CS 1313 010: Programming for Non-majors, Fall 2017
Programming Project #1: Thinking of a Number
Due by 10:20am Wednesday September 6 2017

This first assignment will help you learn to use the Linux computers administered by OU Information Technology for the College of Engineering. An account should have been set up for you automatically. If you have trouble accessing your account, then you **MUST contact Dr. Neeman by no later than Wednesday August 30.** You **MUST** be enrolled in CS1313 to get an account.

Actions and commands that you should perform or type are in the **`computer boldface`** font. Your user name is denoted here as `yourusername`, but will actually be your OU4+4 ID (the first 4 letters of your last name **in all lower case**, followed by the last 4 digits of your OU ID number.

For each step in this project specification, you should do the following, **IN THE FOLLOWING ORDER**:

1. **READ** the full text of that step.
2. **DO** what the full text of that step says to do.
3. **ASK** questions about anything that's unclear about that step.

The requirements for this assignment are listed on the following pages.

I. LOG IN

1. Connect and log in, from a computer that you're sitting at, to:
   `ssh.ou.edu`
   (a) **From a PC in Carson Engineering Center 205, 206 or S-18:**
      i. Press **Ctrl**-**Alt**-**Delete** simultaneously.
      ii. This will take you to the Logon Banner screen. Click the **OK** button or press the **Enter** key on the keyboard.
      iii. This will take you to the logon screen. **CAREFULLY CHECK** whether a username has already been selected for you — that is, just above the text box for the password, is there a text box for a username, or your username (your OU4+4), or some other username (that is, the **WRONG USERNAME**)?
      iv. If the wrong username is shown, then below the text box for the password, click the **Switch User** button.
      v. If you clicked the Switch User button, you should then click the **Other User** button.
      vi. Type your OU4+4 username and password in the appropriate text boxes.
      vii. Click the → button or press the **Enter** key on the keyboard.
      viii. Wait patiently while the PC logs you in and starts up various features.
      ix. There are two ways to run the program named PuTTY, which lets you log in remotely to the computer that we're going to use.
      - From the Start menu:
        A. The Start button (a little Windows logo) will appear in the lower left corner of the screen. (Various other windows may also pop up). Click on the **Start** button.
        B. When the Start menu pops up, scroll down to **Applications** and click on it.
        C. Click on **PuTTY** This will pop up a window titled PuTTY Configuration
      - From the CS1313 website:
        A. On the bottom left of the screen, click on the icon for your preferred web browser (for example, Firefox, Chrome, Microsoft Edge).
        B. In the web browser, go to the CS1313 website.
        C. Scroll down almost to the bottom of the page, to the section titled USEFUL INFORMATION
        D. Click on the following link: **Downloading a Secure Shell Client to Your Desktop**
        E. Follow the instructions to download and run `putty.exe` (the 64-bit version).

x. That'll pop up a window titled
`PuTTY Configuration`
On the left side of the window, under the heading
`Terminal`,
is an entry
`Keyboard`
Click on it.

xi. This will bring up a section titled
`Options controlling the effects of keys`
Under this is a heading
`Change the sequences sent by:`
and beneath that is
`The Backspace key`
Click to select
`Control-H`

xii. A bit lower is
`The Function keys and keypad`
Click to select
`Xterm R6`

xiii. On the left side of the `PuTTY Configuration` window, below
`Terminal`
and its subentries, is an entry
`Window`
Click on it.

xiv. This will bring up a section titled
`Options controlling PuTTY's window`
The first option is
`Set the size of the window`
Choose **EXACTLY** 80 columns (which is the default) and **EXACTLY** 40 rows (which **isn't** the default, so you'll have to change it).

xv. Below that is an option
`When window is resized:`
Click on
`Forbid resizing completely`
NEVER RESIZE THE PUTTY WINDOW USING THE MOUSE, AND NEVER CLICK THE MAXIMIZE BUTTON AT THE TOP RIGHT OF THE WINDOW.

xvi. On the left side of the `PuTTY Configuration` window, below
`Window`
is a subentry
`Translation`
Click on it.

xvii. This will bring up a section titled
`Options controlling character set translation`
Below that is a section titled
`Character set translation`
and below that is a menu titled
`Remote character set:`
In that menu, select
`UTF-8`

xviii. On the left side of the `PuTTY Configuration` window, the first entry is
`Session`
Click on it.

xix. This will bring up a section titled
`Basic options for your PuTTY session`
The first option is
`Specify the destination you want to connect to`
and immediately below it is
`Host Name (or IP address)`
In the text box immediately below that, type the full name of the computer that you are logging into:
`ssh.ou.edu`

xx. Immediately below this is
`Connection type:`
below which make sure that
`SSH`
is selected (it should be the default, so you shouldn't need to click on it).

xxi. At the bottom right of the `PuTTY Configuration` window, click
`Open`

xxii. If a `PuTTY Security Alert` window pops up, click
`Yes`

xxiii. When prompted to `login as`, type your OU4+4 and press **`Enter`**.

xxiv. When prompted for your password, type your password (which should be your OU4+4 password) and press **`Enter`**. **NOTHING WILL APPEAR AS YOU TYPE YOUR PASSWORD.** This is normal for Unix.

xxv. **NOTE:** When you log out of a Windows PC in Carson 205, 206 or S-18, your PuTTY settings will be lost, so you'll have to redo all these settings each time.

