Консультация по программированию.

Контакты ассистентов:

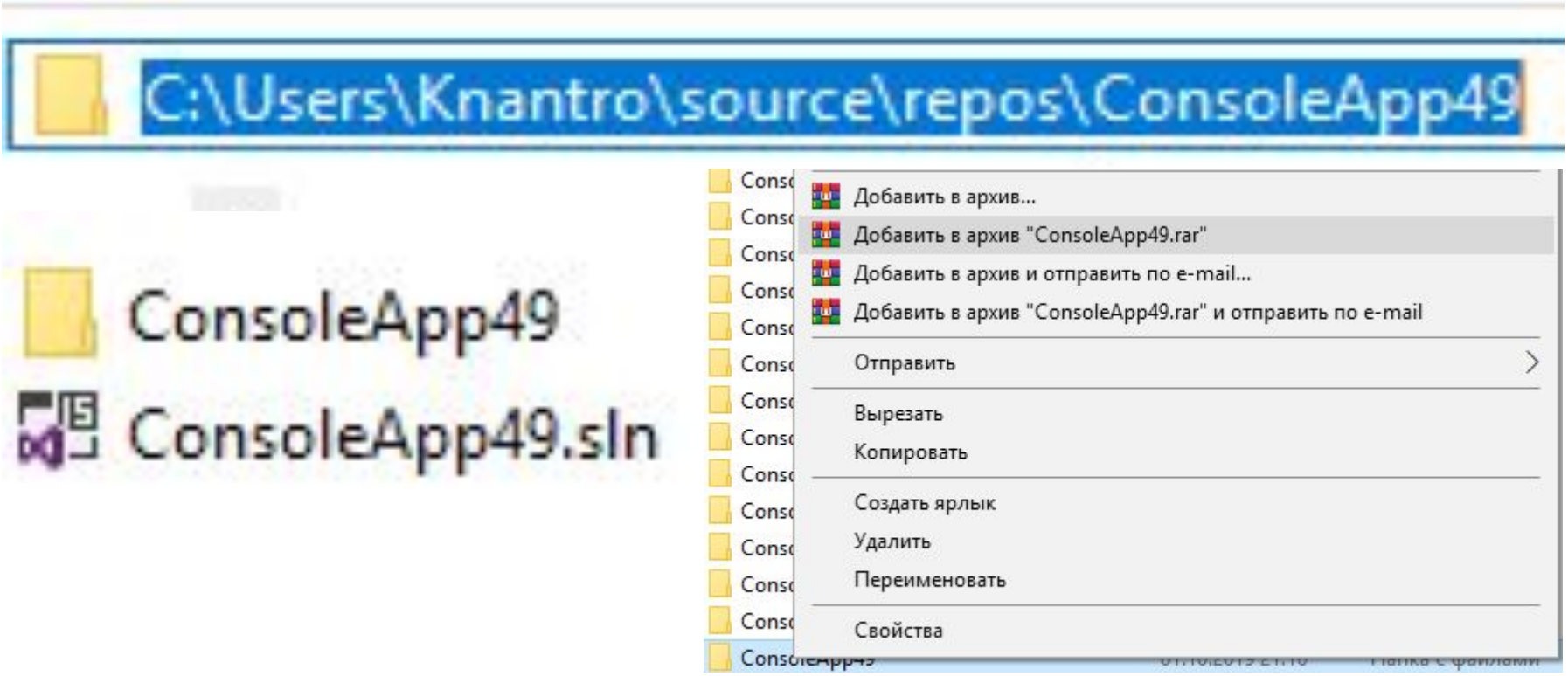
Штеменко Дмитрий:
ВК (приоритет):
vk.com/knantro
vk.com/id170337558
Telegram:
@Knantro

