

## 제5장

# 닷넷 기본과 어셈블리

이 장에서는 닷넷 환경에서 프로그램을 작성하기 위한 기본 개념에 대하여 다룹니다. 특히 어셈블리의 개념과 어셈블리를 분석할 수 있는 ILDASM

- 
- Step1 : 어셈블리 이해하기
  - Step2 : ILDASM 도구로 어셈블리 분석하기
  - Step3 : DLL 어셈블리 분석하기
  - Step4 : CLR 기본 클래스 라이브러리

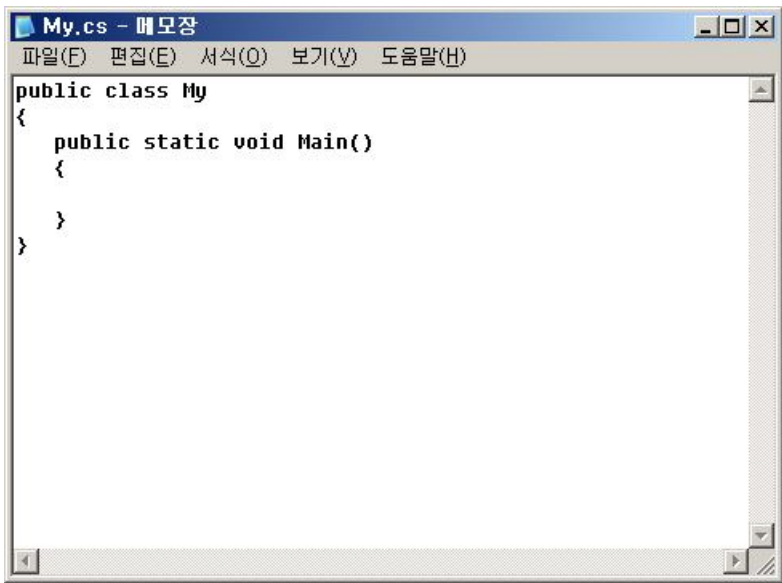


## 어셈블리 이해하기

메모장을 이용하여 다음 코드를 작성한 후 my.cs .

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

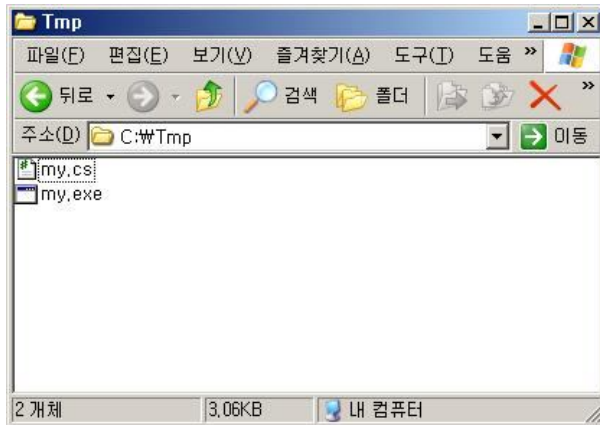
    }
}
```



작성한 프로그램을 도스 환경에서 컴파일해 보겠습니다. 도스 창을 이용하여 다음과 같이 컴파일합니다. csc c-sharp compiler .

```
C:\Tmp>csc my.cs
```

탐색기를 이용하여 생성된 파일을 확인해 봅니다.



참고로 C#

C:\WINDOWS\Microsoft.NET\Framework\v3.5

my.exe 파일은 닷넷 환경에서 실행되는 파일로서 이를 어셈블리(assembly)라고 합니다. hwp.exe 등과 같은 기존의 윈도우 실행 파일과 전혀 다릅니다. 무엇이 다른지를 이해하기 위해 기존 실행 파일의 문제점을 짚어보도록 하겠습니다. 기존 윈도우 실행 파일의 문제점은 크게 다음과 같이 두 가지로 설명할 수 있다.

- 기존 실행 파일에는 특정 CPU 에서 실행되는 기계어로 구성되어 있어서 다른 플랫폼( )
- 기존 실행 파일만 보면 코드가 어떤 내용인지 알 방법이 없습니다. 가령 dll 이 작성되어 있는 라이브러리 파일에는 함수 프로토타입에 관한

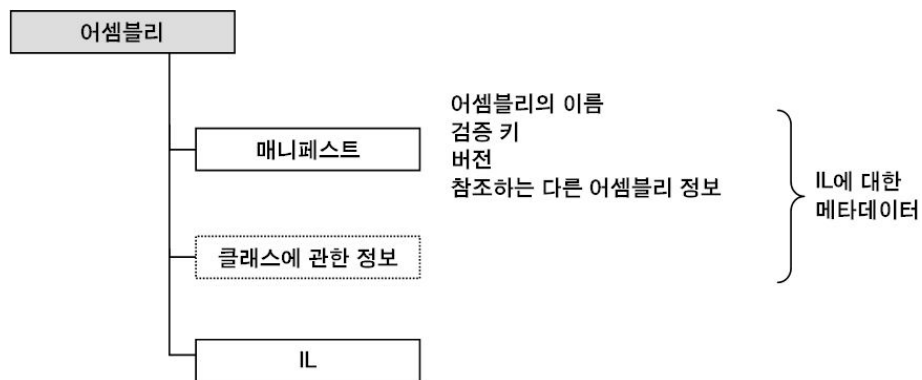
정보가 존재하지 않아서 이 라이브러리를 사용하려면 해당 헤더 파일도 가지고 있어야 합니다.

어셈블리는 위와 같은 문제점을 해결하였습니다. ,

- 어셈블리는 특정 CPU ( 1상 머신) 에서 실행되는 **중간 형태의 언어인 IL(Intermediate Language)** 로 구성됩니다. 그럼으로써 런타임이 설치되어 있기만 하면 다른 플랫폼에서도 실행할 수 있습니다.
- 어셈블리에는 실행 코드인 IL 뿐만 아니라 이에 대한 정보인 메타데이터(metadata) . 가령 어셈블리에 있는 함수의 프로토타입, . 그럼으로써 어셈블리 파일 하나만 있으면 사용이 가능합니다. 헤더 파일이 따로 필요하지 않습니다.

이러한 어셈블리는 다음과 같은 특징을 가지고 있습니다.

- 어셈블리는 설치 및 배포를 위한 최소의 단위입니다.
- csc exe, dll, 그리고 각종 리소스 파일이 모두 어셈블리입니다.
- 결국, (assembly) 를 만드는 것을 의미합니다.
- 어셈블리는 IL IL .
- 메타데이터에는 매니페스트(manifest) , 여기에는 자신이 사용하는 다른 어셈블리에 관한 목록(manifest) 이 들어있으며, , .
- 메타데이터에서 매니페스트 뿐만 아니라 어셈블리에서 사용하는 클래스에 관한 정보가 들어 있습니다.

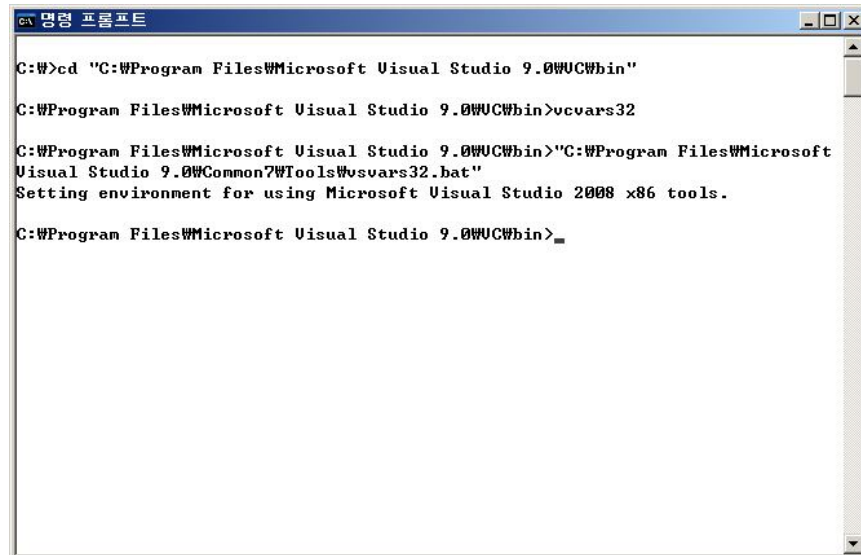


## Step2 ILDASM 도구로 어셈블리 분석하기

ildasm.exe my.exe IL ↵  
 메타데이터를 사람이 읽을 수 있도록 보여줍니다. 도스창 아무데서나  
 ildasm.exe csc.exe 컴파일러를 실행할 수 있도록 하기 위해 두 실행 파일  
 이 있는 경로를 환경변수 Path . . 도스창을  
 이용하여 다음 폴더로 이동합니다.

C:\Program Files\Microsoft Visual Studio 9.0\VC\bin

그리고 vcvars32.bat



