|  |  |
| --- | --- |
| Program: | GDP1 |
| Course: | INFO6023 – Game Algorithms and Gems |
| Professors: | Michael Feeney |
| Project # 1: | Exploring various “containers” |
| Weight: | Divided equally over all the projects |
| Due Date: | Let's try: Saturday, February 10th at 11:59 PM |

*Note: This project can be done alone or in pairs*

Description and Purpose

Explore the basic Standard Template Library (STL) containers and algorithm libraries in terms of basic usage, fitness for purpose, and trade-offs (time and memory). Then create your own implementations of the some containers and algorithms to explore how they function, and the reasons for the trade-offs.

Details

Rather than deliver a complete application, you will deliver code that will be used a "library" of functions.

You will implement the same functionality with six (6) different containers:

* STL: vector, list, and map (or hashmap)
* Your own ("DIY", or "Do It Yourself") equivalents:
  + vector, or "smart array"
  + list, or "linked list" (either singly or doubly linked)
  + map (either a traditional "red-black" tree or as a hash based container)

More details of the functional requirements of your "DIY" containers are listed later.

You will implement the interface described in the GemsP1InterfaceCode.7z file: (this is listed later) 

You may implement this "interface" as a set of functions **or** methods (not a messy combination of the two, please). Note that "exercise code" I will be using will "call" these functions, so don't change the signature.

The data will be in the format of the US Census data (from 2001, I think) in files: "dist.female.first.txt", "dist.male.first.txt", and "US\_LastNames.txt". Note that I may use different data (i.e. different names, etc.), but it will be in the same format as those files.



**"Exercise" code and marking:**

I will be using code, that will "exercise" your code, calling the various interface methods/functions.

Within reason, I can make *trivial* changes to the code, but generally, **keep in mind that if you code doesn't build, then you get zero**; I will not spend hours altering my code to conform to yours - that's not how it works "in industry" as they say. I'm going to be "reasonable", but seriously: that's what "interfaces" are for, for you to write code to match that interface so that it can be easily "dropped into" the existing code base.

So do **NOT** change the signature (class names, function/method names, order and/or type of parameters), etc. You may *add* to the basic class, but keep in mind that *I will be expecting the members of the cPerson class to be present*.

For methods: I should only have to change the class I am using in the declaration, then the rest of the code should work. So for each student, the fist line where I instantiate the classes would change from:

**cSallysProjectClass\* test = new cSallysProjectClass();**

**test.LoadDataFilesIntoContainer("female.txt", "male.txt", "last.txt");**

**// ... and so on**

... to this:

**cAbrahamsProjectClass\* test = new cAbrahamsProjectClass();**

**test.LoadDataFilesIntoContainer("female.txt", "male.txt", "last.txt");**

**// ... and so on**

For (global) functions: Provide your code as a set of separate files, so that the functions can be globally called. In other words, I will run the "exercise" code multiple times, once per container. Again, do *not* change the signature of the functions (like **FindPeopleByName\_Vector()**, **FindPeopleByName\_List()**, etc.)

In other words, I should only have to change the implementation (cpp) files between containers and students - that implementation file should have the code for the specific function.

**Requirements for DIY containers:**

**General:**

* You need to implement the functionality, ***not*** the exact syntax, of the containers. For instance, you **can** have an **insertAt(int index)** method, rather than actually overloading the operator (operator overloading is always looked at with suspicion… mainly because it’s very easy to get it completely wrong, or do something unexpected).
* The containers do ***not*** have to be templated; they can be writing to only handle the "person" class you are using. Note that you *can* use templates if you'd like, but you will *not* receive any additional marks.
* You do not need to implement iterator classes for access (which is common in the STL). Also note that using templates is completely pointless in a projects like this (it’s often pointless in real code, too).
* It should offer approximately the same "gross" performance as the STL containers. For instance, inserting at the end of a vector is fast ("O(1)" in big-O notation, for every insert that doesn't cause a re-size) in comparison to inserting at a random location, which is "O(n)".   
  (See <https://justin.abrah.ms/computer-science/big-o-notation-explained.html> for a brief, and someone non-theoretical, sensible, introduction to "big-O")
* Other than passing and returning information through the STL vectors used in the "exercise" methods/functions, you may ***not***combine different containers between the STL and your DIY containers. i.e. if it’s the “vector” version, then it should *only* use vector and nothing else.
  + For example, you cannot implement the "sort" functionality in the DIY vector by copying the data to an STL map, then copying it back.
* You cannot "hide" the "real" container behind another. For example, if you are implementing the vector, the data must stay in that container "most" of the time. Like the bullet point above, you can’t perform a "vector sort" by copying the data into an map (which is sorted), then back – the data must be in a vector the entire time; you can't just keep the data in multiple containers all the time, or actually store the data in one container (say a map), but then only copy it into your vector when it is needed.