Янкин Антон:
Telegram (приоритет):
@KerJen
ВК:
vk.com/kerjen
vk.com/id468807610

Глущенко Захар:
ВК:
vk.com/mizarion
vk.com/id186816999
Telegram:
@mizarion

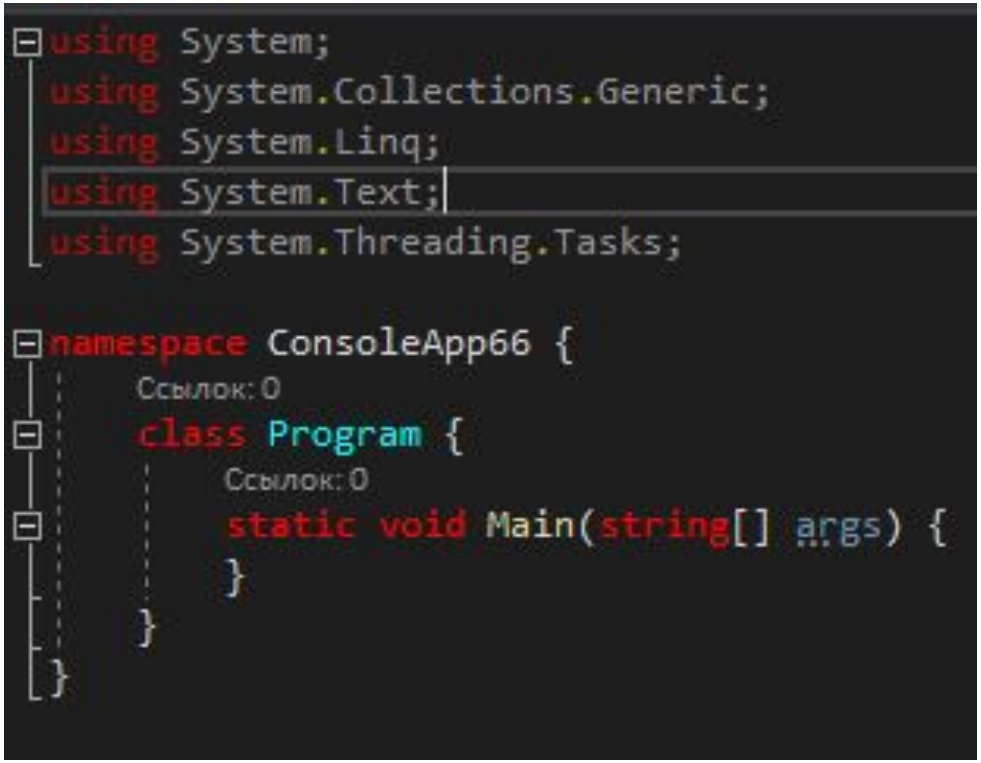
Как отправлять проект?

Если вы ничего не меняли, то по умолчанию, проекты хранятся здесь:



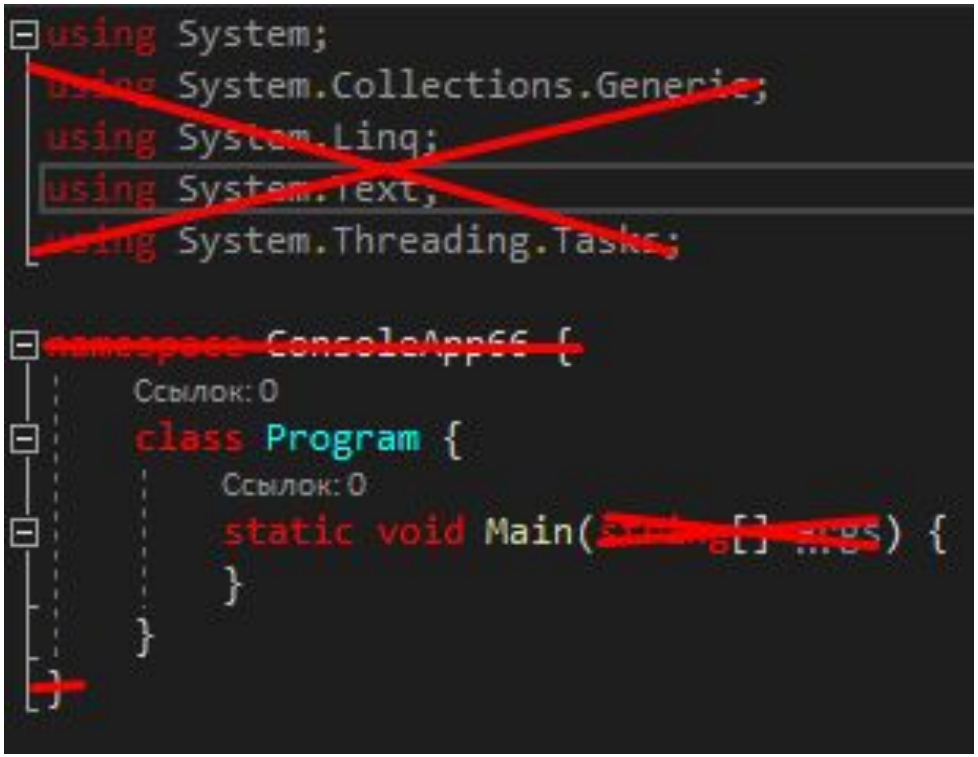
Вы увидите в папке папку и файл `.sln`. Поднимитесь на уровень выше и всю папку с этими двумя файлами заархивируйте и отправьте в Peer Grade. НЕ НУЖНО ОТПРАВЛЯТЬ ГОЛЫЙ `.cs` ФАЙЛ ИЛИ КАК-ТО ИНАЧЕ! За некорректную отправку проекта ставят ноль!

