IT 528
Developing .NET Applications Using C#

Gülşen Demiröz

Summary of the Course

- Hands-on applications programming course
- We will learn how to develop applications using the C# programming language on Microsoft ®.NET Platform
- We will also learn many classes from the Microsoft ®.NET Framework Library

Course Information

- **Website:** http://myweb.sabanciuniv.edu/gulsend/su_current_courses/it-528/
- **Instructor:** Gülşen Demiröz, FENS L015, x9559, gulsend@sabanciuniv.edu
- **Lectures:** Thursdays 19:00-22:00, Karakoy Center
  Saturdays 13:00 - 16:00, FENS G032
- **Textbooks**
  - *Visual C# 2012 How to Program*, 5th Edition, (ISBN: 0132151421), by Harvey & Paul Deitel
  - *C# 2010 for Programmers*, 3rd Edition, (ISBN: 0132618206), by Paul J. Deitel & Harvey M. Deitel
  - *Inside C#*, 2nd Edition, (ISBN: 0735616485), by Tom Archer & Andrew Whitechapel
  - *CLR via C#*, 2ndEdition,(ISBN: 0137144156), by Jeffrey Richter
- **Lecture Notes:** http://myweb.sabanciuniv.edu/gulsend/su_current_courses/it-528/lecture-notes/
  - I can also upload them to SUCourse if you wish
- **Grading:**
  Midterm (30%): 4th week of the course (12 October Saturday 13:00)
  Final Exam (40%): Last (7th) week of the course (14 November Thursday 19:00)
  Homeworks (30% total): 2 homework will be assigned and they are of equal weight
- **Homework:** programming homework, zip the whole solution and send it to me via SUCourse
- **Exams:** programming exams on your laptops in the class, then you e-mail me

About Me & then You

- **Work Experience**
  - 1997-2008 Microsoft Corporation, Redmond WA, USA
    - Senior Development Lead (*Microsoft Online Services*)
    - Senior Software Design Engineer (*Office Outlook*)
    - Software Test Lead (*Windows Networking*)
    - Software Design Engineer (*Windows Networking*)
- **Education**
  - Ph.D. student, Sabanci University, Computer Engineering and Information Science, 2011-Present
  - M.Sc., Bilkent University, Computer Engineering and Information Science, 1997
  - B.S., Bilkent University, Computer Engineering and Information Science, 1995

Course Outline

- Introduction (algorithms, programming languages, .NET Platform, Common Language Runtime, Framework, assemblies, packaging)
- How to use Visual Studio® 2012
- A program's structure, basic data types, arithmetic operations, assignment, implicit casting
- .NET Type System (value types vs. reference types), memory concepts, garbage collector (GC)
- Classes I (constructors-destructor, properties, access modifiers)
- Methods (overloading, pass-by-reference, scope of variables, static methods, operator overloading)
- Control statements (if-else, switch, while, for, do-while)
- Classes II (inheritance, abstract classes, interfaces, is-as)
- Arrays, Collections (foreach, indexers, anonymous types, introduction to LINQ)
- Exception Handling
- Delegates and Event Handlers
- Windows® Forms and introduction to Windows Presentation Foundation
- Strings and StringBuilder
- Files and Streams
- Generics, Generic Collections
- XML and LINQ to XML
- Database access (with ADO.NET and LINQ to SQL)

Before we start, let's install Visual Studio 2012

- Fast Integrated Development Environment (IDE)
- Very good user interface (UI) design
  - easy to find compiler errors and debugging

- Let's install it, detailed instructions on course's web site:
  http://myweb.sabanciuniv.edu/gulsend/su_current_courses/it-528/

- Also install SQL Server 2012 Express from
  http://www.microsoft.com/en-us/download/details.aspx?id=29062

Algorithms

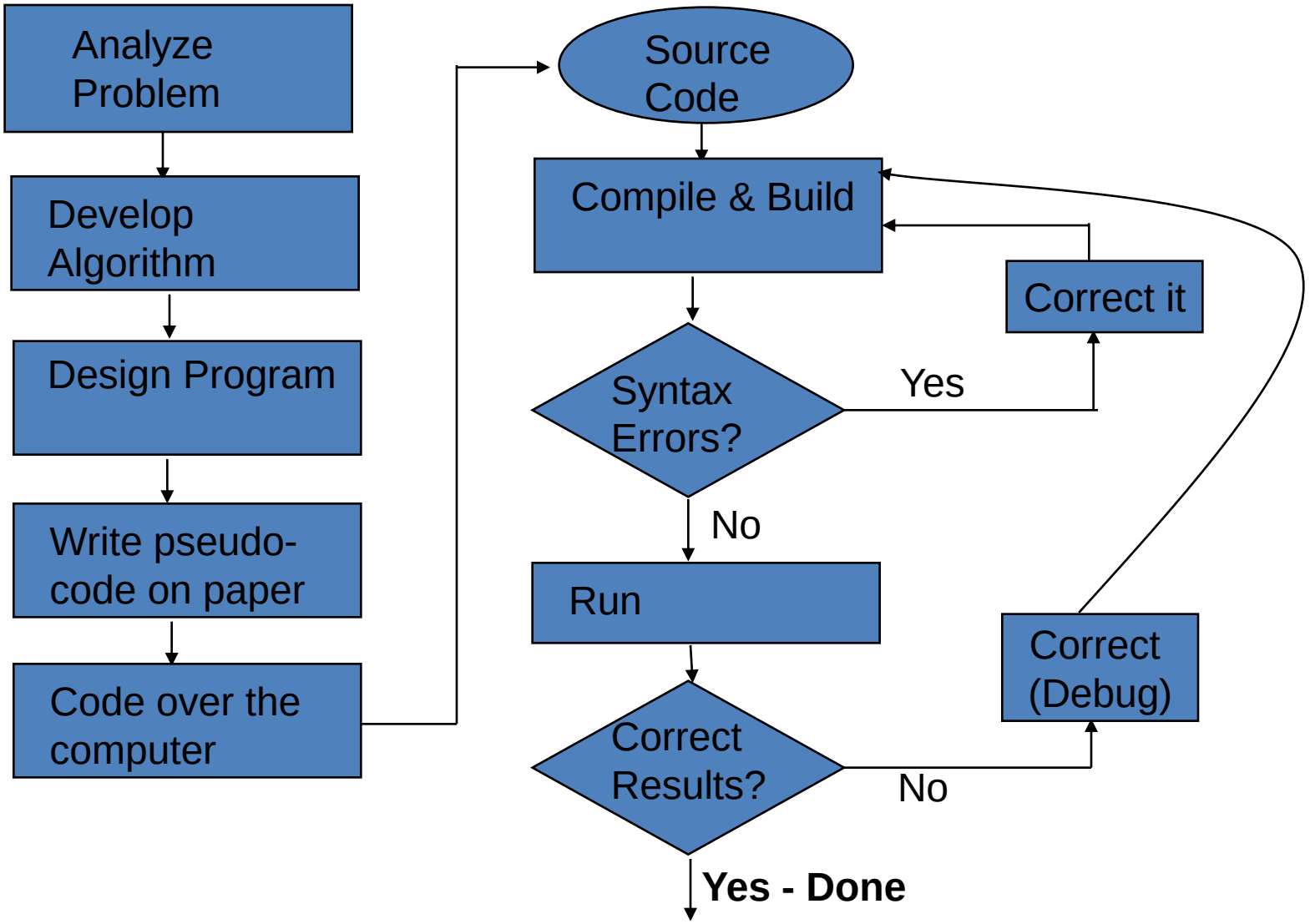
- Arabic-originated word
- Step-by-step process that solves a problem
  - do this, then do that, ...
  - eventually stops with an answer
  - general process rather than specific to a programming language
- Example: cooking pasta (makarna)
- Issues
  - correctness
  - complexity and efficiency
- I picked a number between 1 and 100
  - You will guess it
  - I'll respond "high", "low", "correct".
  - how many guesses needed (worst case)?