Here is a brief comparison of STL container performance:   
[http://john-ahlgren.blogspot.ca/2013/10/stl-container-performance.html](http://john-ahlgren.blogspot.ca/2013/10/stl-container-performance.html%20)

**vector:**

* A "smart" or "dynamically grown" array.
* In general, behaves like a "regular" array
* Insert at end (equivalent to "push\_back")
* Insert and delete anywhere (slower than at end)
* Indexed on integer, starting at 0
* Access using index (STL uses **operator[]** for this)
* Find on index takes O(1) time; finding data takes O(n) time (linear search)

**list:**

* Singly or doubly linked list
* Insert and delete anywhere "O(1)"
* Find takes O(n) time
* "MoveNext", "MovePrevious", "MoveToStart", "MoveToEnd" functionality (implemented in iterators in STL library)
* Find takes O(n) time (linear search)

**map:**

* STL usually implements as a "red-black tree". Suggestion is to use a "hash map"
* Insert, delete, lookup: O(log n) time
* Find takes O(log n) time if searching by index, O(n) time (linear search) otherwise

Grading Scheme

1. Normally a grade of zero will be assigned to any assignment that is submitted late. However, certain rare exceptions apply according to the *Infotech* *Policy on Missed Evaluations and Evaluation Deadlines*.
2. If you code does not even compile, I will not mark it. Period.
3. If you code does not build (i.e. linker error) and run (i.e. no crazy run-time crash that is unexpected), I may investigate this further, but only if there is some simple problem and/or slight configuration error.
4. **I’ll be building it in Visual Studio 2015** (Regarding C++11 stuff: you can’t use the “**auto**” keyword), Make sure it can build using the **C++98** standard. Also, no boost (unless you have a /really/ convincing reason for using it. Good luck convincing me, though).

Marking scheme (“INFO6023\_Gems W2017 - Project 1 - Marking scheme.xlsx”):



70/12-year old “squinty eye” plagiarism test:

* While you may freely “borrow” mine (or anyone other) code ***but*** your code should be “sufficiently” different from mine.
* You should also give credit for where you found the code, if you’ve used a significant portion of it.
* In other words, you *cannot* simply use an existing game engine (or part of a game engine) to complete this assignment; it should be either completely new of significantly modified.
* How will I determine this?

If I showed your application and/or your source code to either my *very* pragmatic 70-year-old mother, or a typical 12-year-old, and they looked at it, tilted their heads, squinted their eyes, and said, “you know, they look the same,” then they ***are*** the same.

In other words, I’m not going to be drawn into a giant debate over how “different” your code is from mine or anyone else’s, if a complete stranger, say a randomly stopped in the hallways, would conclude that the code and/or output is pretty much the same thing, then it’s “the same.”

Project Corrections

If any corrections or changes are necessary they will be posted to the course web site and you will be notified of any changes in class. It is your responsibility to check the site periodically for changes to the project. Additional resources relating to the project may also be posted.

Appendix: Interface code

#ifndef \_cPerson\_HG\_

#define \_cPerson\_HG\_

#include <string>

#include <vector>

enum eContainerType

{

STD\_VECTOR = 1,

STD\_LIST,

STD\_MAP,

CUSTOM\_DIY\_VECTOR,

CUSTOM\_DIY\_LIST,

CUSTOM\_DIY\_MAP,

UNKNOWN

};

struct sPerfData

{

sPerfData() :

elapsedCallTime\_ms(0.0f),

memoryUsageBytes\_min(0.0f),

memoryUsageBytes\_max(0.0f),

memoryUsageBytes\_avg(0.0f) {}

float elapsedCallTime\_ms;

float memoryUsageBytes\_min;

float memoryUsageBytes\_max;

float memoryUsageBytes\_avg;

};

struct sPoint

{

sPoint() : x(0.0f), y(0.0f), z(0.0f) {}

float x, y, z;

};

struct sPerson

{

sPerson() : uniqueID(0), age(0), health(0.0f) {}

unsigned long long uniqueID; // 64 bit unique identifier

std::string first;

std::string last;

int age;

float health;

sPoint location;

};

#endif // \_cPerson\_HG\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#ifndef \_cPersonMotron\_HG\_

#define \_cPersonMotron\_HG\_

#include "cPerson.h"

#include <string>

#include <vector>

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \_ \_\_ \_ \_ \_

// /\_\ \_\_\_ / \_|\_ \_ \_ \_ \_\_| |\_(\_)\_\_\_ \_ \_ \_\_(\_)

// / \_ \ (\_-< | \_| || | ' \/ \_| \_| / \_ \ ' \(\_-<\_

// /\_/ \\_\/\_\_/ |\_| \\_,\_|\_||\_\\_\_|\\_\_|\_\\_\_\_/\_||\_/\_\_(\_)

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Loads data into the "main" container.

// Whenever this is called, any "old" data is destroyed.

bool LoadDataFilesIntoContainer( std::string firstNameFemaleFileName,

std::string firstNameMaleFileName,

std::string lastNameFileName );

// Returns only people with this name

// If the name (first or last) is blank (""), then ignore this parameter.

