Practical Task 1.2

(Pass Task)

Submission deadline: Wednesday, July 27
Discussion deadline: Friday, August 12

General Instructions

This practical task introduces you to ***repetition*** which allows to loop over blocks of code numerous times. The number of iterations is dependent on the problem we are solving and can be controlled using three different loop structures: ***for***, ***while*** and ***do … while***. When using the ***for loop***, we need to be aware of the number of iterations we will be carrying out, however for the ***while*** or ***do … while*** they will repeat until the condition holds. Remember that the ***do … while*** will always carry out the instructions in the loop at least once, and therefore it is very useful for validating a user's input.

The tasks in the first part of this practical provide the step-by-step instructions on how to program in C# using loops. This is followed by a number of tasks for you to complete by applying the knowledge gained in the first few tasks.

1. Design an algorithm, first writing a pseudocode on paper, which prints the sum of 1, 2, 3, ..., to an upper bound determined by the user's input (e.g. 100). Then create a new C# Console Application project and write a program to achieve this. Once coded, it should also compute and display the average of the numbers. The output should look like:
   - The sum is 5050
   - The average is 50.5

   Give a new name to the main class of the project by changing it from the default `Program` to `Repetition` in the Solution Explorer of your IDE (Microsoft Visual Studio or Studio Code). Type the following code into the `Main` method of the `Repetition` class.

```
static void Main(string[] args)
{
    int sum = 0;
    double average;
    int upperbound = 100;

    for (int number = 1; number <= upperbound; number++)
    {
        sum += number;
    }

    //Compute the average as a double - remember that int divided by int produces an int
    //Print the sum and the average and check that your results are correct

}
```

   Complete the missing parts, compile, and run the program. Check whether the produced output is as expected.

2. Add a new block of code into the `Main` method. Modify the code written in the first task to use a ***while loop*** instead of the ***for loop***, which you need to replace with the following code.

```
int number = 1;
while (number <= upperbound)
{
    sum += number;
    number++;
}
```

What is the difference between the ***for*** and ***while loops***? If you want to see what is happening in the loops then trace **sum** and **number**. You can do this by adding the following line inside each of the loops after the sum calculation:

```
Console.WriteLine ("Current number: " + number + " the sum is " + sum);
```

This will allow you to see what is happening during each iteration of the loop.

3. Add one more block of code into the `Main` method. Modify the code to use a ***do … while loop*** instead of the ***while loop***. Replace the ***while loop*** with the following code.

```
int number = 1;
do
{
    sum += number;
    number++;
}while(number<=100);
```

Check that the output from all three loops is the same. What is the difference between ***for***, ***while*** and ***do … while loops***?

4. Find existing issues for all the code snippets presented below. It is highly important to try working this out on paper by tracing through the statements and then input the code into your IDE. Run it to see if you are correct and then see if you can fix the code. You may wish to add additional `Console.WriteLine` statements to check that variables contain the values you expect. Note that there may be more than one mistake.

```
int c = 0, product = 0;
while (c <= 5)
   product = product * 5;
   c = c + 1;
```

```
int a = 31, b = 0, sum = 0;
while (a != b) {
   sum = sum + a;
   b = b + 2;
}
```

```
int x = 1;
int total;
while (x <= 10) {
   total = total + x;
   x = x + 1;
}
```

```
while (y < 10) {
   int y = 0;
   Console.WriteLine(“y” + y);
   y = y + 1;
}
```

```
int z = 0;
while (z > 0) {
   z = z - 1;
   Console.WriteLine("z: " +z);
}
```

```
for(int count = 1, count < 100, count++) {
   Console.WriteLine("Hello");
}
```

```
for(int I =1; i>10; i++) {
   if (i>2) {
       Console.WriteLine (“Flower”);
   }
}
```

5. Rewrite both of the following fragments of code using ***for loops*** instead of ***while loops***.

```
int sum = 0 ;
int j = -5 ;
while ( sum <= 350 ) {
    sum += j ;
    j += 5 ;
}
```

```
int x = 0;
while ( x < 500 ) {
    Console.WriteLine( x );
    x = x + 5;
}
```

6. Write a program that will continue to prompt the user to guess a number until the correct number has been entered. You should also consider user validation; that is, has the user entered a number less than 1 or greater than 10? Have they entered a character? Start by creating a new project and name its main class (and file) as `GuessingNumber`. Then proceed with the following steps:
   - Create an integer variable called **`number`** and set it equal to 5.
   - Prompt the user to guess a number between 1 and 10.
   - If the user enters number 5, print "You have guessed the number! Well done!"
   - If they do not enter 5, continue to prompt the user to enter a number until they enter 5.

