Basic Programming

Chapter 3, (a little) Chapter 4, and Chapter 5, Chapter 6

Topics

- Basic Programming
  - Output
  - Variables
  - Operators
  - Input
  - Selections
  - Loops

Simple C# Program

- **Figure 3.1**
- C# code is stored in files that end in .cs
- Pressing the green button builds and runs program, but the program quickly closes
- Use **CTRL-F5** to keep the program open

- Very similar to Java both in syntax and structure
- **using** is a keyword that allows us to checkout things from the library
  - Analogous to an import statement in Java
  - C# groups classes together in namespaces
  - **using** specifies namespaces so we can use the classes in the specified namespace

- Execution starts in **Main()** method
  - Methods are typically capitalized in C# by convention
- Unlike Java, class names and file names do not have to match

- **Console.WriteLine()** is a method for printing
  - Comparable to System.out.println() in Java
  - Capable of formatted printing, similar to printf()
  - **Console.Write()** also exists and behaves the same, except no newline is automatically added to the end of the output

Hello, World!

- Your turn!
- Make a program that displays the text, **"Hello, world!"**
- When creating a new project in Visual Studio, select **Console Application**
- Name it **HelloWorld**
- Visual Studio puts a lot of the stuff already for us

Variables

- Creating variables in C# is the same as in Java
- C# does have some additional basic data types that are not present in Java
- Some data types are also named slightly differently
  - **bool** in C# VS **boolean** in Java, for example

Data Types

- C# has 13 simple data types
  - 8 for integer types
  - 3 for real number types
  - 2 others
- Though very common, **string** is not a simple data type!

- Some integer types have a **u** and **s** version
  - u = **unsigned**
    - Cannot hold a negative number
  - s = **signed**
    - Can hold a negative number
- In Java, all the integer types are signed

Type	Size in Bits	Value Range
byte	8	0 thru 255
sbyte	8	-128 thru 127
short	16	-32,768 thru 32,767
ushort	16	0 thru 65,535
int	32	-2,147,483,648 thru 2,147,483,647
uint	32	0 thru 4,294,967,295
long	64	~-9 quintillion thru ~9 quintillion
ulong	64	0 thru ~18 quintillion

Type	Size in Bits	Value Range (Approximate)
float	32	-3.4e+38 to -1.4e-45 1.4e-45 to 3.4e+38
double	64	-1.7e+308 to -4.9e-324 4.9e-324 to 1.7e+308
decimal	128	-7.9e+28 to -1.0e-28 1.0e-28 to 7.9e+28

Type	Size in Bits	Value Range
bool	8	**true** or **false**
char	16	'\u0000' thru '\uFFFF' (0 thru 65535)

Decimal Data Type

- Both **double** and **decimal** can be used to represent real numbers
- **double** represents a large range less precisely
- **decimal** represents a smaller range much more precisely
  - Suitable for money

- When a constant with a decimal place is used, compiler treats it as a **double**
  - 19.95 is a **double**
- To tell the compiler to treat it as a decimal type, need to put an **m** at the end of the number
  - 19.95m is a **decimal**

Operators

- Arithmetic Operators: **+**, **-**, ***, **/**, and **%**
- Shortcut Operators: **+=**, **-=**, ***=**, **/=**, and **%=**
- Increment and Decrement Operators: **++** and **--**
  - Both prefix and postfix variants
- **Math.Pow()** and **Math.Sqrt()** for exponents and square roots

- Relational Operators: **>**, **<**, **>=**, **<=**, **==**, and **!=**
- Logical Operators: **&&**, **||**, **^**, and **!**
- Also variant of logical operators that do not perform short circuit behavior
  - **&** and **|**

Non-Short Circuit Logical Operators

- **&&** and **||** perform short circuiting behaviors
- Sometimes in Boolean expressions with **&&** or **||**, it's not necessary to evaluate both operands
  - **false && true**
  - **true || false**
- In both cases, looking at the first operands is sufficient to tell us the result of the entire expression

