Лабораторная работа №1. Исследование .Net сборок с использованием системных средств

Цель:

1) Научиться выполнять основные операции по работе с кодом сборки стандартными средства Visual Studio.
2) Ознакомиться с синтаксисом языка MSIL на базовом уровне.

Инструменты: компилятор CSC, дизассемблер ILDASM, ассемблер ILASM, утилита PEVERIFY.

Задание 1. Создание и дизассемблирование сборки

1. Создайте в текстовом редакторе новый файл кода C# ***HelloProgram.cs*** и определите в нем показанный ниже класс[1]:

```
using System;

namespace HelloProgram
{
   class Program
   {
      static void Main(string[] args)
      {
         string myMessage = "Hello CIL code!";
         Console.WriteLine(myMessage);
         Console.ReadLine();
      }
   }
}
```

2. Откройте в Visual Studio окно «Командная строка разработчика», перейдите в каталог с созданным файлом *.cs и скомпилируйте приложение с помощью компилятора C# **csc.exe**:

>csc *HelloProgram.cs*.

Полученный в результате компиляции файл приложения ***HelloProgram.exe*** является сборкой .Net, состоящей из одного функционального модуля — класса.

3. Просмотрите содержимое сборки, воспользовавшись дизассемблером **ildasm.exe** из состава Visual Studio.

Чтобы запустить утилиту **ildasm.exe**, достаточно в окне «Командная строка разработчика» ввести команду:

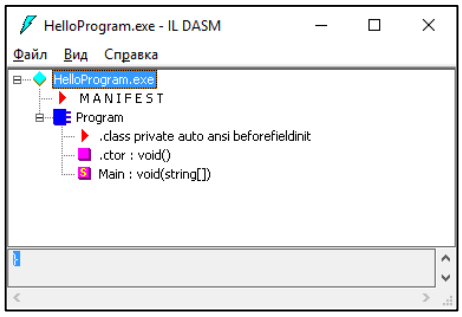
>ildasm *HelloProgram.exe*.

Сборка содержит

- **Манифест** – позволяет получить информацию о том, на какие внешние сборки вызываться из текущей сборки и структуру модулей текущей сборки.

[1] При желании можно создать новое консольное приложение в Visual Studio.

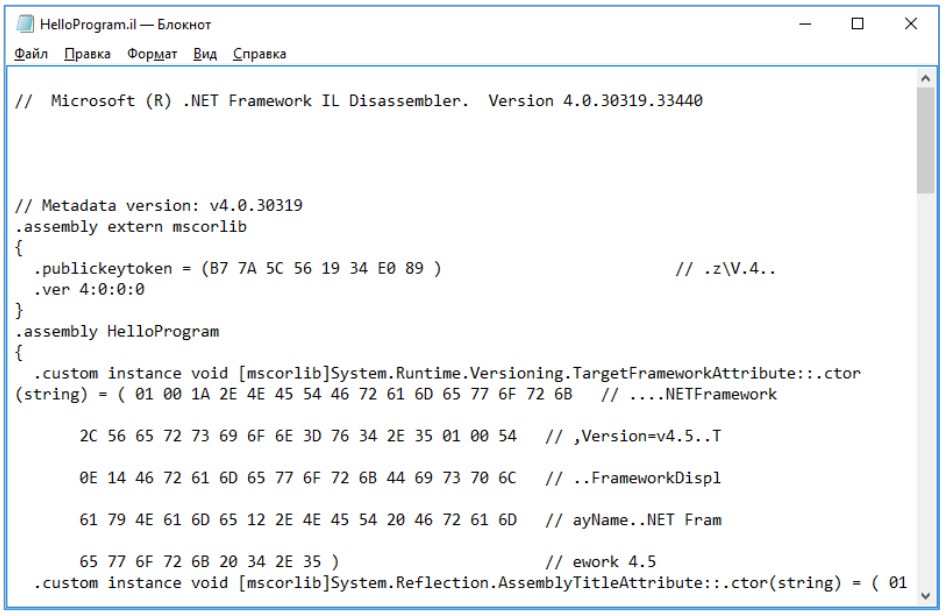
- **Метаданные типов** – содержит сведения о типах данных, используемых в текущей сборке.
- **Код MSIL** - MSIL -инструкции для конкретного члена сборки.



