PE-4 – Flow Control

1. If we have two integers stored in variables `var1` and `var2`, what logical expression with a Boolean operation can we perform to see if one or the other (but not both) is greater than 10?

   Hint: write the expressions to test var1 and var2 which will evaluate to boolean true/false values, then apply the correct Boolean operator which has the following truth table:

operand1	operand2	result
false	false	false
false	true	true
true	false	true
true	true	false

(operand1 and operand2 are boolean expressions involving var1 and var2)

- result = operand1 ^ operand2

2. Write a console application that includes the logic from Exercise 1, that obtains two numbers from the user and displays them, but rejects any input where both numbers are greater than 10 and asks for two new numbers.

   GitHub URL of your project: https://github.com/Sebas11an245/myIGME-201/tree/main/PE4

3. What is wrong with the following code?

```
int i;
for (i = 1; i <= 10; i++)
{
      if ((i % 2) = 0)
      {
            continue;
      }

      Console.WriteLine(i);
}
```

In the if-statement, it is trying to set (i%2) to 0. The fix is changing = to == so that it is comparing.

4. Trying to compile the following lines of code results in the error shown below.

```
int number = 3;
switch (number)
{
      case 1:
            Console.WriteLine("Just");

      case 2:
      case 4:
            Console.WriteLine("an");
            break;

      default:
            Console.WriteLine("example");
            break;
}
```

Error Message: `Control cannot fall through from one case label to another.`

a) Why are you getting that message? Don't just rewrite the error, explain it.
    a. There is no way for the code to get out of case one, i.e, break.
b) How would you fix the code? Describe your fix, or write out just the fixed portion of the code.
    a. Adding break, at the end of each case would fix it.
c) Once the code compiles, what will the output be? __example_____
d) How would you rewrite the code to also execute the default in the case of 2 or 4 (but default will still only display "example"!)
    a. Copy output code into the other cases.

5. Is the following switch statement valid? Why or why not? Alter the code to correct any errors.

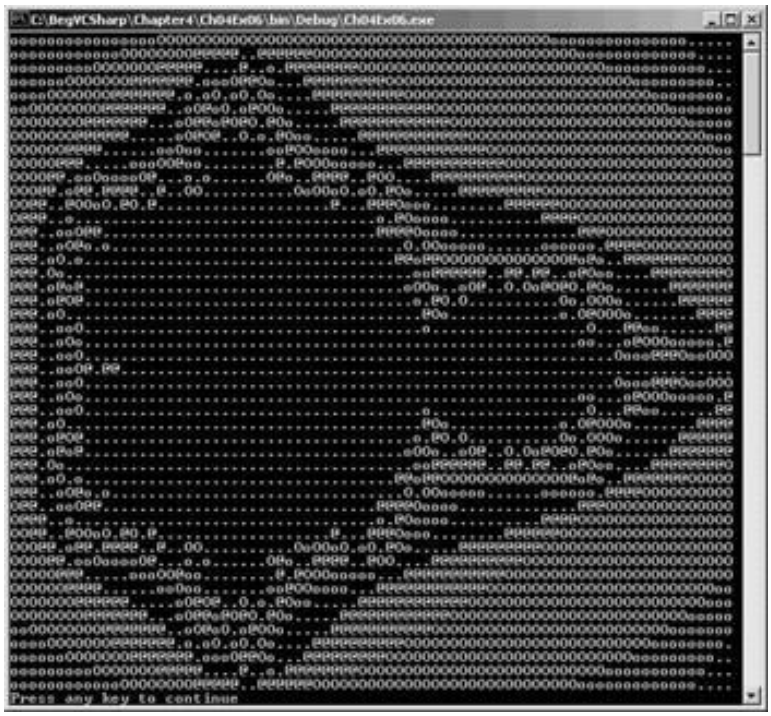
```
string favoriteFood = "spaghetti";
switch("spaghetti")
{
      case ("spaghetti"):
          Console.WriteLine("You seem to really like spaghetti");
           break;
      case ("cake"):
          Console.WriteLine("Wrong. Spaghetti is the best food.");
          break;
      default():
          Console.WriteLine("You should like spaghetti.");
}
```

The code was missing break statements to get out of the cases.

6. An exercise in understanding and modifying (hopefully) interesting code from "Beginning C#" by Karli Watson.

First watch this intriguing video about the Mandelbrot set:
https://rit.zoom.us/rec/share/FFpSZYXtVfNf41vU_6I9FkVrA7cn7bAFAUumJdV_dobxwSye2PleOwjBwDjjvwLs.XMgLvMukJy-i40JV

Copy and Paste the code in the attached Mandelbrot.cs into a new "Mandelbrot" console application. Compile and execute the code:



How it Works

Each position in a Mandelbrot image corresponds to an imaginary number of the form `N = x + y*i`, where the real part is `x`, the imaginary part is `y`, and `i` is the square root of `-1`. The `x` and `y` coordinates of the position in the image correspond to the `x` and `y` parts of the imaginary number.

