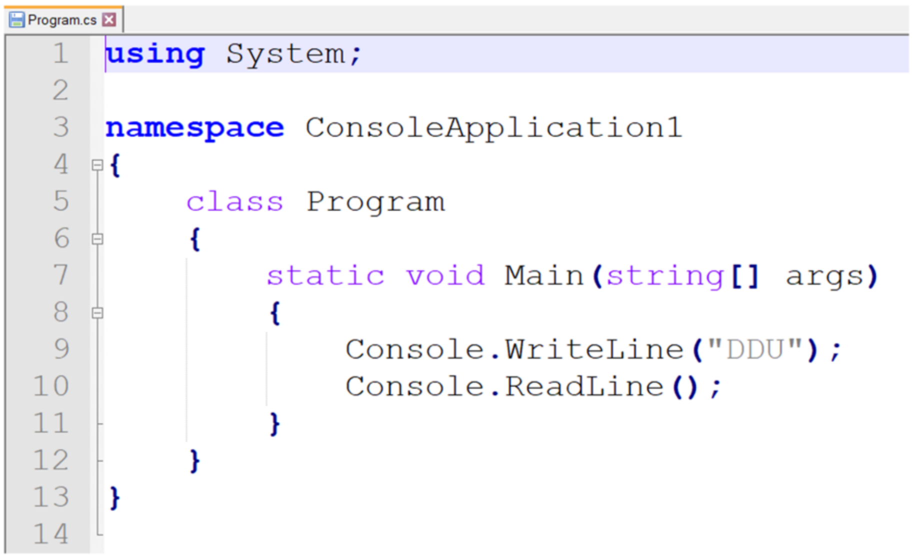
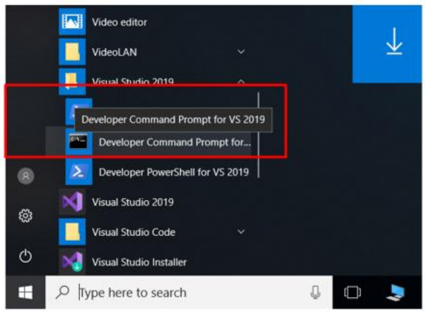
Assembly,
Type and Type members

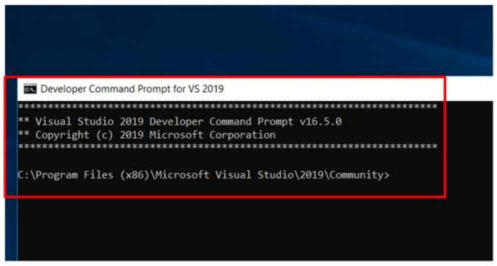
Prepared for Vth semester DDU-CE students
2022-23 WAD

Apurva A Mehta



```csharp
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("DDU");
            Console.ReadLine();
        }
    }
}

```

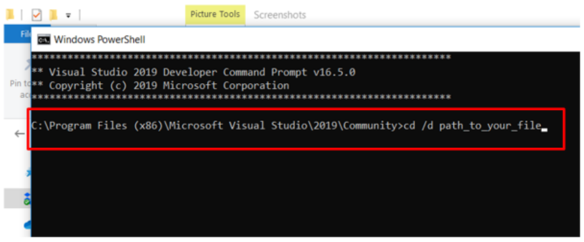




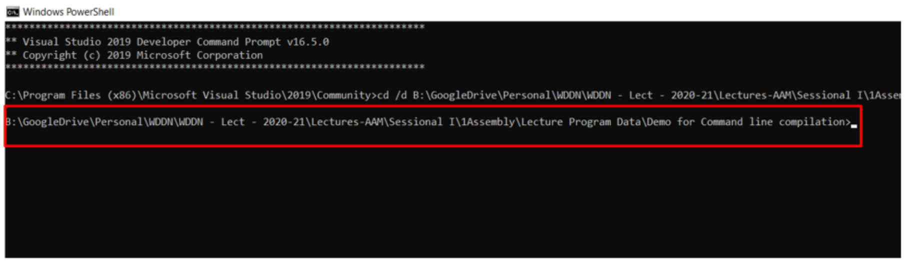
Developer Command Prompt for VS 2019

```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.5.0
** Copyright (c) 2019 Microsoft Corporation
**********************************************************************

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>
```

Find and access VS developer command tool

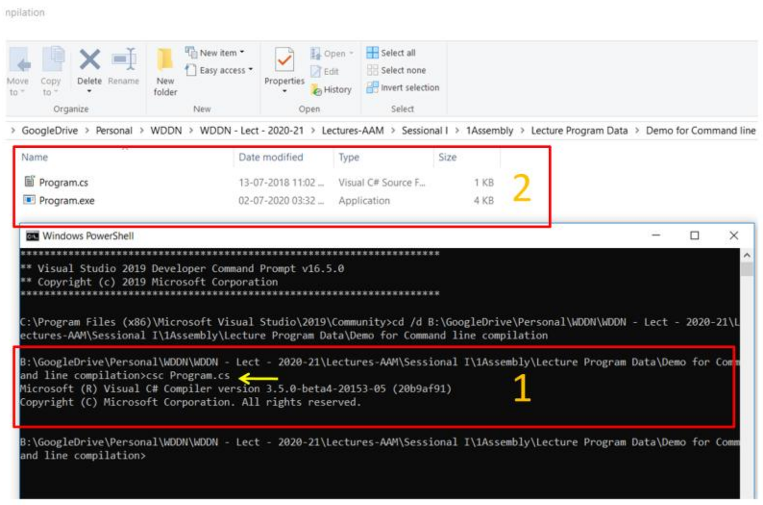


```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.5.0
** Copyright (c) 2019 Microsoft Corporation
**********************************************************************

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>cd /d path_to_your_file_
```



```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.5.0
** Copyright (c) 2019 Microsoft Corporation
**********************************************************************

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>cd /d B:\GoogleDrive\Personal\WDDN\WDDN - Lect - 2020-21\Lectures-AAM\Sessional I\1Asse

B:\GoogleDrive\Personal\WDDN\WDDN - Lect - 2020-21\Lectures-AAM\Sessional I\1Assembly\Lecture Program Data\Demo for Command line compilation>_
```

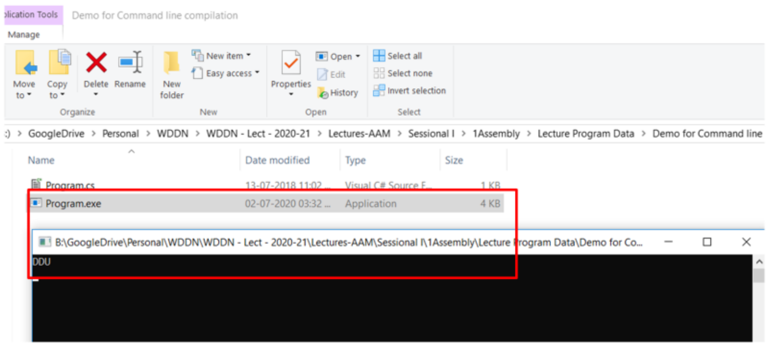
1: Way to change directory for targeting your program directory
2: In your program directory



Name	Date modified	Type	Size
Program.cs	13-07-2018 11:02 ...	Visual C# Source F...	1 KB
Program.exe	02-07-2020 03:32 ...	Application	4 KB

```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.5.0
** Copyright (c) 2019 Microsoft Corporation
**********************************************************************

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>cd /d B:\GoogleDrive\Personal\WDDN\WDDN - Lect - 2020-21\L
ectures-AAM\Sessional I\1Assembly\Lecture Program Data\Demo for Command line compilation

B:\GoogleDrive\Personal\WDDN\WDDN - Lect - 2020-21\Lectures-AAM\Sessional I\1Assembly\Lecture Program Data\Demo for Comm
and line compilation>csc Program.cs
Microsoft (R) Visual C# Compiler version 3.5.0-beta4-20153-05 (20b9af91)
Copyright (C) Microsoft Corporation. All rights reserved.

B:\GoogleDrive\Personal\WDDN\WDDN - Lect - 2020-21\Lectures-AAM\Sessional I\1Assembly\Lecture Program Data\Demo for Comm
and line compilation>
```

1: Initially you have only your .cs file and then you compile .cs with csc.exe
2: it creates .exe file as visible in folder structure.



