File: ./pattern/adapter/program.cs
```
namespace AdapterPattern
{
    // Пространство имен "AdapterPattern"

    internal static class Program
    {
        // Класс "Program", отвечающий за точку входа в программу

        private static void Main()
        {
            // Метод "Main", который является точкой входа в программу

            var turkey = new WildTurkey(); // Создаем экземпляр класса "WildTurkey",
представляющий объект индейки, и присваиваем его переменной "turkey"

            var adapter = new TurkeyAdapter(turkey); // Создаем адаптер класса
"TurkeyAdapter", передавая в его конструктор объект индейки ("turkey"), и присваиваем его
переменной "adapter"

            Tester(adapter); // Вызываем метод "Tester", передавая в него адаптер ("adapter")
        }

        private static void Tester(IDuck duck)
        {
            // Метод "Tester", который принимает объект типа "IDuck"

            duck.Fly(); // Вызываем метод "Fly" на объекте, реализующем интерфейс "IDuck",
чтобы имитировать полет утки

            duck.Quack(); // Вызываем метод "Quack" на объекте, реализующем интерфейс
"IDuck", чтобы имитировать кряканье утки
        }
    }
}

// WildTurkey - класс, представляющий объект индейки. Этот класс реализует свои собственные
методы Gobble и Fly.
// TurkeyAdapter - класс адаптера, который принимает объект индейки и адаптирует его к
интерфейсу IDuck (интерфейс для утки).
// turkey - создание экземпляра класса WildTurkey, который представляет объект индейки, и
присвоение его переменной turkey.
// adapter - создание экземпляра класса TurkeyAdapter, передача объекта индейки (turkey) в
его конструктор для адаптации, и присвоение его переменной adapter.
// Tester(adapter) - вызов метода Tester, передача адаптера (adapter) в качестве аргумента.
// duck.Fly() - вызов метода Fly на объекте, реализующем интерфейс IDuck, чтобы имитировать
полет утки.
// duck.Quack() - вызов метода Quack на объекте, реализующем интерфейс IDuck, чтобы
имитировать кряканье утки.
```

File: ./pattern/average.md
```
[[average]] [[sum]] [[array]]
``` c#
// Код для вычисления среднего значения массива

class AverageSolution {
  public static double FindAverage(double[] array) {
    double average = 0; // Переменная для хранения суммы элементов массива
    if (array.Length > 0) // Проверяем, что массив не пустой
    {
      for (int i = 0; i < array.Length; i++) // Проходим по каждому элементу массива
      {
        average += array[i]; // Суммируем элементы массива
      }
```
```

```
      return average / array.Length; // Возвращаем среднее значение, разделив сумму на
количество элементов
    }
    return 0; // Если массив пустой, возвращаем 0
  }
}
```

``` c#
// Код для вычисления среднего значения массива с использованием LINQ

using System.Linq; // Подключаем пространство имен для использования LINQ

class AverageSolution {
  public static double FindAverage(double[] array) {
    return array.Length == 0 ? 0 : array.Average(); // Возвращаем среднее значение массива с
использованием метода Average() из LINQ. Если массив пустой, возвращаем 0.
  }
}

```

``` c#
// Код для вычисления среднего значения массива с использованием выражения-тела метода и LINQ

using System.Linq; // Подключаем пространство имен для использования LINQ

public class AverageSolution {
  public static double FindAverage(double[] array) => array.Average(); // Возвращаем среднее
значение массива с использованием метода Average() из LINQ. Используем выражение-тело метода.
}
```


File: ./csharp+js/arr-2for-acum.md
```js
function accum(s) {
        let result = [], r0_arr = s.toUpperCase()//все в верхний регистр для первой буквы

  for (let i=0; i

public class Accumul // Объявление публичного класса Accumul
{
    public static string Accum(string s) // Объявление публичного статического метода Accum
с параметром s типа string
    {
        List result = new List(); // Создание экземпляра класса List для хранения результатов
        string r0_arr = s.ToUpper(); // Преобразование строки s в верхний регистр и
сохранение результата в переменную r0_arr типа string

        for (int i = 0; i < r0_arr.Length; i++) // Цикл, проходящий по каждому индексу в
строке r0_arr
        {
            string r0_s = r0_arr[i].ToString(); // Преобразование текущего символа в строке
r0_arr в тип string и сохранение результата в переменную r0_s типа string

            for (int r0_i = 0; r0_i < i; r0_i++) // Вложенный цикл, проходящий от 0 до
текущего индекса
            {
                r0_s += r0_arr[i].ToString().ToLower(); // Добавление символов текущего
символа в строку r0_s в нижнем регистре
            }

            result.Add(r0_s); // Добавление строки r0_s в список result
        }

        return string.Join("-", result); // Соединение элементов списка result с
использованием дефиса и возврат результирующей строки
    }
}
```

```
```

```py
def accum(s):
    result = []
    r0_arr = s.upper()

    for i in range(len(r0_arr)):
        r0_s = r0_arr[i]

        for r0_i in range(i):
            r0_s += r0_arr[i].lower()

        result.append(r0_s)

    return '-'.join(result)
```

