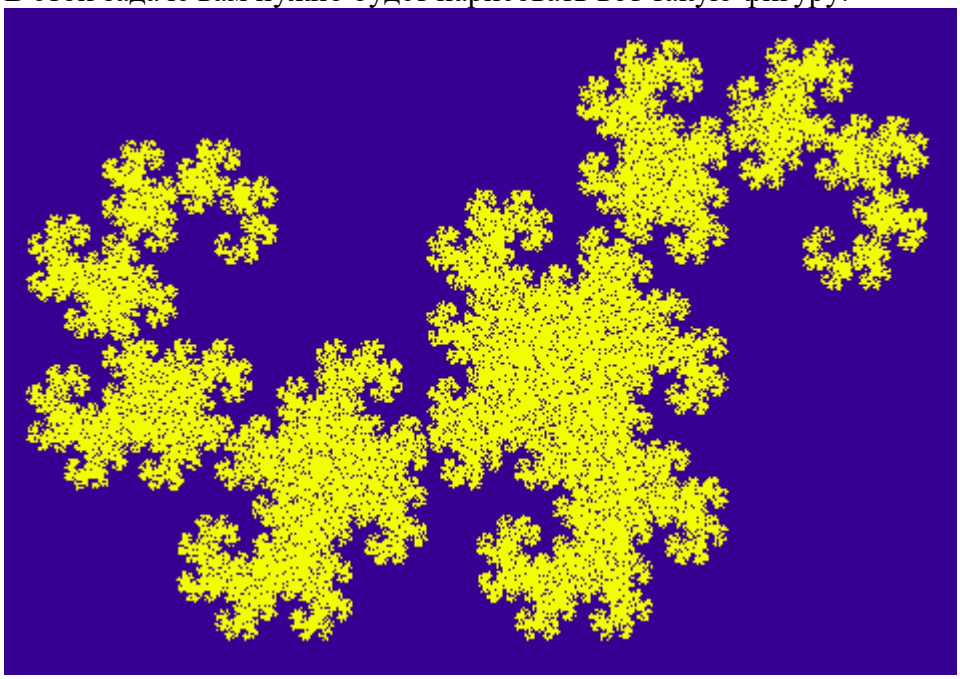
```
// 1. Создание нового генератора последовательности случайных чисел:
var random = new Random(seed);
// seed — число полностью определяющее все последовательность псевдослучайных чисел этого генератора.

// 2. Получение очередного псевдослучайного числа от 0 до 9:
var nextNumber = random.Next(10);
```

Если при инициализации генератора случайных чисел не указывать seed, то используется текущее время компьютера с точностью до миллисекунд. Поэтому если вы создадите два генератора подряд, то с большой вероятностью они проинициализируются одинаково и будут выдавать одну и ту же последовательность.

Типичная ошибка начинающих — поместить обе операции внутрь цикла, тогда как правильно вынести создание генератора за пределы цикла, оставив внутри только получение следующего числа.

Задание

В этой задаче вам нужно будет нарисовать вот такую фигуру:



Тема 7. Основы ООП. Статические структуры. Классы.

Справка

Класс — в объектно-ориентированном программировании, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.

Тема 8. Массивы

Лабораторная «Гистограмма»

Задание

Скачайте проект Names и изучите его. В классе HistogramTask напишите код подготовки данных для гистограммы частоты рождаемости в зависимости от числа (номера дня в месяце) их рождения для заданного имени.

В это задаче используются реальные данные о людях. Но так получилось, что в базе данных для всех людей с неизвестной датой рождения, в качестве даты рождения используется первое число месяца. Во время работы с реальными базами данных, часто всплывают подобные особенности.

Чтобы это не мешало нам в этой задаче, просто не учитывайте тех, кто родился 1 числа любого месяца.

В качестве подписей (label) по оси X используйте массив размером 31 элемент со значениями от 1 до 31.

Если вас пугает незнакомое слово гистограмма — вам как обычно в википедию!

Пример подготовки данных для гистограммы смотри в файле HistogramSample.cs

Проанализируйте наблюдаемый результат для имён Владимир и Юрий. Попробуйте найти объяснение такой форме гистограмм?

Лабораторная «Тепловая карта»

Задание

В том же проекте в классе HeatmapTask напишите код подготовки данных для тепловой карты рождаемости в зависимости от дня и месяца.

Подготовьте данные для построения карты интенсивностей, у которой по оси X — число месяца, по Y — номер месяца, а интенсивность точки (она отображается цветом и размером) обозначает количество рожденных людей в это число этого месяца. Аналогично предыдущей задаче, не учитывайте людей, родившихся первого числа любого месяца.

В качестве подписей (label) по X используйте массив возможных чисел месяца, кроме первого: от 2 до 31. В качестве подписей по Y используйте массив номеров месяцев от 1 (январь), до 12 (декабрь).

Таким образом, данные для карты интенсивностей должны быть в виде двумерного массива 30 на 12 — от 2 до 31 числа и от января до декабря.

В этой задаче, ваш код должен корректно работать на любых данных, а не только на том наборе, которые содержится в архиве с задачей. Это проверяется секретными тестами.

Лабораторная «Интересное про имена»

Справка

Для визуализации вы можете использовать наработки, сделанные в практиках «Гистограмма» и «Тепловая карта», но вы получите больше баллов, если разберетесь в библиотеке для рисования графиков и диаграмм ZedGraph, либо любой другой, и сделаете непохожую на предыдущие визуализацию. Документация на ZedGraph легко ищется в интернете.

Задание

Это творческая задача. Она не отправляется на ulearn, а показывается преподавателю на компьютерной практике.

В предыдущих двух практиках вы исследовали данные об именах и днях рождениях из файла names.txt. Это реальные данные. Продолжите их исследовать. Посчитайте разные статистики, посмотрите на данные с разных сторон и найдите что-то интересное.

Визуализируйте эту интересную статистику в рамках того же проекта. Визуализация должна быть подобрана так, чтобы наглядно демонстрировать интересный факт, который вы обнаружили.

На паре вы будете защищать интересность найденного факта перед преподавателем. За визуализацию первого попавшегося скучного факта вроде «В 1950 году родилось 559 Юр» баллов вы не получите.

Тема 9. Коллекции, строки, файлы

Лабораторная «Парсер предложений»

Справка

- Разделить текст на предложения можно методом string.Split, указав ему все возможные разделители
- Чтобы разделить предложение на слова, придется написать проход по строке циклом. Лучше сразу выделить эту логику во вспомогательный метод
- Распространенная ошибка — забыть обработать последнее слово в предложении.
- Не забудьте отфильтровать предложения, в которых не оказалось слов.

Задание

В этом задании нужно реализовать метод в классе `SentencesParserTask`. Метод должен делать следующее:

1. Разделять текст на предложения, а предложения на слова.

a. Считайте, что слова состоят только из букв (используйте метод `char.IsLetter`) или символа апострофа `'` и отделены друг от друга любыми другими символами.

b. Предложения состоят из слов и отделены друг от друга одним из следующих символов `.!?;:()`

2. Приводить символы каждого слова в нижний регистр.
3. Пропускать предложения, в которых не оказалось слов.

Метод должен возвращать список предложений, где каждое предложение — это список из одного или более слов в нижнем регистре.

Лабораторная «Частотность N-грамм»

Справка

По тексту `a b c d. b c d. e b c a d.` должен быть составлен такой словарь:

```
"a": "b"
"b": "c"
"c": "d"
"e": "b"
"a b": "c"
"b c": "d"
"e b": "c"
"c a": "d"
```

Обратите внимание:

- из двух биграмм "a b" и "a d", встречающихся однократно, в словаре есть только пара "a": "b", как лексикографически меньшая.
- из двух встречающихся в тексте биграмм "c d" и "c a" в словаре есть только более частотная пара "c": "d".
- из двух триграмм "b c d" и "b c a" в словаре есть только более частотная "b c": "d".

Задание

Продолжайте работу в том же проекте.

N-грамма — это N соседних слов в одном предложении. 2-граммы называют биграммами. 3-граммы — триграммами.

Например, из текста: "She stood up. Then she left." можно выделить следующие биграммы "she stood", "stood up", "then she" и "she left", но не "up then". И две триграммы "she stood up" и "then she left", но не "stood up then".

По списку предложений, составленному в прошлой задаче, составьте словарь самых частотных продолжений биграмм и триграмм. Это словарь, ключами которого являются все возможные начала биграмм и триграмм, а значениями — их самые

частотные продолжения. Если есть несколько продолжений с одинаковой частотой, используйте то, которое лексикографически меньше.

Для лексикографического сравнения используйте встроенный в .NET способ сравнения Ordinal, например, с помощью метода string.CompareOrdinal.

Такой словарь назовём N-граммной моделью текста.

Реализуйте этот алгоритм в классе `FrequencyAnalysisTask`.

Все вопросы и детали уточняйте с помощью примера ниже и тестов.

Лабораторная «Продолжение текста»

Справка

Подобные N-граммные модели текстов часто используются в самых разных задачах обработки текстов. Когда поисковая строка предлагает вам продолжение вашей фразы — скорее всего это результат работы подобного алгоритма.

Сравнивая частоты N-грамм можно сравнивать тексты на похожесть и искать плагиат.

Опираясь на N-граммные модели текстов можно улучшать алгоритмы исправления опечаток или автокоррекции вводимого текста.

Задание

В классе TextGeneratorTask реализуйте алгоритм продолжения текста по N-граммной модели.

Описание алгоритма:

На вход алгоритму передается словарь `nextWords`, полученный в предыдущей задаче, одно или несколько первых слов фразы `phraseBeginning` и `wordsCount` — количество слов, которые нужно дописать к `phraseBeginning`.

Словарь nextWords в качестве ключей содержит либо отдельные слова, либо пары слов, соединённые через пробел. По ключу key содержится слово, которым нужно продолжать фразы, заканчивающиеся на key.