Кратко о правилах хорошего тона



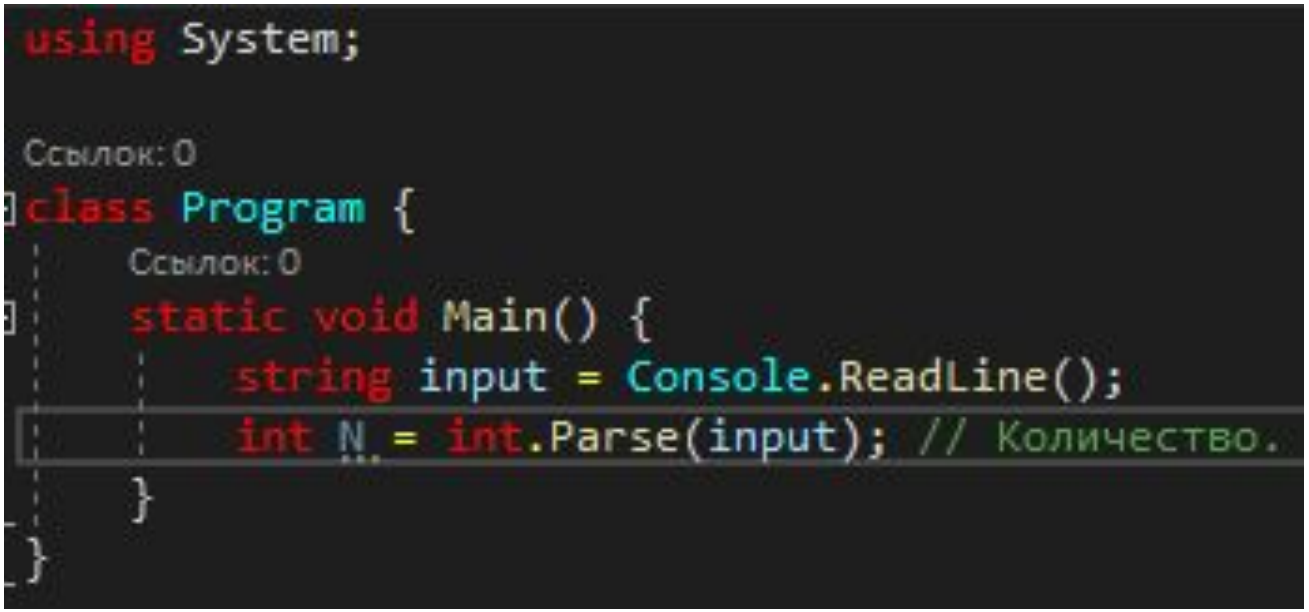
Перед вами – новосозданный проект.
Вопрос в следующем – что нужно удалить из него и нужно ли это вообще?

