Programming Project 2, Phase 1 (revised)

1 Overview

In your second programming project, you will ultimately be working with a dataset consisting of all entrances and exits to the stations of the New York City Transit Authority. This dataset, like the others you have used in this course, is part of the *New York City OpenData* project, and is provided by the Metropolitan Transit Authority (MTA).

In this first part of the assignment, you will implement a hash table, in preparation for the larger assignment. To test your hash table, you will need to write a minimal main program.

2 Objectives

This project's goals are:

- to give you more experience working with real, large, open data sets.
- to give you experience creating a very simple hash table.
- to give you experience programming the on-line disjoint set problem.
- to give you a problem that has practical application and that can be extended to become a useful application.

3 Phase 1 Requirements

The directory on the server,

```
/data/biocs/b/student.accounts/cs335_sw/project2_files
```

contains three header files:

```
gps.h
hash_item.h
hashtable.h
```

The first, `gps.h`, is the interface to a GPS class that you must implement.

The second, `hash_item.h`, is a base class that will serve as the item type for the things that go into your hash table. It is a minimal class:

```
class  __ItemType
{
public:
    /** ItemType() constructor
     *  Creates an item with given values
     */
    __ItemType();


    /** operator==() returns true if given parameter equal item
     *  @precondition: rhs is initialized
     *  @param  __ItemType [in] rhs : item to compare
     *  @return bool 0 if rhs is not equal, 1 if it is.
```

```
     */
    bool operator==( __ItemType rhs) const;

    /** code() returns an unsigned integer for the item
     *  @note Every item, regardless of its key type, should be mapped
     *        to a positive integer value. If the key is a string, this should
     *        assign a number to the string. If it is a number already, this
     *        has the option to assign a different number.
     *  @precondition: item is initialized
     *  @return unsigned int
     */
     unsigned int code();

    /** operator<<() writes item onto the given stream
     *  @precondition: the stream is open and the item is initialized
     *  @postcondition: all data of the item is written to the stream
     *  @param  ostream [in,out] os is the stream to be written to
     *  @param  __ItemType [in]  item is the item to be written
     *  @return the changed stream
     */
    friend ostream &  operator<<( ostream & os, __ItemType item ) ;
};
```

Your job for now is to derive some concrete class from it that you can use for inserting things into your hash table in order to test it. For example, you could give it two member variables, a string and an integer (strong suggestion) and define two items to be equal if they have the exact same string value.

The third file, `_hash_table.h`, defines the abstract base class for a hash table that you must implement. You are free to use whatever hash function you like, but you must use open addressing to resolve collisions, and your solution must be efficient.

```
#include <string>
#include "_hash_item.h"

using namespace std;

// The INITIAL_SIZE should be large enough that it will not need to be
// resized, but you might want to implement resizing in your class.
#define INITIAL_SIZE 4096



class __HashTable
{
public:

    /** find() searches in table for given item
     *  @precondition: item's key value is initialized
     *  @postcondition: if matching item is found, it is filled with value from
     *                  table.
     *  @param  ItemType [in,out] item : item to search for
     *  @return int 0 if item is not in table, and 1 if it is
     */
    virtual int find   ( __ItemType & item ) const = 0;

    /** insert() inserts the given item into the table
```

```
     *  @precondition: item is initialized
     *  @postcondition: if item is not in table already, it is inserted
     *  @param  ItemType [in] item : item to insert
     *  @return int 0 if item is not inserted in table, and 1 if it is
     */
    virtual int insert ( __ItemType item   ) = 0;

    /** remove() removes the specified  item from the table, if it is there
     *  @precondition: item's key is initialized
     *  @postcondition: if item was in table already, it is removed and copied
     *                  into the parameter, item
     *  @param  ItemType [in, out] item : item to remove
     *  @return int 0 if item is not removed from the table, and 1 if it is
     */
    virtual int remove ( __ItemType  item ) = 0;


    /** size() returns the number of items in the table
     *  @precondition: none
     *  @postcondition: none
     *  @return int the number of items in the table
     */
    virtual int size() const = 0;

    /** listall() lists all items currently in the table
     *  @note   This function writes each item in the tabel onto the given stream.
     *          Items should be written one per line, in whatever format the
     *          underlying _ItemType output operator formats them.
     *  @param  ostream [in,out] the stream onto which items are written
     *  @return int the number of items written to the stream
     */
    virtual int listall ( ostream & os ) const = 0;

};



#endif /* __HASH_TABLE_H__ */
```

4 Programming Requirements

Your hash table implementation must not depend on anything not defined in the interface. It must work for any item type that has the interface defined above for the `__ItemType` base class.

If I create a program that makes calls to your hash table class, your class should work regardless of what kinds of things my item type is; it cannot need to be modified.

You must write a main program to test your class. I do not need to see that main program. I do not want to see that main program. I will write my own program to test it. If mine cannot call your class without errors, your class does not implement the interface correctly.

5 Programming Rules

Your program must conform to the programming rules described in the ***Programming Rules*** document on the course website. It is to be your work and your work alone.

6 Grading Rubric

Phase 1 will be graded based on the following rubric, based on 20 points.

- If the `hashtable.cpp` file fails to compile to object code (meaning g++ -c hashtable.cpp fails), on a `cslab` host, it receives 4 points. This is the maximum it can receive. It will be assessed using the rest of the rubric below.
- Correctness of the code is worth 60%
- Design (choices of algorithms, data structures, modularity, organization): 15%
- Documentation: 20%
- Style and proper naming: 5%

7 Submitting the Assignment

Phase 1 of this project is due by the end of the day (i.e. 11:59PM, EST) on April 12, 2019. Create a directory named named *username*`_project2.1`. where *username* is to be replaced by your CS Department network login name. Put all project-related ***source-code*** files and a `README` into that directory. ***Do not place anything else into this directory.*** You will lose 1% for each file that does not belong there, and you will lose 2% if you do not name the directory correctly[1].

Next, create a zip archive for this directory by running the zip command

```
zip -r username_project2.1.zip ./username_project2.1
```

This will compress all of your files into the file named `username_project2.1.zip`. ***Do not use the tar compress utility.***

Assuming this file is in your current working directory you submit by entering the command

```
submit_cs335_hwk 5  username_project2.1.zip
```

because it will be the 5th homework. The program will copy your file into the `hwk5` subdirectory

```
/data/biocs/b/student.accounts/cs335_sw/hwks/hwk5/
```

and if it is successful, it will display the message, "`File ...  successfully submitted.`"

You will not be able to read this file, nor will anyone else except for me. But you can double-check that the command succeeded by typing the command

```
ls -l /data/biocs/b/student.accounts/cs335_sw/hwks/hwk5
```

and making sure you see a non-empty file there.

If you put a solution there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.

[1] I have scripts that process your submissions automatically and misnamed files force me to manually override them.