THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

DEPARTMENT OF
COMPUTER SCIENCE
Te Tari Rorohiko

2024

Contributors

J. Turner

R. Mercado

V. Moxham-Bettridge

A. Hinze

C. Pilbrow

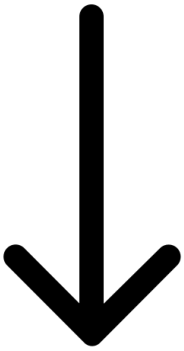
N. Kanji

T. Elphick

S. Cunningham

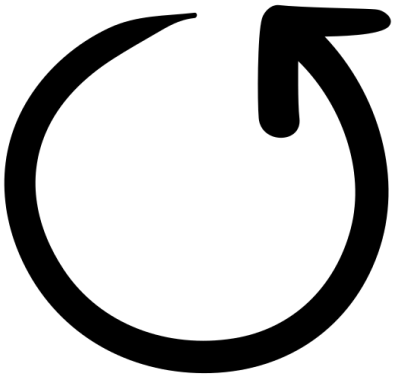
INTRODUCTION TO PROGRAMMING

Today we will have a very brief and quick introduction to common programming concepts to ensure we are all on the same page. In all software there are three main programming structures that are used, these are: sequences, selection and loops.



Sequence

Selection



Loops

Sequences

Programs are executed in sequential order, for example:

```
01    using System;
02
03    namespace Hellow
04    {
05          class MainClass
06          {
07                public static void Main(string[] args)
08                {
09                      Console.WriteLine("Hello World!");
10                }
11          }
12    }
```

The 1st line of the program tells C# to use the System library. The 3rd line defines the namespace for the program (`Hellow`). On line 5 a class called `MainClass` is created to store

the code for this program. Line 7 creates what is called a function of the class, it contains a set of code which is executed each time its name is called (in this case `Main`).

On line 7 we also introduce some keywords that you may not have seen before. The "`public`" keyword means that this function is accessible from other classes. The "`static`" keyword means that we can access the function without instantiating an object (a class is a blueprint for an object, more on this later). The "`void`" keyword means that this function returns nothing, sometimes we might want to return a number (e.g. `int`) or a word (e.g. `string`) from a function to use elsewhere in our code.

Inside the brackets on line 7 we specify the arguments of the function, these are pieces of information that we want the function to use to do a calculation. An array is a collection of items of a specific size.

On line 9 we call the `WriteLine` function for the `Console` object. The Console object is how we access the terminal window that the program displays when you run it (see figure 1).

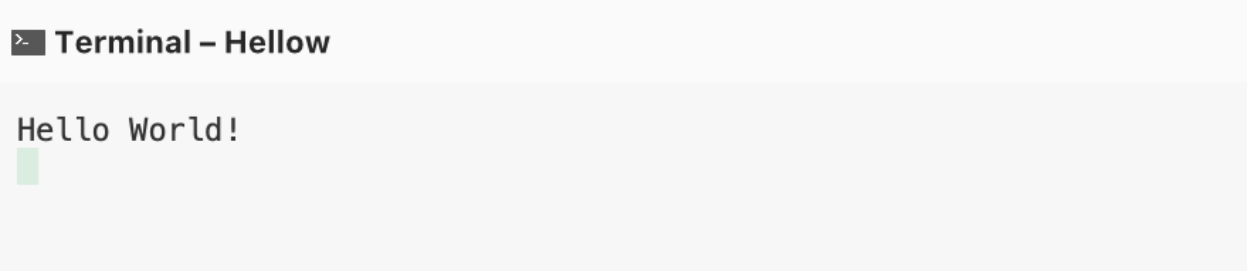


Figure 1: The Terminal Window

The `WriteLine` function is predefined in the class for a `Console` object, it is rather self-explanatory in that it writes a line to the `Console` by taking one argument, in this case the string "`Hello World!`" and displaying it to standard output (the terminal). Recall that a class is the blueprint for an object. A class may contain variables and functions and we use them to encapsulate data (this is one of the fundamental building blocks of object-oriented programming).

