This first assignment will help you learn to use the Linux computers administered by OU Information Technology for the Gallogly 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 Wed Aug 28. 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 a 4 digit number, which may be the last 4 digits of your OU ID number, depending on when you started at OU).

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, Option #1:
   **NOTE:** On some PCs, Option #1 doesn’t work well (sometimes because of screen resolution issues), in which case you should use Option #2, below.
   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. In the far bottom left corner of the screen, click on the start menu (Microsoft Windows icon).
   x. In the popup menu that comes up, scroll the menu down to near the bottom, and click on
      **Windows PowerShell** 
      (Its icon will be a little folder.)
   xi. This will expand the options under
      **Windows PowerShell** 
      Under that folder icon, indented slightly, click on either
      **Windows PowerShell** 
      OR
      **Windows PowerShell (x86)**
   xii. The **Windows PowerShell** window will appear. Hover the mouse cursor over the bar at the top of that window, and right-click over that top bar.
   xiii. In the **Windows PowerShell** menu that comes up, click on
      **Properties**
   xiv. In the "**Windows PowerShell" Properties** window that comes up, click on the tab named **Font**
   xv. In the **Windows PowerShell Properties** window, in the **Font** tab, partway down the window in the section titled **Font**, set the font to **Courier New**
xvi. In the "Windows PowerShell" Properties window, click on the tab named Layout.

xvii. In the Windows PowerShell Properties window, in the Layout tab, partway down the window in the section titled Window Size, set Width to 80 and set Height to 40.

**NEVER RESIZE THE POWERSHELL WINDOW USING THE MOUSE, AND NEVER CLICK THE MAXIMIZE BUTTON AT THE TOP RIGHT OF THE POWERSHELL WINDOW.**

xviii. In the "Windows PowerShell" Properties window, click on the tab named Colors.

xix. In the Windows PowerShell Properties window, in the Colors tab, near the top, click on Screen Text and then, a bit below that, click on either the white square or the yellow square (the rightmost colors in the bar of colors).

xx. In the Windows PowerShell Properties window, in the Colors tab, near the top, click on Screen Background and then, a bit below that, click on either the black square or the dark blue square (the leftmost colors in the bar of colors).

xxi. At the bottom of the "Windows PowerShell" Properties window, click OK.

xxii. Click on the interior of the Windows PowerShell window (the blue part).

xxiii. The Unix prompt in the Windows PowerShell window, which at the moment should be the last line of text inside (the blue part of) that window, will probably be something like:

```
PS C:\Users\YourPCUserName>
```

(except that YourPCUserName will be replaced by your username on the PC that you’re sitting at — if it’s a PC in a Carson PC lab, that’ll probably be your OU4+4, but on a non-Carson PC, it may be something else).

**NOTE:** The Unix prompt is a sequence of characters that tells you that the operating system wants you to enter a Unix command (for example, the Unix command in the very next step, just below).

xxiv. At the Unix prompt, type

```
ssh yourusername@ssh.ou.edu
```

(but replacing yourusername with your OU4+4), and then press the Enter key on the keyboard.

xxv. If you’re asked a yes/no question, enter yes (in all lower case).

(b) **From a PC in Carson Engineering Center 205, 206 or S-18, Option #2:**

i. Do steps I.1.a.i-viii, just above (but AVOID steps I.1.a.ix-xxiv).

ii. Run a program named PuTTY, which lets you log in remotely to the computer that we’re going to use.
   
   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:
      
      http://cs1313.ou.edu/
      
   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).
   
   F. Double-click on the PuTTY icon on your desktop (or wherever you’ve saved it).

iii. 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.

iv. 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

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

vi. On the left side of the PuTTY Configuration window, below
      
      Terminal
      
      and its subentries, is an entry
      
      Window
      
      Click on it.
vii. This will bring up a section titled Options controlling PuTTY’s window
The first option is Set the size of the window
Choose \textbf{EXACTLY} 80 columns wide (which is the default) and \textbf{EXACTLY} 40 rows high (which isn’t the default, so you’ll have to change it).

viii. Below that is an option When window is resized:
Click on \textbf{Forbid resizing completely}
\textbf{NEVER} resize the PuTTY window using the mouse, and \textbf{NEVER} click the maximize button at the top right of the window.

