Practical

Flow Control

It is not always the case that a program will have only one outcome and depending on the data entered may have many outcomes.

To enable a program to change the way it behaves we can use flow control. Flow control enables a program to make decisions based on the data being processed.

For instance your program may not be able to process data that is less than a certain value, so we need methods to define these decisions and enable the flow of the execution to change accordingly.

There are two methods to manage flow control.

One method is the 'if' statement and where you have a structured series of decision to make, you can use a 'switch' statement.

This practical will introduce both methods.

Consider the following problem.

Example 1 : You need to write a program that determines the average of some given data.

average = total/count.

Your program needs to examine the data entered to ensure the count is greater than 0 as the value is used in a calculation that would crash the program or give an illogical result if it was equal to 0.

```
using System;
namespace prac_Ex1
{
   class average
   {
      static void Main(string[] args)
      {
         float total;
         int count;
         float average;
         string tempVal = "";

         Console.Write("Enter the total > ");
         tempVal = Console.ReadLine();
         total = float.Parse(tempVal);
```

```
            Console.Write("Enter the count > ");
            tempVal = Console.ReadLine();
            count = int.Parse(tempVal);

            average = total / count;

            Console.WriteLine("Total : " + total);
            Console.WriteLine("Count : " + count);
            Console.WriteLine("Average : " + average.ToString("F2"));
            Console.ReadLine();
        }
    }
}
```

Try the following:

1. Create a new project prac_Ex1 and enter the code above.
2. Compile and run the code and enter a value of 0 for the count.

Example 2 : To ensure the program produces logical results we can add flow control to the code to ensure the data is valid before processing the average.

```
using System;
namespace prac_Ex2
{
    class average
    {
        static void Main(string[] args)
        {
            float total;
            int count;
            float average;
            string tempVal = "";

            Console.Write("Enter the total > ");
            tempVal = Console.ReadLine();
            total = float.Parse(tempVal);

            Console.Write("Enter the count > ");
            tempVal = Console.ReadLine();
            count = int.Parse(tempVal);

            if (count <= 0)
            {
                Console.WriteLine("Error : the count value entered must be greater
than 0. Please try again");
            }
            else
            {
                average = total / count;
                Console.WriteLine("Total : " + total);
```

```
            Console.WriteLine("Count : " + count);
            Console.WriteLine("Average : " + average.ToString("F2"));
        }
        Console.ReadLine();
    }
  }
}
```

The updated code now contains a control block that ensures only valid output is produced by the program.

If the count is less than or equal to 0 meaning the if statement is true, then an error message is presented and the code to determine and print the average is never executed.

If the count is greater than 0 or the if statement is false, the average is calculated and printed.

Note the indentation and the use of the '{' and '}' to associate code with the flow control outcome.

Try the following:

1. Update project prac5_Ex1 and enter the code above.
2. Compile and run the code and enter a value of 0 for the count to verify the error message is presented.

Example 3 : To illustrate the use of multiple statements in the control block, the following breaks the 'if' statement into two questions that reads as follows.

If the count equals 0 OR the count < 0 then error.

```
using System;
namespace prac_Ex3
{
  class average
  {
    static void Main(string[] args)
    {
      float total;
      int count;
      float average;
      string tempVal = "";

      Console.Write("Enter the total > ");
      tempVal = Console.ReadLine();
      total = float.Parse(tempVal);

      Console.Write("Enter the count > ");
```

```
            tempVal = Console.ReadLine();
            count = int.Parse(tempVal);

            if (count == 0 || count < 0)
            {
               Console.WriteLine("Error : the count value entered must be greater
than 0. Please try again");
            }
            else
            {
               average = total / count;
               Console.WriteLine("Total : " + total);
               Console.WriteLine("Count : " + count);
               Console.WriteLine("Average : " + average.ToString("F2"));
            }
            Console.ReadLine();
         }
      }
   }
```

Try the following:

- Update project prac_Ex1 and enter the code above.
- Compile and run the code and enter a value of 0 for the count to verify the error message is presented.

Example 4 : An alternative to the program could also add code to exit the program once an error is detected. This is sometimes required to ensure no more processing takes place.