- When such situations are identified by C#, the second operand is not evaluated
- This type of behavior is called **short circuit**
- Efficiency is the primary motive for short circuit behavior
  - **isAccessible && performTimeConsumingOperation()**
- The method may take a long time to perform, so we don't want to do it if we don't need to

- Occasionally, we may not want to perform short circuiting behavior
- For these situations, C# has **&** and **|**
- These are exactly the same as **&&** and **||**, except they will evaluate both operands
  - No short circuit behavior with **&** and **|**

Printing Variables

- If want to display what's inside of variable, we can print it
- Simplest way is to pass the variable into **Console.WriteLine()**
- For example,
  - **int x = 5;**
  - **Console.WriteLine(x);**

Formatting Printing

- There's some other ways we can print variables and values as well
- One of these ways involves using a placeholder in the string
- When the string is printed out, the placeholder is replaced with the value listed after the string
- Placeholders look like **{0}**
- In general, ***{integer}***

- Examples:
  - **Console.WriteLine("{0}", quantity);**
  - **Console.WriteLine("{0} and {1}", quantity, rate);**
- Each ***{integer}*** in the string gets replaced with the value it corresponds to thats listed after the string
- Can have multiple placeholders in the same string
- Number of placeholders must match number of values listed afterwards

- Placeholders can also have **format specifiers**
- Letters that go after the number of the placeholder, separated by a colon
  - **{0:C}**
- Some format specifiers can have a number after them
  - **{0:F2}**

Format Specifier	Description	Examples
C	Formats as currency, rounds to appropriate decimal place, and adds currency symbol	{0:C}
N	Adds commas to large integers, default two decimal places	{0:N} {0:N0}
F	Allows us to specify number of decimal places	{0:F2} {0:F5}

Example

- **FormattingExample**

Reading Input

- C# doesn't really have a direct equivalent to Java's Scanner class
- Instead, we get input from the user with **Console.ReadLine()**
  - Prompts the user to type in something
  - Gives us back what the user typed in

- ALWAYS gives back a **string**!
  - We have to convert to appropriate data type
  - **Convert.ToInt32()** and **Convert.ToDouble()**
    - Others exist for other data types too
- **Console.Read()** also exists, but it works differently than one might expect
  - Reads a single character a time
  - Don't use unless behavior is intended

Example: Adding Two Numbers

- **Figure 3.14**

Selections

- C# has the same selection statements as Java
  - **if**, **else if**, and **else**
  - **switch**
    - **switch** statements have some slight differences

switch Statements

- No "fall through" behavior in C# switch
  - Common in C and C++
  - Leads to logic errors
- If a case has code, it must end with a break statement
- Still OK to have multiple cases for the same block of code, however

```
switch (someValue)
{
    case 0:
        // code to run
    case 1:
        // more code to run
        break;
    case 2:
    case 3:
        // code to run
        break;
    default:
        // code to run
        break;
}
```

Random Numbers

- C# uses the **Random** class for generating random numbers
- A **Random** object is a random number generator
- Use methods to get random numbers from the generator

- To set up a random number generator we write:
  - **Random generator = new Random();**
- In this example, **generator** is the name of the random number generator

- To get a random number, we use the **Next()** method on our random number generator:
  - **int randomNumber = generator.Next();**
- Calling **Next()** will give us back a number between **0** and **2,147,483,646** (inclusive)

- Often, we want a random number within a specific range, for example, between 0 through 5
- Can call **Next()**, passing in a maximum range:
  - **int randomNumber = generator.Next(6);**
  - Could give back 0, 1, 2, 3, 4, or 5
    - 1 less than the number passed in

- Can also call **Next()** passing in a minimum number too:
  - **int randomNumber = generator.Next(1, 7);**
  - Gives back either 1, 2, 3, 4, 5, or 6
  - Useful for simulating rolling die

- **Next()** gives integers, if want doubles can use **NextDouble()**
  - **double randNum = generator.NextDouble();**
- **NextDouble()** gives numbers in interval **[0, 1)**

Loops

- C# has four different types of loops
- Three of them are the same as Java
  - **while**, **do while**, and **for**
  - The fourth loop will be covered with arrays
- **break** and **continue** also work the same