**Здравствуйте, Антон.**

Выполняю лабораторные работы, ссылка на GitHab:

<https://github.com/GeorgiyIsaev/ITMO.CSCourse2021.Labs>

По мере выполнения лабораторных работ возникли некоторые вопросы:

Во всех задачах Лабораторной 3 нет коллекции с последними датами месяцев:

var DaysInMonths = new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

foreach (int daysInMonth in DaysInMonths){}

Лабораторная 3, упражнение 3, страница 25 – снова используется коллекция, не указанная в методичке, полагаю такая:

var DaysInLeapMonths = new int[] { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

**Здравствуйте, Антон.**

В лабораторной 05 нахватает хотя бы несколько строчек, о том, как вызывать аргументы в приложении. Я конечно разобрался, но было бы неплохо если было бы, хотя бы что типа такого:

/\* Запуск приложения с аргументами

\* Вариант 01: 1. Открыть командную строку

\* 2. Кинуть скомпилированный файл .exe в командную строку

\* 3. После полного имени файла через пробел ввести несколько аргументов:

C:\Users\Isaev>C:\Repository\ITMO.CSCourse2021.Labs\ITMO.CSCourse2021.Lab

s.Lab05.\_01\bin\Debug\ITMO.CSCourse2021.Labs.Lab05.01.exe help notHelp

\* Вариант 02: Создать ярлык .exe-файла, в свойствах у полного имени объекта прописать аргументы после знака тире:

C:\Repository\ITMO.CSCourse2021.Labs\ITMO.CSCourse2021.Labs.Lab05.\_01\bin\Debug

\ITMO.CSCourse2021.Labs.Lab05.01.exe -help nothelp

\*/

В лабораторной 05 стр 33: нет информации о том, что необходимо подключить using System.IO; для работы с классом FileStream.

В лабораторной 05 стр 33: Из контекста задачи непонятно какого типа файл мы должны открыть через аргументы. Полагаю, это следовало бы прописать, так как это не так уж и очевидно. Я использовал .txt

Вопрос: в нескольких лабораторных по методичке требовалось переименовать класс содержавший точку входу. Я уже увидел, что программа действительно запускается даже с иным именование этого класса. НО допустимо ли делать так в реальных проектах?

**Здравствуйте, Антон.**

В лабораторной 08, упр3: В класс BankAccount требуется добавить в деструктор, метод Dispose.

Отсюда несколько вопросов:

Разве в С# существуют деструкторы? Там же только финализаторы, или я ошибаюсь? Можете уточнить куда нужно помещать метод Dispose(), по методичке не совсем понятно.

Если под деструктором понимается Финализатор, то разве корректно помещать метод Dispose в финализато. У них же вроде разная модель поведения?

**Здравствуйте, Антон.**

Лабораторная 9 упр1:

Просят откомпилировать один файл из всего проекта «Откомпилируйте файл null\_token\_visitor.cs», чтобы проверит его на синтаксические ошибки, как это сделать? В смысле отдельно от проекта, только один файл, чтобы не компилировать все остальные файлы и весь проект целиком.

Лабораторная 9 упр1:

Проделал всю последовательность, указанную в методичке, однако результата не получил, есть вопросы.

1) В качестве аргументов по методичке мы передаем строку:

generate token.cs > token.html

В методе InnerMain(string[] args), передаем ссылку на нулевой элемент в конструктор класса SourceFile. НО у класса SourceFile только один конструктор который принимает не массив, а одно слово, которое использует как имя файла. Как результат передается только слово generate из-за чего выбрасывается исключение, а если его убрать, то читается только файл token.cs, что собственно и требуется, но не полностью!

2) Все чтение и вывод осуществляется в консоль, нет ни одного метода который реализовывал запись в файл. При реализации всех методов Visit используется только Console.Write(). Соответственно весь текст отображается только в консоли и никакого файла не создается.

3) Нужный текст я вижу в консоли, а не в файле. Отсюда еще один очевидный вопрос. Предполагается что студент сам сделать парсер аргументов, затем реализует метод запись в файл, а после перепишет методы Visit, что бы они записывали текст не в консоль а в файл? Если да, то почему об этом не слово не сказано в методичке и описывается так словно это должно работать словно все это есть, хотя этого нет? Если нет, то что я упустил?