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

x. 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 \textbf{UTF-8}

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

xii. 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: \texttt{ssh.ou.edu}

xiii. Immediately below this is Connection type: below which make sure that \textbf{SSH}
is selected (it should be the default, so you shouldn’t need to click on it).

xiv. At the bottom right of the PuTTY Configuration window, click \textbf{Open}
xv. If a PuTTY Security Alert window pops up, click Yes.

xvi. When prompted to login as, type your OU4+4 username with ALL LETTERS IN LOWER CASE (small), and then press Enter.

xvii. When prompted for your password, type your password (which should be your OU4+4 password, which DOESN’T have to be all lower case) and press Enter. NOTHING WILL APPEAR AS YOU TYPE YOUR PASSWORD. This is normal for Unix.

xviii. NOTE: When you log out of a Windows PC in Carson 205, 206 or S-18, your PuTTY settings may be lost, in which case you’d have to redo all these settings each time.


(c) From your own Windows PC, or from a Windows PC not in Carson 205, 206 or S-18: You will need to do either Option #1 (steps I.1.a.i-xxvi) OR Option #2 (steps I.1.b.i-xix), above.

(d) From your own Apple MacOS or a Unix/Linux computer, or a Mac or Unix/Linux computer not in Carson 205, 206 or S-18: Go to the course website, scroll down nearly to the bottom, to the section titled “USEFUL INFORMATION,” and click on the following link: **Downloading a Secure Shell Client to Your Desktop**

Go to the section on MacOS or Unix/Linux, and follow the instructions. 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 EXACTLY 80 columns wide by EXACTLY 40 rows high, and to set the font to a fixed width font such as Courier New.

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 — perhaps a greater-than sign, a percent sign or a dollar sign, maybe preceded by other text — with the text cursor after it, like so:

    >

(The text cursor will probably be green.)

If there is information before the prompt character, it might be the name of the computer that you’ve logged in to (which may be different from ssh.ou.edu), and/or your user name, and/or other information. 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
```

**NOTES:**

- All Unix commands **MUST** be followed by pressing the `Enter` key.
- **DON’T** type the greater than symbol `>`, which indicates the Unix prompt, and thus **ISN’T** part of the `pwd` command.

The `pwd` 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 Unix commands **MUST** be followed by pressing [Enter]):

   ```
   > cp ~neem1883/DOT_student/.[a-z]* ~
   ```

   This command means: “Copy, from a subdirectory of Dr. Neeman’s home directory, specifically the subdirectory named DOT_student, all files whose filenames start with a dot (period .) followed by a lower case letter followed by any number of any characters, into my home directory.” You **WON’T** have to do this for future logins.

**NOTICE:**
- The Unix copy command is `cp`.
- The first filename or directory name after `cp` is the **source** (the thing that you’re making a copy of); the second filename or directory name 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.
- 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/DOT_student/.[a-z]* ~
   ```

2. 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.

3. 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.

4. 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.”
5. Set the permissions on your CS1313 directory so that only you can access it:
   \[ \texttt{chmod u=rwx,go= CS1313} \]
   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.

6. 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:
   > 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 subdirectory:
   > 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
    ```
    (There may be other files listed as well.)

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 C 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 terminal 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 C 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; that is, `-o` means “the output of the compiler,” and the output of a compiler is an 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 C 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 text 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.”

   **NOTE:** If you’d prefer to use another text editor (for example, vim, emacs), you may do so, but your TA won’t have time to help you learn it. **UNDER NO CIRCUMSTANCES SHOULD YOU EDIT A FILE ON A WINDOWS COMPUTER IF IT IS TO BE USED ON A UNIX/LINUX COMPUTER,** because 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` (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 terminal window **MUST** be 80 columns 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:

```c
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:

```c
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. Once you’ve fixed an error, return to step VI.2, above.

   **NOTE:** EVERY SINGLE TIME you edit your C source file `my_number.c`, you should then **IMMEDIATELY** return to step VI.2, above.

