SWE344

Internet Protocols and Client-Server Programming

Module 2a: C# Programming Essentials

Dr. El-Sayed El-Alfy alfy@kfupm.edu.sa

Mr. Bashir M. Ghandi bmghandi@ccse.kfupm.edu.sa

Computer Science Department

King Fahd University of Petroleum and Minerals

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

Agenda

- Operators and expressions
- Math class
- Random numbers
- Flow control
- Arrays

Operators and Expressions

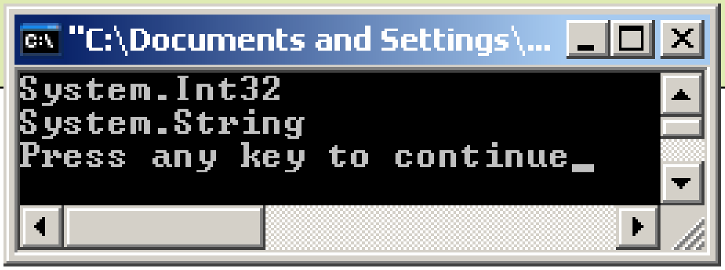
- C# has almost identical set of operators as Java
- An expression is a sequence of operators and operands that specifies a computation
- Operands can be variables, constants, method calls, or an expression
- The precedence of the operators controls the order in which the individual operators are evaluated
- 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);
    }
}
```



```
System.Int32
System.String
Press any key to continue_
```

Math Class

- The **`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 *e*^{x}	`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* (xy)	`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

- Random numbers may be 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
  - goto (not recommended)
  - continue
  - break
  - return
  - throw
- 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

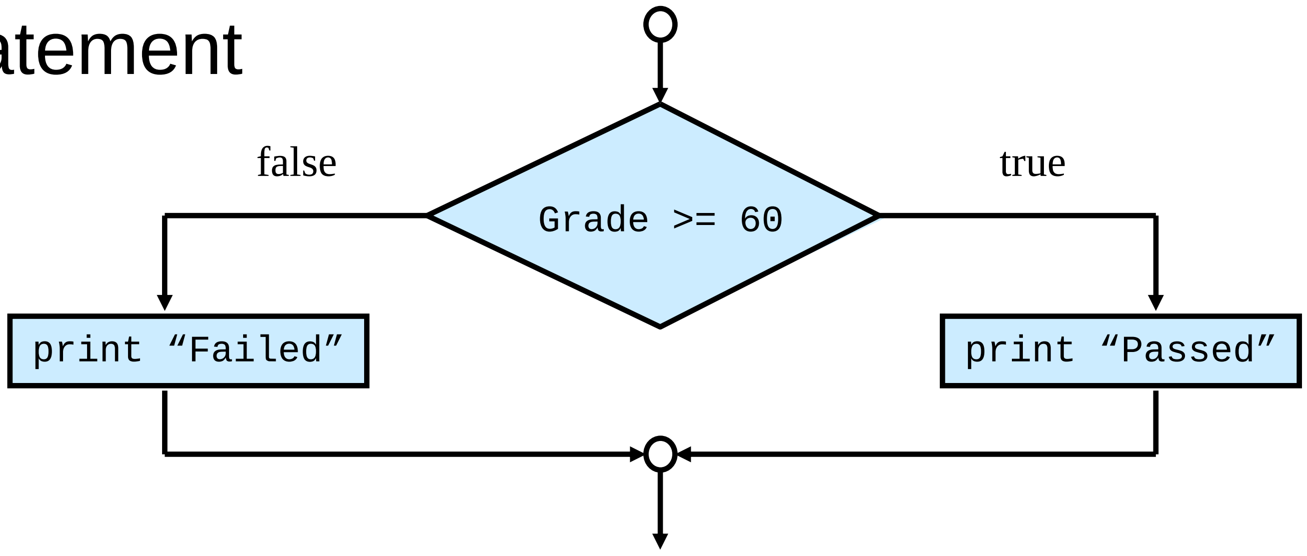
- Example