Алгоритм должен работать следующим образом:

1. Итоговая фраза должна начинаться с `phraseBeginning`.
2. К ней дописывается `wordsCount` слов таким образом:

a. Если фраза содержит как минимум два слова и в словаре есть ключ, состоящий из двух последних слов фразы, то продолжать нужно словом, из словаря по этому ключу.

b. Иначе, если в словаре есть ключ, состоящий из одного последнего слова фразы, то продолжать нужно словом, хранящемся в словаре по этому ключу.

c. Иначе, нужно досрочно закончить генерирование фразы и вернуть сгенерированный на данный момент результат.

Проверяющая система сначала запустит эталонный способ разделения исходного текста на предложения и слова, потом эталонный способ построения словаря наиболее частотных продолжений из предыдущей задачи, а затем вызовет реализованный вами метод. В случае ошибки вы увидите исходный текст, на котором запускался процесс тестирования.

Если запустить проект на выполнение, он предложит ввести начало фразы и сгенерирует продолжение. Позапускайте алгоритм на разных текстах и разных фразах. Результат может быть интересным!

Тема 10. Сложность алгоритмов. Понятие алгоритма. Эффективность. Основы анализа алгоритмов.

Справка

Анализ алгоритмов даёт нам инструмент для выбора алгоритма. Результат анализа алгоритмов — не формула для точного количества секунд или компьютерных циклов, которые потребует конкретный алгоритм.

Тема 11. Тестирование

Лабораторная «Поля в кавычках»

Справка

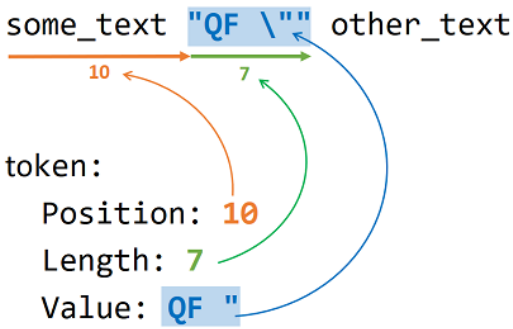
При решении задач разбора текста, например, вычленении из строки фрагментов, приходится отслеживать 2 вещи:

1. Проинтерпретированное значение, соответствующее фрагменту. Например, значение фрагмента `"a\"b"` это три символа: `a`, символ кавычки и `b`, хотя сам фрагмент имеет длину 6 символов.
2. Позицию фрагмента в строке.

Для этого удобно завести свой тип и хранить там эту информацию. Этот тип уже создан для вас — найдите и изучите в проекте класс `Token`. Фрагменты, выделяемые из текста, часто называют токенами или лексемами. Мы дальше будем называть их токенами.

Иллюстрация, поясняющая семантику свойств класса Token при анализе текста:

```
var token = ReadQuotedField(
    @"some_text ""QF \""""" other_text", 10)
```



- Position — номер символа в строке, с которого начинается запись токена.
- Length — количество символов, которое занимает токен в строке (открывающие и закрывающие кавычки включительно).
- Value — это проинтерпретированное значение токена. То есть «очищенное» от ненужных кавычек и слэшей.

Задание

В этой серии задач вам нужно реализовать парсер полей. Парсером обычно называют алгоритм, который из текста делает набор объектов, представленных в этом тексте.

Итак, на вход нашего парсера подается строка текста. В результате выполнения серии этих задач у вас получится алгоритм, возвращающий список полей, извлечённых из текста по описанным ниже правилам, либо пустой список, если полей в исходном тексте не оказалось.

Проверить работу парсера можно запустив программу. Появится окно, в котором можно вводить разные входные строки и смотреть, на какие поля эти строки разбиваются.

В этой задаче реализуйте один вспомогательный метод ReadQuotedField, для чтения полей в кавычках. В следующей задаче вам нужно будет воспользоваться этим методом.

Метод ReadQuotedField принимает строку и позицию в строке. Гарантируется, что на стартовом индексе находится открывающая кавычка поля в кавычках. Метод должен обработать символы до следующей кавычки, если она есть (если нету, то до конца строки), и вернуть токен поля в кавычках. См. пример на схеме выше.

Проверить корректность работы можно запустив программу на выполнение. Но кроме ручного тестирования вам необходимо создать автоматические тесты в классе QuotedFieldTests в том же файле. Удобно создавать тесты, добавляя новые атрибуты TestCase (см. туториал по запуску тестов).

На проверку нужно отправлять файл, в котором есть и решение задачи и тесты.

Лабораторная «Тестирование»

Справка

- Попробуйте из каждого требования к задаче сделать тест
- Иногда удобно использовать символ @ и @"буквальные" строки, внутри которых не нужно экранировать ничего кроме двойной кавычки

Задание

В этой задаче вам не нужно реализовывать алгоритм. Вместо этого напишите набор тестов, который покрывает все основные ситуации для задачи, описанной в прошлом слайде. Обратите внимание, что вам нужны тесты на всё задание, а не только на поле в кавычках.

Каждый тест должен быть новым атрибутом у метода `RunTests`. То есть, просто дописать ещё раз строку `[TestCase(..., ...)]`, где первым аргументом указать входные данные, а вторым ожидаемый вывод.

Не пишите сложных тестов, которые проверяют сразу много различных свойств алгоритма. Если такой тест падает, то сложно понять, в чем на самом деле была ошибка.

Лабораторная «Парсер полей»

Задание

В классе FieldsParserTask реализуйте метод ParseLine, для которого вы создавали тесты в предыдущей задаче.

Создайте модульные тесты на это решение и перенесите разработанные в прошлой задаче тестовые случаи в модульные тесты.

Решение получится более простым, если ваши вспомогательные методы будут использовать `Token` в качестве возвращаемого значения.

В качестве вспомогательных методов могут быть методы, читающие разные виды полей (у вас уже реализован метод `ReadQuotedField`), а также метод пропускающий пробелы между полями.

Обратите внимание на метод `GetIndexNextToToken` в классе `Token`. Он возвращает позицию, с которой нужно продолжить анализ строки.

Тема 12. Сложность алгоритмов

Лабораторная

Задание

Представьте себе робота-уборщика на кухне, которого только что случайно пнула хозяйка. Ему нужно сориентироваться, где он теперь находится и куда повёрнут. К счастью у робота есть камера, а пол на кухне выложен квадратной кафельной плиткой. Осталось немного обработать изображение с видеокамеры, выделить границы объектов и по ним сориентироваться.

Первым шагом нужно перевести цветное изображение в оттенки серого. Его будет проще анализировать.

Лабораторная «Пороговый фильтр»

Справка

- Не забудьте обработать угловые и граничные пиксели
- Имейте в виду, что параметр метода whitePixelsFraction — это доля пикселей, которые должны стать белыми, а не само пороговое значение
- Обратите внимание, что если пиксель некоторого цвета алгоритм сделал белым, то и все другие пиксели того же цвета должны стать белыми
- Например, одноцветное изображение 10x10 при whitePixelsFraction=0.2 должно целиком стать белым. Потому что как минимум 20% пикселей нужно сделать белыми, но остальные 80% имеют тот же цвет, а значит тоже должны стать белыми

Задание

Пора превратить изображение в черно-белое.

Сделать это можно с помощью порогового преобразования. Реализуйте его в методе

```
public static double[,] ThresholdFilter(double[,] original, double whitePixelsFraction)
```

Метод должен заменять пиксели со значением больше либо равному порогу `T` на белый (1.0), а остальные на черный (0.0).

Пороговое значение `T` найдите так, чтобы:

- если N — общее количество пикселей изображения, то как минимум (int)(whitePixelsFraction*N) пикселей стали белыми;
- при этом белыми стало как можно меньше пикселей.

Лабораторная «Фильтр Собеля»

Задание

Перед преобразованием в чёрно-белое, хорошо бы каким-то образом выделить границы объектов, чтобы только они стали белыми, а всё остальное черным.

Оказывается, это не сложно сделать с помощью так называемого фильтра Собеля. Он уже реализован в файле SobelFilterTask.cs. Ваша задача — обобщить этот код. Подробности — в комментариях!

Выполните эту задачу в файле SobelFilterTask.cs

Тема 13. Рекурсивные алгоритмы

Лабораторная «Перебор паролей»

Справка

- Не все буквы имеют верхний и нижний регистр! Например, еврейский алфавит не имеет различий в регистрах.

- У некоторых не букв тоже есть верхний и нижний регистр. Например, у римских цифр (для них есть отдельные символы Unicode) есть регистр, но это не буквы, а цифры.

Задание

Вася сменил пароль на новый и забыл его!

На этот раз он точно помнит, что он сконструировал пароль из старого пароля, поменяв регистр нескольких букв. Он, конечно, не хочет вам говорить старый пароль, поэтому просит написать программу, которая по заданному слову перебирает все возможные пароли, полученные из этого слова заменой регистра.

Для удобства мы создали для вас проект, который будет тестировать вашу программу. Вам лишь остается дописать класс CaseAlternatorTask в одноименном файле.

Для удобства Вася просит, чтобы пароли появлялись в лексикографическом порядке, считая, что маленькие буквы меньше больших. Естественно, регистр нужно менять только у букв.

Например, для входного слова `'ab42'` результат должен быть такой: `'ab42', 'aB42', 'Ab42', 'AB42'`

На вход подается слово в нижнем регистре. В результирующем списке не должно быть повторений слов.

Помните, что у вас в распоряжении есть методы `char.IsLetter`, `char.ToLower` и `char.ToUpper`.

Лабораторная «Хождение по чекпоинтам»

Справка

- Создайте вспомогательный рекурсивный метод, подумайте, какие аргументы ему понадобятся
- В классе PointExtensions есть готовые методы DistanceTo и GetPathLength

Задание

В файле PathFinderTask допишите код функции `int[] FindBestCheckpointsOrder(Point[] checkpoints)`.

Функция принимает массив чекпоинтов. Робот изначально находится в точке `checkpoints[0]`. Вернуть нужно порядок посещения чекпоинтов. Например, если функция возвращает массив {0,2,1}, это означает, что робот сначала поедет в чекпоинт с индексом 2, а из него в чекпоинт с индексом 1 и на этом закончит свой путь.

Действуйте как на лекциях, можете адаптировать код с лекций. Функция должна быть рекурсивной.