Кратко о правилах хорошего тона



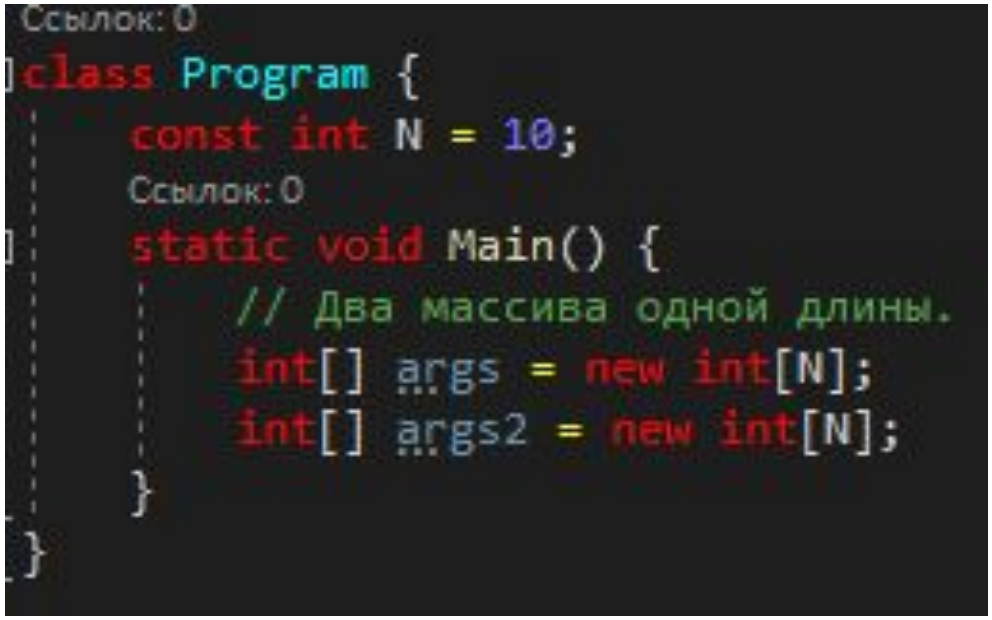
Удаляйте СРАЗУ всё, что вам не потребуется.
Понятное дело, если какие-то директивы `using` вам пригодятся точно и вы это знаете или, например, `string[] args` от `Main()`, то удалять это не нужно, но в большинстве случаев, они вам не пригодятся, поэтому стоит удалять ненужное сразу, чтобы не «пачкать» код.

Кратко о правилах хорошего тона



1. Помните о безопасном вводе. Пока вы не знаете обработку исключений – не используйте метод `Parse()`. Он выбрасывает исключения при любом некорректном вводе. Используйте вместо него `TryParse()`.
2. Не заводите переменные на 1 раз – экономьте память и стек! `string input` нужен только для ввода целого числа и больше не используется.
3. НО!!! В случае, если переменная используется 2 раза и больше – заводите переменные, константы и т.д. Магические константы использовать плохо.

Кратко о правилах хорошего тона



Здесь использована константа N, т.к. у нас 2 массива длины N.
P.S. Константные поля – автоматически являются статические, поэтому попытка добавить к N модификатор static приведёт к ошибке компиляции.
P.S.S. Помните, что базовый тип для целых чисел – Int32 (int), для вещественных – Double (double).

Кратко о правилах хорошего тона

```
using System;

Ссылок: 0
class Program {
    /// <summary>
    /// Вычисляет среднее арифметическое двух натуральных чисел.
    /// </summary>
    /// <param name="a">Первое натуральное число.</param>
    /// <param name="b">Второе натуральное число.</param>
    /// <exception cref="ArgumentOutOfRangeException">Если аргументы метода не являются натуральными числами.</exception>
    /// <returns>Среднее арифметическое двух натуральных чисел.</returns>
    ссылка: 1
    static double NaturalAverage(int a, int b) {
        if (a < 1 || b < 1) {
            throw new ArgumentOutOfRangeException();
        }
        return (a + b) / 2.0;
    }

    Ссылок: 0
    static void Main() {
        Console.WriteLine(NaturalAverage(10, 8));
    }
}
```

```
double Program.NaturalAverage(int a, int b)
Вычисляет среднее арифметическое двух натуральных чисел.

Возврат:
  Среднее арифметическое двух натуральных чисел.

Исключения:
  ArgumentOutOfRangeException

(параметр) int a
Первое натуральное число.

(параметр) int b
Второе натуральное число.
```

Для методов, полей и других функциональных и нефункциональных членов (кроме локальных данных), используйте XML-комментарии.