```
C:\>cd "C:\Program Files\Microsoft Visual Studio 9.0\VC\bin"

C:\Program Files\Microsoft Visual Studio 9.0\VC\bin>vcvars32

C:\Program Files\Microsoft Visual Studio 9.0\VC\bin>"C:\Program Files\Microsoft
Visual Studio 9.0\Common7\Tools\vsvars32.bat"
Setting environment for using Microsoft Visual Studio 2008 x86 tools.

C:\Program Files\Microsoft Visual Studio 9.0\VC\bin>
```

이제 다른 폴더로 이동하여도 csc.exe ildasm.exe 파일을 실행할 수 있습니다. C:\Tmp csc.exe

```
C:\> 명령 프롬프트

C:\WImp>csc
Microsoft (R) .NET Framework 버전 3.5용 Microsoft (R) Visual C# 2008 컴파일러 버전 3.5.30729.1

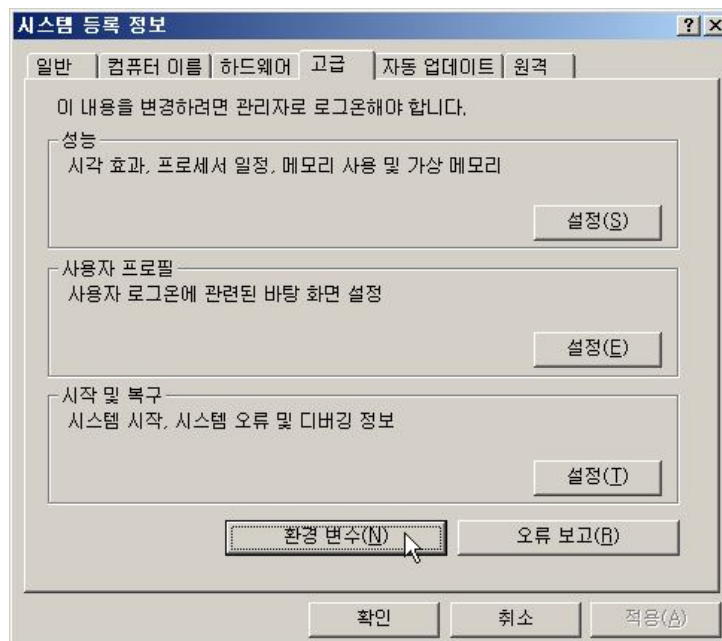
Copyright (c) Microsoft Corporation. All rights reserved.

fatal error CS2008: 지정된 입력이 없습니다.