Реализуйте следующую оптимизацию (отсечение перебора): прекращайте перебор, если текущая длина пути уже больше, чем минимальный путь, найденный ранее.

Тема 14. Поиск и сортировка

Лабораторная «Левая граница»

Справка

- Длина множества фраз слишком большая. Избегайте переполнения при арифметике с индексами.
- При первом вызове left = -1, a right = phrases.Length.

Задание

Во многих программах в разных контекстах можно увидеть функцию автодополнения вводимого текста. Обычно это работает так: есть словарь всех допустимых значений, и когда пользователь вводит начало некоторого слова, ему показывают несколько подходящих слов из словаря, начинающихся с букв, уже введенных пользователем.

Такую функцию очень просто реализовать "в лоб", если словарь небольшой. Если же словарь большой, то необходимо задумываться об эффективности алгоритма.

Запустите проект autocomplete и поизучайте программу. В частности попробуйте набрать префиксы a, ab, zzz. На zzz поиск будет заканчиваться таймаутом.

В следующих трех заданиях нужно будет внедрить в эту программу бинарный поиск и ускорить её!

Начать нужно с простого. В файле LeftBorderTask.cs реализуйте бинарный поиск левой границы в упорядоченном множестве фраз. Подробности в комментариях в файле LeftBorderTask.cs!

Лабораторная «Правая граница»

Задание

По аналогии с предыдущим заданием, в файле RightBorderTask.cs реализуйте бинарный поиск правой границы в упорядоченном множестве фраз. Подробности в комментариях в файле RightBorderTask.cs!

Лабораторная «Автодополнение»

Задание

В файле AutocompleteTask.cs реализуйте методы GetTopByPrefix и GetCountByPrefix. Проверить корректность можно запустив проект autocomplete. Теперь отображаться должен не один вариант, а 10. А в строке статуса отображаться общее количество подходящих фраз.

На эти два метода нужно написать модульные тесты с использованием библиотеки NUnit. Они должны быть в том же файле AutocoompleteTask.cs отдельным классом AutocompleteTests.

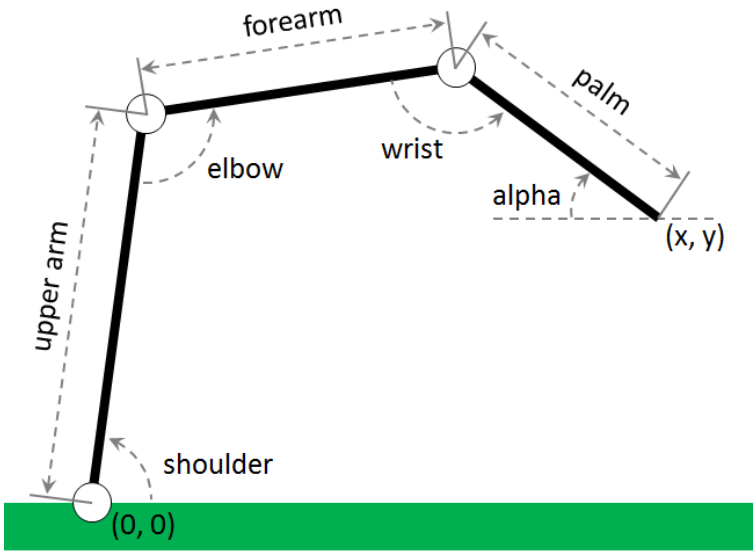
Тема 15. Сложность алгоритмов. Рекурсия. Принцип декомпозиции. Основы анализа рекурсивных алгоритмов.

Справка

Рекурсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя.

Тема 16. Практикум

Справка



Лабораторная «Манипулятор»

Задание

В первой задаче вам нужно по величине углов shoulder, elbow и wrist вычислить координаты соответствующих суставов.

Для этого в классе AnglesToCoordinatesTask реализуйте метод `PointF[] GetJointPositions(double shoulder, double elbow, double wrist)`

В том же файле в классе AnglesToCoordinatesTask_Tests напишите модульные тесты, покрывающие все принципиальные случаи.

Используйте константы UpperArm, Forearm и Palm из класса Manipulator в качестве размеров соответствующих частей манипулятора.

После выполнения этого задания, при запуске проекта, визуализация должна будет отражать положение манипулятора в зависимости от начальных значений.

Лабораторная «Визуализация»

Задание

1. В методе KeyDown сделайте, чтобы манипулятор реагировал на клавиши QASW таким образом:
   - по Q увеличивает угол Shoulder на небольшую величину, а по A — уменьшает;
   - по W увеличивает угол Elbow на небольшую величину, а по S — уменьшает;
   - при любых изменениях пересчитывает Wrist по формуле Wrist = - Alpha - Shoulder - Elbow;
2. В методе MouseMove менял бы X и Y в соответствии со значением из события — координатами мыши относительно окна. Имейте в виду, что ось Y в окне направлена вниз, а в математике — вверх. Поэтому координаты мыши нужно преобразовывать из оконной системы координат в логическую, а при отрисовке наоборот. X и Y в итоге должны хранить логические координаты указателя мыши, относительно shoulderPos — координат единственного неподвижного сустава. Преобразовать оконные координаты в логические и наоборот можно с помощью пары готовых функций ConvertMathToWindow и ConvertWindowToMath.
3. В методе MouseWheel добавить обработку прокрутки колеса мыши. Оно должно менять Alpha.

4. В методе UpdateManipulator вызвать ManipulatorTask.MoveManipulatorTo и обновить значения Shoulder, Elbow и Wrist (это понадобится в последней задаче). UpdateManipulator нужно вызывать после каждого изменения X, Y или Alpha, то есть в методах MouseMove и MouseWheel.
5. В методе DrawManipulator допишите рисование манипулятора. Нарисуйте каждый сегмент манипулятора отрезком прямой, а каждый сустав — окружностью. Координаты сустава получите методом AnglesToCoordinatesTask.GetJointPositions. Не забудьте преобразовать логические координаты в оконные.

Лабораторная «Поиск угла»

Задание

Реализуйте метод `double GetABAngle(a, b, c)`. Он должен возвращать угол в радианах между сторонами a и b в треугольнике со сторонами a, b, c.

Естественно, для практических целей треугольник может быть вырожденным, то есть некоторые стороны могут иметь длину 0. При недопустимых аргументах или при невозможности определить угол в вырожденном треугольнике метод должен возвращать `double.NaN`.

В том же файле напишите модульные тесты, покрывающие все случаи. В том числе и особые, граничные случаи.

Лабораторная «Решение манипулятора»

Задание

Реализуйте метод MoveManipulatorTo в классе ManipulatorTask.

Он должен возвращать массив углов `new[] {shoulder, elbow, wrist}`, необходимых для приведения эффектора манипулятора в точку `(x, y)` относительно крепления манипулятора к столу, и с углом между последним суставом и горизонталью равному `alpha` в радианах.

Если это невозможно, то возвращайте массив из трех `double.NaN`.

Сверяйтесь с чертежом! Не перепутайте углы и их направления!

Тема 17. Основы ООП

Лабораторная «Вектор»

Задание

Создайте новый проект в Visual Studio. Выберите в качестве типа проекта Class Library.

В этом проекте создайте два класса, Vector и Geometry, в пространстве имен GeometryTasks.

В классе Vector должно быть два публичных поля, X и Y, типа double.

В классе Geometry должно быть два статических метода: GetLength, который возвращает длину переданного вектора, и Add, который возвращает сумму двух переданных векторов.

Оба класса разместите в одном файле. Вообще-то так обычно делать не стоит, но так удобнее для нашей автоматической проверки выполнения задания.

Лабораторная «Отрезок»

Задание

Продолжаем разработку геометрической библиотеки.

Создайте класс `Segment`, представляющий отрезок прямой. Концы его отрезков должны задаваться двумя публичными полями: Begin и End типа Vector.

Добавьте метод `Geometry.GetLength`, вычисляющий длину сегмента, и метод `Geometry.IsVectorInSegment(Vector, Segment)`, проверяющий, что задаваемая вектором точка лежит в отрезке.

Сохраните функциональность предыдущего этапа.

Лабораторная «Нестатические методы»

Задание

Вы вдруг поняли, что не очень-то удобно писать имя класса Geometry при выполнении любой операции с векторами и сегментами. Однако, отказаться от этого класса вы не можете, потому что за те несколько минут, пока вы сдавали предыдущую задачу, вашу библиотеку скачали и начали использовать в своих проектах тысячи человек.

Поэтому вы решили сохранить этот класс, но добавить методы Vector.GetLength(), Segment.GetLength(), Vector.Add(Vector), Vector.Belongs(Segment) и Segment.Contains(Vector) не вместо, а вместе с соответствующими методами класса Geometry.

Сделайте это! Каждый из этих методов должен вызывать уже существующий метод класса Geometry, чтобы не дублировать код.

Вся функциональность предыдущего этапа должна остаться!

Лабораторная «256 оттенков серого»

Задание

Некто хочет использовать вашу геометрическую библиотеку для рисования. Для этого ему необходимо, чтобы у вашего класса Segment появился цвет. Однако, вам кажется, что втаскивать цвета в чисто геометрическую сущность - плохая идея.

Сделайте так, чтобы методы GetColor и SetColor появились в вашем классе Segment.

Если цвет не задан, GetColor возвращает Color.Black.

Тема 18. Сложность алгоритмов. Асимптотический анализ алгоритмов. Функция роста.

Справка

Во многих работах описывающих те или иные алгоритмы, часто можно встретить обозначения типа: $O(g(n))$, $\Omega(g(n))$, $\Theta(g(n))$.

Основные классы сложности применяемые при анализе:

$f(n) = O(1)$ константа

$f(n) = O(\log(n))$ логарифмический рост

$f(n) = O(n)$ линейный рост

$f(n) = O(n*\log(n))$ квазилинейный рост

f(n) = O(n^m) полиномиальный рост

f(n) = O(2^n) экспоненциальный рост

Тема 19. Наследование

Лабораторная «Земля и Диггер»

Задание

Вам предстоит наполнить готовую заготовку игровыми элементами. Каждый элемент должен уметь:

- Возвращать имя файла, в котором лежит соответствующая ему картинка (например, "Terrain.png")
- Сообщать приоритет отрисовки. Чем выше приоритет, тем раньше рисуется соответствующий элемент, это важно для анимации.
- Действовать — возвращать направление перемещения и, если объект во что-то превращается на следующем ходу, то результат превращения.
- Разрешать столкновения двух элементов в одной клетке.

