

# 제5장

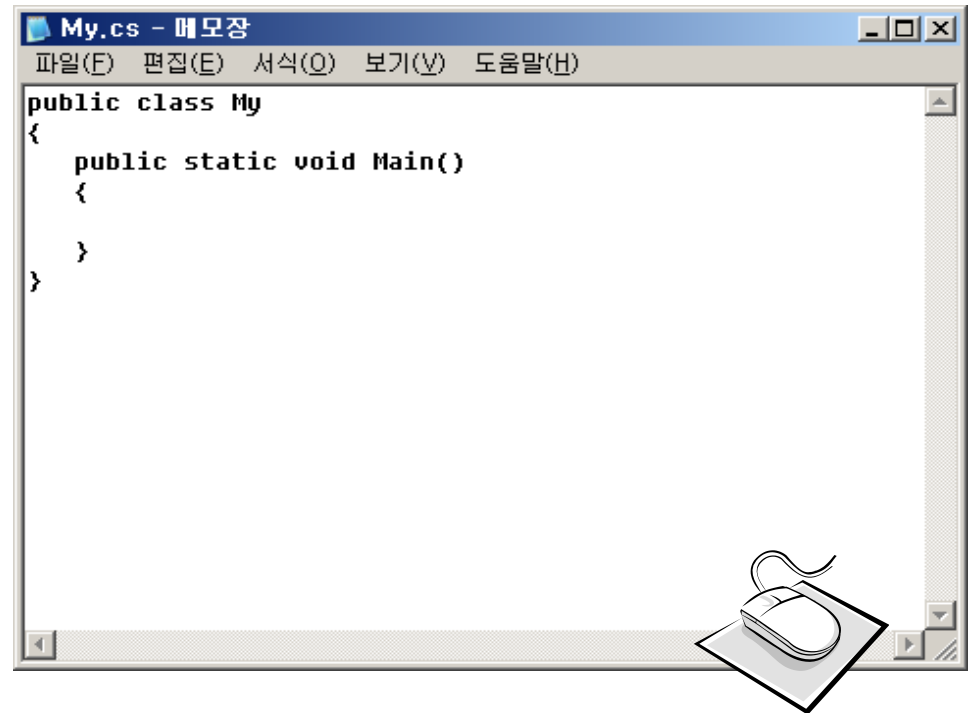
## 닷넷 기본과 어셈블리

제주대학교 컴퓨터공학과  
변영철 교수

# 1. 어셈블리 이해하기

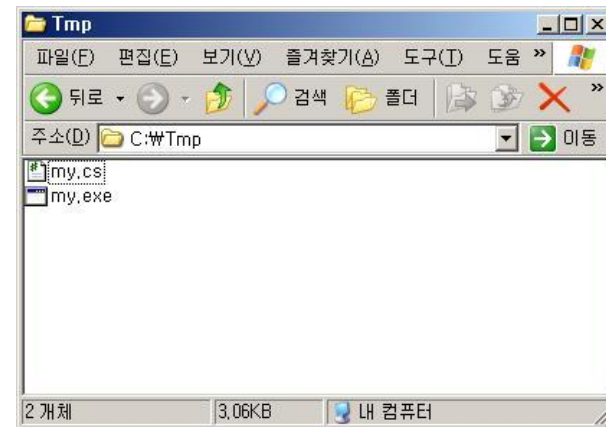
- 간단한 C# 프로그램 작성하기

```
public class My
{
    public static void Main()
    {
    }
}
```



# 1. 어셈블리 이해하기

- 환경설정
  - 컴파일러 csc.exe가 있는 다음 폴더를 PATH에 추가함
    - C:\WINDOWS\Microsoft.NET\Framework\v3.5
- 컴파일 및 실행
  - C:\Tmp>csc my.cs



# 1. 어셈블리 이해하기

- **my.exe**

- 닷넷 환경에서 실행되는 파일
- 기존의 실행 파일(가령 hwp.exe)과 완전히 다름
- **어셈블리**라고 함 (왜?)

- 기존 실행 파일의 문제점

- 가령 hwp.exe, 라이브러리 dll 파일 등
- 다른 OS(가령 유닉스)에서 실행 불가
- dll을 이용하려면 헤더 파일(가령 C 라이브러리 dll을 사용하려면 stdio.h가 반드시 있어야)

# 1. 어셈블리 이해하기

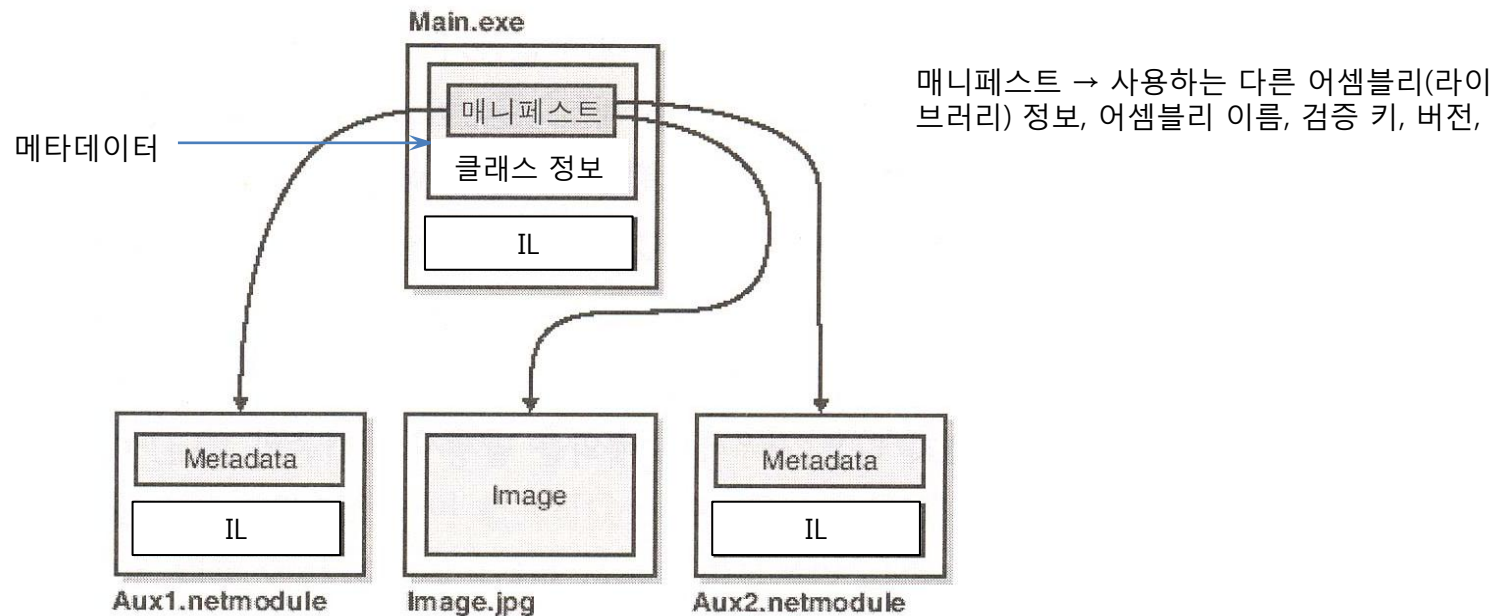
- 어셈블리
  - **중간 형태**의 언어인 IL(Intermediate Language)로 만듦으로써 런타임이 설치되어 있기만 하면 다른 플랫폼(IBM PC, 맥 컴퓨터, 유닉스 머신 등)에서도 실행 가능
  - 어셈블리 = 실행 코드(IL) + 관련 정보(metadata)를 하나의 파일로 합친 것
  - 즉, 어셈블리(assembly)는 '**한데 모은 것**'이라는 의미로 붙여진 것
  - 따라서 헤더 파일이 따로 없어도 됨

