Lab 3 : Programming with Decisions and Loops

Topics covered mixed variable types in expressions, Boolean logic, the `&&`,`||` and `%` operators, desk checking, `if`/`else` statements, `while` loops

---

Exercise 1: Given the following variable declarations and initialisations, evaluate the expressions below and indicate the data type of the expression. The first 2 expressions have been evaluated as examples for you.

```java
int count = 8;
double size = 12.0;
int index = 12;
boolean flag = false;
String str1 = "Hello";
String str2 = "World";
```

`size / count`	result value: **1.5** result type: **double**
`index <= size`	result value: **true** result type: **boolean**
`index / count`	
`count != 8`	
`size + index / count`	
`!(index == size)`	
`count > 8 || !flag`	
`str1 + str2`	
`index % 5 <= 3`	
`(count - 2 > index % 7) && (flag || size < 20)`	
`str1.length() + size`	
`flag || (count == index) || str1.equals(str2)`	
`str1.charAt(3) == str2.charAt(3)`	

Exercise 2: Write a program that reads an integer number from the user and tests to see if the number supplied is both *even* and lies in the range 20 – 200 inclusive, or is both *odd* and negative. For example, '5' fails this test because while it's odd, it's not negative. '22' passes the test because it's both even and in the right range.

Hint: You may want to use the modulus operator `%` , which gives the remainder when one integer is divided by another. Here's an example:

```java
int x = 20;
int y = 2;
if (x % y == 0) {
   System.out.println(y + " is a factor of " + x);
}
```

Exercise 3: One of the most important parts of programming is checking to see if your code performs in the way you expect it to. Perform a **desk check** on your answer of exercise 2 using various inputs to test the program's behaviour.

Remember to try normal, abnormal and boundary values for the input and, most importantly, make sure you do this on paper! Just running the code with the desired inputs does not tell you whether the internal workings of your program are performing as expected.

Exercise 4: Drawing a **flowchart** of the program can also allow you to see flaws in your logic, and will help you choose which values to test in your desk check.

Draw a flowchart for the program you created in exercise 2.

Exercise 5: Write a program that determines whether an input integer is even — but you can't use the modulus operator `%`: for this one you'll have to use *casting*. There are several ways to do this. It may be helpful to draw a diagram of the logic part of this exercise to see if you can figure out how to do it.

Note: **Casting** is the act of treating a variable (or result of an expression) as another data type. The syntax is given by `(<type>)(<expression>)`, where `<type>` is the data type you wish to convert to, and `<expression>` is the variable or expression you wish to convert. Note that if `<expression>` is just a single variable, it does not need to have brackets around it.

Remember: when you cast an expression to a certain data type, this does not affect the variables in memory. It only treats the variable (or result of an expression) as another data type for the duration of that statement. Here is an example:

```java
    int x = 5;
    int y = 2;

    // This will output 2, as we are performing integer division
    System.out.println(x / y);

    // Will output 2.5, as the x is cast to double for the duration
    // of the calculation
    System.out.println((double)x / y);

    // Will output 2.0, as the division is still integer division, but
    // the result is cast to a double
    System.out.println((double)(x / y));
```

Exercise 6: A leap year is a year which contains 366 day(the extra day is the 29th of February). To determine whether a year is a leap year, follow these steps:

Step 1: If the year is divisible by 4, go to step 2. Otherwise, it is not a leap year.
Step 2: If the year is divisible by 100, go to step 3. Otherwise, it is a leap year.
Step 3: If the year is divisible by 400, it is a leap year. Otherwise, it is not a leap year.

For example, 2016, 2000 are leap years, but 1997, 1900 are not leap years.

Create a `LeapYear` class to read a year from the **command-line arguments**, and output if this year is a leap year. Again, perform a desk check and draw a flowchart for your program.

You will need the Integer.parseInt() method to convert the String type input into an integer;

```
  int year = Integer.parseInt(args[0]);
```

```
> java LeapYear 2016
   2016 is a leap year!
> java LeapYear 2011
   2011 is not a leap year!
```

For the next three exercises, you will be creating and modifying a program called `NumberCrunch`, which reads in numbers from the user and then outputs information about those numbers.

Exercise 7: $\boxed{\text{Part 'A'}}$ Create a `NumberCrunch` class, and make it read in **up to** three integers from the user. As soon as the user inputs a negative number, the program should stop trying to read numbers. Display to the user how many positive numbers were read in. If the user has not entered any positive numbers, `NumberCrunch` should tell the user to input at least one positive number.

```
> java NumberCrunch
   Please enter up to three positive numbers:
   2 6 -1
   You have entered 2 positive numbers.
> java NumberCrunch
   Please enter up to three positive numbers:
   -1
   You have not entered any positive numbers. Please input at least one positive number.
```

Exercise 8: $\boxed{\text{Part 'B'}}$ If the user has entered exactly one positive number, `NumberCrunch` should print out all factors of that number. Remember: y is a factor of x if, when you divide x by y, there's no remainder. You should use the *modulus* operator `%` for this. You will have to use a loop for this exercise.

```
> java NumberCrunch
   Please enter up to three positive numbers:
   12 -1
   You have entered 1 positive number.
   The factors of 12 are: 1, 2, 3, 4, 6, 12.
```

Exercise 9: $\boxed{\text{Part `C'}}$ If the user has entered two or three positive numbers, `NumberCrunch` should print out the largest of the input numbers.

```
> java NumberCrunch
   Please enter up to three positive numbers:
   5 7 -1
   You have entered 2 positive numbers.
   The largest is 7.
> java NumberCrunch
   Please enter up to three positive numbers:
   10 30 20 -1
   You have entered 3 positive numbers.
   The largest is 30.
```

Extensions

Extension 1: Modify your `NumberCrunch` program so that when the user inputs a single number, the program prints out all **prime** factors of that number. Prime factorisation is not as simple as regular factorisation!

Extension 2: Modify your `NumberCrunch` program so not all the functionality is built in the `main` method. The methods will begin like this:

```java
    public static void printFactors(int a) {
        // ... your code here
    }
```

```java
    public static void printMax(int a, int b, int c) {
        // ... your code here
    }
```

Your new `main` method will *call* these other methods depending on whether there are 1, 2 or 3 integer numbers to deal with. For example you could do this:

```java
    // assuming n1, n2, n3 are the input numbers
    if (n1 > 0 && n2 > 0 && n3 > 0) {
        printMax(n1, n2, n3);
    }
```