Lab09

1. Lab Topics

This lab covers different topics, but more emphasis is placed on templated functions, function documentation, pass by reference, default arguments and constant parameter. random number generation, pass by reference.

2. Documentation

Proper document of each implemented function should be done along with the function prototype. Refer to the following example:

Function description: Describe what the function does. **Not** how the function works.
E.g. "This function calculates and returns the volume of a sphere given the radius of the sphere"
Pre-condition: What must be true about your arguments for the post condition to be true.
E.g. "The first argument is greater than 0"
Post-condition: States what is true after the function successfully runs.
E.g. "The sum of the two arguments is calculated and returned"

3. Implementing Functions

Step 1: Implement these new functions.

1. Implement a **templatedSwap()** function that employs pass by reference and accepts two arguments. For example, consider having two variables, x and y, initially set to 1 and 2. When the swap function is invoked with these variables as arguments, it exchanges the value of x with y, resulting in the value 1 moving from x to y, while the original value of y goes to x. The strength of this function lies in its templated nature, enabling it to work with variables of various data types.
2. Implement a **validateInput()** function that takes in the argument a **constant** parameters (string preferably) and returns a Boolean value if the input is valid or not. To check if an input is valid or not, some special character must not be anywhere in the string. These characters are **%**, **$**, **@.** Also ensure the length of the argument is below 10 characters.

Step 2: Implement the following in the main function.

1. Create two string variables that take two inputs from the user into these variables, and then validate them using the **validateInput** function. Make sure to ask user for another input if it is invalid.
2. After validating, output those two strings on the same line, swap them using your **templatedSwap** function, and then output them again on the same line.

Example output 1:

```
Enter the first string: Jum
Enter the second string: Bo
Valid input strings: Jum Bo
After swapping: Bo Jum
```

Example output 2; Wrong Input:

```
Enter the first string (valid if it doesn't contain %, $, @ and
length < 50): qwertyuiopasdfghjkl
Enter the first string (valid if it doesn't contain %, $, @ and
length < 50): $werfe
Enter the first string (valid if it doesn't contain %, $, @ and
length < 50): Jump
Enter the second string (valid if it doesn't contain %, $, @ and
length < 50): Pike
Original Strings: Jump Pike
Swapped Strings: Pike Jump
```

Step 3: Bug Smasher Update.

To make the bug smasher game more interests, generate two random numbers between 1 and 10 as the positions of two bug. After 3 attempts failed attempts to smash the first one, swap the position of the bugs, such that now, you are trying to guess the position of the second bug and after another 3 failed attempts, try it for the first one. Look at the example below. For illustration purposes, I have added a cout such that I could "avoid" the position of the bugs such that I don't smash them too easily.

Example Output 3:

```
Bug 1 is at position 4
Bug 2 is at position 6
Welcome to Bug Smasher!
Enter your guess (1-10): 1
The bug is to the right of your guess. Try a higher number.
Enter your guess (1-10): 2
The bug is to the right of your guess. Try a higher number.
Enter your guess (1-10): 3
The bug is to the right of your guess. Try a higher number.
The bugs have swapped positions! Bug 2 is laughing at you now.
Enter your guess (1-10): 4
The bug is to the right of your guess. Try a higher number.
Enter your guess (1-10): 5
The bug is to the right of your guess. Try a higher number.
Enter your guess (1-10): 7
The bug is to the left of your guess. Try a lower number.
The bugs have swapped positions! Bug 1 is laughing at you now.
Enter your guess (1-10): 7
The bug is to the left of your guess. Try a lower number.
Enter your guess (1-10): 6
The bug is to the left of your guess. Try a lower number.
Enter your guess (1-10): 5
The bug is to the left of your guess. Try a lower number.
```

```
The bugs have swapped positions! Bug 2 is laughing at you now.
Enter your guess (1-10): 7
The bug is to the left of your guess. Try a lower number.
Enter your guess (1-10): 4
The bug is to the right of your guess. Try a higher number.
Enter your guess (1-10): 6
Congratulations! You smashed the bug in 3 attempts.
Do you want to play again? (yes or no):
```

Step 4: Integration
Edit your code in lab08 to integrate it with what you've done such that the program now collects the first and last name of the users. Use the validateInput function to confirm if each input is valid, else continue to ask the user for the right input. Use the templatedSwap function to swap the names.

General Hints:

1. Remember to initialize your variables appropriately and at the right location.
2. Implement input validation to ensure user input is within valid ranges.
3. Keep the code organized and use clear variable names.
4. Make sure the program gracefully exits when the user chooses to do so.
5. Feel free to copy the codes from your previous lab.

BONUS (10 points):
Implement a 3D matrix for the smash the bug program such that the user has to enter 3 numbers to represent the X, Y and Z coordinate. The bug will also change its position after 3 failed attempts.
Note: Little/No help will be provided for this bonus. Students are expected to research about it and learn how it is being used.

4. How to get full marks

To get a 100% on this lab your code should:

1. **Proper Documentation of ALL functions**
2. Use good variable names such that one can easily understand a variable's purpose just by looking at the name.
3. The program needs to be intuitive (e.g., display proper messages while you are taking user input or printing the result)
4. Follow all good coding conventions such as proper indentation.
5. Adhere to all coding standards outlined in lab2.

6. Follow the instructions of cloning, making dir, and submitting your code to git as previously discussed in lab01 and lab02
7. Comment your code properly (do not write comments for things that are obvious)
8. Push your most recent code in git and submit through canvas as well. The canvas submission should include following files:
    a. **The cpp file downloaded from git.**
    b. **At least 2 Image files**
        i. **Screenshot of the right result.**
        ii. **Screenshot of Step 3 output similar to what I have in the example output.**