// For example:

// - if last name is "", search only by first name

// - if first name is "", search only by last name

// - if both names are blank, return everyone

bool FindPeopleByName( std::vector<sPerson> &vecPeople, sPerson personToMatch, int

maxNumberOfPeople = INT\_MAX );

bool FindPeopleByName( std::vector<sPerson> &vecPeople, std::vector<sPerson> &vecPeopleToMatch,

int maxNumberOfPeople = INT\_MAX );

// Returns false if not found.

bool FindPersonByID( sPerson &person, unsigned long long uniqueID );

// Finds the closest people (could be zero), from a point and radius.

// Assume that location is "less than or equal" to radius

bool FindPeople( std::vector<sPerson> &vecPeople, sPoint location, float radius,

int maxPeopleToReturn = INT\_MAX );

// Finds people with a cetain health range (inclusive of the min and max values)

bool FindPeople( std::vector<sPerson> &vecPeople, float minHealth, float maxHealth,

int maxPeopleToReturn = INT\_MAX );

// Combination of the two functions above

bool FindPeople( std::vector<sPerson> &vecPeople,

sPoint location, float radius, float minHealth, float maxHealth,

int maxPeopleToReturn = INT\_MAX );

// Returns the enum from the cPerson.h file

eContainerType getContainerType(void);

enum eSortType

{

ASC\_FIRST\_THEN\_LAST,

DESC\_FIRST\_THEN\_LAST,

ASC\_LAST\_THEN\_FIRST,

DESC\_LAST\_THEN\_FIRST,

ASC\_BY\_ID,

DESC\_BY\_ID,

ASC\_BY\_HEALTH,

DESC\_BY\_HEALTH

};

// Sorts the people, then returns the container of sorted people

// - this is from the data loaded by LoadDataFilesIntoContainer()

bool SortPeople( std::vector<sPerson> &vecPeople, eSortType sortType );

// Can be called after every function.

// Calls from any of the above functions overwrites this data.

// Measuring starts from when call is made to just before it returns.

bool GetPerformanceFromLastCall( sPerfData &callStats );

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \_ \_ \_ \_ \_

// /\_\ \_\_\_ \_ \_\_ \_\_\_| |\_| |\_ \_\_\_ \_\_| |\_\_(\_)

// / \_ \ (\_-< | ' \/ -\_) \_| ' \/ \_ \/ \_` (\_-<\_

// /\_/ \\_\/\_\_/ |\_|\_|\_\\_\_\_|\\_\_|\_||\_\\_\_\_/\\_\_,\_/\_\_(\_)

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class iPersonMotron

{

public:

// Loads data into the "main" container.

// Whenever this is called, any "old" data is destroyed.

virtual bool LoadDataFilesIntoContainer( std::string firstNameFemaleFileName,

std::string firstNameMaleFileName,

std::string lastNameFileName ) = 0;

// Returns only people with this name

// If the name (first or last) is blank (""), then ignore this parameter.

// For example:

// - if last name is "", search only by first name

// - if first name is "", search only by last name

// - if both names are blank, return everyone

virtual bool FindPeopleByName( std::vector<sPerson> &vecPeople,

sPerson personToMatch, int maxNumberOfPeople = INT\_MAX ) = 0;

virtual bool FindPeopleByName( std::vector<sPerson> &vecPeople, std::vector<sPerson>

&vecPeopleToMatch, int maxNumberOfPeople = INT\_MAX ) = 0;

// Returns false if not found.

virtual bool FindPersonByID( sPerson &person, unsigned long long uniqueID ) = 0;

// Finds the closest people (could be zero), from a point and radius.

// Assume that location is "less than or equal" to radius

virtual bool FindPeople( std::vector<sPerson> &vecPeople, sPoint location, float radius,

int maxPeopleToReturn = INT\_MAX ) = 0;

// Finds people with a cetain health range (inclusive of the min and max values)

virtual bool FindPeople( std::vector<sPerson> &vecPeople, float minHealth,

float maxHealth,

int maxPeopleToReturn = INT\_MAX ) = 0;

// Combination of the two functions above

virtual bool FindPeople( std::vector<sPerson> &vecPeople,

sPoint location, float radius, float minHealth, float maxHealth,

int maxPeopleToReturn = INT\_MAX ) = 0;

enum eSortType

{

ASC\_FIRST\_THEN\_LAST,

DESC\_FIRST\_THEN\_LAST,

ASC\_LAST\_THEN\_FIRST,

DESC\_LAST\_THEN\_FIRST,

ASC\_BY\_ID,

DESC\_BY\_ID,

ASC\_BY\_HEALTH,

DESC\_BY\_HEALTH

};

// Sorts the people, then returns the container of sorted people

// - this is from the data loaded by LoadDataFilesIntoContainer()

virtual bool SortPeople( std::vector<sPerson> &vecPeople, eSortType sortType ) = 0;

// Can be called after every function.

// Calls from any of the above functions overwrites this data.

// Measuring starts from when call is made to just before it returns.

virtual bool GetPerformanceFromLastCall( sPerfData &callStats ) = 0;

// Returns the enum from the cPerson.h file

virtual eContainerType getContainerType(void) = 0;

};

#endif // cPersonMotron