(b) **From your own Windows PC, or from a Windows PC not in Carson 205, 206 or S-18:** You will need to download the PuTTY SSH client onto your desktop. Follow the instructions above, starting from Step I.1(a)ix.

(c) **From your own Apple Mac or a Unix/Linux computer, or a Mac or Unix/Linx computer not in Carson 205, 206 or S-18:**
You should be able to access the command `ssh` from the Unix command line, like so:
> **`ssh yourusername@ssh.ou.edu`**
where `yourusername` is your user name (that is, your OU4+4).
You may need to resize the terminal window to 80 columns by 40 rows.

2. If you cannot log in to `ssh.ou.edu`, try logging in to:
   `polk.ou.edu`
   It turns out that `ssh.ou.edu` is an *alias* for some PCs that are named for dead presidents: when you log in to `ssh.ou.edu`, you'll actually get logged into one of these.
3. Once you log in, you'll get some text, and then a *Unix prompt* — probably a greater-than sign — with the text cursor after it, like so:
   > ▮
   (The text cursor will probably be green.)
   There may be some information before the prompt character, such as the name of the computer that you've logged in to (which may be different from `ssh.ou.edu`), your user name, and so on. For purposes of CS1313 course materials, we'll generally use the greater-than sign > to indicate the Unix prompt.
4. Check the lines of text immediately above the Unix prompt. If there are lines of text that read something like:

   ```
   No directory /oushomes/Student/yourusername!
   Logging in with home = "/".
   ```

   then you should log out immediately by entering `exit` at the Unix prompt (you may have to do this twice to log out fully), and then log back in.
5. Check to be sure that you're in your *home directory* (a *directory* in Unix is like a folder in Windows, and your *home directory* in Unix is like your desktop in Windows):
   > **pwd**
   `/oushomes/Student/yourusername`
   This command is short for "Print working directory;" that is, "print the full name of the directory that I'm currently in." If your current working directory is just a slash (which means the *root directory*, which is like `C:\` in Windows), rather than something like
   `/oushomes/Student/yourusername`
   then you should log out immediately by entering `exit` at the Unix prompt (you may have to do this twice to log out fully), and then log back in.

   You may find that your home directory is something like:
   `/oushomes/FacStaff/yourusername`
   This is fine, and most likely is because you either currently work for OU or have worked for OU in the past.

II. SET UP (FIRST TIME LOGGING IN ONLY)

1. At the Unix prompt, type **EXACTLY** the bold text below, excluding the greater-than sign, which indicates the Unix prompt (all commands **MUST** be followed by pressing **Enter**):

   > **cp ~neem1883/.login.student ~/.login**

   This command means: "Copy the file named `.login.student` that's in Dr. Neeman's home directory into my home directory, and name my copy `.login`." You **WON'T** have to do this for future logins.

   NOTICE:
   - The Unix copy command is `cp`.
   - The first filename after `cp` is the *source* (the thing that you're making a copy of); the second is the *destination* (the name and/or location of the copy).
   - Dr. Neeman's account name on the IT Linux computers is `neem1883`, **NOT** `hneeman`.
   - The filenames `.login.student` and `.login` both begin with a period (**very important**). They are pronounced "dot login dot student" and "dot login," respectively.
   - In Unix, filenames are *case sensitive,* meaning that it matters whether you use *upper case* (capital) or *lower case* (small) for each letter in a filename.
   - In Unix, filename pieces are separated by slashes, **NOT** by backslashes as in Windows.
   - The symbol `~` (known as a *tilde*, pronounced "TILL-duh") denotes **your** home directory (another way to denote your home directory is `~yourusername`).
   - The substring `~neem1883` means "the home directory of the user named `neem1883`."
   - If for some reason this doesn't work, try
     cp /oushomes/FacStaff/neem1883/.login.student ~/.login

2. Enter the following command:
   > **cp ~neem1883/.profile.student ~/.profile**
   You **WON'T** have to do this for future logins.

3. Enter the following command:
   > **cp ~neem1883/.bashrc.student ~/.bashrc**
   You **WON'T** have to do this for future logins.

4. Enter the following command:
   > **cp ~neem1883/.cshrc.student ~/.cshrc**
   You **WON'T** have to do this for future logins.

5. Enter the following command:
   > **cp ~neem1883/.tcshrc.student ~/.tcshrc**
   You **WON'T** have to do this for future logins.

6. Enter the following command:
   > **cp ~neem1883/.nanorc.student ~/.nanorc**
   You **WON'T** have to do this for future logins.

7. Enter the following command:
   > **cp ~neem1883/.vimrc.student ~/.vimrc**
   You **WON'T** have to do this for future logins.

8. Enter the following command:
   > **`source ~/.profile`**
   This command means: "Execute the Unix commands that are in the file named `.profile`, which is in my home directory." You **WON'T** have to do this for future logins.

9. Create a *subdirectory* named `CS1313`, like so:
   > **`mkdir CS1313`**
   NOTICE: In the subdirectory name `CS1313`, the `CS` **MUST BE CAPITALIZED**; that is, the directory's name is "capital-C capital-S one three one three" with no spaces or other characters in between. This command means: "Create a directory named `CS1313` as a subdirectory inside the directory that I'm currently in" (it's like creating a new folder named `CS1313` on your desktop in Windows). You **WON'T** have to do this for future logins.

