Lab 1 Activities [30 Marks]

An Autograded Evaluation

For the lab portion submitted for autograding, do not use `printf()` or `scanf()`.

IMPORTANT: Your submission must compile without syntactic error to receive grades. Non-compilable solutions will not be graded.

About Lab 1

- Use the following link to accept the lab invitation: https://classroom.github.com/a/QLAbMhPZ
- Lab 1 is due on **Friday September 27, 2024, at 11:59 pm** on Github.

Lab Question 1 – Control Structures [6 marks]

Complete the following three functions that return the integer sum that holds the sum of all the numbers that are multiples of the integer input **`num`** between 1 and 1000 inclusively using for-loop, while-loop, and do-while loop.

- `int Q1_for(int num)`
- `int Q1_while(int num)`
- `int Q1_dowhile(int num)`

In you Git submission, attach the flow chart designs of these three loops to showcase your understanding of each.

This can be done by either placing the word or PDF document containing your flow charts in your local Lab1 folder and then adding, committing and pushing that file to Github along with the rest of your Lab1 work or you can directly upload your flow chart diagram document on GitHub.com into your remote repository. However, if you take the latter approach, you must upload directly to Github only after you have submitted the rest of your lab through VS Code (as discussed in the Lab0 manual).

Marking Scheme

- Code Implementation
  - **[1.5 mark, 0.5 mark per version]** Correct loop implementations for all three specified loops.
- Code Behaviour, Analysis, and Test Plans
  - **[1.5 mark, 0.5 mark per version]** Correct Program Behaviour – passing all test cases.
  - **[1.5 mark, 0.5 mark per version]** Additional Test Cases – providing at least **3 more** custom test cases of your choice **per loop version**.
  - **[1.5 mark, 0.5 mark per version]** Correct Flow Chart for all three loop versions.

Lab Question 2 – Floating Point Number Processing [6 marks]

Complete the function `int Q2_FPN(float Q2_input, float Q2_threshold)` that takes a floating point number input (`Q2_input`) and a floating point number threshold value (`Q2_threshold`), such that the function:

- Returns `0` if input falls in the range of `[-2*threshold, -1*threshold)`
- Returns `1` if input falls in the range of `[-1*threshold, 0)`
- Returns `2` if input falls in the range of `[0, 1*threshold)`
- Returns `3` if input falls in the range of `[1*threshold, 2*threshold]`
- Returns `-999` otherwise.

Attach the flow chart of your implemented logic to showcase your understanding. Refer to the discussion in Q1 about how to upload the flow chart.

Note: `[]` are inclusive brackets indicating the bracketed numbers are included in the range. `()` are exclusive brackets indicating the bracketed numbers are NOT included in the range. For example, `(5, 10]` means all numbers in the range between `5` and `10` – and includes `10` but not `5`.

Marking Scheme

- Code Implementation
  - **[2 marks]** Correct choice of control structure (if-else, NOT switch-case)
  - **[2 marks]** Correct conditional statements to ensure inclusiveness / exclusiveness of ranges
  - **[0.5 mark]** Correctly handling the wildcard case
- Code Behaviour, Analysis, and Test Plans
  - **[0.5 mark]** Correct Program Behaviour – passing all test cases
  - **[0.5 mark]** Additional Test Cases - providing at least **3 more** custom test cases of your choice
  - **[0.5 mark]** Correct Flow Chart

Lab Question 3 – Perfect Number [9 marks]

A positive integer number is said to be a perfect number if its positive factors, including 1 but not the number itself, sum to the number. For example, 6 is a perfect number because 6 = 1 + 2 + 3.

Complete the function `int Q3(int Q3_input, int perfect[])` that determines all perfect numbers smaller than or equal to some integer `Q3_input` (Q3_input > 1).

- The array `perfect[]` should hold all the perfect numbers you find.
- The function should return the total count of perfect numbers you found in the entire calculation process.

Hint: Assuming two positive integers x and y, then x is a **factor** of y if the remainder of y divided by x is 0. For example, 5 is a factor of 15 because 15 can be evenly divided by 5 with remainder of 0.

You may optionally provide a flow chart to showcase your understanding for a 0.5 mark bonus.

Marking Scheme

- Code Implementation
  - **[3 marks]** Correct logic for determining whether a number is perfect
  - **[2 marks]** Correct logic for recording all the perfect numbers found within the specified range
  - **[2 marks]** Correct logic for tracking the total count of the perfect numbers found
- Code Behaviour, Analysis, and Test Plans
  - **[1 mark]** Correct Program Behaviour – passing all test cases
  - **[1 mark]** Additional Test Cases - providing at least 1 more custom test cases of your choice
  - **BONUS: [0.5 mark] Flow Chart Analysis**

Lab Question 4 – Bubble Sort [9 Marks]

Complete the function `int Q4_Bubble(int array[], int size)` that takes in an array containing random integers, and the array size. You will implement the famous simple sorting algorithm, "*Bubble Sort*", to sort the incoming array contents into ***Ascending Order*** using the provided pseudocode below:

1. Given an array and its size, visit every single element in the array up to size-2
   (i.e., do not visit the last element)
2. For every visited element (current element), check its subsequent element (next element).
   If the next element is smaller, swap the current element and the next element.
3. Continue until reaching size-2 element. This is considered One Pass, increment Pass Count by one.
4. Repeat 1-3 until encountering a pass in which no swapping was done.
5. Sorting Completed, Return Pass Count.

