PROBLEM:
Write a function to figure out how many numbers fall within a given range, and then write a makefile to pull the whole program all together.
This function will be in its own file.
The function prototype is:

```
/* function to return average & distributed grade count   */
        void get_range_count(int number_list[],     /* input, array that holds data            */
                             int real_filesize,     /* input, actual size of the data in file  */
                             int *range_count);     /* output, number of values in range       */
```

You will need the file **lab7.c** as your main/driver program for the function. This main program will set things up, read the values from the file, and print the output sentences.
You will also need: lab7.h, three other .c files, two data files.

TO GET THE FILES YOU NEED
First move to your class folder by typing: **cd csc60**
The following command will create a directory named **lab7** and put all the needed files into it below your csc60 directory.

Type: **cp -R /home/college/bielr/files_csc60/lab7 .**
Spaces needed: (1) After the **cp** ↑ *Don't miss the space & dot.*
(2) After the **-R**
(3) After the directory name at the end & before the dot.

After the files are in your account and you are still in **csc60**, you need to type: **chmod 755 lab7**
This will give permissions to the directory.
Next move into lab7 directory, and type: **chmod 644 *.***
This will give permissions to the files.
Your new lab7directory should now contain:
lab7.c, lab7.h, get_data.c, print_all.c, lab7a.dat, lab7b.dat

INPUT/OUTPUT DESCRIPTION:
The input is two lists of unknown length of integer values in the file **lab7a.dat** and **lab7b.dat**.
They represent test grades. The print output statements are provided.

ALGORITHM DEVELOPMENT - Pseudocode:

```
//Lab7.c
int main(void)     /* given to you */
    Loop through each file
        Call get_data function and get the real_filesize.
        Call get_range_count.
        Call print_all and print out the filename, the range_count and the real_filesize.
/*-----------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
int get_data (const char * filename, int number_list[])
        // sub-function given to you in the file named get_data.c
    Open the data file and check for error on open.
    Read and count the values, putting them into an array, returning the real_filesize.

/*-------------------------------------------------------------------*/
void print_all (const char * filename, int real_filesize, int *range_count)
        // sub-function given to you in the file named print_all.c
     Print your name & assignment number.
     Print to the screen, rather than to a file.
     Print the headers, value in the range, and the total number of values in the file

/*-------------------------------------------------------------------*/
// A sub-function for you to write.
// Place it in a separate file by typing:  vim get_range_count.c

/*-------------------------------------------------------------------*/
// Your name here
void get_range_count(int number_list[],     /* input, array that holds data   */
                     int real_filesize,     /* input, actual size of the data */
                     int *range_count)      /* output, number of values in range */

   set *range_count to zero

   for loop from zero to < real_filesize, incrementing by one
   |     if the current number from the array is within 90 through 99
   |          Add one to the *range_count
   |         (Use parentheses as needed)
   |_
   return
/*-----------------------------------------------------------*/

/*-----------------------------------------------------------*/
/* lab7.h                                                    */

#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 50        /* max length of a file of numbers */

/* function to get the data and return real_filesize         */
int get_data(const char *filename,  /* input, current file name */
             int number_list[]);    /* output, the filled array */

/* function to return average & distributed grade count   */
```

```
void get_range_count(int number_list[],     /* input, array that holds data   */
                     int real_filesize,     /* input, actual size of the data */
                     int *range_count);     /* output, number of values in range */

/* function to print information  */
void print_all( const char *filename,  /* input, the current filename    */
                int real_filesize,     /* input, actual size of the data */
                int *range_count);     /* input, # of values in range    */
/*-------------------------------------------------------------*/
```

DEFINED OUTPUT APPEARANCE:

Print statements are included for you.

```
[bielr@ecs-pa-coding3 lab7]$ range

Your Name.  Lab 7.

File lab7a.dat:
There are 12 values in the range of 90 through 99
out of a total of 29 values.


Your Name.  Lab 7.

File lab7b.dat:
There are 10 values in the range of 90 through 99
out of a total of 29 values.

[bielr@ecs-pa-coding3 lab7]$
```

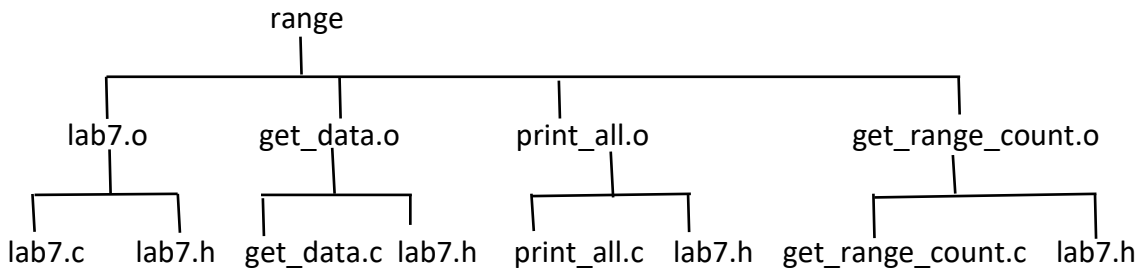
REMINDERS:

(1) Remember to put your name as a comment at the top of each code file you submit.