10. Confirm that you have successfully created your `CS1313` directory by *listing* the directory's contents:
    > **`ls`**
    `CS1313`
    This command means: "List the names of the files and subdirectories in my current working directory." **NOTICE** that the command is "ell ess" — that is, small-L small-S — rather than "one ess" and that `ls` is short for "list."

11. Set the *permissions* on your `CS1313` directory so that only **you** can access it:
    > **`chmod u=rwx,go= CS1313`**
    This command means: "Change the *mode* (list of permissions) on my subdirectory named `CS1313` so that **I** (the **user**) can read files in it, write files in it, and go into (execute) it, but nobody else can." Your `CS1313` directory is now accessible **only to you.** The only other people who can access it are the *system administrators* (*sysadmins* for short) of these computers; that is, IT staff. **The instructor, the TAs and your CS1313 classmates CANNOT access your `CS1313` subdirectory.** You **WON'T** have to do this for future logins.

12. Log out of the Linux computer by entering `exit` at the Unix prompt (you may have to do this twice to log out fully). Once you have completed the setup steps in this section, you **WON'T** have to do them again when you log in later.

III. COPY PP#1 FILES FROM DR. NEEMAN'S HOME DIRECTORY TO YOURS

1. Log in again.
2. Check to be sure that you're in your home directory:
   > **pwd**
   /oushomes/Student/yourusername
3. List the files in your home directory, especially to be sure that you have a `CS1313` subdirectory in your home directory:
   > **ls**
   CS1313
   [There may be other files and subdirectories listed as well.]
4. Go into your `CS1313` subdirectory:
   > **cd CS1313**
   This command means: "Change the working directory to `CS1313`, which is a subdirectory of the current working directory." (This is like double-clicking on a folder icon in Windows.)
   NOTE: ALL CS1313 project files MUST reside in your `CS1313` subdirectory, for this and ALL future programming projects.
5. Check to be sure that you're in your `CS1313` subdirectory:
   > **pwd**
   /oushomes/Student/yourusername/CS1313
6. List the files in your `CS1313` subdirectory in your home directory:
   > **ls**
   [There may be files and subdirectories listed.]
7. **ASIDE:** To learn more about a particular Unix command, type:
   > **man** *commandname*
   For example, try
   > **man chmod**
   which will give you the online *manual page* for the `chmod` command. The output of `man` goes through another command, `more`, which shows one screenful at a time. To get the next **screenful**, press the spacebar; to get the next **line**, press **Enter**. To **quit** `more`, press **Q**.
8. Copy the *C source file* named `my_number.c` from Dr. Neeman's home directory into your `CS1313` directory:
   > **cp ~neem1883/my_number.c .**
   This command means: "Copy the C source file named `my_number.c` from Dr. Neeman's home directory into the directory that I'm currently in." **NOTICE THE PERIOD** at the end of this command; it means "the directory that I'm currently in" and is **VERY IMPORTANT.**
9. Confirm that you have `my_number.c` in your `CS1313` directory by *listing* the directory's contents:
   > **ls**
   my_number.c
   This command means: "List the names of the files and subdirectories in my current working directory." **NOTICE** that the command is "ell ess" — that is, small-L small-S — rather than "one ess" and that `ls` is short for "list."

10. Copy the *makefile* named `makefile` from Dr. Neeman's home directory into your `CS1313` directory:
    > **`cp ~neem1883/makefile .`**
    Again, notice the period at the end of this command.
11. Confirm that you have `makefile` in your `CS1313` directory by listing the directory's contents:
    > **`ls`**
    `makefile my_number.c`
12. **NOTE:** You **WON'T** do this kind of copying for future programming projects; in future you will write your own programs, typically starting from an empty source file.

IV. LOOK AT, MAKE (COMPILE) AND RUN THE ORIGINAL VERSION OF THE PROGRAM

1. If necessary, repeat III.1-4, then definitely repeat III.5-6.
2. For your own understanding, look at the contents of the C source file:

   > **cat my_number.c**

   This command means: "Output the contents of the text file named `my_number.c` to the terminal screen." **NOTICE** that the command to output the contents of a text file to the terminal screen **without** using the `more` command is `cat`, which is short for "concatenate," a word that means "output one text file after another in sequence." The output of the `cat` command goes to the terminal screen, and in this case, we are only concatenating a single text file, so we're simply outputting the text file's contents to the terminal screen.
   If the contents of the file exceeds the height of the `PuTTY` window, then you can scroll up or down using the scrollbar on the right side of the window.
3. For your own understanding, look at the contents of the makefile:

   > **cat makefile**

4. *Make* (compile) the *executable* program for Dr. Neeman's original version of `my_number.c`:

   > **make my_number**
   ```
   gcc -o my_number my_number.c
   ```

   NOTICE:
   - In the `make` command, the *command line argument* `my_number` is the name of the *executable* (the file that can actually be run) that you are making.
   - The `make` command runs the C compiler `gcc` to compile the source file named `my_number.c`. In the compile command, the *command line option*
     `-o my_number`
     indicates that `my_number` is to be the name of the executable; if that option had been left out, then by default the name of the executable would be `a.out` ("the output of the assembler"), **WHICH WOULD BE BAD.**

5. Once you have successfully compiled Dr. Neeman's original version of the program, **RUN** the executable several times, using the following values as inputs, in this order:
   (a) an integer value less than 1;
   (b) an integer value greater than 10;
   (c) an integer value between 1 and 10 (inclusive), but far from 5;
   (d) an integer value close to 5 (within 1);
   (e) 5 (the correct value).