File: ./csharp+js/перебрать.md
```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

public class Example
{
    // Метод для удаления дубликатов из списка
    public static List Distinct(List a)
    {
        // Создаем новый список и копируем в него элементы из исходного списка
        List r = new List(a);

        // Перебираем элементы исходного списка
        for (int i1 = 0; i1 < a.Count; i1++)
        {
            // Вложенный цикл для сравнения элементов с последующими элементами списка
            for (int i2 = i1 + 1; i2 < r.Count; i2++)
            {
                // Если элементы равны, удаляем повторяющийся элемент из нового списка
                //Условное выражение для сравнения элементов списка a и r на равенство.
                if (EqualityComparer.Default.Equals(a[i1], r[i2]))
                {
                    r.RemoveAt(i2);
                    i2--; // Уменьшаем индекс, чтобы корректно обработать следующий элемент
                }
// if (a[i1].Equals(r[i2])) {
//     r[i2] = default(T); // Удаляем повторяющийся элемент путем замены на значение по
умолчанию
// }
            }
        }

        return r; // Возвращаем новый список без дубликатов
    }

    // Метод для удаления дубликатов из списка с использованием фильтра
    //Объявление публичного статического метода DistinctFilter с обобщенным типом T. Метод
принимает список (List a) и возвращает список (List).
    public static List DistinctFilter(List a)
    {
        return a.Distinct().ToList(); // Используем метод Distinct и ToList для получения
списка без дубликатов
    }

    // Метод для удаления дубликатов из списка с помощью цикла
    public static List DistinctLoop(List a)
    {
        List r = new List(); // Создаем новый список для хранения уникальных элементов
```

```
        // Цикл foreach для перебора элементов исходного списка a.
        foreach (T item in a)
        {
            // Условное выражение для проверки, содержит ли список r элемент item.
            if (!r.Contains(item))
            {
                // Добавление элемента item в список r.
                r.Add(item);
            }
        }

        return r; // Возвращаем новый список без дубликатов
    }

    // Метод для проверки, содержит ли строка только символы в верхнем регистре или не
содержит буквы английского алфавита
    // Метод для проверки, содержит ли строка только символы в нижнем регистре с
использованием регулярного выражения
    public static bool IsLowerCaseRegex(string input)
    {
        return Regex.IsMatch(input, @"^[a-z]+$");
    }
}
```

```js
//перебрать и удалить дубликаты
    // через два фор
    const distinct=a=>
    {
        let r = [...a];
        for(let i1=0; i1<=a.length; i1++){
            for(let i2=i1+1; i2<=r.length; i2++){
                if(a[i1]==r[i2]){delete r[i2];}
            }
        }
        //удалить пустые еллемменты массива
        return r.filter(function(el) {return el != null;});;
    }

    //через фильтр
    const distinct=a=>a.filter((item,index)=>a.indexOf(item)===index);

    //пресобрать в новый массив, если в масиве еще нет числа то мы его туда положим
    for(let i = 0; i < arr.length; i++){
        if(!новыйМассив.includes(arr[i]))новыйМассив.push(arr[i]);
    //
    for (const i of names) !r0.includes(i) ? r0.push(i) :0;
    for (const i of names) r0.includes(i) ? 0 : r0.push(i);

    // и послать если совпадает содержимое
        const r0_abc = 'abcdefghijklmnopqrstuvwxyz';
        let r_false = true;
        for(let i=0; i b) {
            rMax = a;
            rMin = b;
        } else {
            rMax = b;
            rMin = a;
        }
        //метод max arr.Max() System.Linq
        int max(int[] arr){ return arr.Max();}
        int min(int[] arr){ return arr.Min();}

        int rMax1 = 0;
        foreach (var i in arr){if (i > rMax1){rMax1 = i;}}

        int rMin1 = 0;
```

```
        foreach (var i in arr){if (i < rMin1){rMin1 = i;}}

        //метод rMax2 arr.Aggregate() System.Linq
        int rMax2 = arr.Aggregate((a, i) => a > i ? a : i);
        //является методом расширения из пространства имен System.Linq. Он используется для
агрегации (объединения) элементов коллекции
        int rMin2 = arr.Aggregate((a, i) => a < i ? a : i);

        List rResult = new List();
        // Создаем новый список для хранения результата
        List rClon = new List(arr);
        // Создаем копию исходного массива
        while (rClon.Count > 0)
        // Пока в копии массива есть элементы
        {
            int rMax3 = rClon.Max();
            // Находим максимальное значение в копии массива
            rResult.Add(rMax3);
            // Добавляем найденное максимальное значение в список результата
            rClon.Remove(rMax3);
            // Удаляем найденное максимальное значение из копии массива
        }
        int result = int.Parse(string.Join("", rResult));
        // Объединяем все элементы из списка rResult в одну строку и Затем строка
преобразуется в целое число с помощью метода Parse


// самый маленький ел в масиве
int arrayMin(int[] arr)
{
    int len = arr.Length; // Получаем длину массива
    int min = int.MaxValue;
// Инициализируем переменную min максимальным значением типа int. Это обеспечит, что первый
элемент массива будет принят за минимальное значение.

    while (len--)
// будет выполняться до тех пор, пока значение len больше или равно 0. При каждой итерации
уменьшаем len на 1.
    {
        if (arr[len] < min)
    // Если текущий элемент массива меньше значения min
        {
            min = arr[len];
 // Присваиваем значение текущего элемента переменной min
        }
    }
    return min; // Возвращаем минимальное значение
}

int arrayMax(int[] arr)
{
    int len = arr.Length; // Получаем длину массива
    int max = int.MinValue;
// Инициализируем переменную max минимальным значением типа int
    while (len--)
    {
        if (arr[len] > max)
// Если текущий элемент массива больше значения max
        {
            max = arr[len];
// Присваиваем значение текущего элемента переменной max
        }
    }
    return max; // Возвращаем максимальное значение
}

    }
}

```

```js
// внемасива
  if(a>b) {rMax=a; rMin=b;} else {rMax=b; rMin=a;};
//Math.
  const max = (arr) => Math.max(...arr);
  const min = (arr) => Math.min(...arr);
// for макс
  let rMax = 0;
  for(const i of rArr){i>rMax?rMax=i:0}
// for мин
  let rMin = rArr[0];
  for(const i of rArr){ia>i?a:i);
let rMin=arr.reduce((a,i)=>aa>i ? a:i);
  rResult.push(rMax);
  rClon.splice(rClon.indexOf(rMax), 1);
}
rResult = Number(rResult.join(''));



/// скорость
function arrayMin(arr) {
  var len = arr.length, min = Infinity;
  while (len--) {
    if (arr[len] < min) {
      min = arr[len];
    }
  }
  return min;
}

function arrayMax(arr) {
  var len = arr.length, max = -Infinity;
  while (len--) {
    if (arr[len] > max) {
      max = arr[len];
    }
  }
  return max;
}

```
File: ./csharp+js/if-math.md
```js
function bmi(weight, height) {
  const rResult = weight / Math.pow(height, 2);
  if (rResult <= 18.5) {
    return "Underweight"
  } else if (rResult <= 25) {
    return "Normal"
  } else if (rResult <= 30) {
    return "Overweight"
  } else if (rResult > 30) {
    return "Obese"
  }
}
```

```C#
using System;

public class Kata
{
    public static string Bmi(double weight, double height)
    {
        double result = weight / Math.Pow(height, 2);
        if (result <= 18.5)
        {
            return "Underweight";
```

```
        }
        else if (result <= 25)
        {
            return "Normal";
        }
        else if (result <= 30)
        {
            return "Overweight";
        }
        else
        {
            return "Obese";
        }
    }
}
```

```py
def bmi(weight, height):
    result = weight / height ** 2
    if result <= 18.5:
        return "Underweight"
    elif result <= 25:
        return "Normal"
    elif result <= 30:
        return "Overweight"
    else:
        return "Obese"

```
File: ./csharp+js/sum.md
```cs
// array Объявление переменной rSum и инициализация нулевым значением
int rSum=0;

// Цикл перебора элементов массива rArr
foreach (var rKey in rArr)
{
    rSum+=rKey; // Прибавление значения rKey к переменной rSum
}
// Вычисление суммы элементов массива rArr
rArr.Aggregate((a,b)=>a+b);

// склеить два массива в один
arr1.Concat(arr2).ToArray();

// obj в масс определённый ел обж
// Объявление переменной rSum и инициализация нулевым значением
int rSum = 0;
foreach (var i in rArr) // Цикл перебора элементов массива rArr
{
    foreach (var rKey in i.Keys) // Цикл перебора ключей объекта i
    {
        if (rKey == "ИмяЕл") // Проверка, равен ли текущий ключ "ИмяЕл"
        {
            rSum += i[rKey]; // Прибавление значения по ключу rKey из объекта i к переменной
rSum
        }
    }
}
// Возврат значения 0, если коллекция countries пуста, иначе возврат значения rSum
return countries.Count == 0 ? 0 : rSum;

```

```js

// array
        for (const rKey of rArr) {rSum += rKey
        // .map(i => i < r1 ? r2 : r3)
```

```
        rArr.reduce((a,b) => a+b);
        // склеить два массива в один
        [...arr1, ...arr2]
// obj
// в масс определённый ел обж
        let rSum = 0;
        for (const i of rArr) {
                for (const rKey in i) {
                        rKey === 'ИмяЕл' ? rSum += i[rKey] : 0
                }
        }
        return countries.length === 0 ? 0 : rSum;

}
```

File: ./question/ответы.md

асинхронность.md

Асинхронность в C# позволяет выполнять операции, которые не блокируют основной поток выполнения, что повышает отзывчивость и эффективность приложения.

