3 Basics

3.1 Preamble

The exercises below are intended to provide a means to test your understanding of the programming-related material covered in the course. It is highly recommended that you work through these exercises as you cover the corresponding topics in the video lectures. By doing this, you will greatly strengthen your understanding of the material in these video lectures, which will greatly reduce the amount of pain and suffering required to complete the programming assignments in the course. In exercises that require the building (i.e., compiling and linking) of code, the CMake tool should be used for this purpose. For additional information about these exercises, refer to Section 1 of this document.

3.2 Topics Covered

These exercises cover language basics, including such things as: objects, types, values, control-flow constructs, functions, parameter passing, pointers, const correctness, default arguments, input/output, and output formatting.

3.3 Exercises

1. **[basic input/output, conditional execution]**
   Write a program that:
   (a) prints to standard output a message prompting the user to enter two real numbers;
   (b) reads two real numbers from standard input;
   (c) prints to standard output whichever one of the following three messages is applicable:
      i. The first number is less than the second.
      ii. The first number is greater than the second.
      iii. The first number is equal to the second.

   For example, running the program would look something like the following:

   ```
   Enter first number: 5.3
   Enter second number: -1.4
   The first number is greater than the second.
   ```

2. **[functions, default arguments, multiple source files, declarations]**
   (a) Write a function called `logarithm` that takes two **`double`** parameters and returns a **`double`**. Let `x` and `b` denote the first and second parameters, respectively. The return value of the function should be the logarithm base `b` of `x`. The second parameter should have a default value of 10.0. [Note: The natural logarithm function in the standard library is called `std::log` and is declared in the header file called `cmath`. The logarithm base b of x can be computed as $\log(x)/\log(b)$.]
   (b) Write a program to test the `logarithm` function developed in part (a). The test program should print the values returned by each of the following:

   ```
   logarithm(10.0)
   logarithm(16.0, 2.0)
   ```

   (c) Split your program into two separate source code (i.e., `.cpp`) files as follows: Place the `logarithm` function in the file `logarithm_func.cpp`; and place the `main` function in the file `logarithm_main.cpp`. (Ensure that the resulting code still compiles and links properly.)
3. **[conditional execution]**
   Download the source code for the `sinc` program contained in the file `sinc.cpp` from the course web site. In this program, rewrite the `sinc` function so that it does not use an **`if`** statement. The function must still handle the case of `sinc(0)` correctly, of course.
4. **[parameter passing]**
   Write a function called `increment` that takes a single **`int`** parameter and does not return any value. When the function is called, it should have the effect of incrementing (by one) the value of the variable passed to it. For example, the following code should print the value 1:

```
int i = 0;
increment(i);
std::cout << i << "\n";
```

Write a program to test the `increment` function.

5. **[pointers, const correctness]**
Note: A NUL-terminated string is a sequence of characters whose end is explicitly marked by the NUL character (i.e., `'\0'`). For example, the NUL-terminated string `"Hi"` is stored in memory as the following sequence of characters: `'H'`, `'i'`, and `'\0'`.
Write a function called `findFirst` that finds the first occurrence of the specified character `c` in the given NUL-terminated string `s`. The function should take the following two parameters (in order):
   (a) a pointer specifying the first character in the NUL-terminated string `s`; and
   (b) the character `c` for which to search.
The function should return a pointer to the first occurrence of the character `c` in the string `s`. If the character is not contained in the string, the null pointer (i.e., `0`) should be returned. The function must not use the array subscripting operator (i.e., `[]`).

6. **[output, output formatting, loop]**
   (a) Write a program that prints to standard output a table for converting from Fahrenheit to Celsius. The table should have two columns. The first and second columns should have the headings “Fahrenheit” and “Celsius”, respectively. Each column in the table should be 10 characters wide. The entries in each column must be properly aligned. The table should cover Fahrenheit values from 32 to 80 in steps of 0.5. [Note: The Celsius value c is related to the Fahrenheit value f as given by $c = \frac{5}{9}(f - 32)$.] [Hint: The following I/O manipulator may be useful: `std::setw`. I/O manipulators are declared in the header file called `iomanip`.] For example, the output of the program should look something like the following:

```
Fahrenheit   Celsius
        32         0
      32.5  0.277778
        33  0.555556
         .         .
         .         .
         .         .
        79   26.1111
      79.5   26.3889
        80   26.6667
```

   (b) Perform additional output formatting so that each temperature value is displayed with exactly two decimal places and the columns are right aligned (as shown below). [Hint: The following I/O manipulators may be useful: `std::fixed`, `std::setprecision`.] For example, the output of the program should be formatted exactly as shown below.

```
Fahrenheit   Celsius
     32.00      0.00
     32.50      0.28
     33.00      0.56
     33.50      0.83
         .         .
         .         .
         .         .
     79.00     26.11
     79.50     26.39
     80.00     26.67
```