

## EXERCISE 04 - IF-STATEMENTS WITH LOGICAL OPERATORS

**IMPORTANT:** Before submission, make a copy of your 'Program.cs' file for each question and then rename each file to the following:

## File Names:

- *last\_name\_first name\_U2\_E04\_1.cs*
- *last\_name\_first name\_U2\_E04\_2.cs*
- *last\_name\_first name\_U2\_E04\_3.cs*

**Note:** Along with last name and first name, make sure the end of the filename (i.e., before the .cs) has the **unit number**, **exercise number**, and **question number**. For example:

smith\_john\_U1\_E03\_2.cs



1. Write a program that asks the user for 3 **unique** integers. Have your program check that each integer is in fact unique (display a message if not), then output to the console which integer is the largest. Keep your code minimal by making use of **logical operators**!
2. The Saffir-Simpson Hurricane Scale provides a rating (a category) depending on the current intensity of a hurricane:

Saffir-Simpson Hurricane Scale for wind speeds:

**Category 1:** 74-95 mph or 64-82 kt or 119-153 km/hr

**Category 2:** 96-110 mph or 83-95 kt or 154-177 km/hr

**Category 3:** 111-130 mph or 96-113 kt or 178-209 km/hr

**Category 4:** 131-155 mph or 114-135 kt or 210-249 km/hr

**Category 5:** greater than 155 mph or 135 kt or 249 km/hr

**Part 1)** Write a program that asks the user for a hurricane speed type and hurricane speed value **exactly** as follows:

Sample **Input:**

- 1) mph
- 2) kt
- 3) km/hr

Which hurricane speed do you wish to use? **2**

How fast is the hurricane (kt): **91**

**Part 2)** Based on the Saffir-Simpson Hurricane Scale above, **output** to the screen which **hurricane category** the speed entered falls into. Keep your code minimal by making use of **logical operators**!

3. **[THINK]** Consider the following code for integer input:

```
int x = 0;
Console.Write("Enter a number: ");
x = Convert.ToInt32(Console.ReadLine());
```

If the user enters an integer, then everything will be fine, however if the user enters a decimal value, then the program will crash:

Sample **Input** & **Output**:

Please enter an integer:

Program runs ok!

Sample **Input** & **Output**:

Please enter an integer:

Program will crash!

This is due to the fact we cannot convert 5.2 to an integer with our above code. One way around this is to change our variable 'x' to the double data type and convert our input to a double:

```
double x = 0;
Console.Write("Enter a number: ");
x = Convert.ToDouble(Console.ReadLine());
```

Sample **Input** & **Output**:

Please enter an integer:

Program runs ok!

Sample **Input** & **Output**:

Please enter an integer:

Program will ok!

Now our program runs fine because double variables can either contain a decimal or not. If we wanted the user to have the option of entering either an integer or a decimal, then this seems like the more appropriate way of doing so. However, what if we wanted to determine if the user typed an integer or a decimal while using the code above? We can accomplish this by extracting the decimal portion of the input. Consider the following code:

```
double x = 0;
Console.Write("Enter a number: ");
x = Convert.ToDouble(Console.ReadLine());

double? dec = x - (int)x;
```

After we ask the user for a number, we can extract the decimal and store it in a separate variable called '**dec**'. Notice we have '**dec**' equal to '**x**' minus the integer cast of '**x**'. This will subtract the entire value of '**x**' by the integer value of '**x**' leaving us with just the decimal. To demonstrate, let's say the user entered '5.2' for '**x**'. The variable '**dec**' would equate as follows:

$$\begin{aligned} \text{dec} &= 5.2 - (\text{int})5.2 \\ &= 5.2 - 5 \\ &= 0.2 \end{aligned}$$

We have successfully extracted the decimal portion of the user input. This is because we subtracted '5.2' by the **integer cast** of '5.2'. The integer cast of '5.2' will simply equal '5'. Therefore 5.2 – 5 equals 0.2.

If the user entered '5' for **x**, the variable '**dec**' would equate as follows:

$$\begin{aligned} \text{dec} &= 5 - (\text{int})5 \\ &= 5 - 5 \\ &= 0 \end{aligned}$$


In this case there is no decimal if the user enters '5' since '5' minus the integer cast of '5' is going to be '0'.

We can now determine if the user typed an integer by checking if the variable '**dec**' is equal to '0' (user typed an **integer**), or '**dec**' is greater than '0' (user typed a **decimal**). **Note:** This will still work if the user types a negative number, try it out!

Write a program that asks the user for a positive, even number. Output to the user if the number they entered is an integer or a decimal but **only** if the number is in fact positive and even. In all other cases tell the user that they did not enter a positive, even number. Keep your code minimal by making use of **logical operators**!

**Hint:** When determining if the number is odd or even, make use the modulus '%' operator. If the user enters a decimal number, then use an **integer cast** when testing the modulus of the inputted number.

Sample Input & Output:



```
Enter a number positive, even number: 4.3
```

```
You entered a positive, even decimal number
```

Sample **Input** & Output:

```
Enter a number positive, even number: 4
```

```
You entered a positive, even integer number
```

Sample **Input** & Output:

```
Enter a number positive, even number: 3.4
```

```
You did not enter a positive, even number!
```

Sample **Input** & Output:

```
Enter a number positive, even number: -4.3
```

```
You did not enter a positive, even number!
```