CS 112 Introduction to Programming

Lecture #2:

C# Program Structure
and Our First C# Programs

http://flint.cs.yale.edu/cs112

Outline

- ❑ Admin.
- ➢ *Programming language levels*
- ❒ Structure of a C# program
- ❒ Compiling and running our first C# programs

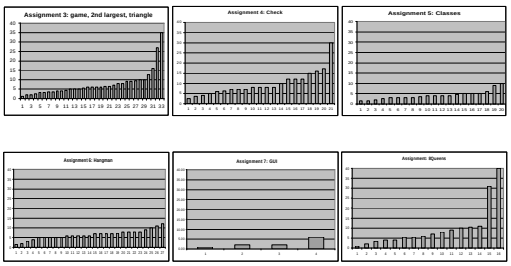
Outline

- ➢ *Admin.*
- ❑ Programming language levels
- ❒ Structure of a C# program
- ❒ Compiling and running our first C# programs

Programming Language Levels

- ❒ Each type of CPU has its own specific *machine language*
- ❒ Other levels were created to satisfy different objectives, e.g., make it easier for a human being to write programs
  - ❍ machine language
  - ❍ assembly language
  - ❍ intermediate language
  - ❍ high-level language

Admin.: Workload In Last Year



The assignment this year will be different.

Example Machine Code Fragment

```
177312 137272 001400 026400 017400 000012 000007 004420
010400 011000 000010 005023 012000 012400 000010 003426
013400 000007 000430 003000 064474 064556 037164 000001
024003 053051 000001 041404 062157 000545 007400 064514
062556 072516 061155 071145 060524 066142 000545 002000
060555 067151 000001 024026 046133 060552 060566 068057
067141 027547 072123 064562 063556 024473 000526 005000
067523 071165 062543 064506 062554 000001 046014 067151
067543 067154 065056 073141 006141 004000 004400 000007
006031 015000 015400 000001 040433 070440 067565 062564
060552 060566 064457 027557 071120 067151 051564 071164
060545 000555 003400 071160 067151 066164 000556 012400
046050 060552 060566 066057 067141 027547 072123 064562
063556 024473 000126 000041 000006 000007 000000 000000
000002 000001 000010 000011 000001 000012 000000 000035
000001 000001 000000 025005 000267 130401 000000 000400
005400 000000 003000 000400 000000 003400 004400 006000
006400 000400 005000 000000 030400 001000 000400 000000
010400 000262 011002 133003 002000 000262 011002 133005
002000 000261 000000 000001 000013 000000 000016 000003
000000 000016 000010 000020 000020 000021 000001 000016
000000 000002 000017
```

A number specifies what action the computer should take.

Example Assembly Code Fragment

```
movl    (%edx,%eax), %ecx
movl    12(%ebp), %eax
leal    0(,%eax,4), %edx
movl    $nodes, %eax
movl    (%edx,%eax), %eax
fldl    (%ecx)
fsubl   (%eax)
movl    8(%ebp), %eax
leal    0(,%eax,4), %edx
movl    $nodes, %eax
movl    (%edx,%eax), %ecx
movl    12(%ebp), %eax
leal    0(,%eax,4), %edx
movl    $nodes, %eax
```

Symbols to help programmers to remember the words.

Example C++/C#/Java Code Fragment

```
bool DetermineNeighbor(int i, int j)
{
   double distanceX = (nodes[i].x - nodes[j].x);
   double distanceY = (nodes[i].y - nodes[j].y);
   double distanceSquare = disx * disx + disy * disy;
   double distance = sqrt(distanceSquare);
   if (distance < radius)
      return true;
   else
      return false;
}
```

You do not need to understand the exact meaning of this program, just the feeling.

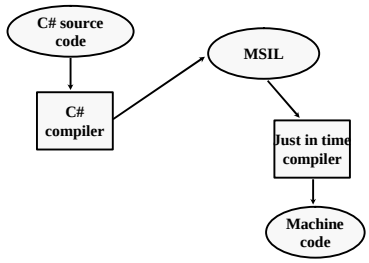
Programming Languages

- A program written in a high-level language must be translated into machine language before it can be executed on a particular type of CPU
- A *compiler* is a software tool which translates *source code* into a specific target language

C# Translation and Execution

- The C# compiler translates C# source code (.cs files) into a special representation called *Microsoft Intermediate Language* (MSIL)
- MSIL is not the machine language for any traditional CPU, but a *virtual machine*
- The Common Language Runtime (CLR) then interprets the MSIL file
  - It uses a just-in-time compiler to translate from MSIL format to machine code on the fly

C# Compilation and Execution



Outline

- Admin.
- Programming languages
  - *Structure of a C# program*
- Compiling and running our first C# programs

A Simple C# Program

```
//============================================================
//
// File: HelloWorld.cs CS112 Assignment 00
//
// Author: Zhong Shao          Email: zhong.shao@yale.edu
//
// Classes: HelloWorld
// --------------------
// This program prints a string called "Hello, World!"
//
//============================================================

using System;


class HelloWorld
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello, World!");
    }
}
```


C# Program Structure

- **Program specifications (optional)**

```
//============================================================
//
// File: HelloWorld.cs CS112 Assignment 00
//
// Author: Zhong Shao          Email: zhong.shao@yale.edu
//
// Classes: HelloWorld
// --------------------
// This program prints a string called "Hello, World!"
//
//============================================================
```

- **Library imports (optional)**

```
using System;
```

- **Class and namespace definitions**

```
class HelloWorld
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello, World!");
    }
}
```