To help you understand *Bubble Sort*, see the graphical representation on the next page.

Hint: You may use the marking scheme to help you develop an incremental design plan.

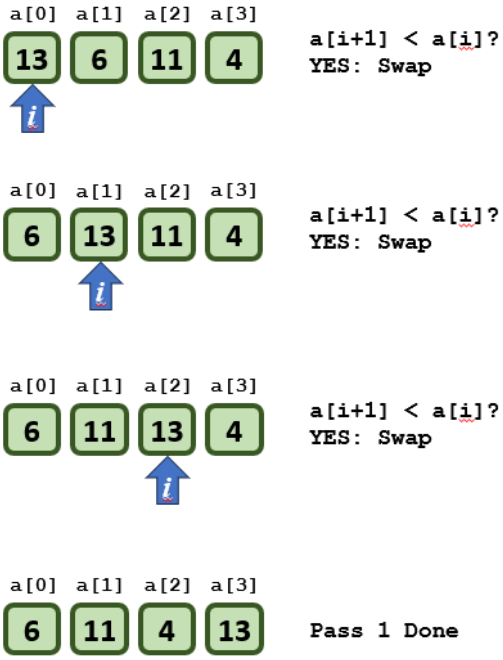
You may optionally provide a flow chart to showcase your understanding for a 0.5 mark bonus.

Marking Scheme

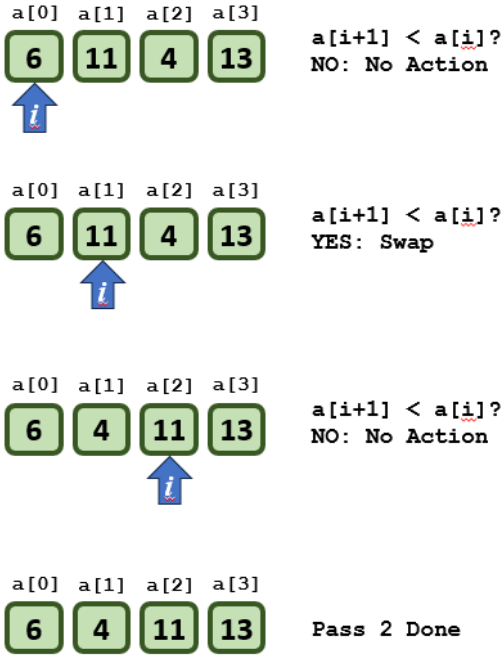
- Code Implementation
  - **[2 marks]** Correct implementation of element iteration logic (up to second last element)
  - **[2 marks]** Correct implementation of swapping logic
  - **[2 marks]** Correct logic of stopping condition
  - **[1 mark]** Correct logic for tracking number of passes
- Code Behaviour, Analysis, and Test Plans
  - **[1 mark]** Correct Program Behaviour – passing all test cases
  - **[1 mark]** Additional Test Cases - providing at least 1 more custom test case of your choice
  - **BONUS: [0.5 mark] Flow Chart Analysis**

Graphical Representation of Bubble Sort

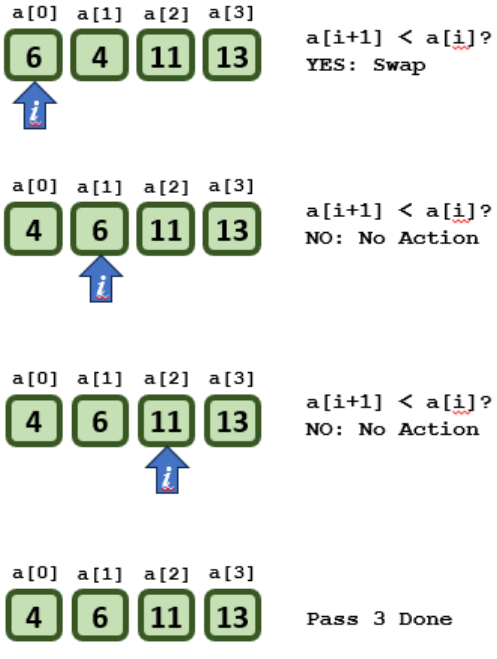
Pass 1



Pass 2



Pass 3



Pass 4