A3: Storing Keys and Values

Overview

You are tasked with creating a data structure to store pairs of data. This basic structure that we are using has a number of applications, including graph representations (adjacency lists) and hash tables with chaining (though not identically to what we are implementing).

Deliverables

- **A3.tar** will be submitted to A3 assignment on Autolab.
    - This will contain your A3.hpp and A3.cpp files (and A3-bonus.cpp and A3-bonus.hpp)
    - All other files will be overwritten, so ensure all your code is contained in your A3.hpp and A3.cpp files. Make sure they are not within any folders.
- **A3-analysis.pdf** will be submitted to A3 Analysis assignment on Autolab.
    - This will be a PDF with pseudocode and an analysis of runtime costs for the copy assignment operator.
    - We recommend typing this.
    - **If you submit a .docx or any other file format you will receive no credit. You can easily save a .docx file as a pdf. Any file that is not a pdf will not be graded.**
    - **This will be due the same day as your code submission.**

Due Date

DUE DATE:

A3.tar and A3-analysis.pdf: 3/19/2017, at 01:59 (this is Sunday, March 19 at 1:59AM, or for those who stay up late, the continuation of Saturday night).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****

Start Early!!!!!!!!

Late Policy

The policy for late submissions is as follows:

- Before the deadline: 100% of what you earn (from your best submission).
- One day late: 50-point penalty (lower your earned points by 50, minimum of 0).
- > 1 days late: 0 points.

Keep track of the time if you are working up until the deadline. Submissions become late after the set deadline. If you are working late, keep in mind that **submissions will close 24 hours after the original deadline** and you will no longer be able to submit your code.

Objectives

In this assignment, you will create a vector of lists of integers along with some basic operations. Additionally, you will create an iterator that traverses across the lists, i.e., traverses the first element in each list, then the second in each list, and so on.

Useful Resources

Review the lecture notes and provided examples to get an idea for using lists and vectors.

Instructions

- Download the file `A3-skeleton.tar` and extract the skeleton files. **Rename the skeleton files appropriately.**
- A `CMakeLists.txt` file is included to properly build your program. You may use this by Importing the folder into CLion, or from the command line by the following commands:

```
cmake .
make
```

- You will find the function declarations in `A3.hpp` and the definitions in `A3.cpp`. You may add any member variables or helper methods/function you wish add to the `A3.hpp` file. All function/method definitions should be in `A3.cpp`.
- **You may not use any other standard libraries within your code aside from vector**.

Required:

Complete the functionality for the PairList class and associated Iterator. You will lose credit if your code leaks memory.

PairList Class:

```
/**
 * Constructor to set the initial size of the number of keys.
 *    The number of keys is fixed during the lifetime of the PairList.
 *
 * @param numKeys - number of keys to allocate for.
 */
PairList(int numKeys);
```

This constructor should set the initial size for the internal storage vector. You may also initialize any other values you wish.

```
/**
 * Copy constructor:
 *    Create a deep copy of the other PairList.
 * @param other - other PairList to copy from.
 */
PairList(const PairList& other);
```

This constructor should perform a deep copy of the other object (see lecture notes).

```
   /**
    * Copy assignment operator:
    *    Create a deep copy of the other PairList.
    * @param other - other PairList to copy from.
    * @return reference to this.
    */
   PairList& operator=(const PairList& other);
```

This operator should perform a deep copy of the other object into this object (see lecture notes).

```
   /**
    * Destructor:
    *    Clean up all memory used.
    */
   ~PairList();
```

You must clean up any memory used/managed by this data structure. Memory leaks will result in a lower score.

```
   /**
    * insert:
    *    insert the value stored in pair under the list
    *    at location indexed by key, maintaining the list in increasing
sorted order.
    *
    * @param pair (key,value)
    * @return true: value was inserted into list under key.
    *         false: value was unable to be inserted
    *                ** Key didn't exist.
    *                ** Value already existed in list under key.
    */
   bool insert (const Pair& pair);
```

The **insert** method will take as input a pair, which contains a key and value. The key references the index in the vector _keys to insert the value into. The value should be added to the stored list in sorted order. The stored lists should be kept in increasing sorted order. Duplicate values should not be inserted. If the value was successfully inserted, return true. Otherwise, return false.

```
   /**
    * contains:
    *    determines whether the (key,value) pair is stored in this list.
    *
    * @param pair (key,value)
    * @return true: value was in the list under key.
    *         false: key didn't exist or value was not in list under key.
    */
   bool contains (const Pair& pair) const;
```

The **contains** method should return true if the list stored in index key contains value. If the index is outside of the range of keys or value is not in the list under the key, then this method should return false.

```
/**
 * find:
 *    determines whether the (key,value) pair is stored in this list
 *    and returns an iterator to the pair if found.
 *
 * @param pair (key,value)
 * @return Iterator referencing the pair.
 *         If not found, return end().
 */
Iterator find (const Pair& pair);
```

