Instructor: Bruce Reynolds

Introduction to Applications in C# Class 2

Concepts from Last Week

- Variable – declaration, initialization, assignment
- Type – int, string, bool
- If statement
- For statement
- Program structure – Main, F5
- Class – Console
- Method - WriteLine
- Expressions – comparison, addition
- Operators - +, = , >, >=

Homework Review

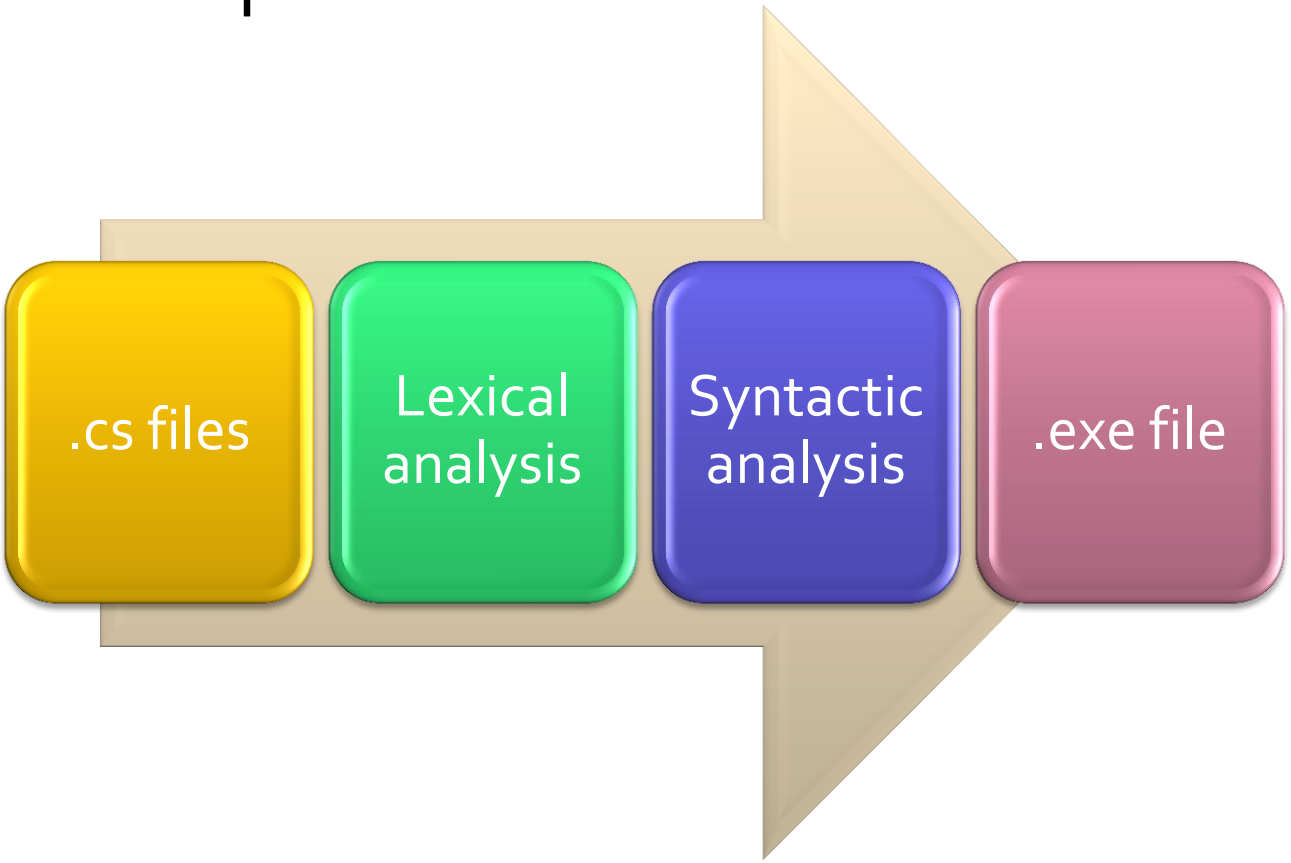
- How "grading" will work:
  - There will be eight assignments
    - Some assignments can be completed in class
  - To receive a grade of "Successful Completion" in this class:
    - Attend at least eight classes
    - Turn in all eight assignments
      - Or get your assignments checked off in class

