Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Fall 2013

Lab 6 - C Program Translation

Objective

- To understand the preprocessing, compilation and linkage steps that occur when the files in a C program are translated from source code to an executable program.
- To understand how C modules are implemented using .c and .h files.

Attendance/Demo

To receive credit for this lab, you must make a reasonable effort to complete the exercises, and demonstrate the code you complete.

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. Finish any exercises that you don't complete by the end of the lab on your own time.

Program Translation

Until now, we've translated our C source code into executable programs simply by selecting the Build command in Pelles C. Although this is convenient, it hides the separate steps that take place during this translation process.

When a .c file is compiled:

- The C *preprocessor* processes all *directives* (statements that begin with #, such as `#include`);
- The *compiler* then checks the syntax of the C statements. If there are no syntax errors, the compiler translates the C statements into the corresponding CPU instructions. The output of this step is known as *object code* and is stored in an *object file*.

These two steps are repeated for every .c file in a project.

After an object code file has been produced for every .c file, the linkage step occurs:

- A *linker* program combines the object files generated from your .c files and any required object code from the standard C library into an executable program. For example, if your code calls `printf`, the object code for this function must be linked to your object files.

In Exercises 1 and 2, you'll explore these translation steps in more detail. You'll also learn how C modules are implemented using .c and .h (header) files.

Background

Consider an object moving with an initial velocity *u* that is subject to a constant acceleration *a* which is aligned in the same direction as the initial velocity. The object's final velocity *v* after an elapsed time *t* is is given by the formula $v = u + at$.

Exercises 1 through 3 deal with a modular C program that models the motion of this object.

Exercise 1

1. Create a new folder named Lab 6.

2. Download lab6_ex1.zip from cuLearn to your Lab 6 folder. Right-click on the icon for this file, and when a pop-up menu appears, select Extract All... This will create a folder named lab6_ex1. Look in this folder. You should a Pelles C Project File, lab6_ex1.ppj, plus three C source files: main.c, motion.c and motion.h.

3. This project has been configured so that compilation stops after the preprocessing phase. **Important:** while working on this exercise, **do not select the Build or Rebuild menu commands from the Project menu, or click the Build button**.

4. Double-click lab6_ex1.ppj to launch Pelles C and open the project. (Use the uncompressed folder, lab6_ex1. Don't work out of the compressed folder; i.e., the zip file.)

5. In the project window (on the left or right-hand side of the IDE), locate the icon for motion.h in the Include files folder. Double-click this icon to open motion.h in an editor window. Read the code - you'll see that this file declares the prototype for a function named `calculate_velocity`.

6. Find the icon for motion.c in the project window. Double-click this icon to open motion.c in an editor window. Read the source code for this module. You'll see the preprocessor directive:

   ```
   #include "motion.h"
   ```

   followed by the definition of `calculate_velocity`.

7. From the menu bar, select Project > Compile motion.c. (Again, **do not select the Build or Rebuild commands**.) The C preprocessor will be run on this file, then compilation will stop. In the project view, you will see a message stating that POCC (the C compiler) has compiled motion.c.

8. From the menu bar, select File > Open... and navigate to the lab6_ex1 folder. From the drop-down menu. pick All files (*.*). Open motion.i. This file contains the output from the C preprocessor when it processed motion.c. Read the source code. Notice that the `#include` statement in motion.c has been replaced with the contents of motion.h.

9. Find the icon for main.c in the project window. Double-click this icon to open main.c in an

editor window. Read the source code - you'll see the preprocessor directives:

```
#include <stdio.h>
```

and

```
#include "motion.h"
```

followed by the definition of a `main` function that calls `calculate_velocity`.

10. From the menu bar, select Project > Compile main.c. (Once again, **do not select the Build or Rebuild commands**.) The C preprocessor will be run on this file, then compilation will stop. In the project view, you will see a message stating that POCC (the C compiler) has compiled main.c.

11. From the menu bar, select File > Open... From the drop-down menu. pick All files (*.*). Open main.i. This file contains the output from the C preprocessor when it processed main.c. Read the source code. Notice that the `#include <stdio.h>` and `#include "motion.h"` statements in main.c have been replaced with the contents of stdio.h and motion.h, respectively.

12. Close this project.

Exercise 2

1. Download lab6_ex2.zip to your Lab 6 folder. Right-click on the icon for this file, and when a pop-up menu appears, select Extract All... This will create a folder named lab6_ex2. Look in this folder. You should a Pelles C Project File, lab6_ex2.ppj, plus three C source files: main.c, motion.c and motion.h. (These are the same files that you used in Exercise 1.)

2. This project has been configured so that debugging information is generated during the compilation and linking phases. **Important:** while working on this exercise, do not select the Build or Rebuild menu commands from the Project menu, or click the Build button, unless you are instructed to do so.

3. Double-click lab6_ex2.ppj to launch Pelles C and open the project. (Use the uncompressed folder, lab6_ex2. Don't work out of the compressed folder; i.e., the zip file.)

