

PONDER 10 : WORD COUNT

Due Saturday at 11:59 PM MST

The next programming assignment will be to implement the map data structure and use it to count the frequency of words in a large file.

Map

Create a class to capture the notion of a map. As you may recall, this is an associative container formed with a key value and a data value. The key value is used to uniquely identify a given data value. Both the key value and the data value can be any data type.

The user of the map container can store any collection of data value items. The only constraint is that the data value knows how to copy itself. In practical terms, this means the assignment operator and the copy constructor must be defined for the data value. Additionally, each data value must have an associated key value. This key value can also be any data type as long as the data type can be compared with another key value. This means the greater than operator and the equivalence operator must be defined.

You are to create a Map class that works with any data type for the key value and data value. The easiest way to accomplish this is by having a BST object as a member variable. This BST will have nodes consisting of a Pairs, each of which having the key value as the first value and the data value as the second. When finished, your Map class will work exactly like the [std::map](http://www.cplusplus.com/reference/map/map/) class. It will need to be defined in its own header file (map.h). The class name must be Map and will need to support the following operations:

* **Constructor**: Default constructor and the copy constructor doing the normal thing.
* **Destructor**: Delete all the elements in the map.
* **operator=**: Assignment operator. Copy one map onto another. As with the copy constructor, the BST assertion is throw if a a node cannot be allocated.
* **empty()**: Determines whether the current map is empty. This can be accomplished by calling the empty() method on the underlying BST class.
* **size()**: Returns the number of elements currently in the map. This can be handled by adding a numItems member variable in the Map class or in the BST class. Either solution works well.
* **clear()**: Delete all the elements in the map, accomplished by calling the underlyingBST::clear() method.
* **find()**: The parameter is the key value to be found and the return value is an iterator referring to the data value in the tree corresponding to the parameter. If no element is found, the end() iterator will be returned.
* **operator[]**: The array index operator has two purposes in a map: The first is to retrieve a given data value from the map; the second is to insert a new data value into the map. In both cases, the parameter to the array index operator is a key value and the return value is the data value. Since the array index operator must work on either the right hand side or the left hand side of the assignment operator, it must return a data value by-reference.
* **begin()**: There is no parameter and the return value is an iterator referring to the left most element in the map.
* **end()**: There is no parameter and the return value is an iterator referring to no element in the map. By convention, this is the NULL iterator.

As mentioned previously, the easiest way to make a map class is to use a BST as a member variable. Since each node in the BST must support two values (the key value and the data value), the BST must be made from the Pair data type. This is provided in /home/cs235/week10/pair.h. Observe how the comparison operators only look at the first value in the pair. This is where the key value must go. Please see the following example:

{

Map < char, string > m;

m['a'] = "Apple";

m['b'] = "Banana";

cout << m['a'] << endl;

}

In addition to the Map class, it is necessary to make a MapIterator class which will iterate through the map. This will be much more straight forward than the previous assignment; it is necessary only to keep a BSTIterator as a member variable inside the MapIterator. All the usual iterator methods will need to be supported.

Tree Balancing

The tree balancing will not be required for this assignment. You do not need to pass the tree balancing part of the testbed.

Driver Program

A driver program is provided. This file (/home/cs235/week10/week10.cpp) will pound-include your header file (map.h) and expect a template class Map to be defined therein. This program will exercise your class, filling the container with user input and displaying the results. As with previous assignments, a makefile will be provided (/home/cs235/week10/makefile). You will need to create a header file (map.h) for your class.

Word Count

In addition to passing the four test functions for the Map class, you will also write a program that uses the Map class to count the number of occurrences of words from a data file. The array indexes will be the words and the value of each element of the array will be a count of the number of occurrences of that word.

Consider the following file fruit.txt with three words:

apple apple orange

When the user runs the program, the following output results:

What is the filename to be counted? fruit.txt

What word whose frequency is to be found. Type ! when done

> apple

apple : 2

> orange

orange : 1

> grapes

grapes : 0

Of course, your program must also be performant with very large data sets. If, for example, the entire Book of Mormon was read as a file, the following output would result:

What is the filename to be counted? /home/cs235/week10/bom.txt

What word whose frequency is to be found. Type ! when done

> Nephi

Nephi : 482

> Lord

Lord : 1575

> the

the : 19127

> computer

computer : 0

A few hints to help you with your program:

* Take a close look at the Pair class in /home/cs235/week10/pair.h. This will come in handy for the Map class.
* Your Map class should have a single member variable: a BST object. Almost all the methods will simply call the corresponding BST method. The only exception is the array index operator.
* While the Map class has two templates (the Key and the Value), the BST has one. This means that the member variable inside the Map class is a BST < Pair < K, V > > bst .
* By default, if you use the array index operator for a key value that does not currently exist in the map, a new node will be created with the default data value. Since there is no default value for an integer, this will mean the data value will be uninitialized unless you do something special.

Common Mistakes

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

* **Broken BST class**. If your BST class has a bug in it, it is highly unlikely that it will work in your Map class will work.
* **Dropped pointer**. When doing the Red-Black tree implementation, it is common to fail to hook up all the pointers correctly. This results in a subtree being left off of the main tree. Check and double-check all your pointer assignments.
* **No BST member variable**. It is possible to create a Map class by modifying your BST class from the previous lesson. It is much easier to instead create a new class with a single member variable: the BST.

Test Bed

The testBed for this assignment is:

testBed cs235/week10 week10.tar

You can also run testBed on the executable:

testBed cs235/week10 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/week10/makefile except with your edits on the comment block.
   * map.h: Your class definition for Map.
   * wordCount.h: The function prototype for wordCount().
   * wordCount.cpp: All the code necessary to count the words in a file.
   * week10.cpp: Unmodified from /home/cs235/week10/week10.cpp.
2. Run the program by hand a few times through all four test cases as well as the binary sort 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 (week10). You submit your file with:

submit week10.tar

Your program will be graded according to the following rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Exceptional 100% | Good  90% | Acceptable  70% | Developing  50% | Missing  0% |
| Map interface  10% | The interfaces are perfectly specified with respect to const, pass-by-reference, etc. | week10.cppcompiles without modification | All of the methods in Map match the problem definition | Map has many of the same interfaces as the problem definition | The public methods and variables in theMap class do not resemble the problem definition |
| Implementation  20% | Passes all four Map 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 |
| MapIterator  10% | Solution works, is elegant, and efficient | Both forward and reverse iterators work | Works in some limited cases | Elements of the solution are present | No attempt was made to iterate through the Map |
| Word Count  10% | The code is elegant and efficient | Passes the word count testBed test | The code essentially works but with minor defects | Elements of the solution are present | The word count problem was not attempted |
| Red-Black Tree  40% | Passes all the Red-Black tests | Pass Case 4a | Pass Case 3 | Pass Case 1 and Case 2 | No Red-Black tree tests pass test-bed |
| Code Quality  20% | 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 amount of coding time required to complete the assignment, and what was the most difficult part. Failure to do this will result in a loss of 10% on the assignment.