Concepts for This Week

- Grammar – lexical and syntactic
- Operators – precedence
- Control structures – switch, while, do
- Variables – value and reference
- .NET Framework classes

The Compiler

- The compiler's work:



Grammars

- The spec has two grammars:
  - The lexical grammar that specifies how tokens, etc. are defined.
    - This takes a few section.
  - The syntactic grammar the specifies how the tokens are translated to programs.
    - This fills the rest of the specification.
- The grammar is based on recursive definitions.

Expression Grammar

- We'll follow one path.
- Consider this expression
  - `1 + 2`
  - *expression:*
    non-assignment-expression
    assignment
  - *non-assignment-expression:*
    conditional-expression
    lambda-expression
    query-expression

Expression – In Plain English

- **Expressions** are constructed from **operands** and **operators**.
  - They usually return a value.
  - Includes literals, variables, function calls.

Operators

- Operators are unary, binary, or tertiary.
- Operators have precedence and associativity.
  - Take a look at the spec.
- Operators are overloaded.
  - Their behavior depends on the types they are used with.

Demo: Guess and Check

- Determine the answer and check using the compiler (Console.WriteLine):
  - 5 + 6 * 7
  - 5 + (6 * 7)
  - (5 + 6) * 7
  - true && false
  - true && false || true
  - true && (false || true)
    - A && B is true if both operands are true
    - A || B is true if either operand is true

Discussion

- Why use parentheses?

While Statement

- The while statement conditionally executes an embedded statement zero or more times.
  - *while-statement:*
    while (*boolean-expression*) *embedded-statement*
- A while statement is executed as follows:
  - The *boolean-expression* (§7.19) is evaluated.
  - If the boolean expression yields true, control is transferred to the embedded statement. When and if control reaches the end point of the embedded statement (possibly from execution of a continue statement), control is transferred to the beginning of the while statement.
  - If the boolean expression yields false, control is transferred to the end point of the while statement.

DEMO: While Statement

- Run this code:

```
int total = 0;
Console.WriteLine("Enter one number on each line,
  blank line to end.");
string answer = Console.ReadLine();
while (answer != "")
{
    total = total + int.Parse(answer);
    answer = Console.ReadLine();
}
Console.Write("The total is ");
Console.WriteLine(total);
```

More Sophisticated

```
int total = 0;
Console.WriteLine("Enter one number on each line,
  blank line to end.");
string answer = "";
while ((answer = Console.ReadLine()) != "")
{
    total += int.Parse(answer);
}
Console.WriteLine("The total is {0}.", total);
```

Discussion

- What does this statement do, and how does operator precedence play a role?
  - `while ((answer = Console.ReadLine()) != "")`

Assignment Operators

- The assignment operators are:
  - `= *= /+ %= += -= <<= >>= &= ^= |=`
- The unary increment/decrement operators are:
  - `++ --`

Demo: ++ and -- Operators

- Predict what this code does, and then check.

```
for (int i = 10; i > 0; i--)
{
    string stars = "";
    for (int j = 1; j <= i; j++)
    {
        stars += "*";
    }
    Console.WriteLine(stars);
}
```

Formatting Output Strings

- Consider this statement:
  - `Console.WriteLine("The total is {0}.", total);`
- Help docs on Composite Formatting:
  - http://msdn.microsoft.com/en-us/library/828t9b9h.aspx
  - http://msdn.microsoft.com/en-us/library/txafckwd.aspx
- The formatting function matches the placeholders with the method arguments.

