CO1404 : Introduction to Programming

Lab 3 : Week-3 - More on Operators & Conditions plus Introduction to Loops

Lab Summary

This lab worksheet helps 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!

Objectives

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

PO.1: xxxxxxx

Lab Activity

This weeks lab activity begins with three precursor tasks (Variable Declarations, Variables and Operators and if statements). These will help you complete stages 1 and 2 marked in the lab sheet. **Stage 1** is the ***absolute minimum*** point you should reach. Hopefully you can reach it in the lab session, if not use the drop-in help sessions and your own time to catch up. Reaching stage 2 suggests you are on target for a top class mark for the module. Set your goals sensibly - don't just aim for the minimum or you may struggle to pass the module.

> **Disclaimer:**
>
> The "Advanced Challenges" are intended as fun exercises designed to push your knowledge to the limits. I encourage you all to give them ago, ask questions if you get stuck but most importantly don't stress about them.

Expressions / Operator Precedence

For each of the following expressions calculate the two possible answers, then say which one C# will use. See the lecture notes for details on this exercise.

Check your answers with the tutor.

- **Good**: *A valid declaration with a readable variable name (see lecture)*.
- **Poor**: *A valid declaration, but a poor choice of variable name*
- **Invalid**: *The declaration has an error - it will not compile*

```
5 + 3 * 6

12 / 3 + 1

4 * 8 / 2
```

Write your answers in a document, paper, or into the worksheet and save it later for reference. It will help you during your revision. Check and explain your answers with the tutor.

Increment, Decrement and Assignment Operators

1. Create a new Visual Studio C# Console project called **MathsTrick**. Refer to week's 1 worksheet if you have forgotten how to create a new project.

2. Copy the code below into the correct place in the shell code. Again, refer to last week's lecture / lab or ask the tutor if you are not sure where to put the code:

```
Console.Write("Enter a number, any number: ");
int userNumber = int.Parse(Console.ReadLine());
int startNumber = userNumber;

Console.WriteLine("Now add 1...");
userNumber = userNumber + 1;
Console.WriteLine("You now have {0}... (press return)", userNumber);
Console.ReadKey();

Console.WriteLine("Double it...");
userNumber = userNumber * 2;
Console.WriteLine("You now have {0}... (press return)", userNumber);
Console.ReadKey();

Console.WriteLine("Now add 4...");
userNumber = userNumber + 4;
Console.WriteLine("You now have {0}... (press return)", userNumber);
Console.ReadKey();

Console.WriteLine("Now divide by 2...");
userNumber = userNumber / 2;
Console.WriteLine("You now have {0}... (press return)", userNumber);
Console.ReadKey();
Console.WriteLine("Now take away the number you first thought of...");
userNumber = userNumber - startNumber;
Console.WriteLine("You now have three : {0}", userNumber);
Console.ReadKey();
```

3. Run the code, it's a simple maths trick - the result is always three.

4. Update the code to use the operators introduced in this week's lecture such as `++` `--` `+=` `-=` `*=` `/=` . Make sure that the tick still works after your changes.
5. Notice how this program uses just one line to read a integer:

   ```
   int userName = int.Parse(Console.ReadLine());
   ```

   In the previous exercises, this was done in two lines and needed an extra `string` variable you might want to use this shorter version in your programs.

   Also this program introduces `ReadKey`, which waits for the user to press any key:

   ```
   Console.Readkey();
   ```

And, OR in `if` Statements

1. Create a new C# console application called **Decades**
2. Copy this code into the shell code. It's the example from the lecture to display if the user is in their twenties.

```
 Console.Write("Enter your age: ");
  int age = int.Parse( Console.ReadLine() );

  if (age >= 20 && age <= 29)
  {
      Console.WriteLine("You are in your twenties");
  }

  Console.ReadLine();
```

3. Add an `else` statement to display a message when the user is **not** in their twenties.

4. Add more `if-else` statements to display if the user is in their thirties, forties, or if they are a teenager. Refer to this weeks lecture on how to chain and nest statements.

