Wesleyan University, Fall 2023, COMP 211

Lab 1: Introduction and first C program

---

1. Overview

The goals of today's lab are the following:

(1) To make sure you have setup your programming environment if you haven't done this already.
(2) To understand at a high level the organization of the file system on your computer.
(3) To learn the basics of using a terminal-based text editor.
(4) To create, compile, and run your first C program, `helloworld.c`.

Today, we'll mostly work through the lab together with me showing my steps on the projector. Very soon, we'll switch to me just overviewing the lab and letting you work by yourself or with others, with me and the CAs available and very willing and happy to help you. Working this way will help you get accustomed to working out by yourself how to attack a programming problem and implementing a solution.

2. Choosing your environment

Based on the email I sent out earlier, you should already have an environment set up in which to work. For those students without a laptop, there are macOS machines in the labs that you can use. Since the computers are wiped daily, you should make sure to copy any files saved there back to your Wesleyan Google Drive or Microsoft One Drive, email them to yourself, or save them to a USB stick.

3. File system organization

During this class we will be focused on working from a terminal. You can think of the terminal as a window into your file system with the ability to launch programs. You are actually already familiar with working with the file system from a different perspective, using what you know as folders. Folders really just correspond to directories on the file system. The file system on your computer is arranged in a structure like an inverted tree, with the root of the file system at the top of the tree and the leaves of tree as files. The file system can thus be thought of as set of nested directories (or folders), starting at the root directory.

The root of the file system is written as a slash, "`/`". Note that you should ignore the quotations here. Your current working directory is written as "`./`." To give an absolute path you start from the root of the file system beginning your path with "`/`". To give a relative path you start from your current working directory, beginning your path with "`./`" and specifying how to get to the location of interest starting from where you are now. For instance to refer to two directories higher than where you are located you could use the relative path "`./../..`", using the "`..`" notation.

Knowing where you are within the filesystem and what is in your current working directory is important since when we run commands from the command line we normally give the name of input files (e.g., C files), and it is important that the C compiler knows where to find those files.

The following commands are useful for locating where you are in the file system, determining what files are currently in a directory, and creating and moving files.

`ls`: List the contents of your current directory.

`ls -a`: List the contents of your current directory, giving additional useful file information.

`pwd`: Print working directory. This is the complete path of where you currently are in the file system starting from the root.

`cd PATH`: Change directories to the directory specified by `PATH`.

`cp PATH_TO_FILE1 PATH_TO_FILE2`: Copy the the file located at location `PATH_TO_FILE1` to become the file named at location `PATH_TO_FILE2`. This results in two copies of the file, at both locations.

`mv PATH_TO_FILE1 PATH_TO_FILE2`: Move the file located at location `PATH_TO_FILE` to become the file named at location `PATH_TO_FILE`. This results in only one copy of the file at the second location.

`rm PATH_TO_FILE`: Remove the file specified at location `PATH_TO_FILE`.

`mkdir PATH_TO_DIRECTORY_NAME`: Create a new directory located at `PATH_TO_DIRECTORY_NAME` with the name given.

`man COMMAND_NAME`: The `man` pages are a useful resource, giving information about various unix commands and are available at the command line.

Now let's create and organize a directory for this class. It will be very important to maintain separate directories for separate code projects, to avoid overwriting code and other confusions, so let's get into the habit of doing this now.

Step 1: Create a directory to hold the files for this class with the command `mkdir comp211`. Change your location within the file system to this directory with the command `cd comp211`.

Step 2: Create a directory to hold your lab files with the command `mkdir labs`. Change to this directory with the command `cd labs`.

Step 3: Create a directory to hold your files for today's lab with the command `mkdir lab1`. Change to this directory with the command `cd lab1`. We call `lab1` a subdirectory of the directory `labs`.

4. Working with a text editor

In lab today, we will focus on one particular command-line text editor, vim. You may be familiar with other command-line editors, such as emacs, or other editors such as Atom or Eclipse. You

```
Name.
Class year.

Briefly describe your prior programming experience. What programming languages
have you used?

Something interesting you would like to tell me.  This can be about
you, about something you find interesting, whatever!
```

FIGURE 1. Information to enter in your file `about.txt`.

should not use an editor such as Microsoft word: this will include characters and formatting that will completely break your code. You **should** use an editor designed for programming, otherwise you will have white space issues and other issues.

Generally, I recommend getting familiar with at least one command-line editor. `vi` (of which `vim` just gives more commands) is a good choice since any terminal on any Linux or Mac OS machine, whether locally or remotely accessed comes with `vi` installed, but may not contain any other editor installed by default.

Now `vim` itself is just a program (written in C), and as such has its own location in the file system located in the directory `/usr/bin/`. I could run `vim` by specifying the full path with `/usr/bin/vim`, but for convenience, the programs in the directory `/usr/bin` do not need the full path specified, since this directory is the main directory holding executable commands on your system.

The `vim` editor can be a bit confusing when you first learn it, since when you use it to open a file, you can't immediately start editing. Instead, you must first enter a command to indicate that you would like to start editing the file; similarly, you must enter a command to indicate that you are done editing.

