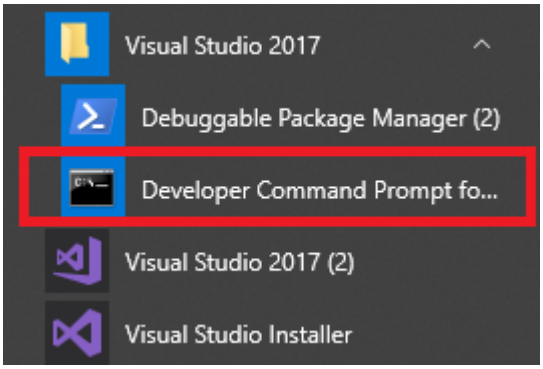
Задание к модулю 1

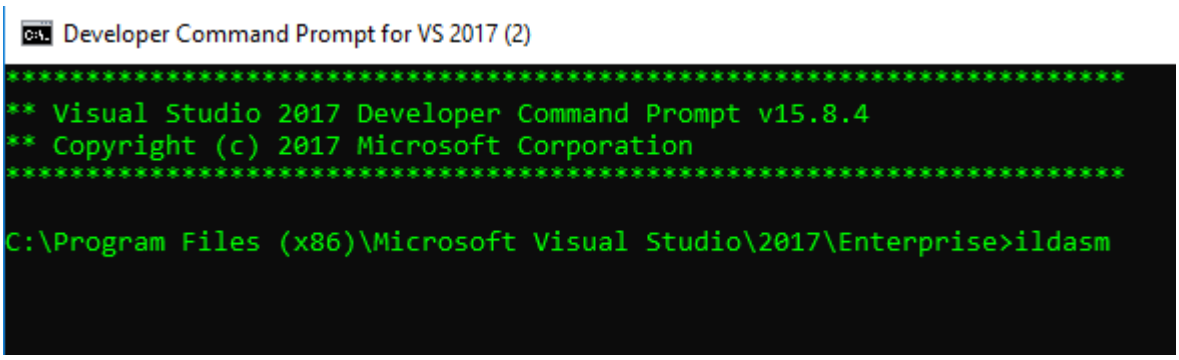
В этом задании предлагаю вам исследовать .NET сборки простейших программ с помощью утилиты ILDASM. Эта утилита используется для дизассемблирования .NET сборок и просмотра их метаданных (метаданных самой сборки - манифеста и метаданных типов).

Целью этой работы является расширение ваших знаний по устройству сборок .NET, знакомство с инструментом их исследования и приобретения понимания, что объявление в сборке любых типов (классов/структур) и членов в них (методов, свойств, событий и т.д.) после компиляции находит отражение в её метаданных .

Запустить утилиту ILDASM можно из командной строки разработчика Visual Studio, которую можно найти в меню «пуск» Windows:



После запуска командной строки, наберите команду **ildasm**



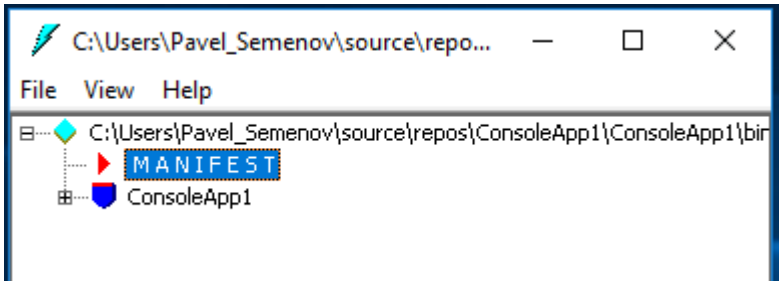
Для начала работы с утилитой ILDASM нужно сначала загрузить в нее сборку:

File -> Open.

В процессе выполнения задания вы будите здесь загружать *.exe или *.dll-файлы сборок.

Для просмотра манифеста сборки нужно:

1) Открыть сборку в утилите;
2) Выполнить мышью doubleclick по слову **MANIFEST**:

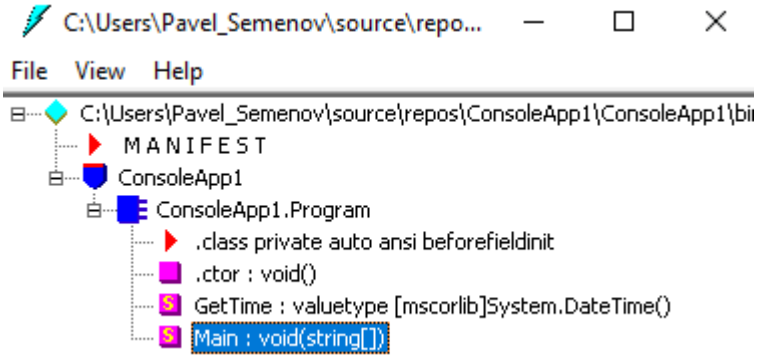


Для просмотра метаданных типов сборки нужно:

