AOOP Final Project

Your final project assignment is to complete an application that uses the framework we have developed all semester to create a simple Person database. A number of new classes will be added, all of which are described here.

Note: Due to my own fault, this assignment was not handed out on time. While you will only have 1 week to complete it, I have attempted to balance your work load by actually creating all the classes for you. In these new classes, you will find that not all the functions are actually implemented. Your responsibility is to locate the methods that need to be completed, and properly do so.

For example, there is an MGArray template class that has an Add function. If you look in the MGArray.h file, you will find the following function amongst other things:

```
bool Add( RT Val )
{
	// Write code here to add item to m_Elements
	return true;
}
```

Note that the comment describes what you need to do. Because the function has a proper return type, the code should compile cleanly. It will not however work completely until you have added all the required code.

You can also download the EXE file for the finished executable, to see how the program should operate once you have completed all your coding. Look for FinalProj_Completed.zip.

Exising Classes

The following classes are all from our previous midterm and homeworks. You are free to use your own version of these classes, but I have included the complete final source for them in the FinalProj.zip file, as part of the project. You shouldn't need to alter these classes in any way.

These classes are:
- MGDate
- MGException
- MGFileException
- MGMemoryException
- MGNameString
- MGObject
- MGPerson
- MGString
- MGStringException

New Classes

The following section describes new classes and their purpose and methods. These are the classes you will need to complete by 'filling in' the proper code in the methods I provided.

MGApplication

This is a wrapper class for a complete application. It provides convenient access to a DOS-based menu system, simplified console input routines, a single data file, and convenient organization of startup, run, and shutdown program stages.

You don't use MGApplication directly. Instead, you create a class derived from MGApplication to perform your programs job. For example:

```
class FinalProj : public MGApplication
{
public:
	bool OnRun();
};

bool FinalProj::OnRun()
{
	cout << "This is where the program would do things" << endl;
	return( true );
}

int main()
{
	try {
		FinalProj Program;
		Program.Run();  // Run calls OnStartup, OnRun, and OnShutdown
	}
	catch( MGException& E )
	{
		cout << "Error: " << E << endl;
	}
	return(0);
}
```

Note that our example derived class *FinalProj* (you will need to complete this class, see the FinalProj.cpp file) has no data members, so the constructor isn't needed. In your version, you will need a constructor.

As needed, you may add methods to the derived class (FinalProj), such as LoadDataFile.

The methods of this class are:

User Interface support

int GetNum(char* Prompt, int Min, int Max);
Prompts the user to input a number between *Min* and *Max*. *Prompt* is displayed before input is begun. The return value is the users choice, and will be between *Min* and *Max*.

void GetLine(char* Dest, int Max);
Inputs a line of text to *Dest* upto *Max* characters.

int Menu(char* MenuItems[]);
Displays the strings from the given array *MenuItems*. Each string is prefixed with its index. For example, if MenuItem[0] was "Add", then output would be "1. Add". The return value is the users menu selection.

Constructors and Destructors

MGApplication(const char* Datafile=0);
The single constructor, initializes the m_DataFile data member with *Datafile*, if present.

virtual ~MGApplication();
Currently, does nothing.

Application Flow

virtual bool Run();
This is called to run the application. It calls OnStartup, and if OnStartup returns *true*, then it proceeds to call *OnRun*. It will always call OnShutdown. The return value is the return value of OnStartup or OnRun.

virtual bool OnStartup() { return(true);}
Intended to be optionally replaced in derived classes, this method defines startup procedures, such as loading a data file. In Derived classes: If the startup fails (i.e., the file can't be opened), then this method returns false, which causes the program to terminate.

virtual bool OnRun() = 0;
This is a pure-virtual method, and you must provide it. This is basically the main processing of your new application, such as the displaying of the menu and the editing of records.

virtual void OnShutdown() { /* Do Nothing */ };
Intended to be optionally replaced in derived classes, this method defines shutodwn procedures, such as saving a data file. This method is always called.

Data Members:

MGString m_DataFile;

MGArray

This is a templated class that contains an STL vector data member. The goal of this class is to wrap as a 'wrapper' to the vector to make it work more like our MGObject framework. It is not inherited from vector.

Methods in this class are as follows:

MGObject derived methods

bool GetAsString(char* Dest) const
bool SetFromString(const char* Src)
These two methods should do nothing, just return *true*. This is a practical issue, and you can leave this code as is.

virtual bool SaveToFile(ofstream& Dest) const
virtual bool LoadFromFile(ifstream& Src)
These two methods must save or load the elements in the m_Elements data member (discussed below) to and from a file. You should assume that m_Elements is a collection of MGObject (or derived) pointers, and therefore can do the following to save your data:

1. Write the count of elements to the disk file
2. Call each elements SaveToFile method, to store their data

The opposite would be done for loading:

1. Read the count of elements from the disk
2. That many times, create an RT object, and then call its LoadFromFile method.

Please note: This class is not designed to mix element types. For example, we will have an RT of MGPerson for each element. Therefore, Run Time Type Identification is not important.

Collection Management

bool Add(RT Val)
Adds *Val* to our m_Elements collection.

void Remove(int Index)
Removes the element at *Index* from our m_Elements collection. Should not accept invalid indices.

RT& operator[](int Index)
Returns a reference to element *Index* from m_Elements. Should not accept invalid indices.

int GetSize() const
Returns the number of elements from our collection.

Data Members

There is one data member, which should not need to be changed: The vector object we are wrapping:

```
vector< RT > m_Elements;
```

Note: Certain sections of the MGArray.h file specify "Don't change this". I hope this is straightforward.

Requirements

You must complete all the required coding as noted in the comments, and any additional code or functions you may find to complete your job (there may be none). You must provide me with a printout and disk containing your project. The due date is 1 week from assignment.

Visit www.openroad.org for electronic versions of this document, the 'starting' point project in VC++ 6.0, and the completed EXE file for demonstration purposes. Please begin immediately, and contact me if you have any problems, and check the website for FAQs that may arise.