A simple `while` Loop

1. Create a new C# console application called **CountingLoop**.

2. Copy this code into the shell code. It's the counting example from the lecture.

```
int counter = 1;

while (counter <= 10)
{
    Console.WriteLine( counter );
    counter++;
}
```

3. Run the code and see that it works. Now add another while loop afterwards that displays **the even numbers only** from 0 to 100. This will be a very similar piece of code.

   > Be careful about declaring the same variable twice. You cannot have two variables called counter. Either reuse the variable counter (don't declare it again, just give it a new value). Or declare a new variable.

4. Use `Write` instead of `WriteLine` so that your output is displayed in a line separated by commas (see below).

```
0, 2, 4, 6, 8, 10, 12, 14 ....
```

5. Change your loop so the numbers appear in reverse order:

```
100, 98, 96...
```

STAGE 1 - Adventure Game

Now we will use the techniques we have covered to create a text-based adventure game. Games students will identify with this week's exercise, but we'll create programs to suit to other courses in later weeks.

1.1 Starting the Game

1. Create a new C# console application called **Adventure**

2. First present an opening message welcoming the user to the adventure. It's a fantasy scenario with warriors and dragons so give it a suitable name.

3. Declare an integer variable called `characterHealth`. Initialise it to `100`.

4. Next the user must choose to be a **warrior** or a **scout**. A warrior is stronger, but a scout is faster. Declare a string variable called `characterType`.
5. Ask the user whether they want to be warrior or scout, and based on their answer assign "Warrior" or "Scout" to your `characterType` variable.

   > In this exercise the exact text of the user messages and the form of the user input is up to you. You may hit problems if you get incorrect user input, e.g. if you are expecting the user to type "Warrior" with a capital 'W', but they type it with a lower case 'w'. Don't worry about the problem this week, we will look at dealing with incorrect input next week.

1.2 The First Trial - A Riddle

At the start of the adventure, the character first reaches a huge stone door, which won't open. A huge voice booms out:

Answer this riddle to pass safely: What can you catch but cannot throw?

1. Write some text introducing this situation to the user. You might want to format your output nicely: use multiple WriteLine statements to stop the words from wrapping. Also you can write blank lines like this:

   ```
   Console.WriteLine();
   ```

2. Write some code to ask the user for the answer to the riddle storing their response in a `string` variable.

3. Then use an `if-else` statement to test if their answer was correct. The answer is **a cold**, but write your `if` statement to accept either **a cold**" *or* just **cold**. Use the C# **or** operator - revisit the lecture if you have forgotten how to.

4. In the first block of the `if-else` statement, when the user guesses correctly, the booming voice should say something like **Well done, you may pass... and the door opens**. Simply display text to describe this.

5. However, in the `else' block`, when the user guesses incorrectly, the voice says nothing, the door opens and fire jets out reducing the character's health. How much damage depends on the character type. Add another `if-else'` statement *inside* this block (this is a nested `if` statement as discussed in the lecture). If the character is a warrior their health is reduced by 20, but (else) a scout will only have their health reduced by 10 as they dodge away from the fire quickly. Use appropriate messages to describe these events.

   Advanced Challenge:

Actually, other reasonable answers to the riddle are "a bus", "a train" etc. Extend your 'if-else' statement (in a similar way to the Decades exercise above) so there are three alternatives: the "correct" answer (a cold), these alternative answers (a bus, a train), and the wrong answer. When the user types an alternative answer, the booming voice should say something sulky ("*There are no such machines in this world...*") but let the character pass unharmed anyway.

1.3 The Second Trial - A Battle

Next the character meets a dragon 🐉 blocking the way. Now there is a *turn-based* battle with the character first attacking the dragon, then the dragon counter-attacking. This will repeat as long as both character and dragon are still alive (health > 0).

1. Display text to introduce this new trial. Then declare an integer variable called `dragonHealth`. Initialise it to `150` (it's not a very big dragon...).

2. Write a `while` loop: first get the condition correct, we want the battle to continue as long as the character's health **and** the dragon's health are both greater than 0.