Кратко о кодстайле...

```
// Генератор множества Nq не повторяющихся
// целых чисел в диапазоне от 0 до Nm-1 (RVA)
1 usage
static void GenSetN(int Nq, int Nm, int[] qN)
{
    Random r = new Random();
    int p, k = 0;
    while (k < Nq)
    {
        p = r.Next(Nm);
        bool b = true;
        for (int i = 0; i < k; i++)
            if (p == qN[i])
            {
                b = false;
                break;
            }
        if (b)
        {
            qN[k] = p;
            k++;
        }
    }
}
```

```
using System;
namespace Игра_Быки_и_Коровы
{
    class Program
    {
```

```
uint[] Vvedch = new uint[100000000];
uint[] Nachch = new uint[100000000];
uint ch;
uint ostatok;
int a1;
int a2;
int y;
uint zagadch1;
int z;
string dem1;
```

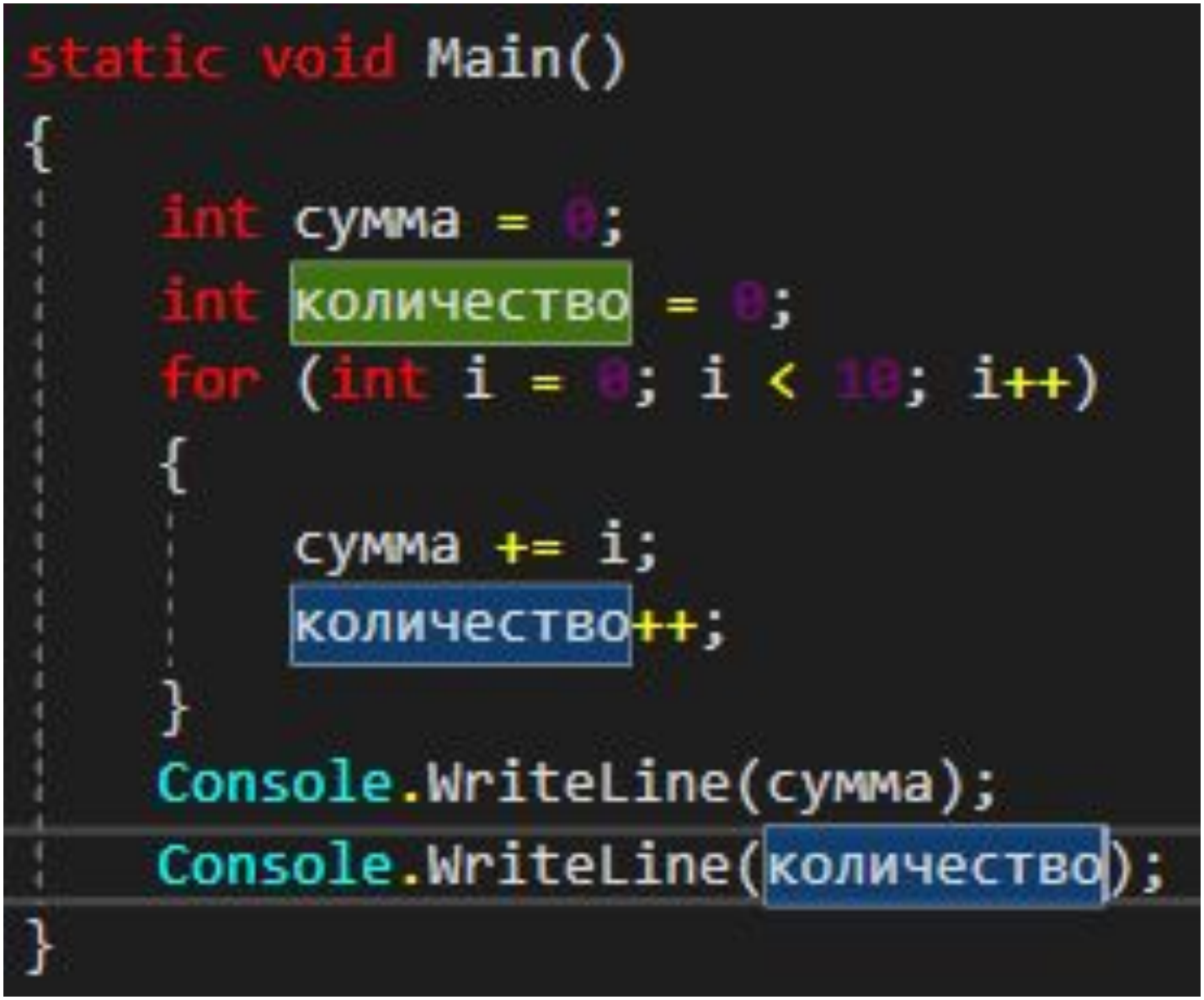
Ещё раз раз ссылка на соглашения по кодстайлу C#:
https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/inside-a-program/coding-conventions

