1 1 задание

ссылка на все выполненные задания Реализуйте метод FailProcess так, чтобы процесс завершался. Предложите побольше различных решений.

```
using System;

class Program
{
    static void Main(string[] args)
    {
        try
        {
            FailProcess();
        }
        catch   {       }

        Console.WriteLine("Failed to fail process!");
        Console.ReadKey();
    }

    static void FailProcess(){      //... write your code here  }
}
```

Так как присутствует try-catch блок, бросить ошибку, чтобы сломать программу не получится. Необходимо использовать функции, прерывающие выполнение программы. Я использовал 3 варианта:

```
Environment.Exit(-1);

Environment.FailFast("Actualy failed!");

Process.GetCurrentProcess().Kill();
```

Environment.Exit(-1) Вызывает завершение программы с кодом ошибки -1. Environment.FailFast("Actualy failed!") завершает процесс и выводит дополнительное сообщение. Process.GetCurrentProcess().Kill() получает информацию о процессе и "убивает"его. Можно было бы придумать более изощренные способы: создать дочерний процесс, убить материнский вместе с ним; создать библиотеку, использующую syscall на C, создать dll библиотеку на C++, вызывающую функцию из библиотеки C, и через DLLImprot вызвать эту функцию на C. Но эти способы не выглядят целесообразными.

2 2 задание

Что выводится на экран? Измените класс Number так, чтобы на экран выводился результат сложения для любых значений someValue1 и someValue2.

```
using System;
using System.Globalization;

class Program
{
    static readonly IFormatProvider _ifp = CultureInfo.InvariantCulture;

    class Number
    {
        readonly int _number;

        public Number(int number)
        {
            _number = number;
        }

        public override string ToString()
        {
            return _number.ToString(_ifp);
        }
    }

    static void Main(string[] args)
    {
        int someValue1 = 10;
        int someValue2 = 5;

        string result = new Number(someValue1) + someValue2.ToString(_ifp);
        Console.WriteLine(result);
        Console.ReadKey();
    }
}
```

Требуется перегрузить оператор сложения для класса Number так, чтобы при сложении со строкой, происходила не конкатенация строк, а сложение чисел. Я перегрузил его слудующим образом:

```
public static string operator +(Number a, string b)
{
    return (a._number + Int32.Parse(b, _ifp)).ToString(_ifp);
}
```

Так как тип result string, то я оставил вывод ответа в тип string. Но можно возвращать Number, что не повлияет на результат работы программы. Но это позволит присваивать возвращаемое значение не только типу string

```
public static Number operator +(Number a, string b)
{
    return Number(a._number + Int32.Parse(b, _ifp));
}
```

3 3 задание

Реализуйте метод по следующей сигнатуре:

```
/// <summary>
/// <para> Отсчитать несколько элементов с конца </para>
/// <example> new[] 1,2,3,4.EnumerateFromTail(2) = (1, ), (2, ), (3, 1), (4, 0)</example>
/// </summary>
/// <typeparam name="T»</typeparam>
/// <param name="enumerable»</param>
/// <param name="tailLength» Сколько элеметнов отсчитать сконца (у последнего элемента tail = 0)</param>
/// <returns></returns>
public static IEnumerable<(T item, int? tail)>
EnumerateFromTail<T>(this IEnumerable<T> enumerable, int? tailLength)
```

Возможно ли реализовать такой метод выполняя перебор значений перечисления только 1 раз?

Ответ: возможно, если использовать дополнительную память, и хранить последние tailLenght элементов. У этого решения есть минус - при большом tailLength (соизмеримом с размером объекта), затрыт по памяти будут соответствующими.

```
public static class Enumeration
{
    public static IEnumerable<(T item, int? tail)>
    EnumerateFromTail<T>(this IEnumerable<T> enumerable, int? tailLength)
    {
        //If tailLenght isn't defined
        var result = Enumerable.Empty<(T, int?)>();
        if (tailLength == null)
        {
            foreach (T el in enumerable)
            {
                result = result.Append((el, null));
            }
            return result;
        }

        //Calculate tale lenght in case tailLenght > length(enumerable)
        int numElements = enumerable.Count();
        int? tail;
        if (tailLength > numElements)
        {
            tail = numElements - 1;
            tailLength = numElements;
        }
        else
        {
            tail = tailLength - 1;
        }

        //Enumerating from tail
        int counter = 0;
        foreach (T el in enumerable)
        {
            if (counter >= numElements - tailLength)
            {
                result = result.Append((el, tail));
                tail--;
            }
```

```
            else
            {
                result = result.Append((el, null));
            }
            counter++;
        }
        return result;
    }
}
```

4 4 задание

Реализуйте метод Sort. Известно, что потребители метода зачастую не будут вычитывать данные до конца. Оптимально ли Ваше решение с точки зрения скорости выполнения? С точки зрения потребляемой памяти?

/// <summary>
/// Возвращает отсортированный по возрастанию поток чисел
/// </summary>
/// <param name="inputStream»Поток чисел от 0 до maxValue. Длина потока не превышает миллиарда чисел.</param>
/// <param name="sortFactor»Фактор упорядоченности потока. Неотрицательное число. Если в потоке встретилось число x, то в нём больше не встретятся числа меньше, чем (x - sortFactor).</param>
/// <param name="maxValue»Максимально возможное значение чисел в потоке. Неотрицательное число, не превышающее 2000.</param>
/// <returns>Отсортированный по возрастанию поток чисел.</returns>

```
IEnumerable<int> Sort(IEnumerable<int> inputStream, int sortFactor, int maxValue)
```

Для начала я отобрал только те элементы из потока, которые удовлетворяют условию. То есть не превосходят максимального значения и входят в диапазон (x-sortFacror).

```
var outputStream = inputStream.Where(el => el <= maxValue); //t = O(n), memory = O(n)
int minValue = outputStream.Max() - sortFactor;             //t = O(n), memory = O(1)
outputStream = outputStream.Where(el => el >= minValue);    //t = O(n), memory = O(n)
```

Дальше я сортировал элементы. Помимо метода OrederBy я также реализовал MergeSort. Оба варианта будут работать за $O(n*log(n))$. Если условие "Известно, что потребители метода зачастую не будут вычитывать данные до конца."означает прерывание выполнения метода, с последующим выводом отсортированной части, то тогда необходимо использовать сортировку пузырьком, так как она имеет отсортированную часть наибольших (наименьших, в зависимости от реализации) значений. Но сложность алгоритма $O(n^2)$

5 5 задание

Программа выводит на экран строку «Муха», а затем продолжает выполнять остальной код. Реализуйте метод TransformToElephant так, чтобы программа выводила на экран строку «Слон», а затем продолжала выполнять остальной код, не выводя перед этим на экран строку «Муха».

```
using System;

class Program
{
    static void Main(string[] args)
    {
        TransformToElephant();
        Console.WriteLine("Муха");
//... custom application code
}

    static void TransformToElephant()   {       //... write your code here  }
}
```

Я не нашел способа, позволяющего сделать это. не используя метки в коде после вывода слова "Муха"и JumpTo из метода на эту метку. Либо же перенос кода в метод и Exit(0).