3. Now the content of the repeated 'while' block. We start with something simple: inside the block subtract 50 from the dragon's health and 30 from the character's health. Display text showing the damage taken and the resulting health each time. Each turn the output should look like this (a blank line between each turn):

   ```
   You strike the dragon for 50 damage, its health is 100
   The dragon counters for 30 damage, your health is 70
   ```

4. After this `while` loop, there are three possible outcomes: the character is dead, the dragon is dead, or both are dead. Use `if-else` statements to run different code for each outcome, think carefully about the conditions required:

   - If the character is alive and the dragon is dead, display a victorious message.
   - If the character is dead and the dragon is alive, display a game over message and wait for a key press. Then add this statement to quit the program.

     ```
     return;
     ```

   - The only other case is if character and dragon died in the same turn. Display a different game over message, wait for a key press and quit the program.

     > you might notice that the dragon gets a final strike even after it dies, this makes the program logic simpler, changing this is an advanced task below.

5. Now let's make the battle a little more random. Copy this line to be the first line in your program (near the `characterHealth` variable declaration):

```
Random randomNumbers = new Random();
```

> This is another part of the C# libraries that we are using here. We are *dynamically* creating an ***object*** here with the statement `new`. However, these topics are beyond the module material so we won't go into more detail

6. Now we can get random numbers at any point in the program. We want the damage dealt in the battle to be random. Meaning the dragon should strike the character for a random damage **between 1 and 30**. We can get a random number like this:

   ```
   int strike = randomNumbers.Next(1,30);
   ```

   Use a `strike` variable and code like this so the damage in the battle is random. The dragon should deal between 1 to 30 damage the character should deal between 1 to 50 damage.

1.4 Victory

As the program quits when the character dies then the only situation left is when the character defeated the dragon.

1. Let's end the game here: the character finds a room full of treasure and wins - display a suitable victory message.

STAGE 2 - Let's Create some new content

This stage consists of a single element: ***Sorting***. This exercise is designed to very difficult but you have already reached the first class stage for this lab sheet so challenges here are supposed to be very tough. You will gain experience just by thinking about this exercise.

1. When battling the dragon, allow the user to choose to attack the head, body or legs of the dragon.
   - Attacking the head will deal damage 1 to 50, but the dragon's counter attack will give damage of 1 to 40.
   - Attacking the body damages 1-30, with a counter attack of 10-20.
   - Attacking the legs damages 1-10 with a counter attack of 1-5.

2. Have a warrior deal **and** receive up to 10 extra points of damage than a scout.

3. Currently the dragon gets a final attack in the turn that it dies, change the program logic so this doesn't happen.

4. In the riddle trial, allow the user to have three attempts to guess, allowing them through on their first correct guess, damaging them with fire only on the third incorrect guess.

> This is probably best done with a loop, it is quite tricky to get all the possibilities correct - test carefully.

5. Have the riddle be chosen at random from one of:

Question	Answer
What can you catch but cannot throw?	A cold
I can run but not walk. Thoughts follow close behind me. What am I?	A door
Give me food, and I will live; give me water, and I will die.	A fire

6. Add a third trial that asks three maths problems, which must be correctly solved to pass. Choose two random numbers *and* a random operator, then ask & check the user's answer, i.e.

```
What is 24 + 65?
What is 13 * 19?
```

Advanced a challenges

This question is beyond the module material. It is only here as a puzzle for those who are interested in the deeper detail of the language. This kind of code is extremely unreadable and should never be written in real programs.

- Add a maze trial...! For example the user enters directions to navigate through it.

```
You can go North, South or West. Which way [N/S/W]?
```

- Look at the strange looking code below. Refer to the advanced note about the `++` and `--` operators in the lecture notes. Now, *without trying the code in the compiler*, try to work out what value will be displayed as a result. Don't cheat, as that makes the exercise useless. When you do have an idea then test the program and see if you were correct. Be ready to explain your answer, your tutor has my explanation.

```
int value = 5;
int result = value++ + ++value;
Console.WriteLine(result);
Console.ReadLine();
```

- What about this variation?

```
int result = ++value + ++value;
```

- Can you explain why this variation of the code compiles correctly despite the extra + sign?

```
int result = value++ + + ++value;
```

- So, what would be the result with this variation?

```
int result = value++ + - ++value;
```

Hopefully it should now be clear why I recommend only using ++ or -- in expressions with no other operators.