   In Unix, you run an executable by entering the name of that executable at the Unix prompt:

   ```
   >  my_number
   ```

   The sequence of runs will look similar to this:

```
>  my_number
Let's see whether you can guess the number that I'm thinking of.
It's between 1 and 10.
What number am I thinking of?
0
Hey!  That's not between 1 and 10!
>  my_number
Let's see whether you can guess the number that I'm thinking of.
It's between 1 and 10.
What number am I thinking of?
11
Hey!  That's not between 1 and 10!
>  my_number
Let's see whether you can guess the number that I'm thinking of.
It's between 1 and 10.
What number am I thinking of?
2
Bzzzt! Not even close.
>  my_number
Let's see whether you can guess the number that I'm thinking of.
It's between 1 and 10.
What number am I thinking of?
6
Close, but no cigar.
>  my_number
Let's see whether you can guess the number that I'm thinking of.
It's between 1 and 10.
What number am I thinking of?
5
That's amazing!
```

V. EDIT THE C SOURCE FILE TO CREATE YOUR OWN UNIQUE VERSION

1. If necessary, repeat III.1-4, then definitely repeat III.5-6.

2. Now that you've run Dr. Neeman's original version of the program, it's time to modify your copy of the source file `my_number.c` to create a version that's uniquely yours. Using the text editor named `nano`, edit your copy of `my_number.c`:
   > **`nano my_number.c`**
   This command means: "Edit the text in the file named `my_number.c` that's in my current working directory, using the text editor program named `nano`." Your TA will be happy to help you learn how to use the `nano` editor, and you can also find links on the CS1313 website to webpages describing how to use `nano`. These links are near the bottom of the page, in the section titled "Useful Information." If you'd prefer to use another editor (for example, `vi`, `emacs`), you may do so, but your TA won't have time to help you learn it. **UNDER NO CIRCUMSTANCES SHOULD YOU EDIT FILES ON A WINDOWS COMPUTER IF THEY ARE TO BE USED ON A UNIX COMPUTER.** Windows editors often embed invisible special characters in text files, and some Unix compilers choke on them.

3. In `nano`, notice the little help messages at the bottom of the screen:

   ```
   ^G Get Help ^O WriteOut ^R Read File ^Y Prev Pg  ^K Cut Text   ^C Cur Pos
   ^X Exit     ^J Justify  ^W Where is  ^V Next Pg  ^U UnCut Text ^T To Spell
   ```

   For example, consider `^W Where is`
   This means that you should press **`Ctrl`**-**`W`** (the caret `^` indicates the **`Ctrl`** key) to search for a particular string of characters. Another example: `^C Cur Pos` is short for "Cursor Position" and causes `nano` to tell you what line number the cursor is located at. Another example: `^K Cut Text` means "delete the line that the cursor is currently on."

4. Using the text editor, make the following changes to `my_number.c`:
   (a) In the *comment block* at the top of the C source file, change the author name and e-mail address, and the lab information, so that they are **your** information.
   (b) Save (see V.5, below), exit `nano` (see V.10, below), compile (see IV.4, above), run (see IV.5, above). This will test your first set of edits.
   (c) In the *declaration section*, change the constant values that the following named constants are initialized to:
   `minimum_number`
   `maximum_number`
   `close_distance`
   `computers_number`
   You may select any **integer** values that you want, as long as they are different from 1, 5, 10 and 1 respectively, and
   `minimum_number < computers_number < maximum_number`
   and they are sufficiently spread out that you can actually do the runs properly.
   (For example, 2, 4, 2 and 3 won't work, because anything close to 3 would be outside the range of 2 to 4.)
   (d) Save, exit `nano`, compile, run. This will test your second set of edits.

(e) In the *execution section* (also known as the *body* of the program), change the following sequences of character text to your own words:

i. `Hey!`
ii. `That's amazing`
iii. `Close, but no cigar.`
iv. `Bzzzt!  Not even close.`

NOTE: You are welcome to say pretty much anything you want, but please avoid foul or inappropriate language. Please be entertaining; we'll have a **lot** of these to grade.

(f) Save, exit `nano`, compile, run. This will test your final set of edits.

5. Every few minutes while you're editing, you should save the work that you've done so far, in case your work is interrupted by a computer crashing. In `nano`, type **Ctrl**-**O** (the letter oh), at which point `nano` will ask you, near the bottom of the screen:

   `File Name to write : my_number.c`

   That is, `nano` wants to know what filename to save the edited text into, with a default filename of `my_number.c`. Press **Enter** to save to the default filename `my_number.c`.

6. A *character string literal constant*, also known as a *character string literal* or a *string literal* for short, is a sequence of characters between a pair of double quotes. For example, in the `printf` statement

   ```
   printf("This is a printf statement.\n");
   ```

   the following is a string literal:

   ```
   "This is a printf statement.\n"
   ```

   We say that the pair of double quotes *delimits* the sequence of characters in the string literal. Note that the `\n` at the end of the string literal tells the program to print a carriage return (also known as a *newline*) at the end of the line of output text.

