SEHH2042 Computer Programming

Tutorial 4 – Repetition statements

Q1. Write a program **GuessNum.cpp** that asks user to guess an integer in 1 – 100. If the guess is smaller than the answer, it shows "Too low. Try again." If the guess is larger than the answer, it shows "Too high. Try again." If the guess is correct, it shows "Excellent! Correct guess." You can use any value (1 – 100) for the answer. Use **while loop** to ask for user's input until the guess is correct.

Sample result (e.g. answer is 31):

```
I have a number between 1 and 100.
Can you guess my number?
Guess: 14
Too low. Try again.
Guess: 55
Too high. Try again.
Guess: 31
Excellent! Correct guess.
```

Q2. Write a program **Average.cpp** that allows user to input integer values continuously. When the user input 0, the program ends and displays the average of all **positive** numbers (value 0 is not counted). Use **do-while loop** in your program and display the average in **3 decimal places**. Assume the user must input some positive numbers.

Sample result:

```
Input an integer: 1
Input an integer: -5
Input an integer: 3
Input an integer: 4
Input an integer: -2
Input an integer: 0
Average of all positive values is 2.667
```

Enhanced version: Update your program by removing the assumption. E.g. what if the first input is 0?

Q3. Write a program **Factorial.cpp** that accepts an integer *n* from the user, and displays the value of *n*!. Use **for loop** in your program. Assume the user must input non-negative numbers.

The factorial of a non-negative integer *n* (*n*!) is defined as:

$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$ (for values of *n* greater than 1)
$n! = 1$ (for *n* = 0 or *n* = 1)

Sample result:

```
n: 5
5! = 120
```

Q4. Write a program **SumOdd.cpp** that accepts two integers, *valueA* and *valueB* from the user. It then adds all odd integers from the smaller of *valueA* and *valueB*, to the larger of *valueA* and *valueB* (use a **for loop** to do so). Finally, it prints *valueA*, *valueB* and the summed result on the screen as shown below.

Sample result 1:

```
Please input value A: 11
Please input value B: 7
Sum of odd integers from 7 to 11 is 27
```

Sample result 2:

```
Please input value A: 5
Please input value B: 8
Sum of odd integers from 5 to 8 is 12
```

Q5. Write a program **Power.cpp** that accepts user's input of the values of *base* and *exponent*, then calculates and displays the result of $base^{exponent}$. Assume the exponent is a positive, non-zero integer. What kind of loop should be used?

Sample result 1:

```
Enter the base value: 2
Enter the exponent value: 3
2 to the power 3 is 8
```

Sample result 2:

```
Enter the base value: 1.5
Enter the exponent value: 4
1.5 to the power 4 is 5.0625
```

Enhanced version: Update your program so that it also allows zero and negative integer exponent.

Q6. Write a program **Prime.cpp** that prints all prime numbers within the range 1 to 200. A number is a prime number if it is divisible by 1 and itself only. Note that 1 is not a prime number. The result is printed with 10 numbers per row and with 5-character width for each number.

Hint: You need a counter variable to count how many prime numbers are printed so far, then determine whether a new line should be inserted.

Sample result:

```
    2    3    5    7   11   13   17   19   23   29
   31   37   41   43   47   53   59   61   67   71
   73   79   83   89   97  101  103  107  109  113
  127  131  137  139  149  151  157  163  167  173
  179  181  191  193  197  199
```

Q7. Write a program **CalcPI.cpp** which calculates the value of PI using the formula below. The result is displayed with 10-character width for the first column and 20-character width for the second column. Use 15 decimal places for the value of PI.

$$PI = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - \ldots\ldots$$

Examples:

Use 5 terms: $PI = 4/1 - 4/3 + 4/5 - 4/7 + 4/9$

Use 10 terms: $PI = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - 4/15 + 4/17 - 4/19$

Sample result:

```
     Terms          Value of PI
     -----          -----------
        10   3.041839618929403
       100   3.131592903558554
      1000   3.140592653839794
     10000   3.141492653590034
    100000   3.141582653589720
```