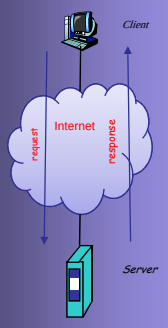


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 2.1: C# Programming Essentials (Part 1)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

- Learn about the C# operators and how they are evaluated in expressions
- Learn the jump and selection constructs
- Learn the loop constructs
- Learn how to declare, instantiate, initialize and use arrays

Operators & Expressions

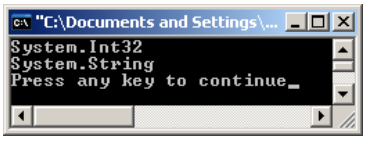
- An expression is a sequence of operators and operands that specifies a computation
  - C# has almost identical set of operators as Java
  - Operands can be variables, constants, method calls, or an expression
- The *precedence* of the operators controls the order in which the individual operators are evaluated
- Operators of the same precedence are evaluated according to their *associativity*
  - Except for assignment operator, all other binary operators are left-associative and are evaluated from left to right.
  - The assignment operator, the unary operator and the conditional operator are evaluated from right to left.

Category	Operators
Primary	[ ] dot new typeof sizeof
Unary	+ - ! ~ ++x --x (casting)x
Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational and type testing	< > <= >= is as
Equality	== !=
Logical AND	&
Logical XOR	^
Logical OR	|
Conditional AND	&&
Conditional OR	||
Conditional	?:
Assignment	= *= /= %= += -= <<= >>= &= ^= |=

Operators …

- The typeof operator is used to obtain the System.Type object for a type.

```
using System;
class Test
{
   static void Main() {
      Type t1 = typeof(int);
      Type t2 = typeof(string);
      Console.WriteLine(t1.FullName);
      Console.WriteLine(t2.FullName);
   }
}
```



Math Class

- Allows the user to perform common math calculations
- Constants
  - **`Math.PI`** = 3.1415926535…
  - **`Math.E`** = 2.7182818285…
- Using methods
  - Math.*MethodName*(*argument1*, *arument2*, …)
- Example

```
area = Math.PI *
    Math.Pow(radius, 2);
```

Method	Description	Example

`Abs( x )`	absolute value of *x*	`Abs( 23.7 )` is `23.7`
		`Abs( 0 )` is `0`
		`Abs( -23.7 )` is `23.7`

`Ceiling( x )`	rounds *x* to the smallest integer not less than *x*	`Ceiling( 9.2 )` is `10.0` `Ceiling( -9.8 )` is `-9.0`
`Cos( x )`	trigonometric cosine of *x* (*x* in radians)	`Cos( 0.0 )` is `1.0`
`Exp( x )`	exponential method *ex*	`Exp( 1.0 )` is approximately `2.7182818284590451` `Exp( 2.0 )` is approximately `7.3890560989306504`
`Floor( x )`	rounds *x* to the largest integer not greater than *x*	`Floor( 9.2 )` is `9.0` `Floor( -9.8 )` is `-10.0`
`Log( x )`	natural logarithm of *x* (base *e*)	`Log( 2.7182818284590451 )` is approximately `1.0` `Log( 7.3890560989306504 )` is approximately `2.0`
`Max( x, y )`	larger value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Max( 2.3, 12.7 )` is `12.7` `Max( -2.3, -12.7 )` is `-2.3`
`Min( x, y )`	smaller value of *x* and *y* (also has versions for `float`, `int` and `long` values)	`Min( 2.3, 12.7 )` is `2.3` `Min( -2.3, -12.7 )` is `-12.7`
`Pow( x, y )`	*x* raised to power *y* (x^y)	`Pow( 2.0, 7.0 )` is `128.0` `Pow( 9.0, .5 )` is `3.0`
`Sin( x )`	trigonometric sine of *x* (*x* in radians)	`Sin( 0.0 )` is `0.0`
`Sqrt( x )`	square root of *x*	`Sqrt( 900.0 )` is `30.0` `Sqrt( 9.0 )` is `3.0`
`Tan( x )`	trigonometric tangent of *x* (*x* in radians)	`Tan( 0.0 )` is `0.0`

Random Numbers

- Generated in the .NET Framework by making use of the System.Random class

```
Random x = new Random();
```