7. The lines of text in the C source file `my_number.c` **MUST** be less than 80 characters long, and ideally no more than 72 characters long. (Your PuTTY window **MUST** be 80 characters wide.)

8. Some text editors, including `nano`, try to help keep text lines short, by breaking a long line into multiple short lines. For example, `nano` might break a line like

   ```
   printf("This is a long line and nano will probably break part of it off.\n");
   ```

   into two separate lines:

   ```
   printf("This is a long line and nano will probably
   break part of it off.\n");
   ```

   That is, `nano` automatically puts a carriage return when the line starts getting too long for `nano`'s taste. Unfortunately, the C compiler will consider this to be an error. Why? Because C cannot allow an individual string literal to use more than one line. So, the correct way to write the above example is:

   ```
   printf("This is a long line and nano will probably");
   printf(" break part of it off.\n");
   ```

9. Like the lines of C source text, the lines of output text **MUST** be less than 80 characters long, and ideally no more than 72 characters long. You can break a long line of output text into shorter pieces by making it into two `printf` statements. For example:

```
printf("Why you big old stinker! That's not between %d and %d!\n",
    minimum_number, maximum_number);
```

This single `printf` statement can be converted into two `printf` statements, like so:

```
printf("Why you big old stinker! That's not between\n");
printf("  %d and %d!\n", minimum_number, maximum_number);
```

10. After you've finished editing, exit the text editor. To do this in `nano`, type **Ctrl**–**X**. If you have made any changes since the last time you typed **Ctrl**–**O**, then `nano` will ask you, near the bottom of the screen,