Сделайте класс Terrain, реализовав ICreature. Сделайте так, чтобы он ничего не делал.

Сделайте класс Player, реализовав ICreature.

Сделайте так, чтобы диггер шагал в разные стороны в зависимости от нажатой клавиши со стрелкой (Game.KeyPressed). Убедитесь, что диггер не покидает пределы игрового поля.

Сделайте так, чтобы земля исчезала в тех местах, где прошел диггер.

Запустите проект — игра должна заработать!

В методе Game.CreateMap вы можете менять карту, на которой будет запускаться игра. Используйте эту возможность для отладки.

Лабораторная «Мешки и золото»

Задание

Сделайте класс Sack, реализовав ICreature. Это будет мешок с золотом.

- Мешок может лежать на любой другой сущности (диггер, земля, мешок, золото, край карты).
- Если под мешком находится пустое место, он начинает падать.
- Если мешок падает на диггера, диггер умирает, а мешок продолжает падать, пока не приземлится на землю, другой мешок, золото или край карты.
- Диггер не может подобрать мешок, толкнуть его или пройти по нему.

Если мешок падает, а диггер находится непосредственно под ним и идет вверх, они могут "разминуться", и диггер окажется над мешком. Это поведение непросто исправить в существующей упрощенной архитектуре, поэтому считайте его нормальным.

Сделайте класс Gold, реализовав ICreature.

- Мешок превращается в золото, если он падал дольше одной клетки игрового поля и приземлился на землю, на другой мешок, на золото или на край карты.
- Мешок не превращается в золото, а остаётся мешком, если он падал ровно одну клетку.
- Золото никогда не падает.
- Когда диггер собирает золото, ему начисляется 10 очков (через Game.Scores).

Лабораторная «Монстры»

Задание

Сделайте класс Monster, реализовав ICreature. Его поведение должно быть таким:

- Если на карте нет диггера, монстр стоит на месте.
- Если на карте есть диггер, монстр двигается в его сторону по горизонтали или вертикали. Можете написать поиск кратчайшего пути к диггеру, но это не обязательно.
- Монстр не может ходить сквозь землю или мешки.
- Если после хода монстр и диггер оказались в одной клетке, диггер умирает.
- Если монстр оказывается в клетке с золотом, золото исчезает.
- Мешок может лежать на монстре.
- Падающий на монстра мешок убивает монстра.
- Монстр не должен начинать ходить в клетку, где уже есть другой монстр.
- Если два или более монстров сходили в одну и ту же клетку, они все умирают. Если в этой клетке был диггер — он тоже умирает.

Тема 20. Целостность данных

Лабораторная «Readonly Vector»

Задание

В пространстве имен ReadOnlyVectorTask сделайте класс ReadOnlyVector с двумя публичными readonly-полями X и Y, которые устанавливаются в конструкторе.

ReadOnlyVector должен содержать метод Add(ReadOnlyVector other), который возвращает сумму векторов.

При работе с readonly классами часто хочется изготовить вектор "такой же, но с другим значением поля X или Y". Обеспечьте такую функциональность с помощью методов WithX(double) и WithY(double)

Лабораторная «Счёт из отеля»

Справка

Ситуация, когда свойства вот так взаимозависимы и при этом каждое имеет сеттер, довольно редка. В норме, Total бы имело только геттер. Однако, классы такого вида тоже встречаются, например в качестве класса-модели для пользовательского интерфейса. Библиотека WPF и некоторые другие библиотеки для построения пользовательских интерфейсов дают возможность так называемого двухстороннего связывания контролах окна со свойствами в объекте модели. При изменении значения в контролах меняются значения свойств объекта, и наоборот, при изменении свойств меняются значения в контролах.

Загляните в файлы MainWindow.xaml и MainWindow.cs, вы увидите, что там нет кода для обновления полей, пересчёта значений и выделения поля с ошибочным вводом. Формат Xaml скорее всего вам не знаком, но тем не менее убедиться, что там нет этой логики несложно. Зато можно увидеть как устанавливается связь между полями графического интерфейса и свойствами модели. Такой подход называется View-ViewModel, в котором MainForm — это View, а сам класс AccountingModel — ViewModel.

WPF — библиотека предназначенная только для Windows. Поэтому на не windows компьютерах вам не удастся увидеть, как именно работает MVVM-связка, но вы можете написать требуемый класс, создав новый проект.

Задание

В файле AccountingModel.cs создайте класс AccountingModel, унаследованный от ModelBase, со следующими свойствами.

- double Price — цена за одну ночь. Должна быть неотрицательной.
- int NightsCount — количество ночей. Должно быть положительным.
- double Discount — скидка в процентах. Никаких дополнительных ограничений.
- double Total — сумма счёта. Должна быть не отрицательна и должна быть согласована с предыдущими тремя свойствами по правилу: `Total == Price * NightsCount * (1-Discount/100)`.

Все поля должны иметь сеттеры. При установке Price, NightsCount и Discount должна соответствующим образом устанавливаться Total, при установке Total — соответствующим образом меняться Discount. В случае установки значения, нарушающего хоть одно условие целостности, необходимо выкидывать ArgumentException.

При изменении значения любого свойства необходимо дополнительно сигнализировать об этом с помощью вызова метода Notify, передавая ему имя изменяемого свойства. Здесь можно воспользоваться ключевым словом nameof.

Лабораторная «Карманный гугл»

Справка

- Remove, Insert, Contains, IndexOf в List имеют сложность O(n).
- Add в коллекциях Dictionary и List имеет среднюю сложность O(1).
- Доступ по ключу в Dictonary имеет среднюю сложность O(1).
- Доступ по индексу в List имеет сложность O(1).
- Remove, ContainsKey в Dictionary имеют среднюю сложность O(1).

Задание

В файле Indexer.cs реализуйте предложенные методы

- Add. Этот метод должен индексировать все слова в документе. Разделители слов: { ' ', '.', ',', '!', '?', ':', '-','\r','\n' }; Сложность – O(document.Length)
- GetIds. Этот метод должен искать по слову все id документов, где оно встречается. Сложность — O(result), где result — размер ответа на запрос
- GetPositions. Этот метод по слову и id документа должен искать все позиции, в которых слово начинается. Сложность — O(result)
- Remove. Этот метод должен удалять документ из индекса, после чего слова в нем искаться больше не должны. Сложность — O(document.Length)

Тема 21. Основы ООП. Динамические структуры. Объекты.

Справка

По определению будем называть объектом понятие, абстракцию или любой предмет с четко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы.

Введение объектов преследует две цели:

- понимание прикладной задачи (проблемы);
- введение основы для реализации на компьютере.

Тема 22. Структуры

Лабораторная «Banchmark»

Задание

В файле `BenchmarkTask.cs` реализуйте в классе Benchmark интерфейс IBenchmark.

Напишите в том же файле unit-тест в методе `StringConstructorFasterThanStringBuilder`. В нём нужно сравнить два способа создания строки, состоящей из 10000 букв 'a'

Лабораторная «Эксперименты»

Справка

Избавиться от дублирования в коде BuildChartDataForMethodCall и BuildChartDataForArrayCreation поможет паттерн абстрактная фабрика.

Задание

В файле `ArrayCreationTasks.cs` есть две реализации уже знакомого вам интерфейса ITask для работы с классом Benchmark. Оба класса создают массив в методе Run. Но один делает массив структур, а второй массив классов.

В классе `ExperimentsTask` реализуйте метод `BuildChartDataForArrayCreation`. Этот метод должен измерять длительность работы метода Run у классов `StructArrayCreationTask` и `ClassArrayCreationTask` с помощью Benchmark из прошлого задания.

Нужно измерить время для структур и классов всех размеров, указанных в `Constants.FieldCounts`. Результаты измерения вернуть в виде объекта ChartData. Дальше в Program.cs эти результаты будут показаны на графиках.

Запустите код на исполнение. Вы должны увидеть первый график скорости работы от количества полей в классе/структуре. На нём должно быть видно, что массивы классов создаются дольше, чем массивы структур.

Аналогично в файле `MethodCallTasks.cs` есть ещё пара реализаций ITask. Они вызывают метод, передавая в качестве аргумента класс или структуру с большим количеством полей.

В том же классе `ExperimentsTask` реализуйте метод `BuildChartDataForMethodCall`.

Избавьтесь от дублирования кода в методах `BuildChartDataForMethodCall` и `BuildChartDataForArrayCreation`. Возможно, для этого понадобится создать новые классы.

Запустите код на исполнение. Вы должны увидеть второй график, показывающий, что большие классы передаются в метод быстрее, чем большие структуры.

Попробуйте объяснить наблюдаемый результат.

Тема 23. Бинарная логика. Двоичное представление данных

Справка

Перевод чисел из произвольной системы счисления в десятичную

Для перевода числа из любой позиционной системы счисления в десятичную необходимо использовать развернутую форму числа, заменяя, если это необходимо, буквенные обозначения соответствующими цифрами.

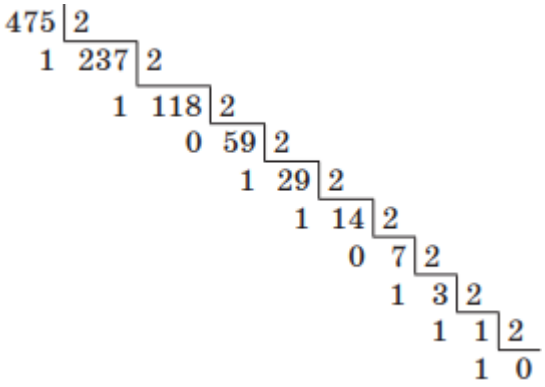
Например:

$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13_{10}$;

Перевод чисел из десятичной системы счисления в заданную

Для преобразования целого числа десятичной системы счисления в число любой другой системы счисления последовательно выполняют деление нацело на основание системы счисления, пока не получат нуль. Числа, которые возникают как остаток от деления на основание системы, представляют собой последовательную запись разрядов числа в выбранной системе счисления от младшего разряда к старшему. Поэтому для записи самого числа остатки от деления записывают в обратном порядке.

