Lab 9: Scrabble Cheater

Due: April 23th at 11:59pm

Overview

The goal of this lab is to implement a separately chained hash table to build a scrabble cheating software. Additionally, you will make use of C++ Standard Template Libraries list implementation (vectors) for your implementation, and you will be required to write appropriate test files to ensure the correctness of your hash table implementation. **This is a group lab, groups of two, but you must work with someone in your lab section.**

Deliverables

Your submission should include requisite code to run your program, but the following files will be the focus of grading:

- `Makefile`
- `README`
- `dictionary.h`
- `hash_table.[h|inl`
- `scrabble_cheat.cpp`
- `test_hash.cpp`
- `library/`
- `TWL06.txt`

All starter code is provided via `update35`.

Submission

You will submit using `handin35`. Be sure to place all relevant files in `cs35/labs/09`.

Scrabble Cheater

Scrabble is a popular board game based on a players ability to generate words from a given set of letters (called tiles). While there are more details needed to learn the game, all that you need to know for this assignment is that a player may only play actual words. That is, a legal play is is a word that exists in the special Scrabble dictionary, which consists of most American English words of at least two letters in length without proper nouns. The official Scrabble word list is provided for you (`TWL06.txt`). An important scrabble skill is the ability to generate a list of possible plays given a finite set of letters. The best scrabble players in the world do this without any assistance, but we are not the "best" scrabble players ... and we know a thing or two about computers.

Your goal in this lab is to program Scrabble cheating software, `scrabble_cheat` which can both look up a word in the Scrabble dictionary and provide a list of possible plays based on a set of input letters. Of course, you would never use this to cheat in a real game of Scrabble, and if you do, I expect you to not get caught. Below is some sample output from running the cheater:

```
$ ./scrabble_cheat TWL06.txt
These are you options:

        (0) Exit (or Ctrl^D)
        (1) See if a play is legal
        (2) Find all legal plays for a set of tiles

Which option would you like? 1

Enter the word you wish to play: college
Congratulations, you have found a legal play!

These are you options:

        (0) Exit (or Ctrl^D)
        (1) See if a play is legal
        (2) Find all legal plays for a set of tiles

Which option would you like? 2

Enter the letters you wish to play: batman
Your legal plays are:
        batman
        bantam

These are you options:

        (0) Exit (or Ctrl^D)
        (1) See if a play is legal
        (2) Find all legal plays for a set of tiles

Which option would you like? (CTRL^D)
```

Starter's Abstract As usual, this is an involved lab requiring you to touch many pieces of code, so below is an abstract of the work required. A more complete description of each of the provided files is found at the end of this document.

1. Complete the implementation of the hash table in `hash_table.[h|inl]`
2. Write a sequence of tests in `test_hash.cpp` to verify your implementation of the hash table.

3. Complete basic word check in `scrabble_cheat.cpp` requiring reading user input and loading the dictionary.
4. Add anagram solving to `scrabble_cheat.cpp`
5. (**Challenge**) Subscripting operator for hash table
6. (**Challenge**) Add options for searching with a blank tile
7. (**Challenge**) Order results by highest word score
8. (**Extra Credit**) Add functionality for full permutation anagrams. Given 7 tiles, the program returns all words of any length that could be generated using those 7 tiles, including blank tiles, and orders the results by word score.

Completing the Hash Table

The first part of the lab requires you to complete the HashTable class and test it in `test_hash.cpp`. Note that the HashTable uses separate chaining and maintains an array of STL vectors. A vector is C++'s version of an ArrayList. It is described in more detail in a later section. Below is the basic declaration of HashTable

```
template <typename K, typename V>
class HashTable : public Dictionary<K,V> {
  private:
    std::vector<Pair<K,V> >* table;

    int size;

    int capacity;

    double MAX_ALPHA;
//...
```

Note that the type of `table` defines a pointer to an array of vectors, not a pointer to a single vector. Further the vectors store pairs of key-values. Additionally, `MAX_ALPHA` describes the maximum load on the hash table before the capacity must be expanded.

Finally, the constructor for the HashTable uses default values:

```
HashTable(int capacity = 53, double maxLoadFactor = 0.8);
```

This functions much like default values in python. The user may provided two parameters, the capacity and the maximal load, but if the user omits these parameters, the values 53 and 0.8, respectively, will be used. This is a standard technique in C++ for specifying optional parameters.

Scrabble Checking and Anagrams

There are two parts to the scrabble cheater. The first part will look up words directly in the hash table; that is, provided a word, is it in the dictionary? The second will take a word and find all anagrams of the word that are legal plays. You will solve both parts using dictionaries. There is limited further details on the implementation as you are encouraged to use creativity in solving these problems.

As a hint, you should probably store the word list in two different dictionaries, one to perform basic word-list look-up and one specially designed for handling anagrams. For the anagram dictionary, one thing to consider in the design is that you want to look up a string as a key and return a list of all anagrams of that key. How you build such a dictionary is the challenge.