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
```

To save your most recent changes to the file (which is probably what you want to do), press the **Y** key; to avoid saving your most recent changes, press the **N** key. After that, `nano` will behave the same as if you had typed **Ctrl**–**O**.

VI. MAKE (COMPILE), RUN AND DEBUG YOUR OWN UNIQUE VERSION OF THE PROGRAM

1. If necessary, repeat III.1-4, then definitely repeat III.5-6.
2. Make (compile) your own unique version of the executable program:

   ```
   >    make  my_number
   gcc -o my_number my_number.c
   ```

3. If the program doesn't compile, then you'll need to edit it and figure out where things went wrong. **ALWAYS FIX THE FIRST ERROR FIRST** — often, some or all of the remaining error messages are side effects of the first error, and will disappear after you fix the first error.
4. In the worst case, if you're totally stumped, then copy the original from Dr. Neeman's home directory again (see III.8), and start editing the fresh copy.
5. Once you have the program compiled, run `my_number` five times, using the following values as inputs, in this order:
   (a) an integer value less than your value for `minimum_number`
   (b) an integer value greater than your value for `maximum_number`
   (c) an integer value between your value for `minimum_number` and your value for `maximum_number` (inclusive), but far from your value for `computers_number`
   (d) an integer value close to your value for `computers_number` (that is, within your value for `close_distance` of your value for `computers_number`)
   (e) your value for `computers_number`

   It'll look similar to the runs you did with Dr. Neeman's original version of the program.
6. If the program doesn't run, or if it runs incorrectly, then just as in Step VI.3 above you'll need to edit it and figure out where things went wrong. Again, in the worst case, if you're totally stumped, then copy the original from Dr. Neeman's home directory again (see III.8), and start editing the fresh copy.

VII. CREATE A SCRIPT FILE

1. If necessary, repeat III.1-4, then definitely repeat III.5-6.

2. Once the program compiles and runs properly, then you're ready to create a *script file*, which is a record of your interactions with the computer. Start the scripting session:
   > **script pp1.txt**
   `Script started, file is pp1.txt`
   Starting a scripting session is like turning on a tape recorder: every keystroke that you input (including backspaces) and every character that the computer outputs will be recorded into the script file, until you terminate the scripting session (see below).
   NOTICE: `pp1.txt` means "the text file that contains Programming Project #1." Thus, the filename is *small-P small-P one dot small-T small-X small-T*, which is to say *small-pea small-pea one dot small-tee small-ex small-tee.* Notice that the third character in the filename is the digit one, **NOT** lower case L.
   IMPORTANT: DON'T use the name of the executable in the name of the script file.

3. Print the working directory:
   > **pwd**
   `/oushomes/Student/yourusername/CS1313`

4. List the contents of the directory, using the long listing `-l` option:
   > **ls -l**
   ```
   -rwxr-xr-x  1 5013  100   13717 Aug 28 18:27 my_number
   -rw-r--r--  1 5013  100    2976 Aug 28 19:10 my_number.c
   ```
   NOTICE that the command is:
   ell ess space hyphen **ell**
   (lower case L, lower case S, space, hyphen, lower case L).
   It is **NOT NOT NOT**
   ell ess space hyphen **one**
   which would be **WRONG WRONG WRONG!!!**

5. Output your makefile to the terminal screen:
   > **cat makefile**
   This command will cause the contents of `makefile` to be sent to the terminal screen, and also to be saved in `pp1.txt`, the script file.

6. Output your C source file to the terminal screen:
   > **cat my_number.c**
   As above, this command will cause the contents of `my_number.c` to be sent to the terminal screen, and also to be saved in `pp1.txt`, the script file.

7. **IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT!**
   Clean out any old executables:
   > **`make clean`**
   If you don't clean out your old executable, or if the compile command isn't shown as a result of cleaning and then making, or if the compile command fails with error messages, then you haven't proven that your program compiles properly, so **YOU WILL LOSE UP TO HALF THE TOTAL VALUE OF THE PROJECT.**
8. **IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT!**
   Make (compile) your executable program:
   > **`make my_number`**
   If you don't make your executable program, or if the compile command isn't shown as a result of cleaning and then making, or if the compile command fails with error messages, then you haven't proven that your program compiles properly, so **YOU WILL LOSE UP TO HALF THE TOTAL VALUE OF THE PROJECT.**
9. Run `my_number`, using the same number of runs with the same input values in the same order as you used in your test in Step VI.5, above.
10. Terminate the scripting session:
    > **Ctrl**-**D**
    ```
    Script done, file is pp1.txt
    ```
    This is like turning off the tape recorder.
11. You should now have a script file named `pp1.txt` that contains a complete record of the scripting session. Check to be sure that you have the file:
    > **`ls`**
    ```
    makefile    my_number    my_number.c    pp1.txt
    ```
12. Enter the following command at the Unix prompt:
    > **`dos2unix pp1.txt`**
    This command will clear out some of the invisible special characters from `pp1.txt` (but unfortunately not all of them).
13. From this point on, you are **ABSOLUTELY FORBIDDEN** to alter your script file **IN ANY WAY, EVER.** This will be true for all future Programming Projects.

VIII. PRINT YOUR SCRIPT FILE

1. Print your script file. The best way to print from a PC in Carson 205, 206 or S-18 is to use MS Wordpad.
   (a) Run `Wordpad` by traversing the menus like so:
   `Start` → `Windows Accessories` → `Wordpad`
   This will pop up a `Wordpad` window.
   NOTE: AVOID `Notepad`!
   (b) In the WordPad window, click **`File`**.
   (c) Click **`Open`**. This will pop up a window titled `Open`.
   (d) On the left side of the `Open` window, click **`This PC`**.
   (e) In the main panel of the window, below `Network locations`, **double-click** **`yourusername (\\washington.ou.edu) (H:)`**
   (f) In the main panel (just to the right), **double-click** on your **`CS1313`** folder.
   (g) At the bottom of the `Open` window, in the menu just to the right of the text box titled `File name:` and just above the `Open` and `Cancel` buttons, select:
   `All Documents (*.*)`
   (h) Double-click on the file that you want to print (for example, `pp1.txt`). **NOTE:** You may not be able to see the file extension `.txt`, but instead there may be a little icon to the left of `pp1` that looks like a little document.
   (i) In the `Home` tab, to the right, in the `Editing` section, click on **`Select All`**. This will highlight all of the text.
   (j) Toward the left in the `Font` section, set the font to **`Courier New`** (which is a "fixed width" font appropriate for computer programs), and the size to **`10`**.