# 1. 어셈블리 이해하기

- 어셈블리 특징
  - 설치 및 배포를 위한 것
  - exe, dll, 리소스 파일 등 모두 어셈블리
  - 닷넷 응용 프로그램 짜는 것 = 어셈블리를 만드는 것
  - 메타데이터 = 매니페스트(manifest) + 클래스 정보
  - 매니페스트 = 사용하는 어셈블리 리스트, 어셈블리 이름, 버전 번호 등

# 1. 어셈블리 이해하기

어셈블리 =  
메타 데이터 + 실행 코드(중간 형태 언어,  
Intermediate Language)



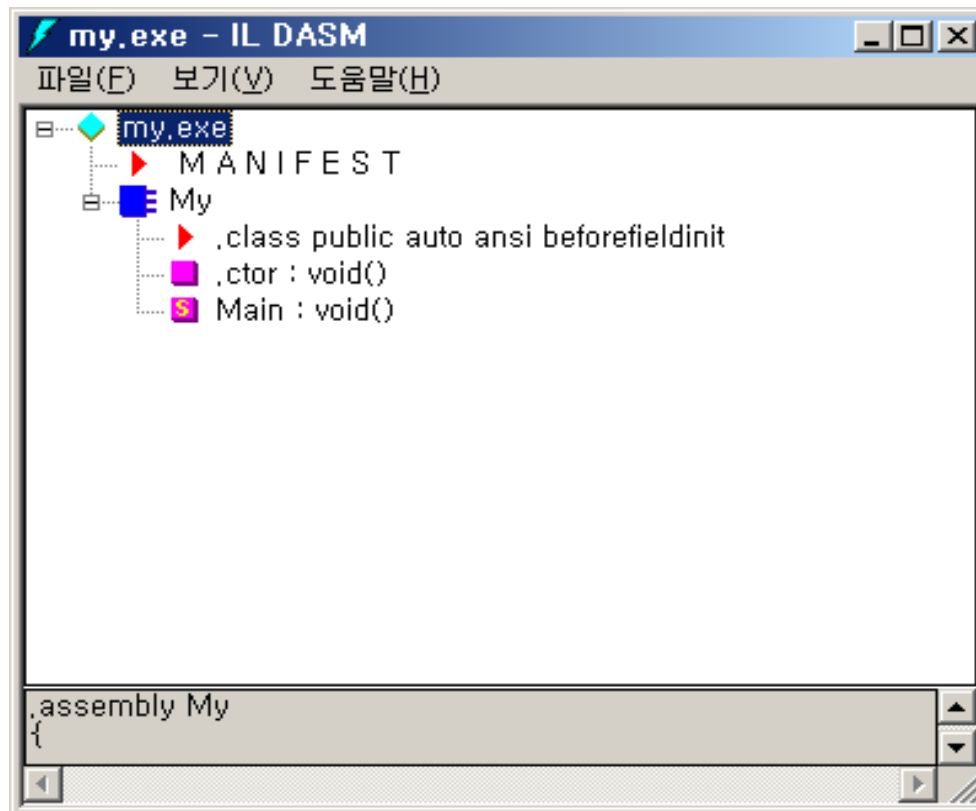
## 2. 어셈블리 분석하기

- ildasm.exe
  - 어셈블리 분석 도구
  - 어셈블리(my.exe)에 들어있는 IL과 메타데이터를 사람이 읽을 수 있도록 보여줌
  - C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin 폴더에 있음  
(PATH에 추가)