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, and then return to step VI.2. 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 ppl.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: ppl.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 21 18:27 my_number
   -rw-r--r-- 1 5013 100 2976 Aug 21 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 ppl.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 ppl.txt, the script file.

7. IMPORTANT! 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 right off the top, before any deductions for mistakes are assessed by the grader.
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** right off the top, before any deductions for mistakes are assessed by the grader.

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.
    You’re welcome to create a new script file up until you submit the final version to Gradescope (see below), using the exact same method as in this section. But, you’re **ABSOLUTELY FORBIDDEN** to edit (with a text editor such as `nano`) a script file that you’ve already created.

14. If you want to create a new script file, then:
    
    (a) If you’re currently in a scripting session, end the scripting session (see VII.10, above).
    (b) Repeat steps VII.2-10, above.
    
    That is, the new scripting session will start by deleting the old script file (of the same filename).
VIII. PRINT YOUR SCRIPT FILE

Print your script file. We recommend that you print from a PC in Carson 205, 206 or S-18, like so:

1. Use the same instructions as in I.1.b.ii.A-F on page 4 to download psftp to the Windows PC that you’re logged in to.
2. Run psftp by double-clicking on its icon.
3. At the psftp prompt, enter:
   \texttt{open ssh.ou.edu}
4. At the login as prompt, enter your OU4+4 username.
5. At the password prompt, enter your OU4+4 password. \textbf{NOTHING WILL APPEAR AS YOU TYPE YOUR PASSWORD.} This is normal for Unix.
6. At the psftp prompt, enter:
   \texttt{pwd}
7. At the psftp prompt, enter:
   \texttt{ls}
8. At the psftp prompt, enter:
   \texttt{cd CS1313}
9. At the psftp prompt, enter:
   \texttt{pwd}
10. At the psftp prompt, enter:
    \texttt{ls}
11. At the psftp prompt, enter:
    \texttt{lcd C:\Users\yourusername\Desktop}
    \textbf{NOTE:} The command is LOWER CASE LCD (ell cee dee, that is, small-L small-C small-D); the first character \textbf{ISN’T} a one. The command name is short for “local change directory,” where local means “on the PC you’re actually sitting at” (instead of the remote PC that you’re logged in to).
    \textbf{ALERT:} The \texttt{lcd} command works in psftp (and other Secure File Transfer Protocol logins) ONLY, not in regular Unix SSH logins (for example, \texttt{lcd} won’t work in a terminal window such as PuTTY).
    \textbf{ALERT:} Your OU4+4 username will only work on the Windows PCs in Carson 205, 206 and S-18 (and perhaps on other OU-owned PCs on campus), but \textbf{NOT} on PCs elsewhere (especially, \textbf{NOT} on your personally owned PC). For example, on your personally owned Windows 10 PC, to find out your username, do as described on page 22.
12. At the psftp prompt, enter:
    \texttt{get pp1.txt}
    \textbf{NOTE:} The filename is LOWER CASE PP1.TXT (pea pea one dot tee ex tee, that is, small-P small-P one period small-T small-X small-T); the third character \textbf{IS} a one.
    \textbf{ALERT:} The \texttt{get} command, and its counterpart, \texttt{put}, work in psftp (and other Secure File Transfer Protocol logins) ONLY, not in regular Unix SSH logins.
13. At the psftp prompt, enter:
    \texttt{exit}
    (Or, click on the little X in the upper right of the PSFTP window.)
14. On your Windows desktop, minimize all of the windows on your desktop, and find the icon for *pp1.txt* (which might just say *pp1*, with an icon showing several lines, representing lines of text).

15. Right click on the icon for *pp1.txt*, and in the popup menu, hover the mouse cursor over: **Open with**

16. That’ll pop up another menu, in which select: **WordPad**

17. In the WordPad window, in the **Home** tab, to the right, in the **Editing** section, click on **Select All**

This will highlight all of the text.

18. Toward the left in the **Font** section, set the font to **Courier New** (which is a “fixed width” font appropriate for computer programs), and set the size to **10**.