Demo for Command line compilation

:) > GoogleDrive > Personal > WDDN > WDDN - Lect - 2020-21 > Lectures-AAM > Sessional I > 1Assembly > Lecture Program Data > Demo for Command line

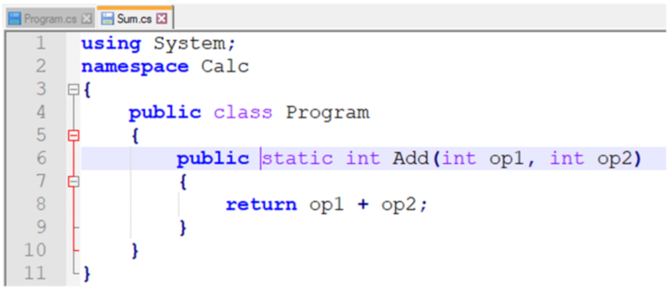
Name	Date modified	Type	Size
Program.cs	13-07-2018 11:02 ...	Visual C# Source F...	1 KB
Program.exe	02-07-2020 03:32 ...	Application	4 KB

B:\GoogleDrive\Personal\WDDN\WDDN - Lect - 2020-21\Lectures-AAM\Sessional I\1Assembly\Lecture Program Data\Demo for Co...

```
DDU
```

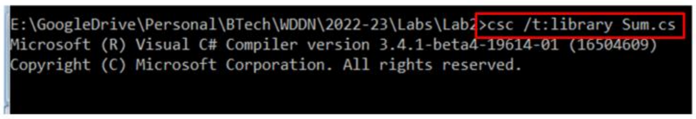
When you execute .exe file you see the corresponding output.

Ex. 1

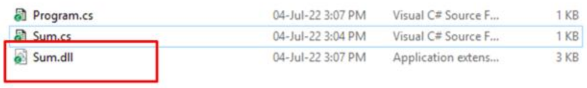


```
using System;
namespace Calc
{
    public class Program
    {
        public static int Add(int op1, int op2)
        {
            return op1 + op2;
        }
    }
}
```

Ex. 1

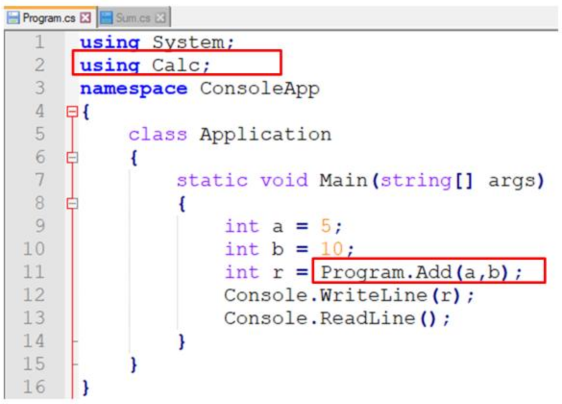


```
E:\GoogleDrive\Personal\BTech\WDDN\2022-23\Labs\Lab2>csc /t:library Sum.cs
Microsoft (R) Visual C# Compiler version 3.4.1-beta4-19614-01 (16504609)
Copyright (C) Microsoft Corporation. All rights reserved.
```



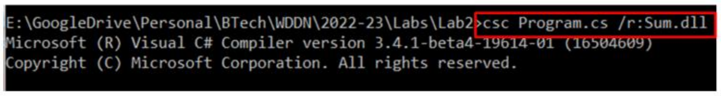
```
Program.cs    04-Jul-22 3:07 PM    Visual C# Source F...    1 KB
Sum.cs        04-Jul-22 3:04 PM    Visual C# Source F...    1 KB
Sum.dll       04-Jul-22 3:07 PM    Application extens...    3 KB
```

Ex. 1

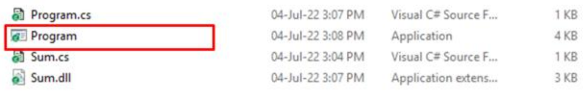


```
using System;
using Calc;
namespace ConsoleApp
{
    class Application
    {
        static void Main(string[] args)
        {
            int a = 5;
            int b = 10;
            int r = Program.Add(a,b);
            Console.WriteLine(r);
            Console.ReadLine();
        }
    }
}
```

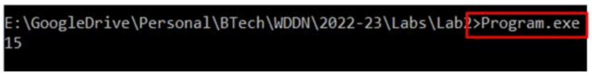
Ex. 1



```
E:\GoogleDrive\Personal\BTech\WDDN\2022-23\Labs\Lab2>csc Program.cs /r:Sum.dll
Microsoft (R) Visual C# Compiler version 3.4.1-beta4-19614-01 (16504609)
Copyright (C) Microsoft Corporation. All rights reserved.
```



```
Program.cs    04-Jul-22 3:07 PM    Visual C# Source F...    1 KB
Program       04-Jul-22 3:08 PM    Application              4 KB
Sum.cs        04-Jul-22 3:04 PM    Visual C# Source F...    1 KB
Sum.dll       04-Jul-22 3:07 PM    Application extens...    3 KB
```



```
E:\GoogleDrive\Personal\BTech\WDDN\2022-23\Labs\Lab2>Program.exe
15
```

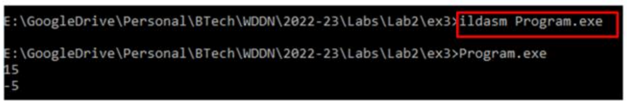
Ex. 2

- Create assembly calc.dll which contains functions to add, subtract, multiply and divide two numbers.
- Create program.cs which contains main method and uses calc.dll file.

Ex. 3

- Create assembly sum.dll and sub.dll which contains functions to add two numbers and subtract two numbers respectively.
- Create program.cs which contains main method and uses sum.dll and sub.dll files.

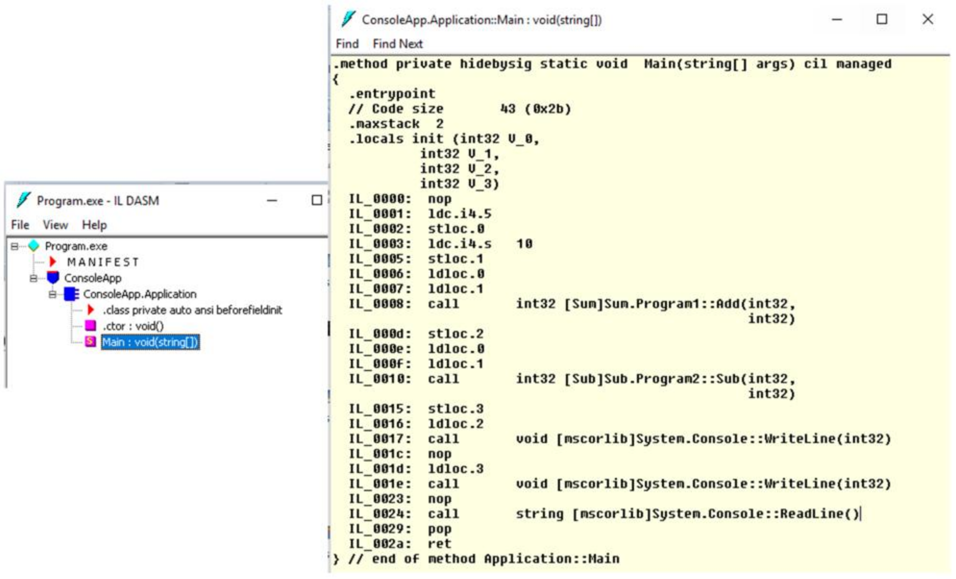
ILDASM



```
E:\GoogleDrive\Personal\BTech\WDDN\2022-23\Labs\Lab2\ex3>ildasm Program.exe

E:\GoogleDrive\Personal\BTech\WDDN\2022-23\Labs\Lab2\ex3>Program.exe
15
-5
```

Disassembler (Ildasm.exe) is included with the .NET Framework SDK. The Ildasm.exe parses any .NET Framework .exe or .dll assembly, and shows the information in human-readable format.
Ildasm.exe shows more than just the Microsoft intermediate language (MSIL) code — it also displays namespaces and types, including their interfaces