```
using System;
class Test
{
    static void Main() {
      int x = 5, y = 8;
      int z = Max(x, y);
      Console.WriteLine(“the max of {0} and {1} is {2}”,
               x, y, z);
      Console.WriteLine(“the max of {0} and {1} is {2}”,
               x, y, Math.Max(x, y));
    }

    static int Max(int a, int b){
       return a>b? a: b;
    }
}
```

Selection Statements

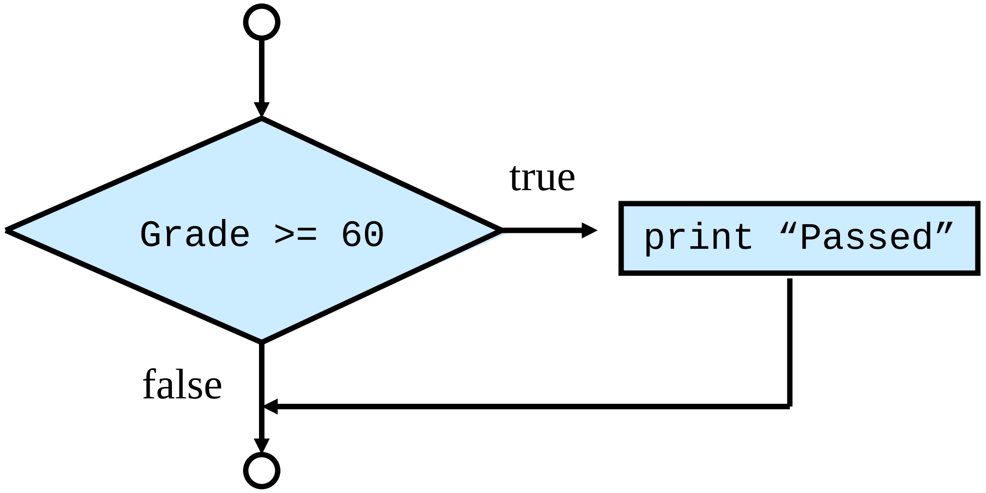
- C# offers the same basic types of selection statements as Java
- if - else statement