White Space and Comments

- White Space
  - Includes spaces, newline characters, tabs, blanklines
  - C# programs should be formatted to enhance readability, using consistent indentation!
- Comments
  - Comments are ignored by the compiler: used only for human readers (i.e., inline documentation)
  - Two types of comments
    - Single-line comments use `//…`
      `// this comment runs to the end of the line`
    - Multi-lines comments use `/* … */`

```
/*  this comment runs to the terminating
 symbol, even across line breaks        */
```


Identifiers

- *Identifiers* are the words that a programmer uses in a program
- An identifier can be made up of letters, digits, and the underscore character
- They cannot begin with a digit
- C# is *case sensitive*, therefore `args` and `Args` are different identifiers
- Sometimes we choose identifiers ourselves when writing a program (such as `HelloWorld`)
- Sometimes we are using another programmer's code, so we use the identifiers that they chose (such as `WriteLine`)

```
using System;
class HelloWorld
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```


Identifiers: Keywords

- Often we use special identifiers called *keywords* that already have a predefined meaning in the language
  - Example: `class`
- A keyword cannot be used in any other way

C# keywords:				
abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	get
goto	if	implicit	in	int
interface	internal	is	lock	long
namespace	new	null	object	operator
out	override	params	private	protected
public	readonly	ref	return	sbyte
sealed	set	short	sizeof	stackalloc
static	string	struct	switch	this
throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using
value	virtual	void	volatile	while

All C# keywords are **lowercase!**

Namespaces

- Partition the name space to avoid name conflict!
- All .NET library code are organized using namespaces!
- By default, C# code is contained in the **global namespace**
- To refer to code within a namespace, must use ***qualified name*** (as in `System.Console`) or import explicitly (as in `using System;`)

```
using System;

class HelloWorld
{
  static void Main(string[] args)
  {
    Console.WriteLine("Hello World!");
  }
}
```

```
class HelloWorld
{
  static void Main(string[] args)
  {
    System.Console.WriteLine("Hello World!");
  }
}
```

More on C# Program Structure

- In C#, a program is made up of
  - Program specifications *(a.k.a. header comments, optional)*
  - Library imports *(optional)*
  - One or more *class* (and *namespace*) definitions
    - A class contains one or more *methods*
    - A method contains program *statements*
- These terms will be explored in detail throughout the course

19

C# Program Structure: Class

```
//  comments about the class
class HelloWorld
{
}
```

class header

class body

Comments can be added almost anywhere

20

C# Program Structure: Method

```
//  comments about the class
class HelloWorld
{
   //  comments about the method
   static void Main (string[] args)
   {
         Console.Write("Hello World!");
         Console.WriteLine("This is from CS112!");
   }
}
```

22

C# Method and Statements

- Methods
  - Building blocks of a program
  - The **`Main`** method
    - Each console or windows application must have exactly one (actually can have more, but it is unlikely that you will see or use)
    - All programs start by executing the **`Main`** method
  - Braces are used to start ({) and end (}) a method
- Statements
  - Every statement must end in a semicolon ;

23

C# Classes

- Each class name is an identifier
  - Can contain letters, digits, and underscores (_)
  - Cannot start with digits
  - Can start with the at symbol (@)
- Convention: Class names are capitalized, with each additional English word capitalized as well (e.g., MyFirstProgram)
- Class bodies start with a left brace ({)
- Class bodies end with a right brace (})

21

```
1  // Welcome1.cs
2  // A program in C#.
3
4  using System;
5
6  class Welcome1
7  {
8     static void Main( string[] args )
9     {
10       Console.WriteLine( "Welcome to C# Programming!" );
11    }
12 }
```

This is a statement.
Console.WriteLine outputs a string.
It instructs the program to do what you want.

Outline

class ...e1.cs ...s.

Program Output

```
Welcome to C# Programming!
```

Outline

- ❑ Admin.
- ❑ Programming languages
- ❒ Structure of a C# program
- ➢ *Compiling and running our first C# programs*

25

Backup Slides

Console Application vs. Window Application

- ❒ Console Application
  - ❍ No visual component
  - ❍ Only text input and output
  - ❍ Run under **Command Prompt** or **DOS Prompt**
- ❒ Window Application
  - ❍ Forms with many different input and output types
  - ❍ Contains ***Graphical User Interfaces*** (GUI)
  - ❍ GUIs make the input and output more user friendly!
  - ❍ Message boxes
    - Within the **`System.Windows.Forms`** namespace
    - Used to prompt or display information to the user

26

Syntax and Semantics

- ❒ The *syntax rules* of a language define how we can put symbols, reserved words, and identifiers together to make a valid program
- ❒ The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- ❒ A program that is syntactically correct is not necessarily logically (semantically) correct
- ❒ A program will always do what we tell it to do, not what we meant to tell it to do

29

```
1   // Welcome4.cs
2   // Printing multiple lines in a d
3
4   using System;
5   using System.Windows.Form
6
7   class Welcome4
8   {
9      static void Main( string[] args )
10     {
11        MessageBox.Show( "Welcome\nto\nC#\nprogramming!" );
12     }
13  }
```

The System.Windows.Forms

This will display the contents in a message box as opposed to in the console window.

Outline

Welcome4.cs

Program Output

Errors

- ❒ A program can have three types of errors
- ❒ The compiler will find problems with syntax and other basic issues (*compile-time errors*)
  - ❍ If compile-time errors exist, an executable version of the program is not created
- ❒ A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
- ❒ A program may run, but produce incorrect results (*logical errors*)

30