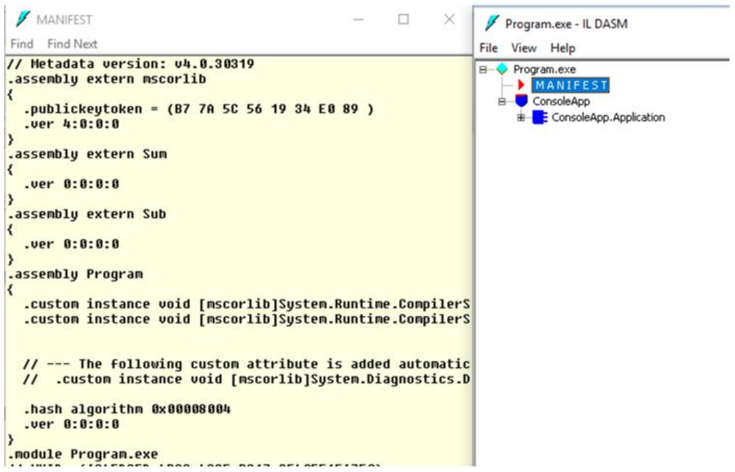
ILDASM



```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       43 (0x2b)
  .maxstack  2
  .locals init (int32 V_0,
           int32 V_1,
           int32 V_2,
           int32 V_3)
  IL_0000:  nop
  IL_0001:  ldc.i4.5
  IL_0002:  stloc.0
  IL_0003:  ldc.i4.s   10
  IL_0005:  stloc.1
  IL_0006:  ldloc.0
  IL_0007:  ldloc.1
  IL_0008:  call       int32 [Sum]Sum.Program1::Add(int32,
                                                    int32)
  IL_000d:  stloc.2
  IL_000e:  ldloc.0
  IL_000f:  ldloc.1
  IL_0010:  call       int32 [Sub]Sub.Program2::Sub(int32,
                                                    int32)
  IL_0015:  stloc.3
  IL_0016:  ldloc.2
  IL_0017:  call       void [mscorlib]System.Console::WriteLine(int32)
  IL_001c:  nop
  IL_001d:  ldloc.3
  IL_001e:  call       void [mscorlib]System.Console::WriteLine(int32)
  IL_0023:  nop
  IL_0024:  call       string [mscorlib]System.Console::ReadLine()
  IL_0029:  pop
  IL_002a:  ret
} // end of method Application::Main
```

Disassembler (Ildasm.exe) is included with the .NET Framework SDK. The Ildasm.exe parses any .NET Framework .exe or .dll assembly, and shows the information in human-readable format.
Ildasm.exe shows more than just the Microsoft intermediate language (MSIL) code — it also displays namespaces and types, including their interfaces

ILDASM



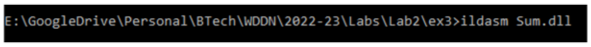
```
// Metadata version: v4.0.30319
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 4:0:0:0
}
.assembly extern Sum
{
  .ver 0:0:0:0
}
.assembly extern Sub
{
  .ver 0:0:0:0
}
.assembly Program
{
  .custom instance void [mscorlib]System.Runtime.CompilerS
  .custom instance void [mscorlib]System.Runtime.CompilerS


  // --- The following custom attribute is added automatic
  //  .custom instance void [mscorlib]System.Diagnostics.D

  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module Program.exe
```

Disassembler (Ildasm.exe) is included with the .NET Framework SDK. The Ildasm.exe parses any .NET Framework .exe or .dll assembly, and shows the information in human-readable format.
Ildasm.exe shows more than just the Microsoft intermediate language (MSIL) code — it also displays namespaces and types, including their interfaces

?

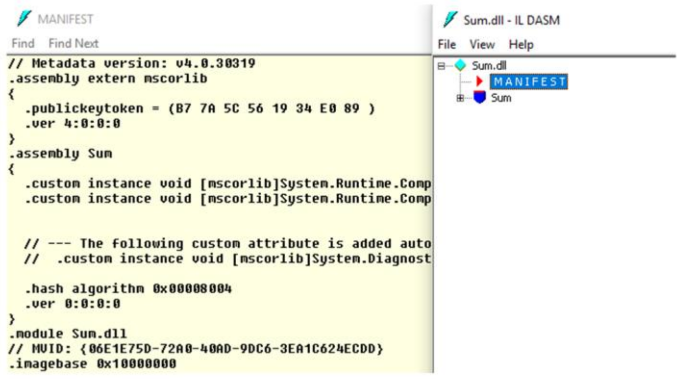
- Say, your friend Mr. Anders Hejlsberg has created an assembly (.dll) file implementing robust and time efficient sorting algorithm for floating point numbers.
- He has shared the assembly with you.
- Can you use (i.e. access sorting method) it in your program? If no, why? If yes, how?

ILDASM



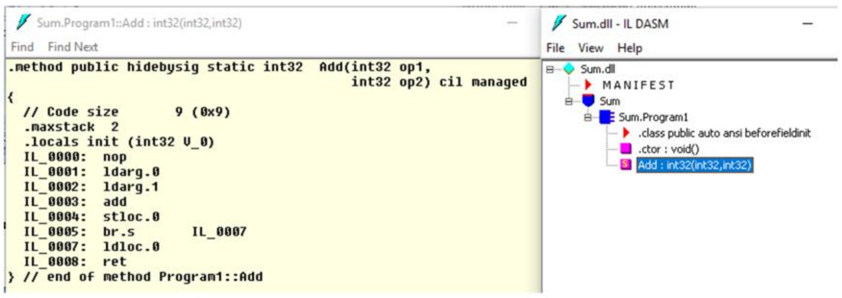
```
E:\GoogleDrive\Personal\BTech\WDDN\2022-23\Labs\Lab2\ex3>ildasm Sum.dll
```

ILDASM

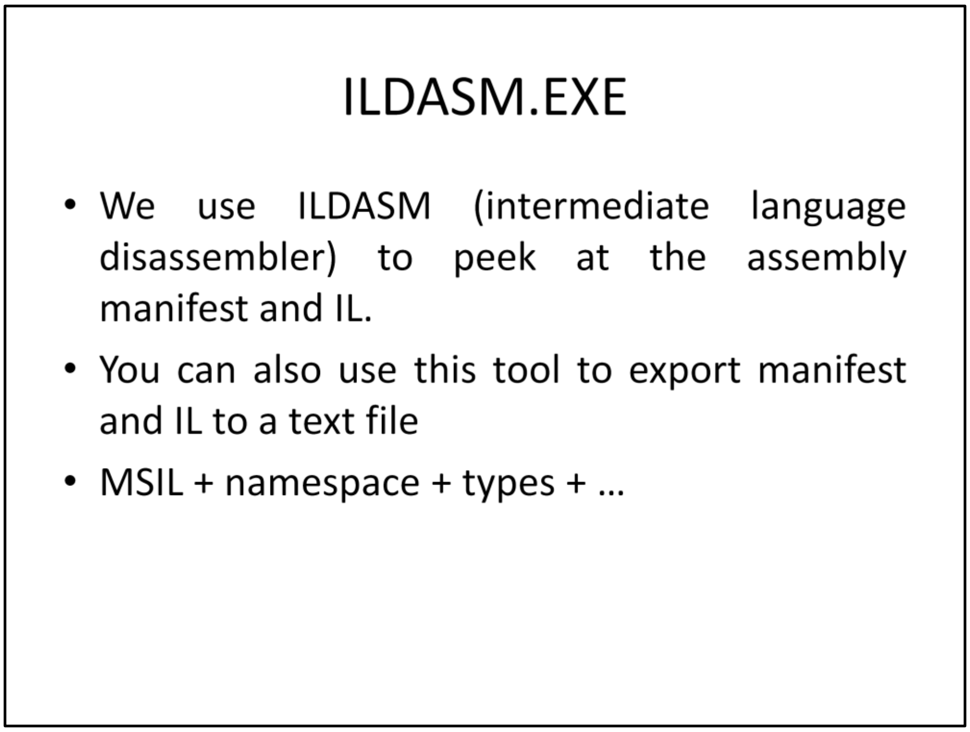


```
// Metadata version: v4.0.30319
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 4:0:0:0
}
.assembly Sum
{
  .custom instance void [mscorlib]System.Runtime.Comp
  .custom instance void [mscorlib]System.Runtime.Comp


  // --- The following custom attribute is added auto
  //  .custom instance void [mscorlib]System.Diagnost

  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module Sum.dll
// MVID: {06E1E75D-72A0-40AD-9DC6-3EA1C624ECDD}
.imagebase 0x10000000
```

