Project 7: Fifo Map

Due: Monday June 18, 11:59 pm

1 Assignment Overview

This project focuses on implementing classes. You will create a hash map that allows for fast searches. Additionally, you will create an iterator for it that visits items *in the order they were inserted*.

2 Assignment Deliverables

You must turn in completed versions of the following files:

- proj07/FifoMap.cpp
- proj07/FifoMap.h

Be sure to use the specified file name and to submit your files for grading via **Mimir** before the project deadline. A partial implementation of the FifoMap.h file can be found on Mimir.

3 Background

C++ includes two dictionary structures, a tree-based (ordered) `map` and an hash-based `unordered_map`. The `unordered_map` is analogous to the `dict` class that you used in Python. Python also has an `OrderedDict` class. Instead of keeping the items in a sorted order (like the C++ `map` class), the `OrderedDict` allows the items to be visited in *the order that they were inserted*. For this project you will be creating a `FifoMap` class which functions like an `unordered_map` except that its iterator will visit the items in the order they were inserted, like the Python `OrderedDict`.

This can be done by including an ancillary structure that stores keys in iteration order. This structure needs to be able to quickly remove keys, which disqualifies `vector` as it can only quickly remove the last item in the sequence. A better option is a doubly-linked `list`, which allows for elements to be rapidly removed as long as you have a pointer to the item. You can do this by keeping a map of keys to iterators as well as keys to values.

C++ allows for the declaration of nested classes. Theses are classes that are declared within the body of another class. Such an inner class is automatically a `friend` and thus has access to the `private` members of the enclosing class. However, it is not associated with any particular instance of the enclosing class. If you need a particular `FifoMap` instance you need to supply it to your iterator.

4 Assignment Specifications

For this project you will be writing two classes; a `FifoMap` which behaves similarly to a Python `OrderedDict` and an inner `Iterator` class. You will not be required to provide all of the functionality included in other maps; only basic ability to insert and remove items will be required and your iterator will be a forward-only

instead of a bi-directional. You will declare both classes in the `FifoMap.h` file and provide implementation in the `FifoMap.cpp` file.

In this and in future projects we will provide *exactly* our function specifications: the function name, its return type, its arguments, and each arguments type. The functions will be tested individually in Mimir using these exact function specifications. If you do not follow the function specifications, these independent tests of your functions will fail. Do not change the function declarations!

What you test on Mimir is a file that contains only the functions. You do not turn in a main program. We can test the functions individually on Mimir. However, you should write your own main program to test your functions separate from Mimir. It is more flexible and you can debug more easily.

4.1 FifoMap

The `FifoMap` class will have the following methods. Most of the operations behave similarly to an `unordered_map`.

function `Default Constructr`
params:
returns: Nothing (constructor)

function `Copy Constructor`
params: map : const FifoMap&
returns: Nothing (constructor)

function `size const`
params:
returns: The number of associations

function `empty const`
params:
returns: If there are no association

function `count const`
params: key : const string&
returns: The number of occurrences of the key

function `clear noexcept`
params:
returns: void. Removes all associations

function `operator[]`
params: key : const string&
returns: A reference to the value associated with the given key, creating one if necessary.

function `operator=`
params: map : const FifoMap&
returns: *this. Copy assignment.

function `erase`
params: key : const string&
returns: void. Removes the association from the map.

function `begin()`
params:
returns: An iterator to the first element

function `end()`
params:
returns: An iterator past the last element

4.2 Iterator

The inner `Iterator` class will have the following methods:

function `Copy Constructor`
params: itr : const Iterator&
returns: Nothing (constructor)
function `operator=`
params: itr : const Iterator&
returns: *this. Copy assignment.
function `operator++`
params:
returns: *this. Advances iterator.
function `operator== const`
params: itr : const Iterator&
returns: If the two iterators point to the same item
function `operator!= const`
params: itr : const iterator&
returns: If the two iterators point to different items
function `operator* const noexcept`
params:
returns: A reference to an association pair
function `operator-> const noexcept`
params:
returns: A pointer to an association pair

5 Assignment Notes

- You will receive no points if your solution does not compile on Mimir.
- Points will be deducted if your solution has any compiler warnings.
- Points will be deducted if you include `using namespace std`.
- You *are* allowed to use other classes as part of your solution, including `unordered_map`.
- The `list` class supports fast deletions as long as you have an iterator to the item to be deleted.
- If a key has a binding replaced, its order in the traversal remains unchanged.
- Most of the operations can easily and succinctly be expressed in terms of these other existing classes.
- You may add any additional fields or member variables that you need. In particular, you will need to provide an alternate constructor for your `Iterator`.
- The required methods above should all be `public`. Any fields or helper methods that you write should be `private`.