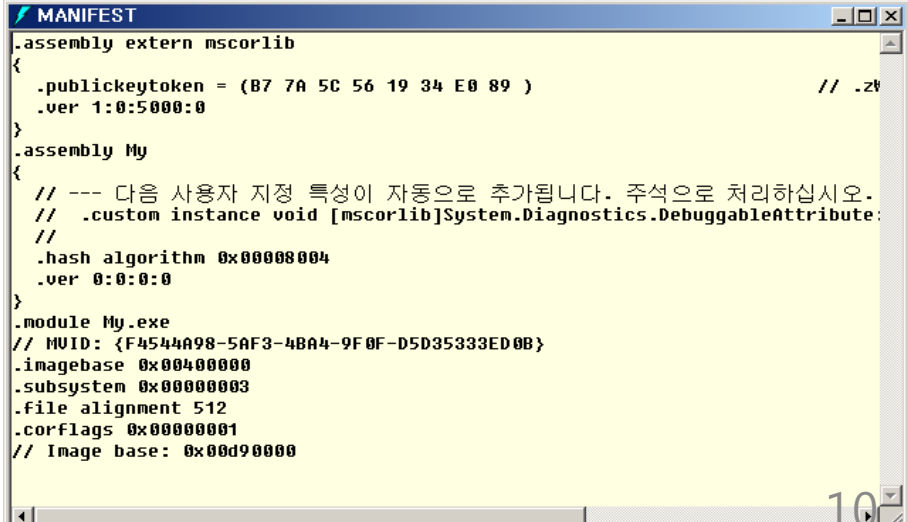
## 2. 어셈블리 분석하기

- C:\W\Tmp>ildasm my.exe



## 2. 어셈블리 분석하기

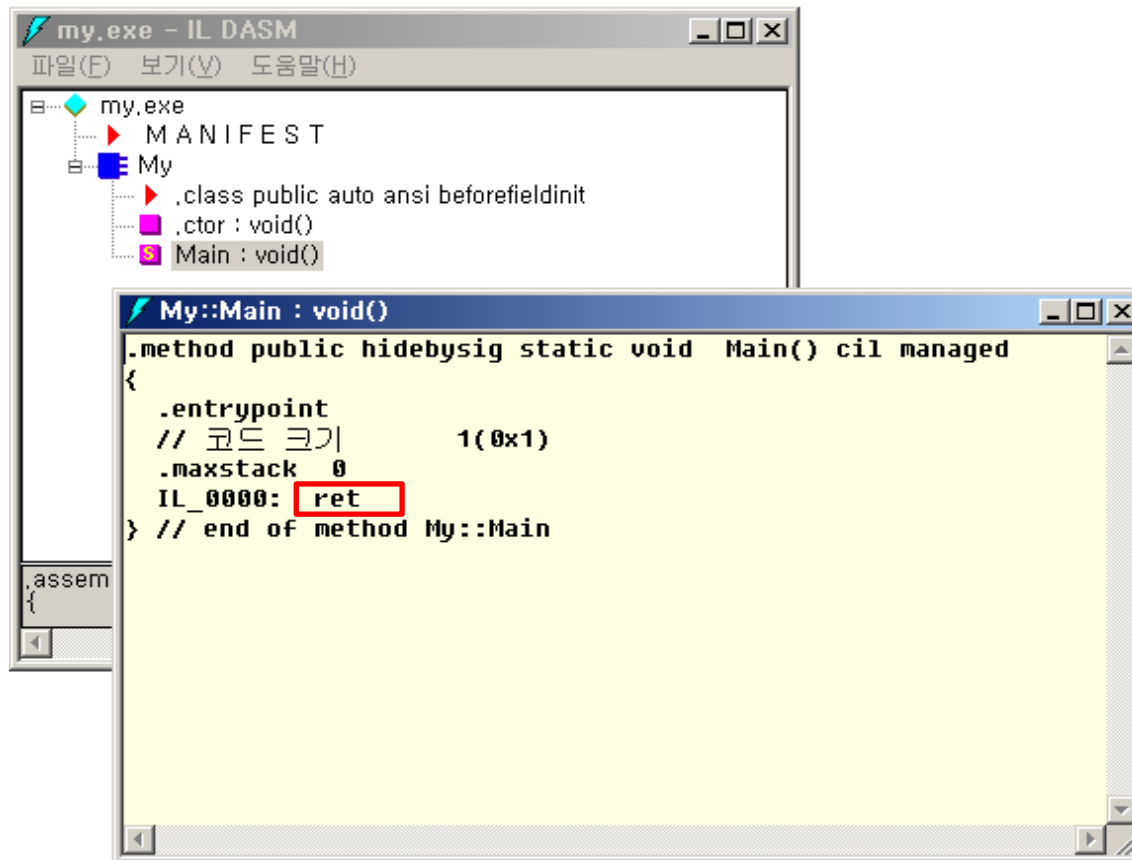
- 매니페스트 확인해 보기
  - 어셈블리 이름(My), 어셈블리 버전(0:0:0:0), 참조하는 다른 어셈블리(mscorlib) 정보
  - mscorlib 어셈블리에 publickeytoken : mscorlib 어셈블리를 인증하기 위한 키로서 이름은 mscorlib로 같지만 전혀 다른 어셈블리가 사용되는 것을 방지



```
MANIFEST
[assembly extern mscorlib]
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )           // .NET
  .ver 1:0:5000:0
}
[assembly My]
{
  // --- 다음 사용자 지정 특성이 자동으로 추가됩니다. 주석으로 처리하십시오.
  // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute:
  //
  .hash algorithm 0x00000004
  .ver 0:0:0:0
}
.module My.exe
// MVID: {F4544A98-5AF3-4BA4-9F0F-D5D35333ED0B}
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
// Image base: 0x00d90000
```

