Exercise III: Projects with Multiple Files and Classes

A project will typically be composed of multiple files rather than just one. Each class in a project will usually have two files associated with it; the source file and header file. In this exercise we will look at adding multiple files to a project, both from scratch and by using existing files.

For this exercise, we will create some of our own source code and header files, but will also use some that have already been written. In order to obtain these prewritten files, you need Internet access, an FTP program and an unzip program. The FTP program I recommend for PC users is WS FTP, but any FTP program will do. The unzip program I recommend for PC users is WinZip, but again, any unzip program will do. If you do not have an FTP and unzip program installed on your machine, you can find and download such programs by going to a site such as **www.download.com** and searching for them.

The prewritten files which will be used in this exercise are located at the anonymous FTP site **ftp.cs.umd.edu** in the **/pub/egolub/VC.workbook** directory in a file named **e3.zip**. You should create a temporary download directory if you do not already have one. Connect to the FTP site using **anonymous** as your login ID and your e-mail address as your password. Download this file using binary transfer mode and unzip it. Remember where you placed these files since we will need to use them later.

Launch Visual C++ on your computer. Create a new, empty, **Win32 Console Application** named **exercise3**. Go to the project settings and disable the language extensions as shown in Exercise II. Add a C++ source file named exercise3.cpp to this project using the techniques showed in Exercise II. Enter the program shown in Figure III.1 in exercise3.cpp.

```
#include <iostream.h>
#include "class1.h"

int
main() {
class1 x,y,z;

     cin >> x >> y >> z;

     cout << x << " " << y << " " << z << endl;

     return 0;

}
```

Figure III.1

Notice that this program uses objects of type class1 and includes class.h at the top. You will now need to add new files called class1.h and class1.cpp to the project. First, let's add class1.h to the project.

Go to the **PROJECT** menu, go to the **Add To Project** submenu and select **New…**. This time, however, ***single click* on C/C++ Header File** from the presented list. In the text entry box labeled **File name:** enter the name **class1.h** and press **OK**. A file named class1.h has now been created in your project's directory, added to the project and opened in the editor panel in Visual C++. Enter the code shown in Figure III.2 in class1.h.

```
#ifndef class1_h
#define class1_h
#include <iostream.h>

class class1 {
     int first, second;
public:
     class1(int=0, int=0);
     class1(const class1&);
     ~class1();

     friend istream& operator >> (istream&, class1&);
     friend ostream& operator << (ostream&, class1);
};

#endif
```

Figure III.2

You can now compile exercise3.cpp to confirm that these two files work together. Note that you have not created class1.cpp yet, so you can not attempt to compile the entire project. You can only compile exercise3.cpp. To accomplish this, ***double click*** on exercise3.cpp under the **FileView**, go to the **BUILD** menu and select **Compile exercise3.cpp**. This is called partial compilation – exercise3.cpp will be compiled and if there are no errors, an object file will be created for that file. When we compile an entire project, each source file is compiled to an object file, and then all of the object files are linked to create the executable.

Next, add another C++ source file to your project, and call this one class1.cpp. Enter the code shown in Figure III.3 in class1.cpp.

```
#include "class1.h"

class1::class1(int f, int s){
     first=f;  second=s;
}

class1::class1(const class1& obj){
     first=obj.first; second=obj.second;
}

class1::~class1(){
}

istream& operator >> (istream& is, class1& obj){
     is >> obj.first >> obj.second;
     return is;
}

ostream& operator << (ostream& os, class1 obj){
     os << obj.first << ":" << obj.second;
     return os;
}
```

Figure III.3

Now, all components of the project as it is currently designed have been created. You can now compile the full project to create an executable file. Do so, and run the program. The program expects a total of six integers to be entered by the user.

We are now ready to add in the final part of this exercise. In exercise3.cpp, prior to the return statement, add the following line of code:

```
funWithRationals();
```

Figure III.4

This is a call to a function that is implemented in the file **fwr.cpp**. That file is among the files you downloaded at the beginning of this exercise. Try to recompile the project now. Visual C++ should inform you that **funWithRationals** is an undeclared identifier. Now enter the line of code:

```
#include "fwr.h"
```

Figure III.5

at the top of **exercise3.cpp** right under the include directive for **iostream.h**. Attempt to compile the project again. This time, Visual C++ should inform you that there is no such file as **fwr.h**. It is now time to make use of those files you downloaded. First, from your Windows environment, copy or move those files from their temporary location into the exercise3 directory.

