6 Assignment P4 [Assignment ID: `library`]

6.1 Preamble (Please Read Carefully)

Before starting work on this assignment, it is **critically important** that you **carefully** read Section 1 (titled "General Information") which starts on page 1 of this document.

6.2 Topics Covered

This assignment covers material primarily related to the following: containers, iterators, algorithms, standard template library.

6.3 Problems

1. Consider the simple program shown below. This program reads integers from standard input until either end-of-file is reached or an error is encountered, storing the read values into a vector. Then, the minimum element in the vector is found, before and after adding the element with value -1. Unfortunately, this program (which compiles and links with no problems) has a number of very serious bugs. For each bug:
   (a) Identify the relevant lines of code in the program by quoting the actual lines of code as well as stating their corresponding line numbers.
   (b) Clearly explain what is wrong with the code (i.e., why the bug is, in fact, a bug).
   (c) Explain how the bug can be fixed. In fixing the bug, you cannot use any functions other than those that were employed in the original program, except `std::vector<int>::empty`.

```
#include <iostream>
#include <vector>
#include <algorithm>

int main(int argc, char** argv)
{
    std::vector<int> values;

    std::vector<int>::iterator begin = values.begin();
    std::vector<int>::iterator end = values.end();

    // Read integers from standard input until end-of-file is reached
    // or an error is encountered, storing the read values in the vector.
    int x;
    while (std::cin >> x) {
        values.push_back(x);
    }

    // Find the minimum element in the vector.
    std::vector<int>::iterator minValueIter = std::min_element(begin, end);

    // Add an element with the value -1 to the vector.
    values.push_back(-1);

    // Find the minimum element in the vector after the new element with
    // the value -1 is added.
    std::vector<int>::iterator newMinValueIter =
      std::min_element(begin, end);

    // Output the two minimum values.
    std::cout << "minimum element from original vector: " << *minValueIter
      << "\n";
```

```
33      std::cout << "minimum element from updated vector: " << *newMinValueIter
34        << "\n";
35
36      return 0;
37  }
```

2. `stlDemo` **PROGRAM**. Write a program called `stlDemo` consisting of a single function `main` that does the following (in order):
   (a) Read a list of real values (separated by whitespace) from standard input into a `std::vector<double>` until end-of-file (EOF) is reached. If any abnormal condition (other than EOF) is encountered on the stream, the program should terminate immediately with an exit status of one. (When a stream is in an abnormal state, one can determine if the cause of the abnormal state was due to EOF being reached by using the `eof` member function of the `std::istream` class. This function returns **`true`** if EOF has been encountered and **`false`** otherwise.)
   (b) If the list of values is empty, terminate normally (i.e., with an exit status of zero).
   (c) Print the list values to standard output (i.e., `std::cout`) in the order that they were originally read. All of the values are to be output on the same line separated by a single space character, with the last value being immediately followed by a single newline character.
   (d) Print the minimum, maximum, mean, and median of the list values to standard output. All of these values are to be output on the same line, separated by a single space character, with the last value being immediately followed by a single newline character.
   (e) Print the list values in ascending order to standard output. All of these values are to be output on the same line, separated by a single space character, with the last value being immediately followed by a single newline character.

   The program must not produce any output to standard output, except that which is explicitly requested. If the program needs to output any other information (e.g., error messages, prompts for the user, etc.), standard error (i.e., `std::cerr`) may be used for this purpose.
   As an example of correct program behavior, the character input sequence "`5 4 3 2 1`" should produce exactly the following output:

```
5 4 3 2 1
1 5 3 3
1 2 3 4 5
```

   You should use the STL to: 1) minimize (as much as possible) the amount of code that you need to write; and 2) minimize the number of explicit looping constructs employed in the `main` function. If you maximally utilize the STL, you can construct a solution with no explicit looping constructs, and you are encouraged to try to find such a solution. An acceptable solution, however, is permitted to have up to two explicit loops (namely, one for outputting the list of values in each of unsorted and sorted orders). All of the source code for this program should be placed in a file called `stlDemo.cpp`. In order to avoid an explicit loop for reading the input data, you should use `std::istream_iterator` and `std::back_inserter`. You may find the following URLs to be helpful references on various parts of the C++ standard library relevant to this problem:
   - http://www.cplusplus.com/reference/algorithm
   - http://www.cplusplus.com/reference/iterator/back_inserter
   - http://www.cplusplus.com/reference/iterator/istream_iterator

   (If you use `std::ostream_iterator`, be careful not to inadvertently write extra unwanted space/whitespace characters to the stream.)
3. `stringDemo` **PROGRAM**. Write a program called `stringDemo` that consists of a single function `main` that does the following. The program reads a single word from standard input (using **`operator`**`>>`), which is stored in a `std::string`, and then computes and outputs the following (in order):
   (a) the length of the string (formatted as an integer);
   (b) the string formed by reversing the order of the characters in the original input string (e.g., the input string `abc123` would yield the reversed string `321cba`); and
   (c) the palindrome produced by concatenating the original input string and the reversed version of the string (e.g.,

the input string `abc123` would produce the palindrome `abc123321cba`).

As an example of correct program behavior, the character input sequence "`abc`" should produce exactly the following output:

```
3
cba
abccba
```

In your solution, you should use the C++ standard library to minimize (as much as possible) the amount of code that you need to write. At least one algorithm from the STL would be very helpful in this exercise. A correct solution can be done in less than 20 lines of code, including **`#include`** directives. Your solution should not have any explicit looping constructs. All of the source code for this program should be placed in the file `stringDemo.cpp`.