CSE 232 Spring 2017

Programming Project #9

Assignment Overview

This project focuses on the use of classes. It is worth 60 points (6% of your overall grade). It is due Monday 10/20 before midnight. That's ***two weeks*** because of the midterm on 10/07.

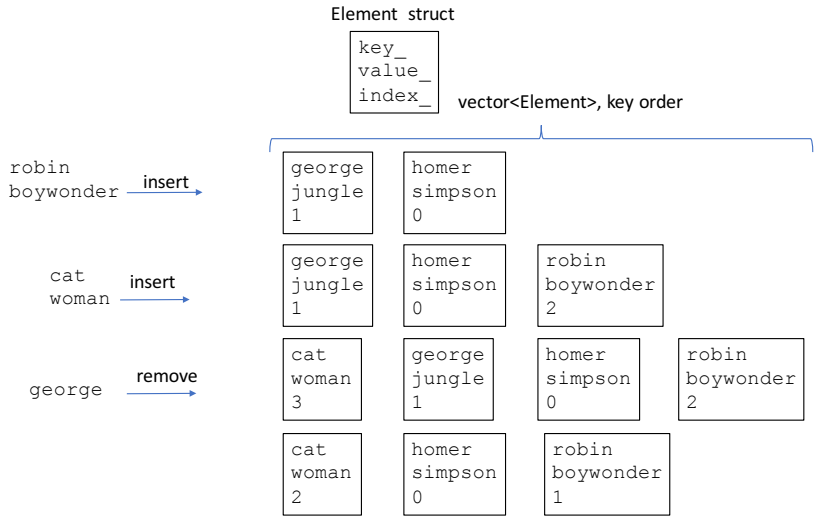
The Problem

You have done work with an STL map, a way to map a key to a value. But there are some restrictions on a map. You can search a map for a key (and then the associated value) but you cannot search a map for a value (and then the associated key). Furthermore, because a map is kept in a particular order, an ordering based on the keys, we cannot know the order of insertion of each element. We are going to fix all that with a new STL-like data structure, the `TriMap`.

Basic Premise

The underlying implementation of an STL map is such that search for a key (which is all you are allowed to do) is very fast. That underlying data structure is not fixed by the STL implementers, but it is likely something like a red-black tree (look it up). That is quite beyond us at this stage.

However, we can keep a vector as the underlying data structure and keep that vector in key-sorted order for fairly fast lookup. That is what we are going to do! We are also going to make a `struct` called `Element` that will store the key, the associated value and the insertion number (called the index here) indicating the order of placing an `Element` in the data structure.



`Element` is shown as a key-value-index triple (a struct). We maintain a `vector<Element>` in key order. When an insert occurs, the new element is placed in the vector so that key-order is maintained. In this way, we can quickly do a search through the vector to find a key (or note that it does not exist) using a binary search (but see below on how to do that). However, with this data structure we can also do a search for an index or a value, though we can only do it by a linear search.

Like a map, `TriMap` only allows one example of a key. That is, only one Element in the TriMap can have a particular key. No duplicates allowed.

Class Element
The class `Element` is listed below. It contains the three data members of Element as described. It is a class because we need `key_` and `index_` to be private while `value_` is public. Note the data member names have an underline as a suffix to indicate class data members. Though we can allow an update to a particular `Element`'s value, changing the key or index within the `vector<Element>` (see below) would ruin our setup (ruin the key-order, modify the index of insertion).

```
class Element{
 private:
  string key_;
  size_t index_ = 0;

 public:
  string value_;

  Element()=default;
  Element(string k, string v, long i): key_(k),  index_(i), value_(v) {} ;

  friend class TriMap;
  friend ostream& operator<<(ostream&, const Element&);
};
```

Provided in the header are a default constructor and a 3-parameter constructor. We also make the class `TriMap` a friend of `Element`, so that the `TriMap` class can access `Element` private data members

Assignment:
You must write the function (it's a ***function***, not a method, there is an Element in the parameter list), `operator<<` Format is `key:value:index` . See Mimir test cases.

Class TriMap
The class TriMap is shown below. It contains a private `vector<Element> vec_` , Elements in key-order, and a `size_t sz_` (note the underlines after the data member names), the number of elements in the vector.

```
class TriMap {
      private:
        vector<Element> vec_;
        size_t sz_ = 0;

      public:
        TriMap() = default;
        TriMap(const Element&);
        TriMap(initializer_list<Element>);

        size_t size();
        bool insert(string,string);
        bool remove(string);
        Element* find_key(const string&);
        Element* find_value(const string&);
        Element* find_index(long);

        friend ostream& operator<<(ostream&, const TriMap&);
};
```

Assignment

Writing the details of TriMap is where most of the work will be.

- `size` class method. No arguments, returns a `size_t`.
  - The number of Elements in the underlying vector.
- `insert` class method.
  - Arguments: `string key` and `string value` of a new Element to insert in the vector
  - `bool` return
  - if the key does not already exist in the underlying vector, it inserts a new Element into the vector in key-order.
    - the inserted Element will have the key, the value and the proper insertion value (which element this is in terms of insertion order)
    - returns true
  - if the key does exist, no action is taken
    - return is false
- `remove` class method.
  - One argument, the `string key` of the `Element` to remove
  - `bool` return
  - if the `Element` with the key is in `vec_` then it is removed.
    - after removal, the `index_` values of `Elements` is updated appropriately (see the Figure)
    - returns true
  - if the `Element` key is not in `vec_`, no action is taken
    - returns false
- `find_key` class method.
  - One argument, the `string key` to find in the underlying `vec_`
  - `Element*` return
  - If the `Element` with the key is found in `vec_, Element*` is returned
    - this is tricky. If you use an algorithm it returns a vector<Element>::iterator, which is ***not*** a pointer (`Element*`) even though it acts like one. To convert an iterator `pos` to a pointer, return `&(*pos)` a pointer (deref the iterator, address of that Element)
  - If the Element is not found, return `nullptr`
  - ***Must use binary search***, as provided by `upper_bounds` or other algorithms.
- `find_value` and `find_index` class methods
  - both take one argument, a `string value` or a `size_t index`
  - Using a linear search, locate the Element with the `value_`/`index_` and return an `Element*`
  - If the Element* cannot be found, return `nullptr`
- `operator<<,` a ***function*** (not a class method, notice the TriMap argument)
  - print the TriMap (see Mimir tests for format)

Deliverables

`proj09_trimap.cpp` -- your completion of the class specs based on `proj09_trimap.h` (both provided).

1. Remember to include your section, the date, project number and comments.

Notes

1. To show how you can use some of the html algorithms, I provided some code in `algorithms.cpp` with comments. I think that will be of some help.