- Generate a random whole number >= 1 and < 2,147,483,647

```
int rnum = x.Next();
```

- Generate a random whole number >= 5 and < 10

```
int rnum = x.Next(5, 10);
```

- Generate a random whole number >= 0 and < 10

```
int rnum = x.Next(10);
```

- Generate a random number >= 0.0 and < 1.0

```
double rnum = x.NextDouble();
```

Flow Control Structures

- C# statements are evaluated in order (sequential flow) unless there is a flow control statement
- Unconditional branching statements (jump)
  - Method invocation
  - return (terminate current method and return to the invoking method)
  - continue (jump to the next loop iteration)
  - break (breaking a loop)
  - Throw (exceptions)
  - goto (jump to a labeled statement; but not recommended)
- Conditional branching statements (decision making, selection)
  - if, if-else, if-else-if statements
  - switch statement
- Loops (Repetition)
  - Iterative statements (while, do-while, for, foreach)
  - Recursive methods

Method Invocation

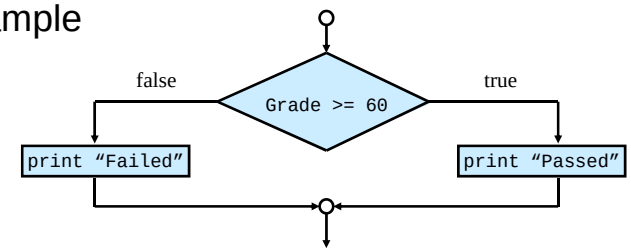
- Example

```
1.  using System;
2.  class Test
3.  {
4.     static void Main() {
5.       int x = 5, y = 8;
6.       int z = Max(x, y);
7.       Console.WriteLine("the max of {0} and {1} is {2}",
8.                 x, y, z);
9.       Console.WriteLine("the max of {0} and {1} is {2}",
10.                x, y, Math.Max(x, y));
11.    }
12.
13.    static int Max(int a, int b){
14.       return a>b? a: b;
15.    }
16. }
```

```
…
…
…
return …
```

Selection Statements

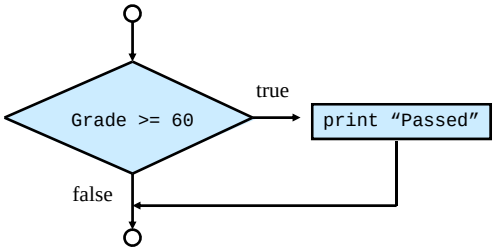
- C# offers the same basic types of selection statements as Java
- if - else statement (2-way branching)
- Example




```
1.  if (grade >= 60)
2.      Console.WriteLine("Passed");
3.  else
4.      Console.WriteLine("Failed");
```


Selection Statements …

- You can have if without else (one-way branching)
- Example




```
1.  if (grade >= 60)
2.      Console.WriteLine("Passed");
```

Selection Statements …

- switch statement
  - Has similar syntax as in Java.
  - Unlike Java, automatic fall-through between cases (if a break statement is not used) is not allowed in C#.
    - explicitly use a break or goto statement to indicate where control should jump to.
  - Example

```
int  a = 2;
switch(a) {
    case 1:
        Console.WriteLine("a>0");
        goto case 2;
    case 2:
        Console.WriteLine(" and a>1");
        break;
    default:
        Console.WriteLine("a is not set");
        break;
}
```


Selection Statements …

- switch statement …
  - An exception to this rule is when a case does not specify an action as in the following example:

```
switch(a) {
    case 1:
    case 2:
        Console.WriteLine(" and a>0");
        break;
    default:
        Console.WriteLine("a is not set");
        break;
}
```

Conditional Operator

- The conditional operator returns one of two values, depending upon the value of a boolean expression.
- Example

```
int i = (x > y) ? 1 : 0 ;
```

Equivalent to

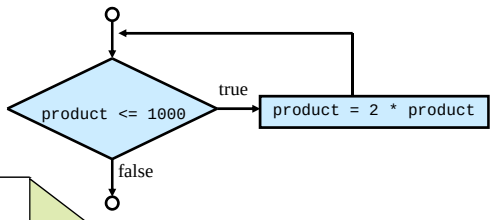
```
int i;
if(x>y)
      i=1;
else
      i=0;
```


