Advanced Computer Programming TICT2134

C# Syntax

- Following code print "Hello World" to the screen:

```
using System;

namespace HelloWorld
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

C# Syntax (Cont.)

- using System means that we can use classes from the System namespace.
- namespace is used to organize your code, and it is a container for classes and other namespaces.
- Console is a class of the System namespace, which has a WriteLine() method that is used to output/print text.
- class is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class Program.
- If you omit the using System line, you would have to write System.Console.WriteLine() to print/output text.
- C# statement ends with a semicolon ";".
- C# is case-sensitive; "MyClass" and "myclass" have different meaning.

C# Variables

- Syntax

```
type variableName = value;
```

- Example

```
string name = "John";
Console.WriteLine(name);
```

C# Constants

- If you don't want others (or yourself) to overwrite existing values, you can add the const keyword in front of the variable type.

```
const int myNum = 15;
myNum = 20; // error
```

C# Display Variables

- The WriteLine() method is often used to display variable values to the console window.
- To combine both text and a variable, use the + character:

```
string name = "John";
Console.WriteLine("Hello " + name);
```

C# Multiple Variables

- To declare more than one variable of the **same type**, use a comma-separated list:

```
int x = 5, y = 6, z = 50;
Console.WriteLine(x + y + z);
```

- You can also assign the same value to multiple variables in one line:

```
int x, y, z;
x = y = z = 50;
Console.WriteLine(x + y + z);
```

C# Identifiers

- All C# **variables** must be **identified** with **unique names**.
- These unique names are called **identifiers**.
- It is recommended to use descriptive names in order to create understandable and maintainable code:

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

C# Data Types

- Data type

```
int myNum = 5;                  // Integer (whole number)
double myDoubleNum = 5.99D;     // Floating point number
char myLetter = 'D';            // Character
bool myBool = true;             // Boolean
string myText = "Hello";        // String
```

Data types and size

Data Type	Size	Description
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
bool	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter, surrounded by single quotes
string	2 bytes per character	Stores a sequence of characters, surrounded by double quotes

C# Type Casting

- Type casting is when you assign a value of one data type to another type.
- In C#, there are two types of casting:
  - Implicit Casting (automatically) - converting a smaller type to a larger type size
  - char -> int -> long -> float -> double
  - Explicit Casting (manually) - converting a larger type to a smaller size type
  - double -> float -> long -> int -> char

C# User Input

- Console.ReadLine() to get user input.

```
// Type your username and press enter
Console.WriteLine("Enter username:");

// Create a string variable and get user input from the keyboard and store it in the variable
string userName = Console.ReadLine();

// Print the value of the variable (userName), which will display the input value
Console.WriteLine("Username is: " + userName);
```

C# User Input and Numbers

- The Console.ReadLine() method returns a string. Therefore, you cannot get information from another data type, such as int. The following program will cause an error:

```
Console.WriteLine("Enter your age:");
int age = Console.ReadLine();
Console.WriteLine("Your age is: " + age);
```

- You can convert any type explicitly, by using one of the Convert.To methods:

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

C# Operators

- Operators are used to perform operations on variables and values.
- Types of operators.
  - Arithmetic
  - Assignment
  - Comparison
  - Logical

C# Arithmetic Operators

- Arithmetic operators are used to perform common mathematical operations:

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

C# Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

C# Comparison Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

C# Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns True if both statements are true	x < 5 && x < 10
||	Logical or	Returns True if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns False if the result is true	!(x < 5 && x < 10)