1) Открыть сборку в утилите;
2) Нажать комбинацию клавиш «Ctrl+M»

Для просмотра cil-кода нужно:

1) Открыть сборку в утилите;
2) Раскыть дерево типов сборки, а затем дерево интересующего типа;
3) Выбрать интересующий метод и выполнить мышью doubleclick по нему:

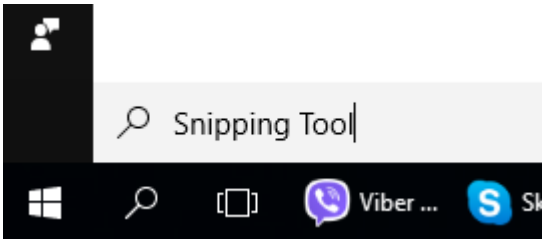


(После выполнения doubleclick по методу Main откроется окно, содержащее его cil-код.)

Вашей отчетностью по заданию к модулю 1 будет являться сделанный вами отчет (файл MS Word), содержащий скриншоты с пометками к каждому подзаданию и текстовыми пояснениями (при необходимости и желании).

Отчеты можете оформлять в произвольной форме и прислать мне почту или скайп.

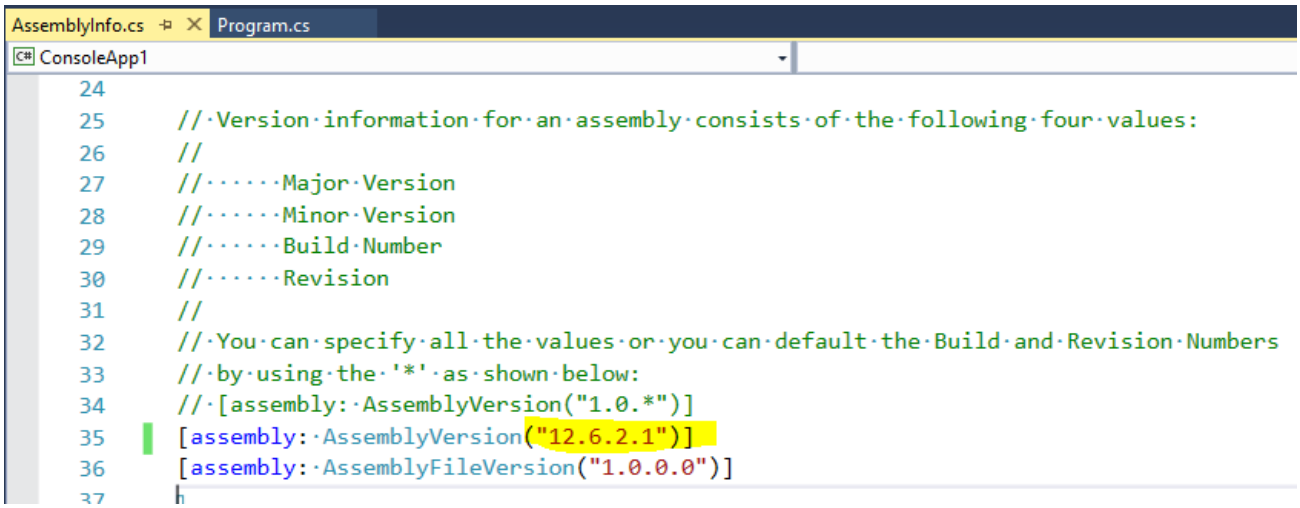
Для создания скриншотов ОС Windows имеет приложение «Snipping tool» («Ножницы»):



С помощью этой программы вы можете очень просто и быстро вырезать интересующие области экрана (кнопка «New»), а также делать пометки/выделения фона на полученном скриншоте (инструменты – “Pen” и “Highlighter”). Для последующего копирования скриншотов в отчет достаточно выполнить команды “Ctrl+C” и “Ctlr+V” без их сохранения в отдельные файлы.

