In this project, you will be writing a `WordCount` class that will implement a hash table with separate chaining to store unique words and the number of occurrences of each word. You will write the contructor, `hash` method using STL's hash template, a `count` method that returns the number of occurrences of the input word, and an `add` method that either increments a word's count or creates a new entry in the hash table with separate chaining. Basically, we are writing pieces of the STL `unordered_map` class. Then, we will use this class to count the number of words in a file and improve this to deal with casing and punctuation. Use the `test_class.cc` and `test_file_wc.cc` files to test your work. Submit your completed `wordcount.cc` file.

WordCount

For our `WordCount` class, we need a container, in addition to hash, count, and add methods. Because we want to mimic the `unordered_map`, our container will be of type `vector<list<pair<string,int>>>`. The vector represents the hash table, and each table entry is a linked list (for separate chaining). Each item in the list is a pair `<string, int>` which will store the word and the number of times it appears. The `print` function is provided for you and prints the words and their counts in alphabetical order.

Constructor

The constructor for the `WordCount` class takes an integer that specifies the **size** of the hash table. Save this parameter to be used as the modulus for the hash function! Also, make sure to **resize** the vector to the given size.

Hash Function

Define the `hash` instance method that takes a string and returns its hash using STL's built-in hash template. Note that `hashFunction` is defined as a private member for you and passing a string to it will return an integer. You will need to use this and add the modulus to constrain the values to the table's size.

Count (Search)

Define the `count` method that takes a word (string) as a parameter and returns the count of occurrences (int) which is zero (0) if the word has not been seen. In order to search for a word in the hash table, we will apply the hash function and then check the list of items at that entry in the hash table. Recall

that each entry in the hash table vector is a **linked list** so we need to search the list to see if the element exists. Consider using the `find_if` method to search the list. Each item in the list is a pair `std::pair<string, int>` where the first item is the word and the second item is the count of the number of times that word appears. If the word exists, return the second half of the pair (the count). Otherwise, return 0.

Add (Insert)

Define the add method that takes a word (string) as a parameter and either increments the count of occurrences or adds the word with a count of one (1) to the hash table. Each word will appear in the hash table as a pair `std::pair<string, int>` where the first item is the word and the second item is the count of the number of times that word appears. We can follow the same basic procedure as the search function to determine if the word already exists. If it exists, the second item in the pair should be incremented. If it doesn't exist, add a new pair with the word and a count of one (1).

File Word Count

Now, using our `WordCount` class, we will use an instance of it to read in the words in a text file and store the number of occurrences of each word. You should define a function `printFileWordCount` that takes a filename as a string along with the table size, constructs a `WordCount` instance, reads in all of the words from that file, adding them to the WordCount container, and then prints out the final word count.

Start by developing a solution that assumes that the text file contains only lowercase words and no punctuation. Then, add functionality to convert all words to lowercase and remove punctuation. Three versions of the files are included (`simple.txt`, `mixed_case.txt`, and `punctuation.txt`) to test your solutions.

To convert to lowercase, the tolower method will be useful, but note that it only works on **one** character at a time. You can use STL's transform to apply this to all characters in a string.

To remove punctuation, the ispunct method will be useful, but again, it only works on one character at a time. Here, we need to erase those locations where the punctuation occurs.