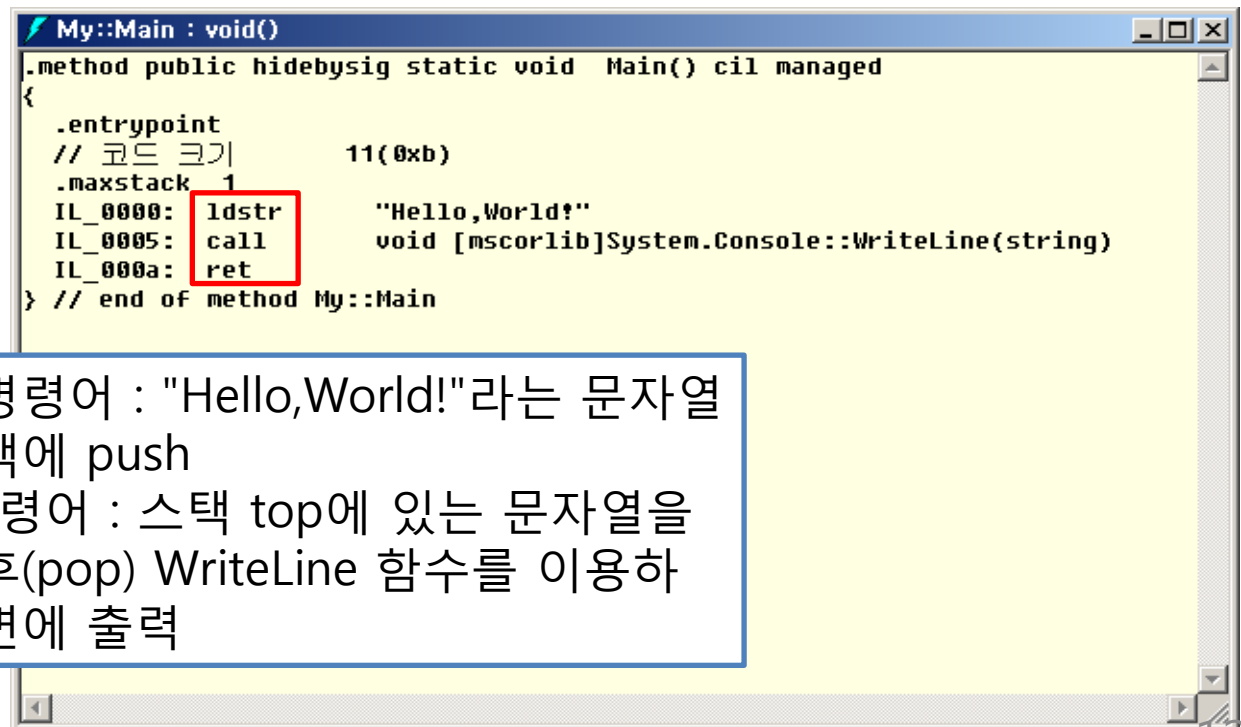
## 2. 어셈블리 분석하기

- IL 코드 보기



## 2. 어셈블리 분석하기

```
class My
{
    public static void Main()
    {
        System.Console.WriteLine("Hello,World!");
    }
}
```



```
My::Main : void()
.method public hidebysig static void Main() cil managed
{
    .entrypoint
    // 코드 크기      11(0xb)
    .maxstack 1
    IL_0000: ldstr      "Hello,World!"
    IL_0005: call      void [mscorlib]System.Console::WriteLine(string)
    IL_000a: ret
} // end of method My::Main
```

1. ldstr 명령어 : "Hello,World!"라는 문자열을 스택에 push
2. call 명령어 : 스택 top에 있는 문자열을 꺼낸 후(pop) WriteLine 함수를 이용하여 화면에 출력

### 3. DLL 어셈블리 분석하기

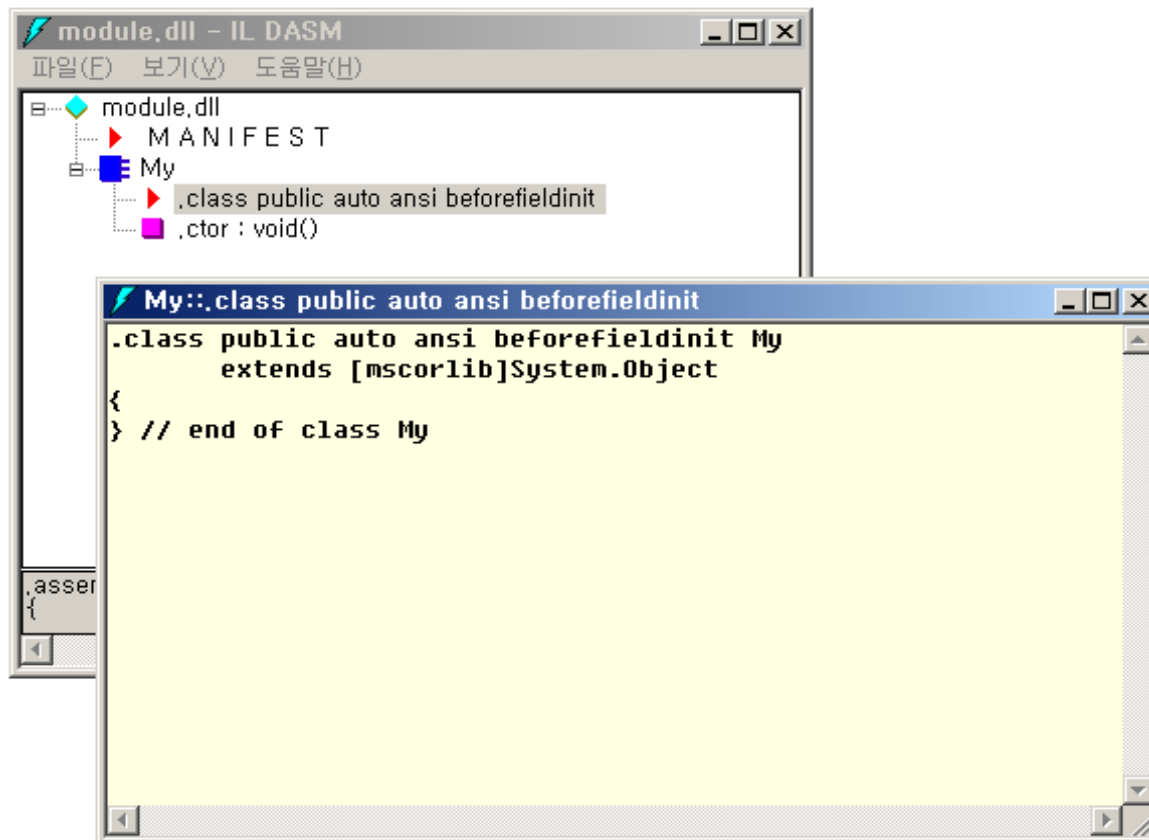
- DLL 만들기 (Main 함수가 없음)

```
//-----  
// Module.cs  
//-----  
public class My  
{  
    public My()  
    {  
    }  
}
```

```
C:\Wtmp>csc /t:library module.cs
```

