1 Introduction

This is a collection of exercises where you are supposed to use STL algorithms. To maximize your learning, try not to use any loops and use the standard library as much as possible.

2 Wordlist

Write a program that read an arbitrary text from `std::cin` and prints how many occurances there were of each unique word. There is a catch: you are only allowed to use `std::vector` and algorithms. So even though `std::map` is a perfect fit for this problem, you should solve it with algorithms instead. You shouldn't have to use `std::for_each`.

3 Result list

A local programming contest has just concluded and the result have been published. The programming contest was based on a scoring system, so a teams placement in the competition is purely based on this score. The organizers for the contest could therefore generate the highscore by simply storing each team and their score as a `std::pair` in a `std::vector` and then sort this vector based on the scores. Once this is done they simply extract the 3 highest scoring teams.

But now the contestants have requested that the organizer also publish the 3 fastest teams. This proves to be a problem for the organizers since they didn't record any times. But they did notice that each team were inserted in the same order as they finnished, so the three fastest teams are the three first elements in the vector.

Your task in this assignment is to generate the following table:

```
Highscore      | Fastest
Team D  : 37   | Team C
Team C  : 12   | Team B
Team B  : 5    | Team A
```

Based on the vector `results` given in `result.cc`. Store the the top three scoring teams in the vector `highscores`. You shouldn't have to modify anything in the given code and you shouldn't have to create a third container.

4 Fibonacci (last time, I promise)

As is the trend of this course, we are about to calculate fibonacci numbers. Again (I'm sorry for my bad imagination). This time we are going to do it with the standard library. Your program should read an integer `N` from `cin` and print the `N` first fibonacci numbers. You should only need one container. Tip: Fibonacci numbers grow quite quickly so you probably want to represent them as `unsigned long long int`.

5 Heaps

The STL contains several functions for making and operating on the data structure "min-max heap". See Wikipedia: `https://en.wikipedia.org/wiki/Min-max_heap` for more information on min-max heaps.

In this assignment you will create a class template called `Heap` that stores a comparator and an `std::vector` which will contain the heap. By default the comparator should be `std::less`.

`Heap` must fulfill the requirements of *Container*: `https://en.cppreference.com/w/cpp/named_req/Container`. Note that most of these requirements can just be the same as for `std::vector`. But keep in mind that to guarantee that the data in the class keeps on being a heap there shouldn't be any way for the user to modify elements in the underlying `std::vector`. Keep in mind that `const_iterator` cannot modify underlying data.

Also note that the rule of 0 is applicable here since we don't need any special behaviour for copying and moving (meaning the default implementations is enough).

To maximize your learning you should try to use STL algorithms as much as possible (specifically the heap operations).

There are tests given in `heap.cc`.