

Copy your `HashTable::addWord()` method into the box below, with the provided prototype. You may use any helper functions as needed, but they must be copied into the answer box.

You may assume that ALL other class methods are defined and available for you to call.

Your answer should look like:

```
void HashTable::addWord(std::string word)
{
    // Your code here
}
```

Answer: (penalty regime: 0 %)

```
1 void HashTable::addWord(std::string word)
2 {
3     if (numItems >= hashTableSize)
4     {
5         int newSize = hashTableSize * 2;
6         wordItem** temp = new wordItem*[newSize];
7
8         for (int i = 0; i < hashTableSize; i++) {
9             temp[i] = new wordItem;
10            temp[i]->word = hashTable[i]->word;
11            temp[i]->count = hashTable[i]->count;
12            temp[i]->next = hashTable[i]->next;
13        }
14        hashTableSize *= 2;
15        delete[] hashTable;
16        hashTable = temp;
17    }
18    wordItem* w = new wordItem;
19    w->word = word;
20    w->count = 1;
21    w->next = nullptr;
22
23    wordItem** temp = new wordItem*[hashTableSize + 1];
```

Check

Test	Expected	Got
✖ Add words to a newly-created hash table: NO COLLISIONS.	<pre> [GRADER] Constructing HashTable, size=53↵ [GRADER] Now adding some words...↵ [GRADER] calling addWord("table") (hash = 4)↵ [GRADER] calling addWord("plant") (hash = 11)↵ [GRADER] calling addWord("lamp") (hash = 36)↵ [GRADER] calling addWord("stairs") (hash = 17)↵ [GRADER] calling addWord("flowers") (hash = 22)↵ [GRADER] calling addWord("books") (hash = 0)↵ ↵ [GRADER] Now, checking for the presence↵ of the above words using our↵ implementation of isInTable()↵ [GRADER] isInTable("table"): True↵ [GRADER] isInTable("plant"): True↵ [GRADER] isInTable("lamp"): True↵ [GRADER] isInTable("stairs"): True↵ [GRADER] isInTable("flowers"): True↵ [GRADER] isInTable("books"): True↵ ↵ [GRADER]: Checking # of unique words & collisions:↵ [GRADER]: numItems: 6↵ [GRADER]: numCollisions: 0↵ ↵ [GRADER] Test complete </pre>	<pre> [GRADER] Constructing HashTable, size=53↵ [GRADER] Now adding some words...↵ [GRADER] calling addWord("table") (hash = 4)↵ [GRADER] calling addWord("plant") (hash = 11)↵ [GRADER] calling addWord("lamp") (hash = 36)↵ [GRADER] calling addWord("stairs") (hash = 17)↵ [GRADER] calling addWord("flowers") (hash = 22)↵ [GRADER] calling addWord("books") (hash = 0)↵ ↵ [GRADER] Now, checking for the presence↵ of the above words using our↵ implementation of isInTable()↵ [GRADER] isInTable("table"): False↵ [GRADER] isInTable("plant"): False↵ [GRADER] isInTable("lamp"): False↵ [GRADER] isInTable("stairs"): False↵ [GRADER] isInTable("flowers"): False↵ [GRADER] isInTable("books"): False↵ ↵ [GRADER]: Checking # of unique words & collisions:↵ [GRADER]: numItems: 6↵ [GRADER]: numCollisions: 0↵ ↵ [GRADER] Test complete </pre>
✖ Add words to a newly-created hash table: WITH COLLISIONS.	<pre> [GRADER] Constructing HashTable, size=53↵ [GRADER] Now adding some words...↵ [GRADER] calling addWord("when") (hash = 6)↵ [GRADER] calling addWord("know") (hash = 6)↵ [GRADER] calling addWord("district") (hash = 21)↵ [GRADER] calling addWord("sunlight") (hash = 21)↵ [GRADER] calling addWord("then") (hash = 50)↵ [GRADER] calling addWord("head") (hash = 50)↵ [GRADER] calling addWord("animals") (hash = 50)↵ ↵ [GRADER] Now, checking for the presence↵ of the above words using our↵ implementation of isInTable()↵ [GRADER] isInTable("when"): True↵ [GRADER] isInTable("know"): True↵ [GRADER] isInTable("district"): True↵ [GRADER] isInTable("sunlight"): True↵ [GRADER] isInTable("then"): True↵ [GRADER] isInTable("head"): True↵ [GRADER] isInTable("animals"): True↵ ↵ [GRADER]: Checking # of unique words & collisions:↵ [GRADER]: numItems: 7↵ [GRADER]: numCollisions: 4↵ ↵ [GRADER] Test complete </pre>	<pre> [GRADER] Constructing HashTable, size=53↵ [GRADER] Now adding some words...↵ [GRADER] calling addWord("when") (hash = 6)↵ [GRADER] calling addWord("know") (hash = 6)↵ [GRADER] calling addWord("district") (hash = 21)↵ [GRADER] calling addWord("sunlight") (hash = 21)↵ [GRADER] calling addWord("then") (hash = 50)↵ [GRADER] calling addWord("head") (hash = 50)↵ [GRADER] calling addWord("animals") (hash = 50)↵ ↵ [GRADER] Now, checking for the presence↵ of the above words using our↵ implementation of isInTable()↵ [GRADER] isInTable("when"): False↵ [GRADER] isInTable("know"): False↵ [GRADER] isInTable("district"): False↵ [GRADER] isInTable("sunlight"): False↵ [GRADER] isInTable("then"): False↵ [GRADER] isInTable("head"): False↵ [GRADER] isInTable("animals"): False↵ ↵ [GRADER]: Checking # of unique words & collisions:↵ [GRADER]: numItems: 7↵ [GRADER]: numCollisions: 0↵ ↵ [GRADER] Test complete </pre>