базовый класс в дотнет

Базовый класс в .NET - это основной класс, от которого наследуются все остальные классы в языке программирования C#. Он предоставляет базовые функции, такие как преобразование в строку, сравнение объектов, получение хеш-кода и получение типа объекта.

боксинг и анбоксинг

Боксинг и анбоксинг в C# относятся к процессу преобразования значимых типов данных в ссылочные типы и обратно. Например, преобразование целого числа в объект и обратно. Это может повлиять на производительность и использование памяти.

гарбедж коллектор

Гарбедж коллектор (сборщик мусора) в C# отвечает за автоматическое управление памятью в программе. Он освобождает память, которая больше не используется объектами, чтобы избежать утечек памяти и повысить эффективность работы программы.

какие типы данных ты знаешь

В C# существует несколько типов данных, таких как целочисленные типы (int, long), типы с плавающей точкой (float, double), символьный тип (char), логический тип (bool), строковый тип (string) и другие. Тип данных определяет характеристики и операции, которые можно выполнять с данными.

разница между класом и структурой

Разница между классом и структурой в C# заключается в том, что классы являются ссылочными типами данных, передаются по ссылке и могут наследоваться, в то время как структуры являются типами значения, передаются по значению и не поддерживают наследование.

структура и класы разница

Структуры и классы имеют различия в поведении, копировании и использовании памяти. Структуры обычно используются для представления небольших значений, в то время как классы используются для представления объектов и сложных данных.

тип значение и тип ссылки

В C# существуют типы значения (value types) и типы ссылки (reference types). Типы значения хранятся непосредственно в памяти, а типы ссылки хранятся в куче и имеют указатели на фактические данные.

что такое ексепшын

Исключение (Exception) в C# представляет ошибку или исключительную ситуацию, которая может возникнуть во время выполнения программы. Исключения позволяют обрабатывать ошибки, управлять потоком выполнения и предоставлять информацию о возникшей проблеме.

что таокое рест

REST (Representational State Transfer) - это архитектурный стиль для разработки веб-сервисов. Он опирается на принципы, такие как использование HTTP-методов (GET, POST, PUT, DELETE) для взаимодействия с ресурсами и передача данных в формате JSON или XML.

юниттесты из чего состоят, и логические блоки

Юнит-тесты состоят из отдельных логических блоков, которые проверяют отдельные части кода на правильность работы. Они используют утверждения (assertions) для проверки ожидаемых результатов и могут включать настройку и очистку окружения для изоляции тестов.

какие джоины бывают

В T-SQL (Transact-SQL), языке запросов для работы с базами данных Microsoft SQL Server, доступны следующие типы соединений (joins):

INNER JOIN: Возвращает строки, которые имеют совпадающие значения в обеих таблицах, на основе условия соединения.

LEFT JOIN (или LEFT OUTER JOIN): Возвращает все строки из левой (первой) таблицы и соответствующие строки из правой (второй) таблицы, на основе условия соединения. Если в

правой таблице нет совпадающих строк, то возвращается NULL.

RIGHT JOIN (или RIGHT OUTER JOIN): Возвращает все строки из правой (второй) таблицы и соответствующие строки из левой (первой) таблицы, на основе условия соединения. Если в левой таблице нет совпадающих строк, то возвращается NULL.

FULL JOIN (или FULL OUTER JOIN): Возвращает все строки из обеих таблиц и соответствующие строки на основе условия соединения. Если нет совпадающих строк, то возвращаются NULL значения для недостающих столбцов.

CROSS JOIN: Возвращает декартово произведение двух таблиц, то есть комбинацию каждой строки из первой таблицы со всеми строками из второй таблицы.

Это основные типы соединений в T-SQL, которые позволяют объединять данные из разных таблиц на основе заданных условий.

File: ./Training marathon 2022/Wwt code display.cs

```
// namespace BSA2021: Определение пространства имен BSA2021
namespace BSA2021
{
// struct Point: Объявление структуры Point, представляющей координаты точки с публичными полями X и Y.
    struct Point
    {
        public int X;
// Объявление публичного поля X типа int в структуре Point
        public int Y;
// Объявление публичного поля Y типа int в структуре Point
    }
//    class Program: Определение класса Program.
    class Program
    {
// static void Main(): Главный метод программы, который будет выполнен при запуске программы.
        static void Main()
        {
// CreatePoint(out Point point): Метод, который создает новую точку и присваивает ее значению переменной point с использованием ключевого слова out.
// Вызов метода CreatePoint с передачей переменной point по ссылке (out)
//вызов CreatePoint(). Его параметр помечен как out , поэтому накладывается условие инициализации внутри метода.
// Структура считается инициализированной, когда инициализированы все ее поля, то есть ошибок нет, а точка имеет значение (2; 2).

//out также указывает, что аргумент передается по ссылке. Этот аргумент называется «выходным параметром» метода и, в отличие от ref , не требует предварительной инициализации параметра. Вместо этого внутри метода аргумент должен быть инициализирован внутри этого метода, иначе CS0177 . возникает ошибка компиляции
            CreatePoint(out Point point);

// ResetPoint(in Point point): Метод, который пытается сбросить координаты точки, переданной в качестве параметра point, используя ключевое слово in, которое указывает на то, что параметр передан по ссылке только для чтения.
// ResetPoint() с параметром in
//В рамках этого метода этот параметр назначается, поэтому мы имеем ошибку компиляции CS0165 .
            ResetPoint(in point);
// Вызов метода ResetPoint с передачей переменной point по ссылке только для чтения (in)

//ref указывает, что аргумент передается по ссылке, а не по значению. Таким образом, любое изменение значения этого аргумента в методе. Изменит соответствующий аргумент, с которым был вызван метод. Требует, чтобы переменная была инициализирована перед использованием ее в качестве параметра, иначе возникнет ошибка CS0165 . компиляции

            IncreasePoint(ref point);
// переназначить локальную переменную
// переменная должна быть инициализирована перед использованием как параметра иначе ошибка компиляции
//аргумент передається за посиланням, а не за значенням
// IncreasePoint(ref Pointв point): Метод, который увеличивает координаты точки, переданной в качестве параметра point, используя ключевое слово ref, которое указывает на то, что параметр передан по ссылке и может быть
```

```
// Вызов метода IncreasePoint с передачей переменной point по ссылке (ref)

            Console.WriteLine($"({point.X};{point.Y})");
// Вывод значения переменных point.X и point.Y в формате координат (X;Y) в консоль
        }

// (только для чтения) In - указывает параметр, передаваемый по ссылке, но внутри метода его
менять запрещено. Это ключевое слово обычно используется, когда внутри метода необходимо
передать структуры, интенсивно использующие память. При попытке изменить этот параметр
возникает ошибка компиляции CS8332 — невозможность изменить переменную readonly. Также
необходимо предварительно инициализировать переменную, переданную в качестве параметра.
// попытка изменить переменную переданную в метод с ключ-словом in приводит к ошибке
        static void ResetPoint(in Point point)
        {
            point.X = 1;
        // Попытка изменить значение поля X структуры Point, но так как point передано по
ссылке только для чтения (in), это вызовет ошибку компиляции
            point.Y = 1;
        // Попытка изменить значение поля Y структуры Point, но так как point передано по
ссылке только для чтения (in), это вызовет ошибку компиляции
        }

        static void CreatePoint(out Point point)
        {
            point.X = 2;
        // Присвоение значению поля X структуры Point значения 2
            point.Y = 2;
        // Присвоение значению поля Y структуры Point значения 2
        }

        static void IncreasePoint(ref Point point)
        {
            point.X *= 3;
        // Умножение значения поля X структуры Point на 3
            point.Y *= 3;
        // Умножение значения поля Y структуры Point на 3
                }
    }
}


структура и три вызова методов
структура хранится  стеке и является типом значения
методы действия над ключевыми словами структуры (in, ref, out), которые служат
модификаторами параметров

д) ошибка компиляции.

File: ./Training marathon 2022/What will someVariable be equal to.js
function func() {
  return this;
}

const outerObj = {
  value: 10,
  innerObj: {
    value: 20,
    method: func,
  },
};

const target = outerObj.innerObj;
const someVariable = target.method();

// я говорю вернет нихуя undefined
// The correct answer - innerObj.

// Если вы не знали, объекты JS — это сложный тип данных, и в отличие от примитивных типов,
таких как число или термин, объекты присваиваются и копируются по ссылке. Другими словами, в
переменной хранится не «значение объекта», а «ссылка» (адрес в памяти) этого значения.
```

