Data Structures and Algorithms LAB – Spring 2026

(BS-IT-F24 Morning & Afternoon)

Lab # 5 (Online Lab)

Submission Deadline: ***Friday, February 27, 2026. (till 11:00 PM)***

Instructions:

- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating or usage of AI tools will result in an **"F"** grade in this course and lab.
- Attempt the following tasks exactly **in the given order**.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.
- Late submissions will NOT be accepted, whatever your excuse may be.

Submission Procedure:

i. There are 2 tasks in this lab. You are required to submit C++ programs for both tasks.

ii. Create 2 empty folders. The folder names MUST be **Task-1**, and **Task-2**. Put all of the ***.CPP, .H, and input text files (if applicable)*** of each task in its appropriate folder. Do NOT include any other files in your submission, otherwise your submission will NOT be graded. Don't forget to mention your Name, Roll Number and Section in comments at the top of each CPP file.

iii. Now, put all the folders created in the previous step into another folder.

iv. Now, compress the folder (that you created in previous step) in **.RAR** or **.ZIP** format. Then rename the RAR or ZIP file ***exactly*** according to the following format:

`Mor-Lab5-BITF24M123` (for Morning section) OR

`Aft-Lab5-BITF24A456` (for Afternoon section),

where the text shown in **BLUE** should be your **complete Roll Number**.

v. Carefully check your **.RAR** or **.ZIP** file to ensure that all above instructions have been followed.

vi. Finally, submit the *single* **.RAR** or **.ZIP** file through **Google Classroom**. Make sure that you press the **SUBMIT** button after uploading your file.

Note: *If you do not follow the above submission procedure, your Lab will NOT be graded and you will be awarded ZERO marks.*

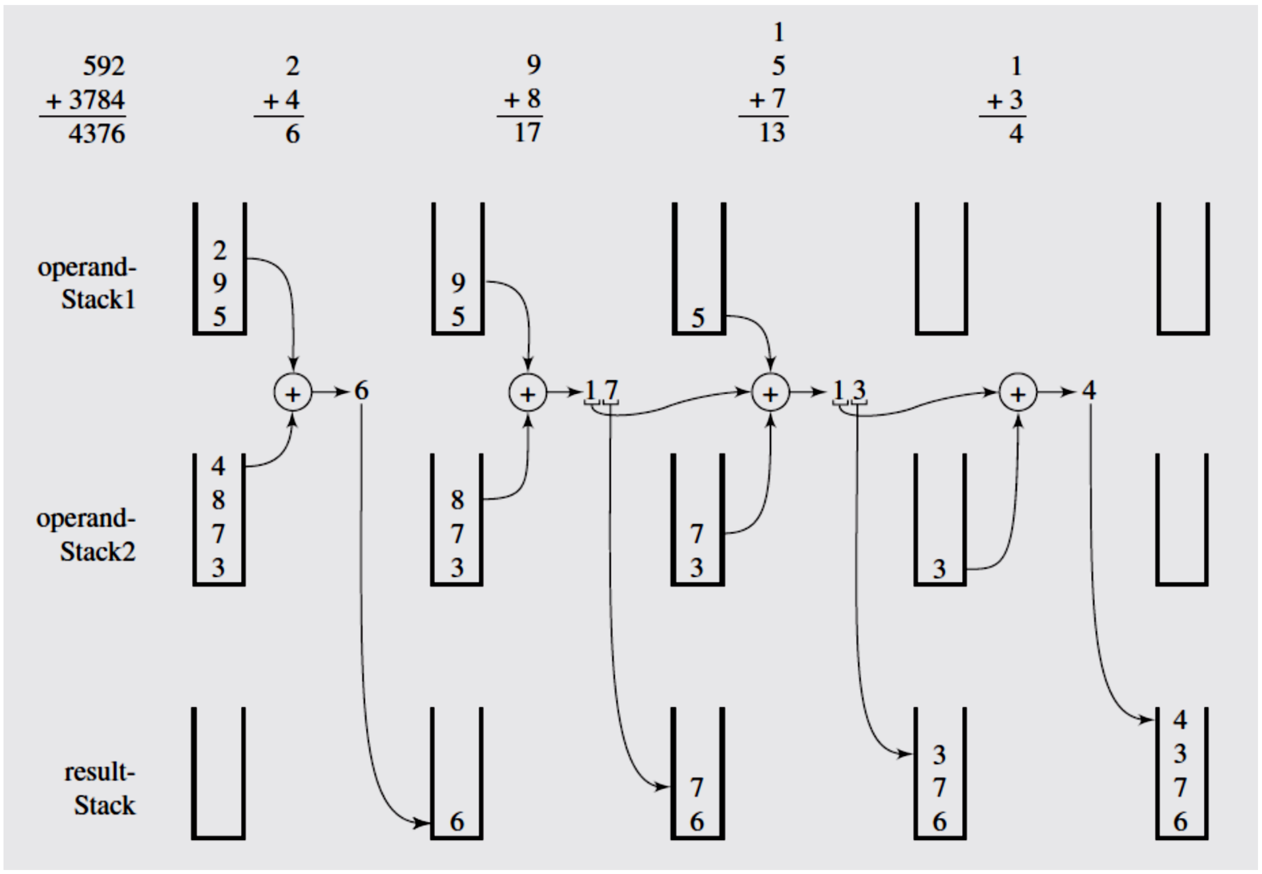
Task # 1

Here is the description of the problem of ***Adding two very large numbers*** *as described in Section 4.1 of the book "Data Structure and Algorithms in C++" (4th Edition) by Adam Drozdek:*