ILDASM



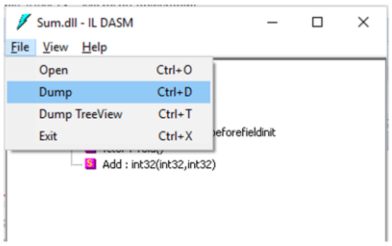
```
.method public hidebysig static int32  Add(int32 op1,
                                           int32 op2) cil managed
{
  // Code size       9 (0x9)
  .maxstack  2
  .locals init (int32 V_0)
  IL_0000:  nop
  IL_0001:  ldarg.0
  IL_0002:  ldarg.1
  IL_0003:  add
  IL_0004:  stloc.0
  IL_0005:  br.s       IL_0007
  IL_0007:  ldloc.0
  IL_0008:  ret
} // end of method Program1::Add
```

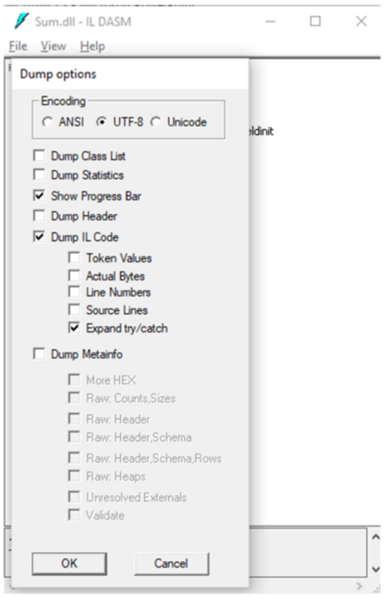

ILDASM.EXE

- We use ILDASM (intermediate language disassembler) to peek at the assembly manifest and IL.
- You can also use this tool to export manifest and IL to a text file
- MSIL + namespace + types + …

C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools
Disassembler (Ildasm.exe) is included with the .NET Framework SDK. The Ildasm.exe parses any .NET Framework .exe or .dll assembly, and shows the information in human-readable format.
Ildasm.exe shows more than just the Microsoft intermediate language (MSIL) code — it also displays namespaces and types, including their interfaces

ILDASM



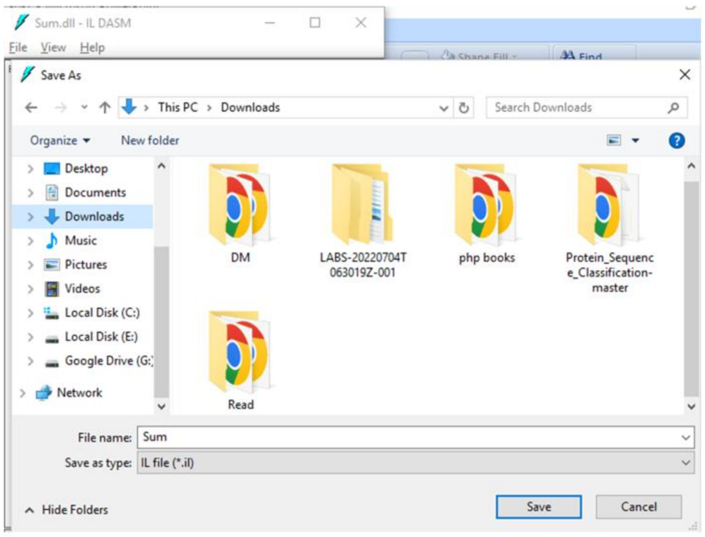


Dump options

Encoding: ANSI [✓ UTF-8] Unicode

- [ ] Dump Class List
- [ ] Dump Statistics
- [x] Show Progress Bar
- [ ] Dump Header
- [x] Dump IL Code
  - [ ] Token Values
  - [ ] Actual Bytes
  - [ ] Line Numbers
  - [ ] Source Lines
  - [x] Expand try/catch
- [ ] Dump Metainfo
  - [ ] More HEX
  - [ ] Raw: Counts,Sizes
  - [ ] Raw: Header
  - [ ] Raw: Header,Schema
  - [ ] Raw: Header,Schema,Rows
  - [ ] Raw: Heaps
  - [ ] Unresolved Externals
  - [ ] Validate

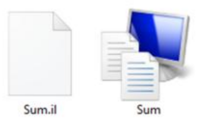
ILDASM

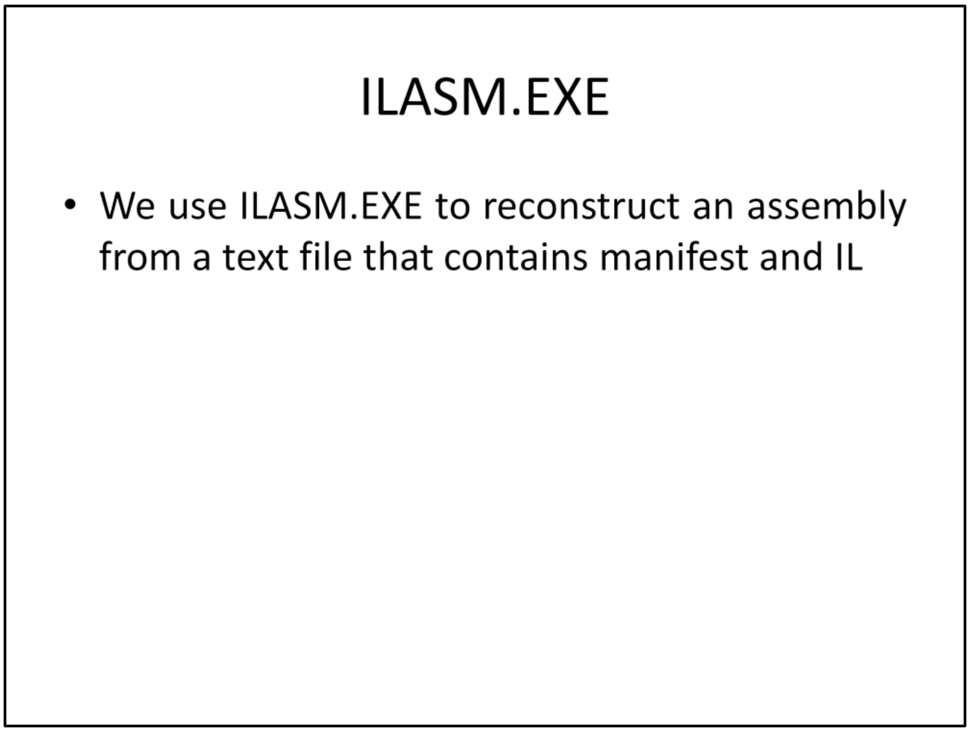


File name: Sum

Save as type: IL file (*.il)