Следовательно, копирование такой переменной или передача ее в качестве аргумента функции копирует ссылку, а не сам объект. Все операции с использованием скопированных ссылок (например, добавление или удаление свойств) выполняются с одним и тем же объектом.
// То есть в нашем случае targetта же innerObj, а не его копия. Просто теперь у нас есть 2 пути к его свойствам и методам — outerObj.innerObjи target
// thisравен объекту перед точкой, и в этом случае объект перед точкой target. Но, как мы поняли выше, targetи innerObjэто не просто два идентичных объекта или две копии — это один и тот же объект. Таким образом, у нас есть someVariable, targetи outerObj.innerObjвсе ссылки на один объект в памяти.

File: ./Training marathon 2022/Wwb result of executing this code.cs

```
//What will be the result of executing this code

class Program
{
    static void Main(string[] args)
    {
        // обьект ферст является екзмпляром класса Секонд с конструктором принимающи строковый параметр
        // вызывается конструктор класа Ферст выводит Ферст Криейтед. Потом вызывается конструктор класа Секонд выводящий "...+"some item""
        // Создание экземпляра класса Second и присваивание его переменной first
        First first = new Second("some item");
        // обьект анозер из екземпляр класса Секонд с конструктором принимающим стрококвый параметр
        // Тхирд наследует от класа Секонд который наследует от класа Ферст то будут вызваны конструкторы всех трех класов Ферст Секонд(является закрытым и не доступен) Тхирд
        // Создание экземпляра класса Third и присваивание его переменной another типа Ферст
        First another = new Third();
    }
}

//  Когда мы создаем экземпляр класса, у которого есть родительский класс, сначала вызывается конструктор родительского класса. По умолчанию для родительского класса будет вызываться конструктор без параметров. base()ключевое слово используется для явного вызова требуемого конструктора.
class First
{
    public First()
    {
        Console.WriteLine("First created");
    }
}
// second наследует от класса ферст
class Second : First
{
    private Second()
    {
        Console.WriteLine("Second created");
    }
    public Second(string item)
    {
        Console.WriteLine("Second created with Item: {0}",  item);
    }
}
// наследует от класа Секонд
class Third : Second
{
    //конструктор
    public Third()
    {
        Console.WriteLine("Third created");
    }
}

// на самом делеКласс First имеет единственный конструктор без параметров, который выводит строку «First created» . в консоль

// Второй класс имеет два конструктора. Один из них не имеет параметров и выводит строку «Второй создан» , а его уровень доступа определяется как private, поэтому он доступен только
```

для вызова "внутри" этого класса. Другой конструктор имеет itemпараметр и печатает сообщение "Second created with Item: {0}". Оба этих конструктора неявно вызывают First()конструктор.

// Третий « класс имеет единственный конструктор без параметров, который печатает Третий создан» в консоли. Однако, поскольку явного вызова конструктора батика с параметрами нет, приватный конструктор Second()будет называться. Это вызовет ошибку компиляции CS0122 .

// Правильный ответ: б) Ошибка компиляции.

// не понял

File: ./Training marathon 2022/Which of the catch blocks will be involved as a result of this code execution.cs

```
// Which of the catch blocks will be involved as a result of this code execution?
static void Recursive()
{
    Recursive();
}
// Это метод Recursive(), который вызывает сам себя рекурсивно. Этот код создает бесконечную рекурсию, так как метод не имеет условия выхода из рекурсии.
try
{
    Recursive();
    //метод вызывает сам себя рекурсивно бесконечное количество раз, возникает исключение StackOverflowException.
}
catch (InsufficientMemoryException)
//исключение указывает на то, что доступной памяти оказалось недостаточно. Это исключение возникает только при явном использовании объекта MemoryFailPoint. Является производным типом от OutOfMemoryException.
{
    // block 1
    throw;
    // Конструкции throw в середине catch приводят к тому, что ошибка «передается на более высокий уровень», и даже если после текущего блока catch есть другой блок, способный перехватить эту ошибку по своему типу, он не будет выполнен.
    //в данном случае этот блок не будет выполнен, так как возникшее исключение является StackOverflowException. Затем исключение снова выбрасывается (throw;), чтобы передать его на уровень выше.
}
catch (OutOfMemoryException)
//сообщает, что недостаточно памяти для продолжения выполнения программы. Исключение возникает либо при превышении максимального значения длины для StringBuilder, либо когда CLR не может выделить достаточно непрерывной памяти для выполнения операции (создание объекта, вызов метода и т. д.). Эта ошибка связана с кучей .
{
    // block 2
    throw;
    //перехватывает исключение OutOfMemoryException, но в данном случае этот блок также не будет выполнен по той же причине, что и предыдущий блок. Исключение снова выбрасывается (throw;).
}
catch (StackOverflowException)
//возникает, когда стек времени выполнения переполняется, потому что он содержит слишком много вызовов вложенных методов. Если эта ошибка не вызывается явно (посредством throw), ее нельзя перехватить с помощью try…catch.
{
    // block 3
    throw;
    //перехватывает исключение StackOverflowException. Так как это исключение было сгенерировано внутри метода Recursive(), этот блок будет выполнен. В данном случае блок просто выбрасывает исключение снова (throw;), чтобы передать его на уровень выше.
}
catch (Exception)
//базовый тип для всех исключений. Используется для объявления перехвата любой ошибки. Теперь рассмотрим, что происходит в блоке try: вызывается метод Recursive(). Этот метод не использует MemoryFailPoint и не создает и не изменяет объекты, поэтому блоки 1 и 2 не будут задействованы. Вместо этого метод создает неконтролируемую рекурсию, которая переполняет стек и вызывает исключение StackOverflowException, сгенерированное системой. Поэтому блоки 3 и 4
```

