For assignments in this course, you will be assessed not only on the functionality of your code but also on its style. Below are the guidelines for C++ code submissions.

Formatting Style:

Use C++ naming conventions for your functions and variables. This means using **camelCase,** starting with a lowercase letter, and capitalizing each new word within the name.

- For example, the variable name dogName is good, but dog_name is not.

Choose meaningful names for your functions and variables.

- For example, numBrightSpots is more helpful and readable than nbs.
- Avoid using single letter variable names, except some commonly acknowledged ones (using i in for loop is definitely OK!)

Use four spaces for indentation in place of tabs.

Put a blank space before and after every operator. For example, the first line below is good but the second line is not:

```
    int b = 3 > x && 4 - 5 < 32;

    int b= 3>x && 4-5<32;
```

For each function, write a comment block according to our design recipe. (See below for guidelines on the content of your comments.)

Put the comment block's closing symbols on their own line.

Each line shouldn't be too long. You should break up long lines as follows:

- If the line contains comma-separated items (such as in a function call), type enter after a comma to continue on the next line like this:

```
   int result = min(1, 2,
                    3, 4);
```

- Triple-quoted strings (if applicable) can span multiple lines by typing enter anywhere within the string:

```
 char* str = R"(
   This is
```

```
a multi-line
string.
)";
```

- For other situations, you can use parentheses to signal that the statement continues on the next line:

```
int myVeryLongVariable = (1000000000000000000000000000 + 10 + 9 + 8 + 7
                          + 6 + 5
                          + 4 + 3 + 2 + 1);
```

Within a comment block, put a blank line between the description and the examples.

Put a blank line between the comment block and the function body.

When writing **structs**, pay attention to the formatting of the struct, both during declaration and use.

Try to use a struct for only passive items that carry data, everything else is a class.

Structs should have all public fields.

Structs should have simple and clear naming conventions and proper indentation and brace placement.

Try to declare structs like the below, with an element in each line.

```
C/C++
struct {          // Structure declaration
 int myNum;       // Member (int variable)
 string myString; // Member (string variable)
} myStructure;    // Structure variable
```

Make sure a block of code enclosed by curly braces are always ended with a semicolon.

Programming Style:

Do not compare a Boolean expression to true or false.

- For example, the first is good, and the second is not:

```
if (a && b) {
    return 100;
}

if ((a && b) == true) {
    return 100;
}
```

Replace if statements of this form:

```
if (x > 100) {
    return true;
} else {
    return false;
}
```

with a single-line statement like this:

```
return x > 100;
```

Replace if statements of this form:

```
if (x > 100) {
    n = n;
} else {
    n = n + 1;
}
```

with a single if statement like this:

```
if (x <= 100) {
    n = n + 1;
}
```

Each function body should contain 20 or fewer lines of code, including blank lines. (This 20 lines does not include the comment block.) If a function body is too long, break the function into helper functions and call those instead. Do not make it shorter by deleting blank lines or otherwise sacrificing readability!

Comments:

For complicated bits of code inside functions, please add an internal comment to explain how the code works. Do not over-comment: assume the reader of the code is a proficient C++ programmer. If you have at most one comment per logical chunk of code (whether that is a small function or a coherent piece of a larger function), you're on track.

Use comments to explain tricky spots in the code.

Use comments to explain the meaning of a variable if it is subtle. A good variable name can go a long way.

- For example, "studentsNum" tells you more than just "num." But sometimes, the meaning is subtle enough that the variable name can't fully convey it (without being ridiculously long!). For instance, you might have a variable called "next" and a comment that says, "The index within s1 from which to search for the next occurrence of s2."

Use comments to explain what you know is true at key moments.

- For example, after a while, the loop has ended, and you know that the loop condition has failed. There are often crucial implications from this that require some reasoning to see. Write those down!

Use comments to summarize chunks of code.

- For example, if 5 lines of code as a whole accomplish some task, it can be helpful to have a short comment summarizing that task. Of course, if you use helper functions well, this won't be necessary terribly often.

Do not write comments that simply rephrase a line of code in English.

- For example, "Increase n by one" does not add anything that "n = n + 1" doesn't already say.

Documentations:

Follow the design recipe for writing functions. This means that in addition to the header and body, you must include a comment block with an example, type contract, and description. Follow these rules when writing the description portion of the comment block:

Describe precisely what the function does.

Do not reveal how the function does it.

Make the purpose of every parameter clear. Use the name of every parameter.

For functions that return values, be clear on what the return value represents.

Explain any conditions that the function assumes are true. These conditions should not refer to type requirements because the type contract already covers those. However, sometimes, there are conditions not covered by types, and you should include those. For example, if a function requires parameters x and y to be even, include x and y are both even.

Be concise and grammatically correct.

Write the description as a command (e.g., Return the first ...) rather than a statement (e.g., Returns the first ...)

Submission Template:

All the submissions must include the following section at the top:

```
// NAME: [Your Name]
// ID: [Your Student ID]
// DATE: YYYY-MM-DD
// DESCRIPTION: [A paragraph explaining what the code does.]
```

And the following at the end:

```
int main() {
    // Call other functions.
    return 0;
}
```