We can use the Environment.Exit(0); class to terminate the program and return a number to indicate the reason for the termination.

You will notice that now we are exiting the program if an error is detected, we do not need the else and the code is not contained with in the '{ .... }' code block.

```
using System;
namespace prac_Ex4
{
   class average
   {
      static void Main(string[] args)
      {
         float total;
         int count;
         float average;
         string tempVal = "";

         Console.Write("Enter the total > ");
         tempVal = Console.ReadLine();
         total = float.Parse(tempVal);
```

```
            Console.Write("Enter the count > ");
            tempVal = Console.ReadLine();
            count = int.Parse(tempVal);

            if (count <= 0)
            {
               Console.WriteLine("Error : the count value entered must be greater
than 0. Please try again");
               Environment.Exit(0);
            }

            average = total / count;
            Console.WriteLine("Total : " + total);
            Console.WriteLine("Count : " + count);
            Console.WriteLine("Average : " + average.ToString("F2"));

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

Try the following:

1. Update project prac_Ex3 and enter the code above.
2. Compile and run the code and enter a value of 0 for the count to verify the error message is presented and the program exits.
3. Extend the code provided in the example to perform the following:
    1. Test the value for **total** is between 0 and 5000
    2. Report an error if the total fails the test and exit the program

Example 5 : Currently the program checks the data entered is a valid value before processing the average but it does not stop a user entering non numbers into the prompt and as a consequence crashing the program.

Try the following:

1. Re run the program and enter a non number value for the **total** such as 'dog'
2. The program will crash as the data entered cannot be converted.

To enable the program to correctly identify the data is valid we can use the TryParse method.

TryParse is similar to the Parse method in that it attempts to Parse a value into the data type of typewhich is making the TryParse call. However, should the parsing fail there will not be an error. So as long as you are aware the method could return a zero (false) when in fact the parsing has failed, then this method can help to prevent errors occurring in your code.

Consider the following code :

```
string tempVal = "54";
int number;
bool parseAttempt = false;
parseAttempt = Int32.TryParse(tempVal, out number);
if (parseAttempt == true)
{
  Console.WriteLine("TryParse was successful and the value is " + number);
}
```

We declare a boolean variable **parseAttempt** to store the result of the **TryParse** and set it to **false**.

```
bool parseAttempt = false;
```

We then call the **Int32.TryParse** method where the number we are trying to parse is stored in the **tempVal** variable.

```
parseAttempt = Int32.TryParse(tempVal, out number);
```

If the **TryParse** is able to parse the **tempVal** string, the result will be converted to an integer and stored in the **number** variable. The **out** keyword is required because we need to pass the reference of the **number** variable. We will discuss this in future practicals.

If the **TryParse** is successful it returns a boolean value **true** and **false** if the **TryParse** could not convert the string **tempVal**.

We then can use an **if** statement on the **parseAttempt** variable to test the outcome and perform the actions assigned.

```
if (parseAttempt == true)
{
  Console.WriteLine("TryParse was successful and the value is " + number);
}
```

The following is the code from example 4 with the **TryParse** method included to check the **total** entered by the users is a valid float.

```
using System;
namespace prac_Ex5
{
   class average
   {
      static void Main(string[] args)
      {
         float total;
         int count;
         float average;
         string tempVal = "";
         bool parseAttempt = false;

         Console.Write("Enter the total > ");
         tempVal = Console.ReadLine();
         parseAttempt = float.TryParse(tempVal, out total);

         if (parseAttempt == false)
         {
            Console.WriteLine("Error : the total entered was not a valid decimal
value. Please try again");
            Environment.Exit(0);
         }

         Console.Write("Enter the count > ");
         tempVal = Console.ReadLine();
         count = int.Parse(tempVal);

          if (count <= 0)
         {
            Console.WriteLine("Error : the count value entered must be greater
than 0. Please try again");
            Environment.Exit(0);
         }

         average = total / count;
         Console.WriteLine("Total : " + total);
         Console.WriteLine("Count : " + count);
         Console.WriteLine("Average : " + average.ToString("F2"));

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

Try the following:

1. Update project prac_Ex4 and enter the code above.
2. Compile and run the code and enter a value of 'dog' for the total to verify the error message is presented.
3. Extend the code provided in the example to perform the following:
    1. Convert the **count** using **TryParse** and report an error if the **count** entered is not a valid integer.