также не будут задействованы.
{
    // block 4
    throw;
    //перехватывает общее исключение Exception. Он будет выполнен только в том случае, если ни один из предыдущих блоков catch не перехватит исключение. В данном случае этот блок не будет выполнен, так как исключение уже было перехвачено блоком catch (StackOverflowException). Исключение снова выбрасывается (throw;).
}

Правильный ответ: д) ни один блок не будет задействован.

File: ./Training marathon 2022/Wwt code display.js
```
//What will this code display?
const obj = {
    a: 1,
    //Внутри функции используется ключевое слово this, которое ссылается на текущий объект, в данном случае obj. Метод b возвращает значение свойства a объекта obj.
    b: function () {
      return this.a;
    },
    //Стрелочные функции не имеют своего собственного контекста выполнения, поэтому ключевое слово this внутри метода c будет ссылаться на контекст выполнения, в котором была определена функция obj. В данном случае, поскольку функция obj была определена в глобальной области видимости, значение this будет ссылаться на глобальный объект
    c: () => this.a,
  };
  // Анонимная функция b ()имеет свой собственный контекст, который он перехватывает в obj.
  console.log(obj.b());// 1
  //Стрелочная функция в момент создания ищет ближайший к ней контекст и запоминает его, а она такая же, как и в случае простого присваивания внутри obj
  //Для стрелочных функций thisопределяется в момент их создания и больше никогда не меняется
  console.log(obj.c());// undefine
```
File: ./Training marathon 2022/Which line execution will cause an error.cs
```
namespace BSA2021
{
  class Program
  {
    static void Main()
    {
      var array = new string[] { "Welcome", "to", "Binary", "Studio", "Academy" };
//var a = array[..0]; - Создается новый массив a, который содержит элементы из исходного массива array с индексами от начала до (не включая) индекс 0. Такой диапазон ..0 означает, что ни один элемент не будет включен в новый массив a. В итоге a будет пустым массивом.
      var a = array[..0];
//var b = array[2..^0]; - Создается новый массив b, который содержит элементы из исходного массива array с индексами от 2 (включительно) до индекса перед последним элементом (используя отрицательный индекс ^0). Таким образом, в b будут содержаться строки "Binary", "Studio".
      var b = array[2..^0];
//var c = array[^0]; - Создается переменная c, которая присваивается значение элемента исходного массива array с индексом, определенным отрицательным индексом ^0. Отрицательный индекс ^0 указывает на последний элемент массива array. Таким образом, c будет содержать строку "Academy".
      var c = array[^0];//IndexOutOfRangeException
    }
  }
}
```

File: ./Training marathon 2022/static-method_and_екземпляр-класа.md
Основная разница между статическим методом и экземпляром класса заключается в том, как они связаны с данными и как к ним обращаться.

    Статический метод:
        Принадлежит классу, а не конкретному экземпляру класса. Он является частью самого класса и доступен через имя класса.
        Не имеет доступа к данным экземпляра класса. Он может работать только с данными, доступными через статические поля или другие статические элементы класса.
        Вызывается непосредственно через имя класса, без необходимости создания экземпляра

класса.
Часто используется для предоставления общей функциональности, которая не требует доступа к данным экземпляра, или для обработки глобальных операций.

Пример статического метода:

csharp

```csharp
public class MyClass
{
    public static void MyStaticMethod()
    {
        // Код статического метода
    }
}

// Вызов статического метода через имя класса
MyClass.MyStaticMethod();
```

Экземпляр класса:
Представляет конкретный объект класса. Методы экземпляра работают с данными, принадлежащими этому объекту, и имеют доступ к состоянию объекта.
Требует создания экземпляра класса перед его использованием. Каждый экземпляр класса имеет свое собственное состояние и может вызывать методы и обращаться к полям, специфичным для этого экземпляра.
Позволяет работать с данными и изменять их на уровне объекта.

Пример экземпляра класса:

csharp

```csharp
public class MyClass
{
    public void MyInstanceMethod()
    {
        // Код метода экземпляра
    }
}

// Создание экземпляра класса
MyClass myObject = new MyClass();

// Вызов метода экземпляра через объект класса
myObject.MyInstanceMethod();
```

Использование статических методов и экземпляров класса зависит от конкретной задачи и требований проекта. Статические методы обычно используются для предоставления общей функциональности или выполнения операций, которые не требуют доступа к данным экземпляра. Экземпляры класса обычно используются для работ с конкретными данными объекта и выполнения операций, специфичных для каждого экземпляра.