```
if (grade >= 60)
      Console.WriteLine(“Passed");
else
      Console.WriteLine(“Failed");
```

Selection Statements …

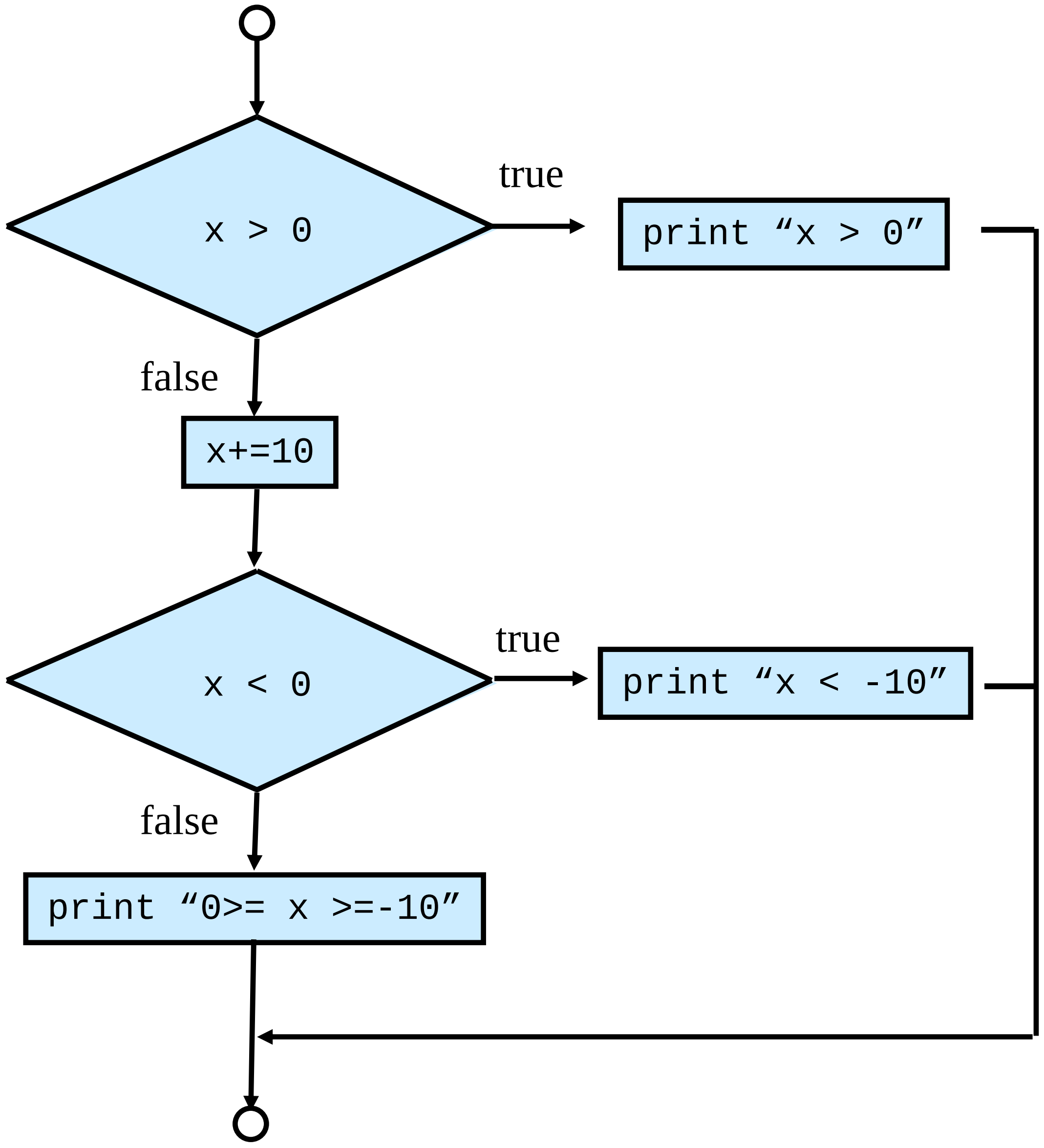
- You can have if without else (one-way branching)



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

Selection Statements …

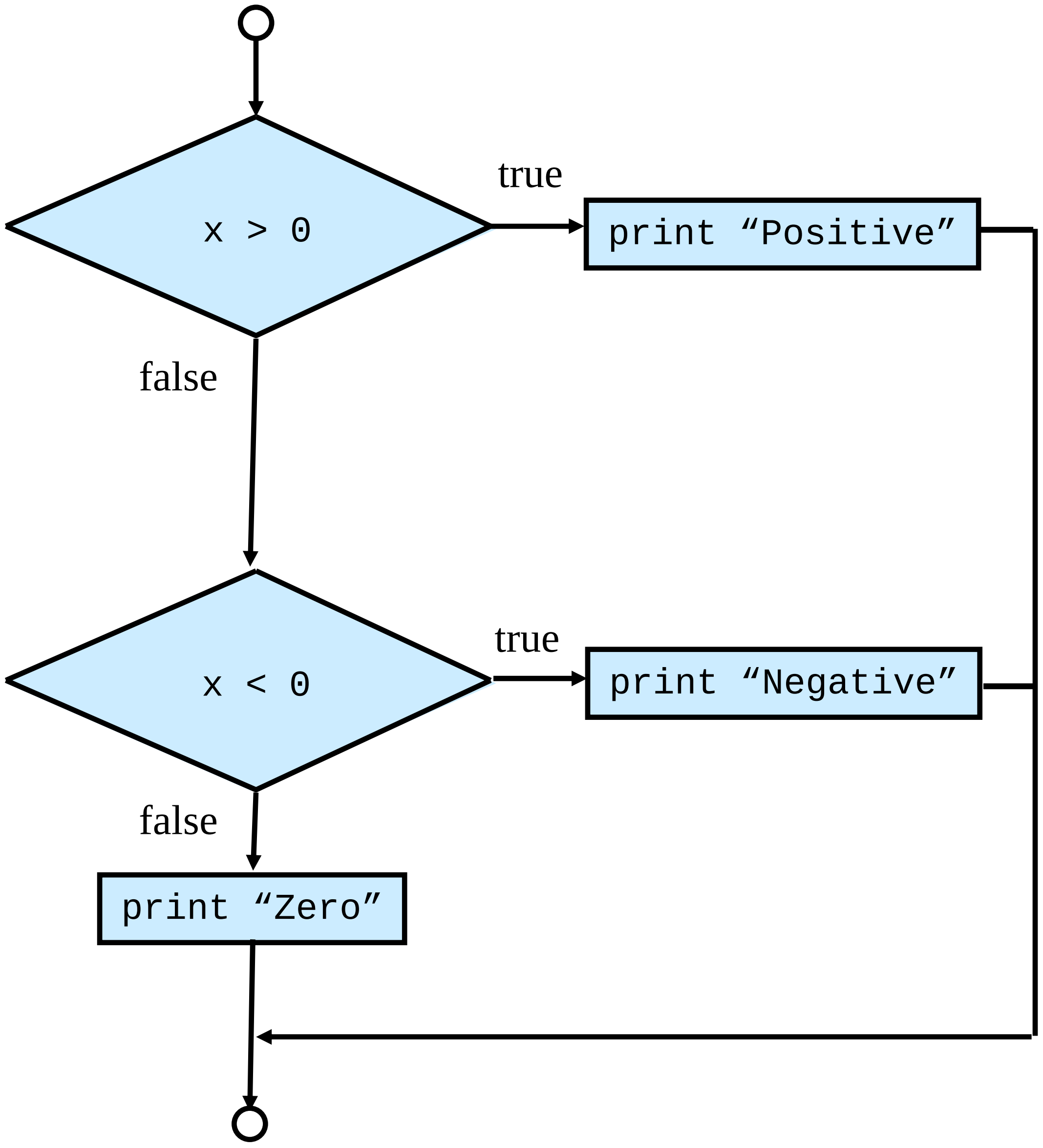
- Nested if



Activity: Write the code using nested if.

Selection Statements …

- if-else-if



Activity: Write the code

- Using nested if.

- Using if-else-if

Selection Statements ...

- switch statement
  - Has similar syntax as in Java.
  - However, C# does not allow automatic fall through between cases, which is the default in Java if a break statement is not used.
  - In C# you must explicitly use a break or goto statement to indicate where control should jump to.