ILDASM



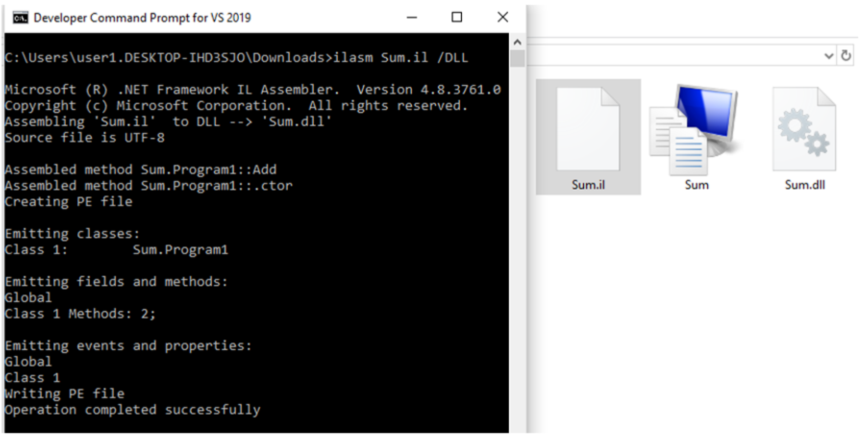


ILASM.EXE

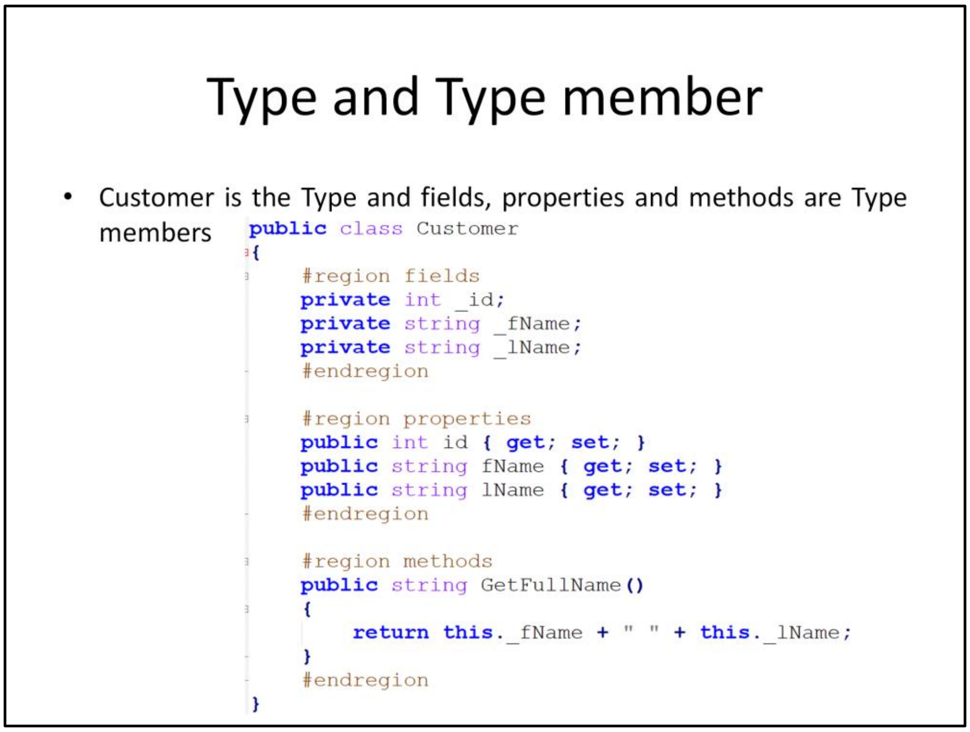
- We use ILASM.EXE to reconstruct an assembly from a text file that contains manifest and IL

ildasm file.exe or ildasm file.dll
dump il file
ilasm file.il will generate respective .dll or .exe file

ILASM.EXE



```
C:\Users\user1.DESKTOP-IHD3SJO\Downloads>ilasm Sum.il /DLL

Microsoft (R) .NET Framework IL Assembler.  Version 4.8.3761.0
Copyright (c) Microsoft Corporation.  All rights reserved.
Assembling 'Sum.il'  to DLL --> 'Sum.dll'
Source file is UTF-8

Assembled method Sum.Program1::Add
Assembled method Sum.Program1::.ctor
Creating PE file

Emitting classes:
Class 1:        Sum.Program1

Emitting fields and methods:
Global
Class 1 Methods: 2;

Emitting events and properties:
Global
Class 1
Writing PE file
Operation completed successfully
```



Type and Type member

- Customer is the Type and fields, properties and methods are Type members

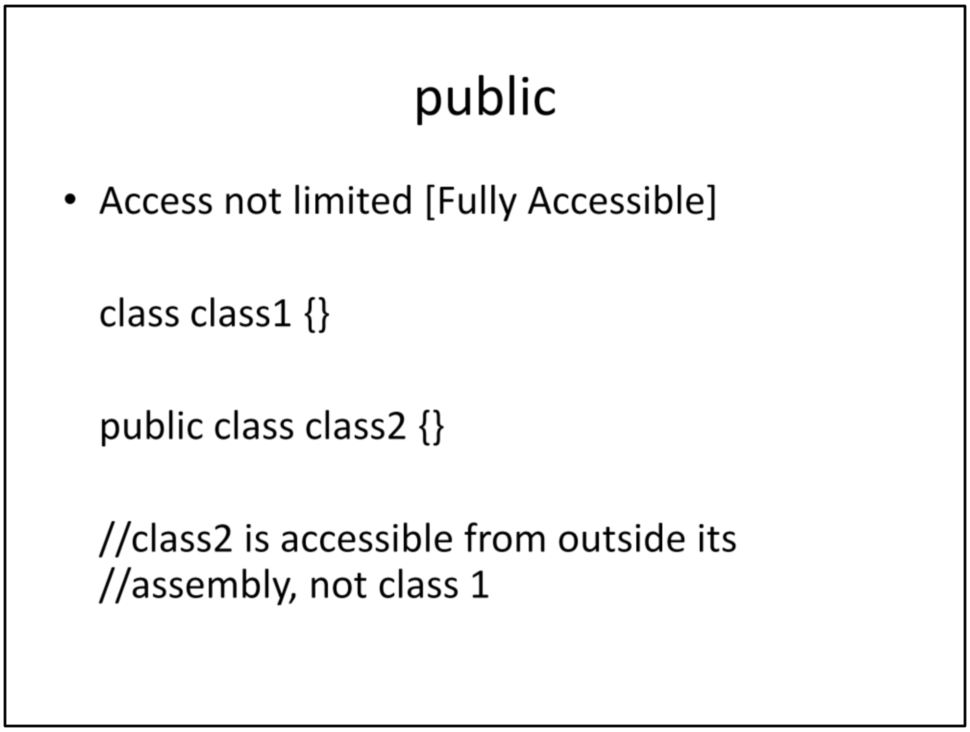
```
public class Customer
{
    #region fields
    private int _id;
    private string _fName;
    private string _lName;
    #endregion

    #region properties
    public int id { get; set; }
    public string fName { get; set; }
    public string lName { get; set; }
    #endregion

    #region methods
    public string GetFullName()
    {
        return this._fName + " " + this._lName;
    }
    #endregion
}
```

Type and Type member

- In general classes, structs, enums, interfaces, delegates are called as types and fields, properties, constructors, methods etc..., that normally reside in a type are called as type members
- In C# there are 5 different access modifiers
  - Private
  - Protected
  - Internal
  - Protected Internal
  - Public
- Type members can have all the access modifiers, where as types can have only 2 (internal, public) of the 5 access modifiers

Access specifiers/ modifiers

- Promote encapsulation
  - Type or type member may limit the accessibility to other types and assemblies
- There are 5 different access modifies in C#
  - Public
  - Private
  - Protected
  - Internal
  - Protected Internal