### 3. DLL 어셈블리 분석하기

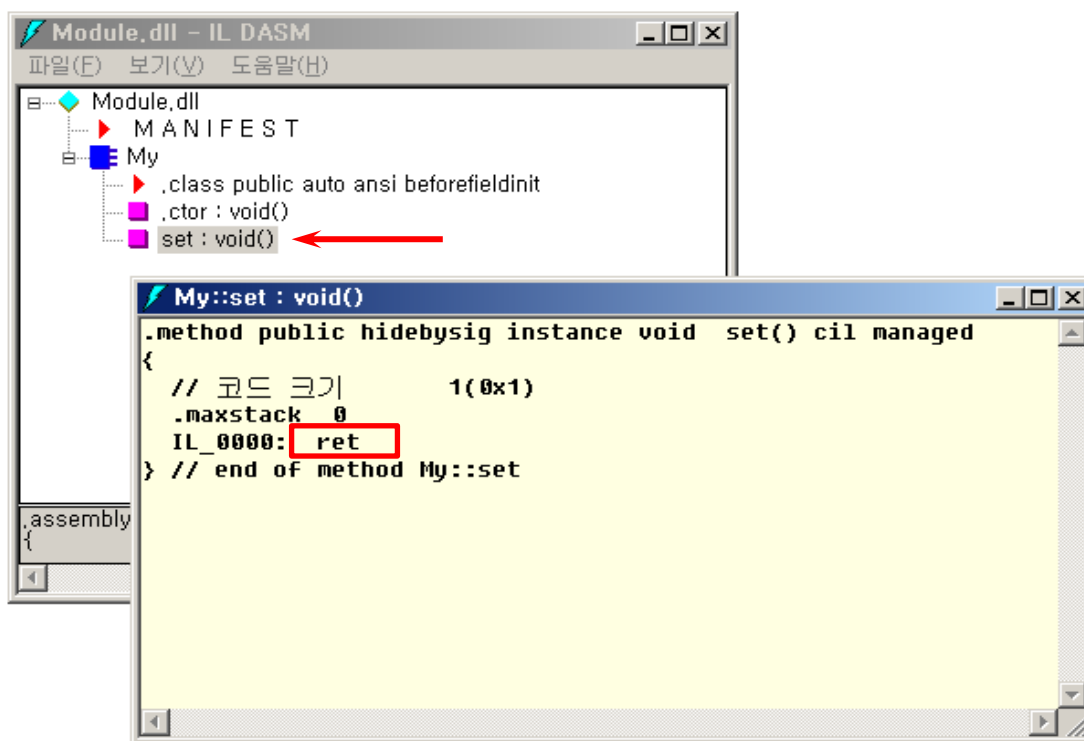
- ildasm으로 분석하기



### 3. DLL 어셈블리 분석하기

- 멤버함수 선언

```
//-----  
// Module.cs  
//-----  
public class My  
{  
    public My()  
    {  
    }  
  
    public void set()  
    {  
    }  
}
```

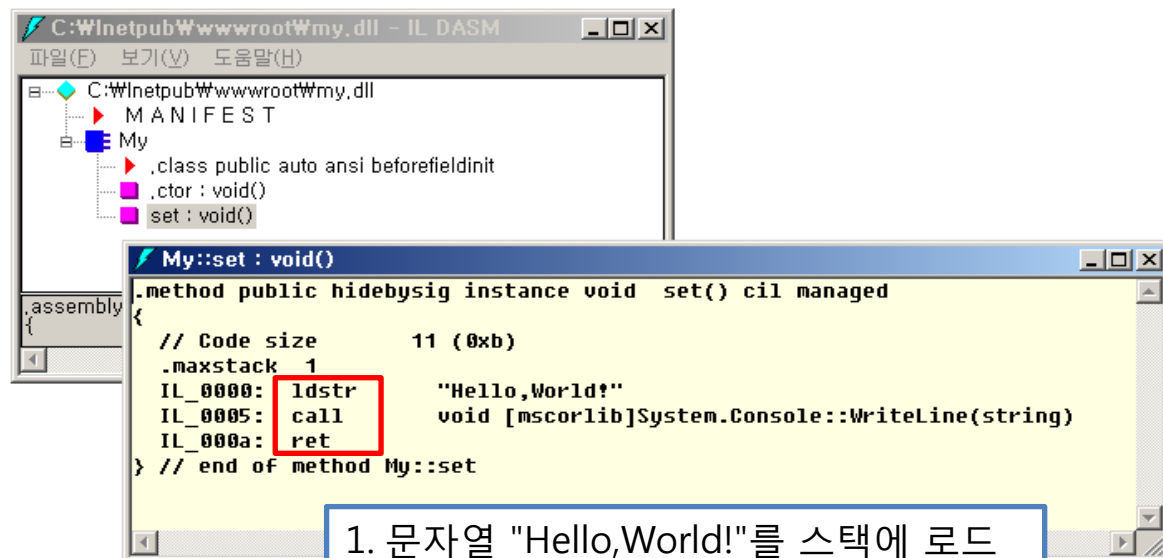


# 3. DLL 어셈블리 분석하기

- 문자열 출력

```
//-----  
// Module.cs  
//-----
```

```
public class My  
{  
    public My()  
    {  
  
    }  
  
    public void set()  
    {  
        System.Console.WriteLine("Hello,World!");  
    }  
}
```



1. 문자열 "Hello,World!"를 스택에 로드 (push)한 후(ldstr)
2. 닷넷 프레임워크에 있는 기본 클래스 라이브러리(Basic Class Library) Console의 WriteLine 함수를 호출(call)하여 문자열을 출력. 이때 WriteLine 함수는 스택 top에 있는 문자열, 즉, "Hello,World!"를 꺼내어(pop) 화면에 출력