Например, переведем десятичное число 475 в двоичную систему счисления. Для этого будем последовательно выполнять деление нацело на основание новой системы счисления, т. е. на 2:



Читая остатки от деления снизу-вверх, получим 111011011.

Тема 24. Очереди, стеки, дженерики

Лабораторная «Limited Size Stack»

Справка

У каждой коллекции в C# доступен метод расширения Last(). Однако, работает он за O(1) только для коллекций, реализующих интерфейс IList (список с доступом к элементам по индексу). Для остальных коллекций он работает за O(N), перебирая её элементы до конца.

Задание

В этой задаче вам нужно реализовать стек ограниченного размера. Этот стек работает как обычный стек, однако при превышении максимального размера удаляет самый глубокий элемент в стеке. Таким образом в стеке всегда будет ограниченное число элементов.

Лабораторная «Отмена»

Справка

В `LimitedSizeStack` нужно хранить какие-то данные, описывающие выполненное действие. В них должно быть достаточно информации для отмены действия.

Задание

Если вы запустите проект на исполнение, то увидите окно приложения, в котором можно добавлять новые дела и удалять уже существующие. Однако кнопка "Отмена" пока не работает. Ваша задача — сделать так, чтобы эта кнопка отменяла последнее действие пользователя.

Изучите класс `ListModel` — в нём реализована логика работы кнопок в приложении.

Реализуйте методы `Undo` и `CanUndo`. Для этого нужно хранить историю последних `limit` действий удаления/добавления. Используйте для этого класс `LimitedSizeStack` из прошлой задачи. Его не нужно включать в отправляемый на проверку файл, считайте, что этот класс уже есть в проекте.

- Метод `Undo` отменяет последнее действие из истории.
- Метод `CanUndo` возвращает `true`, если на данный момент история действий не пуста, то есть если вызов `Undo` будет корректным. Иначе метод должен вернуть `false`.

Лабораторная CVS

Справка

Состояние каждого клона можно описать двумя стеками: стек усвоенных программ и стек отмененных программ. Все описанные в задаче операции — работают с этими стеками.

Вам будет удобнее, если вы введете отдельный класс для описания клона, в котором будете хранить стеки. Тогда операции удобно будет сделать методами этого же класса.

Для эффективной работы операции клонирования нужно использовать стек, реализованный на связных списках. Тогда операцию Clone можно будет реализовать за $\Theta(1)$.

Если вы используете стандартный класс Stack (реализованный на массивах), ожидайте получение MemoryLimit.

Задание

В классе CloneVersionSystem реализуйте метод Execute, принимающий на вход описание команды в виде строки и возвращающий результат в виде строки.

Поддерживаемые команды:

- learn ci pi. Обучить клона с номером ci по программе pi.
- rollback ci. Откатить последнюю программу у клона с номером ci.
- relearn ci. Переусвоить последний откат у клона с номером ci.
- clone ci. Клонировать клона с номером ci.
- check ci. Вернуть программу, которой клон с номером ci владеет и при этом усвоил последней. Если клон владеет только базовыми знаниями, верните "basic".

Выполнение команды check должно возвращать имя программы. Выполнение остальных команд должно возвращать null.

Все команды корректны, в частности, к клону, уже владеющему некоторой программой, learn по ней же применяться не будет. К клону, не владеющему ни одной программой, не применяется rollback. А также relearn возможен только при непустой истории откатов. В запросах может фигурировать только уже существующий клон. Номера клонам присваиваются в порядке их возникновения. Клон, с которого каминуане начали свои эксперименты, имел номер один.

Тема 25. Интерфейсы перечисления. Оператор yield.

Лабораторная «Экспоненциальное сглаживание»

Справка

В этом модуле вы изучили интерфейс IEnumerator. Однако использовать напрямую методы этого интерфейса в своём коде как правило не нужно. Для перечисления элементов есть оператор foreach и методы LINQ. В отличие от интерфейса IEnumerator, они просты в использовании и легко читаются. Используйте работу через методы IEnumerator только, если задача не решается с помощью foreach или уже готовых методов LINQ.

В частности, вся серия задач данного модуля решается без использования методов IEnumerator.

Задание

В классе ExpSmoothingTask реализуйте функцию экспоненциального сглаживания данных.

Отладьте реализацию с помощью приложенных модульных тестов. Запустите тестирующее приложение и объясните наблюдаемый результат.

Лабораторная «Скользящее среднее»

Справка

Для этой задачи вам пригодится структура данных Очередь. Не нужно создавать её самостоятельно, воспользуйтесь готовым классом Queue в пространстве имён System.Collections.Generic.

Задание

В классе MovingAverageTask реализуйте функцию скользящего среднего.

При усреднении с окном размера W, первые W-1 точки результата в действительности должны усредняться по окнам меньшего размера. Так, первая точка должна попасть в результат без изменения. Отладьте реализацию с помощью приложенных модульных тестов.

Лабораторная «Скользящий максимум»

Справка

Итак, давайте, как и в прошлой задаче `MovingAverageTask` хранить элементы текущего окна.

Мы не хотим пересчитывать максимум заново на каждой точке — это даст сложность обработки O(WindowSize), что слишком медленно по условию. Поэтому используем вспомогательную структуру данных, которая поможет это делать быстрее.

Будем отдельно хранить список только тех значений окна, которые потенциально могут стать максимумом в будущем. Значение не может стать максимумом, если после него в окно попало хоть одно значение больше него. Поэтому перед добавлением очередного элемента в этот список (будем считать, что добавляем мы справа) нужно удалить справа все элементы, меньшие нового. Несложно понять, что этот список будет упорядоченным, а значит максимум всех чисел в текущем окне будет в этом списке самым левым.

Для хранения этого списка потенциальных максимумов пригодится структура данных Deque (LinkedList в языке C#), в которой эффективно добавлять и удалять элементы можно с обоих концов списка.

Задание

В классе MovingMaxTask реализуйте функцию максимума в скользящем окне. Для каждой точки найдите максимум всех предшествующих точек в окне указанного размера. Алгоритм должен работать эффективно, то есть тратить на обработку одной точки в среднем O(1) времени, вне зависимости от размера окна.

Тема 26. Бинарная логика. Бинарные операции. Область применения.

Справка

Логическая операция — операция над выражениями логического (булевского) типа, соответствующая некоторой операции над высказываниями в алгебре логики.

Логические выражения могут принимать одно из двух значений — «истинно» или «ложно».

Логические операции служат для получения сложных логических выражений из более простых. В свою очередь, логические выражения обычно используются как условия для управления последовательностью выполнения программы.

Тема 27. Листы и словари

Лабораторная «Readonly bytes»

Справка

Иногда есть смысл в качестве ключей в Dictionary или HashSet использовать массивы байт. Однако по умолчанию массивы сравниваются по ссылкам, а не по содержимому, а часто нужно именно по содержимому. В таких случаях можно написать класс-обёртку над массивом, который переопределит Equals и HashCode так, чтобы сравнение происходило по содержимому. В этой задаче вам нужно создать именно такую обёртку.

Задание

В файле ReadonlyBytes.cs создайте класс ReadonlyBytes так, чтобы все тесты из файла ReadonlyBytesTests.cs компилировались и проходили.

Лабораторная «Ghosts»

Справка

Вспомните устройство Dictionary. HashSet реализован аналогично.

Достаточно сделать так, чтобы GetHashCode до и после вызова DoMagic возвращал разные значения

Задание

Неаккуратная реализация Equals и GetHashCode может приводить к тому, что добавленный в Dictionary или HashSet ключ внезапно исчезает. Чтобы не попадаться на подобные ошибки в будущем, в этом задании предлагается поизучать всевозможные подобные ошибки.

В проекте вам даны несколько классов с уже реализованными GetHashCode и Equals. Вам нужно придумать, как их использовать, чтобы HashSet стал вести себя некорректно.

Изучите тест GhostsTest.cs и в файле GhostsTask.cs создайте класс GhostsTask так, чтобы этот тест проходил.

Тема 28. Делегаты

Справка

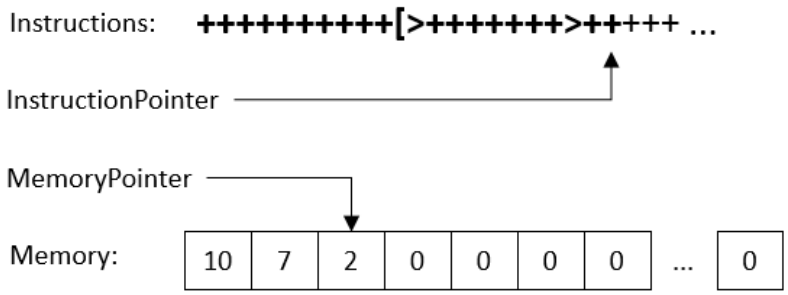
Создавать языки программирования сложно. Но не всегда! Язык программирования Brainfuck — это экстремально простой язык программирования, интерпретатор при желании можно уместить на один экран кода. Программа на Brainfuck состоит только из символов +-<>.,[], поэтому читать такие программы не очень удобно. :-)

В этой серии задач вам предстоит создать этот интерпретатор с возможностью его простого расширения новыми операциями.

В задачах программирования интерпретаторов часто оказываются удобными пройденные в этом блоке делегаты анонимные функции — решите эту задачу с помощью анонимных функций.