C:\WImp>
```

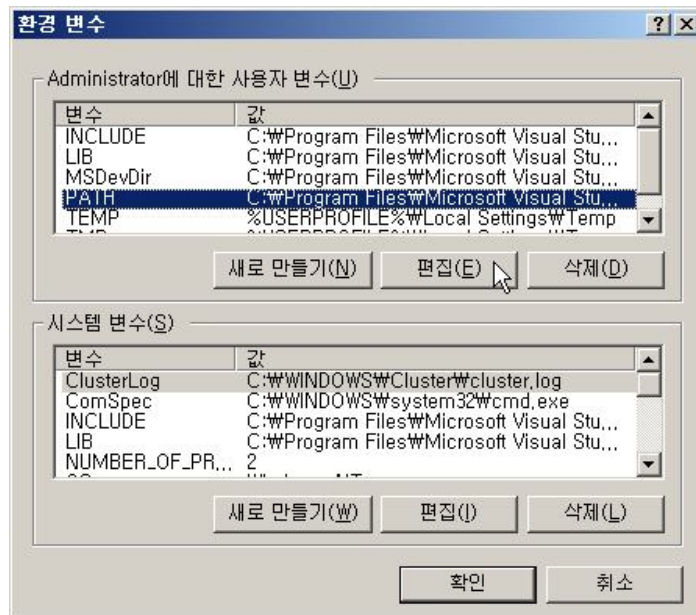
하지만 매번 vcvars32.bat c:\ 폴  
더에 복사해 두면 이후 쉽게 실행할 수 있을 것입니다.

혹은 아래 Path . >  
[ > [ > 환경 변수 버튼을 눌러 설정  
할 수 있습니다.



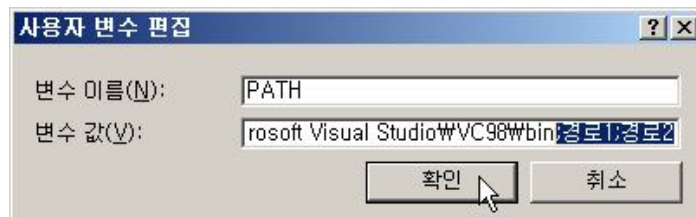
환경변수 PATH



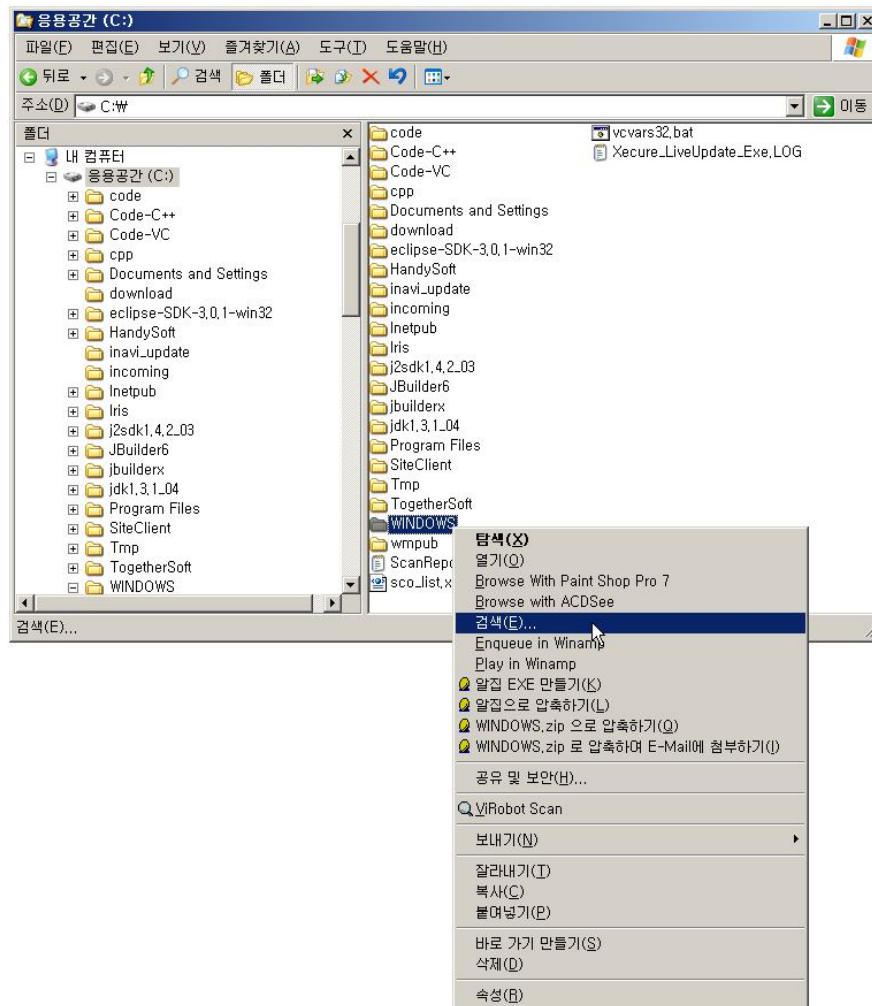


그러면 다음과 같이 편집 대화상자가 표시됩니다.  
파일이 들어있는 폴더 경로1 ildasm.exe  
세미콜론(;) .

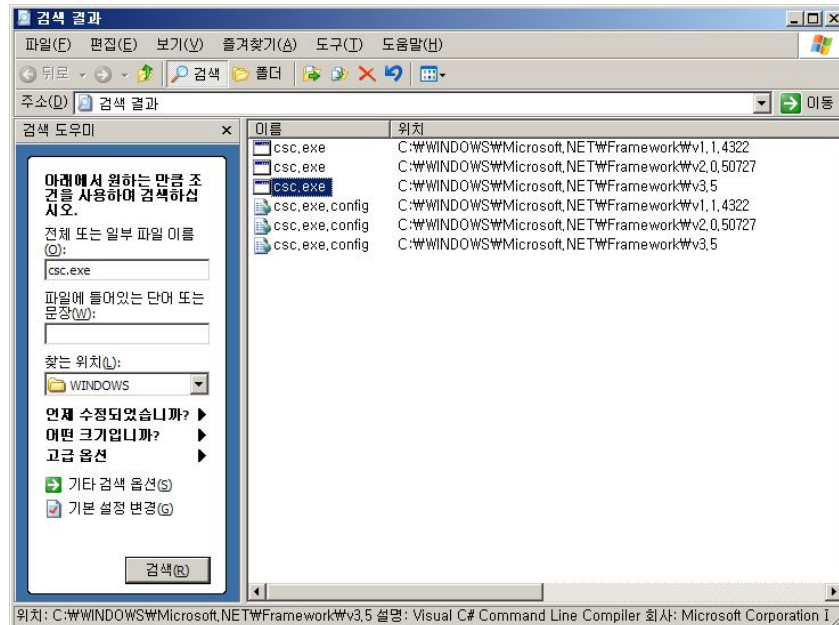
csc.exe 실행  
2 1/2



경로1 2 1/2 알아내기 위하여 다음과 같이 탐색기를 이용하여  
C:\Windows csc.exe .



그 결과 다음과 같이 경로1



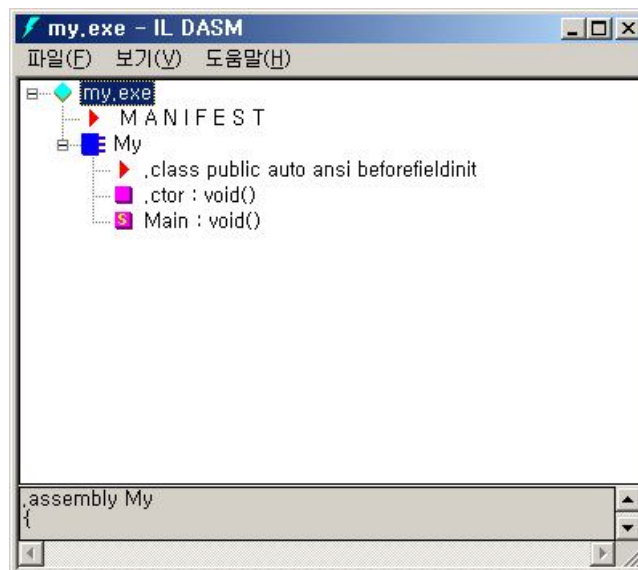
마찬가지 방법으로 C:\Program Files\ildasm.exe 파일이 있는 폴더를 찾습니다.

파일	설명	폴더
ces.exe	C# 컴파일러	C:\WINDOWS\Microsoft.NET\Framework\v3.5
ildasm.exe	중간언어 역어셈블러 도구	C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin

앞으로는 도스창을 새로 실행하여도 두 도구를 바로 실행할 수 있습니다. 그러면 도스창을 실행하여 c:\TMP 폴더로 이동한 후 다음과 같이 ildasm 도구

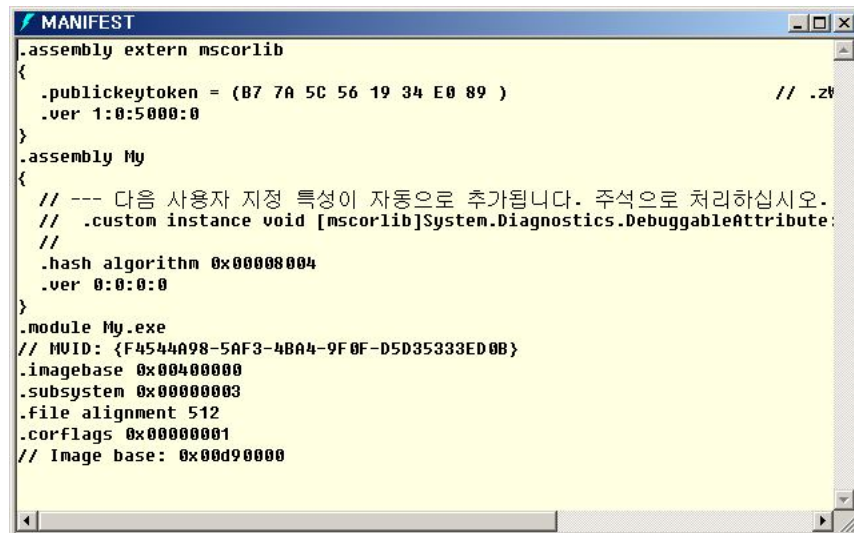
를 실행합니다. my.exe .

