Introduction to C Programming under Ubuntu using Emacs

This week's lab work will have three components.

First, I want you to enter and run a little C program most of whose code is included in this handout. The purpose of this exercise is to familiarize you with the process of entering, compiling and testing C programs using Emacs under Ubuntu Linux. Our goal is to have you finish this exercise during the scheduled lab period. If you finish early, you may use the remainder of the lab period to begin work on the other components of the lab work for the week, but we expect you to do most of the work on the remaining components of this assignment outside the scheduled lab period.

We will use the Ubuntu Linux/Unix workstations in TCL 312 for this and most of our other labs this semester. Therefore, the second component of this week's lab will be to complete several online tutorials that explain how to work with the Unix command line. These tutorials can be found at:

`http://www.ee.surrey.ac.uk/Teaching/Unix/index.html`

The tutorials were designed for use on the computing facilities of the University of Surrey, so there will be a few points where what they say does not reflect our configuration. (For example, they say you are using a shell named tcsh while you will really be using the bash shell.) Overall, however, these tutorial appear to provide an excellent introduction to Unix commands. You should complete the first five of the tutorials this week. Those of you who are already familiar with Unix may find you can just skim these materials.

The final component of this week's lab activity will be to write a short C program of your own named primes. This exercise is shamelessly stolen from Duane Bailey whose description of the program is quoted below.

> Write a program that reads an integer from the input and prints out the primes that are closest (I mean: closest) to it. Think about this very carefully. You might use this in a data structures course if youre looking to size a hash table. Be aware there are no primes less than 2. This program should be stored in a file, prime.c and compiled to an executable prime.

The program you write should use only int variables. As a result, we don't expect the algorithm you implement to be particularly clever or efficient. In particular, to determine whether a number is prime you should simply check to see if it is evenly divisible by any other number between 2 and the original number's square root.

Your primes program is the only work you will need to submit to show that you completed the lab. Instructions for doing this can be found at the end of the handout.

Emacs as IDE

Throughout the semester, we will use the Emacs text editor as a low-budget Integrated Development Environment (IDE). That is, you will use the features of Emacs not only to edit your program but also to compile it, to process error messages produced by the compiler, and to support the process of debugging the code. We will supply a separate handout that provides a quick reference for the features of Emacs that make it usable as an IDE. It is entitled "Using Emacs to Edit, Compile and Run Programs." In the remainder of this handout, we guide you through examples of the use of most of the Emacs features we will depend upon.

First, you need to launch Emacs on your system. Assuming you have already logged in to one of the Ubuntu workstations in TCL 312, you will see a collection of square icons running along the left edge of the screen. This is the "Unity Launcher." Clicking on one of the icons in the launcher starts a new program just as it does under MacOS.

With a bit of luck, one of the launcher icons will start Emacs. If it is there, the icon will show a script "E" and a pen. When you point the mouse at the icon it will display the name "GNU Emacs 23." If your launcher includes this icon, click on it to start Emacs.

If Emacs is not already in your launcher, you should instead click once on the icon at the top of the launcher. This icon activates the "Dash home". There is a field in the window that appears when you click the dash home that allows you to enter a search term. Search for "emacs". The icon for GNU Emacs 23 should appear under Applications in the search window. Drag this icon to your launcher to add it to the collection of applications you can launch directly. Then, click on its icon in the launcher to start Emacs.

Once Emacs is running, you should use it to create a simple C source file. There are several ways to do this. For now, point the mouse at the icon below the "File" menu that looks like a piece of paper with a plus sign over its lower right corner. Click once on this icon and a dialog box should appear. You can use the dialog to name the new file and determine which folder/directory it should reside in.

Let's put your new file in a folder for lab 1 within a folder for cs237.

- Click on the "Create Folder" button in the new file dialog.
- Enter the folder name "cs237" and then press return (DO NOT CLICK "OK" AT THIS STEP).
- Click on the "Create Folder" button again.
- Enter the folder name "Lab1" and then press return.
- Enter the file name "odd.c" and then click "OK" or press return.