Hide differences

Copy your `HashTable::getTotalNumWords` method into the box below, with the provided prototype. You may use any helper functions as needed, but they must be copied into the answer box.
You may assume that ALL other class methods are defined and available for you to call.
Your answer should look like:

```
int HashTable::getTotalNumWords()
{
    // Your code here
}
```

Answer: (penalty regime: 0 %)

```
1 int HashTable::getTotalNumWords()
2 {
3     int totalNumWords = 0;
4     for (int i = 0; i < numItems; i++)
5     {
6         totalNumWords += hashTable[i]->count;
7     }
8     return totalNumWords;
9 }
```

Check

Test	Expected	Got	
✖ 1. Reading in a list of stopwords, 2. Adding words to table using a provided implementation of HashTable::addWord, 3. calling your getTotalNumWords(). This test uses HW7-IgnoreWords.txt, and a hash table size of 53.	[GRADER] creating instance of HashTable [GRADER] reading stop words [GRADER] Adding "table" (hash value = 11) [GRADER] Adding "plant" (hash value = 22) [GRADER] Adding "lamp" (hash value = 7) [GRADER] Adding "stairs" (hash value = 6) [GRADER] Adding "flowers" (hash value = 20) [GRADER] Adding "books" (hash value = 49) [GRADER] Your getTotalNumWords() method returns: 6 [GRADER] test finished	[GRADER] creating instance of HashTable [GRADER] reading stop words [GRADER] Adding "table" (hash value = 11) [GRADER] Adding "plant" (hash value = 22) [GRADER] Adding "lamp" (hash value = 7) [GRADER] Adding "stairs" (hash value = 6) [GRADER] Adding "flowers" (hash value = 20) [GRADER] Adding "books" (hash value = 49) [GRADER] Your getTotalNumWords() method returns: ***Error*** Segmentation fault	✖

Copy your `HashTable::printTopN` method into the box below, with the provided prototype. You may use any helper functions as needed, but they must be copied into the answer box.

You may assume that ALL other class methods are defined and available for you to call.

Your answer should look like:

```
void HashTable::printTopN(int n)
{
    // Your code here
}
```

Answer: (penalty regime: 0 %)

```
1 void HashTable::printTopN(int n)
2 {
3     int length = numItems;
4     for (int i = 0; i < length - 1; ++i) {
5         for (int j = 0; j < length - i - 1; ++j) {
6             if (hashTable[j]->count < hashTable[j + 1]->count) {
7                 int count = hashTable[j]->count;
8                 string word = hashTable[j]->word;
9                 hashTable[j]->count = hashTable[j + 1]->count;
10                hashTable[j]->word = hashTable[j + 1]->word;
11                hashTable[j + 1]->count = count;
12                hashTable[j + 1]->word = word;
13            }
14        }
15    }
16
17    int totalNumberOfWords = getTotalNumWords();
18    for (int i = 0; i < n; i++)
19    {
20        wordItem* w = hashTable[i];
21        cout << fixed << setprecision(4) << (float)w->count/totalNumberOfWords << " - " << w->word << endl;
22    }
23 }
```

Check

Test	Expected	Got
✖ Testing with, N = 10, input file = HarryPotter.txt, stop words file = ignoreWords.txt as stop words, hash table size = 53	[GRADER] creating instance of HashTable [GRADER] reading stop words [GRADER] Processing input file [GRADER] your printTopN() method outputs: 0.0241 - harry 0.0236 - was 0.0158 - said 0.0139 - had 0.0100 - him 0.0081 - ron 0.0068 - were 0.0067 - hagrid 0.0065 - them 0.0052 - back [GRADER] test finished	[GRADER] creating instance of HashTable [GRADER] reading stop words [GRADER] Processing input file [GRADER] your printTopN() method outputs: ***Error*** Segmentation fault

Testing was aborted due to error.

Show differences

Enter the full contents of your Homework 7 file that contains the **main function**. The implementations of all class methods have been provided, we are only testing your main.

You may use the following include statements:

```
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>

#include "HashTable.hpp"
```

Answer: (penalty regime: 0 %)

```
1 #include "HashTable.hpp"
2 #include <iostream>
3 #include <fstream>
4 #include <algorithm>
5
6 using namespace std;
7
8 int main(int argc, char** argv)
9 {
10     int topN = atoi(argv[1]);
11     int N = atoi(argv[4]);
12
13     char* ignoreWordFileName = argv[3];
14     HashTable ignoreWords = HashTable(STOPWORD_TABLE_SIZE);
15     getStopWords(ignoreWordFileName, ignoreWords);
16
17     char* filename = argv[2];
18     ifstream file;
19     file.open(filename);
20     if (file.fail()) {
21         cout << "Error opening file!" << endl;
22         return 1;
23     }
```

Check

	Test	Expected	Got	
✖	Testing with below command line arguments, N = 15, input file = HarryPotter.txt, stopword list = ignoreWords.txt, hash table size = 500	0.0241 - harry 0.0236 - was 0.0158 - said 0.0139 - had 0.0100 - him 0.0081 - ron 0.0068 - were 0.0067 - hagrid 0.0065 - them 0.0052 - back 0.0051 - hermione 0.0047 - its 0.0044 - into 0.0042 - been 0.0040 - off # Number of collisions: 5485 # Unique non-stop words: 5985 # Total non-stop words: 50331	***Error*** Program contains illegal include `#include <algorithm>	✖

Testing was aborted due to error.

Show differences