```
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 ;
```

Activity

- Draw a flowchart for converting a student grade out of 100% to a letter grade (use the university standards)

- Write the equivalent code using nested-if and then using if-else-if

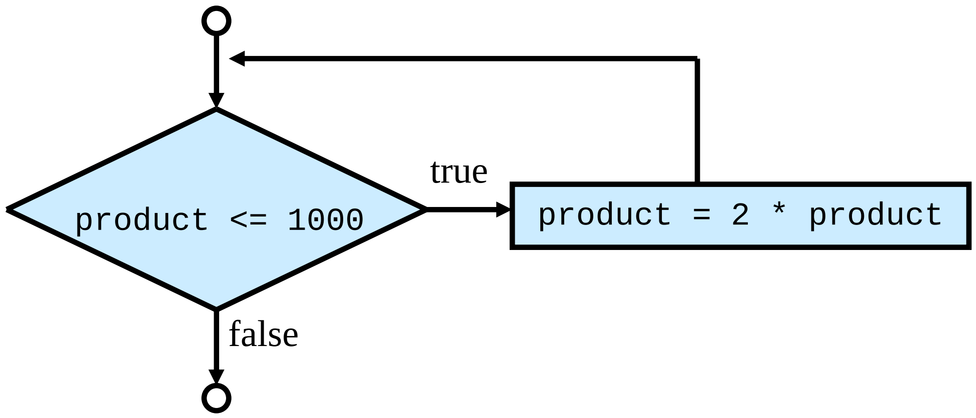
Score	LetterGrade
100 – 95	A+
95 – 90	A
90 – 85	B+
85 – 80	B
80 – 75	C+
75 – 70	C
70 – 65	D+
65 – 60	D
0 – 60	F

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



Activity: Write the code using while loop.

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);
```

```
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
```

