CS304 Assignment 5 – Unordered map

Worth: 10%

Due: Wednesday April 12th, 2023

Outline: in this assignment, you will create two separate implementations of an unordered map in C++:

1) Unordered_map using **hash table with separate chaining**
2) Unordered_map using **red-black tree**
    a. Note – typically, trees are used for *ordered* map, not unordered map, but we will use it in this assignment for academic purposes.

Description: unordered maps are *associative containers* that store elements formed by the combination of a **key value** and a **mapped value**, allowing for fast retrieval of individual elements based on their keys. The key value is used to identify the element, and the mapped value is an object with content associated to this key. Check section 4.8 of the Weiss textbook for more details.

Your map must implement the following methods:

find, erase, insert, operator[], size, clear, count, empty

You must provide a method for all the signatures described in the pages linked to above. For example, erase requires three separate implementations, you must provide all three:

```
by position (1)  iterator erase ( const_iterator position );
by key (2)       size_type erase ( const key_type& k );
range (3)        iterator erase ( const_iterator first, const_iterator last );
```

Most of these functions will just involve calling the respective function on the underlying data structure (hash table or tree). For example, `operator[]` will call `contains` on the tree and then return the element if it exists. Because `erase` requires an iterator, you must provide a full iterator implementation for your unordered map. This means implementing separate iterators for hash table and red-black trees. Page 176 of Weiss gives hints for tree iterator.

Timing experiments: compare the performance of your two map implementations on 100000 elements. first, `insert` all the elements, then find and return them using `operator[]`, then erase them one-by-one using `erase`. Do two versions of the experiment, version 1 using int values as the key in your key-value pair, and version 2 using strings (of at least 8 chars) in your key-value pair. You can use 'float' as value in both cases. Your table should look as follows:

	Hash table (int keys)	Red-black tree (int keys)	Hash table (string keys)	Red-black tree (string keys)
Insert 100k elements				
operator[] 100k times				
Erase 100k elements				

The hash table is expected to perform better, but make sure to accurately report whatever you find!

Hints: you may use the red-black tree and hash table code from the Weiss textbook as the base data structures. However, you will need to modify this code to store key-value pairs, *and* you will need to implement the rest of the map logic yourself, including all the functions outlined above. You will probably want to provide two separate map implementations, example: TreeMap and HashTableMap, instead of making the data structure a template parameter.

Bonus: see if you can speed up your map by using different hash functions, collision resolution methods (quadratic or linear probing), and tree implementations. Also, try to provide a single map class, and make the data structure (tree or hash table) a template parameter (similar to what `stl` does for queue with `class Container`).