Example Algorithm - Find the minimum

- Initial list: 4 6 7 3 9 1 4 5

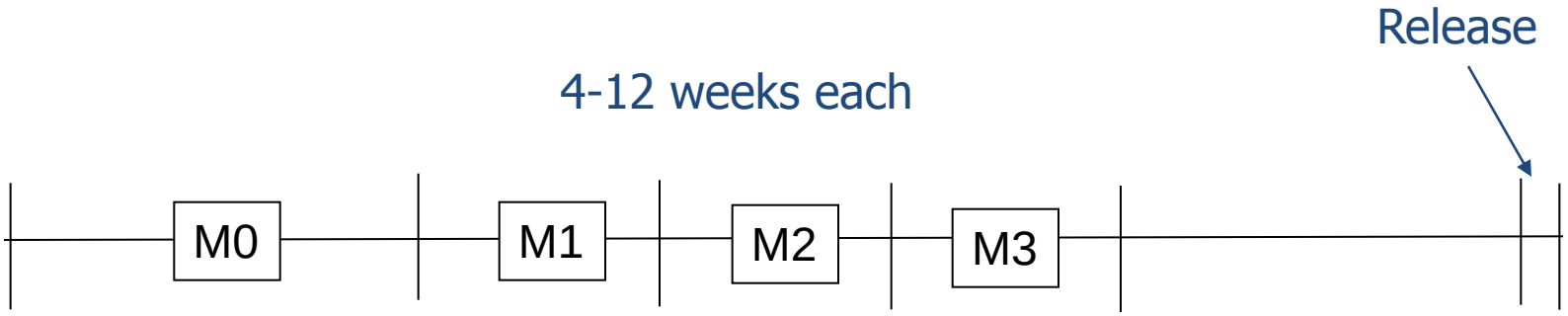
- Should we sort? 1 3 4 4 5 6 7 9
  - The minimum is the first one
- Optimal algorithm - About **n** operations
  - Pick 4 as the minimum
  - Compare 4 to 6 - min is still 4
  - Compare 4 to 7- min is still 4
  - Compare 4 to 3 - Pick 3 as the minimum
  - Compare 3 to 9- min is still 3
  - Compare 3 to 1 - Pick 1 as the minimum
  - Compare 1 to 4- min is still 1
  - Compare 1 to 5 - **We are done and the minimum is 1**

Basic Program Development Steps



Development at Microsoft

- I will talk more about it whenever we get a chance



- **Plan**
- **Schedule**
- **Design**
  (Architecture)

Implement
- Code reviews
- Unit testing

Stabilize
- Testing
- Bug fixing

Programming Languages

- We solve problems with algorithms
- Then we use computers to run these algorithms
- For this, we need programming languages to interact with the computer's hardware
- Computers represent data in numeric format
  - Internal representation (at the lowest level) is in *binary* form: 0 and 1 (4=100, 5=101)
  - 0 and 1's are stored in a *bit,* 8 bits is called a *byte*
- Programs are set of instructions that process data
- These low level instructions are also in binary (0 and 1)
  - machine language: not human readable and programmable!
- Rather than instruct computers at the level of 0's and 1's, higher level languages have been developed.
  - Flexible and easier programming
- Compilers translate a high level language, such as *C*, into machine-specific executable program (0^s and 1^s)

C, C++ and Java

- **C** first gained widespread recognition as the development language of the UNIX operating system.
- **C++** took the C language and provided capabilities for **object-oriented programming (OOP)**.
- **Objects** are reusable software **components** that model items in the real world.
  - Object-oriented programs are often easier to understand, correct and modify.
- Sun Microsystems began development of the **Java** programming language in 1991.
- Java is now used to develop large-scale enterprise applications.

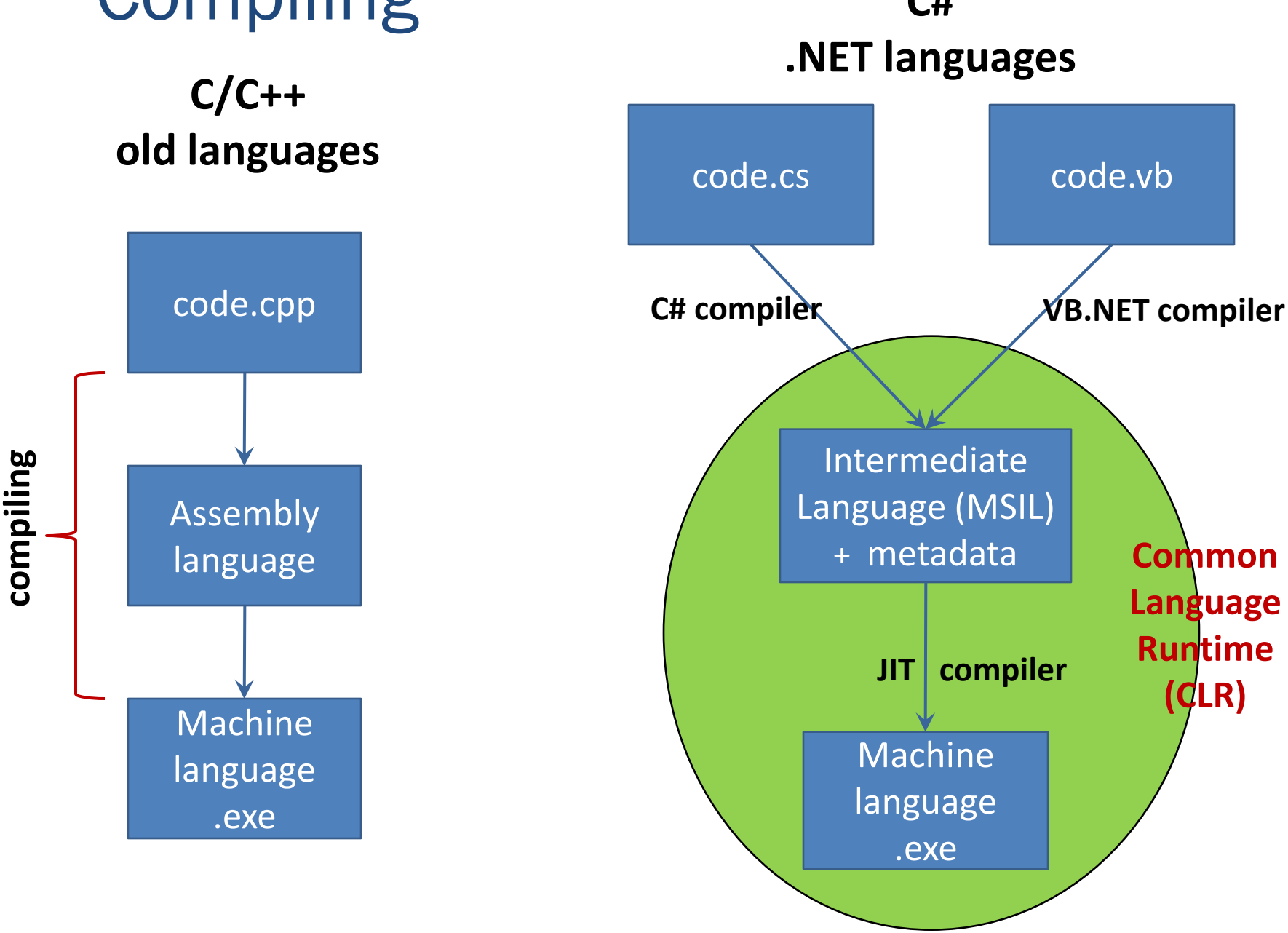
C# (read as “C Sharp”)

- C# was designed specifically for the .NET platform as a language that would enable programmers to migrate easily to .NET.
- C# is object oriented and has access to a powerful **class library** of prebuilt components.
- It has roots in C, C++ and Java, adapting the best features of each.
- Microsoft introduced C# along with its .NET strategy in 2000.
- The **.NET platform** allows applications to be distributed to a variety of devices.