```
C:\Tmp>ildasm my.exe
```



위 그림은 my.exe ,  
my.exe 어셈블리에는 위의 메타데이터뿐만 아니라 마이크로소프트에서 제안  
하는 중간 형태의 언어로서 실제로 .NET IL 코드가 들어있  
습니다.

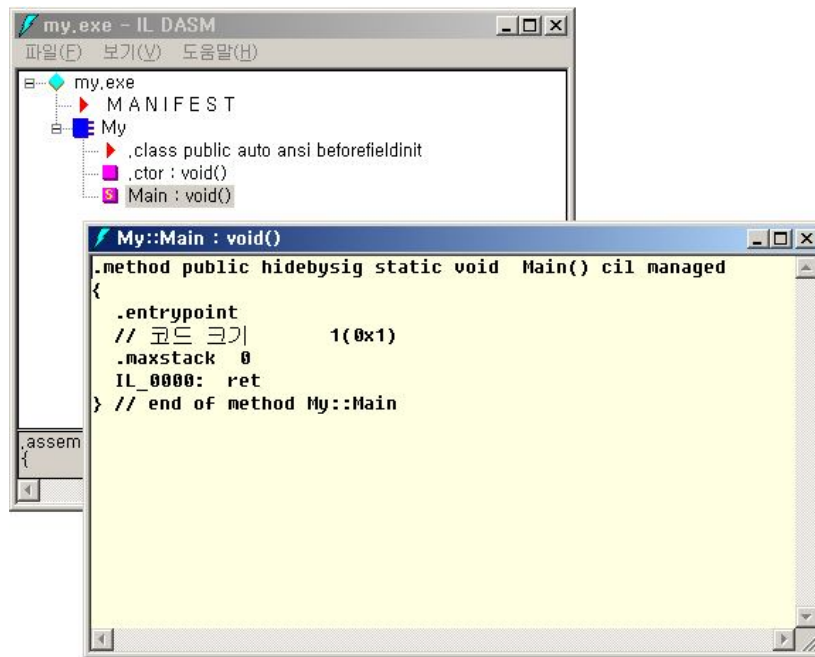
위 그림의 메타데이터를 보면 MANIFEST ,  
여기에는 어셈블리 이름(My), (0:0:0:0), 참조하는 다른 어셈블리  
들(mscorlib) . MANIFEST 부분을  
두 번 클릭하면 다음과 같은 확인할 수 있습니다.



```
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )           // .2v
  .ver 1:0:5000:0
}
.assembly My
{
  // --- 다음 사용자 지정 특성이 자동으로 추가됩니다. 주석으로 처리하십시오.
  // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute:
  //
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module My.exe
// GUID: {F4544A98-5AF3-4BA4-9F0F-D5D35333ED0B}
.imagebase 0x00400000
.subsystem 0x00000003
.file alignment 512
.corflags 0x00000001
// Image base: 0x00d90000
```

위 그림을 보면 mscorlib 어셈블리에 publickeytoken 이라는 것이 들어있는데, mscorlib , 이를 이용하여 이름은 mscorlib

ildasm Main Main IL 코드를 볼 수 있습니다.

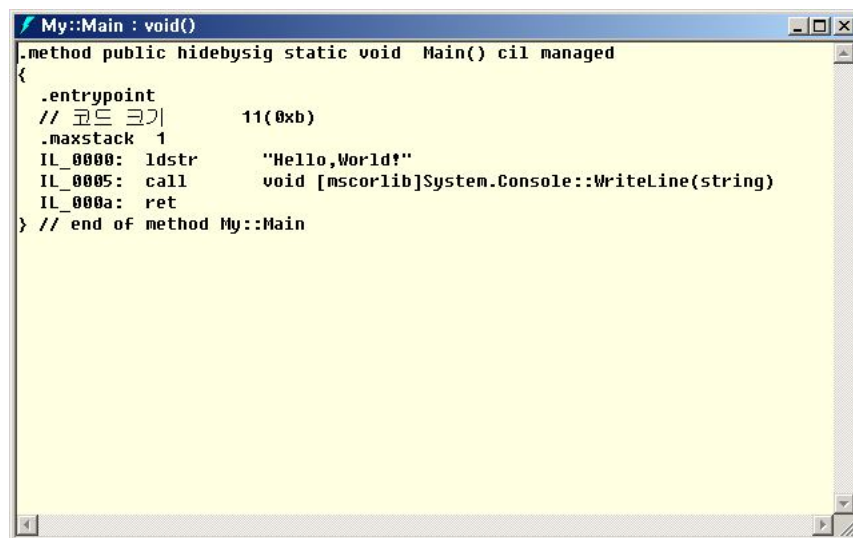


앞서 표시된 코드가 바로 마이크로소프트에서 정의한 중간 언어, IL입니다.

이제 Main 함수에 Hello,World!

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

ildasm                      Main                      IL ㄴ 보면 다음과 같이 바뀌어 있습니다.