Next type the following short program. We have deliberately included a small typo in this program (there should be a semicolon after the zero in the return statement) so that we can illustrate how Emacs handles compile-time errors. Don't worry if you add another typo (or two) as you enter the program.

```
#include <stdio.h>

int main( int argc, char * argv[] ) {
    int number;

    printf( "Enter a number: " );
    scanf( "%d", & number );

    printf( "%d is an odd number!\n", number );

    return 0
}
```

Making a Makefile

Unix includes a program called make that is designed to automate the steps required to construct large programs that include multiple source files. Our program is not large, but we will use make to automate its construction partly because it is good to get in the habit of using make and partly because it is easier to get Emacs to help you compile a program if you use make.

We could use the icon under the Emacs "File" menu to create this file as we did for odd.c, but since there are many other ways to create files in Emacs, we will us a different approach.

- While holding down the control key on your keyboard press the "x" key and then the "f" key.

Many Emacs commands can be executed by pressing sequences of keys while simultaneously depressing the control or alt/meta key. In most Emacs documentation, the need to hold down control or alt/meta is indicated by writing "C-" or "M-" in front of the other key to be pressed. That is, the sequence we just asked you to type would be described as "C-x C-f".

Once this key sequence is entered, Emacs should display a prompt in the bottom line of its screen window asking you to enter the name of the file you want to create. This area at the bottom of the Emacs window is called the minibuffer. Enter "Makefile" (including the capital M) in the minibuffer. Then press return.

Most of the lines included in a make file fall in two categories. The first type are called dependencies. They tell make which files depend on others. If a file depends on another, then the first file must be regenerated whenever the other file changes. The second type of make file lines are commands that describe how to regenerate a file when the dependency lines indicate it is necessary to do so.

For the first line of your Makefile, type:

```
odd: odd.c
```

This is a dependency. It indicates that a file named "odd" needs to be regenerated any time the contents of the file named "odd.c" changes. "odd.c" will contain the C source code for our program while "odd" will contain the executable binary code produced by the C compiler and used when we want to run the program.

The next and last line is a bit trickier. It will specify a command to be executed when odd needs to be regenerated. Such lines must start with a tab character to enable the make utility to distinguish them from dependency lines. So, on the next line, first press the tab key once and then type

```
gcc -o odd -g odd.c
```

This command says to run the C compiler (it is called gcc) to produce an output file (-o is for output) named odd from the source file odd.c. The "-g" indicates that information to assist debugging should be included in the output file.

When you are all done, your Makefile should look like:

```
odd: odd.c
	gcc -o odd -g odd.c
```

Emacs Buffers

You might have noticed (worried? panicked?) that when you created your Makefile, the text of the odd.c file you typed in earlier disappeared from your Emacs window. Don't worry. It is still there.

A single emacs window can handle editing multiple documents simultaneously. Each document is referred to as a buffer. All you need to do to get back to the text of odd.c is tell Emacs we want to see a different buffer.

First, lets make sure your Makefile is saved to disk. As usual, there are multiple ways to accomplish this. The fastest way is to type the sequence C-x C-s. Eventually, this key sequence should become embedded in your muscle memory. Until that happens, you can either select "Save" from the Emacs "File" menu or press the "Save" icon at the top of the Emacs window (it should appear just below the "Buffers" menu).

The big advantage of the last option, is that it shows that Emacs has a "Buffers" menu. If you depress the mouse on the header for this menu, a list of all of the buffers Emacs is working with will appear. This list should include both Makefile and odd.c. It will also include several buffers you did not even know about. Select "odd.c" from this menu and your C code should reappear.

In addition to switching easily from one buffer's contents to another, it is possible to display several buffers in a single window. To see this:

- Select "Split Window" from the Emacs File menu (or just type C-x 2).
- Select "Makefile" from the Buffers menu (or just type C-x b to switch buffers).

You should now see the contents of both of the files you have created in a single window.

Compiling under Emacs

Many of you have used Emacs to edit programs and then compiled your programs by typing commands within a Unix terminal window. We could compile odd.c in this way, but for several reasons it is better to invoke the compiler directly from within Emacs.