Iteration Statements

- While loop
  - A 'while' loop executes a statement, or a block of statements, repeatedly until the condition specified by the boolean expression returns false.
- while loop syntax:

```
while (boolean_expression)
   statement
```

- Example




```
1.  double  product = 1;
2.  while (product  <= 1000)
3.     product = 2 * product;
```

Iteration Statements …

- do-while loop
  - Unlike the while loop, the condition is tested after executing the body
  - Hence, the body of a while loop may never execute (if the condition is initially false)
  - 'do-while' is used when we need the body to execute at least once even if the condition is initially false
- do-while loop syntax:

```
do
   statement
while (boolean_expression);
```

- Example

```
int a = 4;
do{
    System.Console.WriteLine(a);
    a++;
} while (a < 3);
```

Output:
4

Iteration Statements …

- for loop
  - A compact form for counter-controlled loops
- for loop syntax:

```
for (initializers; expression; iterators)
  statement
```

- Example

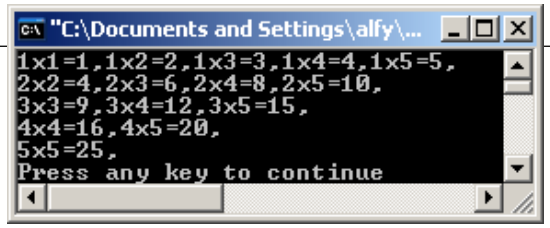
```
for (int a = 0; a<3; a++)
   System.Console.WriteLine(a);
```

Output:
0
1
2

Nested Loops

- Loops of any type can be completely nested (no overlap is allowed)
- Example

```
for(int i = 1; i<=5; i++){
  for(int j = i; j<=5; j++)
     Console.Write("{0}x{1}={2}, ", i, j, i*j);
  Console.WriteLine();
}
```



Recursive Methods

- Example
  - For a non-negative integer n, the factorial function is defined as

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

```
public static long fact (int n)
{
   if (n==0)
       return 1;
   else
       return n*fact(n-1);
}
```

Other flow control statements

- goto statement (usage is not recommended)
  - Used to make a jump to a particular labeled part of the program code
  - It is also used in the 'switch' statement to jump to another case
  - We can use a 'goto' statement to construct a loop
- continue statement
  - Used to return to the top of a loop without executing the remaining statements in the loop
- break
  - Used to break out of a loop and immediately end all further work within the loop
  - Used to get out of a case in a 'switch' statement
- return
  - Exit out of a method and return to the calling method
- throw
  - Throws an exception and exit out of a block
- foreach
  - Iterating through a collection of items (such as an array)

Arrays

- An array is an indexed collection of objects, all of the same type
- C# supports
  - one-dimensional arrays,
  - multidimensional arrays (rectangular arrays) and
  - array-of-arrays (jagged arrays)
- Dealing with arrays
  - Declaring arrays
  - Initializing arrays
  - Accessing array members
  - Arrays are objects
  - Using foreach with arrays
  - Array properties and methods

Declaring Arrays

- When declaring an array, the square brackets [ ] must come after the type, not the identifier.
  - Placing the brackets after the identifier is not legal syntax in C#
- Array types derive from System.Array.
- Examples

```
// declare a one-dimensional array
int[] grades; // not int grades[];

// declare 2-dimensional array (table)
int[,] grades;

// declare a jagged array (array-of-arrays)
int[][] grades;
```


Creating Array

- Declaring arrays does not actually create the arrays
- In C#, arrays are objects and must be instantiated
- Once an array has been created, its length can't be changed
- All elements are automatically initialized to default values

```
//declare and create 1D array having 10 elements
int[] grades = new int[10];

//declare and create 2D array (table)
int[,] grades = new int[3, 4];

//declare and create a jagged array
byte[][] scores = new byte[5][];

for (int x = 0; x< scores.Length; x++)
{
    scores[x] = new byte[4];
}
```

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

Initializing Arrays

- It is possible to initialize the contents of an array at the time it is instantiated by providing a list of values delimited by curly brackets {}.
- C# provides a longer and a shorter syntax:

```
int[] myIntArray = {2,4,6,8,10};
```