   (k) At the top of the WordPad window, click `File` and select **`Page Setup`**. This will pop up the **`Page Setup`** window.
   (l) In the section on `Margins`, set all of the margin widths to **`1`**, then click **`OK`**.
   (m) At the top of the WordPad window, click `File` and select **`Print`** and then select **`Print`** again.
   (n) Select as your printer:
   `COE_CEC205-BW on it-robbins`
   (or whichever is most appropriate; you may need to scroll horizontally to find it).
   (o) Click **`Print`**.
   (p) If a window named `Print Job Notification` pops up, select **`Charge to my personal account`**, then click **`Print`**.
2. If you'd prefer to print from another location (for example, from your PC at home), then you'll need to download your script file to the computer that you want to print from, because only the PCs in the IT PC labs in Carson are directly connected to the disk system that contains your CS1313 files. In that case, go to the CS1313 website, scroll down to near the bottom to the section titled "Useful Information," click on the link that says "Downloading a Secure Copy Client to Your Desktop" and follow the instructions.

3. When you're done, close the WordPad window (for example, by clicking the **X** button in the upper right corner of the WordPad Window), and then log out of both the Linux computer and the Windows PC.

IX. WHAT TO SUBMIT: COVER PAGE, SUMMARY ESSAY, SCRIPT, UPLOADING

1. **COMPOSE AND PRINT OUT** the following items, using a word processor, text editor or typewriter (you are **ABSOLUTELY FORBIDDEN** to submit these items written by hand):
   (a) A cover sheet with the following information:
      i. Project number and name
      ii. Course number, name, year and semester
      iii. Author name and e-mail address
      iv. Lab section, day and time

      For example:
      ```
      Project #1:  Thinking of a Number
      Class:  CS 1313 010 Programming for Non-majors, Fall 2017
      Author:  Kim Lee (kimlee@ou.edu)
      Lab:  Section 014 Fridays 2:30pm
      ```
      NOTICE that this information is the same as in the comment block at the beginning of `my_number.c`.
   (b) A summary essay about the project, in your own words.
      For each CS1313 programming project, the summary essay will be worth **AT LEAST 10% OF THE PROJECT'S TOTAL VALUE** (that is, a full letter grade) and **MUST** cover the following points, with a **SECTION HEADER** for each:
      i. The **nature** of the problem to be solved — typically a restatement, in your own words, of the description of the Programming Project that is found in the early paragraphs of the Programming Project specification.
      ii. An abstract description of the **method** of solving the problem that you used — typically a description of how the program behaves, though in the case of PP#1 you may alternatively describe the method as following the steps in the PP#1 specification.
      iii. A list of the concrete **steps** by which you implemented your method (in the case of PP#1, you may list the major sections of the project)
      iv. The **issues and problems** that you had to address during implementation
      v. The **concepts** that you learned from this project
      vi. **References** as appropriate: **EVERY** summary essay **MUST** end with a clearly labeled references section, which may be marked "none" if there are no references.

      For PP#1, the summary **MUST** be at least half a page single spaced, or a full page double spaced, with a font of 10 to 12 points, and with margins of 3/4 inch to 1 inch on all sides. For future programming projects, the summary may need to be longer.

   NOTE: The cover **MUST** be on a **SEPARATE** page from the summary essay.
2. **Bind** the paper copies of the materials in the following order:
   (a) the cover sheet (on top);
   (b) the summary essay (in the middle);
   (c) the pages of the script file in order (on the bottom);
   (d) the bottom half of the extra credit form (at the very bottom), if applicable (see below).

   Bind them in the above order with at least an adequate staple or heavy duty binder clip. No paper clips, duct tape, brightly colored yarn, etc.

3. You **MUST** also upload **BOTH** your **C source file** **`my_number.c`** and your **script file** **`pp1.txt`** electronically to the PP#1 dropbox on OU's Canvas website. **The exact same lateness rules apply to Canvas submissions as to paper submissions** (see the CS1313 syllabus for lateness policy details). This uploading will be worth at least 5% of your grade (that is, half a letter grade) on this and each future programming project.

   (a) Using the Microsoft Edge web browser, go to:
   `http://canvas.ou.edu/`
   Note that other web browsers such as Firefox sometimes have problems with Canvas.

   (b) Log in; your username will be your OU4+4, and your password will be the password for your OU4+4.

   (c) Find and click on the link for
   `C S-1313-010 Fall 2017`
   You may need to click on the
   `Courses`
   button on the left side of the window, scroll down to the bottom of that menu, and click
   `All Courses`
   and then click on
   `C S-1313-010 Fall 2017`
   (or it might be listed as **`CS1313`** or similar).

   (d) Near the left side of the page, in the vertical stack of words, click on
   `Assignments`

   (e) You should see a list titled **`Upcoming Assignments`**; click on the assignment associated with the current Programming Project — in this case:
   `PP#1`

   (f) You'll now be on the PP#1 page. On the right hand side of the window near the top will be a button labeled
   `Submit Assignment`
   Click on it.

   (g) This will cause to appear, a bit farther down the webpage, some tabs, one of which is:
   `File Upload`
   If it isn't already selected, click on it.

   (h) In the **`File Upload`** tab, click on:
   `Browse`

   (i) This will pop up an internal window titled
   `Choose File to Upload`

(j) Using the instructions on page 16, items VIII.1.d-h (except the title of the popup window will be different), and then scrolling through the list of files, find your C source file — in this case,
my_number.c
which should be on the same
yourusername (\\washington.ou.edu) (H:)
that you printed from, which is also available in the leftmost column of the window. Double click on the filename.
NOTE: The file may appear to be named **my_number**, so carefully inspect the little icon to the left of the filename; if it's a big letter C, then that's actually **my_number.c**.
NOTE: AVOID my_number.c~ (with a tilde after the .c), which is the next-to-last version of your source file, **NOT** the final, compiled version that you want to upload.

(k) Once you've chosen the file, it'll appear in the text box between the label
`File:`
and the button labeled
Browse

(l) Below that, click on:
Add Another File

(m) Repeat the procedure above to find and add your script file:
pp1.txt

(n) You **DON'T** need to write a comment in the comment box, but you're welcome to.

(o) Click the button labeled
Submit Assignment

(p) If anything goes wrong, you can use the button labeled
Re-submit Assignment
in the upper right of the webpage.

(q) Inspect the files that you've uploaded, to make sure that they're the correct files.

(r) On the far left of the webpage, near the top, click on the button labeled
