

PONDER 12 : SPELL-CHECKER

Due Saturday at 11:59 PM MST

The next programming assignment will be to implement the hash data structure and use it to implement a spell checker.

Hash

Create a class to capture the notion of a hash. This class will store a collection of elements and be able to perform constant-time insertion and lookup. Additionally, this class will need to support any data type. Since a hash function needs to be provided and since hash functions are very much data type dependent, we will have to do this in two steps. First, create an abstract class representing a hash where the hash() method is a pure virtual function. Then create a collection of derived classes, each specific to a given data type.

The base class needs to be called Hash and reside in hash.h. The following operations will need to be supported:

* **Constructor**: Non-default constructor and the copy constructor. The non-default constructor takes the number of buckets as a parameter. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate memory for the hash.
* **Destructor**: Delete all the elements in the Hash.
* **operator=**: Assignment operator. Copy one Hash into another. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate memory for the hash.
* **empty()**: Determines whether the current Hash is empty.
* **size()**: Returns the number of elements in the Hash.
* **capacity()**: Returns the number of buckets in the Hash.
* **clear()**: Clear the contents. This method takes no parameters and returns nothing.
* **find()**: The parameter is the value to be found. Returns true if the value is found, falseotherwise.
* **insert()**: Places a new instance of a value in the Hash. The parameter is the item to be inserted; there is no return value.
* **hash()**: This is a pure virtual function taking an element as a parameter and returning an index into the underlying array.

Your Hash class will need to be able to handle hash table collisions. You can do this any way you choose, but perhaps the easiest is to perform chaining using a List or a BST in the hash table.

Because the Hash is an abstract class, the client cannot instantiate an instance. This means we must develop derived classes, each with a defined data type and a defined hash() function. Two of these are provided for the driver program: IHash for integers and FHash for floating point numbers. For example, the IHash class is:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* I HASH

\* A simple hash of integers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class IHash : public Hash <int>

{

public:

IHash(int numBuckets) throw (const char \*) : Hash <int> (numBuckets) {}

IHash(const IHash & rhs) throw (const char \*) : Hash <int> (rhs) {}

// hash function for integers is simply to take the modulous

int hash(const int & value) const

{

return value % capacity();

}

};

There are a few things to notice about this class. First, we are deriving the non-template IHashfrom the template Hash. Second, we must define the hash() function in the derived class because it only knows enough about the data type. Finally, all the normal hash functionality is inherited from the Hash class.

Spell Checker

The second part of this assignment is to use our Hash class to implement a spell checker. We will read a dictionary from a file into a Hash object, then walk through a file looking for spelling errors. If a word in the file is not in the dictionary, it is determined to be spelled incorrectly. Otherwise, it is assumed to be correct.

It is critically important that a spell checker be fast. Ideally, we should be able to confirm the spelling of a word in constant O(1) time. Clearly, a hash table is a great tool for the job.

Please create a derived class from Hash designed specifically for this spell-checker job. It needs to read a dictionary from /home/cs235/week12/dictionary.txt, which will be inserted into the hash table. You can make the capacity of the hash anything you like. Here you need to balance the amount of wasted space with the number of collisions. Please put a short comment in your code describing why you chose a given size.

Please also create a driver for your spell checker. It will prompt the user for the file to be spell-checked, then display the list of all the misspelled words. An example output of the program is:

What file do you want to check?

/home/cs235/week12/nephi.txt

Misspelled: Nephi, yea

Another example:

What file do you want to check?

/home/cs235/week12/twoCities.txt

File contains no spelling errors

Of course, your program must also be performant with very large data-sets. A few hints to help you with your program:

* Leverage the work you did earlier in the semester. Classes you built earlier such as Listand BST may come in handy for the Hash class.
* Your first guess for the spell checker hash function will probably not be the best. You may wish to write some code to help you analyze the quality of your hash function. It might, for example, count the number of empty buckets and the size of the largest bucket.
* Put a detailed description in your hash() function header that describes how it works and why you choose the design you used. This will be necessary for full credit. Please see the rubric for details.
* If your type definitions get too heavy, you might want to use typedef to simplify your code and increase readability. This may mean that you will need to read-up on what typedef is and how to use it.

Common Mistakes

The most common mistakes students make with this assignment include the following:

* **Invalid hash value**. You might want to write an assert to verify that the number returned from the hash() function is between 0 and capacity.

Test Bed

The testBed for this assignment is:

testBed cs235/week12 week12.tar

You can also run testBed on the executable:

testBed cs235/week12 a.out

Of course, you will need to pass testBed to get full credit on the assignment.

Submitting

You will submit this assignment individually using the Linux submit command. Please:

1. Create a TAR file built from the makefile, which will contain several files:
   * makefile: Directly from /home/cs235/week12/makefile except with your edits on the comment block.
   * hash.h: Your class definition for Hash.
   * spellCheck.h: The function prototype for spellCheck().
   * spellCheck.cpp: Your spell checker hash class, implementation of spellCheck(), and any other code you may need to solve the problem.
   * week12.cpp: Unmodified from /home/cs235/week12/week12.cpp.
2. Run the program by hand a few times through all four test cases as well as the hash algorithm.
3. Verify your solution with testBed.
4. Submit your file using the submit command. The submit command will prompt you for your instructor, the class (cs235), and the assignment (week12). You submit your file with:

submit week12.tar

Your program will be graded according to the following rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Exceptional 100% | Good 90% | Acceptable 70% | Developing 50% | Missing 0% |
| Hash interface 30% | The interfaces are perfectly specified with respect to const, pass-by-reference, etc. | week12.cppcompiles without modification | All of the methods in Hash match the problem definition | Hash has many of the same interfaces as the problem definition | The public methods and variables in theHash class do not resemble the problem definition |
| Implementation 10% | Passes all four Hash testBed tests | Passes three testBed tests | Passes two testBed tests | Passes one testBed test | Program fails to compile or does not pass any testBed tests |
| Spell Check 20% | The code is elegant and efficient | Passes the spell check testBed test | The code essentially works but with minor defects | Elements of the solution are present | The word count problem was not attempted |
| Hash Function 20% | The hash function efficiently stores the dictionary and is well-explained in the comments | A clever hash function was defined that works very well | The hash function works | The hash function has a serious bug | The hash function is missing |
| Code Quality 10% | There is no obvious room for improvement | All the principles of encapsulation and modularization are honored | One function is written in a "backwards" way or could be improved | Two or more functions appears "thrown together" | The code appears to be written without any obvious forethought |
| Style 10% | Great variable names, no errors, great comments | No obvious style errors | A few minor style errors: non-standard spacing, poor variable names, missing comments, etc. | Overly generic variable names, misleading comments, or other gross style errors | No knowledge of the BYU-I code style guidelines were demonstrated |

Please make sure to fill out the program header in the makefile with the following information: the name of both programmers, the amount of coding time required by each to complete the assignment, the amount of discussion time (the Pair Programming part), and what was the most difficult part. Failure to do this will result in a loss of 10% on the assignment.

There will be no extra credit opportunities for this programming assignment.