CS 1313 010: Programming for Non-majors in C, Spring 2025
Programming Project #1: Thinking of a Number
Due by Wednesday January 29 2025 9:50am Central Time

`http://cs1313.oucreate.com/`

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 Jan 22 2025 5:00pm Central Time.** 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 OUNetID (the first 4 letters of your last name **in all lower case**, followed by a 4 digit number, which might be 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.

IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT!
NEVER DELETE ANY FILE ASSOCIATED WITH CS1313, EVER!

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

NOTES

1. Except where and as explicitly permitted in writing (for example, in a Programming Project specification), you are **ABSOLUTELY FORBIDDEN** to **COPY EVEN A SINGLE CHARACTER** from, or to have **ANY** shared code with, **ANY** other entity, whether a human being (regardless of whether they're in CS1313 or not), a text resource, a computing resource or anything else, whether in person, on a local computer, online or anywhere else. It's **INCREDIBLY EASY** for us to detect such copying, so **DON'T EVEN THINK ABOUT IT!**

2. **EVERY** rule described in this Programming Project (PP) #1 specification will also apply to **EVERY** future PP, except where and as explicitly stated otherwise, in writing (for example, in a future PP specification). The same will apply to **EVERY** rule described in **EVERY** future PP specification: **EVERY** such rule will apply to **EVERY** PP after that, except where and as explicitly stated otherwise, in writing.

I. LOG IN

1. Connect and log in, from a computer that you're sitting at, to:
   `ssh.ou.edu`

   NOTE: `ssh.ou.edu` **ISN'T** a webpage, so you **CANNOT** access `ssh.ou.edu` from a web browser. Instead, you **MUST** follow the instructions below.

   (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. Type your OUNetID username and password in the appropriate text boxes.

      iv. Click the $\boxed{\rightarrow}$ button, **OR** press the **`Enter`** key on the keyboard.

      v. Wait patiently while the PC logs you in and starts up various features.

      vi. Run a program named `PuTTY`, which lets you log in remotely to the computer that we're going to use, as follows:

         A. On the bottom left of the screen, or on the main desktop, or in the Windows Start menu, click on the icon for your preferred web browser (for example, Firefox, Google Chrome, Microsoft Edge).

         B. In the web browser, go to the CS1313 website:
         `http://cs1313.oucreate.com/`
         NOTE: **NO** `www` at the beginning of the "hostname" of the 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. Go to the link listed for PuTTY on that webpage.

         F. Under
         `Alternative binary files`
         specifically under
         `putty.exe (the SSH and Telnet client itself)`
         find the 64-bit x86 version and right-click on the link
         `putty.exe`

         G. Save `putty.exe` to your Windows desktop (or wherever you want it).

         H. Double-click on the `PuTTY` icon on your desktop (or wherever it's saved).

      vii. 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.

viii. 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`

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

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

xi. 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 wide (which is the default) and
EXACTLY 40 rows high (which **isn't** the default, so you'll have to change it).

xii. 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.

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

xiv. 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`
(This might already have been selected by default.)

xv. On the left side of the `PuTTY Configuration` window, the fourth (and final) major entry is
`Connection`
Click on it.

xvi. This will bring up a section titled
`Options controlling the connection`
Below that is a section titled
`Sending of null packets to keep session active`
and below that is an item titled
`Seconds between keepalives (0 to turn off)`
In the text box on the right side of that line, set the value to
`30`

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

xviii. 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`

xix. 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).

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

xxi. If a `PuTTY Security Alert` window pops up, click
`Accept`

xxii. When prompted to `login as`, type your OUNetID username with
ALL LETTERS IN LOWER CASE (small), and then press **`Enter`**.

xxiii. When prompted for your password, type your password (which should be your OUNetID 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.

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

xxv. Go to step I.2 on page 5.

(b) **From your own Windows PC, or a Windows PC not in Carson 205, 206 or S-18:**

Once you've logged in to that PC, do steps I.1.a.vi-xxv, above.
If you're using your own PC, you might want to save your PuTTY configuration:
In the `Session` section of PuTTY, to the right, click on
`Save`
(On OU's PCs, you might not be able to do that.)

(c) **From your own Apple MacOS or a Unix/Linux computer, or a Mac or Unix/Linx computer not in Carson 205, 206 or S-18:**

Once you've log in to that MacOS or Unix/Linux computer, in your preferred web browser, 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 use the `ssh` command from the Unix command line, like this:

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

where `yourusername` is your user name (that is, your OUNetID, with letters in **all lower case**).

You should make sure that the terminal window is **EXACTLY** 80 columns wide by **EXACTLY** 40 rows high, so you might need to resize the terminal window to that size.

