

# 1 Introduction

These exercises are all dealing with C++ decisions and repetition structures. Each exercise should be its own separate project. When you are ready to submit your work create a folder called Homework02 in that folder have separate folders for each project, one folder per project. Put all the code files needed for that project in its respective folder. Do not include the files that the IDE creates, I just want the code files. Zip the entire Homework02 folder up into a single zip file and submit it.

**Remember to follow the coding and documentation standards for the class listed below and on the MyClasses page.**

# 2 Exercises

1. Write a program that will continually roll 5 dice (all 6-sided) at a time and return the number of rolls needed for all the dice to be the same value. The program should return the number of rolls needed for all the dice to be equal. The main should set the seed of the random number generator to the time of the system clock. Three runs are below.

```
The number of rolls to get all 5 dice equal was 722.
```

```
The number of rolls to get all 5 dice equal was 204.
```

```
The number of rolls to get all 5 dice equal was 3038.
```

2. Write a program that will continually roll a die until a given run length of the same roll happens. Specifically, have the user input the desired roll value 1–6, this is of course a 6-sided die. Also have the user input the run length, 1 or greater. Error check the inputs and reask for inputs if needed. The program should then simulate rolling the 6-sided die until the desired roll comes up the desired number of times consecutively. For example, say the user wants a 4 to come up 3 times in a row. If the simulation is,

```
2 5 4 3 6 4 4 2 1 1 1 3 2 4 5 4 4 4
```

Then the simulation would stop and the count would be 18. The main should set the seed of the random number generator to the time of the system clock. Three runs are below.

```
Input a die roll (1 - 6): 4
Input a run length (>= 1): 3
The number of rolls to get 3 consecutive 4's was 8.
```

```
Input a die roll (1 - 6): 5
Input a run length (>= 1): 10
The number of rolls to get 10 consecutive 5's was 126911171.
```

```
Input a die roll (1 - 6): 7
```

```

Invalid input, please input a number (1 - 6).
Input a die roll (1 - 6): 10
Invalid input, please input a number (1 - 6).
Input a die roll (1 - 6): 1
Input a run length (>= 1): 0
Invalid input, please input a number >= 1.
Input a run length (>= 1): 7
The number of rolls to get 7 consecutive 1's was 800653.

```

3. Write a program that will continually roll a die until a given run length of the same roll happens. This is similar to the exercise above except that the user will not specify the value to tracked. The user will input the run length, 1 or greater. Error check the input and reask for input if needed. The program should then simulate rolling the 6-sided die until the same roll comes up the desired number of times consecutively. For example, say the user wants a run of 4. If the simulation is,

```
2 5 4 3 6 4 4 2 1 1 1 3 2 4 5 4 4 4 2 5 3 3 3 3
```

Then the simulation would stop and the count would be 24. The main should set the seed of the random number generator to the time of the system clock. Three runs are below.

```

Input a run length (>= 1): 5
The number of rolls to get 5 consecutive equal rolls was 3482.

Input a run length (>= 1): 10
The number of rolls to get 10 consecutive equal rolls was 18083797.

Input a run length (>= 1): 7
The number of rolls to get 7 consecutive equal rolls was 17787.

```

4. Write a program that will take one input integer from the user,  $n$ , greater than 1. The program will then generate the  $3n + 1$  sequence, print it to the screen along with the length of the sequence. The  $3n + 1$  sequence is defined as a list of numbers that starts with a positive integer larger than 1. Each number in the sequence is determined by the previous number. If the previous number is even then the next number is the previous divided by 2. If the previous number is odd then the next number is the previous times 3 and add one. We stop when the last number is 1. Have the program display the sequence and count the number of numbers in the sequence. Also do some error checking on the input. You may assume the person types in an integer, if the integer is not greater than 1 have the program display an input error and ask for the number again. Two runs are below.

```

Input an integer (> 1): 7
Sequence: 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Number of integers in list = 17

Input an integer (> 1): -3
Invalid input, please input a number > 1.
Input an integer (> 1): 0
Invalid input, please input a number > 1.
Input an integer (> 1): 17
Sequence: 17 52 26 13 40 20 10 5 16 8 4 2 1
Number of integers in list = 13