To do this,

- Select "Compile" from the Emacs Tools menu.

Emacs should display the command "make" in the minibuffer. This is the command used to cause the make program to compile odd.c and produce an executable file named odd. If you had not created a Makefile or needed to use a different command, you could edit the command displayed in the mini buffer. Since we have created an appropriate Makefile, you can just press return at this point to start the compilation process.

Whenever you tell it to compile, Emacs checks to see if there are any buffers that you have modified but not saved. For each such buffer it asks you whether or not to save that buffer before doing the compile (this is one of the advantages of compiling this way instead of using a separate terminal window). If Emacs asks you such questions about odd.c or Makefile, you should answer by typing "y" for "yes".

If you typed exactly what we asked you to type into the odd.c file, the compiler should fail with a syntax error because of the missing semicolon on the last line of the code. Emacs will display the error messages in one of the sub-windows of its window (probably replacing one of the two buffers you had displayed). For a short program like odd.c, it should be pretty easy to find the line referred to by any such error message, but this can be difficult when you are working with large source files. Fortunately, Emacs provides a convenient mechanism for processing compiler error messages.

- Enter the key sequence `` C-x ` ``. The quote entered has to be of the single, left-leaning variety probably found near the upper-left corner of your keyboard rather than the double or straight, single quotes found near the right side of the keyboard.

Emacs will now place a little arrow just to the left of the first error message and display the text of the file containing the error with the cursor positioned on the line mentioned in the error message. If the only error in your odd.c file is the deliberate one, the cursor will be positioned by the ending curly brace for the main method. This is just after the place where the missing semi-colon should appear.

If you managed to make more mistakes, you can fix your first error and then press `C-x `` again to position the cursor at the location of the second error. You can repeat this process to work through as many error message as you like. Since a few real mistakes confuse the compiler enough that it reports errors on lines that are really correct, it is usually best to select “Compile” from the Tools menu again after fixing just a few syntax errors.

Using these mechanisms, edit odd.c and keep compiling until all errors are eliminated. You will know you are done when the compiler output in the Emacs windows ends by saying that compilation “finished” rather than saying that it “exited abnormally”.

Using GDB under Emacs

Now that your program is compiled, you can use Emacs to run it under the control of the Unix debugger, gdb (Gnu DeBugger). The easiest way to accomplish this is to select the Debugger item in the Emacs Tools menu (if you prefer keystrokes, type “M-x gdb” followed by the return key). Emacs will then display the command it plans to use to start gdb in the minibuffer. Pressing return at this point will start gdb.

Emacs displays several lines of gdb startup messages in one of the buffers in your Emacs window followed by the gdb prompt “(gdb)”. To run your program, just type “run” at this prompt. Enter some number when your program asks you to and it should inform you that the number is odd (even if it is even).

Odd vs. Even

To make your program a bit smarter, I would like you to add a function named isOdd that tests whether a number passed to it as a parameter is odd. This test is so simple that it really is not worth defining a method to make this test. For this exercise, however, we will define a separate method because doing so mimics the structure you should use when writing your “prime” program.

In Java, isOdd would return a boolean value. C has no booleans. Instead, the int value 0 is interpreted as false and any non-zero value is treated as true. Therefore, reasonable code to define isOdd would be

```
// Give names to the ints used to represent boolean values
#define TRUE 1
#define FALSE 0

// Determine if a number is odd
int isOdd( int value ) {
     if ( value % 2 == 0 ) {
          return FALSE;
     } else {
          return TRUE;
     }
}
```

(Yes, the entire method's body could be replaced by the single statement "return value % 2;", but the less concise version will help demonstrate the behavior of gdb a bit later.)

- Enter this code between the #include at the start of odd.c and the definition of main. (You may have to first use the Buffers menu to make odd.c visible again.)

Note that we are trying to set a good example by including comments and using defines to associate names with constant values. You may want to clean up the style of the code you have already entered by adding some comments (at least one with your name) or fixing the indentation.