You should also make sure that the font is a fixed width font such as Courier New, so you might need to set that font.

2. If you cannot log in to `ssh.ou.edu`, try logging in to:
   `coe-polk.ou.edu`
   OR
   `coe-kennedy.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:
   ```
   > █
   ```
   (In `PuTTY`, the text cursor will probably be green; other terminals might have other colors for the text cursor.)

   If there is information before (to the left of) the prompt character, it might be the name of the computer that you've logged in to (which might 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 something like this:

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

   then you should log out immediately by entering `exit` at the Unix prompt (you might have to do this twice to log out fully), and then log back in.
   The same is true if there are lines of text something like this:

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

   (This might happen if you are, or recently were, an OU employee.)
   In which case, do as described just above.

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
   ```

   Or you might find that the output of the `pwd` command is:

   ```
   /oushomes/FacStaff/yourusername
   ```

   if you are, or recently were, an OU employee.

   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 might have to do this twice to log out fully), and then log back in.

   You might 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 *small-L small-S* **INSTEAD OF** *one small-S*, and that `ls` is short for "list."

5. 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.

6. Log out of the Linux computer by entering `exit` at the Unix prompt (you might 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. UNDERSTANDING SECURE SHELL

If you've got some confusion about what `PuTTY` or the Terminal are, here's an analogy that might help.

Suppose I tell you that I want you to pick up some carrots off a counter and place them in a bowl.

But, I want you to pick up a specific group of carrots from a specific counter, and place them into a specific bowl.

(For simplicity, let's call them "my carrots," "my counter" and "my bowl.")

You could go home to your house/apartment/residence hall with a box of carrots and a bowl, and put your carrots into your bowl.

But the task I've assigned requires you to pick up **my** carrots from **my** counter and put them into **my** bowl.

So picking up your carrots off your counter and putting them into your bowl wouldn't successfully complete the assignment: you need access to my carrots, my counter, and my bowl.

And no, you can't come over to my house to do it.

But now suppose that I point you to an app that enables you to manipulate a robot in my kitchen that can pick up my carrots from my counter and put them into my bowl.

If you install that app (on your phone, tablet, PC, whatever) and then manipulate that robot to do that task, would you have properly completed the assignment?

Here's an example, though not involving carrots:

`https://www.youtube.com/watch?v=1bEaiADnxZc`

(The sound on this video is very annoying, so we recommending muting it.)

In CS1313, we use a "Secure Shell client" app (for example, PuTTY, or the MacOS Terminal) to enter commands on ssh.ou.edu.

Note that ssh.ou.edu isn't your PC, but instead is a couple of server computers in a data center somewhere on the OU Norman campus.

(Where they are physically doesn't matter.)

Chances are that you'll never physically see, let alone touch, ssh.ou.edu.

And you aren't typing on the keyboard of ssh.ou.edu (there might not even be a keyboard for it).

Instead, you're remotely accessing ssh.ou.edu and typing your Unix commands (and editing etc) on your local keyboard on your own PC (or one of the PCs in Carson 205, 206 or S-18), **AS IF** you were using the keyboard of ssh.ou.edu.

That is, you're manipulating ssh.ou.edu remotely, via Secure Shell (ssh) login, for example through the PuTTY Secure Shell client app or a Terminal window.

IV. 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 might 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 might 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 *small-L small-S* **INSTEAD OF** *one small-S*, 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 might 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.

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

1. If necessary, repeat IV.1-4, then definitely repeat IV.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!
   ```

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

1. If necessary, repeat IV.1-4, then definitely repeat IV.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 Help  ^O Write Out  ^W Where Is  ^K Cut    ^T Execute  ^C Location
   ^X Exit  ^R Read File  ^ Replace    ^U Paste  ^J Justify  ^_ Go To Line
   ```

   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 Location` causes `nano` to tell you what line number the green text cursor is located at. Another example: `^K Cut` means "delete the entire line that the green text 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 email address, and the lab information, so that they are **your** information.
   IMPORTANT: This is the ONLY comment you should edit!
   (b) Save (see VI.5, below), exit `nano` (see VI.10, below), compile using the `make` command (see V.4, above), run (see V.5, above). This will test your first set of edits.
   (c) Edit your C source file `my_number.c` again. In the *declaration section*, specifically in the *named constant subsection*, change the numeric values of the named constants (the number after each of those equal signs) 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) Edit your C source file `my_number.c` again. 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 capital letter O), 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:

   ```
   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**.