# 3. DLL 어셈블리 분석하기

## • 멤버변수 선언

```
using System;
```

```
public class My  
{
```

```
    private int iNumber;
```

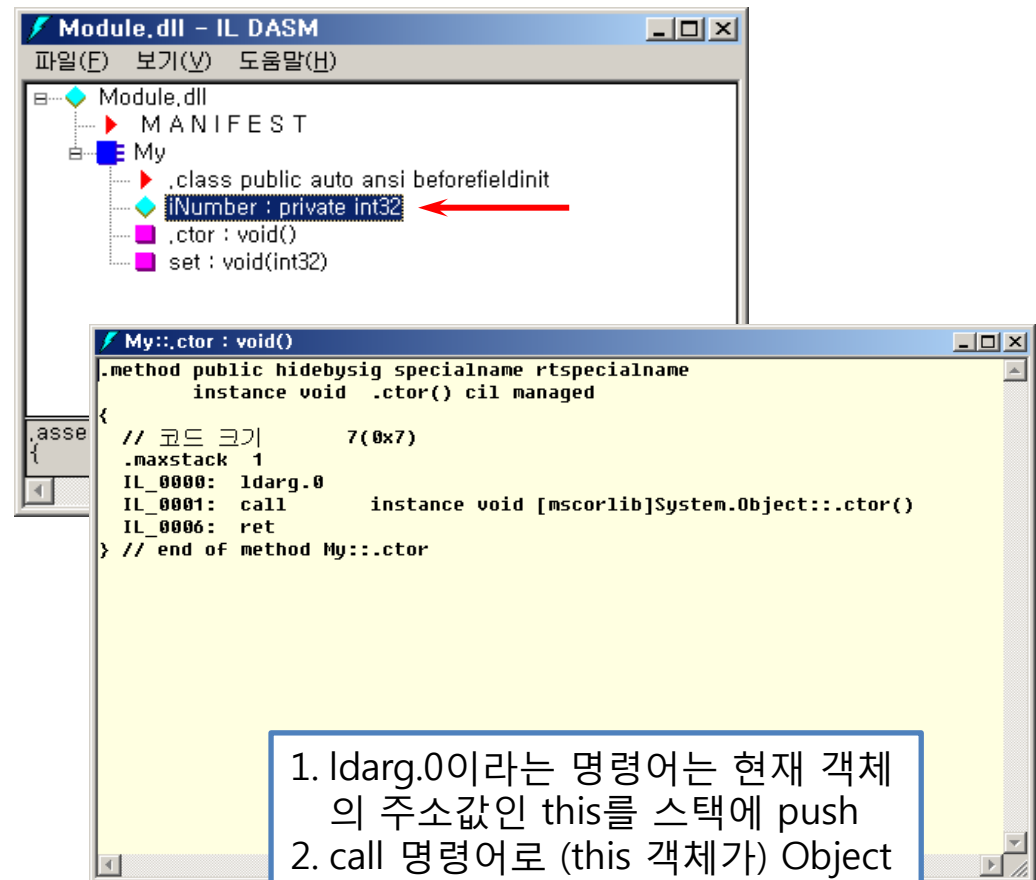
```
    public My()  
{
```

```
}
```

```
    public void set()  
{
```

```
}
```

```
}
```



1. ldarg.0이라는 명령어는 현재 객체의 주소값인 this를 스택에 push
2. call 명령어로 (this 객체가) Object 클래스로부터 상속받은 생성자 함수를 호출

# 3. DLL 어셈블리 분석하기

- 멤버변수 초기화

```
using System;
```

```
public class My  
{
```

```
    private int iNumber = 0;
```

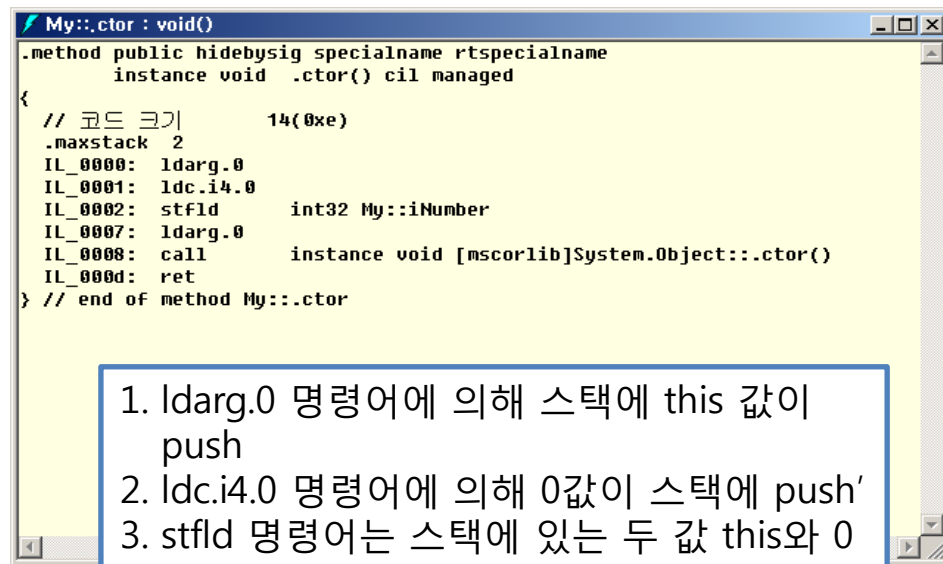
```
    public My()  
{
```

```
}
```

```
    public void set()  
{
```

```
}
```

```
}
```



```
My::.ctor : void()  
.method public hidebysig specialname rtspecialname  
    instance void .ctor() cil managed  
{  
    // 코드 크기      14(0xe)  
    .maxstack 2  
    IL_0000: ldarg.0  
    IL_0001: ldc.i4.0  
    IL_0002: stfld      int32 My::iNumber  
    IL_0007: ldarg.0  
    IL_0008: call      instance void [mscorlib]System.Object::.ctor()  
    IL_000d: ret  
} // end of method My::.ctor
```

1. ldarg.0 명령어에 의해 스택에 this 값이 push
2. ldc.i4.0 명령어에 의해 0값이 스택에 push'
3. stfld 명령어는 스택에 있는 두 값 this와 0을 pop하여 this.iNumber 멤버 변수에 0값을 저장. 즉, 초기화 수행.

