1 Assignment 3 Introduction

This assignment is due *before* 11:59pm on the listed date. Which means, submissions made *on or after* 11:59pm will be counted as a late submissions. Late submissions *before* 11:59pm on the following day will receive a 25% point deduction as penalty. Submissions made *on or after* 11:59pm on the same day will *not* be graded. We strongly recommend completing the assignment before then.

Assignments due on Mondays generally involve materials covered in lectures from the previous week, whereas assignments due on Thursdays involve materials covered in lectures from the running week. So be sure to watch the lectures and go over the reading materials before attempting the assignments.

Good luck!

2 Illuminati Generator (40 points)

You are tasked with creating a scalable Illuminati image generator. Your program should take the height of the desired image, and output the image in the following manner:

Input:

```
5
```

Output:

```
    *
   * *
  *<O>*
 * * * *
* * * * *
```

The eye is made with the less than sign (‘<’), the uppercase-case letter ‘O’, and the greater than sign (‘>’). In this scenario, the eye is located at the center of row 3, where ‘O’ replaced the middle ‘*’, while ‘<’ and ‘>’ replaced the two adjacent whitespaces on both sides of the middle ‘*’. However, this is not always the case. Consider the following example:

Input:

```
7
```

Output:

```
      *
     * *
    * * *
   * <O> *
  * * * * *
 * * * * * *
* * * * * * *
```

In order for the eye to be at the center, ‘<’ and ‘>’ replaced two ‘*’s, while ‘O’ replaced the whitespace between them. Now what if the provided height is an even number?

Input:

```
6
```

Output:

```
    *
   * *
  * * *
 * <O> *
* * * * *
* * * * * *
```

In the above example, there are three rows above the row with the eye, and two rows below it.

```
void GenImg(int height) {
    //Your code here
}

int main() {
    int height{0};
    std::cin >> height;
    GenImg(height);
    return 0;
}
```

General Instructions:

- When writing your program, you may only use concepts you have learned in the course thus far.
- Your program should compile and run. You can assume the input will always be an integer that is greater than or equal to `5`.
- The formatting must match exactly as shown in the example above, including whitespaces.

Hints and Suggestions:

- You will need to have a good understanding of loops and conditional statements to solve this problem, and will have to make use of both nested for loops and if-else statements.
- This is not a string manipulation problem, and you should not use `string` or `char` other than for the purpose of printing to terminal.
- It might be a good approach to first write a function that generates the image only using '`*`'s and whitespaces (i.e., without the eye), and then figure out how to add the eye.

3 Location Arithmatic Converter (60 points)

John Napier was a Scottish mathematician who lived in the late 16th and early 17th centuries. He is known for a number of mathematical inventions, one of which is termed location arithmetic.

Location arithmetic is a way to represent numbers as binary values, using a notation that is not positional, but representational. Napier used letters to represent powers of two. The position of these letters is unimportant, allowing for multiple representations of the same number. For example, in location arithmetic:

$$a = 1, b = 2, c = 4, d = 8, e = 16, f = 32, \ldots$$

For example:

$$acf = 1 + 4 + 32 = 37$$

And:

$$caf = 4 + 1 + 32 = 37$$

For easier reading, the letters are typically sorted. Napier allowed for redundant occurrences of letters, though he acknowledged that there is a normal form that has no repeatations. He described this as the extended (repeats allowed in any order) vs. abbreviated (no repeats allowed, and sorted) form. To create the abbreviated form, any pair of letters can be reduced to a single occurrence of the next 'higher letter'.

Consider the following example:

$$cabdbc = 4 + 1 + 2 + 8 + 2 + 4 = 21$$

Now, *cabdbc* can be sorted, and then simplified in the following manner:

$$cabdbc \rightarrow abbccd \rightarrow acccd \rightarrow acdd \rightarrow ace = 1 + 4 + 16 = 21$$

Your task is to write functions that can convert back and forth between location and decimal representations, as well as some support functions for the process. The following function converts a location arithmatic string to an integer:

```
std::int64_t LocToDec(std::string const & loc)
```

The following function takes a location string and reduces it to its abbreviated form:

```
std::string Abbreviate(std::string const & loc)
```

We want you to experiment with string manipulation, so you may not convert it to an integer first. You must do the abbreviation directly.