```
.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
```

ldstr 명령어는 "Hello,World!" 라는 문자열을 스택에 push                      . 그 다음 호출한 call                      top 1 있는 문자열을 꺼낸 후(pop) WriteLine 함수를 이용하여 화면에 출력합니다.



### DLL 어셈블리 분석하기

이번에는 dll . 메모장을 이용하여 다음  
프로그램을 작성한 후 Module.cs .

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

위 프로그램은 Main . (\*.exe) ㄹ 만들 수 없  
으며, dll 라이브러리를 생성하도록 옵션을 입력하여 컴파일합니  
다.

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



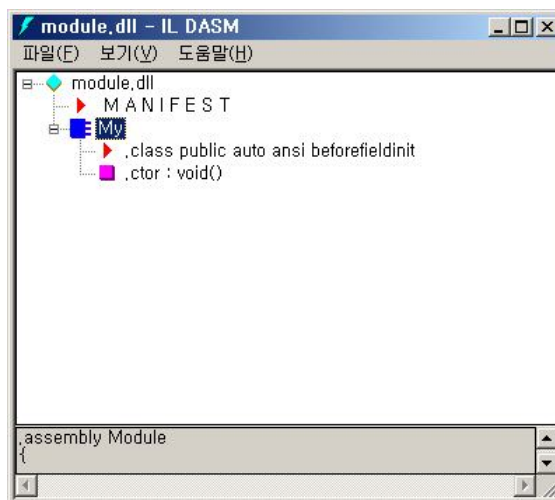
```

C:\Wtmp>csc /t:library Module.cs
Microsoft (R) Visual C# .NET Compiler version 7.10.3052.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

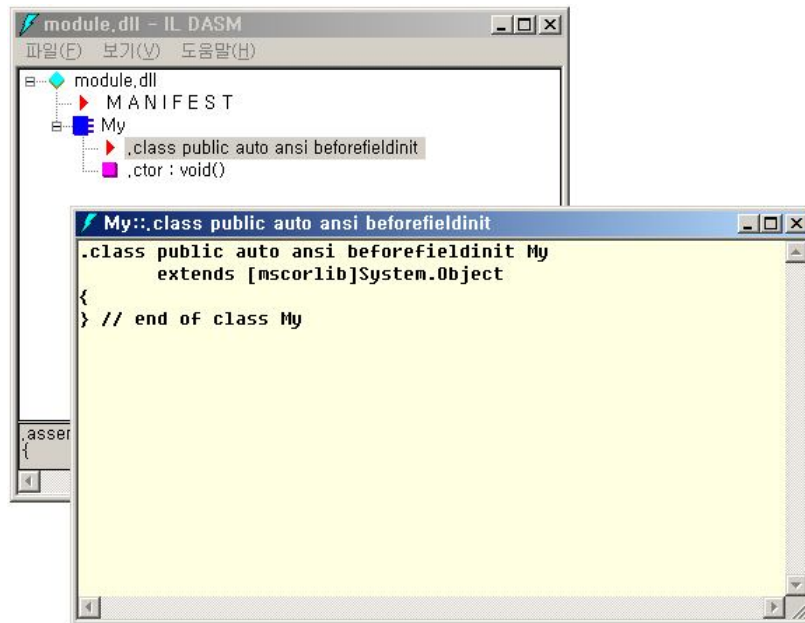
C:\Wtmp>

```

ildasm . My public 로 선언되어 있고, .



다음은 클래스 정보를 본 모습입니다.



다음과 같이 멤버 함수를 추가해 봅니다.

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

```

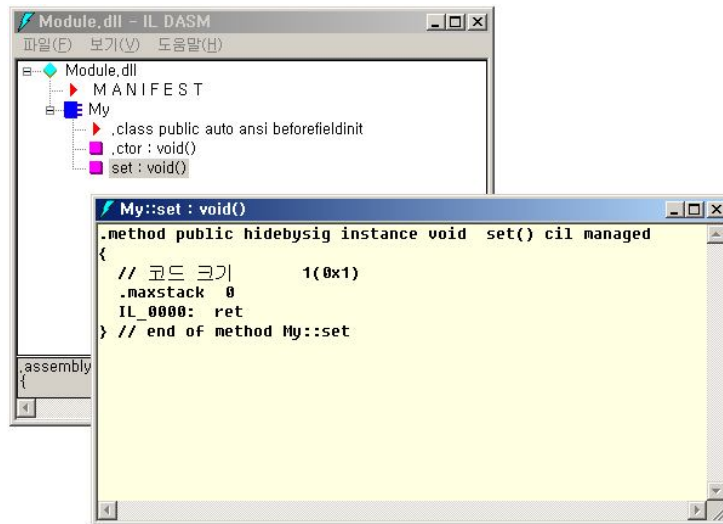
    {

    }

}

```

추가한 멤버 함수 IL



set . IL 명령어 ret 를 이용하여 그냥 반환합니다.

```

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

public class My
{
    public My ()
    {

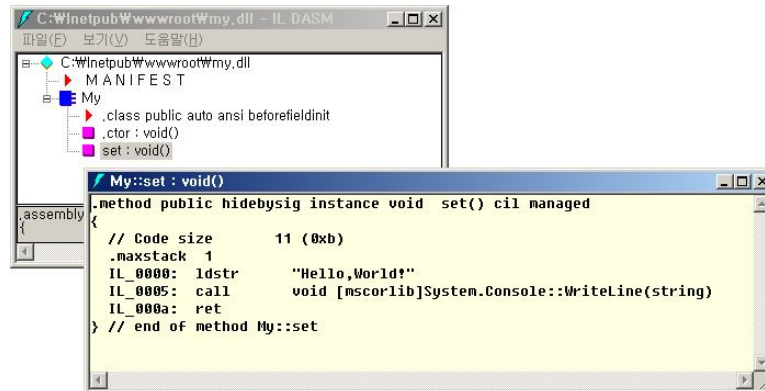
```

```

    }

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

```



문자열 "Hello,World!" (push) 후(ldstr) 닷넷 프레임워크에 있는 기본 클래스 라이브러리(Basic Class Library) Console.WriteLine 함수를 호출 (call) . WriteLine top | 있는 문자열 , "Hello,World!" (pop) .

다음은 멤버 변수를 정의한 예입니다. set 멤버 함수에 작성한 코드를 지웠습니다.