19. In the WordPad window, in the upper left, click on: **File**

20. In the popup menu, select: **Page Setup**

21. This will pop up the **Page Setup** window. In the section on **Margins**, set all of the margin widths to **1**.

22. In the **Page Setup** window, click: **OK**

23. In the WordPad window, in the upper left, click on: **File**

24. In the popup menu, click on: **Print**

25. In the **Print** popup window, in the (first) section titled **Select Printer**, click on: **Microsoft Print to PDF**

26. At the bottom of the **Print** window, click on: **Print**

27. This will pop up a window titled **Save Print Output As**. On the left side, click on: **Desktop**

28. In the textbox labeled **Filename**, set the filename to **pp1**

   (The extension **.pdf** will be added automatically.)

29. At the bottom right of the **Save Print Output As** window, click **Save**

30. Open up your preferred e-mail client for using your official OU e-mail address (for example, Outlook365).

31. Create a new e-mail, with the file you want to print (in this case, **pp1.pdf**) as an attachment. **NOTE:** Look **CAREFULLY** at the icon for the file you want to attach, to make sure that it’s a PDF file.

32. Send that e-mail to: **print@wepanow.com**
33. Shortly after, you’ll get a reply e-mail from print@wepanow.com with a Release Code.
34. Go to any Wepa printer, for example the one in the far front corner of Carson 206 (near the door).
35. On the Wepa print display, below **Insert your Sooner Card**, tap on:
   - **I have a Wepa Code**
36. Type the Wepa Release Code that came in the confirmation e-mail.
37. Tap on:
   - **Submit**
38. If necessary, tap the relevant document so that it has a checkmark by it.
39. Tap the big icon that says:
   - **Print**
40. Select the payment method and follow the instructions to pay for your printing.
41. Collect your printout.
42. If necessary, tap on:
   - **Tap to log out**

When you’re done printing, 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 the Linux computer, and, if you’re logged in to a Windows PC in Carson 205, 206 or S-18, log out of that Windows PC.

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.

For a Windows PC, using the information above, do steps VIII.1-24, then print via your preferred printer in the way you usually print from that printer.

**NOTE:** As described in item VIII.11, above, your OU4+4 won’t work on PCs not in Carson 205, 206 or S-18 (and perhaps other OU-owned PCs on campus). For example, if your personally owned PC has Windows 10, then you can find your username as follows:

1. Open any folder (or click on any open folder).
2. At the top of the folder window, click **View**
3. At the far left of the **View** pane that appears, click on **Navigation Pane**
4. In the popup menu that appears, make sure that **Navigation Pane** is checked.
5. In the navigation pane, click on **Windows (C:)**
6. Find the folder named **Users**
7. In the resulting list of folders, you should be able to identify your username on that PC.

For a MacOS or Linux PC, 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.
IX. WHAT TO SUBMIT: COVER PAGE, SUMMARY ESSAY, SCRIPT, UP LOADING

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 2019
      
      **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 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/full page double spaced, with a font of 10 to 12 points and margins of 3/4 inch to 1 inch on all sides.
      
      **NOTE:** The cover MUST be on a SEPARATE sheet of paper 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 (near the top);
   
   (c) the pages of the script file in order (in the middle);
   
   (d) the pages of the checklist, below, in order (near the bottom);
   
   (e) optionally, the bottom half of the extra credit form, below, if any (at the bottom),

   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 the Gradescope website. **The exact same lateness rules apply to Gradescope 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) Log in to Canvas.

(b) In Canvas, go into the CS1313 lecture course, section 010.

(c) On the CS-1313-010 page, almost all the way to the left, are a stack of words such as Home and Announcements. One of these words should be **Gradescope**

   Click on it.

   **NOTE:** If you **don’t** have the word Gradescope, please **IMMEDIATELY** send an e-mail to all five of us (Dr. Neeman and all TAs), alerting us that you don’t have the Gradescope link.

(d) Clicking on the **Gradescope** link will pop up a new tab or a new window for Gradescope.