Demo: Boolean Option

- This code uses a bool sentinel value.

```
string answer = "";
Console.WriteLine("Enter one string on each line, Sunday to
   end.");
bool found = false;
while (!found)
{
    answer = Console.ReadLine();
    if (answer.StartsWith("Sunday"))
    {
        found = true;
    }
}
Console.WriteLine("Sunday finally found!");
```

The Do Statement

- The do statement conditionally executes an embedded statement one or more times.
  - *do-statement:*
    do *embedded-statement* while (*boolean-expression*) ;
- A do statement is executed as follows:
  - Control is transferred to the embedded statement.
  - When and if control reaches the end point of the embedded statement (possibly from execution of a continue statement), the *boolean-expression* (§7.19) is evaluated. If the boolean expression yields true, control is transferred to the beginning of the do statement. Otherwise, control is transferred to the end point of the do statement.

Demo: Do Statement

- Run this code:

```
Console.WriteLine("Enter one number on each line, blank line to
   end.");
int total = 0;
string answer = "0";
do
{
    total += int.Parse(answer);
    answer = Console.ReadLine();
} while (answer != "");
Console.WriteLine("The total is {0}.", total);
```

Motivation: Cascading Ifs

```
Console.Write("Enter red, blue, or green: ");
string answer = Console.ReadLine();
if (answer.ToLower() == "red")
{
    Console.ForegroundColor = ConsoleColor.Red;
}
else if (answer.ToLower() == "blue")
{
    Console.ForegroundColor = ConsoleColor.Blue;
}
else if (answer.ToLower() == "green")
{
    Console.ForegroundColor = ConsoleColor.Green;
}
else
{
    Console.ForegroundColor = ConsoleColor.White;
}
Console.WriteLine("Your color choice!");
```

Switch Statement

- Here's a basic example:

```
switch (i)
{
     case 0:
          CaseZero();
          break;
     case 1:
          CaseOne();
          break;
     default:
          CaseOthers();
          break;
}
```

Demo: Switch Statement

- Run this code:

```
Console.Write("Enter red, blue, or green: ");
string answer = Console.ReadLine();
switch (answer.ToLower())
{
    case "red" :
        Console.ForegroundColor = ConsoleColor.Red;
        break;
    case "blue" :
        Console.ForegroundColor = ConsoleColor.Blue;
        break;
    case "green" :
        Console.ForegroundColor = ConsoleColor.Green;
        break;
    default :
        Console.ForegroundColor = ConsoleColor.White;
        break;
}
Console.WriteLine("Your color choice!");
```

Falling Through and Multiple Labels

- Your code cannot "fall through" one case (*switch-section*) to the next. That is, this is not permitted:

```
case "red":
    Console.ForegroundColor = ConsoleColor.Red;
case "blue":
    Console.ForegroundColor = ConsoleColor.Blue;
```

- You can have multiple labels on each case:

```
case "red":
case "RED":
    Console.ForegroundColor = ConsoleColor.Red;
```

Discuss

- What are the advantages of using a switch statement over cascading if statements?

Framework Classes

- The Framework is made of up classes and structures organized into namespaces.
- Classes are the blueprints for an object.
- The classes have members, such as:
  - Constructors
  - Methods – define the behavior of an object
    - Methods can return data, which has type, or nothing, void.
  - Properties - describe the data and state of the object.
  - And others…

.NET – 10,000 Types

- How do you find all these methods?
  - Search (google, live, msdn). Look to see if someone else has solved the same problem.
  - Books, magazines, and blogs.
  - The .NET documentation. When I use a class for the first time, I look at the documentation and see what methods the class has.
  - Guess and check.
  - IntelliSense and F1.
  - Classes.
  - Developer Center (MSDN).

Classes and Instances

- An instance is a of a class is an object that has a location in memory and has data.
- You can also create classes in your code, which we'll do in week 4.

The new Operator

- The new operator creates an instance of a class.
- It returns a reference to the location of the instance in memory.
- It calls the constructor of the class to create and initialize the data of the instance.

