LAB 6
Spring 2011, BESE- 16
Composition in C++

Objective:

The objective of this lab is to familiarize students with concepts of:

- Use enumerations
- Use Constant and Static Class members
- Understand and use the concept of Composition
- Create a multi-file program with class declaration in a file separate from the class definition file

Submission Requirements

You are expected to complete the assigned tasks within the lab session and show them to the lab engineer/instructor. You need to submit the report even if you have demonstrated the exercises to the lab engineer/instructor or shown them the lab report during the lab session.

Separating the Interface and Implementation of a class

Interface of a Class

Interfaces define and standardize the ways in which things such as people and systems interact with one another. For example, a radio's controls serve as an interface between the radio's users and its internal components. The controls allow users to perform a limited set of operations (such as changing the station, adjusting the volume, and choosing between AM and FM stations). Various radios may implement these operations differently, some provide push buttons, some provide dials and some support voice commands. The interface specifies what operations a radio permits users to perform but does not specify how the operations are implemented inside the radio.

Similarly, the interface of a class describes what services a class's clients can use and how to request those services, but not how the class carries out the services. A class's interface consists of the class prototype (the declaration only) since that defines what functions the class contains, what are their argument and return types. However, the implementation details (bodies of member functions) can be separated from the interface.

It is the usual C++ practice to separate the interface from the implementation by placing them in separate files. If client code does know how a class is implemented, the client code programmer might write client code based on the class's implementation details. Ideally, if that implementation changes, the class's clients should not have to change. Hiding the class's implementation details makes it easier to change the class's implementation while minimizing, and hopefully eliminating, changes to client code.

Header Files

Each of the examples that we have studied till now consists of a single .c pp file, also known as a source-code file, that contain s a class definition and a main function. When building an object-oriented C++ program, it is customary to define reusable source code (such as a class) in a file that by convention has a .h filename extension known as a header file. Programs use #include preprocessor directives to include header files and take advantage of reusable software components, such as type string provided in the C++ Standard Library and user-defined types.

Implementation Files

Source-code file defines class's member functions, which were declared in header file. Remember that each member function name is preceded by the class name and ::, which is known as the binary scope resolution operator. This "ties" each member function to the (now separate) class definition, which declares the class's member functions and data members. Without "classname::" preceding each function name, these functions would not be recognized by the compiler as member functions of class, the compiler would consider them "free" or "loose" functions, like main. Such functions cannot access class's private data or call the class's member functions, without specifying an object. So, the compiler would not be able to compile these functions. To indicate that the member functions in classname.cpp are part of class, we must first include the classname.h header file

How Header Files Are Located

Remember to add the name of the classname.h header file is enclosed in quotes (" ") rather than angle brackets (< >). Normally, a program's source-code files and user-define d header file s are placed in the same directory. When the preprocessor encounters a header file name in quotes, the preprocessor attempts to locate the header file in the same directory a s the file in which the #include directive appears. If the preprocessor cannot find the header file in that directory, it searches for it in the same location(s) as the C++ Standard Library header files. When the preprocessor encounters a header file name in angle brackets (e.g., <iostream>), it assumes th at the header is part of the C++ Standard Library and does not look in the directory of the pro gram that is being preprocessed.

Task to Do

Take the time class separate the interface and implementation of the class into two files, time.h and time.cpp. Then write a test program (saved as test.cpp) to create objects of the time class in main(). First save the class declaration (only the prototype) in time.h. Then put the implementation of all time class functions in time.cpp. The procedure to run a multi-file program in Borland C++ is given below:

Time Project:

//time.h file has declaration of following class members

1. Three integers: hours, minutes, seconds. A static integer count to display the number of objects created.
2. A default no-argument constructor that sets values to zero and increments count.
3. A three integer argument constructor that sets the values of hours, min, sec to user given values and increments count.
4. A gettime method that returns void and takes no arguments. It takes input from user and sets value for hours, min, sec.
5. A printUniversal method that returns void and takes no arguments. It displays data in 24 hour time format.
6. A printStandard method that returns void and takes no arguments. It displays data in 12 hour time format.
7. A getconut static method. It returns void and takes no arguments. It displays the value of count (no of objects created).

//time.cpp file

The time.cpp file defines all class members declared in time.h. It includes the time.h file: #include "time.h". it does not require any main or any class definition.

//test.cpp file

The test.cpp file has the main() function. Perform the following operations in main:

1. Declare a ‘Week’ enumeration with values: Mon, Tue, Wed, Thur, Fri, Sat, Sun.
2. Create two variables of Week day1, day2 and assign them values.
3. Create three variables of class time. Initialize first to default values, second to user given values by using constructor.get values for third object using gettime() method.
4. Create a constant object ‘noon’, initialize it to standard 12:00:00.
5. Get count of the objects created.
6. Print universal and standard times for each object.
7. Print values of day1 and day2.
8. Print the day that appears first in the list (or has smaller value).

Compiling multi-file programs in Borland C++ version 5.02

1. Go to File -> New -> Project
2. Name the project **test.ide**
3. In the **target type** field, select **EasyWin[.exe]**
4. Uncheck Class Library, BWCC and No Exceptions, then click on OK
5. If there are any nodes created in addition to test.exe, delete them by right clicking on each of them and selecting delete node
6. Right click on **test.exe** and select add node. From the options select **time.h**
7. Similarly add the file **time.cpp** to your project
8. Create a node named **test.cpp** and write your main program in that file to test the time class.
9. Go to Project -> Build all. If there are no errors, click on Run and that should execute your project.