```

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

```

```
using System;

public class My
{
    private int iNumber;

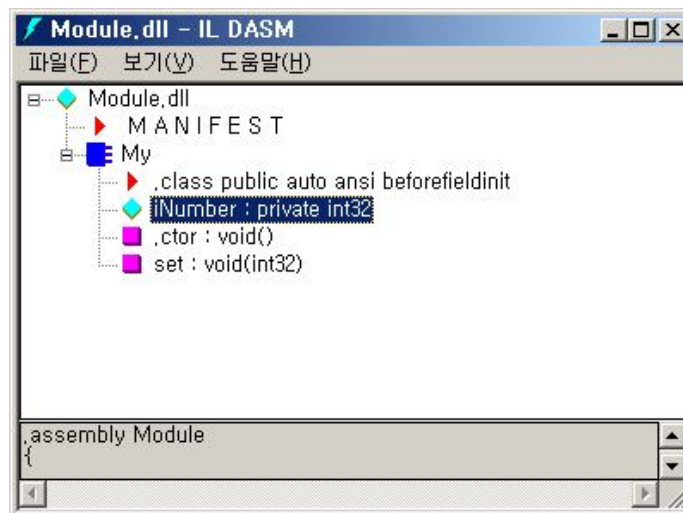
    public My()
    {

    }

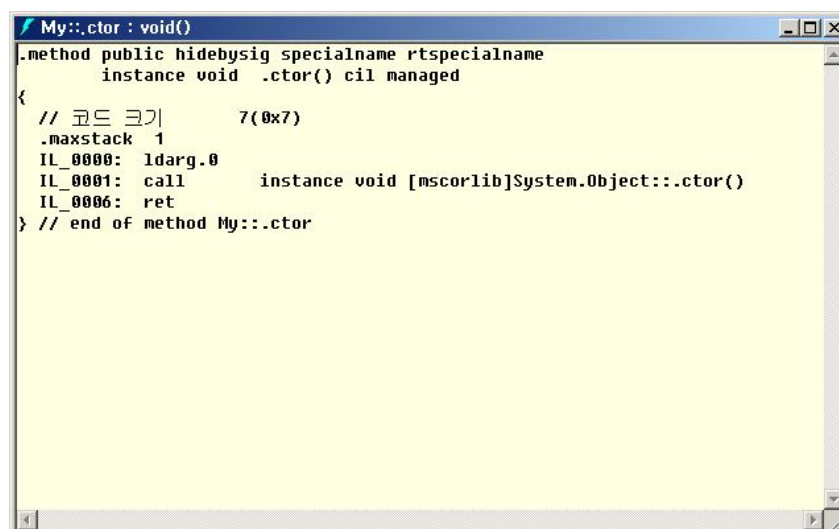
    public void set()
    {

    }
}
```

ildasm 도구로 어셈블리를 보면 다음과 같이 멤버 변수가 있음을 알 수 있습니다.



멤버 변수를 초기화하지 않았습니다. IL .



ldarg.0이라는 명령어는 현재 객체의 주소값인 this push 는 명령어  
입니다. call (this ) Object 클래스로부터 상속받은 생

생성자 함수를 호출합니다.

초기화를 수행하도록 코드를 작성해 보겠습니다.

```
//-----  
// Module.cs  
//-----  
public class My  
{  
    private int iNumber = 0;  
  
    public My()  
    {  
  
    }  
  
    public void set()  
    {  
  
    }  
}
```

생성자 함수의 IL

```

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

```

ldarg.0 명령어에 의해 스택에 this      push      . ldc.i4.0 명령어에 의해 0 값이 스택에 push      . stfld 명령어는 스택에 있는 두 값 this    0    pop    하여 this.iNumber      0      . ,      . 이로써 멤버 변수를 초기화하면 생성자 함수에 어떤 코드가 작성되는지를 알 수 있습니다.

Set      .

```

//-----
// Module.cs
//-----
public class My
{
    private int iNumber = 0;

```



```
public My()
{

}
```

```
public void set(int i)
{

}
```

ildasm                      set                      IL 코드를 보면 다음과 같습니다.

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

32                      i                      . 이제 아래와 같이 작성하겠습니다.

```
//-----
// Module.cs
//-----
public class My
{
    private int iNumber;
```

```

        public My()
        {

        }

        public void set(int i)
        {
            iNumber = i;
        }
    }

```

다음과 같이 set

```

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

```

set  
 습니다. . 하지만 실제로는 그렇지 않  
 넘겨줍니다. this set 멤버 함수의 파라미터로

```

set(this, 7);

```

현 객체 gildong ( this) 0 번째 인자로 자동으로 전송되  
 고 있다는 뜻이지요. 7 : 사실은 두 번째 파라미터로 전달됩니  
 다.

아래의 코드를 보면 0 this ( ) : 스택에  
 로드하기 위하여 ldarg.0 . ldarg load argument

```

.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

```

`this.iNumber`                      `i`                      .                      . 또  
 한 1                      `i`                      `stfld int32 My::iNumber` 1호출  
 하면 됩니다.

```

this.iNumber = i;

```

이제까지 작성한 클래스를 XXX 네임스페이스를 갖도록 선언해 보겠습니다.

```

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

```

```

namespace XXX
{
    public class My
    {
        private int iNumber = 0;

