ECE 70 – Engineering Computations in C

Professor Kriehn – Fall 2016

Code Due By: Midnight on Wednesday, November 30, 2016
Writeup Due By: Class on Thursday, December 01, 2016

HOMEWORK #31 – HW #22 Revisited
Multiple Source Files & The Creation of Static Libraries

Copy your "`HW22`" project over to "`HW31`". Rename the "`hw22.c`" source file to be "`hw31.c`". Then perform the following operations:

1. Modify your code so that you no longer prompt the user for the array size or the array elements. Assume that the array data is in a file called "`zero.dat`" (see the website). The file contains 11 numbers, so declare your double floating point array to be 11 elements long. (Use a constant.)
2. Modify your code to read in numbers from the "`zero.data`" text file into your array.
3. Move all the statistics functions into a new file called "`statistics.c`". Include the new source file in your Netbeans Project under "`Source Files`".
4. Create a file called "`statistics.h`" that contains the appropriate function prototypes. Include the new header file in your Netbeans Project under "`Header Files`".
5. Include the proper preprocessor directives into **ALL** your source files to ensure they compile correctly.
6. Compile, run, and debug your program using Netbeans – it is extremely important that you learn how to successfully compile multiple source files into a working program before the Final Project is assigned.
7. Don't forget to copy the "`zero.dat`" file over to your working Netbeans directory using **`WinSCP`** or **`scp`** to test your program.

Compiling and Linking the Code by Hand

Once you have finished debugging your project in Netbeans, login to a terminal window and also compile your code by hand. To do this, compile both source files at once and link them together to create an executable file using the following command:

```
~> gcc -lm –std=c99 main.c statistics.c -o hw32
```

The general problem with this approach is that any time you have to make a change to either file, you will have to recompile and relink everything back together. A much better process is to compile the library file separately without invoking the linker to create a library object file:

```
~> gcc -c –std=c99 statistics.c
```

This will create a library file "`statistics.o`". Next, we can use the `ar` command (archive) to create a static library file out of the "`statistics.o`" file:

```
~> ar cr libstats.a statistics.o
```

The `cr` option stands for “create and replace” and will create a new library file called “`libstats.a`”. If the library does not exist, it is first created. If it does exist, the original files within the library are replaced with the new ones. The archiver “`ar`” also provides a “table of contents” option “`t`” to list any object files in a library. For example, typing:

```
~> ar t libstats.a
statistics.o
~>
```

indicates that there is one object file within the library, called “`statistics.o`”. We can now compile our main program “`main.c`” and link the library to it. This can be done one of two ways. The first way is to type the following:

```
~> gcc -lm –std=c99 main.c libstats.a -o hw32
```

The main program, “`main.c`” is compiled and then liked against the object files found in the library file “`libstats.a`” to produce the final executable “`hw32`”.

The second option is to directly call the linker using the “`-l`” option. In this case, we do not need to call the full filename of the library file directly:

```
~> gcc -lm –std=c99 main.c -L. -lstats -o hw32
```

The “`-L.`” option indicates for the linker to look in the current directory for existing library files, and the “`-lstats`” option tells the linker to automatically include and link the “`labstats.a`” library file into the executable. Note that the “`lib`” and “`.a`” portions of “`libstats.a`” are missing from the “`-lstats`” option. This is because the linker will automatically know to look for a file called “`libstats.a`” based upon the library that we are trying to include. If we were to include the option “`-llibstats.a`” instead, the linker would fail.

Why do all of this? Because once we are finished creating our “statistics” library, we never have to touch it again! If we want to use the functions in future programs, we only have to link the library into the source code we want to use it with, and we are done. Furthermore, we can also modify our original source code “`main.c`” and not have to recompile our “`statistics.c`” file.

To help you understand what is going on, I have found an excellent online reference that helps explain everything:

http://www.network-theory.co.uk/docs/gccintro/index.html

Please read Chapter 2 and Section 10.1 as part of working on this assignment. It will help explain what is happening with the compiling and linking process. For those of you seriously interested in "real-world" C/C++ programming, I recommend reading the entire book in your free time and coding all of the examples presented in the document.

When you are satisfied with the creation of your library, submit your `main.c` file to the grader program as HW 32 – the last homework assignment of the semester.