University of Central Lancashire
UCLan

CO1404 : Introduction to Programming

Lab 4 : More Loops, User Input Validation

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: **Explore operators, chaining & nesting statements and loops**

Lab Activity

This weeks lab activity begins with three precursor tasks. These will help you complete stages 1 and 2 marked in the lab sheet. 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.

`for` Loops

1. Create a new Visual Studio C# Console project called **ForLoops**.

2. Write a `for` loop that displays the numbers 1 to 10. Refer to the lecture notes, they contain examples of

3. Using `Write` instead of `WriteLine` and the `{0}` syntax, make the output look like this (all numbers on the same line separated by commas):

   ```
   1, 2, 3, 4, 5, 6, 7, 8, 9, 10
   ```

4. The last line of your program (and all our programs so far) wait for the user to press return before exiting:

   ```
   Console.ReadLine();
   ```

   Update it to this:

   ```
   Console.ReadLine();
   Console.Clear();
   ```

   Now it will clear the screen after the user presses return. **Now answer the next exercise immediately after this code**. Adding this code after each exercise will let you answer many exercises in the same project. When you run your program, just press return to show each of your answers in turn. You can display titles for each exercise if you like.

5. In the same project, write the following two `for` loops. Separate each answer as described above:
   - Display the odd numbers from 9 to 49.
   - Display the even numbers from 100 down to 50 in reverse order, **but not including 50**

Text Input Validation

1. Create a new Visual Studio C# Console project called **InputValidation**.

2. It is very common to ask the user a Yes/No question:

   ```
   Are you sure you want to delete the record? (y/n)
   ```

Write a program to ask this question and read the answer from the user. The program should repeatedly ask again if the user types something invalid. **There is an example of text input validation in the lecture notes.**

3. After the loop display an appropriate message depending on the user's choice. This needs an 'if' statement. The output for your program might read like this:

```
Are you sure you want to delete the record? (y/n) xyz
Sorry, you must answer 'y' or 'n'.

Are you sure you want to delete the record? (y/n) n
Record was not deleted.
```

4. We should allow the user to type `y` or `n` in lower case or upper case. Make the appropriate changes to your program to allow this. Again refer to the lecture if you are unsure.

5. Make sure you test your program thoroughly with all possible inputs. Try to break your program with unexpected inputs!

Numeric Input Validation

1. Answer this next exercise in the same project, using `Console.Clear()` as you did in the `for` loops exercise.

2. Ask the user for the year of their birth and put the result in an integer variable. The valid range for the years is from 1900 to 2011. If the user types a year outside that range *or* if the user types something that is not a number, then the program should ask the user to try again. **Again refer to the lecture.**

3. When you have a valid input display their age (approximate as we don't know their exact birthday). Example output:

```
What year were you born? 2048
Sorry, please enter a valid year.

What year were you born? Not telling you...

What year were you born? 1990
You are about 21 years old.
```

STAGE 1

This exercise tests your ability to work with a program as it becomes complex. It is easy to make small mistakes when copying earlier work.

> **Tips:**
>
> - Be careful with variable names.
> - Comment sections of code, you can put lots of blank lines between parts if you like
> - Try not to get confused as the program becomes longer and more complicated!

1.1 Simple Calculator

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

2. Copy this calculator code to the correct place. It is similar to the example from the lecture.

```
string yesNo;
do
{
    Console.WriteLine("Enter first number: ");
    double userNum1 = double.Parse(Console.ReadLine());
    Console.WriteLine("Enter second number: ");
    double userNum2 = double.Parse(Console.ReadLine());

    double userTotal = userNum1 + userNum2;
    Console.Write("{0} + {1} = {2}", userNum1, userNum2, userTotal);

    Console.Write("Do you want another go (y/n): ");
    yesNo = Console.ReadLine();

} while (yesNo == "y");
```

3. Add user input validation to the program.

- The yes/no user input is almost identical to the earlier exercise, just adapt your earlier solution into this program. Be careful, follow the tips above.
- The numbers that are input must be checked with `TryParse`, but can be in any range. Again reuse the appropriate parts of the earlier solution.

4. At the start of the program, ask the user what operation they want to perform:

   ```
   Which operation do you want + or - ?
   ```

   Validate their input (a string containing `+` or `-`).

5. Update the calculation and answer display to use the user's choice of operator.

1.2 Calendar Program - The Prototype

Now we will make a program to display a calendar of a given year. This has some very tricky parts, so we start with a ***prototype*** (a simple test version).