Vectors

Vectors are expandable lists that come standard in C++. They are most akin to the Circular Array List implementation. Vectors are well documented online, and I encourage you to read the documentation there. Here, I provide a very basic overview of the operations.

First off, for access items in a vector, you can use standard `[ ]` operators, like in arrays. You can also set values using the `[ ]` operator. Vectors have special iterators that indicate at which point an operation should occur on the vector, such as assigning to a location in the vector. Iterators are also used to iterate over the vector in a for loop like matter. You can also iterate using standard index based iteration in a for loop. Vectors come with a standard `push` and `pop` operations, named `push_back()` and `pop_back()` since they operate at the back of the vector. For further details, read the STL reference documentation: `http://www.cplusplus.com/reference/vector/vector/`

Handling User Input

Your program should handle user input, both from the command line and within the program appropriately. That is, when a user provides a bad input, say he/she forget to include the dictionary file on the command line, an error should be reported and useful information provided. Similarly, within the program, if a user provides a bad choice or selection in the menu, you should report an error and allow the user to select again. **Your program should never crash due to bad input, and if it cannot proceed, useful information should be reported**.

Error Handling

You should carefully consider how errors should be handled in your program, particularly for your hash table. Throwing an exception is the logical choice; identifying where and when to throw an exception is an important part of this lab.

Grading and Extra Credit

Rubric

This lab will be graded on a 50 point scale:

- (15 Points) Hash Implementation
- (10 Points) Hash Test routines
- (15 Points) Scrabble Cheater
- (10 Points) Coding Style and Memory

Note that there are points dedicated to aspects of your lab that are not part of the solution. Do pay attention to coding style and managing memory, design and lay out your code effectively so that it is readable and easily debugged. Run your program through `valgrind` and plug any memory leaks.

Challenge and Extra Credit

There are three relatively easy challenge problems and a single extra credit. To receive the extra credit, you must complete all the challenges as well.

- (**Challenge**) Add subscripting operator to your hash table so that you can get and set key-values of the hash table using the `[ ]` operators. For example, consider the `get()` function for CircArrayList:

```
template <typename T>
T CircArrayList::getint i) {
  if (i < 0 || i >= size) {
    throw std::runtime_error("Attempted to index out of bounds.");
  }
  return values[(head+i)%capacity];
}
```

  We can redefine this operation using the subscription like so:

```
template <typename T>
T& CircularArrayList::operator[](int i) {
  if (i < 0 || i >= size) {
    throw std::runtime_error("Attempted to index out of bounds");
  }
  return values[(head+i)%capacity];
}
```

  Notice that this looks almost exactly the same. The main difference is in the return value. `get()` returns a value of type `T`. Just as we pass-by-value in C++, by default we return-by-value. That means that the value returned is a copy of the original item stored in the `CircArrayList`.

  The subscript operator is used to get values, and thus the `get()` return-by-value would be suitable. But we also wand to use indexing to modify or set values e.g., `list[i] = 7` To do this, we need to have access to the actual piece of memory where the data is stored. So, we return a data type of `T&` . This tells C++ not to return a copy, but to return the actual chunk of memory where the data is stored so that it can be modified.

- (**Challenge**) Allow users to enter in a blank tile using "-" (a hyphen) for doing anagrams. A blank tile can be any letter.

- (**Challenge**) Order results by highest word score in scrabble. Each tile has an associated score value; thus each word has a total score which is the sum of the scores on the tiles. Sort the results using these metric.

- (**Extra Credit**) Add functionality for full permutation anagrams, so given a 7 tiles, all words of any length that is can be created using any of those 7 tiles, including blanks, is returns to the user and sorted by the word score.

Provided Code

- `Makefile` : The makefile. You may need to edit this file if you add new code to your project.
- `README` : The README file. You will need to edit this file.
- `dictionary.h` : The header file declaring the abstract/pure-virtual dictionary. You will not need to edit this file, but you should read it.
- `hash_table.[h|inl` : The header and inline implementation of the hash table. You will need to edit the inline implementation, and you should read the header file.
- `scrabble_cheat.cpp` : The scrabble cheat program. You will need to edit this file.
- `test_hash.cpp` : The hash table test program. You will need to edit this file.
- `library/` : The library of previously implemented data structures. You do not need to edit this direcrory, but you may copy any previously implemented data structures here if you want access to them.
- `TWL06.txt` : The official scrabble word list from 2006.

Compilation

Compilation will occur using `make`. To compile your program

```
bash> make
```

If you add extra functionality contained in other files, you should edit the `Makefile`, but it is not necessary. To compile the test code:

```
bash> make test
```

Which will compile an executable `test_hash` at first, you can add more tests as you develop.

Execution

```
>./sorted_search
Usage: ./scrabble_cheat <word-list>
Required options:
   <word-list>:  List of words in the scrabble dictionary
```