Example 6: The following code presents a menu to the user and waits for a response.

It then uses a nested 'if' statement to test the response and outputs a corresponding message to confirm the response entered.

```
using System;
namespace prac_Ex6
{
   class menu
   {
      static void Main(string[] args)
      {
         int response = 0;
         string tempVal = "";
         bool parseAttempt = false;

         Console.WriteLine("1. Make a phone call");
         Console.WriteLine("2. Send an SMS");
         Console.WriteLine("3. Find a contact");
         Console.WriteLine("4. Quit");
         Console.Write("Enter your selection > ");

         tempVal = Console.ReadLine();
         parseAttempt = Int32.TryParse(tempVal, out response);

         if (parseAttempt == false)
         {
            Console.WriteLine("Error : the response entered is not a valid number.
Please try again");
            Environment.Exit(0);
         }

         if(response == 1)
            Console.WriteLine("User selected to make a phone call, option " +
response);
         else
         if(response == 2)
            Console.WriteLine("User selected to send an SMS, option " +
response);
         else
         if (response == 3)
            Console.WriteLine("User selected to find a contact, option " +
response);
         else
         if (response == 4)
            Console.WriteLine("User selected to Quit, option " + response);
         else
            Console.WriteLine("Unknown selection, option " + response);

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

Try the following:

1. Create a new project prac_Ex6 and enter the code above.
2. Compile and run the code and test the response entered is correct.
3. Now remove the **else** statements and examine what occurs.
4. Add another menu item to the menu **"View SMS messages received"** and add the code to the **if** statement to process the response.

Example 7: The code presented in **Example 6** can become very cumbersome if the size of the nested **if** statement grows.

To provide a more concise and structured way of managing this type of processing we can use the **switch case** statement.

```
using System;
namespace prac_Ex7
{
   class menu
   {
      static void Main(string[] args)
      {
         int response = 0;
         string tempVal = "";
         bool parseAttempt = false;

         Console.WriteLine("1. Make a phone call");
         Console.WriteLine("2. Send an SMS");
         Console.WriteLine("3. Find a contact");
         Console.WriteLine("4. Quit");
         Console.Write("Enter your selection > ");

         tempVal = Console.ReadLine();
         parseAttempt = Int32.TryParse(tempVal, out response);

         if (parseAttempt == false)
         {
            Console.WriteLine("Error : the response entered is not a valid number.
Please try again");
            Environment.Exit(0);
         }

         switch (response)
         {
            case 1 :
               Console.WriteLine("User selected to make a phone call, option " +
response);
               break;
            case 2 :
               Console.WriteLine("User selected to send an SMS, option " +
response);
               break;
            case 3 :
```

```
                Console.WriteLine("User selected to find a contact, option " +
response);
                break;
            case 4 :
                Console.WriteLine("User selected to Quit, option " + response);
                break;
            default :
                Console.WriteLine("Unknown selection, option " + response);
                break;
        }
        Console.ReadLine();
    }
  }
}
```

The **switch** examines the value inside the brackets then tests it against each **case** statement until it finds a match.

It then processes all the commands up until the next **break** statement. The **default** is used where no match can be found.

Try the following:

1. Adjust project prac_Ex6 and enter the code above.
2. Compile and run the code and test the response entered is correct.
3. Add another menu item to the menu **"View SMS messages received"** and add the code to the **switch** statement to process the response.

Question 1: Write a program that acts like a reverse polish calculator.

Your program should prompt for two decimal numbers then the action to perform on the numbers followed by the result.

For example :

Enter number > 34

Enter number > 2

Enter action (+ - * /) > +

Result is 36

Your program should do the following:

1. Declare two decimal variables

2. Declare a char for the action
3. Prompt for the first number
4. Check the number is valid using TryParse and report an error and exit if an error is detected
5. Prompt for the second number
6. Check the number is valid using TryParse and report an error and exit if an error is detected
7. Prompt for the action
8. Using a nested **if** statement or a **switch** statement test the **action** entered and apply the corresponding process to the numbers
9. Output the result