Module 2 Assignment (Homework Project #1)

System Development in the Unix Environment (605.614)

Overview

The homework assignment for this class is to set up a project hierarchy as described below. Each appropriate level of the hierarchy will contain a Makefile, which will support a set of project management tasks. To allow the hierarchy to be exercised, two small C-programs shall be written, along with some support libraries that these programs shall use. This assignment makes use of material included in Modules 2 and 3.

Getting Started

Be sure to review module content regarding how to use Make in a software project. Also, review the video explaining how your project should work. Then, build your directory hierarchy as specified, and begin to populate the hierarchy with the proper Makefiles.

Objectives

After completion of this project, you will be able to:

- Create Makefiles that properly build and install libraries and programs using the libraries
- Create a directory hierarchy with proper Makefiles to build the programs and libraries properly at each level of the hierarchy
- Create a simple script (in Perl or Python) to package your hierarchy in both source and binary form.
- Build source files and Makefiles according to a supplied style guide.

Permissions and Submission

Completion of the assignment involves completing, on or before the due date, the following materials:

- The directory hierarchy and Makefiles along with the required source code.
- A comment block at the beginning of each Makefile, program, and library source file shall include the following information: the student, the course, the name of the file, and a short description of the file's purpose.
- The Assignment should be compiled and run on the (UNIX) system (dev4.jhuep.com) and Linux system (absaroka.jhuep.com). The

executable programs must be installed in the structure mandated in the last section of this document.

The fundamental rules governing the Assignment (see the Syllabus) are:

- The programs must be written in the 'C' or the 'C++' programming language.
- No copying or plagiarism. If copying or plagiarism occurs, the person or persons involved will receive no credit. They will be treated as not having submitted the assignment.
- Assignments are due by midnight of the due date as specified in the Course Calendar.
- Extensions are only granted if arrangements are made with the instructor well in advance of the due date of the assignment. No last minute extensions will be given; if a student has not completed the assignment by the due date, the student has the choice of receiving a grade based on the work completed to that point, or receiving a 5% penalty for each day (or part of a day) it takes to complete the assignment. In these cases, the assignment is not considered complete until the instructor is notified by the student of the completion of the assignment.

To submit your homework, you will run the Perl/Python script developed as part of this assignment. The output of the script will be used by the grader. You should ensure that the class group has at least read permission on the files.

NOTE: *Your assignment is not considered complete until you submit the assignment in BlackBoard. You will not need to include any source or executables in BlackBoard, but this notifies the grader that your assignment tar files are complete and ready for grading.*

Grading of the Assignment

The homework will be graded according to the following criteria:

1. The executable programs must work correctly on both UNIX (dev4) and Linux (absaroka) platforms.

2. The project hierarchy must be as specified.

3. Makefiles must exist in the appropriate directories and must function properly, supporting all targets as specified below.

4. The Perl/Python script must be used to create three *tar* files described above. There will be two binary releases appropriately named for absaroka and dev4, and one source release. These *tar* files will be copied to a separate directory for grading.

The Assignment will be graded based on 100 points as follows:

The directory hierarchy	15 points
The delivery Perl/Python script	15 points
The programs and libraries	25 points
The Makefiles and Makefile structure	45 points

Homework Requirements

Program Descriptions

This homework shall include two programs. The sole function of these programs is to get some input, process the input using some library functions that are described below, and to print out the answer.

conv_to_fahr

The first program, named conv_to_fahr, will be a simple program that prompts the user for a number of Centigrade degrees and prints out the corresponding number of Fahrenheit degrees. For example:

% conv_to_fahr

Enter number of centigrade degrees: 0

0.0 degrees centigrade is 32.0 degrees Fahrenheit.

This program will only contain the code to prompt the user for input, then call a library function (described below) to perform the conversion, then print the result. It is not permitted to directly perform the conversion in this program.

conv_to_cent

The second program, conv_to_cent, shall perform the inverse function of the above. However, rather than prompting the user for the input, the user will type the Fahrenheit temperature on the command line. For example:

% conv_to_cent 32

32.0 degrees Fahrenheit is 0.0 degrees centigrade.

This program will use a library function to perform the conversion, and another

library function (again, described below) to get the command line option.

Be sure to check for invalid or no data when taking input for the two programs.