.NET Platform

- Provides programmers to develop their components in any language and easily share them (old way: COM).
- Very rapid development with the help of already built-in classes or platforms.
- Applications in any .NET-compatible language can interact with each other.
- The .NET strategy allows programmers to concentrate on their specialties without having to implement every component of every application.
- End to DLL hell with versioning.

Compiling


C/C++
old languages
code.cpp
compiling
Assembly language
Machine language .exe
C#
.NET languages
code.cs
code.vb
C# compiler
VB.NET compiler
Intermediate Language (MSIL) + metadata
JIT compiler
Machine language .exe
Common Language Runtime (CLR)

Common Language Runtime (CLR)

- Programs are compiled first into **Microsoft Intermediate Language** (**MSIL**) and **metadata**. This is called a ***managed module***.
- When this managed application runs, first the CLR's mscoree.dll's _CorExeMain function is executed
- Then the **just-in-time (JIT) compiler** translates the MSIL in the executable file into machine-language code.
  - CLR does not need to know which language was used.
- Once the code is compiled into machine-language by the JIT, it is not needed to be compiled again.
- End users need CLR on their machine to execute managed code, which comes with the .NET Framework installation.

Parts of a Managed Module

- Managed module is a ***PE (portable executable)*** that requires CLR to execute.
- It contains:
  - PE32 or PE32+ header (32-bit vs. 64-bit)
  - CLR header
  - Metadata
  - IL (Intermediate Language) code

Automatic Memory Management

- One of the services that the common language runtime provides during **Managed Execution.**
- Allocation and releasing of memory is managed by the CLR: **Garbage collection.**
  - No more memory leaks ☺

Common Type System (CTS)

- The common type system defines how types are declared, used, and managed in the runtime.
- A *type* contains zero or more members:
  - Field
  - Method
  - Property
  - Event
- A *type*'s members can have the following accessibility:
  - Private
  - Family (*protected*)
  - Family and assembly
  - Assembly (*internal*)
  - Family or assembly
  - Public
- Important part of the support for cross-language integration.
  - CTS together with Common Language Specification (CLS) enables cross-language integration.

.NET Framework Class Library (FCL)

- Set of classes, interfaces, and value types that exposes some functionality for re-use.
- The foundation on which .NET Framework applications, components, and controls are built.
- Thousands of types are organized into **namespaces**
  - Example: **`Object`** base type and types for integers, characters are in the **`System`** namespace
- Uses a dot syntax naming scheme that connotes a hierarchy.
  - Groups related types into namespaces so they can be searched and referenced more easily.
  - The first part of the full name — up to the rightmost dot — is the namespace name.
  - The last part of the name is the type name.
  - Example: **`System.Collections`**`.`**`ArrayList`**

.NET Framework Class Library (FCL)

- We will use and learn classes from some of these FCL libraries in this class.
- Examples:
  - `System`
  - `System.Collections`
  - `System.IO`
  - `System.Windows.Forms`
  - `System.Linq`
  - `System.Net`
  - `System.Text`
- Full list:
  http://msdn.microsoft.com/en-us/library/ms229335.aspx

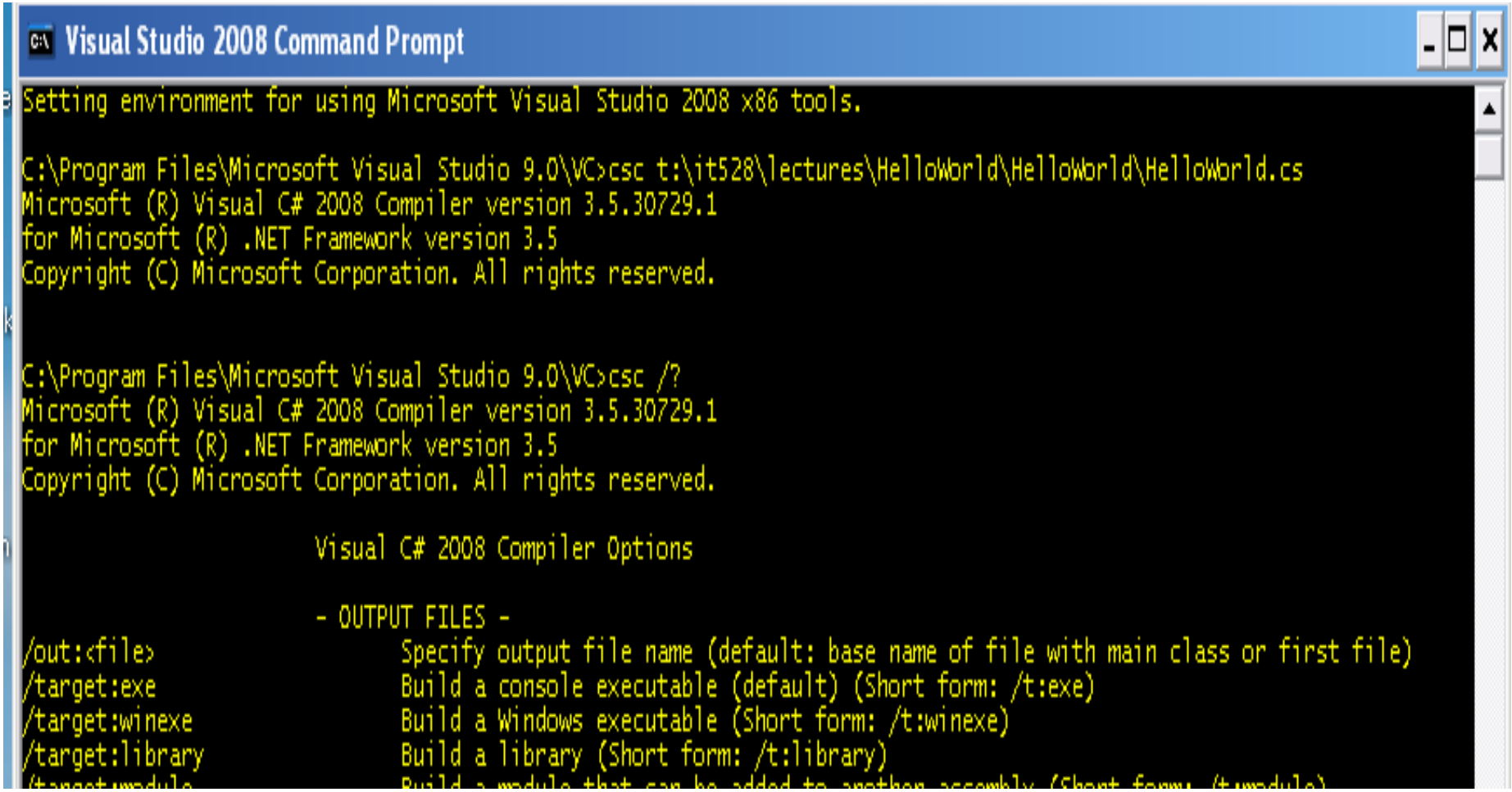
Visual Studio 2012

- Fast Integrated Development Environment (IDE)
- Very good user interface (UI) design
  - easy to find compiler errors and debugging
- Heavy visual support to develop UI
  - Drag and drop controls for WinForms and ASP.NET
- Easy tools to access databases and view data relationships
- Let's install it, detailed instructions on course's web site:
  http://myweb.sabanciuniv.edu/gulsend/su_current_courses/it-528/

“Hello World” Program

- Let’s develop our very first application using Visual Studio 2012
  - Create a project
  - Build, compile, run and debug
  - Useful windows and customizing its locations
    - Solution Explorer
    - Toolbox
    - Properties
    - Error List
    - Debugging windows
  - Intellisense
  - Menu and the toolbar
    - Enable Line numbers: Tools\Options\Text Editor\All Languages\Line numbers checkbox.
  - Help and MSDN

C# command-line compiler: csc.exe



What is an Assembly?