File: ./Training marathon 2022/bsa-week2-calc/Calculator.cs

```csharp
// create пространство имен
namespace Calc
{
// в нем создаем доступный всем клас
    public class Calculator
    {
        // переменную доступную только через этот класс
        // приватное поле для хранения значения
        private int _value;
        // екземпляр который принимает переменную
        // имеет метод Адд
        // Функция Add возвращает экземпляр класса Calculator, что позволяет использовать цепочку вызовов методов. Такой подход, известный как паттерн "Fluent API" или "Fluent Interface", позволяет записывать последовательные операции более компактно и удобно.
        public Calculator Add(int value)
        {
            // меняем значение
            _value += value;
```

```
            return this;
        }

        // возвращает финальное число
        public int Equals() => _value;
    }
}

File: ./Training marathon 2022/bsa-week2-calc/stubs/NET Calculator.cs
using System;

namespace CalculatorApp
{
    // Пространство имен "CalculatorApp"
        // чтобы создать екстеншын метод нужен любой статик класс
    public static class Calculator
    {
        // Публичный класс "Calculator", представляющий калькулятор
        private double _result;
        // Приватное поле "_result" для хранения результата вычислений

                //
        public Calculator(double initialValue)
        {
            // Конструктор класса Calculator, принимающий начальное значение

            _result = initialValue;
            // Инициализация поля "_result" начальным значением
        }

        public Calculator Add(double value)
        {
            // Метод "Add", который добавляет значение к результату

            _result += value;
            // Добавление значения к текущему результату

            return this;
            // Возвращаем текущий экземпляр калькулятора для возможности цепочки вызовов
методов
        }

        public Calculator Subtract(double value)
        {
            // Метод "Subtract", который вычитает значение из результата

            _result -= value;
            // Вычитание значения из текущего результата

            return this;
            // Возвращаем текущий экземпляр калькулятора для возможности цепочки вызовов
методов
        }

        public Calculator MultiplyBy(double value)
        {
            // Метод "MultiplyBy", который умножает результат на значение

            _result *= value;
            // Умножение текущего результата на значение

            return this;
            // Возвращаем текущий экземпляр калькулятора для возможности цепочки вызовов
методов
        }

        public Calculator DivideBy(double value)
        {
            // Метод "DivideBy", который делит результат на значение
```

```
            if (value == 0)
            {
                throw new ArgumentException("Division by zero is not allowed.");
            }
            // Проверка, что значение не равно нулю. Если равно, выбрасываем исключение.

            _result /= value;
            // Деление текущего результата на значение

            return this;
            // Возвращаем текущий экземпляр калькулятора для возможности цепочки вызовов
методов
        }

        public double Equals()
        {
            // Метод "Equals", который возвращает текущий результат

            return _result;
            // Возвращаем текущий результат
        }
    }

    class Program
    {
        // Класс "Program", отвечающий за точку входа в программу

        static void Main(string[] args)
        {
            // Метод "Main", который является точкой входа в программу

            var result = new Calculator(5)
                .Add(10)
                .MultiplyBy(2)
                .Equals();
            // Создаем экземпляр калькулятора с начальным значением 5, добавляем 10, умножаем
на 2 и получаем результат

            Console.WriteLine(result); // Выводим результат в консоль. Ожидаемый вывод: 30

            try
            {
                result = new Calculator(10)
                    .DivideBy(0)
                    .Equals();
                // Создаем экземпляр калькулятора с начальным значением 10, пытаемся поделить
на 0 (деление на ноль не разрешено), и получаем результат
            }
            catch (ArgumentException ex)
            {
                Console.WriteLine(ex.Message); // Выводим сообщение об ошибке в консоль.
Ожидаемый вывод: Division by zero is not allowed.
            }
        }
    }
}

File: ./Training marathon 2022/bsa-week2-calc/bsa-week2-calc.csproj



    Exe
    net7.0
    bsa_week2_calc
    enable
    enable
```

```
File: ./Training marathon 2022/bsa-week2-calc/Program.cs
// обращаемся к пространству имен
using Calc;
// создаем переменную и кладем в нее класс из нашего пространства имен
var result = new Calculator()
    //через клас хватаем функцию адд и кладем в нее значение для переменной велью
    .Add(5)
    // вызываем нашу функцию которая возвращает переменную обновленную в адд
    .Equals();
// почему мы не сделали все в Адд ? зачем создавать снаружи Еквалс?
Console.WriteLine(result);
File: ./Training marathon 2022/bsa-week2-calc/.возвращает базу метод/Program copy.cs
using Calc;

var result = new Calculator().Add(5);

Console.WriteLine(result);
File: ./Training marathon 2022/bsa-week2-calc/.возвращает базу метод/Calculator copy.cs
namespace Calc
{
    public class Calculator
    {
        private int _value;
        public Calculator Add(int value)
        {
            _value += value;
            return this;
        }
    }
}

File: ./Training marathon 2022/bsa-week2-calc/.возвращает базу метод/bsa-week2-calc
copy.csproj



    Exe
    net7.0
    bsa_week2_calc
    enable
    enable




File: ./Training marathon 2022/quest-3max/Program.cs
// Подключаем пространство имен System, которое предоставляет основные классы и
функциональность в C#
// using - директива, позволяющая использовать классы и методы из указанного пространства
имен без указания полного имени
using System;// System - пространство имен, содержащее базовые классы и функциональность
.NET Framework
using System.Linq;// Linq - пространство имен, содержащее классы и методы для работы с
запросами и операциями над данными (Language Integrated Query)

// public - модификатор доступа, позволяющий обращаться к классу из других частей программы
public class Program// Объявляем публичный класс Program
{
    public static void Main(string[] args)
    {
        // Объявляем метод Main, который является точкой входа в программу
        // static - модификатор, означающий, что метод Main является статическим и доступен
без создания экземпляра класса
        // void - тип возвращаемого значения, указывающий, что метод Main не возвращает
результат
        // Ввод пользовательских целых неотрицательных чисел для массива, разделенных
пробелом
        Console.WriteLine("Введите целые неотрицательные числа для массива, разделяя их
пробелом:");
```

```
        // Считываем введенную строку с целыми числами
        string input = Console.ReadLine();

        // Разделяем введенную строку на отдельные числа, используя пробел в качестве
разделителя, и преобразуем их в целые числа
        string[] numStrings = input.Split(' ');
        int[] nums = numStrings.Select(int.Parse).ToArray();

// Находим - указывает на то, что следующая часть кода выполняет поиск
// третьего максимального элемента в массиве

int thirdMax = // Объявляем переменную thirdMax, которая будет хранить третий максимальный
элемент

nums.Distinct() // Оставляем только уникальные элементы в массиве nums
// Distinct() - метод, который возвращает последовательность, содержащую только уникальные
элементы

.OrderByDescending(x => x) // Сортируем элементы массива в порядке убывания
// OrderByDescending() - метод, который сортирует элементы последовательности в порядке
убывания

.Skip(2) // Пропускаем первые два элемента после сортировки
// Skip() - метод, который пропускает указанное количество элементов

.FirstOrDefault(); // Возвращаем первый элемент после пропуска
// FirstOrDefault() - метод, который возвращает первый элемент последовательности или
значение по умолчанию, если последовательность пуста

        // Находим третий максимальный элемент в массиве
//        int thirdMax = nums.Distinct().OrderByDescending(x => x).Skip(2).FirstOrDefault();

        // Находим максимальный элемент в массиве
        int max = nums.Max();

        // Проверяем, если третий максимальный элемент не равен 0, выводим его
        if (thirdMax != 0)
        {
            Console.WriteLine("Третий максимальный элемент: " + thirdMax);
        }
        // В противном случае выводим максимальный элемент
        else
        {
            Console.WriteLine("Максимальный элемент: " + max);
        }
    }
}
```

```
public class ThirdMaxElement {
    // Объявляем публичный класс ThirdMaxElement

    public int ThirdMax(int[] nums) {
        // Объявляем публичный метод ThirdMax, принимающий массив целых чисел nums и
возвращающий целое число

        if(nums.Length == 0){
            throw new ArgumentException("nums should have at least one element");
        }
        // Проверяем, если массив nums пуст, выбрасываем исключение с сообщением об ошибке
```

```
        return _ThirdMaxRec(nums, 1, nums[0], null, null);
        // Возвращаем результат рекурсивного вызова приватного метода _ThirdMaxRec,
        // передавая ему массив nums, начальный индекс 1, первый максимальный элемент nums[0]
        // и значения второго и третьего максимального элементов, равные null
    }

    private int _ThirdMaxRec(int[] nums, int index, int fMax, int? sMax, int? tMax){
        // Объявляем приватный метод _ThirdMaxRec, принимающий массив nums, индекс, первый
максимальный элемент fMax,
        // и значения второго и третьего максимальных элементов sMax и tMax (может быть null)

        // terminate
        if(index == nums.Length)
        {
            return tMax ?? fMax;
        }
        // Базовый случай: если индекс достигает длины массива, возвращаем третий
максимальный элемент tMax,
        // если он определен, в противном случае возвращаем первый максимальный элемент fMax

        // handle duplicates
        if(nums[index] == fMax || nums[index] == sMax || nums[index] == tMax)
        {
            return _ThirdMaxRec(nums, index + 1, fMax, sMax, tMax);
        }
        // Обрабатываем случай, когда текущий элемент nums[index] является дубликатом одного
из максимальных элементов.
        // Рекурсивно вызываем _ThirdMaxRec, переходя к следующему индексу, и передаем те же
значения максимальных элементов

        // handle replacements
        if(nums[index] > fMax)
        {
            return _ThirdMaxRec(nums, index + 1, nums[index], fMax, sMax);
        }
        // Обрабатываем случай, когда текущий элемент nums[index] больше первого
максимального элемента fMax.
        // Заменяем значения максимальных элементов и рекурсивно вызываем _ThirdMaxRec,
        // переходя к следующему индексу и передавая новые значения максимальных элементов

        if(sMax == null || nums[index] > sMax)
        {
            return _ThirdMaxRec(nums, index + 1, fMax, nums[index], sMax);
        }
        // Обрабатываем случай, когда текущий элемент nums[index] больше второго
максимального элемента s

File: ./Training marathon 2022/quest-3max/3max.js
```js

