CSE 330 LABORATORY -- Week 8, Spring 2018

Instructor: Kerstin Voigt

In this lab we will implement the ADT Map in terms of a "set of pairs." The Set ADT, in turn, is essentially a BinarySearchTree data structure with several modifications that are meant to support Map.

Exercise 1: Start by obtaining a copy of Set.h from the instructors Linux directory. In its current form, this file originates from our earlier BinarySearchTree.h.

Modify the insert() member functions (both, public ("external") and private ("internal") so that instead of void both functions will returen a iterator to the newly inserted item.

Exercise 2: Create a new header file Pair.h for a template Pair<K,V> as shown below.

```
#ifndef PAIR_H_
#define PAIR_H_
using namespace std;

template <typename K, typename V>
class Pair
{
 public:

  Pair() {}

  Pair(K thekey)
    :first(thekey) {}
  Pair(K thekey, V theval)
    : first(thekey), second(theval) {}

  Pair(const Pair& rhs)
    : first(rhs.first), second(rhs.second) {}

  bool operator == (const Pair<K,V>& rhs) const
  {
    return  first == rhs.first;
  }

  bool operator != (const Pair<K,V>& rhs) const
  {
    return first != rhs.first;
  }


  bool operator < (const Pair<K,V>& rhs) const
  {
    return first < rhs.first;
  }
```

```
  bool operator > (const Pair<K,V>& rhs) const
  {
    return first > rhs.first;
  }

  K first;
  V second;
};

#endif
```

In C++, a std::pair structure already exists. However, what we need for Map is a pair that compares its instances based on the the first component only. In order to have full control over our pair structure, we define our own.

Exercise 3: You are now ready to implement the ADT Map. Notice how it adapts a Set data structure of Pairs. Enter the following into a header file Map.h.

```
#ifndef MAP_H_
#define MAP_H_

#include "Pair.h"
#include "Set.h"     // must have insert that returns iterator!!!
using namespace std;

template <typename K, typename V>
class Map
{
public:
  Map() {}

  void printMap() const
  {
    typename Set<Pair<K,V> >::iterator itr = themap.begin();
   for (; itr != themap.end(); ++itr)
      {
      cout << (*itr).first << ":" << (*itr).second << endl;
      }
    return;
  }

  V & operator [](K index)
    {
      typename Set<Pair<K,V> >::iterator here;
      Pair<K,V> probe(index, V());
      here = themap.insert(probe);
      return (*here).second;
    }

  void remove(K & key)
  {
    Pair<K,V> probe;
    probe.first = key;
    themap.remove(probe);
    return;
```

```
  }

 private:

  Set<Pair<K,V> > themap;
};

#endif
```

Exercise 4: Test your new Map data structure with an int main() test stub of your choice. The test should include the entering of data items of some value type V, associated with keys of some other data type K. E.g.,

```
Map<string,int> basket;
basket[“apple”] = 5;
basket[“plum”] = 7;
…
```

Once entered, let the program output the contents of the Map, in order to verify that items were stored as intended. E.g.,

```
apple:5
plum: 7
…
```

Next test the removal of an item. E.g.,

```
basket.remove(“apple”);
```

Output the content of the Map after removel to verify that the item has been removed.

Finally test the change of a value under a given key. E.g.,

```
basket[“plum”] = 3;
```

Output the content of the Map to verify the new value.

Exercise 5: If there is time left … add to class Map two additional member functions: (1) a function that will return a vector of all keys that are used to index values in the Map; (2) a fucntion that will return a vector of all values that are listed under keys in the Map.

Credit for this lab: (1) After working diligently on the above, sign up on the signup sheet. **(2) email to Sarthak by midnight, Wednesday, May 23, your best effort at solving Exercises 1-4.**