The following function converts an integer to an abbreviated location string:

```
std::string DecToLoc(std::int64_t dec)
```

And, the following function takes two location strings, adds them, and provides the integer result:

```
std::int64_t AddLoc(std::string const & loc1, std::string const & loc2)
```

Write a `main()` function that demonstrates the above functions. The general workflow of your `main()` is a follows:

- Prompt for two elements: a location string, and an integer.
- Using `LocToDec()`, print the location string and the resulting integer.
- Using `Abbreviate()`, print the location string and its reduced form.
- Using `DecToLoc()`, print the integer and its location string.
- Prompt for two more location strings.
- Using `AddLoc()`, add the two location strings and print the result.

Following is an example interaction with the final program, where line 1, 3, and 8 are prompts, line 2, 4, 9, and 10 are inputs, and line 5, 6, 7, and 11 are outputs:

```
Give me a location string:
abbccd
Give me an integer:
37
abbccd to dec: 21
abbccd abbreviated: ace
37 to loc: acf
Give me two more location strings:
abc
bcd
Sum of abc and bcd is: 21
```

Note that we have not covered sorting yet. However, it is fairly straightforward, and the following code is provided for you to use as reference:

```
#include <iostream>
#include <string>
#include <algorithm>

int main() {
    std::string my_str = "aebcd";
    std::cout << "my_str before sorting: " << my_str << std::endl;
    std::sort(my_str.begin(), my_str.end());
    std::cout << "my_str after sorting: " << my_str << std::endl;
}
```

The above code outputs the following:

```
my_str before sorting: aebcd
my_str after sorting: abcde
```

`std::sort()` sorts the individual elements (instances of type char, in this case) of a collection given two points, `my_str.begin()`, and `my_str.end()`. We will learn more about what these methods do later on in the course. Note that `std::sort()` changes the string in-place.

General Instructions:

- When writing your program, you may only use concepts you have learned in the course thus far, along with materials provided to you in the assignment.
- Your program should compile and run. You can assume the input location strings (for location arithmetic) will always be in lowercase, and the input integers will always be positive.
- You do not need to consider the case of abbreviating multiple '`z`'s.
- The formatting should match exactly as shown in the example above, including white-spaces.

Hints and Suggestions:

- Defining a constant for each letter of the alphabet would be extremely tedious. There is a faster approach to this. The smallest character of a location string is $'a'$, which represents 2 to the 0 power (i.e., $20 = 1$). The difference between any letter and $'a'$ is the power of 2 that letter represents. For example:

  $'a' -' a' = 0$, which in location arithmetic is $2^0 = 1$

  $'c' -' a' = 2$, which in location arithmetic is $2^2 = 4$

  $'h' -' a' = 7$, which in location arithmetic is $2^7 = 128$

- The function `DecToLoc()` consists of creating a 'long string', and then using `Abbreviate()`.
- The function `AddLoc()` consists of concatenating two strings, then using `Abbreviate()`, and then finally using `LocToDec()`.
- Consider using the following functions where necessary:

  `.substr()` - Takes two parameters: a position, and a length. It returns the substring specified within that range. The length defaults to the end of the string (or, if the value is beyond the length of the string, it defaults to the end).

  `.erase()` - Takes two parameters: a position, and a length (just like `.substr()`). It removes all characters within the specified range.

  `.push_back()` - Appends a given char to the end of the string.

  `static_cast<char>` - Casts into a character. Necessary after doing addition or subtraction on a character.

  Indexing via the `[]` operator.

4 Assignment 3 Submission Process

- Create a folder, name it `your_msu_id3`. For example, if your MSU email is `johndoe@msu.edu`, then you should name the folder `johndoe3`.
- For each programming task, create a sub-folder inside your `your_msu_id3` folder, and name it as the number that corresponds to the programming task number. For this assignment, there should be two sub-folders named '2' and '3'.
- Inside each sub-folder, put the `main.cpp` for the appropriate solution.
- Compress/Zip `your_msu_id3` folder and name it `your_msu_id3.zip`. For example, if the name of your folder is `johndoe3`, then you need to create a zip file named `johndoe3.zip`. Zip file guide: `https://copyrightservice.co.uk/reg/creating-zip-files`.
- Submit the zip file through D2L.