- When we compiled HelloWorld.cs using C# compiler, we created an assembly called HelloWorld.exe
- An assembly is a .NET unit of modules put together that the runtime (CLR) can run
- An assembly could be:
  - EXE (/target:exe or /target:winexe)
  - DLL (/target:library)
  - Module (/target:module)
- Visual Studio generates either an EXE or a DLL.
- An assembly could be a single file or contain multiple files
  - Multiple files could be .NET modules or resource files (gif/jpg)
  - csc /addmodule:*<file list>*

What is in an Assembly?

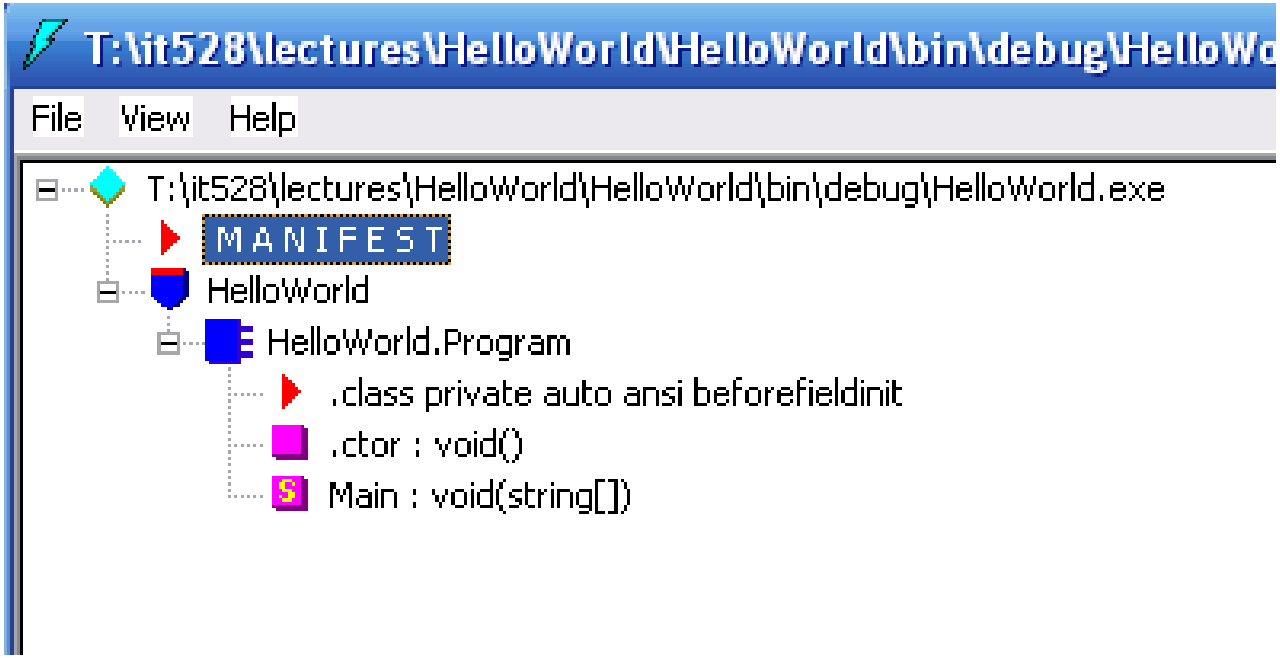
- **Summary***:* an assembly is a .NET executable file with one or more type definitions and resources in it.
- An assembly is the smallest unit of deployment in .NET
  - performance improvement to load multiple modules in one assembly
- An assembly contains a manifest to describe itself to the runtime → self-describing

Assembly Manifest

- Assembly name
- Versioning information
  - major and minor version, revision and build number
- Culture (language)
- Shared name (optional) and signed assembly hash
- List of files that exist in the assembly
- Referenced assemblies
- Types
  - All types in the assembly with a mapping to the module containing the type
- Security
  - List of security permissions refused by the assembly
- Custom attributes
- Product information
  - Company, Trademark, Product and Copyright

.NET Tool: ildasm.exe

- Let's analyze "Hello World" program with ildasm.exe



Deployment of Assemblies

- Private Assemblies
  - This is the default
  - You just copy them to a folder

- Public (Shared) Assemblies
  - Needs a *shared* (*strong*) name, why?
  - Needs to be signed with a public key:
    - You can use Project Properties\Signing tab
    - Or you can use the Strong Name tool (`sn -k IT528Key.key`) to create a key and then
      - Use the al.exe tool with /keyfile option OR
      - Add *AssemblyKeyFile* attribute to the source file
  - After signing, now you can share this assembly

Example: HelloWorldLibrary.dll

- Let's create a library (DLL)
- `csc /target:library c:\Users\gulsen\Documents\it528\week1\HelloWorldLibrary\HelloWorldLibrary\HellowWorldLibrary.cs`

Public Assemblies and GAC

- Global Assembly Cache (GAC) is a code cache
  - Code downloaded from the Internet or other servers
  - Components shared by multiple .NET applications
  - Your code that has been JIT'ted the first time it's run
- Where is it?

```
C:\WINDOWS\assembly>dir %windir%\assembly
 Volume in drive C is System
 Volume Serial Number is BCBC-E9D7

 Directory of C:\WINDOWS\assembly

02/26/2009  01:36 PM    <DIR>          GAC
02/26/2009  01:36 PM    <DIR>          GAC_32
02/26/2009  01:36 PM    <DIR>          GAC_MSIL
02/26/2009  01:36 PM    <DIR>          NativeImages_v2.0.50727_32
02/26/2009  01:36 PM    <DIR>          temp
02/26/2009  01:33 PM    <DIR>          tmp
               0 File(s)              0 bytes
               6 Dir(s)  22,953,021,440 bytes free
```

- View the GAC: **`gacutil -l`**
- Install an assembly to GAC:

  `gacutil -i HelloWorldLibrary.dll`
- Uninstall an assembly from the GAC:

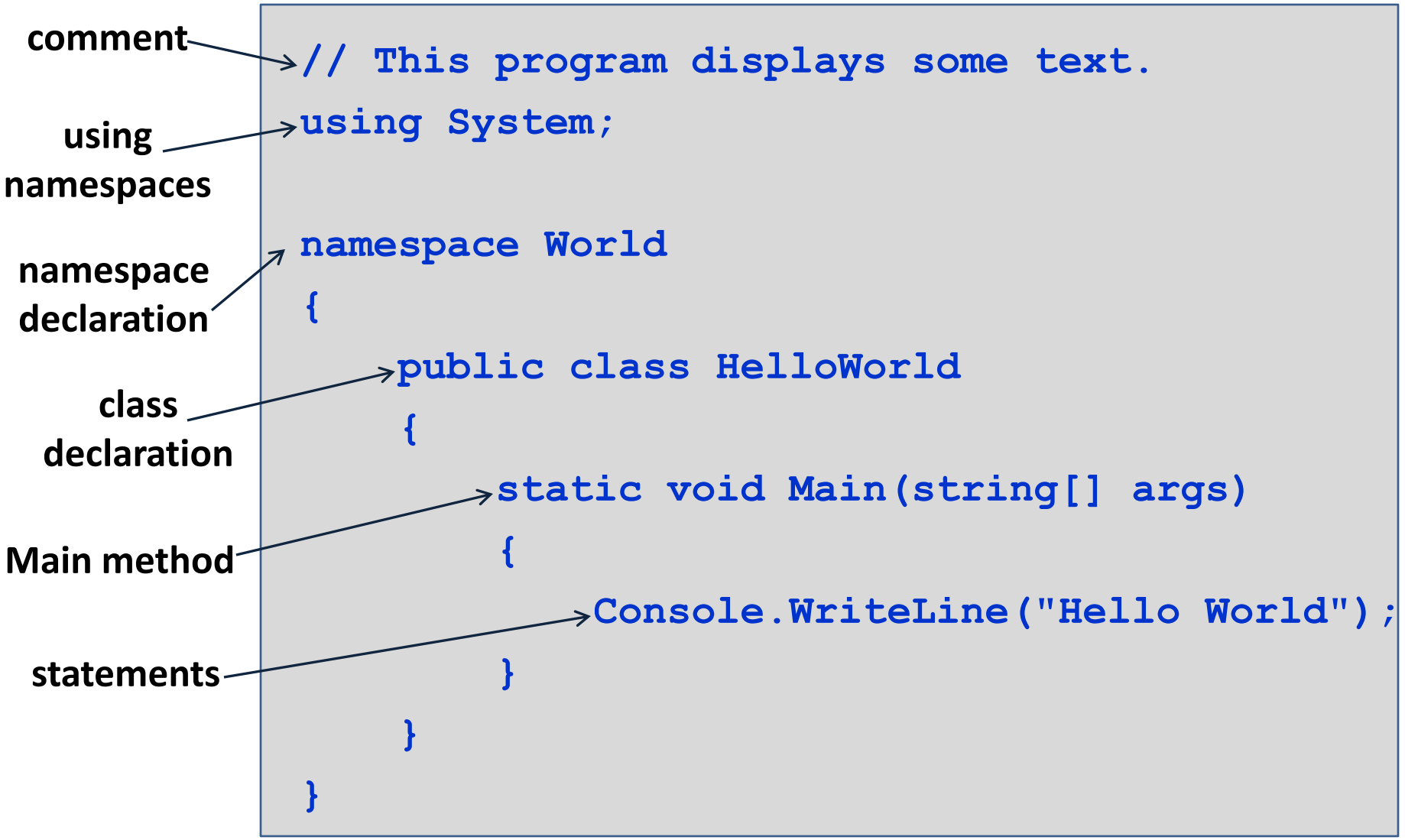
  `gacutil -u HelloWorldLibrary`
- Unfortunately gacutil is not enough to add your library as a reference ☹

Example

- Change HelloWorld.exe to use HelloWorldLibrary.dll
- ildasm HelloWorldLibrary.dll
- Strongly name HelloWorldLibrary.dll
- ildasm HelloWorldLibrary.dll
- Put it in the GAC

```
  .publickey = (00 24 00 00 04 80 00 00 94 00 00 00 06 02 00 00   // .$..............
                00 24 00 00 52 53 41 31 00 04 00 00 01 00 01 00   // .$..RSA1........
                11 20 2A 82 2D 5B 1B 73 EC 74 47 70 7E C9 CA AF   // . *.-[.s.tGp~...
                D4 B1 65 CF 5F 07 FF 75 C6 73 9F 91 7D 4F 69 AD   // ..e._..u.s..}Oi.
                ED 8F 23 19 41 77 3A 8A 89 F4 6C 56 7D FF 2C AB   // ..#.Aw:...lV}.,.
                95 28 D7 56 C0 2B CE 81 BB FA 6E 87 63 04 BF 69   // .(.V.+....n.c..i
                27 24 91 5A 54 B3 63 13 27 2C 47 27 4F 93 DD F6   // '$.ZT.c.',G'O...
                A6 E4 0B B8 25 40 02 13 57 FC B4 AD 43 D0 83 6C   // ....%@..W...C..l
                8F 13 35 EE FD F8 0C 8B DB 95 05 47 6F EB 6E F7   // ..5........Go.n.
                D2 88 A1 25 41 94 DA DC 6D 00 AF 87 7C 8E 06 F7 ) // ...%A...m...|...
  .hash algorithm 0x00008004
  .ver 1:0:0:0
}
.module HelloWorldClassLibrary.dll
// MVID: {98CE8A14-E6EA-4656-BFE1-7F409F9F1713}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       // WINDOWS_CUI
.corflags 0x00000009    //  ILONLY
// Image base: 0x031F0000
```

HelloWorld.cs program's Structure



C# Program's Structure

- C# is 100% object-oriented:
  - Everything is a class → the program itself has to be a class

```
class Program
{ // classes start with a {
  …
} // classes end with a }
```

- Classes are grouped into namespaces
  - You can use existing namespaces by `using` directive

```
using System;
```

  - You can create your own namespace

```
namespace World
{  // namespaces start with a {
   // class definition goes here
} // namespaces end with a }
```

C# Program's Structure

- Programmers use blank lines and space characters to make applications easier to read.
- Together, blank lines, space characters and tab characters are known as **whitespace**. Whitespace is ignored by the compiler.
- Certain indentation makes the code easier to read. You can let the IDE format your code by selecting

  Edit > Advanced > Format Document.
- Set tab size: Tools\Options\Text Editor\C#\Tabs\Tab size.

C# Program's Structure

- Classes have methods (*functions*)
- Methods start with a { and end with a }
- For each application, one of the methods in a class must be called `Main`; otherwise, the application will not execute

```
static void Main(string[] args)
```

  - Main is where the program starts executing
- Methods have <n> statements inside { }

```
static void Main(string[] args)
{
    statement_1;
    …
    statement_n;
}
```

statements must end with ;

- Keyword **void** indicates that this method will not return any information after it completes its task.
- Only statement of Main method in HelloWorld.cs:

```
Console.WriteLine("Hello World");
```

C# Program's Structure

- .NET class library has thousands of methods

```
Console.WriteLine("Hello World");
```

```
using System;
```

```
System.Console.WriteLine("Hello World");
```

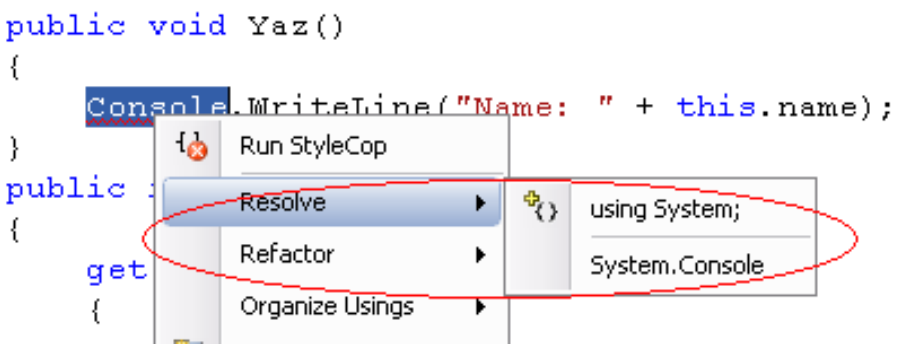
namespace *class* *method* *string*

- **using** directive tells the compiler where to look for a .NET class used in this application
- The **Console.WriteLine method** displays a line of text in the console window.
- The string in parentheses is the **argument** to the **Console.WriteLine method**.
- Method **Console.WriteLine** performs its task by displaying its argument in the console window.