<https://github.com/GeorgiyIsaev/ITMO.CSCourse2021.Labs/tree/master/ITMO.CSCourse2021.Labs.Lab09.E1.CSInHtml>

**КОМЕНТАРИИ ПРЕПОДАВАТЕЛЯ К ВЫПОЛНЕННЫМ ЛАБОРАТОРНЫМ РАБОТАМ:**

1) Не рекомендуется разделять объявление и реализации переменной если она нигде более не используется:

Пример лаб01 (плохо):

string temp;

temp = Console.ReadLine();

Пример лаб01 (хорошо):

string temp = Console.ReadLine(); //Хорошо

2) Ситуации, потенциально взывающие исключение, необходимо обрабатывать. Если в выполняемой операции может произойти деление на ноль, следует поймать и обработать исключение DivideByZeroException. При обротке исключения рекомендуется предоставлять возможность перевызова метода его выбросившего.

3) Правила именования энументора гласят что содержащиеся в нем элементы должны начинаться с заглавной буквы: https://docs.microsoft.com/ru-ru/dotnet/standard/design-guidelines/naming-guidelines

Пример лаб01 (плохо):

public enum AccountType{ checking,deposit}

Пример лаб01 (хорошо):

public enum AccountType{ Checking,Deposit}

4) Если инициализация массива происходит сразу при объявлении, то необходимость указывать размер отсутствует.

Пример лаб03 (плохо):

var DaysInMonths = new int[12] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

Пример лаб03 (хорошо):

var DaysInMonths = new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

5) Закомментированный код, который более не будит использовать необходимо удалять сразу! Иначе технический долг!

6) В операторах ветвления нeследует использовать числа, так как это может привести к трудности его последующего чтения. При необходимости использования константных значений в сравнениях рекомендуется создавать отдельные константные переменные.

Пример лаб03 (плохо):

if (dayNum < 1 || dayNum > 365) {}

Пример лаб03 (хорошо):

const int maxDayYear = 365;

const int minDayYear = 1;

if (dayNum < minDayYear || dayNum > maxDayYear){}

7) Не стоит создавать отдельные логические переменные, если их сразу можно поместить в условие. Их создание допустимо при отладке, но с последующим возвращением на место.

Пример лаб04 (плохо):

bool ok = Utils.Factorial(x, out f);

if (ok){}

Пример лаб04 (хорошо):

if (Utils.Factorial(x, out f)){}

8) Названия методов, возвращающих результат успешности выполнения операции (удалось, не удалось) и возвращающих параметр out принято начать с приставки Try.

Пример лаб04 (плохо):

if (Utils.Factorial(x, out f)){}

Пример лаб04 (хорошо):

if (Utils.TryFactorial(x, out f))

9) Объекты реализующие интерфейс IDisposable необходимо помещать в конструкцию using(){}

Пример лаб05 (хорошо):

using (FileStream stream = new FileStream(fileName, FileMode.Open)){}

10) При взаимодействии с объектами предполагающие закрытие неких процессов по окончанию работы с ним, и использованием обработчика исключении, следует предусмотреть закрытие эти процессов при выбрасывании исключений. Иначе будит утечка памяти!

Пример лаб07 (хорошо):

try

{

srFrom = new StreamReader(sFrom);

swTo = new StreamWriter(sTo);

try

{

while (srFrom.Peek() != -1)

{

string sBuffer = srFrom.ReadLine();

sBuffer = sBuffer.ToUpper();

swTo.WriteLine(sBuffer);

}

}

finally{swTo.Close();srFrom.Close();}

}

catch{}

11) При обработке исключений следует проводить их диагностику и выводить информацию о пойманном исключении.

Пример лаб07 (плохо):

catch (FileNotFoundException)

{

Console.WriteLine("Input file not found");

}

Пример лаб07 (хорошо):

catch (FileNotFoundException e)

{

Console.WriteLine("Input file not found");

Console.WriteLine(e.ToString());

}

12) Интерфейс IDisposable предоставляет механизм для освобождения неуправляемых ресурсов через конструкцию using(объект){}. По окончанию вызова using(){} вызывается метод Dispose() интерфейса IDisposable. В случаи если объект не был помещен в using(){}, то при окончании использования этого объект он будит освобожден сборщиком мусора в случайное время. Сборщик мусора вызовет Финализатор объекта. Освобождение ссылочных типов не детерминировано.

Так как Финализатор и интерфейс IDisposable используют разные механики освобождения памяти не корректно помешать метод Dispose() в Финализатор. У него должна быть собственная реализация.

В примере лабораторной 08 метод Dispose() вызывает подавление Финализатора [GC.SuppressFinalize(this);]. В результате при вызове Финализатора происходит вызов метода Dispose(), который подавляет Финализатор, что собственно не имеет смысла!