```

5. Write a program that will take a double from the user and a tolerance, call them  $n$  and  $tol$  respectively. Have it declare a double called  $r$  initialized to 1. Then it is to continually replace  $r$  with the value of

$$\frac{1}{2} \left( r + \frac{n}{r} \right)$$

until two consecutive values of  $r$  are within the tolerance of each other. Finally have the function return the result. Also do some error checking on the input. You may assume the person types in a decimal number of each input, if the number is not greater than or equal to 0 have the program display an input error and ask for the number again. If the tolerance is not greater than 0 have the program display an input error and ask for the tolerance again. Two runs are below.

```
Input a number (>= 0): 2
Input a tolerance: 0.0000000001
1.41421356237309

Input a number (>= 0): -4
Invalid input, please input a number >= 0.
Input a number (>= 0): 7.5
Input a tolerance (> 0): 0
Invalid input, please input a number > 0.
Input a tolerance (> 0): 0.000001
2.73861278752583
```

6. Write a program that does a simple Caesar cipher on an input message.

In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.<sup>1</sup>

So with a shift of 3 the letter A would be replaced with D, B with E, C with F, and so on. We would also wrap around at the end of the alphabet. So W would be replaced with Z, X with A, Y with B, and Z with C. To decrypt you shift in the other direction. So again with a shift of 3, we would decrypt D with A, E with B and so on. Write a program that will take in a message from the user, a shift between 0 and 25, and either E or D to indicate encryption mode or decryption mode. Then have the program either encrypt or decrypt the message with the desired shift. Error check the range of the shift and error check the input of E or D but allow the user to input either uppercase or lowercase on the mode. The program should take the message and convert it to all uppercase and remove any characters that are not alphabetic characters, then encrypt or decrypt the message. One hint here is if you take each character of the message and find its distance from A, add the shift modulo 26 for encryption and add that onto a character A then you have the encrypted character. Same goes for decryption except that you subtract the shift. Several runs are below.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Caesar\\_cipher](https://en.wikipedia.org/wiki/Caesar_cipher)

```
Input the message: attack at dawn
Input the shift (0 - 25): 3
Input E to encrypt and D to decrypt: e
Encrypted Message: DWWDFNDWGDZQ

Input the message: DWWDFNDWGDZQ
Input the shift (0 - 25): 3
Input E to encrypt and D to decrypt: d
Decrypted Message: ATTACKATDAWN

Input the message: time for coffee
Input the shift (0 - 25): 21
Input E to encrypt and D to decrypt: e
Encrypted Message: ODHZAJMXJAAZZ

Input the message: ODHZAJMXJAAZZ
Input the shift (0 - 25): 21
Input E to encrypt and D to decrypt: d
Decrypted Message: TIMEFORCOFFEE

Input the message: attack at dawn
Input the shift (0 - 25): 44
Invalid input, please input a number (0 - 25).
Input the shift (0 - 25): -3
Invalid input, please input a number (0 - 25).
Input the shift (0 - 25): 3
Input E to encrypt and D to decrypt: t
Invalid input, please input E or D.
Input E to encrypt and D to decrypt: u
Invalid input, please input E or D.
Input E to encrypt and D to decrypt: y
Invalid input, please input E or D.
Input E to encrypt and D to decrypt: e
Encrypted Message: DWWDFNDWGDZQ
```

### 3 Coding Standards

- Code
  - AStyle alignment is correct. I will accept either ANSI style alignment which is the default for Code::Blocks or Java style alignment which is the default for Eclipse C++.
  - No global variables.
  - All variables must be initialized.
  - Constants are to be all uppercase.
  - Prefer using return values over output parameters: they improve readability, and often provide the same or better performance. If output-only parameters are used, they should appear after input parameters.
  - Prefer small and focused functions.
  - Default arguments are allowed when the default is guaranteed to always have the same value.
- Commenting

- Your name, date, and description comments at the top of each program.
- Comments before each loop on its purpose.
- Comments before each decision on its purpose.
- All functions must have a description to what the function does.

## 4 Submit Your Work

As before, submit each of the code files through the file uploads to the MyClasses system.

When you are ready to submit your work create a folder called `Homework02` in that folder have separate folders for each project, one folder per project. Put all the code files needed for that project in its respective folder. Do not include the files that the IDE creates, I just want the code files. Zip the entire `Homework02` folder up into a single zip file and submit it.