(e) It’s possible that you will be asked to log in to Gradescope, in which case, log in with your official OU e-mail address, the long form (typically `yourfirstname.yourmiddleinital.yourlastname-yournumber@ou.edu` using your OU4+4 password.

(f) Once you’re logged in to Gradescope, you will probably see a panel of buttons for one or more of your courses that are using Gradescope. Click on the panel for **CS-1313-010**

(g) You will then be taken to the “dashboard” for CS1313. Click on **PP1**

(h) This will pop up a little window titled **Submit Programming Assignment**. Notice the large area in this popup window that is labeled **DRAG & DROP**.

(i) **DRAG & DROP**, there are three radio buttons. Click on **Upload**

   (if it isn’t already selected).

(j) If your web browser is in full screen mode, then:

   - In Windows: In the upper right corner of your browser window, there’s the dash for Minimize, the box for Maximize (full screen), and the big X for Close. Get out of full screen mode by clicking on that full screen box.

   - In MacOS: In the upper left corner of the browser window, there are three colored circles: red (leftmost), yellow (middle) and green (rightmost). Get out of full screen mode by clicking on the green (rightmost) circle.

(k) If necessary, drag and/or resize your browser window so that you can see the icons for `my_number.c` and `pp1.txt`. You need to be able to see both of those icons, as well as the Drag-and-Drop zone of the Gradescope popup window.
(l) Drag and drop your script file `pp1.txt` into the Gradescope Drag-and-Drop zone.  
(m) Upon dragging and dropping `pp1.txt` into the Gradescope Drag-and-Drop zone, the Submit Programming Assignment popup window will change, to show that `pp1.txt` is ready to be uploaded.  
(n) Drag and drop your C source file `my_number.c` into the Gradescope Submit Programming Assignment popup window, right over the name of `pp1.txt`, even though the popup window no longer has a Drag-and-Drop zone.  
(o) Upon dragging and dropping `my_number.c` into the Gradescope Submit Programming Assignment popup window, the Submit Programming Assignment popup window will change, to show that `my_number.c` is ready to be uploaded, as is `pp1.txt`.  
(p) In the lower left corner of the Submit Programming Assignment popup window, click [Upload]  
(q) This will pop up a new popup window that says PPl submitted successfully! (along with some additional text). This popup window will disappear after several seconds.  
(r) After the second popup window disappears, your web browser will be on a Gradescope page titled Autograder Results  
Just below that title, you may see a message such as:  
The autograder failed to execute correctly. Contact your course staff for help in debugging this issue. Make sure to include a link to this page so that they can help you most effectively.  
If you see that message, IGNORE IT! That message is intended for assignments that use automatic grading software, which PP#1 doesn’t use.  
(s) On the Gradescope Autograder Results webpage, near the top right, there are two buttons: [Results] and [Code]. Initially, [Results] will be highlighted. Click on [Code]  
(t) On the [Code] side, you should be able to see the files that you have uploaded. If you can only see a filename but not its contents, you can expand it to see its contents by clicking on the filename. (Likewise, you can compact it by clicking on the filename again.)  
CAREFULLY INSPECT BOTH OF YOUR FILES, to ensure that you’ve uploaded the correct files successfully, and that they contain the correct content (as described above). UPLOADING THE WRONG FILES WILL COST YOU MANY MANY POINTS!  
(u) You can exit Gradescope by closing the Gradescope tab.  

You will need to upload BOTH your C source file AND your script file for EVERY programming project in CS1313, unless otherwise announced.
4. **Submit** the paper submission (see IX.2, above) in lecture by the deadline (for PP#1, Wed Sep 4 2019 by 10:20am Central Time). If necessary, you can submit **BEFORE THE DEADLINE**. If submitting before the deadline, you’re allowed to submit in lecture of course, but also in a lab session or a help session, though these aren’t preferred.

If you choose not to submit a paper submission, then you will only be eligible for the 5% credit for uploading to Gradescope (that is, you’ll lose 95% of the credit).
EXTRA CREDIT

(A) HELP SESSION BONUS EXTRA CREDIT

You can receive an extra credit bonus of 2.5% 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 4.
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. If you leave the help session without getting the form signed, you **CANNOT** get extra credit for attending that help session; your form **CANNOT** be signed later.
5. Attach the bottom half of the form to your PP#1 script printout, at the **VERY END**, and keep the top half for your own records.

