

Assignment 5: Maps for word completion

Out: 26th March 2014

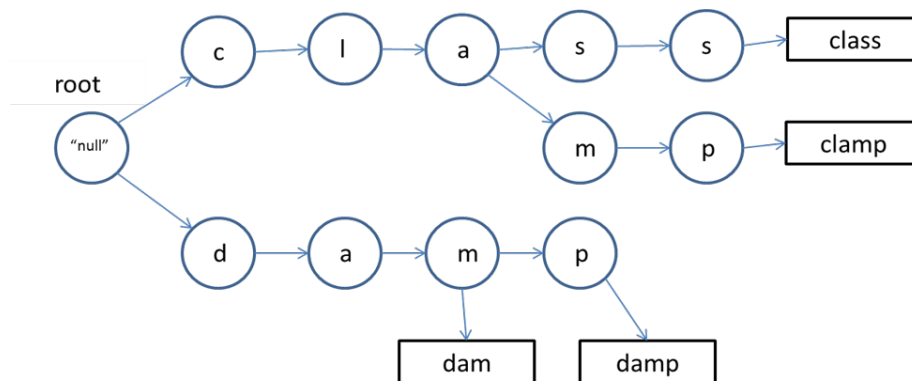
Due: 11th April 2014 at 11:55pm

In this assignment you will use two versions of maps in a program that is virtually the same as Assignment 2: it implements a “word-complete” feature (i.e. after typing some letters, it will suggest words from the dictionary that start with them in order). This program replaces the array of nodes with a map, which you will implement. The first map will be a simple STL-based map, and the second will be based on an AVL tree implementation.

Provided code and what it means

Some code has been provided to you as a start, along with a dictionary file (english.0). After compiling the linking the program using the provided Makefile, the program should be run as “./PrefixFinder english.0”. The program as-is will compile, although it will produce a segmentation fault when you run it because it is incomplete.

The code provides a word manager that contains all the words from the provided dictionary. This data structure can be thought of as a tree. When a new word is to be added, the word manager breaks it into individual letters and adds it as follows: the first letter is used by the root to get a child node corresponding to that letter, followed by the second letter, and so on. When the node corresponding to the last node is reached, the word is stored there. This data structure is called a “trie”.



Given a letter, each node of the trie uses a **map** to determine the next node. The virtual class “MapInterface” provides the intended interface for a map that the WordManager uses. The first template parameter “S” is the key of the map, while “T” is the value. Any data-type of S should be comparable, i.e. you should be able to compare two things of that type. “T” has no such restriction. The map cannot contain duplicate keys.

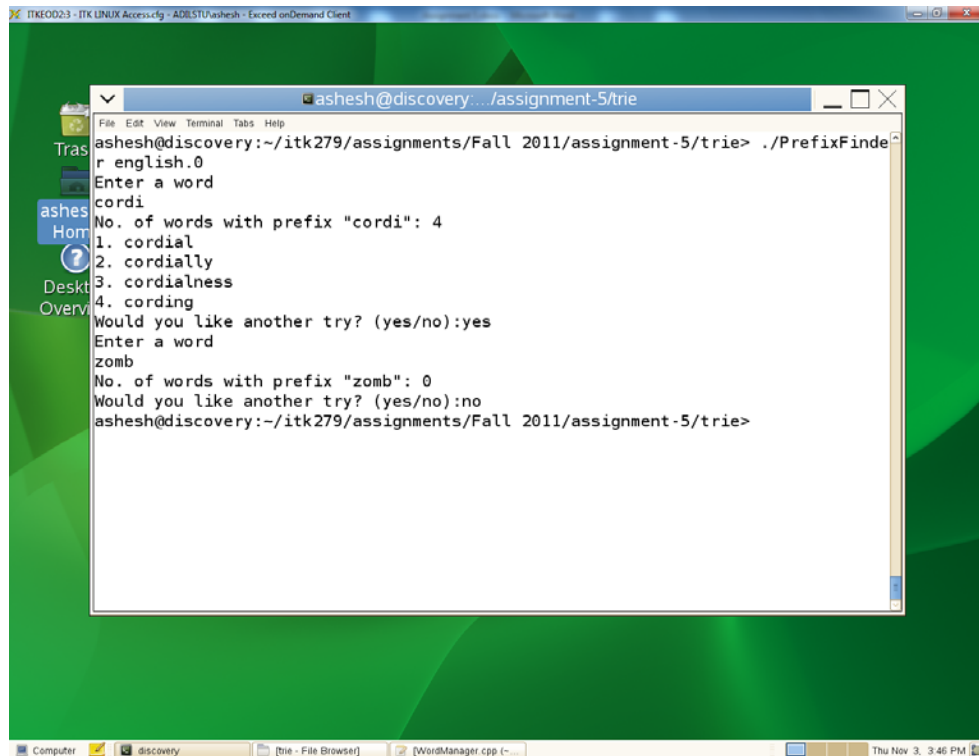
The various functions are:

1. An operator []: Given a key, this should return a reference to its corresponding value. If no such entry exists, the map should create a new entry with the provided key and a default value, and return a reference to it.
2. bool find(key): Given a key, find whether there is a (key,value) pair in the map that has the same key. If it exists, return true otherwise return false.
3. getKeys(): This returns all the keys currently existing in the map in a vector. This is called so that one can iterate over the keys to iterate through the map.

What to do

1. An empty STLMap class has been provided to you. Implement all its functions using a STL map. Make sure the behavior of all functions is exactly as described above.

Now your program should compile, link and run successfully. An example run would be as follows:



```
ITK0023 - ITK LINUX Access.clg - ADISTU\ashesh - Exceed onDemand Client
ashesh@discovery:~/assignment-5/trie
File Edit View Terminal Tabs Help
ashesh@discovery:~/itk279/assignments/Fall 2011/assignment-5/trie> ./PrefixFinde
r english.0
Enter a word
cordi
No. of words with prefix "cordi": 4
1. cordial
2. cordially
3. cordialness
4. cording
Would you like another try? (yes/no):yes
Enter a word
zomb
No. of words with prefix "zomb": 0
Would you like another try? (yes/no):no
ashesh@discovery:~/itk279/assignments/Fall 2011/assignment-5/trie>
```

2. Implement another map class “AVLMap” that is based on an AVL tree. This should be a template class like the STLMap. Each node of the tree will store a key and a value. The tree itself is arranged and searched based on the keys at the nodes. Make sure that the only public functions of the AVLMap class are those specified by the MapInterface class. You are free to write other private functions or different classes.

When you have implemented it, the only changes to your program should be to create objects of your class in the constructors of the Node and the WordManager class, instead of the STLMap!

What to submit

Please submit all .cpp and .h files (including the ones provided to you), the Makefile and the data file in a single zipped file using Reggienet.