   Compile, run and check the program for potential errors.

   Now, alter the program so that it takes a number entered by a user (say, user 1). This number is set as the number to be guessed by the second user. Then ask user 2 to input their own number. Check to see if the number that user 2 entered is equal to the number user 1 entered. Continue this process until the user 2 guesses the correct number. Remember that for this program you should validate the user's input. Finally, run and check the program again.

7. Write a program which prints all the numbers between 1 and n, where n is provided by the user, that are evenly divisible by four but not by five. You should use the ***for loop*** for this question.

   Create a new project and name its main class (and file) as DivisibleFour. Compile and run the program. Check whether the produced output is as expected.

Further Notes

- You can familiarize yourself with the principles and implementation aspects of the ***for***, ***while*** and ***do … while*** statements by reading Section 2.2.3 of SIT232 Workbook available in CloudDeakin → Learning Resources.
- Access more code examples by exploring the following links:
  - https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/for
  - https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/while
  - https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/do
- The outline of how to debug your C# program code is given in Section 2.10 of SIT232 Workbook. More details are available online. For example, explore
  - https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-debugger?view=vs-2019
  - https://docs.microsoft.com/en-us/dotnet/core/tutorials/with-visual-studio-code

  for Microsoft Visual Studio and Visual Studio Code, respectively.
- If you seriously struggle with the remaining concepts used in this practical task, we recommend you to start reading the entire Sections 1 and 2 of SIT232 Workbook.

- In this unit, we will use Microsoft Visual Studio 2019 to develop C# programs. Find the instructions to install the community version of Microsoft Visual Studio 2019 available on the SIT232 unit web-page in CloudDeakin in Learning Resources → Software → Visual Studio Community 2019. You however are free to use another IDE, e.g. Visual Studio Code, if you prefer that.

Submission Instructions and Marking Process

To get your task completed, you must finish the following steps strictly on time.

- Make sure your programs implement the required functionality. They must compile and have no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your programs thoroughly before submission. Think about potential errors where your programs might fail.
- **Submit** the expected code files as a solution to the task via OnTrack submission system. You must record a short video explaining your solution to the task. Upload the video to one of accessible resources and refer to it for the purpose of marking. You must provide a working link to the video to your marking tutor in OnTrack. Note that the video recording must be made in the **camera on mode**; that is, the tutor must see both the presenter and the shared screen.

  For this task, in your video recording, you must **demonstrate how you run your program in the debug mode** and how you can use it to detect and fix potential errors. Note that debugging is an essential skill necessary to implement errorless code: The sooner you start debugging your programs the sooner you will become proficient with programming and understanding the theory part. We do not expect you to demonstrate this skill for every program you wrote in this task; you only need to select the most difficult program as an example. In the video, complement the debugging process with your verbal explanation.
- Once your solution is accepted by the tutor, you will be invited to **continue its discussion and answer relevant theoretical questions** through the Intelligent Discussion Service in OnTrack. Your tutor will record several audio questions. When you click on these, OnTrack will record your response live. You must answer straight away in your own words. As this is a live response, you should ensure you understand the solution to the task you submitted.

  Answer all additional questions that your tutor may ask you. Questions will cover the lecture notes; so, attending (or watching) the lectures should help you with this **compulsory** discussion part. You should start the discussion as soon as possible as if your answers are wrong, you may have to pass another round, still before the deadline. Use available attempts properly.

Note that we will not accept your solution after **the submission deadline** and will not discuss it after **the discussion deadline**. If you fail one of the deadlines, you fail the task, and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Remember that this is your responsibility to keep track of your progress in the unit that includes checking which tasks have been marked as completed in the OnTrack system by your marking tutor, and which are still to be finalised. When grading your achievements at the end of the unit, we will solely rely on the records of the OnTrack system and feedback provided by your tutor about your overall progress and the quality of your solutions.