

CO1404: Introduction to Programming

Lab 8 : Switch Statments, Constants and Cool Tricks (plus Coursework Q&A)

Lab Summary


The lab worksheets are designed to help you write your first programs. Students who have experience programming should find it straightforward at first. However, there will be advanced questions later on. There's a lot of reading in this worksheet as we introduce the tools required. Later worksheets will tend to be shorter, with more work for you to do!

Remember:

- If you have any questions or require assistance, please get your tutors attention. We are all here to help!
- Refer to previous weeks materials for guidance.

Objectives

This Lab aims to achieve the following objective(s) below:

- **PO.1 : Put your new found skills into practice**
 - More on Methods
 - Constants and Literals
 - Switch Statements
 - Not  Operator

Lab Activity

This lab will explore how we create and interact with arrays. It consists again of just a single stage. As usual this is required to be completed by the end of this week. As always refer to the 4 Ls: Lectures, Lecture Notes, Lab sheets and Lab Projects.

Switch Statement

1. Create a new Visual Studio C# Console project called **Misc**. All the first few exercises should be written in the same project, separated by a call to `ReadLine`.
2. Copy and paste this code into your Main method:

```
Console.Write("Choose a character: Warrior, Mage, Scout or Adventurer: ");
string characterType = Console.ReadLine().ToLower();

int health;
int strength;
int magic;
if (characterType == "warrior")
{
    health = 100;
    strength = 125;
    magic = 50;
}
else if (characterType == "mage")
{
    health = 100;
    strength = 50;
    magic = 125;
}
else if (characterType == "scout")
{
    health = 125;
    strength = 80;
    magic = 80;
}
else // Adventurer and any other input goes here (no validation here)
{
    health = 100;
    strength = 100;
    magic = 100;
}

Console.WriteLine("You begin with {0} health", health);
Console.WriteLine("You have {0} strength and {1} magic", strength, magic);
Console.ReadLine();
```

3. Convert this chain of if-else statements into a switch statement. Refer to the lecture slides. Make sure the program still works, test with all the character types.

Do you prefer this style? Remember it is only useful when you are only testing a sequence of `==` conditions.

Constants

1. In the same project add the method `DrawHorizontalLine` below. Make sure you put it in the correct place in your code. Refer to the previous lecture and lab if you've forgotten where (look at your own solutions).

```
static void DrawHorizontalLine()  
{  
    for (int i = 0; i < 80; i++)  
    {  
        Console.Write("-");  
    }  
}
```

2. Call this method twice at the start of your program, before the code for the first exercise (so you see two horizontal lines before it asks the user's character type).

In the lecture notes we see that the 80 is a "magic number" (slides 8-10), which is undesirable. Anyone looking at this code would not know why 80 was chosen. It is in fact the width of the console window in characters.

3. Update your program to use a constant for the console width, as shown on lecture slide 14. This will make the meaning of the 80 much more clear.

Stage 1

1. Now add this `DrawRightAlignedText` method in the correct place in your code:

```
static void DrawRightAlignedText( string text )
{
    // There are 80 characters on a line. Subtracting the number of
    // characters in the text. The remainder must be spaces at the left
    int leftSpaces = 80 - text.Length;
    for (int i = 0; i < leftSpaces; i++)
    {
        Console.Write(" ");
    }
    Console.Write(text);
}
```

2. This method writes text at the right of the console window rather than the left. Call this method *in between* your two horizontal lines. The method requires a parameter, so your call needs a string in the brackets. We are going to get the effect of a title so chose the text "Title Text" (or something more imaginative). The result should look something like this (note for presentation purposes the horizontal line has been reduced.):

```
-----
Title Text
-----
```

3. Update the `DrawRightAlignedText` method with the constant for the console width. You don't need a new constant, you can reuse the existing one. This is how constants should be used in programs.
4. Now write a new method of your own called `DrawCentreAlignedText`. Start by copying the code from `DrawRightAlignedText`. Read how that code works; the only difference for centred text is that the spaces on the line are half on the left, half on the right of the text. So you only need to output half the number of spaces. Make this very small change to the code and check it works by calling `DrawCentreAlignedText` to display the title at the start of your code.