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

1. If necessary, repeat IV.1-4, then definitely repeat IV.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 VII.2, above.
   NOTE: **EVERY SINGLE TIME** you edit your C source file `my_number.c`, you should then **IMMEDIATELY** return to step VII.2, above.
4. In the worst case, if you're totally stumped, then copy the original from Dr. Neeman's home directory again (see IV.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 VII.3 above you'll need to edit it and figure out where things went wrong, and then return to step VII.2. Again, in the worst case, if you're totally stumped, then copy the original from Dr. Neeman's home directory again (see IV.8), and start editing the fresh copy.

VIII. CREATE A SCRIPT FILE

1. If necessary, repeat IV.1-4, then definitely repeat IV.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
   ```
   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*. 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.
   EXPLANATION: 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).
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 Jan 28 18:27 my_number
   -rw-r--r--  1 5013  100    2976 Jan 28 19:10 my_number.c
   ```
   NOTICE that the command is:
   small-L small-S space hyphen small-L
   It is **NOT NOT NOT**
   small-L small-S 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 output 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 output 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
   ```
   WARNING WARNING WARNING!!! 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
   ```
   WARNING WARNING WARNING!!! 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 VII.5, above.
10. **IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT! IMPORTANT!**
    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 Canvas (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 VIII.10, above).
    (b) Repeat steps VIII.2-10, above.

    That is, the new scripting session will start by deleting the old script file (of the same filename).

NOTE: It's fine to create a new script file (as many times as you want), but once a script file has been created, it's **ABSOLUTELY FORBIDDEN** to edit it in any way.

15. **IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!**
    Carefully **PROOFREAD** your script file, especially:

    - Did you do the following command?
      `pwd`
    - Did you do the following command?
      `ls -l`
      Especially, is it *small-L small-S space hyphen small-L*?
      Look CLOSELY at its output, which should look something like this:
      ```
      -rw-r--r--  1 5013  100       83 Jan 17 12:15 makefile
      -rwxr-xr-x  1 5013  100    13717 Jan 28 19:10 my_number
      -rw-r--r--  1 5013  100     2976 Jan 28 18:57 my_number.c
      -rw-r--r--  1 5013  100        0 Jan 28 19:22 pp1.txt
      ```
    - Did you do the following command?
      `cat makefile`
    - Did you do the following command?
      `cat my_number.c`
    - Did you do the following command?
      `make clean`
      Was the output as follows?
      ```
      rm -f my_number
      ```
      Were there any error messages or warning messages?
    - Did you do the following command?
      `make my_number`
      Especially, did you use the name of the executable, **WITHOUT** the .c extension?
      Was the output as follows?
      ```
      gcc -o my_number my_number.c
      ```
      Were there any error messages or warning messages?
    - Did you run the program the correct number of times, with the correct inputs, using the following command?
      `my_number`

IX. DOWNLOAD YOUR SCRIPT FILE AND YOUR C SOURCE FILE

— 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 to, or Printing a File from, a PC Not in Carson” and follow the instructions.

— For a Windows PC:

1. Use the same instructions as in I.1.a.vi.A-H on page 2 to download, to the Windows PC that you’re logged in to, the 64-bit version of `psftp` (instead of downloading `PuTTY` as you originally did).
2. Run `psftp` by double-clicking on its icon.
3. At the `psftp` prompt, enter:
   `open ssh.ou.edu`
4. You might be prompted to ask a Yes/No question, for example:
   `Store key in cache?  (y/n)`
   If that happens, enter:
   `y`
5. At the `login as` prompt, enter your OUNetID username.
6. At the `password` prompt, enter your OUNetID password. **NOTHING WILL APPEAR AS YOU TYPE YOUR PASSWORD.** This is normal for Unix.
7. At the `psftp` prompt, enter:
   `pwd`
8. At the `psftp` prompt, enter:
   `ls`
   Remember, that’s *small-L small-S*, **NOT** *one small-S*.
9. At the `psftp` prompt, enter:
   `cd CS1313`
   Remember, that’s *big-C big-S* one three one three – Unix is case sensitive.
10. At the `psftp` prompt, enter:
    `pwd`
11. At the `psftp` prompt, enter:
    `ls`

(These instructions continue on the next page.)

12. At the `psftp` prompt, enter:
    `lcd C:\Users\yourusername\Desktop`
    <u>**NOTE:**</u> The command is LOWER CASE LCD (*small-L small-C small-D*); the first character <u>**ISN'T**</u> 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).

    <u>**ALERT:**</u> The `lcd` command works in `psftp` (and other Secure File Transfer Protocol logins) **ONLY**, not in regular Unix SSH logins (for example, `lcd` **WON'T** work in a terminal window such as PuTTY).