Лабораторная «Виртуальная машина Brainfuck»

Справка



Виртуальная машина хранит следующее:

- Массив памяти, каждая ячейка которого хранит 1 байт. По умолчанию размер памяти — 30000 ячеек.
- Указатель на текущую ячейку памяти. Изначально, указатель указывает на нулевую ячейку.
- Выполняемую программу. Она состоит из инструкций, каждая обозначается одним символом. Программа начинает выполняться с первого символа последовательно.
- Номер выполняемой в данный момент инструкции. После выполнения любой инструкции номер увеличивается на единицу. Как только номер инструкции выходит за пределы программы, выполнение заканчивается.

Задание

В этой части вам нужно реализовать виртуальную машину в классе VirtualMachine.cs так, чтобы проходили все тесты из файла VirtualMachineTests.cs.

Лабораторная «Простые команды Brainfuck»

Задание

Изучите класс Brainfuck.cs, в частности то, как он использует реализованный ранее класс VirtualMachine.

В классе BrainfuckBasicCommands реализуйте метод, регистрирующий следующие простые команды в виртуальную машину:

Символ	Значение
.	Вывести байт памяти, на который указывает указатель, преобразовав в символ согласно ASCII
+	Увеличить байт памяти, на который указывает указатель
-	Уменьшить байт памяти, на который указывает указатель
,	Ввести символ и сохранить его ASCII-код в байт памяти, на который указывает указатель
>	Сдвинуть указатель памяти вправо на 1 байт
<	Сдвинуть указатель памяти влево на 1 байт
A-Z, a-z, 0-9	сохранить ASCII-код этого символа в байт памяти, на который указывает указатель

Например, программа `++>+++.<.` выводит два символа с ASCII кодами 2 и 3, а память после выполнения команды будет выглядеть так [2, 3, 0, 0, ... 0].

Для ввода и вывода используйте переданные в метод Run функции `Func<int> read` и `Action<char> write`.

Тут `read` по аналогии с `Console.Read` возвращает либо код введенного символа, либо -1, если ввод закончился. Считайте, что на вход будут подаваться только символы с кодами 0..255 — они точно помещаются в один байт.

Лабораторная «Циклы Brainfuck»

Задание

В классе BrainfuckLoopCommands реализуйте метод, регистрирующий следующие команды в виртуальную машину:

Символ	Значение
[	(Начало цикла) Перескочить по программе вправо на соответствующий (с учетом вложенности) символ `]`, если текущий байт памяти равен нулю. Продолжать исполнение с этого символа.
]	(Конец цикла) Перескочить по списку инструкций влево на соответствующий (с учетом вложенности) символ `[`, если текущий байт памяти НЕ равен нулю. Продолжать исполнение с этого символа.

Например, программа `++++++++[>++++++++<-]>+.` выводит букву `A` (ASCII-код 65 получается увеличением 8 раз второй ячейки на 8, а потом добавлением ещё единицы).

Тема 29. Бинарная логика. Обработка данных в двоичном виде.

Справка

& (логическое умножение)

Умножение производится поразрядно, и если у обоих операндов значения разрядов равно 1, то операция возвращает 1, иначе возвращается число 0. Например:

| (логическое сложение)

Похоже на логическое умножение, операция также производится по двоичным разрядам, но теперь возвращается единица, если хотя бы у одного числа в данном разряде имеется единица.

^ (логическое исключающее ИЛИ)

Также эту операцию называют XOR, нередко ее применяют для простого шифрования:

Здесь опять же производятся поразрядные операции. Если у нас значения текущего разряда у обоих чисел разные, то возвращается 1, иначе возвращается 0. Таким образом, мы получаем из 9^5 в качестве результата число 12. И чтобы расшифровать число, мы применяем ту же операцию к результату.

~ (логическое отрицание или инверсия)

Еще одна поразрядная операция, которая инвертирует все разряды: если значение разряда равно 1, то оно становится равным нулю, и наоборот.

Операции сдвига также производятся над разрядами чисел. Сдвиг может происходить вправо и влево.

- **`x<<y`** - сдвигает число x влево на y разрядов. Например, `4<<1` сдвигает число 4 (которое в двоичном представлении 100) на один разряд влево, то есть в итоге получается 1000 или число 8 в десятичном представлении.
- **`x>>y`** - сдвигает число x вправо на y разрядов. Например, `16>>1` сдвигает число 16 (которое в двоичном представлении 10000) на один разряд вправо, то есть в итоге получается 1000 или число 8 в десятичном представлении.

Тема 30. Элементы функционального программирования

Лабораторная «Лямбды и делегаты»

Справка

Методы должны возвращать лямбды. Изучите типы делегатов, чтобы понять как должны выглядеть эти лямбды.

Задание

В этой задаче в классе ForcesTask нужно реализовать три вспомогательных метода, преобразующих одни делегаты в другие.

Чтобы лучше понимать зачем эти методы нужны, изучите проект, в частности места использования этих методов. Это будет полезно и для следующих заданий.

После выполнения этого задания, при запуске проекта Каракуля должен летать и управляться клавишами A и D.

Лабораторная «Уровни»

Задание

В этой задаче в классе LevelsTask нужно добавить в игру ещё несколько уровней.
В результате должны быть следующие уровни:

1. Zero. Нулевая гравитация. Начальное положение ракеты и положение цели см. в коде. Если не указано иное, на других уровнях начальное положение цели и ракеты такое же.
2. Heavy. Постоянная гравитация 0.9, направленная вниз.

3. Up. Гравитация направлена вверх и значение её модуля вычисляется по формуле 300 / (d + 300.0), где d — это расстояние от нижнего края пространства. Цель должна иметь координаты (X:700, Y:500)
4. WhiteHole. Гравитация направлена от цели. Модуль вектора гравитации вычисляется по формуле 140*d / (d^2+1), где d — расстояние до цели.
5. BlackHole. В середине отрезка, соединяющего начальное положение ракеты и цель, находится аномалия. Гравитация направлена к аномалии. Модуль вектора гравитации равен 300*d / (d^2+1), где d — расстояние до аномалии.
6. BlackAndWhite. Гравитация равна среднему арифметическому гравитаций на уровнях WhiteHole и BlackHole.

Все уровни должны удовлетворять таким дополнительным условиям:

1. Расстояние от начального положения ракеты до цели должно быть в пределах от 450 до 550.
2. Угол между направлением на цель и начальным направлением ракеты должен быть не менее PI/4.

Постарайтесь избежать дублирования кода в этой задаче.

Лабораторная «Управление»

Задание

В этой задаче в классе ControlTask нужно реализовать метод управления ракетой. В результате ракета должна достигать цели в уровнях Zero, Heavy, Up и WhiteHole.

Тема 31. LINQ

Лабораторная «Median & Bigrams»

Задание

В файле ExtensionsTask реализуйте два метода расширения: для вычисления медианы и для вычисления списка биграмм.

Эти методы пригодятся в будущем. Вы сможете их использовать на ряду и в перемешку с остальными методами LINQ.

Есть важное замечание по деталям реализации.

Создавая методы, работающие с `IEnumerable` стоит придерживаться следующих рекомендаций:

1. Если это возможно, не перечисляйте входной IEnumerable до конца. Потому что IEnumerable может теоретически быть бесконечным.
2. Не перечисляйте больше элементов, чем нужно для работы IEnumerable. Возможно, при перечислении лишнего элемента случится ошибка или другой нежелательный побочный эффект.
3. Не полагайтесь на то, что `IEnumerable` можно будет перечислить дважды. Этого никто не гарантирует. Кстати, некоторые IDE, автоматически находят нарушение этого пункта. Например, подобные предупреждения умеют показывать JetBrains Rider и Visual Studio с установленным Resharper.

Лабораторная «Чтение файла»

Справка

Не используйте циклы в решении. Вместо этого используйте LINQ.

Обратите внимание, что в разных методах предлагается реализовать разную реакцию на некорректные строки файлов: в одном случае — игнорировать их, а в другом

— выбрасывать исключение на первой же ошибочной строке. Это сделано исключительно в учебных целях — в реальных проектах стоит, конечно, придерживаться какой-то одной выбранной стратегии.

Задание

Исходные данные содержатся в двух файлах:

1. slide.txt содержит информацию про каждый из слайдов — идентификатор, тип слайда (теория, задача или тест), и тема соответствующей недели. Пример файла slides.txt:

```
SlideId;SlideType;UnitTitle
0;theory;Первое знакомство с C#
1;quiz;Первое знакомство с C#
2;theory;Первое знакомство с C#
      3;exercise;Первое знакомство с C#
```

2. visits.txt содержит по одной записи на первое посещение слайда каждым пользователем. Запись состоит из идентификатора пользователя, идентификатора слайда, даты и времени посещения этим пользователем этого слайда. Пример файла visits.txt:

```
UserId;SlideId;Date;Time
0;5;2014-09-03;12:20:28
1;6;2014-09-03;12:25:09
1;4;2014-09-03;12:25:24
```

В этой задаче в классе ParsingTask нужно реализовать методы чтения этих файлов.

Лабораторная «Статистика»

Задание

В файле StatisticsTask реализуйте метод GetMedianTimePerSlide. Он должен работать так.

Обозначим T(U, S) время между посещением пользователем U слайда S и ближайшим следующим посещением тем же пользователем U какого-то другого слайда S2 != S.

T(U, S) можно считать примерной оценкой того, сколько времени пользователь U провел на слайде S.

Метод должен для указанного типа слайда, считать медиану значений T(U, S) по всем пользователям и всем слайдам этого типа.

Нужно игнорировать значения меньшие 1 минуты и большие 2 часов при расчете медианы.

Гарантируется, что в тестах и реальных данных отсутствуют записи, когда определенный пользователь заходит на один и тот же слайд более одного раза.

Время нужно возвращать в минутах.

Воспользуйтесь реализованными ранее методами Bigrams и Median.

Лабораторная «Метод Гауса»

Справка

Алгоритм Гаусса — это алгоритм решения системы линейных уравнений. Ниже описаны основные этапы работы этого алгоритма.

