What the Program Does

My project is split into four main files, with three classes. The three classes are: QuadProbingHashTable (which handles quadratic probing), ChainingHashTable (which handles chaining), and ChainingHashTableEntry (entries in the Chaining Hash Table, which function as Linked List nodes).

ChainingHashTableEntry only stores the value (word) at a given index along with a next pointer that points to any words which also hash to that index.

Both HashTable classes (ChainingHashTable and QuadProbingHashTable) create private arrays for hash tables, and both have insertion and search functions. Where they differ is that ChainingHashTable stores pointers to ChainingHashTableEntry objects and uses linked list logic to insert and search for values, while QuadProbingHashTable stores strings and uses recursion to insert and search for values.

The main.cpp file is the driver file, which contains both the main function and a function that "cleans" the words being read in from the dictionary (decapitalizing all characters, removing commas/dashes). It does this by looping through the string parameter passed to the function, and checking each individual character, "cleaning" it, then pushing the cleaned character to a string that is returned by the function.

In the main function, I first read in the dictionary file (with all cleaned words) into an array, and do the same with the words to be spellchecked. This is to slightly optimize the program so that the words are cleaned before the construction of the hash table, not during it. It also prevents the program from having to read the words twice (once for Quad Probing, once for Chaining). Next, I loop through the dictionary array and insert each string into a QuadProbingHashTable array using the insert function. Last, I loop through the spellcheck array, running the search function on each value in the array to check whether or not the string is in the hash table. This is repeated for the ChainingHashTable.

The efficiency of my program is O(n) to construct the hash table (where n = number of values in dictionary) because the hash function runs in constant time, meaning that the only thing which increases time is number of entries in the dictionary. It also has n memory usage, because each new entry takes up a constant amount of memory. I got the following time usages:

	Misspelled Words (Quad Probing)	Misspelled Words (Chaining)	Time (Quad Probing)	Time (Chaining)
Simple	0	0	0 ns	0 ns
1	132	132	1.62*10^8 ns	1.81*10^8 ns
2	1094	1094	6.2*10^8 ns	3.71*10^8n ns

I optimized time usage by storing words in a dictionary/spellcheck array first as mentioned above, and I also optimized time usage by not allowing the numbers in the hash function to grow too large by constraining how large the exponent could grow (the hash function I used multiplied each character's ascii value by a value k^n, which was a set constant (17) to the power of the position of that character).