The largest magnitude of integers is limited, so we are not able to add **18,274,364,583,929,273,748,459,595,684,373** and **8,129,498,165,026,350,236**, because integer variables cannot hold such large values, let alone their sum. The problem can be solved if we treat these numbers as strings of characters, store the numbers (digits) corresponding to these characters on two stacks, and then perform addition by popping numbers (digits) from the stacks. The pseudo-code for this algorithm is as follows:

```
Algorithm addingLargeNumbers()
    a. create 3 empty stacks of integers (stack1, stack2, and resultStack)
    b. read the digits of the first number (as characters) and store the
       numbers/digits corresponding to them on stack1
    c. read the digits of the second number (as characters) and store the
       numbers/digits corresponding to them on stack2
    d. integer carry = 0
    e. while at least one stack is not empty
         i. pop a number from each nonempty stack and add them to carry
        ii. push the unit part of the sum on the resultStack
       iii. store carry (the tens part) in the variable carry
    f. push carry on the resultStack if it is not zero
    g. pop numbers from the resultStack and display them
```

The following figure (taken from Drozdek's book) shows an example of the application of this algorithm. In this example, the two numbers **592** and **3784** are being added.



The step-by-step working of the above example is explained below:

1. Numbers corresponding to digits composing the first number (i.e. **592**) are pushed onto **operandStack1**, and numbers corresponding to the digits of the second number (i.e. **3784**) are pushed onto **operandStack2**. Note the order of digits on the stacks.
2. Numbers **2** and **4** are popped from the stacks, and the result, **6**, is pushed onto **resultStack**.
3. Numbers **9** and **8** are popped from the stacks, and the unit part of their sum, **7**, is pushed onto **resultStack**; the tens part of the result, number **1**, is retained as a carry in the variable **carry** for subsequent addition.
4. Numbers **5** and **7** are popped from the stacks, added to the **carry**, and the unit part of the result, **3**, is pushed onto **resultStack**, and the carry, **1**, becomes a value of the variable **carry**.
5. One stack is empty, so a number is popped from the nonempty stack, added to **carry**, and the result is stored on **resultStack**.
6. Both operand stacks are empty, so the numbers from **resultStack** are popped and printed as the final result.

You are required to write a **C++ program** which implements the above algorithm for adding two very large numbers. Here are a few instructions that you need to keep in mind:

1. Implement and use the **`Stack`** class (for storing **integers**) that we have seen in the lecture.
2. Your program should take two numbers from the user and store them as two null-terminated **c-strings**. You can assume that the maximum number of digits in a number is **40**.
3. After taking input, your program should determine the sum of these two large numbers using the above-mentioned algorithm (***`addingLargeNumbers`***) and display the sum on screen.

Two sample runs of your program may produce the following output (Text shown in **Red** is entered by the user):

Sample Run # 1

```
Enter 1st number: 123456789123456789123456789
Enter 2nd number: 123454321123454321

Sum of the two numbers is: 123456789246911110246911110
```

Sample Run # 2

```
Enter 1st number: 123456789123456789123456789
Enter 2nd number: 987654321987654321987654321

Sum of the two numbers is: 1111111111111111111111111110
```

Task # 2

Consider the following declaration:

```
struct LetterInfo {
        char letter;        // A letter of the English alphabet
        int freq;           // Frequency of the above letter
};
```

Write a C++ program which should, first of all, read an input text file and determine the frequency of each alphabet in the input file. Note that:

- You are required to count the frequencies of alphabet characters ONLY (i.e., from **A** to **Z**). Punctuation marks, numbers (digits), and other special characters are NOT to be counted.
- Lower-case and Upper-case alphabets should be counted as same.
- In order to read the contents of the input file, you are only allowed to declare/use a single **char** variable. You are NOT allowed to read/store the contents of the input file using **string**(s) or **c-string**(s).

After storing the frequencies of all alphabets in an array of type `LetterInfo`, your program should sort this array (using **Insertion sort**) in **decreasing order** of letter **frequency**. At the end, this sorted array of type `LetterInfo` should be displayed on screen in a **neat and readable way**.

VERY IMPORTANT

In the next Lab, you will need some or all of the functions from Today's Lab. So, make sure that you have the working implementation of **ALL** the functions of Today's Lab, when you come to the next Lab.