    <u>**ALERT:**</u> Your OUNetID username will only work on the Windows PCs in Carson 205, 206 and S-18 (and perhaps on other OU-managed PCs on campus), but **NOT** on PCs elsewhere (especially, **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 23.

    <u>**ALERT:**</u> You can use
    `lcd C:\Users\yourusername\Desktop`
    on Windows 10 (and older), but on Windows 11, you might need to use
    `lcd C:\Users\yourusername\Documents`
13. At the `psftp` prompt, enter:
    `get pp1.txt`
    <u>**NOTE:**</u> The filename is LOWER CASE PP1.TXT (*small-P small-P one period small-T small-X small-T*); the third character <u>**IS**</u> a one.

    <u>**ALERT:**</u> The `get` command, and its counterpart, `put`, work in `psftp` (and other Secure File Transfer Protocol logins) **ONLY**, not in regular Unix SSH logins.
14. At the `psftp` prompt, enter:
    `get my_number.c`
15. At the `psftp` prompt, enter:
    `exit`
    (Or, click on the little X in the upper right of the PSFTP window.)
16. 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, or a lizard on a piece of notebook paper with a pencil, or something else). If you can't find the icon for `pp1.txt` on the desktop, then in the bottom left of the screen, click on the folder icon for File Explorer, and when the File Explorer folder opens, on the left side click on **`This PC`**, double-click on **`Desktop`** icon, and then you'll see `pp1.txt` and `my_number.c`, though the extension after the dot might not be shown for one or both of those filenames.

    <u>**ALERT:**</u> On Windows 11, you might instead need to go into your **`Documents`** folder instead of your **`Desktop`**.

FINDING YOUR USERNAME ON A WINDOWS 10 PC: As described in item IX.12, above, your OUNetID might not work on PCs not in Carson 205, 206 or S-18 (and perhaps other OU-managed PCs on campus), especially on PCs not configured and monitored by OU IT. 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
   `OS (C:)` **OR** **`Windows (C:)`** **OR** **`This PC`**
   (It might be necessary to double-click, or single-click might work.)
6. If you clicked on **`This PC`**, then you might now need to (single- or double-) click on
   `OS (C:)` **OR** **`Windows (C:)`**
   (or similar).
7. Find the folder named
   `Users`
   and double-click on it.
8. In the resulting list of folders, you should be able to identify your username on that PC.

X. WHAT TO SUBMIT: SUMMARY ESSAY, C SOURCE FILE, SCRIPT

1. **COMPOSE** the following document, using a word processor or text editor (you are **ABSOLUTELY FORBIDDEN** to submit these items written by hand):
   (a) 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 might 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.

   Your summary essay **MUST** be named:
   `pp1_essay.docx` **OR** `pp1_essay.doc` **OR** `pp1_essay.pdf`

   NOTE: The first part of your summary essay's filename **MUST** be
   small-P small-P **ONE** underscore `essay` dot
   followed by the filename extension (`docx` or `doc` or `pdf`).

   Especially, your summary essay filename is **ABSOLUTELY FORBIDDEN** to have a small-L in it.

2. You **MUST** upload **ALL** of the following to the PP#1 upload box ou OU's Canvas website:
   - your summary essay (in Word or PDF format)
   - your **C source file** **`my_number.c`**
   - your **script file** **`pp1.txt`**

   NOTE: DON'T upload your executable named `my_number` (without the `.c` extension at the end of the filename), nor your makefile named `makefile`.

   (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 — you're welcome to try another browser, but it might not work.

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

   (c) Find and click on the link for
   `C S-1313-010 Spring 2025`
   You might 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 Spring 2025`
   (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`
   If you don't see that word, then at the bottom of that vertical stack of words, click on
   `New Analytics`
   and then 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:
   `PP1`

   (f) You'll now be on the PP#1 page. On the right hand side of the window near the top, click on the button labeled
   $\boxed{\texttt{Start Assignment}}$

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

   (h) In the **`File Upload`** tab, click on:
   $\boxed{\texttt{Browse}}$

   (i) This will pop up a window titled
   `File Upload`
   On the left side of the `File Upload` window, if it's visible, click on:
   `Desktop`
   Or, navigate to your desktop in the window named `File Upload`.

(j) In the main panel of the `File Upload` window, double-click on:
`my_number.c`
The file might appear to be named **`my_number`**, so carefully inspect the little icon associated with the filename, to see whether it's a letter C; if that's the case, then that's actually **`my_number.c`**. You can also look at the text to the right of the filename, which typically explains what kind of file it is — you want `C Source` **OR** `C File`.
NOTE: AVOID `my_number.c~` (with a tilde after the .c), which is the next-to-last version of your C source file, **NOT** the final version that you want to upload.

