CMPUT 350 Lab 7 Exercise Problems

Rules:

- You can use all course material and man pages, but no other information such as web pages, books, or written notes. Using other information sources during the exercise constitues cheating.
- Your programs must compile without warning using

  ```
  g++ -Wall -Wextra -Wconversion -Wsign-conversion -O -g -std=c++17 ...
  ```

  In case there are compiler warnings or errors, marks will be deducted.
- Test your programs with different values. For now, the speed of your program is irrelevant. So don't spend time on optimization
- You must check for the appropriate preconditions/postconditions. Your program shouldn't crash or have undefined behaviour (**hint**: use asserts)!
- Your programs must be well structured and documented. Use ctrl-x t in Emacs to pretty-print it. Marks are assigned to functionality, program appearance, and comments.
- In case your program hangs, use ctrl-c to terminate it.
- Remember that you need to include the appropriate header files. To find out which ones you need for specific functions such as printf, use then man command.

Submit your solution files `Histo.h histoMain.cpp Sorter.h sorterMain.cpp` on eClass under "Lab 7 / Submission".

Important: Submit often (the system will only accept solutions submitted before 16:50)

---

1. Histogram [25 marks]

Implement a program that reads words from `stdin` and keeps track of word counts and when done prints the histogram in non-increasing count order like so:

```
the 392
moon 10
bar 1
clown 1
```

In case of identical counts the lexicographically smaller word is printed first (e.g., bar before clown above)

For this purpose, define class `Histo` in file `Histo.h` that has the following public methods:

```
// increment count for word w
void count(const std::string &w) {
    ... implement
}

// print histogram in non-increasing count order to stream os
void print(std::ostream &os) const {
    ... implement
}
```

Implement these functions on the spot (i.e., there is no `Histo.cpp` file). Make use of STL containers and algorithms (`map`, `vector`, `sort` ...) as much as possible. If unsure about STL data types and algorithms consult your lecture notes or the official C/C++ reference page.

In file `histoMain.cpp` implement function `main()`, which uses `Histo` for the program's task described above. You can assume that each word occurs at most 2 billion times.

Test it thoroughly and make sure your program doesn't leak memory and is well documented.

Hints:

- `std::string s; cin >> s;` reads the next word into string variable `s`
- consider range-based for loops and `auto`, or `cbegin()`, `cend()` when iterating through containers in constant member functions
- sort doesn't work with associative containers because they are sorted implicitly

```
Test input:

c b a
zz zz xx xx zz xx

Test output:

xx 3
zz 3
a 1
b 1
c 1
```

---

2. Sorter [15 marks]

For this question you will produce a simple class Sorter in file `Sorter.h` that can sort data by different criteria. It must accept some number of points given by two coordinates of type `double` each, specified by `add_point(x, y)`, and store them. Then, when `print_sorted_closest_to(x, y)` is called, it must sort the points based on their distance to the specified coordinates, and write them to `stdout` - one point per line - from closest to farthest. Implement all methods of `Sorter` in `Sorter.h`

To test class Sorter write a program in `sorterMain.cpp` that reads $n > 0$ points given by 2 `double` values each from `stdin` and prints the last $n - 1$ points in order closest to farthest from the first point.

Make sure your program doesn't leak memory and is well documented. Test it thoroughly.

Hints:

- The square root function sqrt is defined in header file cmath. Think about a way around using sqrt, because it is slow
- Consider using a functor object that stores the first point

```
Test input:

1 1
3 3
2 2

Test output:

2 2
3 3

===========

Test input:

1 1
2 2
2 2

Test output:

2 2
2 2
```