To open a file, e.g., `test.txt` using `vim`, in a terminal type the following at the prompt:

```
vim test.txt
```

In `vim`, the following keys in command mode are useful. Any of these keys when pressed will cause you to leave command mode. To re-enter command mode, press the `escape` key.

- `i`: puts you in insert mode, letting you insert text just before the cursor position. `i` inserts at the start of a line.
- `a`: puts you in insert mode, letting you insert text just after the cursor position. `a` inserts at the end of line.
- `o`: creates a new line below the line where the cursor is and enters insert mode.
- `x`: deletes the character under the cursor.
- `w`: writes the file to disk. I.e., the file is saved.

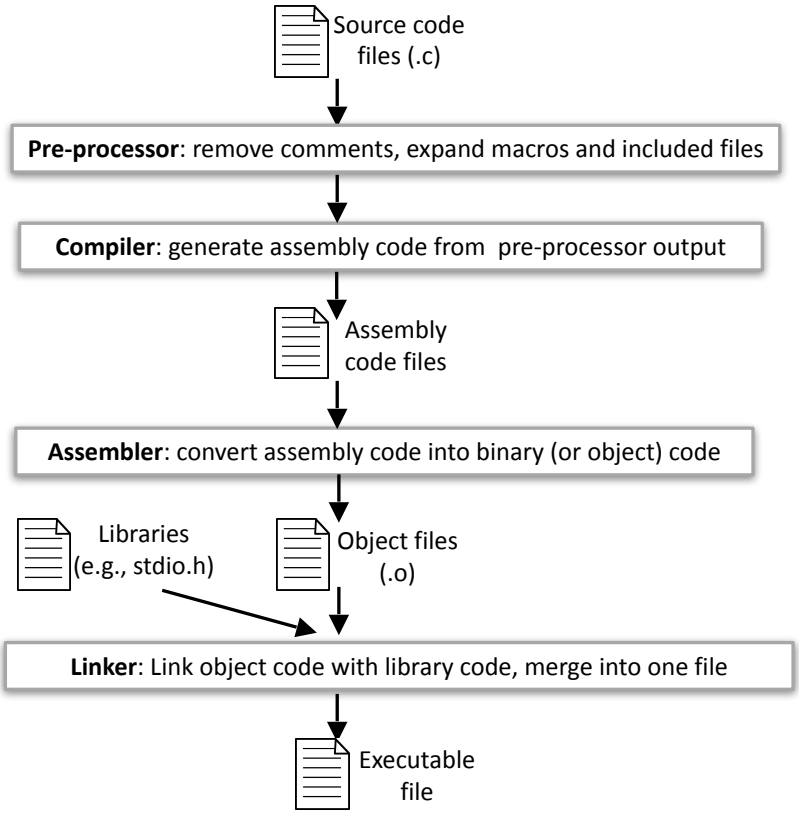

FIGURE 2. Overview of the compilation process for C programs.

`q:` quits `vim`. You can combine commands, such as typing `wq` to write the file to disk and quit at the same time.

The `vim` editor is extremely powerful, and has many more key combinations to help you quickly move around a file and manipulate text. See the resources link on the class webpage and tutorials below to find out more about how to use vim effectively. Some vim tutorials:

```
https://www.openvim.com/
http://www.ee.surrey.ac.uk/Teaching/Unix/
http://www2.geog.ucl.ac.uk/~plewis/teaching/unix/vimtutor
```

Now, using `vim`, create a file named `about.txt` in the `lab1` directory you created earlier, containing the information shown in Figure 1, replacing the questions with your answers. First, change `Name` to your name. Then provide your `Class year` at the appropriate location. Then answer the question about prior programming experience on a line following the question, leaving a blank line between the question and your answer. Finally, delete the `Something interesting` block of text and replace it with your response. Please email to me the file you create by the end of lab: `vumanfredi@wesleyan.edu`.

```c
/* Include functions from standard input/output library such as printf */
#include <stdio.h>

/* Every program you write must have a main: this is where your C program
 * begins operating. In C, main is a special function.
 */
int main(void)
{
    /* The main function calls the library function printf to print hello
     * world.  \n is an escape sequence meaning new line. Escape sequences are
     * useful for hard to type and invisible characters.
     */
    printf("Hello World!\n");

    return 0;
} /* Statements of main function are enclosed in curly brackets */
```

PROGRAM 1. helloworld.c.

5. CREATING AND COMPILING YOUR FIRST C PROGRAM: HELLOWORLD.C

In your `lab1` directory, open a new file using vim by typing `vim helloworld.c`, which will contain your first C program. In this file, type in the program shown in Program 1. Note that everything between `/* */` is considered a comment in C, and you need not type the comments in.

In the terminal, making sure you are in the same `lab1` directory as the file you just created, type the following at the command-line to compile your program.

```
gcc helloworld.c
```

This compiles your C program and creates a your program executable. Since we didn't specify the name of an output file, gcc automatically uses `a.out` as the name of the output. Now type the following to run your program. The preceding `./` specifies the directory path to the file, indicating that it is in current directory.

```
./a.out
```

Figure 2 overviews what happens during compilation; as the course progresses, we will see some of the intermediate files generated once our programs become more complicated.