Library Descriptions

Two libraries will be created to support the programs.

temp_conv

The temp_conv library should be built from two source files, fahr_convert.c and cent_convert.c. The fahr_convert.c function should contain one function, convert_to_fahr(); the cent_convert.c file need only contain the function convert_to_cent(). The declarations of these functions should appear as follows:

double convert_to_cent (double fahr);

double convert_to_fahr (double cent);

These declarations should be contained in an appropriate “include” file (residing in ~/614/homework1/include).

arg

The arg library will contain one file and one function. This file, called getarg.c, will contain the function get_argument(), which will have the following declaration:

int get_argument (int argc, char *argv[ ], double *return_value);

This function will place the value of atof (argv[1]) into the double pointed to by return_value if there is an argv[1]. In this case, get_argument() will return OK (defined to be 0) indicating successful completion. If argv[1] does not exist, get_argument() will return ERROR (defined to be -1). Again, an include file will exist that contains, at a minimum, this function declaration.

Perl/Python Script

A Perl or Python script will be written that will create tar files of your homework1 for grading. This script shall be called “release.pl” or “release.py”, and it should reside in your “homework1‟ directory.

The script will support two options.

1.) Binary release (-b)

The script will confirm to the user that they have requested a binary release be

generated. If the user confirms this is correct via (Y/N) on standard input, then the script will prompt the user for a hostname. (ex. absaroka)

The script will then create a *tar* file of the homework1 directory containing the root directory (homework1) and the binary directory (bin) and its contents (the executables). The filename of this tar file will contain the homework assignment number (homework1), the hostname, and the .tar extension. (ex. homework1_absaroka.tar)

2.) Source release (-s)

The script will confirm that the user has requested a source release be created. If the user confirms this is correct via (Y/N) on standard input, then the script will perform a 'make clean' on the homework1 directory structure to remove dependency and object files, and will then create a *tar* file of the entire homework1 structure. The resulting file will have a filename that includes the assignment number (homework1) and the .tar extension. (ex. homework1.tar)

Homework Implementation

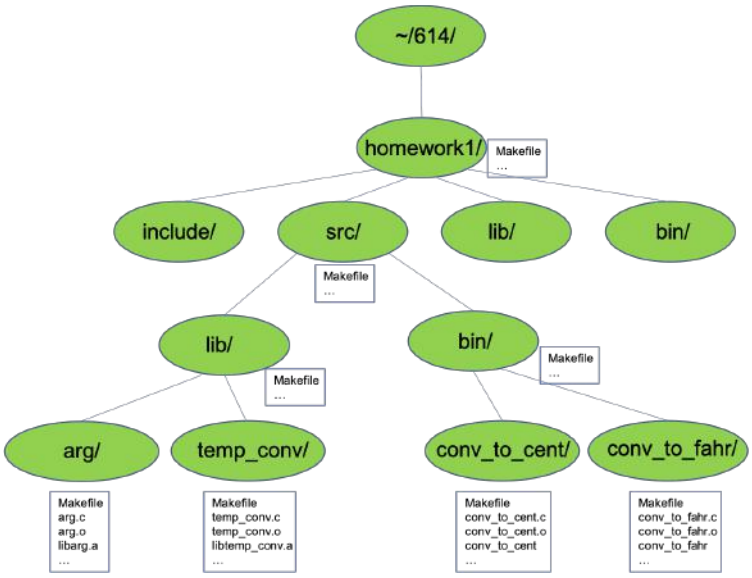
The design and implementation of the homework shall follow these guidelines:

- The homework shall consist of the two programs and two libraries organized as described in this document. There shall be a minimum of two header (include) files that shall reside in the homework1/include directory.

- The following directory structure shall be enforced:

- In the home directory, there will be a directory called "614/homework1". The Perl or Python script for the assignment will reside at this level.

- The directory hierarchy under this directory shall look as follows:



The directory "homework1/lib" will contain project-specific libraries that you shall create (libtemp_conv.a and libarg.a).

The directory "homework1/bin" will contain project-specific programs that you shall create (conv_to_cen and conv_to_fahr).

The directory "homework1/src" will contain the source code for the project, organized as shown.

The directory ``homework1/include'' will contain project-specific include files (include files that need to be shared among distinct programs).

The source for each program will be in the deepest directories of the above hierarchy. Each of these directories will contain a Makefile; the function of the Makefile will allow the Unix command “make” to make the executable (or the library) and install it in the appropriate installation directory (either homework1/bin or homework1/lib).

In addition, each directory marked with an (*) above should contain a Makefile; these Makefiles should allow 'make' to be run in these directories, the effect being to run make with the specified argument in each of the directories underneath.

All Makefiles should be called “Makefile” and should contain support for the following targets at a minimum:

- it
- install
- depend
- clean

Since your system project may be built in a different location by extracting your project from the “tar‟ files created from your Perl/Python script, you should make sure your Makefiles do not depend on a particular directory location for your project. In other words, avoid using full pathnames in your Makefiles.

Final Words

The instructor reserves the right to clarify any point that is unclear in this document, and to correct any errors.

. . . . . .

605.614 System Development in the Unix Environment