# 3. DLL 어셈블리 분석하기

- 매개변수 선언

```
using System;

public class My
{
    private int iNumber = 0;

    public My()
    {

    }

    public void set(int i)
    {

    }
}
```

```
.method public hidebysig instance void set(int32 i) cil managed
{
    // 코드 크기      1(0x1)
    .maxstack 0
    IL_0000: ret
} // end of method My::set
```

# 3. DLL 어셈블리 분석하기

- 코드 추가

```
using System;
```

```
public class My  
{
```

```
    private int iNumber = 0;
```

```
    public My()  
{
```

```
}
```

```
    public void set(int i)  
{
```

```
        iNumber = i;
```

```
}
```

```
}
```

실제로 객체를 생성하여 호출하는 코드의 예

```
My gildong = new My();  
gildong.set(7);
```

내부적으로 set 함수가 호출되는 모양

```
set(this, 7);
```

즉, 현 객체 gildong의 주소값(숨겨진 포인터 this)이 0번째 인자로 자동으로 전송

```
.method public hidebysig instance void set(int32 i) cil managed  
{  
    // 코드 크기      8(0x8)  
    .maxstack 2  
    IL_0000: ldarg.0 //0번째 인자(숨겨진 포인터 this)를 스택에 로드함(push)  
    IL_0001: ldarg.1 //1번째 인자 i에 있는 값을 스택에 로드함  
    //this 객체의 멤버 iNumber에 1번 인자의 값을 저장함  
    IL_0002: stfld    int32 My::iNumber  
    IL_0007: ret  
} // end of method My::set
```

### 3. DLL 어셈블리 분석하기

- 네임스페이스 선언

```
namespace XXX {  
    using System;  
  
    public class My  
    {  
        private int iNumber = 0;  
  
        public My()  
        {  
  
        }  
  
        public void set(int i)  
        {  
            iNumber = i;  
        }  
    }  
}
```



## 4. CLR과 기본 클래스 라이브러리

- 기본 클래스 라이브러리
  - Base Class Library
  - Console과 같이 기본적으로 존재하는 클래스들  
기본 클래스 라이브러리라고 함
  - 기본 클래스 라이브러리는  
C:\Windows\Framework 폴더에 있는  
mscorlib.dll 파일에 작성되어 있음

```
public class My
{
    public static void Main()
    {
        System.Console.WriteLine("Hello,World!");
    }
}
```

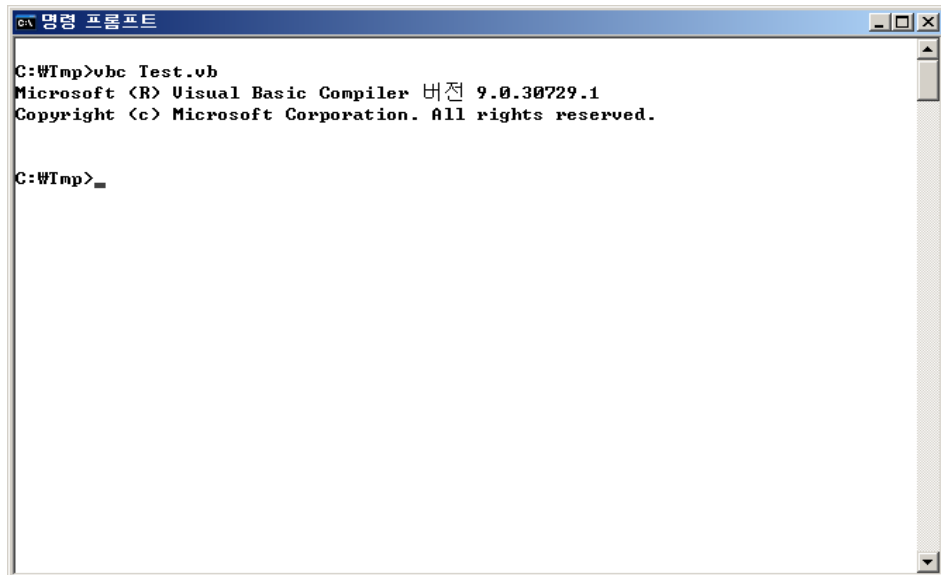
## 4. CLR과 기본 클래스 라이브러리

- 비주얼 베이직 프로그래밍

```
Module Test
    Sub Main()
        System.Console.WriteLine("Hello,World!")
    End Sub
End Module
```

- 컴파일

C:\Wtmp>vbc My.vb



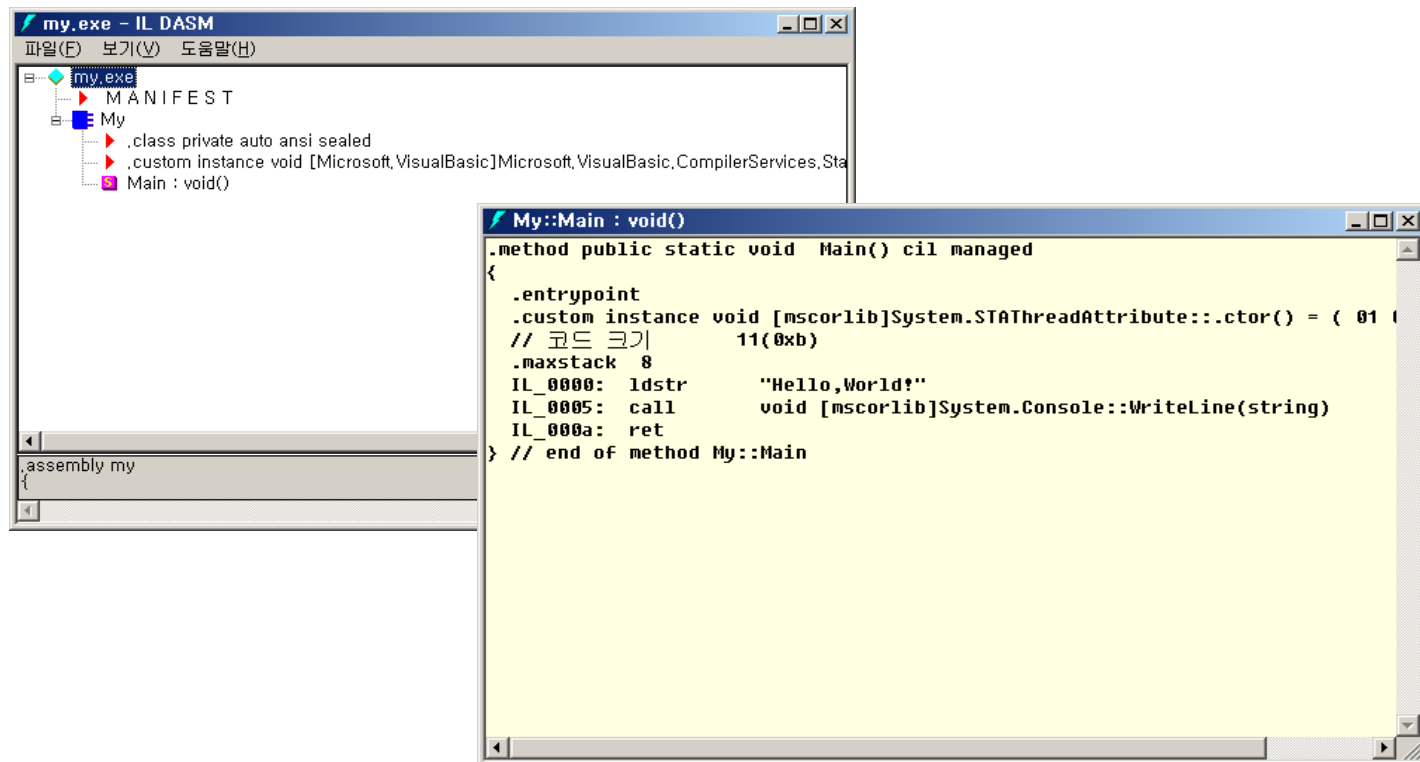
```
C:\명령 프롬프트
C:\Wtmp>vbc Test.vb
Microsoft (R) Visual Basic Compiler 버전 9.0.30729.1
Copyright (c) Microsoft Corporation. All rights reserved.

C:\Wtmp>
```

