FUNDAMENTALS OF PROGRAMMING WITH C#

CONSOLE INPUT AND OUTPUT

Liu Fan
isslf@nus.edu.sg

Total:26

Objectives

- Write programs that print information to the screen with appropriate format
- Write a program that takes input from users and convert it to the appropriate types

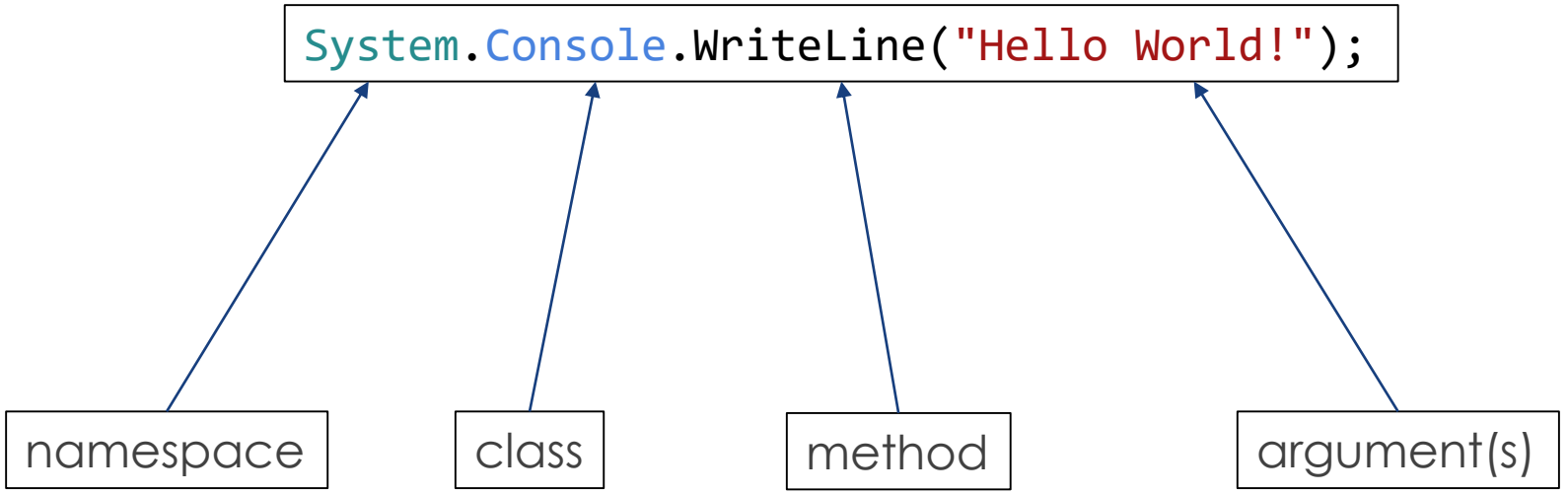
Agenda

- Printing output to screen
  - WriteLine and Write
  - Handling literal, variables and expressions
  - Escape sequences
  - Placeholders
  - String interpolation
- Read input from users
  - ReadLine
  - Converting string to other types

Printing text to screen

- To print text to screen, we can use
  - System.Console.WriteLine("Hello") or
  - System.Console.Write("Hello")

- To recap the syntax:



Write vs. WriteLine

Program

```
System.Console.WriteLine("Welcome ");
System.Console.WriteLine("to ISS");
```

Output

```
Welcome
to ISS
```

Program

```
System.Console.Write("Welcome ");
System.Console.WriteLine("to ISS");
```

Output

```
Welcome to ISS
```

The current position where the next character will be printed is commonly called as **cursor**. WriteLine will cause the cursor to move to the next line.

Imagine a typewriter, where you have to push a carriage return lever to turn the paper up and moves the carriage back to the start of the next line

Concatenation (Join two strings)

Program

```
System.Console.WriteLine("Welcome " + "to ISS");
```

Output

```
Welcome to ISS
```

- String is just another word for text – the term comes from the phrase
  "a string of characters"
- The plus symbol indicates concatenation if one of the argument is a string

Numbers (other data type)

Program

```
System.Console.WriteLine(5.2);
System.Console.WriteLine("Five point two");
```

Output

```
5.2
Five point two
```

- WriteLine and Write can accept numbers and other types of values as arguments
- In C#, all objects can be converted into string.
- When an object is passed as an argument to Write or WriteLine, the string conversion of the object will be printed out to the screen.