Syntax Errors

- The **syntax** of a programming language specifies the rules for creating a proper application in that language.
- A **syntax error** occurs when the compiler encounters code that violates C#'s language rules**.**
- *Example*: Forgetting to include a using directive for a namespace that contains a class used in your application results in a syntax error, containing a message such as:

  "The name 'Console' does not exist in the current context."
- When this occurs, check that you provided the proper using directives and that the names in the using directives are spelled correctly, including proper use of Uppercase and Lowercase letters.
- To find the namespace:



Comments

- Comments make programs readable by humans (and yourself!)
  - Easier maintenance
  - Try to use natural language, do not repeat the code!
    - Bad example

```
area = pi * r * r; /* area is pi*r*r */
```

    - Better example

```
area = pi * r * r; /* calculate area */
```

    - Best example

```
area = pi * r * r; /* calculate area of a
                      circle of radius r */
```

  - Two ways of commenting
    - Using // make the rest of the line comment

```
area = pi * r * r; // calculate area
```

    - Between /* and */

```
/*
  Calculate area of a circle of radius r
*/
area = pi * r * r;
```

  - Compiler disregards comments
  - Comments in your homework affect your grades
  - In Visual Studio, comments are in **green**

Literals

```
Console.WriteLine("Hello World");
```

string literal

- Fixed (constant) values
  - They cannot be changed during program's execution
- They can be output by `Console.WriteLine`
- Different format for different types:
  - String literals
  - Sequences of characters
  - Within double quotes (quotes are not part of the string)
  - Almost any character is ok (letters, digits, symbols)

```
" 10 > 22 $&*%? "
```

  - Numeric literals
  - Integer

```
3    454    -43    +34
```

  - Real

```
3.1415          +45.44       -54.6      1.2334e3
```

`1.2334e3` is 1.2334 times 10 to the power 3 (scientific notation)

Identifiers

- Names of programmer defined elements in a program
  - Names of classes, methods, variables, etc.

```
namespace World
{
    public class HelloWorld
    {
```

- Syntax (rules):
  1. Sequence of letters (a .. z, A ..Z), digits (0 ..9) underscore _
  2. Cannot start with a digit or underscore
  3. Case-sensitive (**n**umber1 and **N**umber1 are **not** the same)
- Examples:

Program1	valid
number_1	valid
mySum	valid
1number	not valid

- Pick meaningful names to improve readability and understandability of your program (be consistent)

Console Output

```
// This program displays some text.
using System;

namespace World
{
    public class HelloWorld
    {
        static void Main(string[] args)
        {
            Console.Write("Welcome to ");
            Console.WriteLine("C# Programming");
        }
    }
}
```

The `Write` method does not move the cursor to a new line after displaying its argument.

Console Output

- A single statement can display multiple lines by using newline characters.
- Like space characters and tab characters, newline characters are whitespace characters.
- The below application outputs 4 lines of text, using newline characters to indicate when to begin each new line.

```
using System;

namespace World
{
    public class HelloWorld
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome\nto\nC#\nProgramming");
        }
    }
}
```

Console Output

- The **backslash** (\) is called an **escape character**, and is used as the first character in an **escape- sequence.**
- The escape sequence \n represents the **newline character**.

Common Escape Sequences	Description
\n	Newline. Positions the screen cursor at the beginning of the next line.
\t	Horizontal tab. Moves the screen cursor to the next tab stop.
\r	Carriage return. Positions the screen cursor at the beginning of the current line—does not advance the cursor to the next line.
\\	Backslash. Used to place a backslash character in a string.
\“	Double quote. Console.Write(**"\"in quotes\""**); displays "in quotes"

Console Output

- Console methods Write and WriteLine also have the capability to display formatted data.
- Method WriteLine's first argument is a **format string** that may consist of **fixed text** and **format items**.
- Each format item is a placeholder for a value, corresponding to an additional argument to WriteLine.
  - {0} is a placeholder for the first additional argument.
  - {1} is a placeholder for the second, and so on.
- Format items also may include optional formatting information.

```
using System;

namespace World
{
    public class HelloWorld
    {
        static void Main(string[] args)
        {
            Console.WriteLine("{0}\n{1}", "Welcome to", "C# Programming");
        }
    }
}
```

Method `WriteLine`'s first argument is a **format string** that may consist of **fixed text** and **format items**.

Keywords (reserved words)

- Special and fixed meanings
  - built-in in C# language
  - always spelled with all lowercase letters
- You cannot use a reserved word as a user-defined identifier
- Cannot be changed by programmer
- Examples:
  - The `class` keyword introduces a class declaration and is immediately followed by the class name.
  - `using`
  - `namespace`
  - `static`
  - `void`
- Full list: http://msdn.microsoft.com/tr-tr/library/x53a06bb(en-us).aspx
- Identifiers may be preceded by the @ character to interpret a keyword as an identifier (e.g. **`@class`**).

Variables

- A **variable** is a location in the computer's memory where a value can be stored for use later in an application.

Example Program: Addition

```
static void Main(string[] args)
{
   int number1;     // declare first number to add
   int number2;     // declare second number to add
   int sum;         // declare sum of first and second number

   Console.Write("Enter first integer:");
   // read first number from user
   number1 = Convert.ToInt32(Console.ReadLine());

   Console.Write("Enter second integer:");
   // read second number from user
   number2 = Convert.ToInt32(Console.ReadLine());

   sum = number1 + number2; // add numbers
   Console.WriteLine("The sum of {0} and {1} is {2}",
                     number1, number2, sum);
}
```

Variables

- A **variable declaration** specifies the name and type of a variable.

```
type name;
int  sum;
```

  - A variable's name enables the application to access the value of the variable in memory—the name can be any valid identifier.
  - A variable's type specifies what kind of information is stored at that location in memory.

- Several variables of the same type may be declared in one declaration.

```
type name1, name1, name2;
int  number1, number2, sum;
```

- The variables can also be initialized when declared.

```
int  sum = 0;
int  number1 = 1, number2 = 2, sum = 0;
```

Variables

- Variables of type **int** store **integer** values (whole numbers such as 7, –11, 0 and 31914).
- Types **float**, **double** and **decimal** specify real numbers (numbers with decimal points).
- Type **char** represents a single character.
- These types are called **simple types**. Simple-type names are keywords and must appear in all lowercase letters

Console Input

- The Console's **ReadLine** method waits for the user to type a string of characters at the keyboard and press the *Enter* key.
- ReadLine returns the text the user entered.
- The **Convert** class's **ToInt32** method converts this sequence of characters into data of type int.
- ToInt32 returns the int representation of the user's input.

Assignment Operator

- A value can be stored in a variable using the **assignment operator**, **=**.
- Operator = is called a **binary operator**, because it works on two pieces of information, or **operands**.
- An **assignment statement** assigns a value to a variable.
- Everything to the right of the assignment operator, =, is always evaluated before the assignment is performed.

Good Programming Practice

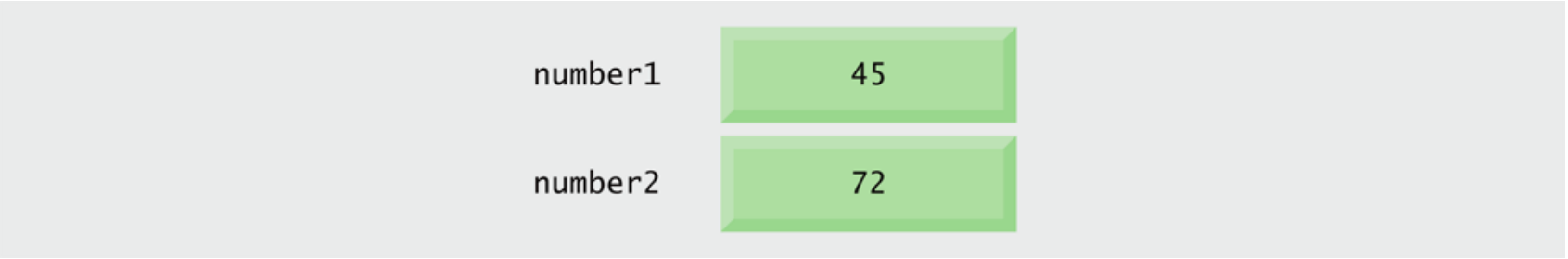
Place spaces on either side of a binary operator to make it stand out and make the code more readable.

Expression

- An **expression** is any portion of a statement that has a value associated with it.
  - The value of the expression `number1 + number2` is the sum of the numbers.
  - The value of the expression `Console.ReadLine()` is the string of characters typed by the user.
- Calculations can also be performed inside output statements.