Exercises:

1. Create a new console project in Visual Studio called Hellow (ask for help if you are unsure).
2. Type in the above program.
3. Did it run too quickly? Try adding "`Thread.Sleep(1000);`" after line 9 so you can see the display.
4. Can you change the program so that it says "Hello <*your name*>!" where you replace *"<your name>"* with your own name?

Selection

In a program we can make a choice based on a condition. For example, if we have exactly three items in an array then we might want to print the length, if not we might want to tell the user that the array is too short. Let's consider what this program would look like in C#:

```
01    using System;
02
03    namespace Hellow
04    {
05          class MainClass
06          {
07                public static void Main(string[] args)
08                {
09                      string[] arr = new string[3];
10                      if(arr.Length==3) {
11                            Console.WriteLine("Array size: " +
                              arr.Length);
12                      } else {
13                            Console.WriteLine("Array is too short");
14                      }
15                }
16          }
17    }
```

Lines 1-8 of the program are the same as our last example, so let's focus on line 9. Here we are creating a variable called `arr`. Every time you create a variable in C# it must have a data type associated with it, we have already seen a few data types such as a `string` (for a sequence of characters, or a word) or an `int` (for a negative or positive whole number). In this case we are creating a string array, arrays are denoted by specifying the data type of the array followed by square brackets and then the name. The "=" character here denotes assignment in that what comes next is what is "stored" in the variable. You can think of a variable as a container which holds data of a specific type. In this case we want to create a new string array so we use the new keyword followed again by the data type string with the size of the array in square brackets.

If we consider what the array looks like we have the following empty structure:

0	1	2

Figure 2: Empty Array of Size 3

Note that the positions in the array start at 0, we call this the index into the array. Indexes are important because they allow us to retrieve or store data from or in the array. For example:

```
arr[0] = "Hello!";
```

This line of code allows us to set position 0 in the array to be the string "Hello!". Internally the array now looks like this:

0	1	2
Hello!		

Figure 3: Array of Size 3 with a Single Item

If we then wanted to retrieve that same string we would also use the notation "`arr[0]`" as this tells C# to access the string array `arr` and get the information at position 0. For example, if we wanted to create a new variable to store what was at position 0 we could use the following:

```
string first_item = arr[0];
```

This would allow us to store just the first position of the array into a new variable called `first_item`.

Now we consider the if statement, or rather conditional statement on line 10. This asks the question "does the array `arr` have a length/size of 3?" if it is true line 11 will be executed, if it is false then line 13 will be executed. This is why we call this selection, as the code which is executed is dependent on if the conditional statement is true or false.

Note that here we use "==" for comparison of two statements. This is simply because we use a single "=" for assignment and C# will assume assignment when it sees a single equals sign. This is a common cause for typos in your programs so watch out for this easy to make mistake.

Conditional statements can take various forms. For example:

- A single condition: `if(arr.Length==3) { … }`
- A single condition with an else: `if(arr.Length==3) { … } else { … }`
- Multiple conditions: `if(arr.Length==3) { … } else if(arr.Length==4) { … } else if ....`
- Multiple conditions with a final else: `if(arr.Length==3) { … } else if(arr.Length==4) { … } else { … }`

However, you cannot have an "`else`" or "`else if`" without an initial if statement as above, as the else is dependent on the first condition.

There are several operators that we can use inside of our conditional statements. For now we will only focus on the "simple" comparators:

- Comparison: ==
- Greater than: >
- Less than: <
- Greater than or equal to: >=
- Less than or equal to: <=
- Not equal to: `!=`

You can use any of the above comparators for comparing data types like strings, ints and arrays (and obviously other data types we haven't seen yet!).