0	1	2	3	4
2	4	6	8	10

- Rectangular arrays can be initialized as follows

```
int[,] rectangularArray =
   {{0,1,2}, {3,4,5}, {6,7,8}, {9,10,11} };
```

- Jagged arrays can be initialized as follows

```
int[][] jaggedArray = {
    new int[] {0,2},
    new int[] {3,4,5},
    new int [] {6,7,8}};  // new int[] is necessary
```


Accessing Array Members

- Access the elements of an array using indexed variables
- The number of elements in an array is given by the property Length
- Array objects can be indexed from 0 to Length-1

```
1.  // double [] scores
2.  for(int i = 0; i<scores.Length; i++)
3.          Console.WriteLine(scores[i]);
```

```
1.  // double [, ] scores
2.  // scores.Length gives the total number of elements
3.  // scores.getLength(0) number of rows
4.  // scores.getLength(1) number of columns
5.  for(int i = 0; i<scores.getLength(0); i++)
6.        for(int j = 0; j<scores.getLength(1); j++)
7.              Console.WriteLine(scores[i][i]);
```

Accessing Array Members…

- foreach loop
  - Used to iterate through *all* the items in a collection (such as a one-dimensional array)
- foreach loop syntax

```
foreach (itemType variable1 in variable2)
    Statement[s];
```

- Example

```
int[] a = {1, 3, 5, 7, 9};
foreach (int i in a)
      Console.WriteLine(i);
```


Accessing Array Members …

- Rectangular arrays

```
Console.WriteLine(scores[2, 1]);
```

- Jagged arrays

```
Console.WriteLine(scores[2][1]);
```

Array Properties and Methods

- System.Array class provides methods for creating, manipulating, searching, and sorting arrays.

Method or property	Purpose
BinarySearch( )	Overloaded public static method; searches a 1D sorted array.
Clear( )	Public static method; sets a range of elements in the array either to 0 or to a null reference.
Copy( )	Overloaded public static method; copies a section of one array to another array.
CreateInstance( )	Overloaded public static method; instantiates a new instance of an array.
IndexOf( )	Overloaded public static method; returns the index (offset) of the first instance of a value in a 1D array.
LastIndexOf( )	Overloaded public static method; returns the index of the last instance of a value in a 1D array.
Reverse( )	Overloaded public static method; reverses the order of the elements in a 1D array.
Sort( )	Overloaded public static method; sorts the values in a 1D array.

Array Properties and Methods …

Length	Public property; returns the length of the array.
Rank	Public property; returns the number of dimensions of the array.
Equals()	Overloaded; returns a boolean value that specifies whether two Object instances are equal
GetLength( )	Public method; returns the length of the specified dimension in the array.
GetLowerBound( )	Public method; returns the lower boundary of the specified dimension of the array.
GetUpperBound( )	Public method; returns the upper boundary of the specified dimension of the array.
GetType( )	Returns the type of the current instance
GetValue( )	Overloaded; returns the element at the specified index in a 1D array
Initialize( )	Initializes all values in a value type array by calling the default constructor for each value. With reference arrays, all elements in the array are set to null.
SetValue( )	Overloaded public method; sets the specified array elements to a value.

Example 1

- Practice using flow control, arrays and strings

```
using System;
public class ControlStructures {
 public static void Main() {
   String input;
   do {
     Console.Write("Type int values to add or stop to exit: ");
     input = Console.ReadLine();
     if (input.ToLower() != "stop") {
           char[] delimiters = {' ', '\t', ','};
           String[] tokens = input.Split(delimiters);
           int sum = 0;
           foreach (String token in tokens)
                  sum += int.Parse(token);
           Console.WriteLine("The sum is: "+sum);
     }
   } while (input.ToLower() != "stop"); // compare strings
  }
}
```


Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- Books
  - C# 3.0 The Complete Reference, 3E, 2005
  - C# 3.0 in a Nutshell: A Desktop Quick Reference, 2007
  - Pro C# 2008 and the .NET 3.5 Platform, 4E, 2007
  - C# How to Program, By Deitel
  - Richard Blum, C# Network Programming. Sybex 2002.
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm