Laboratory 6: *Strings & Separate Compilation of Files*

Overview: This week in the lab we will be exploring the technique of creating a project using separate files that contain the main program, the function headers, and the function implementations. This technique is used during the development of almost all significant programs. In addition, we will practice working with strings and classes.

Objectives: Before you leave lab today you should understand, among other things: how to create a multifile project, how to add nodes to the project, how to divide programs into appropriate header, function implementation, and main files. You should gain insight into the use of strings using both string libraries and string classes.

1. Setting up for the lab

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

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

b. Put your lab diskette into the A: drive. Check the space on the disk. If there is not at least 1.1 Mbytes, remove unnecessary files before you proceed. If necessary, try different disks to find one with enough space.

c. Make a subdirectory LAB06 on your disk.

d. Download the file lab06.exe from the class web site into your LAB06 directory. This file is an archive of several files that are stored in a compressed form. Execute this file by double clicking on it. This should cause the 11 files to be extracted from the lab06.exe archive. When prompted for the folder to put the extracted files, type in a:\labs\lab06.

2. Creating Projects Using Multiple Files

The C++ model of building a program is to write a concise, straightforward and logical main program which uses abstraction in the form of functions that are stored in files distinct from the file containing the main program. Until now, we have simply stored these functions in the same file as the main program. We will now practice using projects in Borland C++ to put function implementations into separate files.

Creating the Grade Project

Open the program GRDFUN.CPP. This program, which computes letter grades, contains function prototypes, main, and function implementations in a single file. For small programs when one is learning C++ this is acceptable, but a monolithic program of this sort is not convenient or acceptable for more complex programs.

First, you should study the program and try to understand it. Run it a few times.

Next, open in turn the following files and examine each one carefully. Please note that for programs that require teams of programmers for their creation, a single program that contained everything would never exist. Normally, the problem would be to *assemble the work of the different programmers* into an executable project.

grade.cpp	Contains the function main(), but it does not contain the function prototypes or function implementations. It does contain an #include file for the function prototypes.
grdlib.h	Contains the function prototypes. Notice the use of the preprocessor commands in this file.
grdlib.cpp	Contains the implementation of the functions.

Think about how the three files described above contain the information needed for the complete program.

You now to create a project which describes how to compile and link these files into one executable file.

a. Select *New Project* from the **Project** menu. Change the project name to **a:\labs\lab06\grade.ide**. By convention the project file name should be the same as the main program file name only with an extension of **.ide**. Make sure that the Target Type selected is *EasyWin*, and that the box labeled "Target Model" says *Large*. Now click OK.

   The Project Window will appear at the bottom of the screen. If it doesn't, go to the **View** menu and select *Project.* You may also need to enlarge the Borland C++ outer window to see the project window. Do this by dragging down the bottom edge of the window. You should see these files:

   grade (.cpp)
   grade (.rc)
   grade (.def)

   beneath the name of the executable file **grade (.exe)** that they will help create.

If you don't see these files, click on the + beside **grade.exe**. The .**rc** and the .**def** files are not necessary for simple compilation of implementation files so we can delete them from the project. However, notice that our **grdlib.cpp** is not listed, so we need to add that implementation file.

b. With the right mouse button, click once on **grade (.rc)**. Then, select the option *Delete Node*. Do the same thing for **grade (.def)**.

c. With the right mouse button, click once on **grade (.exe)**. Select the option *Add Node*. In the dialog box, choose **grdlib.cpp** to add the function implementation file to your project. When you close the dialog box, you should see **grdlib (.cpp)** in the list of files.

d. To compile the program, go to the Project menu and select the "Build All" option. Once the program successfully compiles, you can run it by going to the Debug menu and selecting the "Run" option. (Performing the compiling and running tasks separately for a project like this tends to be more reliable than using the lightning bolt.) If the program won't compile because the compiler cannot find an include file, then you need to make sure that the default library directories are correct. Go to the Options menu and click on "Project". On the right side of the pop-up window you should see that the Include directory is **c:\bc45\include**, and the Library directory is **c:\bc45\lib**.

Your project setup is now complete. Under the **Project** menu option you should see options for opening and closing projects. Use these options when you want to access your project in the future. √

Creating the String Function Project

The string function project will implement the string operations described in Section 7.2 of the textbook.

testlib.cpp	Contains the function main(), but it does not contain the function prototypes or function implementations. It does contain an #include file for the function prototypes.
strlib.h	Contains the function prototypes for strings. Notice the use of the preprocessor commands in this file.
strlib.cpp	Contains the implementation of the functions for strings.

a. Close the Grade project, and now create a *new* project named **testlib.ide**.

b. Add the strlib.cpp file to the project.

c. Build All and Run while your project window is active.

d. Test the program and try to understand the results. √

Close the project.

Creating the String Class Project

The string function project will implement the string class program described in Section 7.3 of the textbook.

stats.cpp	Contains the function main(), but it does not contain the function class member functions or function class . It does contain an #include file for the function prototypes.
cstrlib.h	Contains the class declaration section for the string class.
cstrlib.cpp	Contains the implementation section for string class.

a. Create a project named **stats.ide**.

b. Add the cstrlib.cpp file to the project.

c. Build All and Run while your project window is active.

d. Test the program and try to understand the results. √

Close the project.

Your project setup is now complete. Under the **Project** menu option you should see options for opening and closing projects. Use these options when you want to access your project in the future.

3. Modifying the class

1. Now, re-read the file stats.cpp so that you understand how it counts words. Now, modify it so that it only increments the word count if the "word" actually contained any vowels. The next two steps indicate how to accomplish this.

2. Modify the declaration file cstrlib.h and implementation file cstrlib.cpp to create this new member function *contains_vowel*().

3. Inside stats.cpp, modify the code that counts the words to invoke the member function *contains_vowel()*, which is now part of our string class.

4. Compile and test your program again. Call me over for a demonstration. √

4. Finishing up

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

b. Exit all applications and shut down your computer.