The **find** method should return an iterator to the pair value if found. If the pair is not found, then return end iterator.

```
/**
 * remove:
 *    removes the pair from the representation.
 *
 * @param pair (key,value)
 * @return true: value was in the list under key and was removed.
 *         false: key didn't exist or value was not in list under key.
 */
bool remove (const Pair& pair);
```

The function **remove** should remove the value from the list stored at index key. If the value was successfully removed, return true. If the key was invalid or the list at key did not contain value, remove should return false.

```
/**
 *  associatedList:
 *     return a list of pairs associated with the requested key.
 * @param key
 * @return nullptr: key was invalid or no data stored.
 *         PairNode* pointing to first element in generated list of pairs.
 */
PairNode* associatedList(const int& key) const;
```

The function **associatedList** should return a list containing all the pairs stored at index key. Return **nullptr** if the key was invalid or no pairs were stored under that key.

```
/**
    * begin:
    *    return an iterator to the first pair in the data structure.
    * @return iterator to first pair stored.
```

```
    */
   Iterator begin();
```

Return an iterator to the first pair stored.

```
   /**
    * end:
    *    return an iterator to just after the last pair in the data
    *    structure.
    * @return iterator to "beyond the last pair"
    */
  Iterator end();
```

Return an iterator to the end of the data structure. This should not be located at any pair that is stored.

PairList Iterator Class:

The iterator for this PairList class should start at the smallest key and first element in the list. Incrementing should occur across the first item in the list for each key, before moving on to the second item in each list, and so on. Note that the lists can be of differing lengths. You will probably need to add member variables to your Iterator class to store the state.

```
     /** Prefix incrcement:
       *    Move to the next element stored,
       *    in correct order.
       *
       *    @return Reference to updated iterator.
      */
      Iterator& operator++();
```

Increment the iterator to the next state. This is the prefix increment.

```
      /** Postfix increment:
       *    Move to the next element stored,
       *    in correct order.
       *
       *    @return Copy of current iterator before increment.
      */
      Iterator operator++(int);
```

Increment the iterator to the next element. This is the postfix increment. You need to return a copy of the original iterator before incrementing.

```
      /** Indirection operator:
```

```
     *    Access value in position of iterator.
     *
     *    @return Pair representing data stored in structure.
    */
    Pair operator*();
```

This should return the pair that is referenced by the Iterator.

```
    /** Equality operator:
     *    Check if two iterators are representing the same position
     *    in the same PairList.
     *
     *    @return true: same PairList and same position.
     *            false: different PairList or position.
    */
    bool operator==(const Iterator& rhs) const;
```

This should return true if the two Iterators are in the same state, and false otherwise.

```
    /** Inequality operator:
     *    Check if two iterators are representing the different
     *    positions or iterate different PairLists.
     *
     *    @return true: different PairList or position.
     *            false: same PairList and same position.
    */
    bool operator!=(const Iterator& rhs) const;
```

This should return true if the two Iterators are in a different state, and false otherwise.

Bonus (10 points): Reversing the Iterator

Create a copy of your files and name them **A3-bonus.hpp** and **A3-bonus.cpp** for this functionality. Include these new files in the same **A3.tar** submission.

The reason for separate files is there could be quite a large amount of changes to be made to your code. For this bonus portion, feel free to use any other standard libraries you wish.

Add a prefix decrement operator to the iterator for the PairList. This should go in reverse order, starting with the last element. You will need to add four functions:

```
Iterator PairList::r_begin();
Iterator PairList::r_end();
```

```
Iterator& PairList::Iterator::operator--();
Iterator PairList::Iterator::operator--(int);
```

- **r_begin** should return an Iterator to the last pair stored.
- **r_end** should return an Iterator to just past the first pair (the state you would end up in from calling pl.begin()--;).
- Define the prefix and postfix -- operators.

Notes on this bonus portion:

- Don't add these changes to your original files (especially since extra headers will block your code from being graded).
- You may use additional standard libraries.
- You may want to modify the Node class.

Testing

To test your code, a main function in **driver.cpp** that inserts a number of pairs has been provided. This inserts a number of pairs and prints out the order they are traversed in as well as what is expected.

Submission

Code Submission (90 points):

You must submit your A3.tar file to the A3 assignment on Autolab.

- Make sure you test extraction of the files prior to submission to ensure all files are present.
- Your submission must compile and run on Autolab. I suggest that you compile often as you build your code to avoid difficulties debugging syntax and other errors.

Tests will be run on your file and your score will be reported to you when finished. For this assignment there is not limit on the number of submissions you may perform. That said, you should not be using the submission system to test and debug your code.

Runtime Bonus (+10 points):

If you pass all tests for the decrement iterator and your runtime is faster than our code's runtime, you will receive 10 points bonus.

Code Analysis Submission (10 points):

You must submit a pdf with pseudocode and a runtime analysis of the copy constructor. This submission should be uploaded to the A3-analysis assignment. The due date for the analysis will be the same as the assignment due date.

DUE DATE:

A3.tar and A3-analysis.pdf: 3/19/2017, at 01:59 (this is Sunday, March 19 at 1:59AM, or for those who stay up late, the continuation of Saturday night).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****