Lastly, let's discuss line 11, here we can see an example of string concatenation (i.e. the process of "adding" strings). This allows us to build strings by using variables of different types. `Length` is a property of the array class, for every array you create you can find out its length by requesting the length property in this way. A property is simply a publicly accessible variable inside a class. As the property is a variable it has an associated data type, in this case the length variable is an `int`. Typically, you can't mix different data types, in this case a string "`Array size: `" and an int `3` but C# knows string concatenation is taking place and therefore automatically converts the `int` to a `string` using the `ToString()` function.

The `ToString()` function exists for every object defined in C# and you can call it by using the following "`arr.Length.ToString()`". It contains the representation of a particular object as a word.

In terms of combining strings (string concatenation) this is useful for including variables inside of strings that you want to display. For example:

```
01    int index = 0;
02    string display = "The item at index " + index.ToString()
      + " is " + arr[index];
03    Console.WriteLine(display);
```

If we assume the array has the contents from figure 3, the output for this in the terminal window is "`The item at index 0 is Hello!`".

Exercises:

1. Create a new console project in Visual Studio called Selections.
2. Type in the example program from above and test it to ensure it works as expected.
3. Can you modify the program so that it always prints the size of the array if it is greater than 0?

4. Can you modify the program so that you fill each space in the array?

Loops

In most programming languages there are two different types of loop structures to consider, while and for loops. While loops use a conditional statement similar to an if statement which continues to execute the code inside the loop while the conditional statement is true. Alternatively, for loops are best used for iterating through a collection of items, such as an array. Next we alter our above code to print out all the values inside the array. We will look at two examples, one which uses a while loop and one which uses a for loop.

```
01    using System;
02
03    namespace Hellow
04    {
05          class MainClass
06          {
07                public static void Main(string[] args)
08                {
09                      string[] arr = new
                        string[3]{"one","two","three"};
10                      if(arr.Length==3) {
11                            Console.WriteLine("Array size: " +
                              arr.Length);
12                            int count = 0;
13                            while(count < arr.Length){
14                                  Console.WriteLine("arr[" + count +
                              "]=" + arr[count]);
15                                  count++;
16                            }
17                      } else {
18                            Console.WriteLine("Array is too short");
19                      }
20                }
21          }
22    }
```

The new code is on lines 9 and 12-16. On line 9 we have used a shorthand notation to specify the contents of our array. This results in the following internal structure:

0	1	2
one	two	three

Figure 4: Contents of Array `arr`

Next on line 12 we create a new `int` variable called `count` which stores the number 0. Line 13 is the declaration of the while loop and inside our condition we specify that the loop should continue to execute while the count is less than the length of our array `arr`.

On line 14 we print out the information stored at the current index by using the count variable as the index into the array. Then on line 15 we increment the count by using the notation "++". This is shorthand for specifying "`count = count + 1;`" and will always increment the `count` by exactly one `int`.

Lines 14-15 will continue to be executed until the condition on line 13 is false, that is when count is equal to or greater than the array length. In this case the loop will "break" when the count is equal to 3 as this is the length of our array. After the loop breaks the next line to be executed is line 16 which simply signals the end of the while loop.

Now let's consider what this would look like with an equivalent for loop:

```
01    using System;
02
03    namespace Hellow
04    {
05          class MainClass
06          {
07                public static void Main(string[] args)
08                {
09                      string[] arr = new
                        string[3]{"one","two","three"};
10                      if(arr.Length==3) {
11                            Console.WriteLine("Array size: " +
                              arr.Length);
12                            for(int count=0; count < arr.Length;
                              count++){
13                                  Console.WriteLine("arr[" + count +
                              "]=" + arr[count]);
14                            }
15                      } else {
16                            Console.WriteLine("Array is too short");
17                      }
18                }
19          }
20    }
```

Note that in this version of the program the length of the code is reduced due to the format of the for loop statement. Every for loop is made up of 3 separate statements: iterator variable, condition, and increment. If we consider line 12 our iterator variable is an `int` called `count` which initially stores the value `0`; the condition is that the `count` must be less than the array `arr` length for the loop to continue to execute, like a while loop when this is false the loop breaks; lastly we specify the increment, in this case we increment the `count` by one on each iteration of the loop.