Задание 1 «Hello world console application»

1) В среде Visual Studio создайте простейшее, косольное приложение, выводящее в консоль строку «Hello world».
2) Найдите в проекте файл AssemblyInfo и присвойте вашей сборке номер, например 12.6.2.1:



3) Исследуйте метаданные полученной сборке с помощью ILDASM
   - Сделайте скриншорт манифеста сборки
   - На скриншоте выделите с помощью инструмента «Pen»:
   a) описание внешних сборок (по ключевому слову **extern**), на которые ссылается ваша сборка и их имена;
   b) описание вашей сборки, в котором выделите целевую платформу (Target Framework) и ее версию, также найдите и выделите версию вашей сборки, которую вы задавали при создании приложения (12.6.2.1).
   - Откройте метаданные типов сборки (Ctrl+M) и найдите по ключевому слову **TypeDef** – типы, определенные в вашей сборке и по слову **TypeRef** – типы, на которые ссылается ваша сборка.

a) Сделайте скриншот метаданных вашего типа (class Program) и на нём выделите метод Main (вы его найдете по ключевому слову **ENTRYPOINT**) и метод-конструктор класса (по ключевому .ctor).

Метод Main на платформе .NET является точкой входа и может содержаться только в исполняемых сборках (.exe файлах). Работа любого приложения начинается с поиска точки входа и его запуска. Сборки-библиотеки типов (*.dll) не имеют точки входа, поэтому их нельзя запустить как *.exe файл. Для их использования они подключаются к исполняемым сборкам (загружаются в память).*

Скопируйте скриншоты с пометками в ваш отчет.

Сделайте для себя выводы относительно содержимого метаданных в сборке .NET.

Задание 2 «Method GetCurrentTime()»

Добавьте в класс Program новый статический метод GetCurrentTime и внутри метода Main выведите в консоль результат метода GetCurrentTime:

```
0 references
class Program
{
    0 references
    static void Main(string[] args) {
        Console.WriteLine(GetCurrentTime());
    }

    1 reference
    static DateTime GetCurrentTime() {
        return DateTime.Now;
    }
}
```

Сбилдите приложение и снова отройте его сборку в утилите ILDASM.

Просмотрите метаданые типов (Ctrl+M). Найдите метаданные типа, определенного в сборке (класса Program) и сделайте его скриншот. На скриншоте найдите и выделите метаданные метода GetCurrentTime. Скопируйте скриншот в отчет. Сделайте для себя вывод, как изменяются метаданные типа при добавлении в него новых методов, полей и свойств (можете с этим поэкспериментировать).

Задание 3 «Custom DLL»

Добавьте в солюшен новый проект типа “Class Library”, имя проекта можно оставить по умолчанию. Затем добавьте в проект класс – TimeService, со следующим кодом:

```
namespace ClassLibrary1
{
    0 references
    public class TimeService
    {
        0 references
        public DateTime GetCurrentTime() {
            return DateTime.Now;
        }
    }
}
```

Добавьте ссылку на проект ClassLibrary к основному приложению (ConsoleApp -> References -> Add reference ->Projects).

Из метода Main удалите определение метода GetCurrentTime, а вместо него вызовите одноименный метод из класса TimeService кастомной dll:

```
using ClassLibrary1;
using System;

namespace ConsoleApp1
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args) {
            var timeService = new TimeService();

            Console.WriteLine(timeService.GetCurrentTime());
        }
    }
}
```

Сбилдите солюшен и откройте утилитой ILDASM сборку консольного приложения (*.exe) для исследования. Откройте манифест сборки и сделайте его скриншот, на скриншоте найдите и выделите ссылку на сборку нашей кастомной DLL (по ключевому слову **extern**). Сделайте для себя выводы, где в метаданных сборки хранятся ссылки на внешние библиотеки Dll. Скопируйте скриншот в отчет.

Задание 4 «Debug vs Release»

Создайте новое консольное приложение, в методе Main которого выведите на консоль произведение чисел i*j, где i, j = 0,1, …9:

```
0 references
class Program
{
    0 references
    static void Main(string[] args) {
        for(var i=0; i<10; i++) {
            for(var j=0; j<10; j++) {
                Console.WriteLine($"{i}*{j}={i * j}");
            }
        }
    }
}
```

Сбилдите это приложение в двух конфигурациях поочередно – Debug и Release. Соответствующие *.exe файлы приложений будут храниться с соответствующих папках:
...\ConsoleApp2\ConsoleApp2\bin\Debug\

...\ConsoleApp2\ConsoleApp2\bin\Release\

Версия debug обычно используется девелоперами в процессе разработки и отладки приложения, а версия release – это уже конечная (продакшн) версия программного продукта (которая в случае веб-приложений деплоится на продакшн сервера).

Откройте поочередно файлы сборок в ILDASM и посмотрите на cil-код метода Main.

Сделайте скриншоты cil-кода метода Main каждой версии (debug и release), сравните их между собой по числу операций. Найдите вверху метода и выделите на каждом скриншоте параметр **«Code size»** (за комментарием). Сделайте для себя выводы, как изменяется работа компилятора C# при изменении конфигурации Debug/Release. Скопируйте скриншоты в отчет.