        public My()
        {

```

```

    }

    public void set(int i)
    {
        iNumber = i;
    }
}
}

```

ildasm                      XXX                      My 클래스가 선언되어 있음을  
볼 수 있습니다.



#### Step4

### CLR과 기본 클래스 라이브러리

다음 프로그램을 살펴보겠습니다.

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

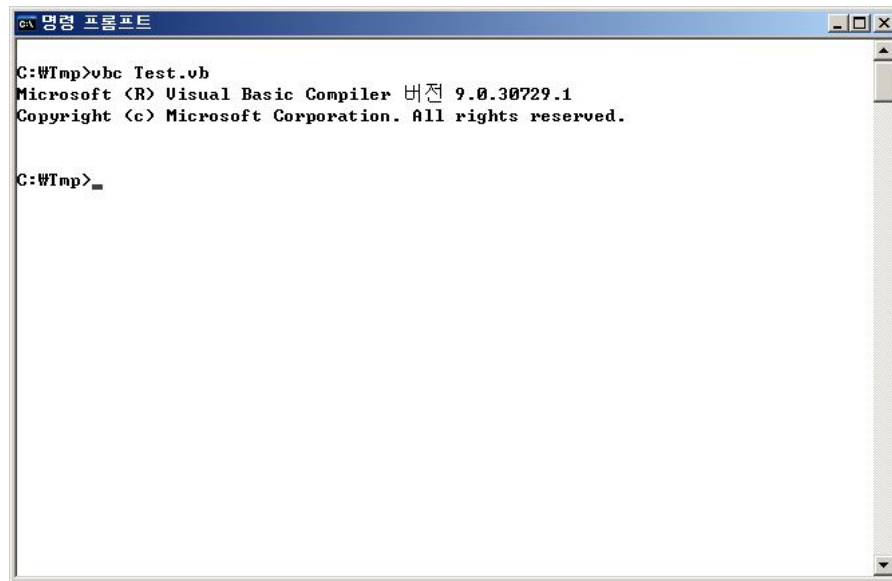
위 코드를 보면 Console.WriteLine 정적 멤버 함수를 호출하고 있습니다. Console 클래스와 Console.WriteLine은 Console 클래스와 함께 기본적으로 존재하는 클래스들을 기본 클래스 라이브러리 (Base Class Library)에 포함하고 있습니다.

C#과 함께 Visual Basic.NET도 사용할 수 있습니다. C#과 함께 Visual Basic.NET을 사용할 때는 C:\Tmp\Test.vb라는 파일로 저장합니다.

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

비주얼 베이직 컴파일러는 vbc.exe입니다. 따라서 다음과 같이 컴파일합니다.

C:\Tmp>vbc My.vb

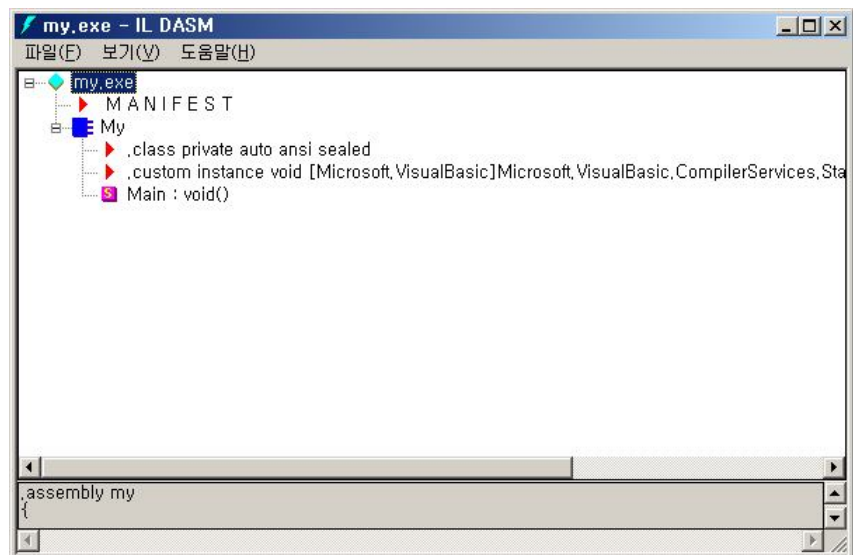


```
C:\Tmp>vbc Test.vb
Microsoft (R) Visual Basic Compiler 버전 9.0.30729.1
Copyright (c) Microsoft Corporation. All rights reserved.