(B) CHECKLIST BONUS EXTRA CREDIT

You can receive an extra credit bonus of up to 2.5% of the total value of Programming Project #1 by doing the following:

1. Print out the checklist on pages 30-33 of this PP#1 specification. You **MUST** provide the **ENTIRE** checklist, all 4 pages (preferably double sided on 2 sheets of paper).
2. Complete the checklist, by checking the checkbox for every item that you have performed as described.
3. Include the checklist in your paper submission, as described in this PP#1 specification, pages 24-25, item IX.3.

NOTES:

(i) These extra credit bonus items **WON’T** be available on any other programming project unless explicitly stated so in each project’s specification.

(ii) For the checklist bonus extra credit, if you mark any of the items **INCORRECTLY** (that is, you said that you did do something that you actually didn’t do), then we reserve the right to decide to reduce or eliminate some or all of the bonus extra credit for PP#1, at our sole discretion.
COMMON PROBLEMS

- **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 A FILE ON A WINDOWS COMPUTER IF IT IS TO BE USED ON A UNIX/LINUX 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 C 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 C 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.
PP#1 CHECKLIST (Print this checklist, check all that apply, and include this in your paper submission, for up to 2.5% BONUS EXTRA CREDIT.)

NAME ........................................................................................................... LAB SECTION ..............

☐ SSH window size: In the window that I use to access ssh.ou.edu (for example, PowerShell or PUTTY in Microsoft Windows or the MacOS terminal window in MacOS), I always verify that my window size is EXACTLY 80 columns wide by EXACTLY 40 rows high (as described in the PP#1 specification, I.1.a.xvii/I.1.b.vii/I.1.d).

☐ SSH window resizing FORBIDDEN (Microsoft Windows only): When I’m logging in to ssh.ou.edu from a Microsoft Windows PC via PuTTY, in the PuTTY window that I’m using, I always verify that I’ve set that window to forbid resizing (as described in the PP#1 specification, page 3, item I.1.b.xvii).

☐ Copying DOT_STUDENT files: I have successfully copied all of Dr. Neeman’s .a-z]* files from his DOT_STUDENT subdirectory to my home directory (as described in the PP#1 specification, page 6, item II.1).

☐ CS1313 subdirectory creation: I have successfully created my CS1313 subdirectory, using the mkdir command (as described in the PP#1 specification, page 6, item II.3).

☐ CS1313 subdirectory permissions: I have successfully set the permissions on my CS1313 subdirectory to be limited to me only, using the chmod command (as described in the PP#1 specification, page 9, item II.5).

☐ CS1313 subdirectory use: ALL of my PP#1 work is in my CS1313 subdirectory, and this will be true for ALL of my future CS1313 work (as described in the PP#1 specification, page 10, item III.4).

☐ Copy Dr. Neeman’s my_number.c: I successfully copied Dr. Neeman’s original C source file my_number.c to my CS1313 subdirectory (as described in the PP#1 specification, page 10, item III.8).

☐ Verify that my_number.c is in CS1313 subdirectory: I verified that I successfully copied Dr. Neeman’s original C source file my_number.c to my CS1313 subdirectory, by using the ls command (as described in the PP#1 specification, page 10, item III.9).

☐ Copy Dr. Neeman’s makefile: I successfully copied Dr. Neeman’s makefile to my CS1313 subdirectory (as described in the PP#1 specification, page 11, item III.10).

☐ Verify that makefile is in CS1313 subdirectory: I verified that I successfully copied Dr. Neeman’s makefile to my CS1313 subdirectory, by using the ls command (as described in the PP#1 specification, page 11, item III.11).

☐ Look at my_number.c: I examined Dr. Neeman’s original C source file, named my_number.c, using the cat command (as described in the PP#1 specification, page 12, item IV.2).

☐ Look at makefile: I examined my makefile using the cat command (as described in the PP#1 specification, page 12, item IV.3).