For each position on the image we look at the argument of `N`, which is the square root of `x*x + y*y`. If this value is greater than or equal to 2 we say that the position corresponding to this number has a value of 0. If the argument of `N` is less than 2 we change `N` to a value of `N*N - N` (giving us `N = (x*x-y*y-x) + (2*x*y-y)*i)`, and check the argument of this new value of `N` again. If this value is greater than or equal to 2 we say that the position corresponding to this number has a value of 1. This process continues until we either assign a value to the position on the image or perform more than a certain number of iterations.

Based on the values assigned to each point in the image we would, in a graphical environment, place a pixel of a certain color on the screen. However, as we are using a text display we simply place characters on screen instead.

Let's look at the code, and the loops contained in it.

We start by declaring the variables we will need for our calculation:

```
double realCoord, imagCoord;
double realTemp, imagTemp, realTemp2, arg;
int iterations;
```

Here, `realCoord` and `imagCoord` are the real and imaginary parts of `N`, and the other `double` variables are for temporary information during computation. `iterations` records how many iterations it takes before the argument of `N` (`arg`) is 2 or greater.

Next we start two `for` loops to cycle through coordinates covering the whole of the image (using slightly more complex syntax for modifying our counters than `++` or `--`, a common and powerful technique):

```
for (imagCoord = 1.2; imagCoord >= -1.2; imagCoord -= 0.05)
{
   for (realCoord = -0.6; realCoord <= 1.77; realCoord += 0.03)
   {
```

I've chosen appropriate limits to show the main section of the Mandelbrot set. Feel free to play around with these if you want to try 'zooming in' on the image.

Within these two loops we have code that pertains to a single point in the Mandelbrot set, giving us a value for `N` to play with. This is where we perform our calculation of iteration required, giving us a value to plot for the current point.

First we initialize some variables:

```
iterations = 0;
realTemp = realCoord;
imagTemp = imagCoord;

arg = (realCoord * realCoord) + (imagCoord * imagCoord);
```

Next we have a `while` loop to perform our iterating. We use a `while` loop rather than a `do..while` loop, in case the initial value of `N` has an argument greater than 2 already, in which case `iterations = 0` is the answer we are looking for and no further calculations are necessary.

Note that I'm not quite calculating the argument fully here, I'm just getting the value of `x*x + y*y` and checking to see if that value is less than 4. This simplifies the calculation, as we know that 2 is the square root of 4 and don't have to calculate any square roots ourselves:

```
while ((arg < 4) && (iterations < 40))
{
   realTemp2 = (realTemp * realTemp) - (imagTemp * imagTemp) -
               realCoord;
   imagTemp = (2 * realTemp * imagTemp) - imagCoord;
   realTemp = realTemp2;
   arg = (realTemp * realTemp) + (imagTemp * imagTemp);
   iterations += 1;
}
```

The maximum number of iterations of this loop, which calculates values as detailed above, is 40.

Once we have a value for the current point stored in `iterations` we use a switch statement to choose a character to output. We just use four different characters here, instead of the 40 possible values, and use the modulus operator (%) such that values of 0, 4, 8, and so on give one character, values of 1, 5, 9, and so on give another character, etc.:

```
switch (iterations % 4)
{
   case 0:
      Console.Write(".");
      break;
   case 1:
      Console.Write("o");
      break;
   case 2:
      Console.Write("O");
      break;
   case 3:
      Console.Write("@");
      break;
}
```

Note that we use `Console.Write()` here rather than `Console.WriteLine()`, as we don't want to start a new line every time we output a character.

At the end of one of the innermost `for` loops, we do want to end a line, so we simply output an end of line character using the escape sequence we saw earlier:

```
}
Console.Write("\n");
```

```
    }
```

This results in each row being separated from the next and lining up appropriately.

The final result of this application, though not spectacular, is fairly impressive, and certainly shows how useful looping and branching can be.

Your task:

Modify the Mandelbrot set application to request image limits from the user and display the chosen section of the image. Make every image chosen fit in the same amount of space to maximize the viewable area.

In other words, prompt the user for new start and end values of imagCoord and realCoord for the 2 loops. Note that imagCoord decrements and realCoord increments, so prompt the user logically and suggest what the default values are as a guideline. Adjust the increment for each loop so that there are 48 values of imagCoord (which is the number of horizontal lines the image will display in) and 80 values of realCoord (which is the number of vertical columns the image will display) so that the data range maximizes the viewable area.

Do not allow the user to enter invalid values: imagCoord must start from a higher value than it ends, and realCoord must start from a lower value than it ends.

/* imagCoord controls each line of the display, the 0.05 increment for the default range generates 48 lines of the image */

```
for (imagCoord = 1.2; imagCoord >= -1.2; imagCoord -= 0.05)
{
   /* realCoord controls each character on each line of the output.  The 0.03 increment for the
      default range generates 80 characters on each line of the image */
   for (realCoord = -0.6; realCoord <= 1.77; realCoord += 0.03)
   {
```

GitHub URL for your Mandelbrot project:

Submission

Upload this completed document.
Add, Commit and Push the projects for #2 and #6 to your GitHub repository.