4. Просмотрите содержимое манифеста, дважды щелкнув по значку **MANIFEST**; метаданных типов, нажав комбинацию клавиш <**Ctrl+M**>; код методов программы, дважды щелкнув по названиях методов.
5. Сохраните IL –код сборки в новом файле *.il (***HelloProgram.il***), воспользовавшись командой дизассемблера **Файл-Дамп**.

Задание 2. Анализ кода сборки

1. Откройте файл кода ***HelloProgram.il*** в любом текстовом редакторе:



2. Ознакомьтесь с объявлением внешних сборок. Их перечень указывается в начале файла кода с помощью лексемы **.assembly extern**. Так, запись **.assembly extern mscorlib** указывает на постоянно присутствующую сборку **mscorlib.dll**.
3. Ознакомьтесь с определением текущей сборки ***HelloProgram.exe***. Оно располагается сразу после объявления внешних сборок и начинается с записи **.assembly HelloProgram**. Сборка описывается с применением различных директив IL , таких как **.module**, **.imagebase** и т.п.
4. Ознакомьтесь с определением типа **Program**. Оно располагается вначале секции **CLASS MEMBERS DECLARATION** (сразу после определения текущей сборки) и начинается с директивы **.class**.
5. Найдите в определении типа **Program** код реализации методов класса. Они определяются (частично) с помощью директивы **.metod**.
6. Проанализируйте IL-код метода Main():

```
.method private hidebysig static void  Main(string[] args) cil managed
```

```
{
  .entrypoint
  // Размер кода:       21 (0x15)
  .maxstack  1
  .locals init (string V_0)
  IL_0000:  nop
  IL_0001:  ldstr      "Hello CIL code!"
  IL_0006:  stloc.0
  IL_0007:  ldloc.0
  IL_0008:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_000d:  nop
  IL_000e:  call       string [mscorlib]System.Console::ReadLine()
  IL_0013:  pop
  IL_0014:  ret
} // end of method Program::Main
```

При анализе следует учесть, что в MSIL невозможно получать доступ к элементам данных, включая локально определенные переменные, входные аргументы методов и данные полей типа. Вместо этого элемент данных должен быть явно загружен в стек и затем извлекаться оттуда для последующего использования.

Ячейка для хранения локальной переменной определяется с помощью директивы **.locals**. Локальная строка затем загружается и сохраняется в этой локальной переменной с использованием кодов операций **ldstr** (загрузка строки) и **stloc.0** (сохранение текущего значения в локальной переменной, находящейся в ячейке 0). После этого значение (по индексу 0) загружается в память с помощью кода операции **ldloc.0** (загрузка локального аргумента по индексу 0) для применения в вызове метода **System.Console.WriteLine ()** (представленного кодом операции **call**). И, наконец, возврат из функции обеспечивается посредством кода операции **ret**.

Каждая строка в коде реализации предваряется лексемой в форме IL_XXX: (например, IL_0000 :, IL_0001: и т.д.). Такие лексемы называются *метками кода*. При сбросе содержимого сборки в файл утилита **ildasm.exe** автоматически генерирует метки кода, которые следуют соглашению об именовании IL_XXX: (но могут быть переименованы в произвольной манере при условии, что они не дублируются в рамках одного и того же члена). В действительности метки кода по большей части являются не обязательными. Единственный случай, когда метки кода совершенно необходимы, связан с написанием IL-кода, в котором используются различные конструкции ветвления или зацикливания.

Важно помнить, что при взаимодействии с типами .NET (такими как **System.Console**) в MSIL всегда должно применяться полностью заданное имя нужного типа. Кроме того, полностью заданное имя типа всегда должно снабжаться префиксом в форме дружественного имени сборки, в которой тип определен (в квадратных скобках). Например: [**mscorlib]System.Console::WriteLine(string)**.

7. Проанализируйте IL-код стандартного конструктора класса:

```
.method public hidebysig specialname rtspecialname
        instance void  .ctor() cil managed
```

```
{
  // Размер кода:       7 (0x7)
  .maxstack  8
  IL_0000:  ldarg.0
  IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
  IL_0006:  ret
} // end of method Program::.ctor
```

Конструктор определяется с помощью директивы **.ctor**.