## 4. CLR과 기본 클래스 라이브러리

- 어셈블리 분석

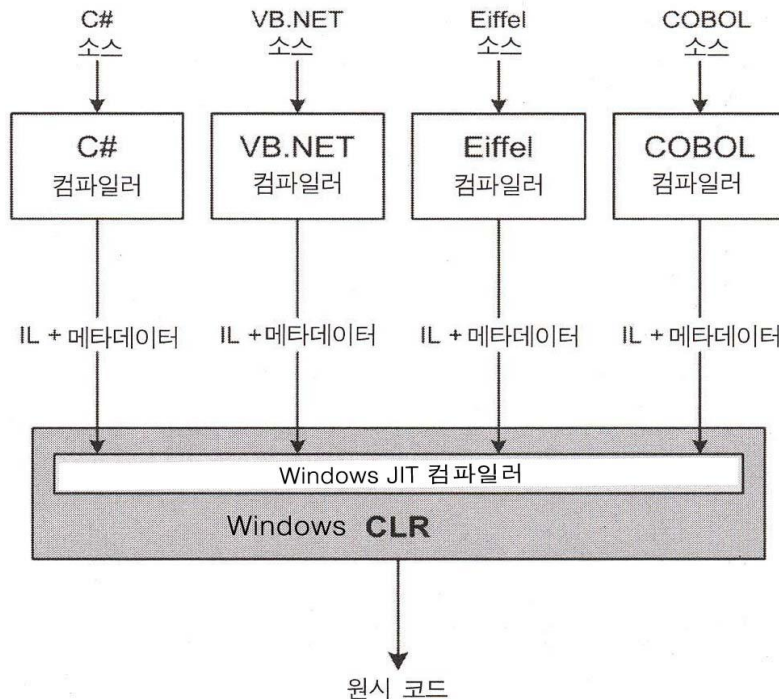
C:\Wtmp>ildasm My.exe





## 4. CLR과 기본 클래스 라이브러리

- C# vs. Visual Basic
  - 어떤 언어로 작성하든 어셈블리는 동일



## 4. CLR과 기본 클래스 라이브러리

- 따라서 IL을 사용하는 이유
  - 원하는 언어(C#, VC++, VB 등)로 응용 프로그램을 작성할 수 있음
  - 언어는 달라도 생성되는 어셈블리(IL 코드)는 같음
  - 운영체제 혹은 플랫폼(IBM PC, 맥 컴퓨터, 유닉스 머신 등)에 관계없이 실행

## 4. CLR과 기본 클래스 라이브러리

- CLR(Common Language Runtime)
  - 런타임, 가상머신 (\*자바가상머신)
  - 런타임은 동적 할당된 객체를 자동으로 제거 (garbage collection)
  - 이처럼 .NET에 의해 객체가 제거되고 관리되는 코드를 관리 코드(managed code)라 함

## 4. CLR과 기본 클래스 라이브러리

- JIT(Just-In-Time) 컴파일러
  - (실행될)때맞춰, 시간에 맞춰
  - CLR 구성 요소 중 하나
  - 실행하는 my.exe 어셈블리를 해당 플랫폼(가령 윈도우 운영체제)에서 실행되는 exe 파일로 just-in-time(제때에, 실행될 때 바로) **변환한 후 실행**
  - 실행 속도가 빠르다.

## 4. CLR과 기본 클래스 라이브러리

```
int a;  
Point gildong = new Point();
```

- 자료형 분류 방법 1
  - 기본(built-in) 자료형 (ex, int 등 이미 있는 것)
  - 사용자 정의(user-defined) 자료형 (ex, Point와 같이 우리가 만든 것)
- 자료형 분류 방법 2
  - 값(value) 자료형
    - int, double 등. new 생성자 없이 객체가 생성됨.
  - 참조(reference) 자료형
    - new 명령어로 객체를 생성해야 함.
    - 사용자 정의 클래스 + Object + String 자료형

## 4. CLR과 기본 클래스 라이브러리

- 정리
  - 어셈블리(assembly)
  - 중간 언어(IL) 코드
  - 메타데이터
  - 매니페스트(manifest)
  - ildasm 도구
  - CLR(Common Language Runtime) 혹은 런타임
  - JIT 컴파일러
  - 쓰레기 수집
  - 관리 코드
  - 기본 클래스 라이브러리(Base Class Library)
  - mscorlib.dll 어셈블리
  - .NET 자료형