1. Формируем матрицу из коэффициентов при неизвестных переменных и свободных членов. Ниже на иллюстрации, свободные члены в матрице отделены линией для удобства.
2. По очереди рассматриваем все столбцы слева направо
    1. Ищем ещё не использованную строку матрицы, в которой в нашем столбце находится ненулевой элемент. Помечаем ее, как использованную. Позже мы с помощью этой строки найдем значение переменной, соответствующей текущему столбцу.
    2. Если такой строки не нашлось, переходим к следующему столбцу. В этом случае неизвестная переменная, соответствующая этому столбцу может принимать любое значение, например, 0.
    3. Используем эту строку, чтобы получить нули во всех остальных строках в текущем столбце. Для этого складываем найденную строку с остальными строками, умножая её на подходящее число.
    4. После выполнения этого шага в текущем столбце должно остаться не более одного ненулевого значения.
3. После того, как все столбцы рассмотрены, матрица должна иметь удобный вид, для того, чтобы вычислить значение всех неизвестных переменных.

Задание

Вам необходимо в методе `Solver.Solve` реализовать алгоритм решения системы линейных уравнений. Система передается методу Solve в виде матрицы коэффициентов перед неизвестными и в виде массива свободных членов. Возвращать Solve должен любое из возможных решений в виде массива значений неизвестных.

Для решения этой задачи используйте алгоритм Гаусса. Постарайтесь использовать методы LINQ для упрощения вашего кода.

Вам доступны модульные тесты для проверки своего решения. При наличии ошибок, начинайте отладку с самого простого непройденного теста.

Тема 32. Наследование, дженерики и порождение типов

Справка

Универсальные шаблоны вводят на платформе .NET концепцию параметров универсального типа. Благодаря им вы можете создавать классы и методы с типами, спецификация которых отложена до момента объявления и создания экземпляров в клиентском коде.

Тема 33. Графы и обходы

Лабораторная «Поиск в ширину»

Справка

Начните поиск в ширину из точки start и продолжайте, пока очередь поиска в ширину не опустеет.

Чтобы быстро проверять, содержит ли клетка лабиринта сундук, можно воспользоваться уже знакомой структурой данных Dictionary или изучить более подходящую для данной задачи структуру данных HashSet.

Реализация поиска в ширину немного упростится, если хранить в очереди не просто координаты точки, а объект SinglyLinkedList, сохраняющий ещё ссылку на предыдущую точку пути.

Односвязные списки путей до разных сундуков могут иметь общие «хвосты». Примерно так же как было в задаче CVS в одной из предыдущих недель.

Возвращать пути до всех достижимых сундуков можно «лениво» с помощью оператора yield return

Задание

На карте расположено несколько сундуков. Для тех сундуков, до которых существует путь от точки start, необходимо найти путь от сундука до точки start в виде односвязного списка `SinglyLinkedList`.

Для этого в классе `BfsTask` нужно реализовать поиск в ширину с указанной сигнатурой. Кстати, он вам понадобится и для следующей задачи!

Проверить корректность своего решения можно запустив тесты в классе `Bfs_Should`. Там же, по тестам, можно уточнить постановку задачи на различных крайних случаях.

После корректного выполнения задания, можно будет запустить проект. Кликнув на пустую ячейку, вы увидите найденный вашим алгоритмом путь.

Лабораторная «Вынеси клад!»

Справка

- Придумайте, как можно наиболее экономно свести эту задачу к поиску кратчайших путей
- Можно обойтись всего двумя поисками в ширину — одним от стартовой точки и ещё одним от выхода
- Не забудьте о декомпозиции — разбейте ваше решение на логические блоки и каждый поместите в свой метод с понятными именами!
- Объединить результаты двух поисков в ширину сделать проще простого с помощью Linq-метода Join. Изучите его самостоятельно — это будет полезно!
- Reverse — это одновременно метод класса List, а также Linq-метод расширение. Но в первом случае он модифицирует список и не имеет возвращаемого значения, а во втором ничего не модифицирует, а возвращает новую перевёрнутую коллекцию. Смотрите, не запутайтесь!
- Преобразовать список точек в список MoveDirection можно без циклов с помощью Linq-методов Zip и Skip

Задание

Подготовка закончилась, и вы в настоящем лабиринте с сокровищами! Сил хватит только на один сундук и то еле-еле. Найдите кратчайший путь из начальной точки до выхода, проходящий через хотя бы один сундук.

Решайте задачу в классе `DungeonTask`.

Детали реализации для граничных случаев можно найти в классе с тестами `Dungeon_Should`. Сделайте так, чтобы все тесты проходили.

После выполнения этого задания, при запуске проекта можно увидеть визуализацию пути. Наслаждайтесь найденными сокровищами!

Эту задачу можно элегантно решить без циклов, используя LINQ.

Лабораторная «Поделить территорию»

Справка

- Подумайте, как эффективно решить эту задачу с помощью поиска в ширину
- Охотников за сокровищами может быть много. Очень много!
- Поиск в ширину можно начинать сразу из нескольких стартовых точек

Задание

Оказалось, что в лабиринте есть и другие охотники за сокровищами. Естественно, кто первый доберется до сундука, тот его и заберёт себе.

Неплохо бы знать, кто из соперников до каких клеток лабиринта успеет добраться быстрее других.

В классе RivalsTask реализуйте функцию разделяющую карту между игроками.

Нужно определить, до каких из клеток карты каждый игрок сможет дойти быстрее, вне зависимости от тактики остальных. Ходят игроки по очереди, начиная с первого.

Сделайте так, чтобы все тесты в классе Rivals_Should проходили.

После выполнения этого задания, при запуске проекта можно увидеть визуализацию процесса раздела карты.

Тема 34. Жадные алгоритмы

Лабораторная «Путь в лабиринте»

Справка

Описание лабиринта передаётся в метод в объекте типа `State`:

- в поле `CellCost` находится двумерный массив трудностей всех клеток (0 означает стену).
- есть методы, которые помогут проверить, что какая-то клетка является стеной или находится внутри лабиринта.
- там же есть свойство `Chests`, но в данной задаче вместо него используйте список целей, переданный в метод аргументом.

Все тесты в классе `DijkstraPathFinder_Should` должны завершиться успехом.

Задание

Для того, чтобы сдать задачу, в файле `DijkstraPathFinder.cs` реализуйте метод `GetPathsByDijkstra`. Ему на вход поступают: лабиринт, начальная позиция Жадины, список целей — клеток, до которых нужно найти кратчайшие пути. Он должен возвращать пути до всех целей в виде IEnumerable в порядке увеличения трудности пути до них. При этом вычислять пути он должен лениво, то есть не вычислять пути до далёких сундуков и не обрабатывать весь лабиринт, пока это не запросили из IEnumerable.

Лабораторная «Жадина в лабиринте»

Справка

Используйте класс `DijkstraPathFinder`, реализованный в предыдущей задаче. Его не нужно включать в отправляемый на проверку файл, считайте, что этот класс уже есть в проекте. При проверке этой задачи будет использоваться авторская реализация `DijkstraPathFinder`, а не ваша.

Гарантируется, что если рассмотреть множество всех сундуков и добавить в него исходную позицию, то в нём не существует тройки A, B, C, такой, что от A добраться до B

так же трудно, как и от A до C. Другими словами, у Жадины всегда есть только один вариант дальнейших действий.

Тесты в классах `GreedyPathFinder_Should` и `GreedyTimeLimit_Tests` должны завершаться успехом.

Задание

Для того, чтобы сдать задачу, в файле `GreedyPathFinder.cs` реализуйте метод `FindPathToCompleteGoal`. Он должен возвращать путь передвижения Жадины. Путь не должен содержать исходную позицию — ту из которой Жадина начинает движение. Если подходящего пути не существует, метод должен возвращать пустой список.

Текущее состояние уровня передается в метод в объекте типа `State`.

Лабораторная «Оптимальный маршрут»

Справка

Используйте класс `DijkstraPathFinder`, реализованный в предыдущих задачах.

Задание

Реализуйте метод `FindPathToCompleteGoal` в классе `NotGreedyPathFinder` так, чтобы тесты в классе `NotGreedyPathFinder_Should` завершались успешно.

Тема 35. Динамическое программирование

Лабораторная «Антиплагиат»

Справка

- Напишите вспомогательный метод вычисления расстояния между двумя документами. За основу можно взять каркас решения из упражнения «Расстояние Левенштейна»
- Можно радикально сократить количество используемой памяти, если не хранить двумерный массив. Достаточно хранить только две строки массива динпрога: ту, что сейчас заполняем и предыдущую.
- Совпадающие токены — это частный случай замены одного токена на другой. Используйте это знание, чтобы упростить код.

Задание

В этой задаче вам необходимо реализовать класс `LevenshteinCalculator`, который получает на вход список документов и возвращает список попарных сравнений каждого документа с каждым другим.

Мы хотим, чтобы разница в пробелах, пустых строках или небольшом переименовании переменных не сбивала наш алгоритм. Поэтому вам нужно реализовать модифицированный алгоритм Левенштейна:

1. Он должен анализировать не последовательности символов, а последовательности **токенов** — лексических единиц. Например, в коде `force = mass * acceleration` 5 токенов: `force`, `=`, `mass`, `*`, `acceleration`. Код разбиения на токены уже реализован и на вход вашему алгоритму поступает список токенов. Один документ представляется типом `DocumentTokens` (который объявлен, как синоним `List<string>`).

2. Если два токена различаются, то будем учитывать ещё степень различия. Стоимость замены одного токена на другой в алгоритме Левенштейна будем вычислять с помощью формулы коэффициента Жаккара. Она тоже реализована за вас в методе `GetTokenDistance` класса `TokenDistanceCalculator`. Стоимость удаления/добавления токена равна единице, как и в оригинальном алгоритме.

Лабораторная «Diff Tool»

Справка

Поиск длины наибольшей общей подпоследовательности — ещё одна классическая задача динамического программирования. Она решается, как и расстояние Левенштейна, с помощью построчного заполнения двумерного массива, назовём его `opt`, с таким инвариантом:

`opt[i1, i2]` — это длина наибольшей общей подпоследовательности префикса первого документа длины `i1` и префикса второго документа длины `i2`.

В частности `opt[0, 0] = 0` (оба префикса пустые). А `opt[first.Length, second.Length]` — это длина искомой подпоследовательности для документов `first` и `second`. Начните с того, что напишите формулу для расчёта значения очередной ячейки `opt[i1, i2]` через уже известные ячейки `opt`.