Expression

Program

```
System.Console.WriteLine(5.2 + 1.1);
System.Console.WriteLine("Six point three");
```

Output

```
6.3
Six point three
```

- Expression is something that can be evaluated to a single value
  - 5.2 + 1.1 can be evaluated into 6.3 (a single value)
- Expression can be made of a sequence of operand and operator (like in our case)
  - 5.2 and 1.1 are the operands
  - + symbol is the operator

Variables

Program

```
double a = 5.2;
System.Console.WriteLine(a + 1.1);
System.Console.WriteLine("Six point three");
```

Output

```
6.3
Six point three
```

- We can use variables instead of a literal value.
- Example of literal value
  - 1.1
  - "Six point three"
- Variable is a named "container" that contain a value
  - Example:
    - a is a variable that contain value of the type "double" – number that contains decimal point

Variables and String

Program

```
double a = 5.2;
System.Console.WriteLine(a + 1.1);
System.Console.WriteLine("a" + "1.1");
```

Output

```
6.3
a1.1
```

- In the first WriteLine, the plus symbol adds the value of a with 1.1
- In the second WriteLine, the plus symbol joins two strings: "a" and "1.1"
- Both lines look somewhat similar
  - One of the common source of beginner's error

Escape Sequence

- Escape sequence are special characters in a string for formatting purposes
  - Escape sequence starts with backslash character
- Some common escape sequences

Escape Sequence	Meaning
\n	Newline. Position the screen cursor to the beginning of the next line. Useful when you want to write multiple line in a single Write statement
\t	Horizontal Tab. Move the screen cursor to the next tab stop
\\	Backslash. Used to print backslash character
\"	Double quote. Used to print a double quote character

https://docs.microsoft.com/en-us/cpp/c-language/escape-sequences?view=msvc-160

Escape Sequence

Program

```
System.Console.WriteLine("Welcome \nto ISS");
System.Console.WriteLine("Welcome\tto\tISS");
System.Console.WriteLine("He said \"Thanks\" and bowed");
```

Output

```
Welcome
to ISS
Welcome     to     ISS
He said "Thanks" and bowed
```

- In the first WriteLine, the \n starts a new line
- In the second WriteLine, the \t represent a horizontal tab movement
- In the third WriteLine, the \" represents the double quotation mark

Complex concatenation

Program

```
double a = 5.2;
double b = a * 2;
System.Console.WriteLine("a = " + a);
System.Console.WriteLine("b = " + b);
System.Console.WriteLine("a = " + a + ", b = " + b);
```

Output

```
a = 5.2
b = 10.4
a = 5.2, b = 10.4
```

- It is quite common that we need to write complex concatenation when writing console applications
- The statements can become quite complex and error prone – see the third WriteLine
  - There are better ways to write these statements

Placeholder

Program

```
double a = 5.2;
double b = a * 2;
System.Console.WriteLine("a = {0}", a);
System.Console.WriteLine("b = {0}", b);
System.Console.WriteLine("a = {0}, b = {1}", a, b);
```

Output

```
a = 5.2
b = 10.4
a = 5.2, b = 10.4
```

- The third WriteLine statement look simpler
  - First argument takes care of the placement of values
  - Second and third arguments contains the value to be replaced into the placeholder
  - 2nd argument goes into {0}, 3rd argument goes into {1} and so on

Placeholder

Program

```
double a = 5.2;
double b = a * 2;
System.Console.WriteLine("a = {1}, b = {0}", b, a);
```

Output

```
a = 5.2, b = 10.4
```

- There is no requirement that {0} must appear before {1} in the string.
  - The above example works too but more unnatural to read – more confusing – reduce code readability
- Advice: go from left to right and from 0, 1, 2, etc

Placeholder

Program

```
double a = 5.2; double b = a * 2;
System.Console.WriteLine("a = {1}", b, a);
```

Output

```
a = 5.2
```

Program

```
double a = 5.2; double b = a * 2;
System.Console.WriteLine("a = {1}", a);
```

Output

```
Error! Why?
```

If the highest placeholder index is 1, then there must be at least 2 additional arguments after the string

- What if you give more arguments?

