6 Standard Library

6.1 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.

6.2 Topics Covered

These exercises cover the standard library, including such things as: containers, iterators, algorithms, and functors.

6.3 Exercises

1. **[containers, iterators, algorithms]**
   Write a program that reads real numbers from standard input until end-of-file is encountered (or some other error condition occurs), storing the real numbers into a `std::vector<double>` container called `values`. After all of the numbers have been read, the program should do the following (in order):
   (a) Use a **`for`** loop and an iterator (for the `vector` container) to print to standard output each of the elements stored in the container `values` in the order in which they are stored in the container.
   (b) Use a **`for`** loop and a reverse iterator to print to standard output each of the elements stored in the container `values` in the order opposite from how they are stored.
   (c) Use `std::count_if` in conjunction with the `isInteger` function given below to count how many of the elements of the container `values` are integer valued. Print the resulting count to standard output.

```
#include <cmath>

bool isInteger(double x)
{
        return floor(x) == x;
}
```

   (d) Use `std::transform` in conjunction with the `square` function given below to replace each element in the container `values` with the square of its value. Then, print to standard output each of the elements in the container (which are now the squares of the original values).

```
double square(double x)
{
        return x * x;
}
```

   (e) After the code has finished squaring the elements of the container `values`, create a const reference to this container called `values2`. That is, define `values2` as follows:

```
const std::vector<double>& values2 = values;
```

   Then, using a **`for`** loop and an iterator, print each of the elements in the container `values2`.
   (f) Rewrite the entire program so that it does not contain any explicit looping constructs whatsoever (e.g., no **`for`** loops, no **`while`** loops, no **`do`** loops, etc.). To do this, you will find the following to be useful: `std::copy`, `std::istream_iterator`, `std::ostream_iterator`, and `std::back_inserter`.
2. **[functors]**
   Write a program that will read a list of integers from standard input and then sort them in ascending order by absolute value. To perform the sorting operation, use the `std::sort` template function and a functor.