/** Находит третий максимальный элемент */
@param {number[]} array
// - массив с по крайней мере одним элементом
@return {number | null}
// Третий максимальный элемент или максимальный элемент, если в массиве меньше трех различных
значений. Если `array` пуст, будет возвращено `null`

// Этот блок является комментарием в стиле JSDoc и предоставляет документацию для функции
maxThirdElement. Он объясняет, что делает функция, ожидаемый входной параметр и возвращаемое
значение. Сказано, что функция находит третий максимальный элемент в заданном массиве или
максимальный элемент, если в массиве меньше трех различных значений. Если массив пуст, будет
возвращено null.

function maxThirdElement(array){

// Эта строка объявляет функцию с именем maxThirdElement, которая принимает один параметр
array.

const set = new Set(array);
```
```

```
// Эта строка создает новый объект Set из заданного массива. Set - это встроенный объект JavaScript, который хранит только уникальные значения, удаляя все дубликаты. Это гарантирует, что в последующих шагах будут рассматриваться только уникальные значения.

const sorted = Array.from(set).sort((a, b) => b - a);
// Эта строка преобразует объект Set обратно в массив с помощью метода Array.from(). Затем массив сортируется в порядке убывания с использованием метода .sort() и функции сравнения (a, b) => b - a. Этот порядок сортировки гарантирует, что самые большие числа будут находиться в начале массива.

return (sorted.length >= 3 ? sorted[2] : sorted[0]) ?? null;
// Эта строка с использованием условного (тернарного) оператора определяет значение, которое нужно вернуть. Если длина отсортированного массива больше или равна 3, возвращается элемент с индексом 2 (sorted[2]), который представляет третий максимальный элемент. В противном случае возвращается элемент с индексом 0 (sorted[0]), который представляет максимальный элемент. Оператор ?? используется для обработки случаев, когда массив пуст, в этом случае будет возвращено значение null.

// В целом, эта функция принимает массив, удаляет дубликаты, сортирует его по убыванию и возвращает либо третий максимальный элемент (если он доступен), либо максимальный элемент (если в массиве меньше трех различных значений). Если массив пуст, возвращается null.

}

```
File: ./Training marathon 2022/quest-3max/quest-3max.csproj

    Exe
    net7.0
    quest-3max
    enable
    enable

File: ./Training marathon 2022/Wwb printed in console.js
//What will be printed on the console?
//setTimeoutвыполняется асинхронно переданный обратный вызов будет вызван после завершения цикла.

// a)

for (var i = 0; i < 5; i++) {
  setTimeout(() => console.log(i), 1);
}
//В этом коде происходит цикл for, который выполняется пять раз. При каждой итерации выполняется функция setTimeout, которая откладывает выполнение переданной ей функции колбэка на 1 миллисекунду. Внутри колбэка вызывается функция console.log(i), где i - это значение переменной i на момент выполнения колбэка. Однако, из-за того, что выполнение откладывается на 1 миллисекунду, цикл for выполняется полностью, и к моменту выполнения колбэков переменная i будет иметь значение 5. Поэтому, в результате будут выведены пять раз число 5.

// b)