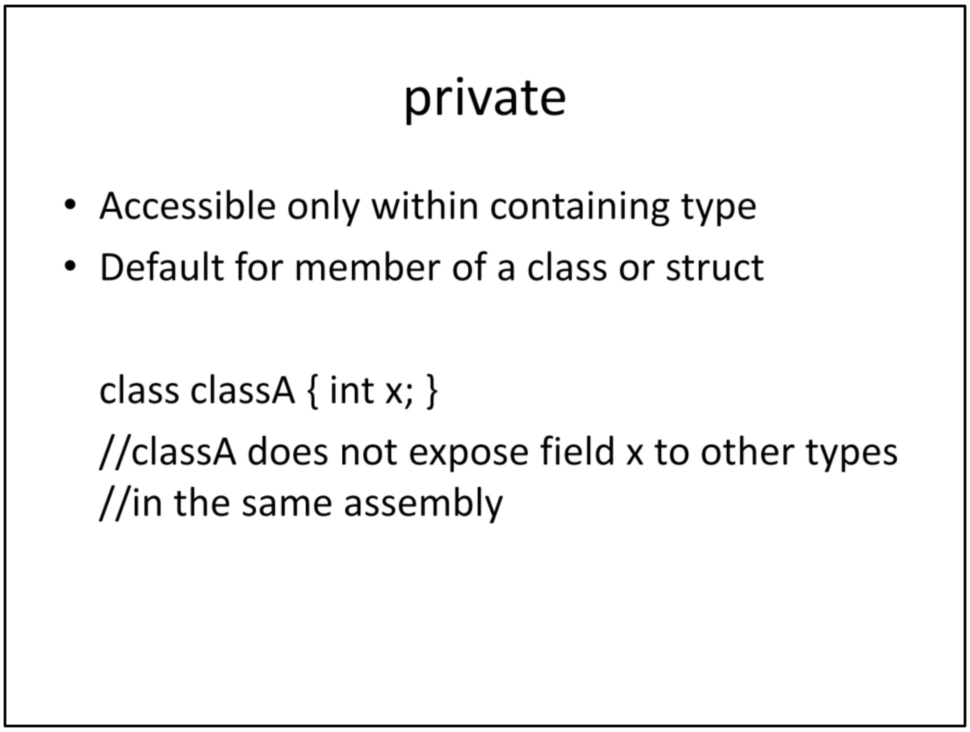
public

- Access not limited [Fully Accessible]

```
class class1 {}

public class class2 {}

//class2 is accessible from outside its
//assembly, not class 1
```

Public access specifier allows a class to expose its member variables and member functions to other functions and objects. Any public member can be accessed from outside the class.

Class1 is internal (default)

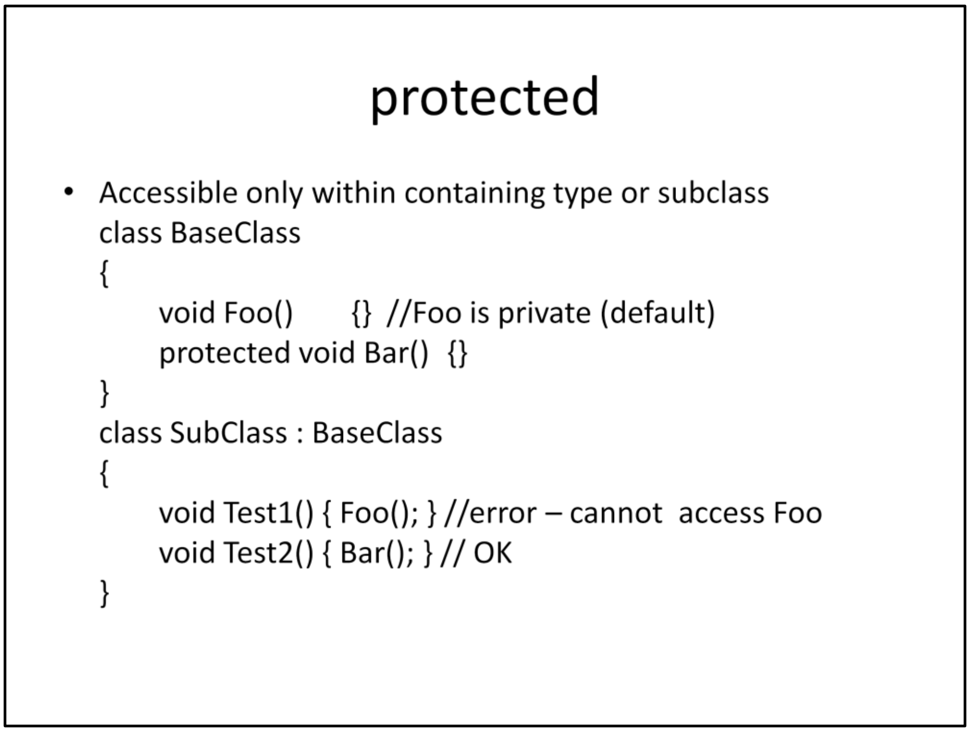


private

- Accessible only within containing type
- Default for member of a class or struct

```
class classA { int x; }
//classA does not expose field x to other types
//in the same assembly
```

X is private (default)

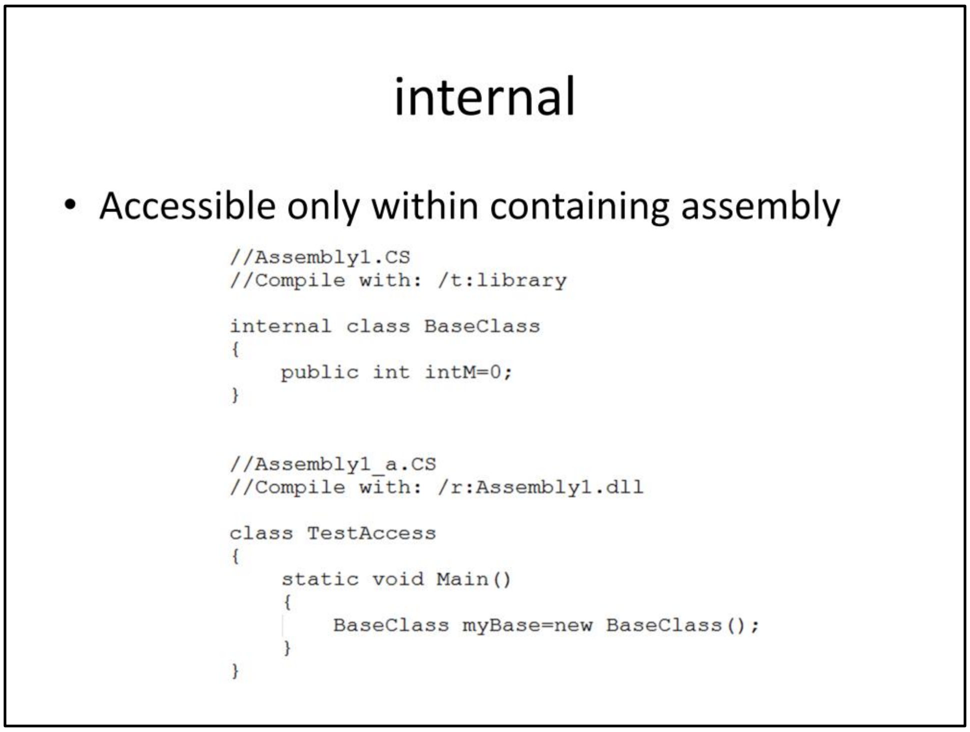


protected

- Accessible only within containing type or subclass

```
class BaseClass
{
      void Foo()       {}  //Foo is private (default)
      protected void Bar()  {}
}
class SubClass : BaseClass
{
      void Test1() { Foo(); } //error – cannot  access Foo
      void Test2() { Bar(); } // OK
}
```

Protected members are available , with in the containing type and to the types that derive from containing type



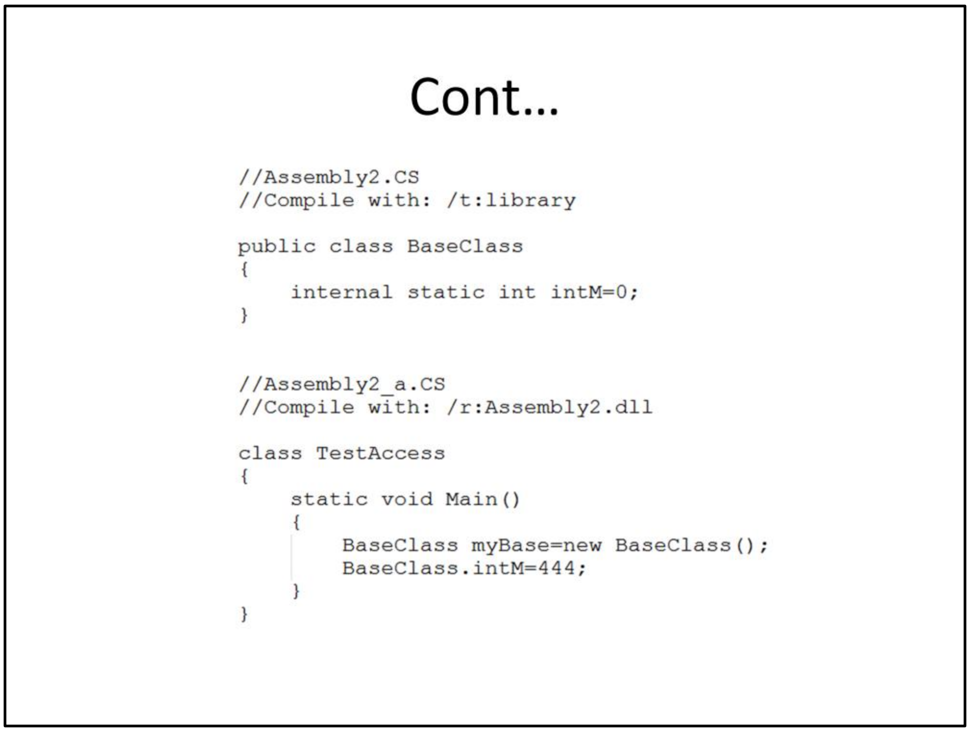
internal

- Accessible only within containing assembly

```
//Assembly1.CS
//Compile with: /t:library

internal class BaseClass
{
    public int intM=0;
}


//Assembly1_a.CS
//Compile with: /r:Assembly1.dll

class TestAccess
{
    static void Main()
    {
        BaseClass myBase=new BaseClass();
    }
}
```