(k) Once you've chosen the file, it'll appear just to the right of the button labeled
`Browse`
Below that, click on:
`Add Another File`

(l) Repeat the procedure above to find and add your script file:
`pp1.txt`
The file might appear to be named **`pp1`**, so carefully inspect the little icon associated the left of the filename; if it's a little box with a pencil, then that's actually **`pp1.txt`**. You can also look at the text to the right of the filename, which typically explains what kind of file it is — you want `Text Document`.
NOTE: AVOID `pp1.pdf` if it's there.

(m) Repeat the procedure above to find and add your summary essay file:
`pp1_essay.pdf` OR `pp1_essay.docx` OR `pp1_essay.doc`
The file might appear to be named **`pp1_essay`**, so carefully inspect the little icon associated with the filename.

(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
`New Attempt`
in the upper right of the webpage. **NOTE**: We will grade the final version that you upload; older versions will be discarded without being looked at.

(q) On the right hand side of the webpage (in your web browser), inspect the files that you've uploaded, to make sure that they're the correct files, with **EXACTLY** the following filenames:
`pp1_essay.pdf` **OR** `pp1_essay.docx` **OR** `pp1_essay.doc`
`my_number.c`
`pp1.txt`

(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 your summary essay, your C source file and your script file for **EVERY** programming project in CS1313, unless otherwise announced.

XI. EXTRA CREDIT

HELP SESSION BONUS 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 Jan 29.
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. You may collect this extra credit bonus only **ONCE** for PP#1. That help session bonus will be worth an extra 5% of the value of PP#1.

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

COMMON PROBLEMS

- Case sensitivity: Unix and C are **case sensitive.**
- CS1313 subdirectory: **ALL** of your CS1313 work should be in your `CS1313` subdirectory, including but not limited to PP#1.
- The dot: Remember the dot at the end of some of the `cp` commands.
- `ls`: The command is *small-L small-S*
  NOT *one small-S*.
- `ls -l`: The command is *small-L small-S space hyphen small-L*,
  NOT *small-L small-S 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.
- **PROOFREAD PROOFREAD PROOFREAD** your script.
- 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.**
- Upload: Be sure to upload the **SUMMARY ESSAY** file **AND** 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 might 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 might be assigned mini-projects that require you to modify a completed project; if you've deleted that project, then you might 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 might 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

- ☐ 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, page 3, item I.1.a.xi and page 5, item I.1.c).
- ☐ 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.a.xii).
- ☐ 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 7, 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 7, 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 8, 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 IV.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 IV.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 IV.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 IV.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 IV.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 V.2).
- ☐ Look at `makefile`: I examined my `makefile` using the `cat` command (as described in the PP#1 specification, page 12, item V.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 V.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 V.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 VI.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 VI.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 VI.4.c).
- ☐ String literals: In my version of the C source file `my_number.c`, in the execution section, 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 VI.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 VI.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 VI.7).
- ☐ String literals: In my version of the C source file `my_number.c`, in the execution section, I verified that each string literal is contained entirely on a single line (as described in the PP#1 specification, page 15, item VI.8).
- ☐ 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 VI.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 VI.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 VII.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 VII.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 VIII.2).

☐ Script session `pwd`: In my scripting session, I properly did the
`pwd`
command (as described in the PP#1 specification, page 18, item VIII.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 VIII.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 VIII.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 VIII.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 VIII.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 VIII.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 VIII.9 and page 17, item VII.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 VIII.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 VIII.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 VIII.13).

☐ Script proofread: After finishing my script file `pp1.txt`, I thoroughly proofread my entire script file (as described in the PP#1 specification, page 20, item VIII.15).

CHECKLIST ITEMS FOR SUBMISSION

- ☐ Downloads: I downloaded my C source file and my script file to the PC that I wanted to upload to Canvas from (as described in the PP#1 specification, pages 21-22, items IX.1-16).
- ☐ 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 X.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 X.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 X.1.b).
- ☐ Proofreading: Before submitting, I thoroughly **PROOFREAD** every part of my submission: 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 Canvas upload box for PP#1, my **SUMMARY ESSAY** file, 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 25-27, items X.2.a-t).
- ☐ Upload verification: I verified that I uploaded the correct files — and only the correct files — to the Canvas upload box for PP#1.
- ☐ Files **NEVER** deleted: For the entire semester, I will **NEVER** delete my C source file nor my script file, even after this programming project is graded and returned to me.