Information

There will probably be a display bug at the right of your text! This happens because the code is not moving to the next line after outputting the title. Fix this simply by using `WriteLine` in the correct place. Note that the `DrawRightAlignedText` code given to you earlier cannot use `WriteLine` or it displays an extra line. Details like this often affect display layout.

Your centred text should look like this. This code might be useful to make your programs look more attractive.

```
-----  
                        Title Text  
-----
```

5. Remove the two `DrawHorizontal` line *calls* from the start your code (don't remove the method itself).
6. Now write a new method of your own called `DrawTitle`. It should accept a string parameter like the previous methods. It should display centred text in a box of `*` symbols. Unlike before this method should draw *all* the lines and the text. The result required is this:

```
*****  
*                                           *  
  *                                     *  
                                Title Text  
*                                           *  
*****
```

7. Test your method by calling `DrawTitle` to display a title at the start of your code.
 - The method will contain quite a lot of code, it needs to draw the top and bottom lines, two almost blank lines with a `*` at each end, and the title line, with centred text and a `*` at each end.
 - Make sure you use your console width constant throughout the code.
 - You may get tricky display errors on the central title line. It is much easier if you use a title with an even number of characters (such as "Title Text"). However, to solve this exercise properly, your method should work for and odd number of characters too (e.g. "Title").

Stage 2

1. There may have been a lot of similar code in your `DrawTitle` method. The `DrawTitle` method itself can call further methods. Consider what the best method to write to simplify `DrawTitle` and make it more readable

Tip: `DrawHorizontalLine` could be made much more flexible with parameters?

Recursive Methods

Advanced Challenge

A method can call *itself*. Such a method is called a **recursive** method (or function). This seems quite strange the first time you think about it. A recursive function written like this would be a problem:

```
static void MyFunction()
{
    // Do something...

    MyFunction();

    // Do something else...
}
```

This will endlessly call itself and never escape (in fact it will run out of memory and crash). However, if we put the *recursive call* in an if statement or similar, then as long as that if statement eventually fails then the *recursion* will end:

```
static void MyFunction()
{
    // Do something...

    if (some_condition)
    {
        MyFunction();
    }

    // Do something else...
}
```

There is a mathematical operation called the *factorial*. It is often used when calculating probabilities (for example it is used to calculate your chances of winning the lottery). The factorial is written with an **!** sign and can be described this:

```
1! = 1                (the factorial of 1 is 1)
2! = 1 x 2 = 2        (the factorial of 2 is 2)
3! = 1 x 2 x 3 = 6    (the factorial of 3 is 6)
4! = 1 x 2 x 3 x 4 = 24 (... )
5! = 1 x 2 x 3 x 4 x 5 = 120
etc...
```

Notice this property of factorials:

```
2! = 1! x 2
3! = 2! x 3
4! = 3! x 4
5! = 4! x 5
etc...
```

i.e. The factorial of A = (A times the factorial of A-1). Is a *recursive* definition.

1. Implement a recursive method `Factorial`. It should accept an integer parameter and return an integer result. Make sure the recursive call is guarded in an if statement. The recursion should *terminate* when you reach factorial of 1, when the result is just 1 and no recursion is needed.

Time for Review and Coursework Q&A

Use the remainder of this lab session to go back and review previous labs that you were unsure about or didn't complete. Try the advanced sections that you didn't do first time round, or attempt to reach Stage 2 to push your skill further. Also use this time to ask questions of your tutor on topics you feel weak or need extra help on. Alternatively, work on your Coursework!

CO1404 Introduction to Programming Coursework
School of Psychology and Computer Science