We will need to know the number of days in a particular month / year. The C# libraries contain a method to get this, so we don't have to program this part ourselves. The following code returns the number of days in October (month 10) of 2011, the result 31 will be put into the `numberDays` variable:

```
int numberDays = DateTime.DaysInMonth(2011, 10);
```

1. Create a new Visual Studio C# Console project called **Calendar**.

2. Ask the user for a year and validate that the year is a number and in the range 1 to 9999. Repeatedly ask the user until they enter a valid value.

3. Use a `for` loop to display for each month, the month number and the number of days in the month:

   ```
   Enter a year: 2011
   Month 1
   31 days

   Month 2
   28 days
   ...
   ```

4. In your program, select the text `DaysInMonth` and press F1 (if you are on Windows). After a moment the C# documentation will be shown for this method (you might need a Microsoft account to view the documentation). Scroll down to the "**Exception**" section. You will see that this method sets the limitation for the year within 1 to 9999. This is a good example of why documentation is important.

1.2.1 Names for Months and Days

We want to show the month and day as names rather than numbers. We could do this with lots of `if` statements, but we'll use the C# libraries again to save us that work. This is how to get the text name of the month and day from a given date (assuming you have set up the year, month and day variables):

```
// Always using day = 1 when getting month name
string monthName = new DateTime(year, month, 1).ToString("MMMM");

// Short day name (e.g. Mon Tue Wed)
string shortDayName = new DateTime(year, month, day).ToString("ddd");

// Full day name (e.g. Monday Tuesday Wednesday)
string dayName = new DateTime(year, month, day).ToString("dddd");
```

1. Within the month `for` loop create another `for` loop that count through the days in that month. You have already found the number of days, so you have the value that this new `for` loop should count up to. For each day, display its name and number. Use simple and sensible variable names as you build up this program.

   The output required is this, follow it carefully:

   ```
   Enter a year: 2011
   January
   Sat 1, Sun 2, Mon 3, Tue 4, ...

   February
   Tue 1, Wed 2, ...
   ```

2. Find the day of the week you were born.

STAGE 2

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.

2.1 Calendar Program - To completion

1. Now, we attempt a standard calendar display. We'll first assume that all months start on a Sunday to make the first step simpler. Try to make this output display each month like the example below:

```
Enter a year: 2011

January
Sun Mon Tue Wed Thu Fri Sat
01  02  03  04  05  06  07
08  09  10  11  12  13  14
15  16  17  18  19  20  21
22  23  24  25  26  27  28
29  30  31

February
Sun Mon Tue Wed Thu Fri Sat
01  02  03  04  05  06  07
```

> It might be tempting to use another nested loop for each row of the output. However, that's not the simplest way. Instead create a new integer variable `weekDay` that starts each month at 0. Increment this variable in the days loop, and test `if` it reaches 7, when it does do an extra `Console.WriteLine()` and reset the variable to 0. This creates the grid-like effect and makes the next stage simpler

2. We also need to format the numbers a little: we need an extra 0 at the start of the single digit numbers. This can be done easily with Write / WriteLine:

```
Console.Write("{0:00}  ", day);
```

The extra `:00` indicates to pad the output with `0`'s to at least two digits

Final Steps

Finally, before outputting the days for each month have a opening loop that outputs the `--` for the days not used at the beginning of the month. You can find how many `--` are needed with the C# libraries:

```
int weekDay = (int)(new DateTime(year, month, 1).DayOfWeek);
```

> *Note**
>
> I also suggest to initialise your `weekDay` variable with this value, so that the extra lines used to create the grid effect take account of the extra `--`.

The final output should look like the output below (with all the months displayed):

```
Enter a year: 2003

January
Sun Mon Tue Wed Thu Fri Sat
--  --  --  01  02  03  04
05  06  07  08  09  10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28  29  30  31

February
Sun Mon Tue Wed Thu Fri Sat
--  --   --  --  --  -- 01
02  03   04  05  06  07 08
...
```

Remember to make sure today's date has the correct day!

Advanced Challenge

The Fibonacci series is an infinite sequence of numbers. It starts with 0, 1, then each subsequent number is the sum of the last two numbers. This series occurs frequently in nature, and is the basis of various artistic ratios:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 ...
```

The series can be generated with a single `for` loop. Here's the shell, fill in the final 'step' expression to make it work. You cannot use methods/functions. Don't search an answer for this exact problem on the internet as that defeats the point of the exercise.

```
// n1, n2 are two adjacent values in the series, starting at 0,1
int n1 = 0;
for (int n2 = 1; n1 < 1000; ??? )
{
    Console.Write("{0}, ", n1);
}
```

> **Tips**
>
> This is probably best done with the rarely used **comma (`,`) operator** (look it up). But as an additional challenge, see if you can do it with only the operators covered so far in the module (clue: `a = b = 1` is a valid C# expression).