Что плохо в этом коде?

```
using System;

class Program
{
    static void Main()
    {
        int a = 5;
        int b = int.Parse(Console.ReadLine());
        Console.WriteLine(a + b);
    }
}
```

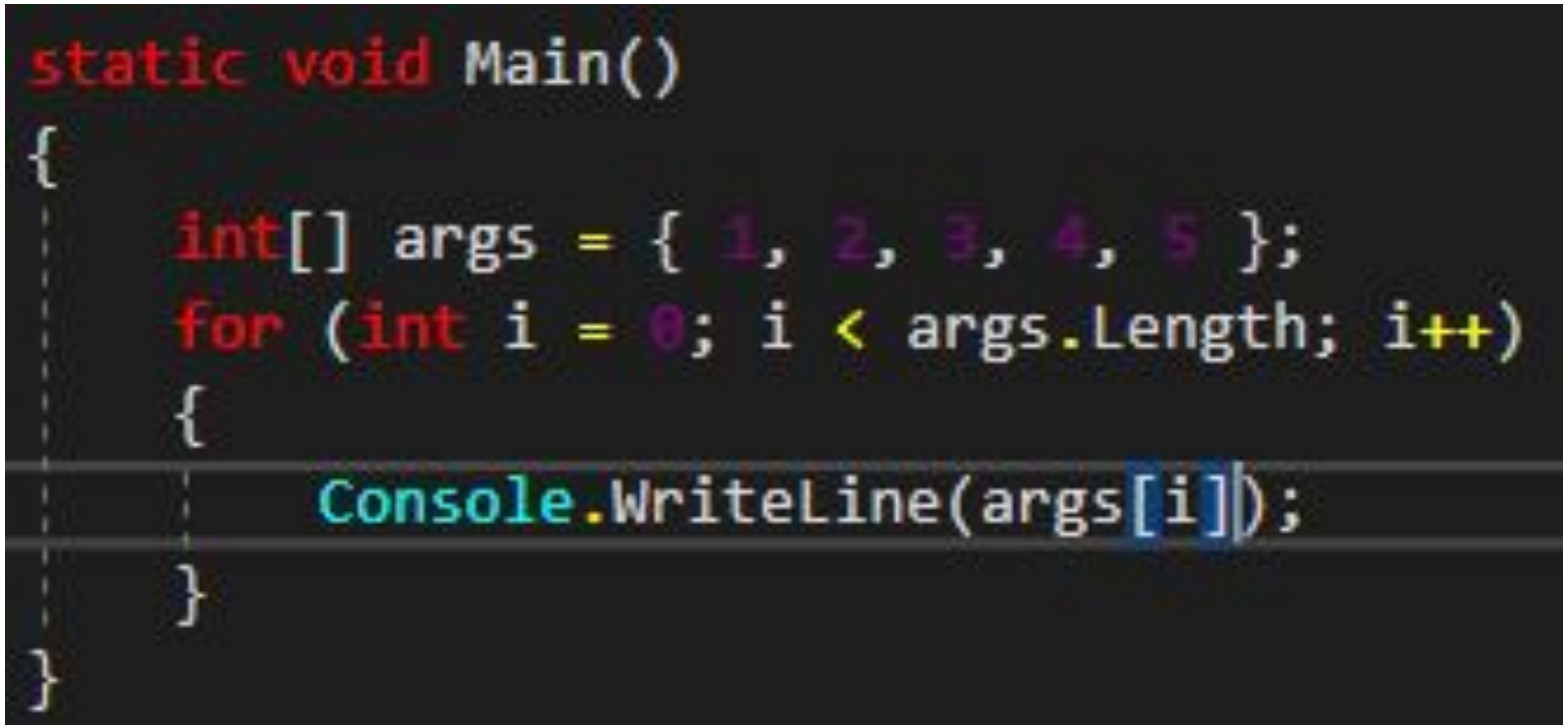
Что плохо в этом коде?



Что плохо в этом коде?

```
static void Main()
{
    try
    {
        int a = 5;
        int b = int.Parse(Console.ReadLine());
        Console.WriteLine(a + b);
    }
    catch (Exception ex)
    {
        Console.Write(ex.Message);
    }
}
```

Что плохо в этом коде?