☐ Compile (make) Dr. Neeman’s my_number: I successfully compiled Dr. Neeman’s original C source file, named my_number.c, using the make my_number command (as described in the PP#1 specification, page 12, item IV.4).

☐ Run (execute) Dr. Neeman’s my_number executable: I successfully ran (executed) the executable made from Dr. Neeman’s original C source file, named my_number, using the my_number command, for all 5 runs, with the appropriate input values in the appropriate order (as described in the PP#1 specification, page 13, item IV.5).
CHECKLIST ITEMS FOR CREATING AND TESTING YOUR OWN my_number.c

☐ Edit in Unix, NOT Windows: When editing the C source file to create my own version, I edited my C source file directly on ssh.ou.edu using a Unix text editor such as nano, NOT in Microsoft Windows using a Microsoft Windows editor (NOR in MacOS using a MacOS editor) (as described in the PP#1 specification, page 14, item V.2).

☐ Comment block: In my version of the C source file my_number.c, in the comment block near the top of the C source file, before the main function, I changed the information to be my information (as described in the PP#1 specification, page 14, item V.4.a).

☐ Constant values: In my version of the C source file my_number.c, in the declaration section, I chose new constant values for the named constants described in the PP#1 specification, and the values that I chose for these named constants are far enough from each other that all required test cases are valid and meaningful (as described in the PP#1 specification, page 14, item V.4.c).

☐ String literals: In my version of the C source file my_number.c, in the comment block, I changed the text of the correct string literals, and I didn’t change the text of any other string literals not required in the PP#1 specification (as described in the PP#1 specification, page 15, item V.4.e.i-iv).

☐ Saving regularly while editing: When editing my version of the C source file my_number.c (or any other file), I saved my work regularly and repeatedly (as described in the PP#1 specification, page 15, item V.5).

☐ Line lengths in C source code: In my version of the C source file my_number.c, in the execution section, I verified that every line of C source code text is less than 80 characters long, the width of my terminal window (as described in the PP#1 specification, page 15, item V.7).

☐ Extra carriage returns/line breaks: In my version of the C source file my_number.c, throughout the entire file, I verified that there are no extra carriage returns/line breaks (as described in the PP#1 specification, page 15, item V.8).

☐ Line lengths in output of runs: In my runs of my version of the C source file my_number.c, I verified that every line of output text is less than 80 characters long, the width of my terminal window (as described in the PP#1 specification, page 16, item V.9).

☐ Compile (make) your my_number: I successfully compiled my version of the C source file named my_number.c, using the make my_number command (as described in the PP#1 specification, page 17, item VI.2).

☐ Run (execute) your my_number executable: I successfully ran (executed) the executable made from my version of the C source file, using the my_number command, for all 5 runs, with the appropriate input values in the appropriate order (as described in the PP#1 specification, page 17, item VI.5).
CHECKLIST ITEMS FOR SCRIPTING

☐ Start script session: I successfully started my scripting session, using the correct script command, with the correct filename, which for PP#1 is `pp1.txt` (small-P small-P one period small-T small-X small-T), NOT `ppl.txt` (small-P small-P small-L period small-T small-X small-T, which would be INCORRECT).

    script pp1.txt

    (as described in the PP#1 specification, page 18, item VII.2).

☐ Script session `pwd`: In my scripting session, I properly did the `pwd` command (as described in the PP#1 specification, page 18, item VII.3).

☐ Script session `ls -l`: In my scripting session, I properly did the `ls -l` command (small-L small-S space hyphen small-L, NOT small-L small-S space hyphen one, which would be INCORRECT) (as described in the PP#1 specification, page 18, item VII.4).

☐ Script session `cat makefile`: In my scripting session, I properly did the `cat makefile` command (as described in the PP#1 specification, page 18, item VII.5).

☐ Script session `cat my_number.c`: In my scripting session, I properly did the `cat my_number.c` command (as described in the PP#1 specification, page 18, item VII.6).

☐ Script session `make clean`: In my scripting session, I properly did the `make clean` command (as described in the PP#1 specification, page 18, item VII.7).