for (let i = 0; i < 5; i++) {
  setTimeout(() => console.log(i), 1);
}

//В этом коде также используется цикл for, но с ключевым словом let для объявления переменной i. В отличие от ключевого слова var, которое имеет функциональную область видимости, ключевое слово let имеет блочную область видимости. Поэтому каждая итерация цикла создает новую переменную i со своим собственным значением. Внутри колбэка функции setTimeout вызывается функция console.log(i), которая будет выводить значение соответствующей переменной i на момент выполнения колбэка. В результате будут выведены числа от 0 до 4, поскольку каждый колбэк получит свое собственное значение переменной i в пределах своей области видимости блока for.
```
```

```
b) a) 5 5 5 5 5; b) 0 1 2 3 4
// Он будет перезаписываться при каждой итерации, и когда setTimeoutвызывается, оно будет
иметь последнее значение

//Область видимости переменной, объявленной с помощью varявляется его текущим контекстом
выполнения.  Который может быть ограничен функцией или быть глобальным для переменных,
объявленных вне функции.

// letимеет область действия блока. Блок — это фрагмент кода, ограниченный фигурными
скобками. {}. Все, что находится внутри фигурных скобок, принадлежит блоку. Таким образом,
переменная, объявленная в блоке с letбудут доступны только внутри этого блока.


File: ./Training marathon 2022/questSkobkiBSA/Program.cs
using System;
using System.Collections.Generic;

// Объявление класса Program
public class Program
{
    // Метод для проверки валидности скобок. Допустимы ли круглые скобки.
    public static bool IsValidParentheses(string input)
    {
        // Создание нового стека для хранения открывающих скобок.
        // оюбьявление переменной типа Stack
        // с имененем stack
        // new создает екземпляр обьекта
        // Stack(): Это вызов конструктора класса Stack, который создает новый экземпляр
стека символов.
        Stack stack = new Stack();

        // Перебор каждого символа во входной строке
        foreach (char c in input)
        {
            // Если символ - открывающая скобка '(',
            // помещаем его в стек
            if (c == '(')
            {
                stack.Push(c);
            }
            // Если символ - закрывающая скобка ')',
            // проверяем, есть ли соответствующая открывающая скобка в стеке
            // Если нет или стек пуст, возвращаем false (недействительная последовательность
скобок)
            // Иначе, удаляем соответствующую открывающую скобку из стека
            else if (c == ')')
            {
                if (stack.Count == 0 || stack.Pop() != '(')
                {
                    return false;
                }
            }
        }

        // Проверяем, остались ли непарные открывающие скобки в стеке
        // Если стек пуст, возвращаем true (действительная последовательность скобок)
        // Иначе, возвращаем false (недействительная последовательность скобок)
        return stack.Count == 0;
    }

    // Главный метод программы
    public static void Main(string[] args)
    {
        // Выводим сообщение с просьбой ввести строку со скобками
        Console.WriteLine("Введите строку со скобками:");

        // Считываем введенную пользователем строку
        string input = Console.ReadLine();

        // Проверяем введенную строку на валидность скобок
```

```
        bool isValid = IsValidParentheses(input);

        // Выводим результат проверки на экран
        Console.WriteLine("Результат проверки: " + isValid);
    }
}



// Действительные скобки
// Binary Studio AcademyMarch 09, 2021

// Напишите функцию, которая принимает строку скобок *'(', ')'*, и определяет, является ли
порядок скобок действительным. Функция должна возвращать true, если строка является
допустимой, i false, если она недействительна.

// Приклад:

// "()"              =>  true
// ")(()))"          =>  false
// "("               =>  false
// "((()))(()())()"  =>  true


// Вимоги:

// Длина входящей строки:  1<= N <=100

// Радом со скобками открытия *(* и закрытия *)* входящая строка может содержать любые
допустимые символы ASCII. Кроме того, она может быть пустой и/или вообще не содержать скобок.

// Не рассматривайте другие формы скобок (например, [], {}, <>)!

File: ./Training marathon 2022/questSkobkiBSA/quest.csproj



    Exe
    net7.0
    enable
    enable




File: ./Training marathon 2022/What type should return this method.cs
// What type should return this method
// Определение публичного статического метода с именем GetNums и возвращаемым типом, который
будет определен позднее

public static {ReturnType} GetNums()
{
    var number = 0;
    while (true)
    {
        if (number > 5)
        // Выход из цикла и завершение итератора
            yield break;
        // Возврат текущего значения number, а затем его инкрементирование
        yield return number++;
    }
}
// snippet - фрагмент
// yield keyword indicates iterator method
// return type of an iterator method should be IEnumerable, where T is the type of the
elements being returned.
// Тип возвращаемого значения метода итератора должен быть IEnumerable, где T — это тип
возвращаемых элементов.
// Безымянный итератор объявляется для класса путем определения GetEnumerator()метод
// значение, возвращаемое итератором, должно быть интерфейсом IEnumerable
```

```
// ответ:
    // гпт IEnumerable
    //


File: ./Training marathon 2022/Решение задачи на умножение.md
#Задание:

Найти результат умножения двух длинных чисел.

#Вводные данные:

В первой строке содержится одно число длиной не более 255 знаков. Во второй строке содержится
операция (* - умножение). Третья строка содержит второе число, также длиной не более 255
знаков.

#Исходные данные:

Вывести единственное число – результат выполнения заданой операции над этими двумя числами.

```csharp
using System;

class Program
{
    static void Main()
    {
        string number1 = Console.ReadLine();
        char operation = Console.ReadLine()[0];
        string number2 = Console.ReadLine();

        int[] num1 = ConvertToDigitsArray(number1);
        int[] num2 = ConvertToDigitsArray(number2);

        int[] result = MultiplyNumbers(num1, num2);

        Console.WriteLine(ConvertToString(result));
    }

    static int[] ConvertToDigitsArray(string number)
    {
        int[] digits = new int[number.Length];

        for (int i = 0; i < number.Length; i++)
        {
            digits[i] = int.Parse(number[i].ToString());
        }

        return digits;
    }

    static int[] MultiplyNumbers(int[] num1, int[] num2)
    {
        int m = num1.Length;
        int n = num2.Length;
        int[] result = new int[m + n];

        for (int i = m - 1; i >= 0; i--)
        {
            for (int j = n - 1; j >= 0; j--)
            {
                int product = num1[i] * num2[j];
                int sum = product + result[i + j + 1];

                result[i + j] += sum / 10;
                result[i + j + 1] = sum % 10;
            }
        }
```
```

```
        return result;
    }

    static string ConvertToString(int[] number)
    {
        string result = "";

        foreach (int digit in number)
        {
            result += digit.ToString();
        }

        return result.TrimStart('0');
    }
}
using System;

class Program
{
    static void Main()
    {
        // Считываем первое число
        string number1 = Console.ReadLine();

        // Считываем операцию
        char operation = Console.ReadLine()[0];

        // Считываем второе число
        string number2 = Console.ReadLine();

        // Преобразуем числа в массивы цифр
        int[] num1 = ConvertToDigitsArray(number1);
        int[] num2 = ConvertToDigitsArray(number2);

        // Умножаем числа и получаем результат в виде массива цифр
        int[] result = MultiplyNumbers(num1, num2);

        // Выводим результат
        Console.WriteLine(ConvertToString(result));
    }

    // Метод для преобразования строки с числом в массив цифр
    static int[] ConvertToDigitsArray(string number)
    {
        // Создаем новый массив для хранения цифр
        int[] digits = new int[number.Length];

        // Проходимся по каждому символу в строке
        for (int i = 0; i < number.Length; i++)
        {
            // Преобразуем символ в целое число и сохраняем в массив
            digits[i] = int.Parse(number[i].ToString());
        }

        // Возвращаем массив цифр
        return digits;
    }

    // Метод для умножения двух чисел, представленных массивами цифр
    static int[] MultiplyNumbers(int[] num1, int[] num2)
    {
        // Получаем длины массивов
        int m = num1.Length;
        int n = num2.Length;

        // Создаем массив для хранения результата умножения
        int[] result = new int[m + n];

        // Выполняем умножение
        for (int i = m - 1; i >= 0; i--)
```

```
        {
            for (int j = n - 1; j >= 0; j--)
            {
                // Вычисляем произведение цифр и сумму с текущим значением результата
                int product = num1[i] * num2[j];
                int sum = product + result[i + j + 1];

                // Обновляем значения в массиве результата с учетом переносов
                result[i + j] += sum / 10;
                result[i + j + 1] = sum % 10;
            }
        }

        // Возвращаем массив результата умножения
        return result;
    }

    // Метод для преобразования массива цифр в строку
    static string ConvertToString(int[] number)
    {
        // Создаем пустую строку
        string result = "";

        // Проходимся по каждой цифре в массиве
        foreach (int digit in number)
        {
            // Преобразуем цифру в строку и добавляем к результату
            result += digit.ToString();
        }

        // Удаляем ведущие нули и возвращаем строку
        return result.TrimStart('0');
    }
}
```

Главной идеей задачи было не использовать сами числа, а работать исключительно со строками.

Такие задачки принадлежат к длинной арифметике.

Возьмем за пример числа:

12345 та 56789

На первом этапе там нужно перемножить каждую цифру первого числа на каждую цифру другого входящего числа, в результате у нас выйдет несколько новых значений.

На следующем этапе по правилам умножения в столбик нужно добавить ранее найденные значения начиная с последнего к первому со смещением на 1 в левую сторону при каждой итерации.

Для ускорения выполнения задачи следует выполнять последний шаг на каждом этапе умножения первого числа на некоторую цифру другого, в этом случае не нужно будет запоминать все полученные значения, а достаточно будет иметь текущий результат умножения на цифру и промежуточный общий результат.