Exercise 1: Creating std::vector

Starting with my skeleton code, uncomment one line at a time to construct `vector v` using each of the following variations, and note any interesting observations.

vector<int> v{10,5};

The program outputted 10, 5. This is how I expected the program to run as vector v was both declared and initialized in this sentence.

vector<int> v = {10, 5};

The program outputted 10,5. I initially didn't expect this program to work because I assumed the keyword 'new' would had to be required.

vector<int> v(10,5);

The program outputted 5 5 5 5 5 5 5 5 5 5. An interesting observation is that C++ could have actually read this as needing to fill a vector with the value 5, 10 times.

vector<int> v = (10,5);

The compiler told me 'error: conversion from 'int' to non-scalar type 'std::vector' requested'. This could be because it is improper to use parenthesis when setting the values in the vector on the right of the equals sign. When I changed it to curly brackets, it outputted the result 10, 5 (just like in example 2).

Exercise 2: std::map vs std::unordered_map

What do you notice about the final ordering of the `map` vs the `unodered_map`?

The order of the map was in the same as the data inputted in the main. The unordered map was a little different as I noticed the order of the [1, "one"] was different to how it was inputted in the main.

What does this suggest about the underlying data structures? Try to verify this using the documentation for each collection on cppreference.com.

This suggests that map uses a data structure that keeps the keys in sorted order, while unordered_map doesn't. According to cppreference.com, map is typically implemented using a self-balancing binary search tree, and unordered_map is implemented using a hash table.

What happens to the load factor as you add items to the `unordered_map`? Explain why this happens.

It rises over time. The load factor is a ratio between the number of elements stored in a container and the number of buckets available in that container. Over time, we add more elements and the bucket size stays fixed at 13. Therefore, as we add more elements, the load factor increases as well.

Exercise 3: Algorithms

Upload your code (x1) and its output (x1) to GitHub.

v = 0 0 0 0 0 0 0 0 0 0 0 0 0 0
v = 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
v = 5 6 7 8 9 10 11 12 13 15 14 16 17 18 19
v = 20 20 20 20 9 10 11 12 13 15 14 16 17 18 19
v = 20 9 10 11 12 13 14 15 16 17 18 20 20 20 19
v = 20 10 12 14 16 18 20 20 20 9 11 13 15 17 19