(2) You should examine the data files and confirm the correctness of the answer produced by your program.

➔ More on next page

CREATING A MAKE FILE: Use the slides 12&14 of 5-UNIX as a reference which are pasted at the end of this file.



Make comment with your name in it.
Start with a CC = gcc line.
Second line: use a name like *range*, followed by the *.o files and the *.h file
Third line: one or two tabs followed by the *.o files and the rename of the executable
Fourth and Fifth lines: the *o file name, followed by a colon, followed by the *.h filename
Above and below the fourth text line, include empty lines.

PREPARE YOUR FILE FOR GRADING:

When all is well and correct,
Type: **script StudentName_lab7.txt** [Script will keep a log of your session.]
Type: **touch lab7.h** to force a recompilation (not necessary every time you compile)
Type: **make** to compile the code
Type: **range** to run the program to show the output of the program
(or whatever name you use **./range** or **lab7** or **a.out** for the executable)
Type: **exit** to leave the script session

Turn in your completed session: 25 points

Go to Canvas and turn in:

1. get_range_count.c
2. print_all.c (with your name added)
3. makefile
4. your script session (StudentName_lab7.txt).

This assignment is available on 3/15.
This assignment is **due** by the end of 4/7 for a chance at full points (25 points).
If turned in before the end of 4/21 you lose 2 points. (23 points)
This assignment will not be accepted for any points after 4/21.

➔ More on next page

Helpful slides:

Slide 12:

Second pass at a makefile:
Look at its contents. For lab7, we will have a custom header file.

>**cat makefile**

```
# Your name here

power2: power2.o compute.o  p2.h
    gcc power2.o compute.o  -o power2

power2.o: power2.c  p2.h
    gcc -c power2.c

compute.o: compute.c  p2.h
    gcc -c compute.c
```

Slide 14: ***/* Helpful Comments */***

- Start by opening *vim*, and typing in the commands to a file named *makefile*.
  Close *vim* and then at the prompt, type: **make**
- When you enter **vim**, type: **:set list**
  This will show the non-printable characters:
  ^I = tab
  $ = end of line
- To reverse the setting, type: **:set list!**
- To create a tab on *athena*, you may have to hit the tab key **twice** in a row.

Create a Makefile - Written Step-by-Step Assistance

- Type: **vim makefile** to create a makefile
- On the first lines, use "#" at the start of each line for comments of your name and lab5
- Write the first and final rule to link it all together.
  - Line 1 of the rule: Put the name of the executable **lab5**, followed by a colon, followed by all the function names ending with a ".o"
  - Line 2 of the rule: Hit the key: **tab**, then type: **gcc**. Enter the names of all the functions again ending with "**.o**". Add in a **-o lab5** for the change of the executable name.
  - Example from another program:

```
radii: lab5.o find_two_radii.o
    gcc lab5.o find_two_radii.o -o radii -lm
```

- Next, we must figure out what to do if any of those files listed above need to be recompiled. The make utility will check the date of the **.c** file against the date of **.o** file. If they are out of sync, then the **.c** file will get recompiled. The next step is to create multiple rules to take care of each file. So, to do that……
    - Line 1 of the rule: put the name of the .o file followed by a colon. Then add the name of the **.c** and **.h** files that the **.o** file is dependent on.
    - Line 2 of the rule: Hit the tab key, then type **gcc -c** , then the name of the **.c** file
    - Example from another program:

        find_two_radii.o: find_two_radii.c lab5.h
            gcc -c find_two_radii.c -lm

- We need to repeat the above so there is a rule for each file. An empty line between each rule makes for readability. A final example for this other program would be:

```
#Your Name Lab 5

radii: lab5.o find_two_radii.o
        gcc lab5.o find_two_radii.o -o radii

lab5.o: lab5.c  lab5.h
        gcc -c lab5.c

find_two_radii.o: find_two_radii.c lab5.h
        gcc -c find_two_radii.c
```