A member with internal access modifier is available anywhere with in the containing assembly. It’s a compile rime error to access, an internal member from outside the containing assembly.

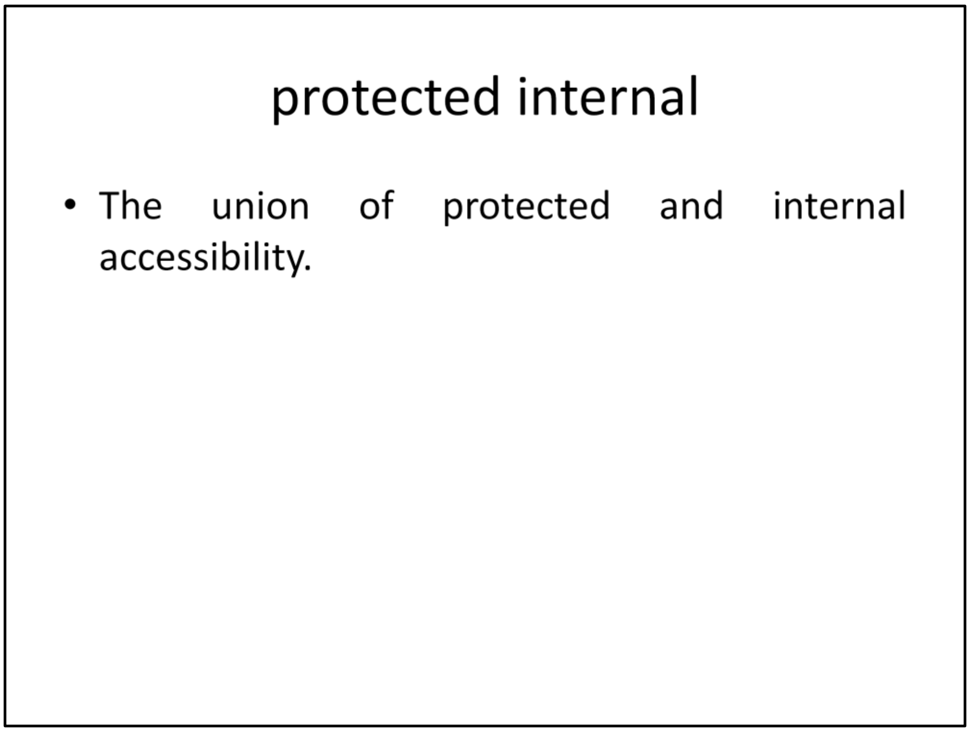
This example contains two files, Assembly1.cs and Assembly1_a.cs. The first file contains an internal base class, BaseClass. In the second file, an attempt to instantiate BaseClass will produce an error.



Cont…

```
//Assembly2.CS
//Compile with: /t:library

public class BaseClass
{
    internal static int intM=0;
}


//Assembly2_a.CS
//Compile with: /r:Assembly2.dll

class TestAccess
{
    static void Main()
    {
        BaseClass myBase=new BaseClass();
        BaseClass.intM=444;
    }
}
```

In this example, use the same files you used in example 1, and change the accessibility level of BaseClass to public. Also change the accessibility level of the member IntM to internal. In this case, you can instantiate the class, but you cannot access the internal member.

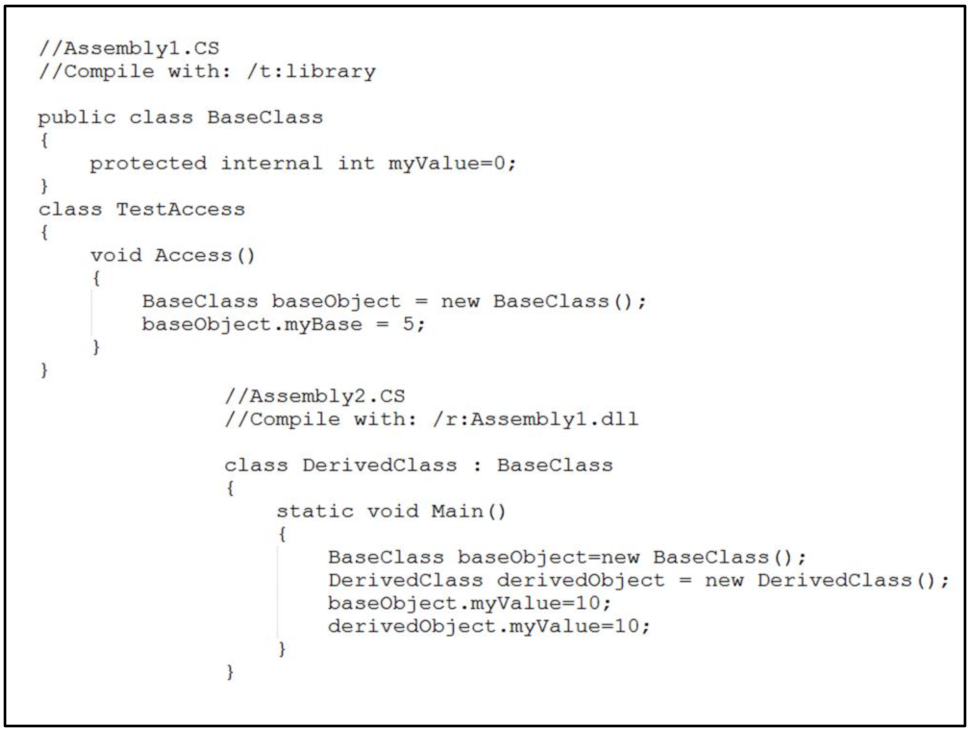


protected internal

- The union of protected and internal accessibility.

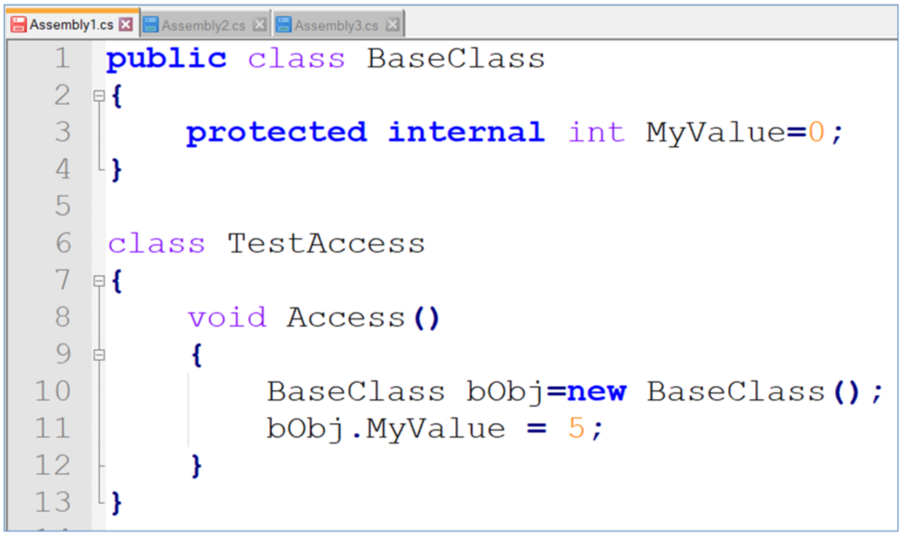
Protected internal members can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly. It is a combination of protected and internal.
A protected internal member of a base class is accessible from any type within its containing assembly. It is also accessible in a derived class located in another assembly only if the access occurs through a variable of the derived class type. For example, consider the following code segment:

This example contains two files, Assembly1.cs and Assembly2.cs. The first file contains a public base class, BaseClass, and another class, TestAccess. BaseClass owns a protected internal member, myValue, which is accessed by the TestAccess type. In the second file, an attempt to access myValue through an instance of BaseClass will produce an error, while an access to this member through an instance of a derived class, DerivedClass will succeed.
Struct members cannot be protected internal because the struct cannot be inherited.