В реализации конструктора используется еще одна инструкция, связанная с загрузкой — **ldarg.0**. В этом случае значение, загружаемое в стек, представляет собой не специальную переменную, а ссылку на текущий объект.

Также обратите внимание, что в стандартном конструкторе явно производится вызов конструктора базового класса, которым в данном случае является хорошо знакомый класс **System.Object**.

Задание 3. Модификация кода сборки

Выполните модификацию кода в существующем файле ***HelloProgram.il*** как описано ниже:

- добавьте ссылку на сборку **System.Windows.Forms.dll**;
- загрузите локальную строку внутри метода **Main ()**;
- вызовите метод **System.Windows.Forms.MessageBox.Show ()**, используя локальную строковую переменную в качестве аргумента.

Для выполнения первой части задания необходимо вставить в файл *. il сразу же после ссылки на внешнюю сборку **mscorlib** следующий код:

```
.assembly extern System.Windows.Forms
{
.publickeytoken = (B7 7A 5C 56 19 34 E0 89)-
.ver 4:0 : 0 : 0
}
```

Значение, присваиваемое директиве **.ver**, может отличаться в зависимости от версии платформы .NET, установленной на машине разработки.

Для выполнения оставшихся частей задания необходимо изменить реализацию метода **Main ()** следующим образом:

```
.method private hidebysig static void Main(string[] args) cil managed
{
.entrypoint
.maxstack 8
ldstr "CIL is way cool"
call valuetype [System.Windows.Forms] System.Windows.Forms.DialogResult
[System.Windows.Forms]System.Windows.Forms.MessageBox::Show(string)
pop
ret
```

```
}
```

Задание 4. Рекомпиляция модифицированного кода сборки

1. Рекомпилируйте измененный файл ***HelloProgram.il*** в новую сборку ***.exe**, введя в командной строке разработчика следующую команду:

>ilasm /exe *HelloProgram.il* /output=*NewAssembly.exe*

Здесь ключ **/exe** позволяет указать тип генерируемого файла (*.exe или *.dll), а **/output** –указать имя и расширение выходного файла.

Будьте осторожны: если ключ **/output** не указан, данные перезаписываются в исходный файл!

2. Проверьте, является ли скомпилированный двоичный образ правильно оформленным образом .Net, с помощью утилиты командной строки peverify.exe:

> peverify *NewAssembly.exe*

Эта утилита проверит достоверность всех кодов операций CIL внутри указанной сборки.

3. Убедитесь в работоспособности новой сборки, запустив на выполнение файл ***NewAssembly.exe.***

Запуск нового приложения приводит к отображению диалога с измененным текстом.

Задание 5. Нейтрализация простой парольной защиты

1. Создайте в Visual Studio прототип консольного приложения, защищенного паролем от запуска. При запуске такого приложения должен запрашиваться пароль. В случае ввода пользователем правильного пароля выводится поздравление, в противном случае осуществляется выход из приложения.
2. Дезасемблируйте приложение с помощью ILDASM.
3. Добавьте в MSIL -код комментарии, поясняющие работу программы
4. Отредактируйте MSIL -код таким образом, чтобы пароль не запрашивался.
5. Рекомпилируйте модифицированный код с помощью ILASM.
6. Проверьте правильность оформления сборки с помощью утилиты PEVERIFY.

Требование к отчету по лабораторной работе

Отчет должен содержать титульный лист с указанием темы и цели работы, результаты выполнения задания 5, а также выводы по работе.

Контрольные вопросы

1. Что называется сборкой .Net?
2. Какие возможности для компиляции приложений предоставляет Visual Studio?
3. Каково назначение утилиты **ildasm.exe** из состава Visual Studio?
4. Какие сведения содержатся в манифесте сборки?
5. Как отобразить метаданные типов сборки?
6. Как выгрузить MSIL –код сборки в текстовый файл?
7. Как в MSIL объявляются внешние сборки?
8. Как в MSIL реализуется стековый доступ к данным?
9. Когда использование меток кода в MSIL являются обязательным?
10. Как в MSIL выполняется описание класс?
11. Какая директива MSIL позволяет создать конструктор класса?
12. Как в MSIL описывается метод класса?
13. Что такое «рекомпиляция» и как она выполняется?

14. Каково назначение утилиты PEVERIFY?