4. Find the icon for motion.c in the project window. Double-click this icon to open motion.c in an editor window.

5. From the menu bar, select Project > Compile motion.c. The C preprocessor will generate motion.i from motion.c. Next, the C compiler will compile motion.i and generate an object code file. In the project view, you will see a message stating that POCC (the C compiler) has compiled motion.c.

6. Look in the lab6_ex2 folder. It will now contain a folder named output. Look inside this

folder. It now contains a file named motion.obj. This is the *object file* containing the compiled C code; that is, the CPU instructions that correspond to the C statements in motion.c.

7. Find the icon for main.c in the project window. Double-click this icon to open main.c in an editor window.

8. From the menu bar, select Project > Compile main.c. The C preprocessor will generate main.i from main.c. Next, the C compiler will compile main.i and generate an object code file. In the project view, you will see a message stating that POCC (the C compiler) has compiled main.c.

9. Look in the output folder. It now contains a file named main.obj. This is the *object file* containing the CPU instructions that correspond to the C statements in main.c.

10. There doesn't appear to be a way to run the linking phase as a separate step from within the Pelles C IDE. Instead, we'll rebuild the entire project; i.e., recompile both C files, then link them into an executable program. From the menu bar, select Project > Rebuild lab6_ex2.exe. Do not select the Build menu command, or click the Build button.

11. In the project view, you will see a message stating that POCC (the C compiler) was run twice, because main.c and motion.c were compiled independently. This will be followed by a message stating that POLINK (the linker) was run. The linker produces an executable program, by linking main.obj and motion.obj to object files in the standard library (e.g, the object code for `printf`).

12. Look in the lab6_ex2 folder. It will now contain a file named lab6_ex2.exe. This is your executable program.

13. From the menu bar, select Project > Debug lab6_ex2.exe. A window labelled Debugger - Stopped will open, displaying the source code from main.c.

14. Move the mouse cursor anywhere in this window, then right-click. From the pop-up menu, select Show disassembly. Right-click again, and select Show code bytes. You'll see the sequence of CPU instructions that correspond to each C statement, represented using *assembly language* mnemonics. To the left, you'll see the hexadecimal (base-16) numbers that represent these CPU instructions.

15. From the menu bar, select Debug > Go to execute the program.

Exercise 3 starts on the next page.

Exercise 3

Consider an object moving with an initial velocity u that is subject to a constant acceleration a which is aligned in the same direction as the initial velocity. The object's displacement s after an elapsed time t is given by the formula:

$$s = ut + \frac{1}{2}at^2 .$$

You'll now going to add a function named `calculate_displacement` to the motion module. This function will have three parameters: the object's initial velocity (measured in m/s), its constant acceleration (measured in m/s^2), and an elapsed time (measured in s). This function will return the object's displacement after the elapsed time.

1. Create a new project named lab6_ex3 inside the Lab 6 folder. The project type must be Win32 Console program (EXE). You should now have a folder named lab6_ex3 inside a folder named Lab 6. Check this.

2. Copy/paste main.c, motion.c and motion.h from your lab6_ex2 folder to your lab6_ex3 folder. **You must also add main.c and motion.c to your project:** from the menu bar, select Project > Add files to project... In the dialogue box, select main.c, then click Open. An icon labelled main.c will appear in the Pelles C project window. Repeat this for motion.c. Pelles C will automatically add motion.h to the project.

3. Build the project. It should build without any compilation or linking errors.

4. Edit motion.h, so that it contains the declaration of the function prototype for `calculate_displacement`. (Don't delete the prototype for `calculate_velocity`.)

5. In motion.c, define the complete implementation of the `calculate_displacement` function. (Don't delete the definition of `calculate_velocity`.)

6. Add code to main.c to exercise `calculate_displacement` (don't delete the code that exercises `calculate_velocity`). Your code should should print the values of the function's three arguments, the expected result, and the actual result returned by the function.

7. Compile and execute your modified version of the program. Verify that the value returned by `calculate_displacement` is correct.

Wrap-up

1. Remember to have a TA review and grade your solutions to the exercises before you leave the lab.

2. The next thing you'll do is package the project from Exercise 3 in a ZIP file (compressed folder). From the menu bar, select Project > ZIP Files... A Save As dialog box will appear. Click Save. Pelles C will create a compressed (zipped) folder named lab6_ex3.zip, which will contain copies of the the source code and several other files associated with the project.

(The original files will not be removed). The compressed folder will be stored in your project folder (i.e., folder lab6_ex3).

3. Log in to cuLearn, click the Submit Lab 6 link and submit lab6_ex3.zip. After you click the Add submission button, drag the file to the File submissions box. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status for your file is "Draft (not submitted)". You can resubmit the file by clicking the Edit my submission button. After you've finished uploading your file, remember to click the Submit assignment button. This will change the submission status to "Submitted for grading".

Posted: October 15, 2013.