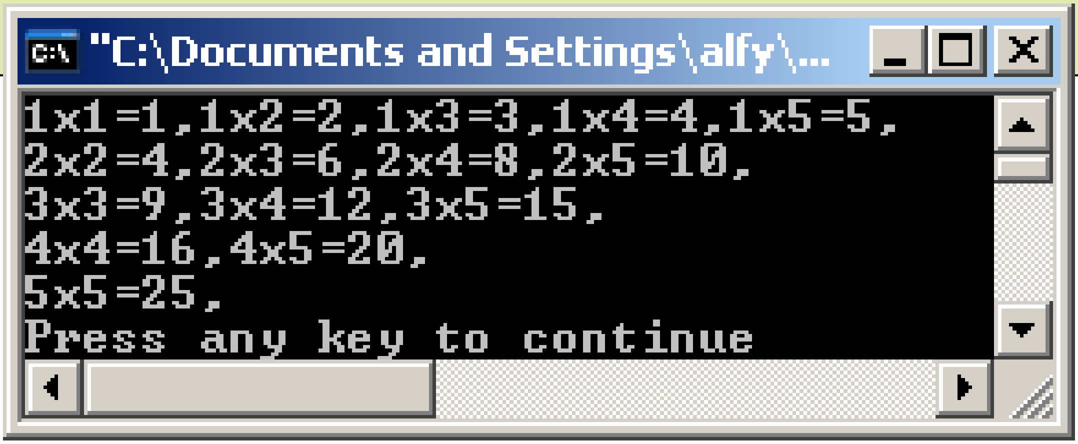
- Example

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

```
Output:
0
1
2
```

Nested Loops

```
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();
}
```



"C:\Documents and Settings\alfy\...

```
1x1=1,1x2=2,1x3=3,1x4=4,1x5=5,
2x2=4,2x3=6,2x4=8,2x5=10,
3x3=9,3x4=12,3x5=15,
4x4=16,4x5=20,
5x5=25,
Press any key to continue
```

Recursive Methods

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

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

```java
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
  - single-dimensional arrays,
  - multidimensional arrays (rectangular arrays) and
  - array-of-arrays (jagged 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.

```
// declare a single-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
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];
}
```

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 = new int[5]{2,4,6,8,10};
int[] myIntArray = new int[]{2,4,6,8,10};
int[] myIntArray = {2,4,6,8,10};
```

- Rectangular arrays can be initialized as follows

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

- Jagged arrays can be initialized as follows

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

Initializing Arrays …

- Jagged arrays can be initialized as follows

```
int[][] rectangularArray = new int [3][]{
    new int[2] {0,2},
    new int[3] {3,4,5},
    new int [3] {6,7,8}};

int[][] rectangularArray = new int [][]{
    new int[] {0,2},
    new int[] {3,4,5},
    new int [] {6,7,8}};

int[][] rectangularArray = {
    new int[] {0,2},
    new int[] {3,4,5},
    new int [] {6,7,8}};
```

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

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

```
// double [, ] scores
// scores.Length gives the total number of elements
//scores.getLength(0) number of rows (first dimension)
//scores.getLength(1) number of columns (second
dimension)
for(int i = 0; i<scores.getLength(0); i++)
      for(int j = 0; j<scores.getLength(1); j++)
             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]);
```

Lower Bounds

- The Array class can also be created by using the overloaded static method CreateInstance
  - returns an Array
  - takes three parameters: an object of type Type (indicating the type of object to hold in the array), an array of integers indicating the length of each dimension in the array, and a second array of integers indicating the lower bound for each dimension

```
int[] lengthsArray = { 3, 5 };
int[] boundsArray = { 2, 3 };
Array multiDimensionalArray = Array.CreateInstance(
                typeof( String ), lengthsArray, boundsArray );
```

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

Array Properties and Methods …

Length	Public property that returns the length of the array.
Rank	Public property that returns the number of dimensions of the array.
Equals()	Overloaded. Returns a bool that specifies whehter two Object instances are equal
GetLength( )	Public method that returns the length of the specified dimension in the array.
GetLowerBound( )	Public method that returns the lower boundary of the specified dimension of the array.
GetUpperBound( )	Public method that 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 one-dimensional 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 that 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
  }
}
```

Lab Exercises

1. Write a program that reads two integers and print out the maximum, the minimum, the sum and the average (i) using the Math class (ii) by defining your own static methods
   - Compile and run
   - Trace the program execution step by step
2. Design a menu-driven console application to help an instructor teaching a specific course to manage student grades. The instructor should be able to enter information about students in his class once. This information includes number of students, their names and their grades in a number of quizzes. Then, he should be able to display grade roster showing all grades, the total and the average for each student, and the average for each quiz and total average for the whole class. Also he should be to delete a student, update student information, add a new student, display students sorted by name or by total grade, etc. Choose a design approach and justify your choice.
   - Use parallel arrays
   - Use OOP and array of objects
   - Pay attention to the user interface to be more flexible, appealing, etc