This will give us exactly the same output as our while loop. Determining which type of loop to use is dependent on the situation for which you are iterating. In this instance it is obvious that a for loop is the better selection as it reduces the length of the code while not obscuring its meaning. However, in other situations, such as when we want to specify multiple conditions to loop on, a while loop is the better choice. Selecting the correct loop can enhance the performance of your program and quality of your code.

Exercises:

1. Create a new project in Visual Studio called Looping.
2. Type in the example programs to ensure that they work as expected.
3. In your for loop version can you work out how to add an additional for loop to print each individual character of the string? (ask for help if you are stuck!)

Summary

Here's the good news, now you have been introduced to the basics to programming structures, you can program in any programming language you want! However, different languages have different syntax and "quirks" and are used for different purposes. For example, SQL is a language that is used for querying databases while PHP is used for server side scripting (for websites amongst others).

The bad news is that while you have seen the basics of programming structures, they are exactly that "the basics". There are so many different concepts we haven't explained in detail such as how variables are assigned in memory, other collections like lists or hash tables, complex data types, programming paradigms, how computers understand programs, algorithms, usability, human computer interaction and so on.

Programming is simply scratching the surface of the computer science and software engineering subject areas and simply a tool that software developers or engineers use to automate and build technological solutions to big problems (e.g. COVID-19 Vaccine Rollout, Internet Banking, iOS, Android etc.). To quote E.W. Dijkstra (a famous computer scientist): *"Computer Science is not about computers, any more than astronomy is about telescopes."*

Advanced Exercises

Already know about the basics of programming structures? See if you can create a Rock Paper Scissors game as a Console application. The program should allow a user to play against the computer. Use a random object to create a random selection for the computer (rock, paper, or scissors) and allow the user to type in their own selection into the terminal. Using conditional statements determine who the winner will be and print this to the console.

1. Can you iterate the game so that it can be played more than once?
2. Can you modify the game to meet the conditions of Rock Paper Scissors Lizard Spock (from The Big Bang Theory, see figure 5)?
3. How can the user type in the word "quit" to end the game and break the loop?

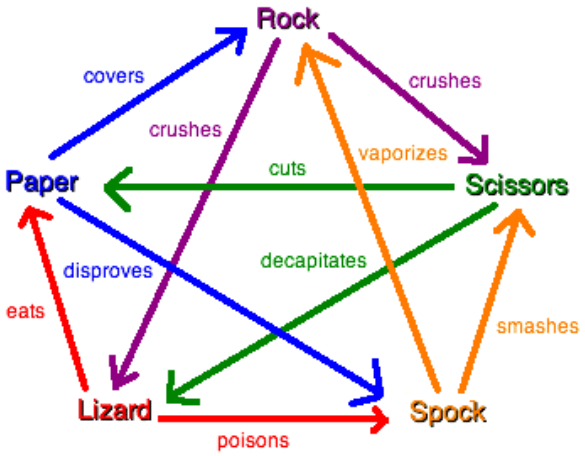


Figure 5: Rock Paper Scissors Lizard Spock

Retrieved from: https://upload.wikimedia.org/wikipedia/en/c/cc/Rock_paper_scissors_lizard_spock.png

Useful Resources

- C# Documentation: https://docs.microsoft.com/en-us/dotnet/csharp/
- W3 Schools C# Tutorials: https://www.w3schools.com/cs/index.php
- Edsger Dijkstra IEEE Profile: https://www.computer.org/profiles/edsger-dijkstra
- C# Random Objects: https://docs.microsoft.com/en-us/dotnet/api/system.random?view=net-5.0#Instantiate
- C# User Input: https://www.w3schools.com/cs/cs_user_input.php
- Rock Paper Scissors Lizard Spock: https://www.youtube.com/watch?v=zjoVuV8EeOU