Program

```
double a = 5.2; double b = a * 2;
System.Console.WriteLine("a = {1}", b, a, "Hello");
```

Output

```
a = 5.2
```

String interpolation (C# 6.0 feature)

Program

```
double a = 5.2;
double b = a * 2;
System.Console.WriteLine($"a = {a}");
System.Console.WriteLine($"b = {b}");
System.Console.WriteLine($"a = {a}, b = {b}");
System.Console.WriteLine("a = {a}, b = {b}");
```

Output

```
a = 5.2
b = 10.4
a = 5.2, b = 10.4
a = {a}, b = {b}
```

- The $ special character identifies a string literal as an interpolated string. An interpolated string is a string literal that might contain interpolation expressions.
- To identify a string literal as an interpolated string, prepend it with the $ symbol. There is no any white space between the $ and the " that starts a string literal
- Arguably better than placeholder
- It's not recommended to put a long expression inside the curly braces
  - Reduce code readability

String interpolation (C# 6.0 feature)

Program

```
double a = 5.2; double b = a * 2;
string myText = $"a = {a}, b = {b}";
System.Console.WriteLine(myText);
```

Output

```
a = 5.2, b = 10.4
```

- One benefit of string interpolation that it works when we create a string value
  - Works for all cases involving string
- Placeholder is a specific feature of Write and WriteLine methods

Number Formatting

- Placeholder and string interpolation can be used with formatting information
- Number formatting is quite common especially for scientific and financial applications

Format Specifier	Meaning
#	Digit placeholder – replaced with digit if a digit is present e.g. 6543.21 (#.#) = 6543.2 6543.21 (#.###) = 6543.21 64.65 (#.#) = 64.7 Will not be replaced with a non-significant 0 e.g. 007 (###) = 7
0	Zero placeholder – replaced with digit if a digit is present otherwise zero e.g. 6543.21 (0.0) = 6543.2 6543.21 (0.000) = 6543.210 64.65 (0.0) = 64.7 007 (000) = 007

Number Formatting

Format Specifier	Meaning
.	Decimal point – determines the location of the decimal separator in the result.
,	Group separator – to group the numbers for ease of reading e.g. 6543.21 (#,#.#) = 6,543.2 6543.21 (#,#.000) = 6,543.210

- Other formatting rules can be found in .NET language documentation. Please review both *standard* and *custom number formatting* which can be quite useful for your program.

https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-numeric-format-strings

Number Formatting

To specify a number formatting, write the format in the curly braces together with the number for placeholder or the expression in string interpolation separated with colon :

Example	Result
`Console.WriteLine($"{a}");`	6543.21
`Console.WriteLine($"{a:#.#}");`	6543.2
`Console.WriteLine($"{a:#.###}");`	6543.21
`Console.WriteLine($"{a:0.0}");`	6543.2
`Console.WriteLine($"{a:0.000}");`	6543.210
`Console.WriteLine($"{a:0,0.00}");`	6,543.21
`Console.WriteLine("{0}", b);`	7
`Console.WriteLine("{0:###}", b);`	7
`Console.WriteLine("{0:000}", b);`	007

Reading Input from Console

- To read input from console, we can use `System.Console.ReadLine` method.

Program

```
Console.Write ("Please enter your name: ");
string name = Console.ReadLine();
Console.WriteLine("Your name is {0}", name);
```

Output

```
Please enter your name: Jane
Your name is Jane
```

Reading a number from user

- `System.Console.ReadLine()` always returns a string.
- Sometimes we need to convert this string into number so that we can use it for calculation.
- To do that we can use the following methods
  - `Convert.ToInt32()`
  - `int.Parse()`
- The main difference between `int.Parse()` and `Convert ToInt32()` is that passing a `null` value to `int.Parse()` will throw an `ArgumentNullException` while passing a `null` value to `Convert.ToInt32()` will give zero.

Reading a number from user

Program

```
Console.Write("Please enter a number: ");
int number = Convert.ToInt32(Console.ReadLine());
Console.WriteLine($"{number}+1 = {number+1}");
```

Output

```
Please enter a number: 100
100+1 = 101
```

Program

```
Console.Write("Please enter a number: ");
int number = int.Parse(Console.ReadLine());
Console.WriteLine($"{number}+1 = {number + 1}");
```

Output

```
Please enter a number: 100
100+1 = 101
```

Conversion and Parsing for other data types

Data Type	Example
double	`double value = double.Parse(input);` `double value = Convert.ToDouble(input);`
int	`int value = int.Parse(input);` `int value = Convert.ToInt32(input);`
bool	`bool value = bool.Parse(input);` `bool value = Convert.ToBoolean(input);`

Summary

- Difference between Write() and WriteLine()
- Concatenation
- Placeholder
- String Interpolation
- Number Formatting
- Reading Input using ReadLine()
- Converting string into numbers