C:\Tmp>
```

다음과 같이 입력하여 ildasm

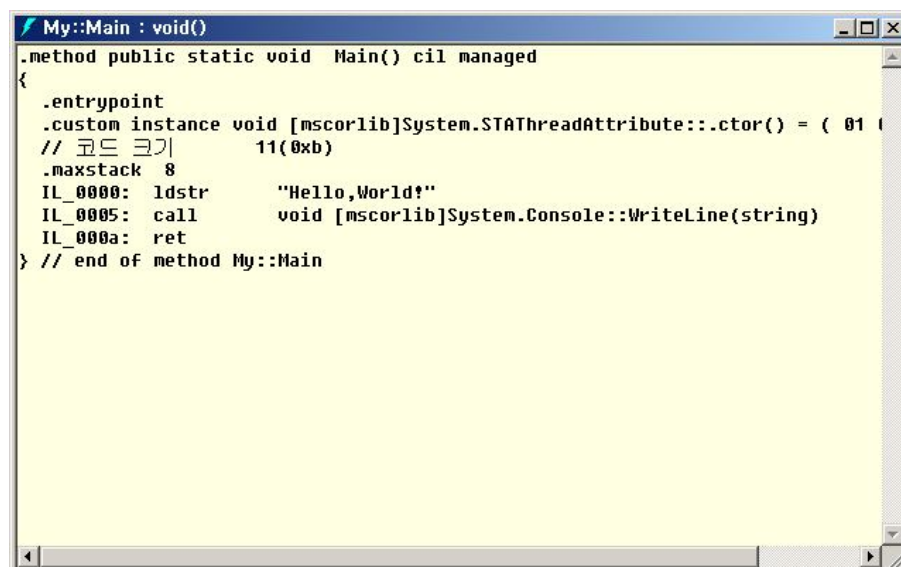
C:\Tmp>ildasm My.exe



Main

IL

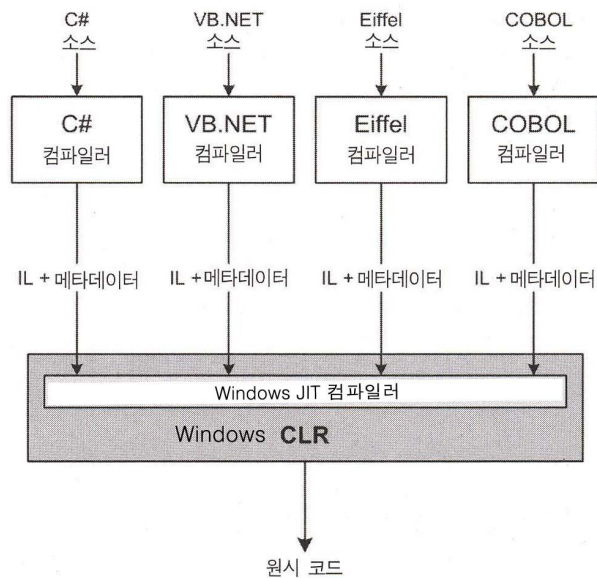
.



C#

IL 명령어

레벨에서는 다르지 않음을 알 수 있습니다. 이는 어떤 언어를 이용하여 작성하든지 상관없이 어셈블리는 동일하다는 것을 의미합니다.



따라서 중간 형태의 언어(IL)를 사용하는 이유는 다음과 같이 정리할 수 있습니다.

- 여러분이 사용하기 편한 언어를 이용하여 똑 같은 C# 응용 프로그램을 작성할 수 있습니다. 언어에 관계없이 생성되는 어셈블리는 같으므로 하나의 프로그램 ( ) : 여러 언어를 이용하여 작성할 수도 있습니다.
- .NET 어셈블리는 운영체제 혹은 플랫폼에 관계없이 실행됩니다. .NET 런타임만 설치되어 있으면 윈도우든 UNIX IL



코드를 실행할 수 있으므로 어셈블리는 플랫폼에 독립적입니다.

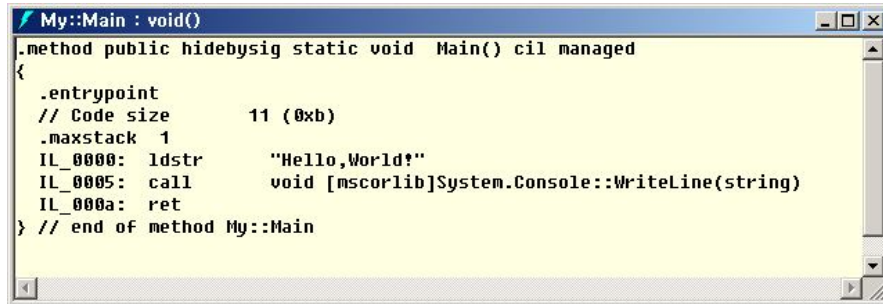
CLR(Common Language Runtime, 혹은 줄여서 런타임) : 여러분이 어셈블리를 실행할 경우 다양한 언어로 만들어질 수 있는 어셈블리를 하드 디스크에서 찾아 메모리로 로드하고 실행합니다. C# 로 작성된 어셈블리를 실행하면 런타임이 이를 찾아 메모리로 로드한 후 Main 함수를 찾아 실행합니다. 또한 어셈블리의 메타데이터에 있는 정보를 이용하여 실행에 필요한 다른 어셈블리 및 모듈들을 로드하기도 합니다.

CLR | 구성 요소 중의 하나인 JIT(Just-In-Time) my.exe  
를 실행하면 실행 바로 전에 my.exe ( 명령 윈도우 운영체제) exe ( hwp.exe) just-in-time( , )  
검파일한 후 실행합니다. , 어셈블리를 실행하면 중간 코드를 한 줄 한 줄  
인터프리트 방식으로 실행하는 것이 아니라 해당 플랫폼에 맞는 원시(native, cpu-specific) 코드를 바로 생성한 후 일괄적으로 실행하기 때문에 어셈블리 실행 속도가 상당히 빠릅니다.

또한 런타임은 닷넷 응용( ) |서 동적 할당된 객체를 자동으로 제거해줍니다. .NET | 어떤 언어를 이용하여 프로그램을 작성할 경우 동적 할당된 객체를 직접 제거하는 코드를 작성할 필요가 없습니다. 런타임의 이러한 기능을 쓰레기 수집(garbage collection) . 이처럼 .NET | 의해 객체가 제거되고 관리되는 코드를 관리 코드(managed code) 라고 합니다. my.exe . 이에 반해 기존의 윈도우의 실행 파일, 가령 워드 프로그램이나 아래아 한글 프로그램 등을 관리되지 않는, 혹은 비관리 코드라고 합니다.

C#, Visual C++, Visual Basic 등과 같이 서로 다른 언어를 이용하여 개발한 어셈블리들이 하나의 공통 실행 환경(Common Language Runtime) |서 실행

C# Visual Basic 3 이용하든 컴파일할 경우 생성되는 어셈블리는 기본 클래스 라이브러리를 이용합니다. , ildasm 도구를 이용하면 쉽게 확인할 수 있습니다.



C:\Windows\Framework 폴더에 있는 mscorlib.dll 파일에 작성되어 있습니다.

자료형에는 누가 만들었느냐에 따라 두 가지로 구분할 수 있습니다. 여러분이 직접 만든(user-defined) 것과 기존에 이미 만들어진(built-in) 것으로 구분할 수 있습니다. 가령 여러분이 직접 만든 클래스나 구조체 등을 사용자 정의 자료형이라고 하고, .NET (built-in) 자료형을 기본 (built-in) 자료형은 앞서 살펴본mscorlib.dll

- 기본(built-in) 자료형
- 사용자 정의(user-defined) 자료형

자료형을 다른 방법으로도 분류할 수 있습니다. (value) 자료형이고 다른 하나는 참조(reference) . 값 자료형은 선언하자마자 값을 저장할 수 있는 객체가 바로 만들어지는 자료형이고, 참조 자료형은 단지 참조만 만들어지고 객체는 만들어지지 않는 자료형이다.

앞서 설명한대로 값 자료형으로 정의된 객체( ) : 정의하자마자 바로 객체가 만들어집니다. 함수가 실행될 때 만들어지고 함수가 끝날 때 사라지는데, . 참조 자료형은 객체가 만들어지지 않고 단지 참조만 만들어지기 때문에 객체에 값을 저장하거나 멤버 함수를 호출하려면 반드시 new 명령어로 객체를 실행 시간에 만들어야 합니다.

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

한편, .NET } 런타임이 쓰레기 수집을 통해 불필요할 경우 스스로 제거할 수 있도록 하기 위하여 메모리 영역 중 힙(heap)이라는 영역에 만들어집니다. 다음은 값 자료형과 참조 자료형에 속하는 자료형들을 기술한 것입니다.

- 값(value) : Object String 자료형을 제외한 모든 자료형, 그리고 구조체
- 참조(reference) : 모든 사용자 정의 자료형과 기본 자료형 중 Object String 자료형

# .NET built-in 타입

.NET	C#	VB .NET	IL	값 또는 참조
System.Boolean	bool	Boolean	bool	값
System.Byte	byte	Byte	unsigned int8	값
System.Char	char	Char	char	값
System.DateTime	-	Date	-	값
System.Decimal	decimal	Decimal	-	값
System.Double	double	Double	float64	값
System.Int16	short	Short	int16	값
System.Int32	int	Integer	int32	값
System.Int64	long	Long	int64	값
System.object	<u>object</u>	object	object	참조
System.SByte	sbyte	-	int8	값
System.Single	float	Single	float32	값
System.String	<u>string</u>	String	string	참조
System.UInt16	ushort	-	unsigned int16	값
System.UInt32	uint	-	unsigned int32	값
System.UInt64	ulong	-	unsigned int64	값

다음은 값 자료형을 사용하는 예입니다.

```
class My
{
    public static void Main()
    {
        int a; //값
        a = 2;
        System.Console.WriteLine(a);
    }
}
```

다음은 참조(reference)

My

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

```
class My
{
    public static void Main()
    {
        int a; //값
        a = 2;
        System.Console.WriteLine(a);

        My gildong; // 레퍼런스 선언
        gildong = new My(); // 힙에 객체 정의
        gildong.Study();
    }
}
```

요약 정리하겠습니다.

- 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