Саму подпоследовательность можно найти по заполненной матрице, начав с `opt[first.Length, second.Length]`.

Задание

Алгоритм из предыдущей задачи находит похожие пары документов. В этой задаче вам предстоит проанализировать два документа и найти в них повторяющиеся части. Подобную задачу решают так называемые Diff Tools — инструменты для сравнения текстовых файлов.

Для этого вам нужен алгоритм, который будет по двум последовательностям токенов возвращать их наибольшую общую подпоследовательность. Она должна состоять из токенов первой последовательности, которые в том же порядке присутствуют и во второй последовательности (токены не обязательно должны идти подряд). И из всех таких последовательностей вернуть нужно самую длинную. Если самых длинных несколько, можно вернуть любую.

Например, у документов `a1 b2 ab1 b21 b2` и `b2 b2 a1` (токены разделены пробелом) наибольшая общая подпоследовательность — это `b2 b2` и имеет длину 2 токена.

Реализуйте это в методе `Calculate` в классе `LongestCommonSubsequenceCalculator` и отладьте реализацию на тестах `LongestCommonSubsequenceCalculator_Tests`. Как и в прошлой задаче документы приходят в ваш метод уже разбитые на токены, вам этого делать не нужно.

Лабораторная «Счастливые билеты»

Задание

Необходимо посчитать количество «счастливых» билетов с заданной суммой цифр, среди тех, номер которых состоит из 2N разрядов. «Счастливым» является билет, у которого сумма первых N цифр равна сумме N последних цифр.

Тема 36. Структуры данных

Лабораторная «Add и Contains»

Справка

Эти два метода должны быть реализованы без использования рекурсии.

Задание

В этой и следующей задаче нужно будет реализовать структуру данных «Бинарное дерево поиска».

Лабораторная «Enumerable и Индексатор»

Справка

Для решения этой задачи для каждого узла дерева вам придется хранить и поддерживать еще и размер его поддерева.

Для простоты, обе операции реализуйте с использованием рекурсии.

Задание

Реализуйте интерфейс `IEnumerable<T>` в вашем дереве так, чтобы перечисление элементов происходило в порядке их возрастания и имело сложность O(n)O(n)O(n), где nnn — количество элементов в дереве.

Реализуйте индексатор `T this[int i]` у дерева, возвращающий i-ый по порядку (в порядке возрастания) элемент, содержащийся в дереве.

Сложность этой операции должна быть O(h)O(h)O(h), где hhh — высота дерева.

Лабораторная «Disk Tree»

Справка

- Директории имеют древовидную структуру. Храните каждую директорию, как вершину дерева.
- Создайте фиктивную вершину, которая будет соответствовать корневой директории.
- Методы, работающие с директориями удобно перенести в класс этих директорий.
- Для сравнения строк в лексикографическом порядке используйте StringComparer.Ordinal

Задание

На вход ваша программа принимает список строк, соответствующих полным путям директорий. Каждый путь не содержит пробелов, не превосходит по размеру 80 символов, содержится в списке единственный раз и состоит из имени директорий, разделенных символом \.

На выход ваша программа должна возвращать список строк, соответствующих отформатированному дереву каталогов. Перед каждым именем каталога должны стоять пробелы, которые соответствуют степени вложенности каталога. Подкаталоги должны быть перечислены после имени каталога в лексикографическом порядке, и их имена должно предварять на один пробел больше, чем имя родительского каталога. Перед именами каталогов верхнего уровня не должно быть пробелов, а сами они должны быть перечислены в лексикографическом порядке. Изучите тесты для лучшего понимания требуемого формата.

Тема 37. События

Справка

События позволяют классу или объекту уведомлять другие классы или объекты о возникновении каких-либо ситуаций. Класс, отправляющий (или порождающий) событие, называется издателем , а классы, принимающие (или обрабатывающие) событие, называются подписчиками.

Тема 38. Оконные приложения

Задание

1. В каждой команде ровно 2 человека (если вам не сказал об ином преподаватель). В группе с нечетным количеством человек может быть одна команда с тремя участниками.
2. Можно использовать либо Windows Forms, либо Unity.
    - Unity доступен, только если разрешил ваш преподаватель практики. Преподаватели знают Windows Forms. Не все преподаватели знают Unity и смогут помочь вам в случае затруднений.
    - Порог входа в Unity выше, чем в Windows Forms, так что на его изучение у вас уйдет больше времени. Выбирайте Unity, если у вас есть лишнее свободное время, готовность самостоятельно разбираться и вы хотите освоить профессиональный инструмент для разработки игр. Выбирайте Windows Forms, если вы хотите потренироваться продумывать структуру крупного проекта на чистом C#.
3. Это должна быть 2d игра.
4. Можно использовать картинки из отрытых источников или рисовать свои.
5. При реализации игры требуется использовать нетривиальный алгоритм. Алгоритмы вы пройдете в темах «Графы и обходы», «Жадные алгоритмы», «Динамическое программирование», «Структуры данных».
6. В предыдущей части курса мы писали игру Диггер. Поэтому вариации на тему Диггера запрещены.
7. Нельзя реализовать существующую игру с классическими правилами (шашки, покер, pac-man). Ваша игра должна быть в значительной мере уникальна.
8. За 5 минут игры игрок должен столкнуться со всеми важными фишками игры. Потому что время на демонстрацию игры ограничено.

Тема 39. Программирование по контракту

Справка

Контрактное программирование — это метод проектирования программного обеспечения. Он предполагает, что проектировщик должен определить формальные, точные и верифицируемые спецификации интерфейсов для компонентов системы. При этом, кроме обычного определения абстрактных типов данных, также используются предусловия, постусловия и инварианты

Тема 40. Многопоточное программирование

Лабораторная «Поток для AI»

Задание

В файле `Channel.cs` вам нужно реализовать все методы в соответствии с их описанием.
Класс `Channel` должен быть потокобезопасным. То есть все его методы должны корректно работать, если их вызывают одновременно из нескольких разных потоков.

Отладьте свою реализацию на тестах `ChannelTests.cs`. После этого, при запуске приложения, Каракуля будет обходить флаги в нужном порядке.

Лабораторная «Параллельный AI»

Справка

Для того, чтобы повысить производительность, можно распараллелить поиск очередного хода. Например, в два потока можно искать ход следующим образом: половину последовательностей перебирать в одном потоке, половину — в другом, а затем из двух полученных последовательностей ходов выбрать лучшую. Тогда время на вычисление очередного хода у вас сократится в два раза.

Если посмотреть на конструктор класса `Bot`, можно заметить, что в нём никак не используется параметр `threadsCount` — количество потоков, в которое будет вычисляться очередной ход. Перепишите метод `GetNextMove` (он находится в отдельном файле `Bot_Parallel.cs`) так, чтобы он искал следующий ход в указанное количество потоков. Обратите внимание, что класс `Random` не является потокобезопасным. То есть, у каждого потока должен быть свой объект этого класса.

Задание

В этой задаче вам нужно распараллелить искусственный интеллект в несколько потоков. Изучите класс `Bot`. Метод `SearchBestMove` ищет оптимальный следующий ход для ракеты: повернуть влево, вправо или продолжить лететь прямо. Он делает это, перебирая заданное количество случайных последовательностей ходов, и выбирает первый ход той последовательности, которая дала наилучший результат, т.е. собрано больше всего флажков и вы находитесь ближе всего к следующему флажку. Понятно, что чем больше бот успеет перебрать случайных последовательностей ходов, тем больше у него будет шансов найти решение лучше. Однако, вместе с качеством увеличится и время вычисления очередного хода.

Тема 41. Рефлексия типов

Лабораторная «Документация»

Справка

Если документация хранится отдельно от кода, она очень легко и быстро устаревает и становится неактуальной. Один из способов сохранять документацию в актуальном состоянии — это писать её максимально близко к коду. Иногда для этого используют атрибуты, информацию из которых извлекают и собирают с помощью рефлексии.

Задание

В файле Specifier реализуйте методы, возвращающие структурированное описание методов класса, ориентируясь на атрибуты, которыми этот класс размечен.

Начните с изучения класса `VkApi`, на котором будет тестироваться ваш `Specifier`. Это всего лишь пример класса, который мы хотим документировать, поэтому ни один метод там не реализован (и реализовывать их не нужно).

Изучите тест `Specifier_should`. В нём зафиксированы требования к поведению вашей реализации `ISpecifier`.

СПИСОК ЛИТЕРАТУРЫ

Основная литература:

1. Мейер Б., Основы программирования. [Электронные ресурс]: учебник / Б. Мейер – 2-е изд. – Москва: ИНТУИТ, 2016 – 422 с. - Режим доступа: https://e.lanbook.com/book/100317 — ЭБС «Лань» (дата обращения: 26.04.2020). - Режим доступа: для авториз. пользователей.

Дополнительная литература:

1. Гуриков, С. Р. Введение в программирование на языке Visual C#: учеб. пособие / С.Р. Гуриков. — Москва: ФОРУМ: ИНФРА-М, 2018. — 447 с. — ISBN 978-5-16-105882-4. - Текст: электронный. - URL: https://znanium.com/catalog/product/967691 — ЭБС «Знаниум» (дата обращения: 26.04.2020). – Режим доступа: по подписке.

2. Селиванова И.А. Построение и анализ алгоритмов обработки данных [Электронный ресурс]: учебно-методическое пособие/ Селиванова И.А., Блинов В.А.— Электрон. текстовые данные. — Екатеринбург: Уральский федеральный университет, ЭБС АСВ, 2015. — 108 с.— Режим доступа: http://www.iprbookshop.ru/68277.html — ЭБС «IPRbooks» (дата обращения: 26.04.2020)

Интернет-ресурсы:

1. Национальный открытый университет «ИНТУИТ» http://www.intuit.ru/
2. Интерактивные онлайн-курсы по программированию https://ulearn.me/

Современные профессиональные базы данных и информационные справочные системы:

1. Национальная электронная библиотека https://rusneb.ru/