Что плохо в этом коде?

```
static void Meth(int[] args, out int res)
{
    res = 0;
    for (int i = 0; i < args.Length; i++)
    {
        res += args[i];
    }
}
static void Main()
{
    int res;
    int[] args = { 1, 2, 3, 4, 5 };
    Meth(args,out res);
    Console.Write(res);
}
```

Что плохо в этом коде?

```
try
{
    int x = int.Parse(Console.ReadLine());
}
catch(FormatException f)
{
    Console.WriteLine(f.Message);
}
catch (ArgumentNullException arg)
{
    Console.WriteLine(arg.Message);
}
catch (OverflowException over)
{
    Console.WriteLine(over.Message);
}
```

Что плохо в этом коде?

```
static void Main()
{
    ConsoleKeyInfo ck;
    do
    {
        ck = Console.ReadKey();
        Console.WriteLine("Press <esc> to exit, any key to continue....");
    } while (ck.Key != ConsoleKey.Escape);
}
```

Что плохо в этом коде?

```
/// <summary>
/// Считает сумму элементов массива.
/// </summary>
/// <param name="args"></param>
static int Sum(int[] args)
{
    int res = 0;
    for (int i = 0; i < args.Length; i++)
    {
        res += args[i];
    }
    return res;
}
```

Что плохо в этом коде?

```
static int sum(int[] args)
{
    int res = 0;
    for (int i = 0; i < args.Length; i++)
    {
        res += args[i];
    }
    return res;
}
```

Что плохо в этом коде?

```
static int sum(int[] args)
{
    int res = 0;
    for (int i = 0; i < args.Length; i++)
    {
        res += args[i];
    }
    return res;
}
```

Что плохо в этом коде?

```
static void Main()
{
    int x;
    Console.WriteLine("Input x: ");
    while (!int.TryParse(Console.ReadLine(), out x))
        Console.WriteLine("Incorrect input! Input x again: ");
}
```

Файлы – фанаты исключений. Поэтому, когда работаете с ними, пихайте на них `try-catch` ВСЕГДА, иначе мы вашу программу сломаем. :-)

Уловки для экзамена.

Можно ускорить немного написание кода, если использовать СНИППЕТЫ.

Сниппеты используются так: пишется короткая комбинация букв, потом 2 раза нажимается кнопка «tab», после чего появляется фрагмент кода.
Примеры сниппетов:
for – цикл for с i до length (переменная).
cw – Console.WriteLine();
while – цикл while (true);
ctor – конструктор класса.

Уловки для экзамена.

Поскольку выпишете на своих компьютерах, вы можете некоторые фрагменты кода написать заранее:

```
static void Main()
{
    int x;
    Console.Write("Input x: ");
    while (!int.TryParse(Console.ReadLine(), out x))
        Console.Write("Incorrect input! Input x again: ");
}

do
{

    Console.WriteLine("Press <esc> to exit, any key to continue....");
} while (Console.ReadKey(true).Key != ConsoleKey.Escape);
```

Ввод числа с повтором ввода или повтор решения, который будет повышать вам баллы, это сэкономит ещё время на СР.

Уловки для экзамена.

Поскольку выпишете на своих компьютерах, вы можете некоторые фрагменты кода написать заранее:

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header> <!-->Основные данные о сниппете-->
      <Title>whilevar</Title> <!--Название сниппета-->
      <Shortcut>whilevar</Shortcut> <!--Комбинация символов для вызова сниппета-->
      <Description>Code snippet for looped variable input with checking</Description>
<!--Описание сниппета-->
      <Author>Knantro</Author> <!--Автор сниппета-->
      <SnippetTypes>
        <SnippetType>Expansion</SnippetType> <!--Тип сниппета-->
      </SnippetTypes>
    </Header>
    <!--Декларации для реализации гибкого динамического сниппета-->
    <Snippet>
      <Declarations>
        <Literal> <!--Задание литералов для сниппета-->
          <ID>var</ID> <!--Идентификатор (имя) литерала для сниппета-->
          <ToolTip>Name of variable</ToolTip> <!--Всплывающая подсказка с описанием
                                                  литерала-->
          <Default>x</Default> <!--Дефолтное значение литерала при использовании
                                  сниппета-->
        </Literal>
        <Literal>
          <ID>type</ID>
          <ToolTip>Type of variable</ToolTip>
          <Default>int</Default> <!--Дефолтный тип переменной-->
        </Literal>
        <Literal Editable="false">
          <ID>SystemConsole</ID>
          <Function>SimpleTypeName(global::System.Console)</Function>
        </Literal>
      </Declarations>
      <!—Код для вставки по сниппету-->
      <Code Language="csharp">
      <!—Блок для расположения заранее заданных литералов-->
        <![CDATA[$type$$end$ $var$;
        $SystemConsole$.Write("Input $var$: ");
          while(!$type$.TryParse($SystemConsole$.ReadLine(), out $var$))
          $SystemConsole$.Write("Incorrect input! Input $var$ again: ");
          ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

Ввод числа с повтором ввода или повтор решения, который будет повышать вам баллы, код будет писаться быстрее.