- Clean up the code you entered previously as needed.
- Add an if statement to your code so that the program only says that the number input is odd if it actually is. If you cannot see all of your code at this point because it does not fit within half the emacs window you can:
    1. scroll up and down using the wheel on your mouse, or
    2. scroll up and down using the keystroke sequences described in the Emacs quick reference handout we provided, or
    3. enlarge the Emacs window with the mouse, or
    4. make odd.c the only buffer displayed in the Emacs window by typing C-x 1 while the cursor is positioned somewhere within odd.c
- Compile the modified program until all syntax errors are removed.

Using GDB

Now that your program's control structure is a little bit more interesting, we can use it to illustrate how gdb can be used to track a program's execution. Your program is probably completely correct at this point, but if you use your imagination a bit you should be able to see how the techniques for using gdb presented here could be used to debug an incorrect program.

- First, just run your program using gdb as we did earlier:
    - Select Debugger from the Emacs Tools menu.
    - Press return when Emacs displays the command "gdb –annotate=3 odd" in the minibuffer.
    - Type "run" at the gdb prompt.
- Next, make sure that the odd.c buffer is visible in your Emacs window.
- Set a breakpoint on the first line of code after the scanf in main (this should be an if statement). There are two ways you can do this:
    1. If you like using keystrokes, position the mouse anywhere within the line where you want the breakpoint (the first line of code after the scanf in main) and then type C-x followed by hitting the space bar once.
    2. If you like using the mouse, click the mouse once while pointing at the dark gray margin just to the right of the scroll bar at the left edge of the Emacs window with the mouse aligned vertically with the line of text where you want to place the breakpoint. A little red circle (think stop sign) will appear to represent the new breakpoint. Clicking repeatedly will toggle the breakpoint on and off.

- Type "run" at the gdb prompt to run the program again.

Because you set a breakpoint, as soon as you respond to your program's "Enter a number" prompt, gdb should stop your program's execution and display the "(gdb)" prompt to let you know it is ready to accept debugging commands. It should also indicate the point at which execution was suspended by displaying an arrow at the left edge of the line in odd.c that follows the scanf.

- Enter the gdb command "print number" at the gdb prompt. Gdb should display the value of the variable number (i.e., the number you just typed in).
- Next enter the gdb command "step". This tells gdb to execute the next step in your program. Assuming your breakpoint was placed on an if statement that uses isOdd, this should put you at the first line of code in the isOdd method. If not, type "step" (or just "s") until you get to isOdd.
- Once you are in isOdd, type "print value" to verify that the correct parameter value was passed.
- Type "step" once more to see which branch of the if statement in isOdd gets chosen.
- Finally, type "cont" for continue to tell gdb to continue the execution of your program until it completes or encounters another breakpoint.

The Rest

That concludes our little introduction to using Emacs as a C IDE. Recall that you have two other tasks to complete to finish this assignment. Read through the online Unix command-line tutorials from the University of Surrey and write the prime program described on the first page. The process of entering, compiling, and running/debugging your prime program should resemble what you just did with isOdd. In particular, the prime program should probably include an isPrime method. You might want to first write and debug this method using a simple main program like the one in odd.c before writing code to look for closest primes.

The University of Surrey tutorials do assume you have access to a terminal window. You can create one on your Ubuntu desktop much like you created your first Emacs window.

- There should be a "terminal" icon in your launcher. If so, just click on it.
- If terminal is not in your launcher, click on the Dash home icon (at the top of the launcher), type "terminal" as a search term, and then drag the terminal icon to your launcher.

You will also need a terminal window to submit your work once you have completed the prime program. Within the terminal window you should use the cd command to make the directory containing prime.c your current working directory. Then type the command:

```
turnin -c 237 prime.c
```

Respond to the prompts appropriately and your code should be submitted.

Whenever you leave the lab, please remember to log out. You can do this by selecting the "Log Out" item from the menu that appears when you depress the mouse while pointing at the little gear-like icon that appears in the upper right corner of your screen.