Now that the files are in the exercise3 directory, once again attempt to compile the project. This time, **fwr.h** will be automatically added to the **External Dependencies** folder under the FileView, but the project will still not create an executable during the linking stage since not the function **funWithRationals()** was never implemented as far as Visual C++ is concerned. If you fully expand the FileView tree, it will appear similar to the following:

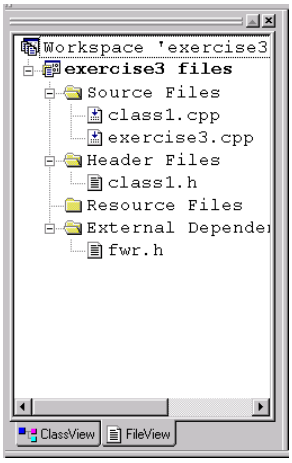

Figure III.6

To add existing files to the project, go to the **PROJECT** menu and move your mouse to the **Add To Project** sub-menu. From the **Add To Project** sub-menu, select **Files…**. You should now be presented with a dialog box similar to the following:

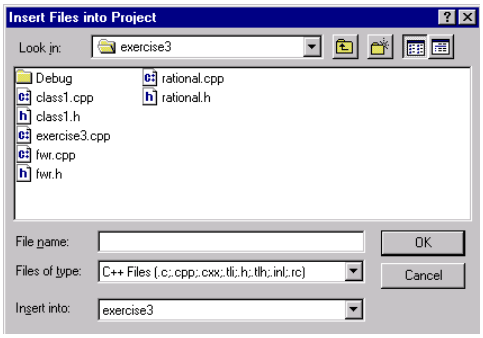

Figure III.7

To add the files **fwr.h**, **fwr.cpp**, **rational.h** and **rational.cpp** to the project, while holding down the **Control** key, ***single click*** on each of their names. Then, release the **Control** key and click on the **OK** button. Once again, attempt to compile to project to an executable. This time, Visual C++ has all of the resources it requires, and an executable is created.

The program that you have created now requires a total of nine integers to be entered by the user. Run the program now.

Now that we have created this project, we can revisit the ClassView to see how it displays information about the classes within a project. ***Single click*** on the ClassView tab to bring up the ClassView page.

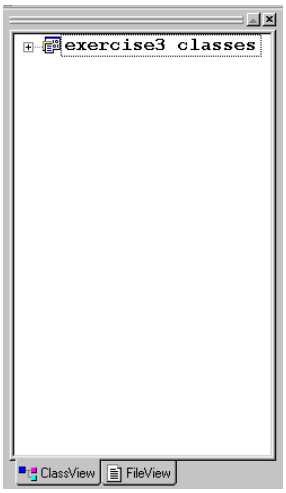

Figure III.8

When the ClassView first comes up, the tree should be fully collapsed as it is in Figure III.8. Expand the first level of the tree by ***single clicking*** on the + next to **exercise3 classes**.

Figure III.9

We now see the names of the classes that are part of this project. If you ***single click*** on the + signs next to either class name (**class1** or **rational**) that part of the tree will expand to display the members of that class. Notice that none of the standalone functions such as **main()** or **funWithRationals()** or the non-member operators appear.

If you have not done so yet, ***single click*** on the + sign next to **rational** to expand that part of the tree. Notice (as shown in Figure III.10) that the icons next to each component represent information such as whether a member is data (blue block) or a function (purple block), and whether it is private (a lock appears) or public (no extra icon).

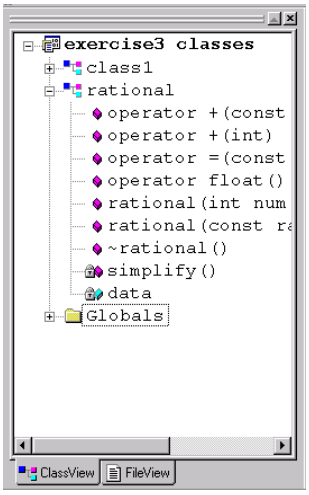

Figure III.10

For any classes that you expanded, ***single click*** on the **–** sign next to the class name to collapse that part of the tree so that your ClassView appears as it had in Figure III.9. Now, ***single click*** on the **+** sign next to the **Globals** folder.

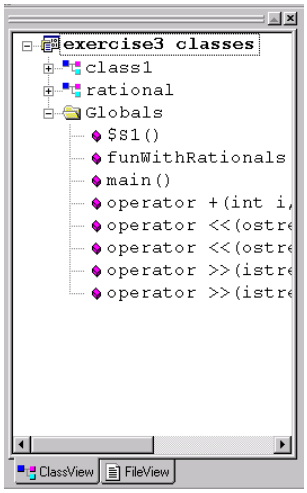

Figure III.11

Any functions that are part of the project, but are not members of any class within the project will be placed in the **Globals** folder under ClassView. We will revisit ClassView again in Exercise X to see some if its other uses.

Congratulations! You have now compiled and executed your third exercise.

This exercise will be the foundation of the next few exercises as well. In each of these exercises, we will begin by instructing you to create a new project, and copy the source and header files from this exercise into that project's directory. This is done so that each exercise may be self-contained.

To leave the Visual C++ environment, go to the **FILE** menu and select **Exit**.