CSE 232 Spring 2017

Programming Project #11

Assignment Overview

This project focuses on the use of classes. It is worth 65 points (6.5% of your overall grade). It is due Monday 12/06 before midnight. That is the Wed of the last week of class.

The Problem

You did a `TriMap` in Project 9, and we're going to do it again with a variation. It is going to be a templated class and it is going to use as its underlying representation a linked list (instead of a dynamically allocated array).

Class Element

The variation of the class `Element` is listed below. It contains two templated class data members `key_` and `value_` As before, `key_` is private and `value_` is public. The `key_` is templated on the first type, which I have called `K`, and `value_` is templated on the second type which I have called `V` . There is also a `size_t index_` to help remember which entry this `Element` is in the `TriMap`.

Different is the `Element *next_` member. This is a pointer to the next `Element`, necessary for making linked list of `Element`.

```
template <typename K, typename V>
class Element{
private:
  K key_;
  size_t index_ = 0;
  Element *next_ = nullptr;
public:
  V value_;

  Element() = default;
  Element(K key, V val, size_t i) : key_(key),index_(i),
                                    next_(nullptr), value_(val) {};

  friend ostream& operator<<(ostream& out, Element& e){

  // replace this with your own code

}
  friend class TriMap<K,V>;
};
```

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.

You must write your code for the `operator<<` right there in the class definition where "`replace this with your own code`" appears.

Class TriMap

The class TriMap is shown below. It has two pointers: head_ which points to the first Element in the linked list and tail_ which points to the last. Initially they both point to nullptr (to nothing).

```
template<typename K, typename V>
class TriMap{
 private:
  Element<K,V> *head_ = nullptr;
  Element<K,V> *tail_ = nullptr;
  size_t index_ = 0;
  void print_list(ostream& out);
 public:
  TriMap() = default;
  TriMap(K, V);
  TriMap(const TriMap&);
  TriMap& operator=(TriMap);
  ~TriMap();
  bool insert(K,V);
  bool remove(K);
  Element<K,V>* find_key(K);
  Element<K,V>* find_value(V);
  Element<K,V>* find_index(size_t);


  friend ostream& operator<<(ostream& out, TriMap<K,V>& sl){
    sl.print_list(out);
    return out;
  };
};
```

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.
  - Templated arguments: `K key` and `V value` of a new `Element<K,V>` to insert in the linked list
  - `bool` return
  - if the key does not already exist in the underlying list, it inserts a new `Element<K,V>`into the list in key-order.
    - the inserted `Element<K,V>` 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 `K key` of the `Element<K,V>` to remove
  - `bool` return
  - if the `Element` with the key is in the list then it is removed.
    - after removal, the `index_` values of the `Element<K,V>` is updated appropriately (see the Figure)
    - returns true

    - if the `Element` key is not in the list no action is taken
        - returns false
- `find_key` class method.
    - One argument, the `K key` to find in the underlying list.
    - `Element<K,V>*` return
    - If the `Element<K,V>` with the key is found in the list, `Element<K,V>*` is returned
    - If the Element is not found, return `nullptr`
    - ***You cannot use binary search in a singly linked list***, so you are free to do a linear search.
- `find_value` and `find_index` class methods
    - both take one argument, a `V value` or a `size_t index`
    - Using a linear search, locate the Element with the `value_/index_` and return an `Element<K,V>*`
    - If the Element cannot be found, return `nullptr`
- `template<typename K, typename V>`
  `void TriMap<K,V>::print_list(ostream &out)`
    - ***function*** (not a class method, notice the TriMap argument)
    - print the TriMap (see Mimir tests for format)

Deliverables

`proj11/proj11_trimap.h` -- your completion of the class specs to Mimir

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

Notes

To make things easier I gave you a `proj11_skeleton.h`, which is all the declarations and the beginning of the definitions of the two classes. You can copy that directly to `proj11_trimap.h` and begin your work. *Look the comments*, they are there to be helpful.