Calling Instance Methods

- If a member isn't static, then it's a member method.
- To call member methods, you have to create an instance of the class and then call the method:

```
System.Random rand = new Random();
int randomNum = rand.Next();
```

Constructors

- When you use the new operator, you are calling the class constructor.
  - The constructor is code that knows how to initialize the data of the class.
  - A class may have more than one constructor:

```
Random rand = new Random();
Random randSeed = new Random(25);
```

  - You can find the constructors in the Help.

Calling Static Methods

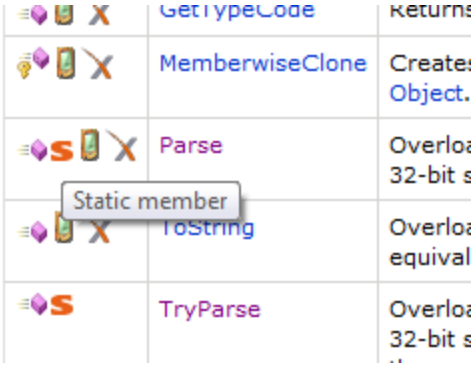
- Some methods are static. You do not have to create an instance to call them.
- This is the case when:
  - You don't have an object to call the method on, such as Int32.Parse.
  - There can only be one instance of the object, such as Console.
- To call these methods, use the class name and the method name:
  - `Console.WriteLine("hello");`
  - `double sqrt16 = Math.Sqrt(16);`

Static and Instance Methods

- The syntax block and the properties page tells you if a member is static or instance.
  - The members page has icons for static members.
  - Static members have “static” in the syntax. Instance members don’t.

Static and Instance Methods

STATIC ICON



STATIC SYNTAX

Syntax

C#

```
public static int Parse(
    string s
)
```

Reference and Value types

- We didn't have to use new to create an integer variable.
  - Random is a reference type.
  - Int is a value type.
- The syntax block in the documentation tells you if something is a value or a reference type.
  - Classes are reference types.
  - Structs are value types.

Classes and Structs

CLASS

Syntax

C#

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public class Random
```

DATETIME

Syntax

C#

```
[SerializableAttribute]
public struct DateTime : IComparable, IFormattable, IConvertible,
    ISerializable, IComparable<DateTime>, IEquatable<DateTime>
```

Accessing Properties

- Properties are class members that provide access to class data.
- Properties can be static or instance.
- Properties have type.
- They are called like methods, only without the parentheses.

Variable Summary

- Bool summary

```
bool guessed;                 // declaration
bool found = false;           //declaration and initialization
Console.WriteLine(found);     // write a bool variable
guessed = true;                  // assign a boolean literal to a bool variable
Guessed = found;                 // assign a string variable to a string variable
```

While Loop Summary

```
bool condition = true;
while (condition)
{
    // add code here
    // the loop won't stop unless you add code to set condition to false
    condition = false;
}
```

Switch Example

```
Console.Write("Enter red, blue, or green: ");
string answer = Console.ReadLine();
switch (answer.ToLower())
{
    case "red" :
        Console.ForegroundColor = ConsoleColor.Red;
        break;
    case "blue" :
        Console.ForegroundColor = ConsoleColor.Blue;
        break;
    case "green" :
        Console.ForegroundColor = ConsoleColor.Green;
        break;
    default :
        Console.ForegroundColor = ConsoleColor.White;
        break;
}
Console.WriteLine("Your color choice!");
```

New Summary

```
// call an instance method on a class
Random rand = new Random();
int number = rand.Next(1, 6);

// access an instance property on a class
System.IO.FileInfo file = new System.IO.FileInfo("somefile.txt");
string name = file.Name;

// call a static method (class or struct)
Int.Parse(“23”);

// access a static property
Console.Title = “My Program”;
```

Reading 2

- From the specification:
  - Declaration statements 8.5
  - Iteration statements 8.8
  - Switch statement 8.7.2
  - Operators 7.2
- From the book:
  - Chapter 4: Classes & Objects
  - Chapters 5 & 6: Control Statements
  - Last Week: Chapters 1, 2 & 3

Assignment 2