Laboratory 10: *Using Arrays of Class Objects*

Overview: In today's lab we will explore a very useful combination of two object types in C++. Specifically, we will see how arrays of complex user-defined class objects can be used to store and manipulate data effectively and conveniently.

Objectives: Before you leave lab today you should first of all have a very thorough understanding of classes and their use. Don't even bother with the remaining objectives if classes still aren't clear to you. Work on that first! In addition, when you are finished you should understand, among other things: what an array is, how to define one and create *indexed variable objects* that access it. You should know how to write a loop that traverses (goes through) the elements of an array. You should understand that arrays can contain user-defined objects instantiated from some class and not just fundamental object types. Finally, you should have a good idea of when to use such an *array of class objects* structure, and how to declare one.

1. Setting up for the lab

Please perform the following activities as you wait for the lab to formally begin:

a. Boot up the machine and log on to the CS Department server.

b. Put your lab diskette in the *A:* drive and make a subdirectory *LAB10* in the *LABS* directory.

c. Download the from the class Web site to your new directory. The URL is http://s9000.furman.edu/chealy/cs11/lab10.

2. Arrays of class objects

As we have discussed at great length, the ability to define our own class types is one of the most powerful and unique features of C++. A class defines both attributes (data members) and actions (member functions) which are the encapsulated components of any object instantiated from that class.

One of the best features of a class object is that it can contain data members of *different types*. For instance, suppose we want to store the following information for a student:

- name (a string)
- ID number (an integer)
- current grade point average (a floating point value)
- number of CLPs (an integer)

Since several pieces of related data are to be stored, you might think of using an array. But the different types rule that out since all of the elements of an array must be the same type.

The best idea is clearly to encapsulate this data into a class definition called something like *Student*, together with any relevant member functions. Then you can declare as many objects as you like to be of type *Student—student1*, *student2*, *student3*, and so on.

There is still a potential problem though. If we were going to use this class to store data for all 2500 Furman students, we would hardly want to declare 2500 separate objects! The logical and convenient solution is to group all of the class objects into an array structure. A definition such as the following would do the trick:

```
Student data[2500];
```

Accessing individual data items from this structure is fairly straightforward, as long as you keep clearly in mind how it is contructed, and also build your accesses from most general to most specific. For instance, to print the name and GPA of the 100th student:

```
cout << data[99].name << endl;
cout << data[99].GPA << endl;
```

Give some thought to this example, and ask if you are at all confused.

3. Defining an array of class objects

Let's assume that we have stored information about satellite-produced images in a file. In particular, suppose the file includes the following information about each image. (Data types are given in parentheses beside each attribute name.)

• *ImageNum*	(integer)
• *Satellite*	(string)
• *Latitude*	(float — latitude of the center of the target area in degrees)
• *LatDirection*	(char — one character to denote whether latitude is N or S)
• *Longitude*	(float — longitude of the center of the target area in degrees)
• *LonDirection*	(char — one character to denote whether longitude is E or W)
• *Altitude*	(float)

You will be supplied with a file containing such data. It is a text file with the data for individual images held on single lines. Each line will be in a format like this example:

```
101 DELW1 42.3 N 123.5 E 23.4
```

A single space separates each data item. There are seven values, representing the seven data items specified above. You cannot assume that you know how many lines of data will be in the file.

This example is typical of many programming tasks that access data stored in files. A frequent constraint in these cases is that our programs must be constructed to match someone else's file design. We may prefer to have the file format designed differently, but we're stuck with an existing format nonetheless.

We wish to produce a program that will provide a user the option of retrieving and displaying this information in a flexible way. To provide for the most flexible display of the information, we will store it in an array of class objects.

a. Start up Borland C++. Open the project *IMAGE.IDE* and each of the files *IMAGE.CPP*, *IMAGE.H*, *IMGMAIN.CPP*.

b. Examine these files carefully. Read the comments. Ask me to help you understand the details.

c. Run the program. Note that the name of the input file is *image1.dat*. You should notice that the output corresponds to the information provided by the input file. Do you understand what the output means? Is there any information missing? What code would need to be modified to make the output more meaningful? √

4. Processing an array of class objects

a. Now complete the implementation of the << operator in image.cpp as well as the function *print_images* in the main program. This function should display the file contents in a format similar to the following. Notice how the latitude and longitude data are displayed. Run your program again using *IMAGE1.DAT* and compare the output with the follwing. If there are discrepancies, make the necessary changes and test the program again on the same file. √

```
 Image   Satellite   Latitude  Longitude  Altitude
------------------------------------------------------------
  101      DELW1      42.3N    123.2E        23.4
  202      APC34      34.5S    108.5E        76.7
  204      APC34      44.6N     98.5W        44.7
  302      DELW1       64.5N    128.3W        36.8
  400      APC34      34.5S    108.5E        41.3
```

b. The user would like to have the ability to restrict which image information gets printed to the screen. In particular, we would like to ask the user to enter a hemisphere restriction: print only those images that are in the northern or southern hemisphere. Write a new member function for the image class called is_northern() that returns 1 if the object is in the northern hemisphere and 0 if it is in the southern hemisphere. Modify main() so that it asks the user to enter ‘N’ or ‘S’ as the hemisphere restriction. Finally, modify the print_images function so that it will only print the appropriate image data. (Hint: you may need a new parameter.) Run and test the revised program. √

5. Finishing up

a. Close all windows and exit Borland C++.

b. Exit all applications and shut down your computer.

You’ve done a great job in lab this term!!! I hope to see you in another Computer Science class soon.