Session 2

C# Language Fundamentals

Lessons

1. Overview
2. Using Predefined Types
3. Writing Expressions
4. Creating Conditional Statements
5. Creating Iteration Statements

C# Program Structure

- **`using` keyword references namespaces**
- **Execution begins at Main()**
- **Statements perform actions**
  - Program made up of statements
  - Statements separated with a semicolon
  - Braces group statements

How to Format Code in C#

- **C# is case sensitive**
- **Whitespace is ignored**
- **Single-line comments use //**
- **Multi-line comments use /*   and */**

New Programmers

- **Remember Algebra?**

$$x = 3x + 4$$

-- or --

$$F(x) = 3x + 4$$

- **x is a variable**
- **x = 3x+4 is an expression**
- **F() is a function with 1 parameter x**

Using Predefined Types

- **Those provided by C# and the .NET Framework**
- **Types are used to declare variables**
- **Variables store data based on its type**
- **Variables must be declared (and initialized) before used**
- **Predefined Types include:**
  - byte (0 to 255)
  - sbyte (signed bytes; -128 to 127)
  - short (-32,768 to 32,767)
  - ushort (0 to 65,535)
  - int (-2,147,483,648 to 2,147,483,647)
  - uint (0 to 4,294,967,295
  - long (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

C# Types

Predefined type	Definition	# Bytes
byte	Integer between 0 and 255	1
sbyte	Integer between -128 and 127	1
short	Integer between -32,768 and 32,767	2
ushort	Integer between 0 and 65535	2
int	Integer between -2147483648 and 2147483647	4
uint	Integer between 0 and 4294967295	4
long	Integer between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8
ulong	Integer between 0 and 18,446,744,073,709,551,615	8
bool	Boolean value: true or false	1
float	Single-precision floating point value (non-whole number)	4
double	Double-precision floating point value	8
decimal	Precise decimal value to 28 significant digits	12
object	Base type of all other types	N/A
char	Single Unicode character between 0 and 65,535	2
string	An unlimited sequence of Unicode characters	N/A

Declare and initialize a variable

- Variable is a storage location for a particular type
- Variable must be declared before use
- Declared only once within a scope
- Choose meaningful name
- Use camel case (fist letter lowercase, then each work starts with capital letter. i.e. randomNumber or use _lowercase)
- Do not use C# keywords
- Don’t create variable which differ only by case
- Can contain letters, numbers, or _
- Cannot begin with a number

```
int x = 12345;
Bool isOpen = false;
```

Literal values

- **Sometimes the compiler needs some help**
- **1.254 is assumed to be a double (use M if decimal)**

Category	Suffix	Description
Integer	U L UL	Unsigned Long Unsigned long
Real number	F D M L	Float Double Decimal Long

Characters & Strings

- **Use single-quotes for a character**
  `char x = ‘b’;`
- **Use double-quotes for a string**
- **A collection of characters is a string**
  `string x = “the lazy dog jumped”;`
- **Some characters cannot be represented unless escaped**

Escape sequence	Character name
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

Enumerated Types

- A group of named numeric constants
- Is a user-defined type
- Makes code easier to read and maintain
- Allows you to define a type restricted to certain values

```
public enum Planet
{
    Mercury ,
    Venus,
    Earth,
    Mars
}
```

Enumerated Types (cont.)

- **Any integral type, default is int**
- **When no value given, assumed to start at 0**

```
public enum Planet: int
{
    Mercury = 0,
    Venus = 1,
    Earth = 2,
    Mars = 3
}

Planet p = Planet.Mercury;
```

Converting Between Types

- Sometimes necessary when performing operations on values of different types
- Implicit conversion – when no loss of data compiler does it
- Explicit conversion – you do it
- Cannot convert between string (or other objects) and numeric
- Use methods of the System.Convert class
- Use ToString() method of any object to create a string

```
decimal d = 1234.56M;
int x = (int)d;
int y = Convert.ToInt32(“12345”);
string s = y.ToString();
```

Expressions

- **Purpose is to perform an action and return a value such as**
  - Assign a value to a variable
  - Perform a mathematical calculation
- **Use operators such as +,-,=**

Operators

- **Primary: Order of precedence ()**
- **Assignment : use the equals sign**

```
int a = 1234;
```

- **Mathematical: +, -, /,%, ***

```
int b = 5 * 5;
int x = 5 * (2+7);
int y = 5/3; // = 1
int z = 5%3 // = 2
```

Operators

- **Conditional: results in a bool (true/false), >,<,==,!=, &&, ||**

```
bool eq = (a==b);
bool gt = (x > z && eq);
bool eq = (a==b && b > 5);
bool gt = (x > z || eq);
```

Conditional Statements

- **if or if … else is used to manage the flow of control**

```
if (x < 5)
{
        Console.WriteLine(“X is small”);
}
else
{
        Console.WriteLine(“X is large”);
}
```

Conditional Statements

- if or if … else is used to manage the flow of control

```
if (sales > 100000)
{
        bonus += .05 * sales;
}
else
{
        bonus += .015 * sales
}
```

Switch Statement

- **used in place of many if..else statements**

```
switch (myPlanet)
{
    case Planet.Mercury:
          Console.WriteLine(“Hot!”);
          break;
    case Planet.Venus:
          Console.WriteLine(“Cloudy”);
          break;
    case Earth:
    case Mars:
          Console.WriteLine(“Nice place”);
          break;
    default:
          Console.WriteLine(“Far Out”);
          break;
}
```

Iteration Statements

- **C# provides several looping statements**
- **Used to execute a block of code repeatedly**
- **Stops when some condition is met**
- **3 types:**
  - for loop
  - while loop
  - do loop

For Loop

- **Execute a set number of times**
- **Used when you know in advance how many times to loop**

```
for (int i=0; i<5; i++)
{
    Console.WriteLine(“ i = {0}”, i);
}

------------------------------------------

 int[] nums = new int[] { 1, 2, 3, 4, 5 };

for (int i = 0; i < nums.Length; i++)
     Console.WriteLine(nums[i]);
```

While Loop

- **Pre-Test loop**
- **Executes 0 or more times**

```
string[] names = new []{“bob”,”keith”,”amy”};
int i = 0;

while(i < names.Length)
{
    Console.WriteLine(names[i]);
    i = i + 1;
}
```

Do Loop

- **Pre-Test loop**
- **Executes at least once**

```
string[] names = new []{“bob”,”keith”,”amy”};
int i = 0;
do
{
    Console.WriteLine(names[i]);

} while(i++ < names.Length)
```

Loop keywords

- **break – exits the loop**
- **continue – goes back up to the top**

```
string[] names = new []{“bob”,”keith”,”amy”,”\n”};
int i = 0;

do
{
      if (names[i] == “bob”)
            continue;
      Console.WriteLine(names[i]);

      if (names[i] == “amy”)
            break;
      i++;
} while(names[i] != “\n”)
```