Account

(s) Log out of Canvas by clicking the button labeled
Logout

You will need to upload **BOTH** your C source file **AND** your script file for **EVERY** programming project in CS1313.

EXTRA CREDIT

You can receive an extra credit bonus of 5% of the total value of Programming Project #1 by doing the following:

1. Attend at least one CS1313 help session for at least 30 minutes, through Wed Sep 6.
2. During the help session that you attend, work on CS1313 assignments (ideally PP#1, but any CS1313 assignment is acceptable). **YOU CANNOT GET EXTRA CREDIT IF YOU DON'T WORK ON CS1313 ASSIGNMENTS DURING THE HELP SESSION.**
3. Before you leave the help session, fill out **BOTH** halves of the form on the last page of this project specification and have the help session leader (instructor or TA) sign **BOTH** halves. **THE FORM CANNOT BE SIGNED UNTIL IT IS COMPLETELY FILLED OUT. DON'T** fill it out in pencil or in water-soluble ink.
4. Attach the bottom half of the form to your PP#1 script printout, **AFTER** the script itself, and keep the top half for your own records.

NOTE: This extra credit bonus **WON'T** be available on any other programming project unless explicitly stated so in each project's specification.

COMMON PROBLEMS

- PuTTY: Remember to select SSH.
- Case sensitivity: Unix and C are **case sensitive.**
- CS1313 subdirectory: **ALL** of your CS1313 work should be in your CS1313 subdirectory.
- The dot: Remember the dot at the end of some of the cp commands.
- ls: The command is *ell ess*, **NOT** *one ess*.
- ls -l: The command is *ell ess space hyphen ell*, **NOT** *ell ess space hyphen one*.
- Using Windows: **UNDER NO CIRCUMSTANCES SHOULD YOU EDIT FILES ON A WINDOWS COMPUTER IF THEY ARE TO BE USED ON A UNIX COMPUTER.**
- Constant values: In the declaration section of my_number.c, when you choose constant values, be sure that the values that you choose allow all of the runs. For example, don't have the minimum and maximum too close to each other.
- What to change: In the execution section of my_number.c, be sure that the **ONLY** things you change are the ones specified. **DON'T change anything else!**
- Editing: Save your work frequently.
- Line lengths in source code: In the execution section of my_number.c, be sure that each line is less than 80 characters long.
- String literals: In the execution section of my_number.c, be sure that each string literal is contained entirely on a single line.
- nano: When editing, if you see a line that ends in a dollar sign, probably that means that the line is too long. Also, be careful of nano putting in extra carriage returns.
- Running the program: Be sure that all outputs are less than 80 characters long.
- Script: Remember to
  `make clean`
  Failure to do so will cost you half the value of the project, right off the top.
- Script: Remember to
  `make my_number`
  Failure to do so will cost you half the value of the project, right off the top.
- make: The command is
  `make my_number`
  NOT
  `make my_number.c`
- Script: Be sure to do the correct number of runs, and in the correct order.
- Printing: Be sure to use the Courier New font in 10 point size.
- Printing: Be sure to set the margins to 1 inch all around.
- **PROOFREAD PROOFREAD PROOFREAD** your script printout.
- Cover page: Be sure to list all required information.
- Summary essay: Be sure that it's long enough.
- Summary essay: Be sure that you have a references section, **even if you have no references.**
- Binding order: Be sure that the parts of the project are bound in the correct order.
- Upload: Be sure to upload the **SOURCE** file **AND** the **SCRIPT** file **BUT NOT** the executable file.

NOTES

READ THIS PROJECT SPECIFICATION SEVERAL TIMES, CAREFULLY. It is **YOUR** responsibility to read and comply with **EVERY WORD**. Failure to follow directions **IN EVERY DETAIL** will cost you a significant amount of points on this and all assignments. The fact that you didn't notice something **WON'T** excuse you from complying with it.

You will use the same basic approach for every programming project in this course. Since your programming projects are 45% of your grade, each one may be worth half a letter grade or more. You'll want to do them all, and to do them well.

For **EVERY** programming project, you are expected to keep a copy of your source code and your script file on your IT Linux account **THROUGH THE END OF THE SEMESTER** until your overall letter grade for the course has been officially posted. **NEVER DELETE EITHER FILE!** If something goes wrong with your printout, these files will be your only proof that you've done the work. In addition, you may be assigned mini-projects that require you to modify a completed project; if you've deleted that project, then you may have to do the whole thing from scratch in a very limited amount of time.

We strongly recommend that you **DON'T** attempt extra, unrequested tasks on any assignment. While doing extra work is admirable in principle, in practice it creates a significant chance that you will be unable to complete the assignment on deadline. Unrequested extra work **WON'T** gain you extra credit. In some cases we may assign bonus work, which will be worth extra credit and which we encourage you to try; otherwise, it may be foolhardy to complicate a given assignment unnecessarily.

"The perfect is the enemy of the good." If you have to choose between submitting an imperfect project on time or submitting a perfect project late, **CHOOSE CAREFULLY.** Remember that you lose the equivalent of **TWO LETTER GRADES FOR EACH LECTURE PERIOD** that your submission is late. If your program compiles and runs at all, even with errors, it will probably be wiser to submit it on time, rather than to continue to refine it and then submit it late, thereby accruing a lateness penalty.

To be a good programmer, you need the following:

- **Patience**
  Designing, writing and debugging programs takes a **lot** of time.
- **Persistence**
  Often, you will find yourself stuck without knowing how to proceed; **DON'T** give up.
- **Pessimism**
  - Just because you have a design, that doesn't mean it'll be easy to write the program.
  - Just because you've written the program, that doesn't mean it'll compile.
  - Just because it compiles, that doesn't mean it'll run.
  - Just because it runs, that doesn't mean it'll produce the correct answer.
  - Just because it produces the correct answer, that doesn't mean that the printer works.
- **Practice**
  Just like writing prose, or welding, programming is learned by doing, not by theorizing.

THIS PAGE INTENTIONALLY LEFT BLANK.

CS1313 PROGRAMMING PROJECT #1 BONUS REQUEST FORM

Name _____ Lab _____
Help Session Date _____
Help Session Time (Arrive) _____ Help Session Time (Depart) _____

Instructor Signature _____

Keep this copy for your records.

CS1313 PROGRAMMING PROJECT #1 BONUS REQUEST FORM

Name _____ Lab _____
Help Session Date _____
Help Session Time (Arrive) _____ Help Session Time (Depart) _____

Instructor Signature _____

Submit this copy.
In your submission, attach this copy **AFTER** your script file printout.
If you put this in the wrong place in your submission, then you **WON'T** get the extra credit.