☐ Script session `make my_number`: In my scripting session, I properly did the `make my_number` command (as described in the PP#1 specification, page 19, item VII.8).

☐ Script runs: In my scripting session, I did the correct number of runs, in the correct order, with appropriate values (as described in the PP#1 specification, page 19, item VII.9 and page 17, item VI.5).

☐ Script session termination: In my scripting session, after completing the appropriate commands, I terminated the scripting session using `[Ctrl]-D`

    (as described in the PP#1 specification, page 19, item VII.10).

☐ Script file cleanup with `dos2unix`: After my scripting session, I cleaned up my script file `pp1.txt` using the `dos2unix pp1.txt` command (as described in the PP#1 specification, page 19, item VII.12).

☐ Script file unedited: After cleaning up my script file `pp1.txt` using the `dos2unix` command, I NEVER edited or altered my script file `pp1.txt` in any way (as described in the PP#1 specification, page 19, item VII.13).
CHECKLIST ITEMS FOR SUBMISSION

☐ Printing font: In printing my script file pp1.txt, I used the Courier New font in 10 point size (as described in the PP#1 specification, page 21, item VIII.18).

☐ Printing margins: In printing my script file pp1.txt, I used one inch margins on all sides (as described in the PP#1 specification, page 21, item VIII.21).

☐ Cover page: On my cover page, I verified that I have listed all of the required information, in the correct order (as described in the PP#1 specification, page 23, item IX.1.a).

☐ Summary essay: In my summary essay, I verified that I have included all of the required sections and information, in the correct order (as described in the PP#1 specification, page 23, item IX.1.b).

☐ Summary essay references section: In my summary essay, I verified that I have included a references section, even if I have no references (as described in the PP#1 specification, page 23, item IX.1.b.vi).

☐ Summary essay long enough: My summary essay is at least a half page single spaced/full page double spaced (as described in the PP#1 specification, page 23, item IX.1.b).

☐ Summary essay font size: My summary essay is printed with a font size of 10 to 12 points (as described in the PP#1 specification, page 23, item IX.1.b).

☐ Summary essay margins: My summary essay is printed with margins of 3/4 inch to 1 inch on all sides (as described in the PP#1 specification, page 23, item IX.1.b).  

☐ Cover page and summary essay on separate sheets: My cover page is on a separate sheet of paper from my summary essay (as described in the PP#1 specification, page 23, item IX.1.b).

☐ Binding order: My submission is bound in the proper order: (1) cover sheet, (2) summary essay, (3) script file with its pages in the correct order, (4) this checklist (completed) with its pages in the correct order, (5) optionally the bottom half of the completed bonus extra credit form if any (as described in the PP#1 specification, page 23, item IX.2).

☐ Binding: My submission is bound with a proper fastener such as a staple or binder clip (as described in the PP#1 specification, page 23, item IX.2).

☐ Proofreading: Before submitting, I thoroughly PROOFREAD every part of my submission: my cover page, my summary essay, my script printout, this completed checklist, and optionally the bottom half of the completed bonus extra credit form if any.

☐ Upload: I uploaded, to the Gradescope dropbox for PP#1, BOTH my C SOURCE file my_number.c AND my SCRIPT file pp1.txt BUT NOT my executable file nor any other file (as described in the PP#1 specification, pages 24-25, item IX.3).

☐ Upload verification: I verified that I uploaded the correct files — and only the correct files — to the Gradescope dropbox for PP#1.

☐ Files NEVER deleted: For the entire semester, I NEVER will delete my C source file nor my script file, even after this programming project is graded and returned to me.

☐ Graded paper submission NEVER thrown away: For the entire semester, I NEVER will throw away my graded paper submission, even after this programming project is graded and returned to me.

☐ Paper submission: I will submit my paper submission (as described in the PP#1 specification, page 26, item IX.4).

☐ Paper submission on deadline: I will submit my paper submission on time.

☐ Paper submission late: I will submit my paper submission late.
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.

Submit this copy.

In your paper submission, attach this copy at the VERY END.
If you put this in the wrong place in your submission, then you WON’T get the extra credit.