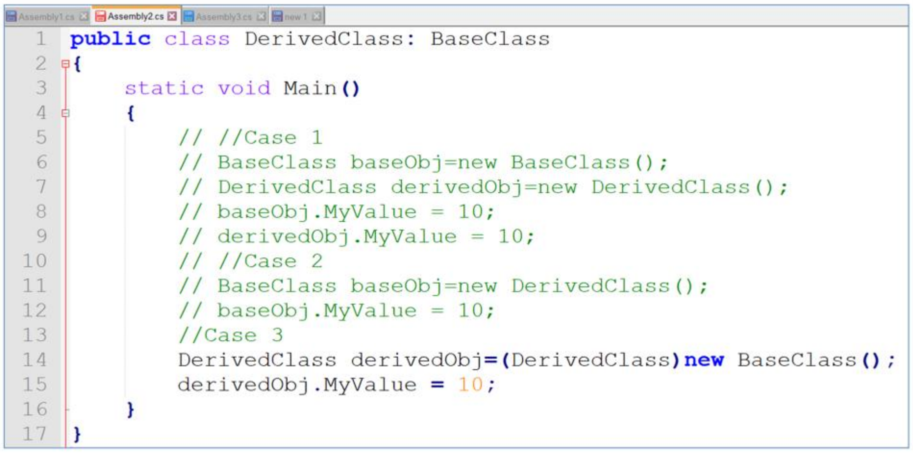
```
//Assembly1.CS
//Compile with: /t:library

public class BaseClass
{
    protected internal int myValue=0;
}
class TestAccess
{
    void Access()
    {
        BaseClass baseObject = new BaseClass();
        baseObject.myBase = 5;
    }
}
```

```
              //Assembly2.CS
              //Compile with: /r:Assembly1.dll

              class DerivedClass : BaseClass
              {
                  static void Main()
                  {
                      BaseClass baseObject=new BaseClass();
                      DerivedClass derivedObject = new DerivedClass();
                      baseObject.myValue=10;
                      derivedObject.myValue=10;
                  }
              }
```


Assembly1.cs Assembly2.cs Assembly3.cs

```
public class BaseClass
{
    protected internal int MyValue=0;
}

class TestAccess
{
    void Access()
    {
        BaseClass bObj=new BaseClass();
        bObj.MyValue = 5;
    }
}
```

More examples
Assembly1.cs


Assembly1.cs Assembly2.cs Assembly3.cs new 1

```
public class DerivedClass: BaseClass
{
    static void Main()
    {
        // //Case 1
        // BaseClass baseObj=new BaseClass();
        // DerivedClass derivedObj=new DerivedClass();
        // baseObj.MyValue = 10;
        // derivedObj.MyValue = 10;
        // //Case 2
        // BaseClass baseObj=new DerivedClass();
        // baseObj.MyValue = 10;
        //Case 3
        DerivedClass derivedObj=(DerivedClass)new BaseClass();
        derivedObj.MyValue = 10;
    }
}
```

Assembly2.cs
// //Case 1

// BaseClass baseObj=new BaseClass();
// DerivedClass derivedObj=new DerivedClass();

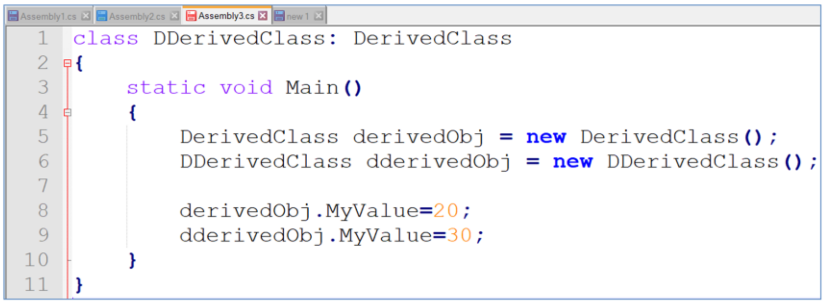
// baseObj.MyValue = 10; //Error
// derivedObj.MyValue = 10; //Works

// //Case 2
// BaseClass baseObj=new DerivedClass();

// baseObj.MyValue = 10; //Error

//Case 3
DerivedClass derivedObj=(DerivedClass)new
BaseClass();

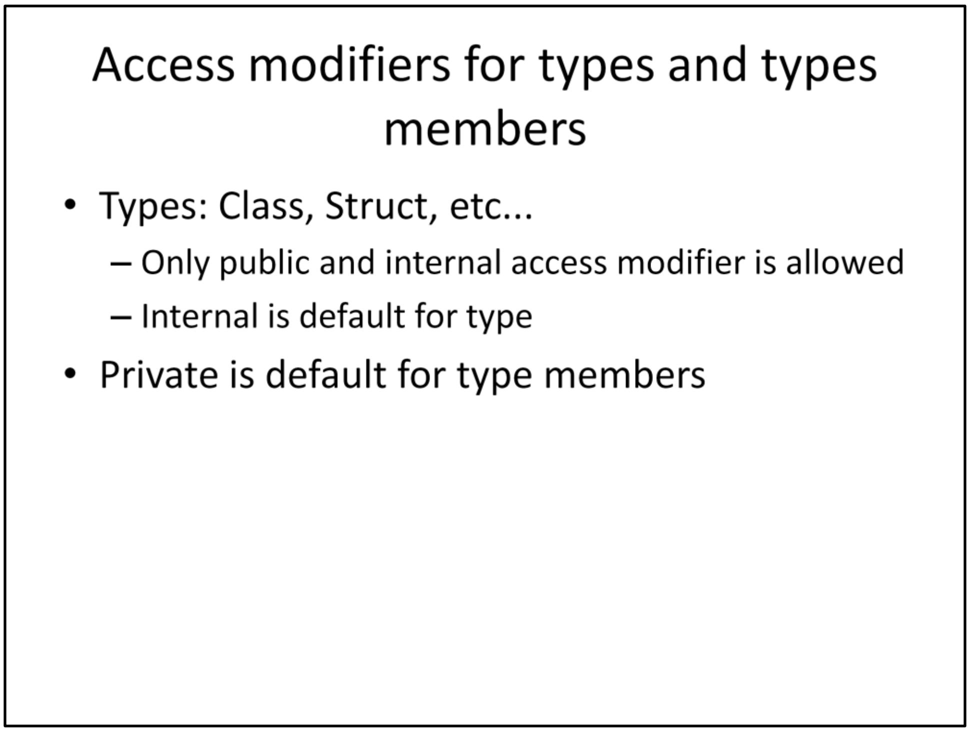
derivedObj.MyValue = 10; //Works



```
class DDerivedClass: DerivedClass
{
    static void Main()
    {
        DerivedClass derivedObj = new DerivedClass();
        DDerivedClass dderivedObj = new DDerivedClass();

        derivedObj.MyValue=20;
        dderivedObj.MyValue=30;
    }
}
```

Assembly3.cs
DerivedClass derivedObj = new DerivedClass();
DDerivedClass dderivedObj = new DDerivedClass();

derivedObj.MyValue=20; //Error
dderivedObj.MyValue=30; //Works



Access modifiers for types and types members

- Types: Class, Struct, etc...
  - Only public and internal access modifier is allowed
  - Internal is default for type
- Private is default for type members