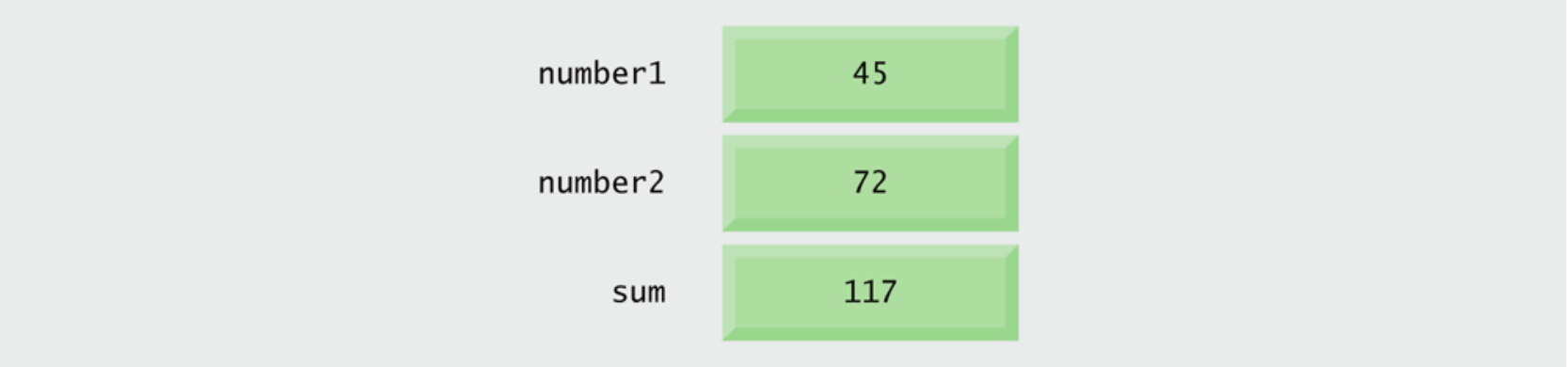
Memory Concepts

- Variable names actually correspond to **locations** in the computer's memory.
- Every variable has a **name**, a **type**, a **size** and a **value**.
- In Figure below, the computer has placed the value 45 and 72 in the memory locations corresponding to number1 and number2.



Memory Concepts (Cont.)

- After sum has been calculated, memory appears as shown in figure below:



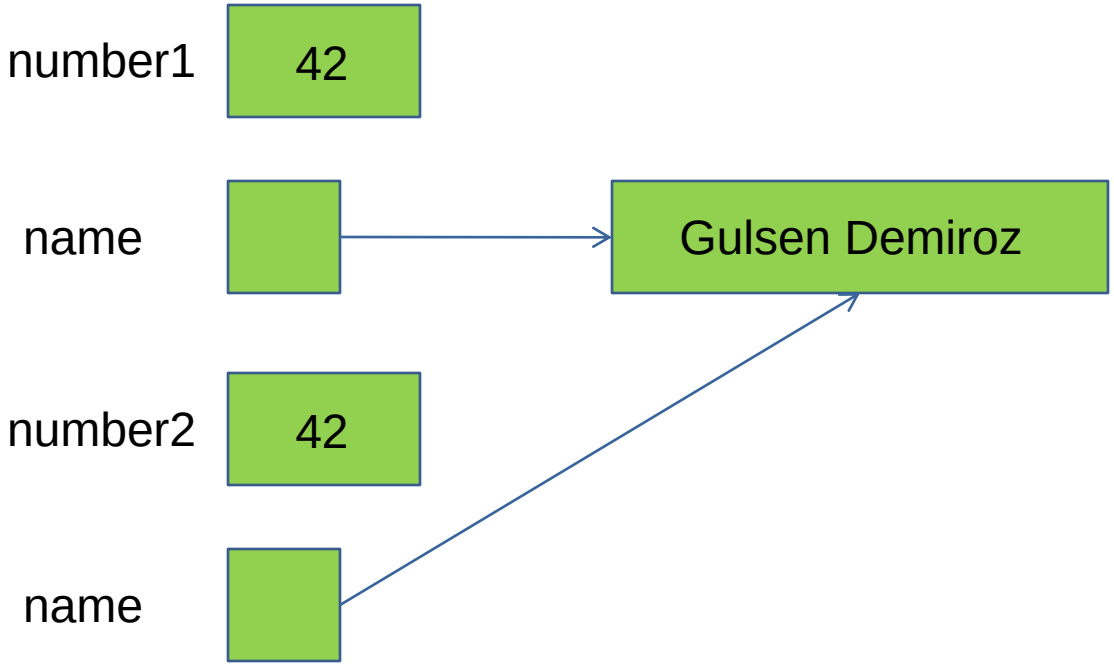
- Whenever a value is placed in a memory location, the value replaces the previous value in that location, and the previous value is lost.

Data Types

- Two general categories of types:
  - ***Value*** types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure.
    - `int` is a value type.
  - ***Reference*** types store a reference to the value's memory address, and are allocated on the heap.

Value vs Reference Types

```
int     number1 = 42;
string  name = "Gulsen Demiroz";
int     number2 = number1;
string  text = name;
```



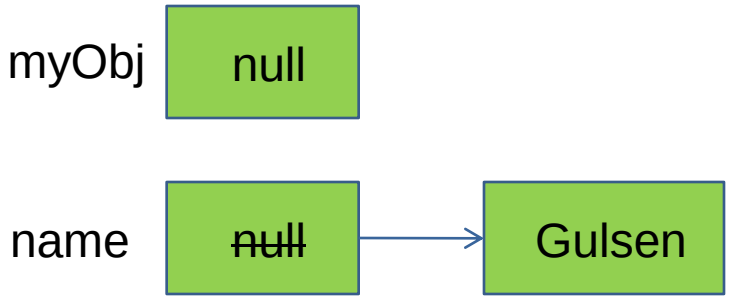
Reference Types

- Built-in reference types:
  - **System.Object**

```
object  myObj;
```

  - **System.String**

```
string  name;
name = "Gulsen";
```



- User-defined reference types:
  - Classes
  - Arrays
  - Interfaces
  - Delegates

Value Types

Type	System namespace	Definition	Uninitialized value
short	System.Int16	16 bit signed integer	0
int	System.Int32	32 bit signed integer	0
long	System.Int64	64 bit signed integer	0
ushort	System.UInt16	16 bit unsigned integer	0
uint	System.UInt32	32 bit unsigned integer	0
ulong	System.UInt64	64 bit unsigned integer	0
float	System.Single	32 bit real number	0.0
double	System.Double	64 bit real number	0.0
decimal	System.Decimal	128 bit real number	0
bool	System.Boolean	true or false	false
char	System.Char	16 bit Unicode character	'\0'
byte	System.Byte	8 bit unsigned integer	0
sbyte	System.SByte	8 bit signed integer	0
enum	user-defined	defines a type for a closed set	0 index value
struct	user-defined	defines a compound type that consists of other types	assumed value types, null reference types

.NET Common Type System

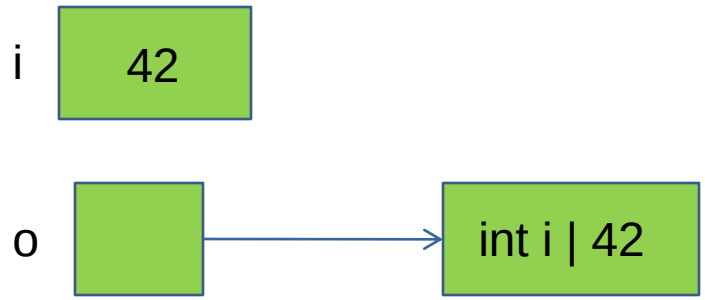
- All types derive from `System.Object` base type.
- `System.Object` allows you to:
  - Compare two instances for equality
  - Obtain a hash code for the instance
  - Query the true type of an instance
  - Perform a shallow (bitwise) copy of the instance
  - Obtain a string representation of the instance's object's current state