(-) Не выводится масса объекта. (-) Некорректная реализация метода ChangeColor(). Не соблюдена спецификация, которая предусматривает смену цвета на новый, отличного от старого. (-) Некорректная реализация пункта 3.4, нет энумератора, реализованного вручную. Значительное отклонение от спецификации, оценка не выше 5. (+) Хорошая обработка исключений. (-) try-catch на весь Main(). (-) 1 верный LINQ-запрос. (-) Информация об объектах дописывается в конец файла, из-за чего десериализация может быть невозможна. Выполнение примечания: к первостепенным задачам относится успешная сериализация и десериализация, при невыполнении которых выставляется оценка не выше 3 из 10.	3
(-) Вручную реализованный энумератор возвращает коллекцию, включая элементы с массой, равной 0. (-) Неверная смена цвета вывода в консоли. (-) Плохая декомпозиция. (-) 1 верный LINQ-запрос. (-) Необработанное исключение: System.InvalidOperationException. При отсутствии файла и неудачной десериализации программа продолжает работу с пустой коллекцией. (-) Лишние директивы using. (-) Некорректная реализация метода ChangeColor(). Не соблюдена спецификация, которая предусматривает смену цвета на новый, отличного от старого.	3
(-) Некорректная реализация метода ChangeColor(). Не соблюдена спецификация, которая предусматривает смену цвета на новый, отличного от старого. (-) В пункте 4.4 добавляется целое, а не вещественное число. (-) В пункте 4.4 не используется переопределённая операция сложения. (-) try-catch на весь Main(). (+) 3 верных LINQ-запроса.	6
(-) Необработанное исключение: System.InvalidOperationException. Первая программа может сериализовать пустую коллекцию, а вторая программа вне зависимости от размера коллекции, обрабатывает эту коллекцию, из-за чего происходит исключение, так как коллекция пустая. (-) Необработанное исключение System.FileNotFoundException при попытке сериализовать удалённый файл. Нет обработки исключений при работе с файлами. (+) 3 верных LINQ-запроса. (-) 2 класса в одном файле.	3
(-) Не компилируется.	1

Примеры комментариев к работам. Справа указаны проставленные оценки за работу. Комментарии стоит писать безличные, максимально

А теперь посмотрим пример одного из прошлогодних заданий и попробуем проверить работу:

Вариант 1.

Напишите консольное приложение, реализующее следующее:
1) Сформируйте 2 массива длиной **N** (**N** вводится пользователем с клавиатуры) состоящий из **N** случайных целых чисел в диапазоне [-10; 10] каждый.
2) Выведите на экран результаты операций сложения, вычитания, умножения и деления между соответствующими элементами первого и второго массива.
Пример вывода:
Массив 1: 10, 5, 1.
Массив 2: 4, 5, 6.
Результат сложения: 14, 10, 7.
Результат вычитания: 6, 0, -5.
Результат умножения: 40, 25, 6.
Результат деления: 2, 1, 0.
То есть для вычитания и деления нужно вычитать из элемента первого массива элемент второго массива и делить элемент из первого массива на элемент из второго массива (всё делать для целых чисел без преобразования в вещественный тип).
В случае, если произойдёт деление на ноль, вывести вместо результата деления: "-".

3) Сформировать третий массив **C** длины **N**, который содержит в себе элементы по следующей формуле:
(первый массив – **A**, второй массив – **B**)
$C[i] = A[i] * 2 + B[i] * 3;$

4) Сохраните все три массива в файле "arrays.txt", расположенный в одной папке с исполняемым файлом, в формате:
Первый массив: <элементы массива через запятую и пробел>
Второй массив: <элементы массива через запятую и пробел>
Третий массив: <элементы массива через запятую и пробел>
Пример:
Первый массив: 10, 5, 1
Второй массив: 4, 5, 6
Третий массив: 32, 25, 20