Boxing and Unboxing

- With boxing and unboxing, you can use any value type as an object when needed.
- Boxing: converting a value type to an object
- Unboxing: converting an object (which was boxed before) to a value type

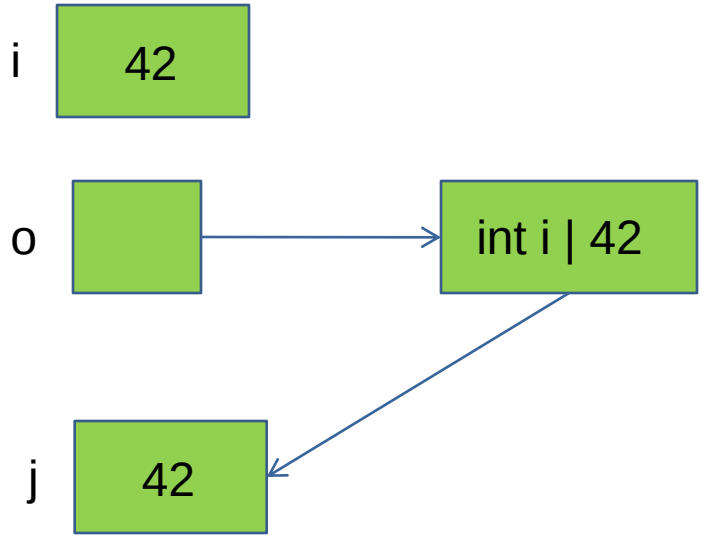
Boxing

```
int  i = 42;
object o = i;
```



Unboxing

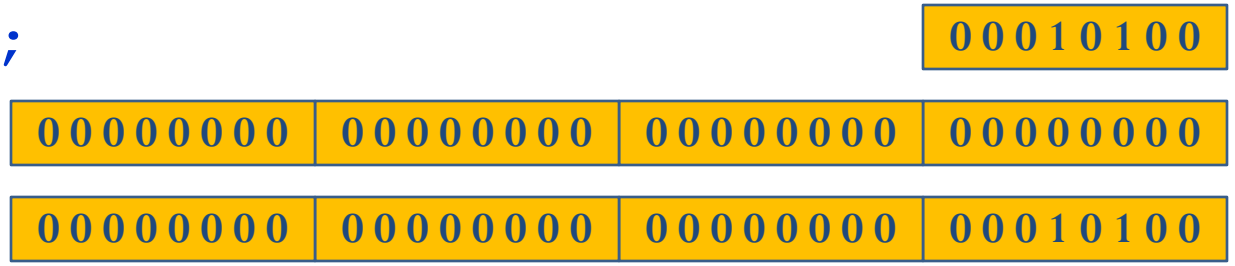
```
int  i = 42;
object o = i;
int j = (int)o; // casting is needed
```



Casting Between Types

- Implicit type-casting

```
byte a = 20;
int b;
b = a;
```



  - safe when smaller→ bigger
  - .NET has forbidden bigger → smaller

```
byte a = 5;
byte b = 3;
byte c = a + b;
```

```
Compiler error:
  Cannot implicitly convert type 'int' to 'byte'.
  An explicit conversion exists (are you missing a cast?)
```

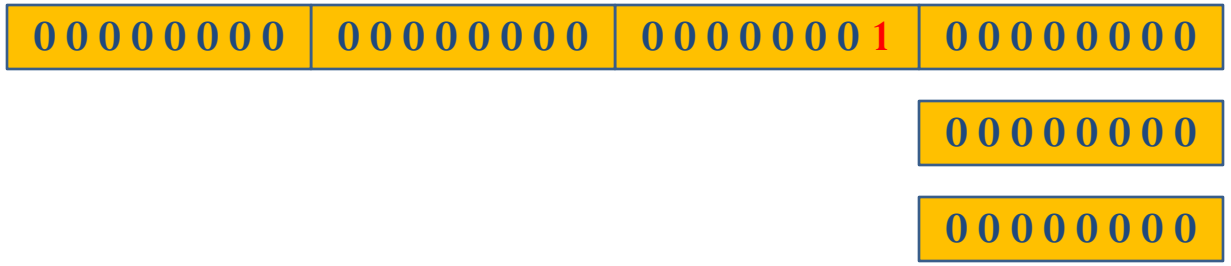
Valid Implicit Castings

Type	Types It Can Be Converted To
sbyte	short,int,float,long,double,decimal
byte	short,ushort,int,uint, long,ulong,float,double,decimal
short	int,long,float,double,decimal
ushort	int,uint,long,ulong,float,double,decimal
int	long,float,double,decimal
uint	long,ulong,float,double,decimal
long, ulong	float,double,decimal
char	float,double,decimal
float	double

Explicit Casting

- Needed when implicit casting is not allowed by the compiler
- Smaller Type → Bigger Type is the same as implicit
- Could cause data loss when Bigger Type → Smaller Type
- Need to use a type-casting operator:

```
int i = 256;
byte b;
b = (byte)i;
```



checked & unchecked

- To prevent data loss: put code that could cause data loss inside a checked block
- In case of data loss, it will throw `System.OverflowException` exception
- To ignore possible data loss: put code that could cause data loss inside a unchecked block

Convert.ToInt32

```
int number1;    // declare first number to add

Console.Write("Enter first integer:");
// read first number from user
number1 = Convert.ToInt32(Console.ReadLine());
```

- `Convert` is a class in `System` namespace
- `Convert.ToInt32` method converts a `string` into an `int`

`Convert` class

Method	
Convert.ToBoolean(string str)	converts string str to bool
Convert.ToByte(string str)	converts string str to byte
Convert.ToSByte(string str)	converts string str to signed byte
Convert.ToInt16(string str)	converts string str to short
Convert.ToUInt16(string str)	converts string str to unsigned short
Convert.ToInt32(string str)	converts string str to integer
Convert.ToUInt32(string str)	converts string str to unsigned integer
Convert.ToInt64(string str)	converts string str to long
Convert.ToSingle(string str)	converts string str to float
Convert.ToDouble(string str)	converts string str to double
Convert.ToDecimal(string str)	converts string str to decimal
Convert.ToChar(string str)	converts string str to char type

Arithmetic Operations

- Operators: + - * / %
- Operands: values that operator combines
  - variables or literals
- Combination of operators and operands is called *expression*
- Syntax and semantics for arithmetic operations:

Addition Subtraction	Multiplication	Division	Modulus
23 + 4	23 * 4	21 / 4 is 5	21 / 4 is 5
x + y	x * 3.0	21 / 4.0 is 5.25	18 % 2 is 0
d - 14.0 + 23	d * 23.1 * 4	x / 4	x % 4
5 - 3 + 2	5 - 3 * 2	x / y	x % y

Operator Precedence

- Upper operator groups have precedence

Operator	Explanation	Associativity
+ -	plus and minus signs	right-to-left
* / %	multiplication, division and modulus	left-to-right
+ -	addition, subtraction	left-to-right
= += -= *= /= %=	assignment operators	right-to-left

Assignment operator

- Assigning single expression to several variables

  $variable_1$ = $variable_2$ = $variable_3$ = ... $variable_n$ = expression;

  - all variables are assigned the same value of expression
  - example:

```
int x, y, z;
x = y = z = 6;
```

  - x, y and z contain 6

- Arithmetic assignment operators

  += -= *= /= %=

  - Combines arithmetic operation and assignment in one operator
  - *variable += expression* is the same as *variable = variable + expression*
  - Example: x += 1 is the same as x = x + 1
  - Same for -= *= /= and %=

    x -= 1 x *= 3 x /= 2 and **x %= 2**

- Example: operators.cpp