**PROBLEM**:

1. Write a *function* **class\_stats** that returns the average of a set of grades and their distribution. The function will be in its very own **file,** separate from the other code.
2. Write a *makefile* to pull everything together.

# The function prototype is:

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

void **class\_stats**(int grades[], /\* input, array that holds data \*/

int number\_of\_grades, /\* input, actual size of the file \*/

double \*average, /\* output, average of the array \*/

int distribution[]); /\* output, distribution count \*/

**Files needed:**

* You will need to copy the file **lab5.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 need to copy the file **lab5.h** which I will give you.
* You will need to *write* **class\_stats** in its own separate file.
* You will need to *write* **makefile** to pull all the files together.

**TO GET THE FILES YOU NEED:**

First move to your class folder by typing: **cd csc60**

The following command will create a directory named **lab5** and put all the needed files into it below your csc60 directory.

Type: **cp -R / home/college/bielr/files\_csc60/lab5 .**

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 lab5**

This will give permissions to the directory.

Next move into lab5 directory (**cd lab5**), and type: **chmod 644 \***

This will give permissions to the files.

Your new lab5 directory should now contain: lab5.c, lab5.h, lab5a.dat, lab5b.dat, lab5s.dat

Note: lab5.c includes two other functions: get\_data, print\_all.

The make program deals with the two files, not the 4 functions. See chart, next page.

Reminder: the function class\_stats must be in its own file named class\_stats.c

These files are also downloadable from Canvas.

**INPUT/OUTPUT DESCRIPTION**:

The **input** is two lists of unknown length of integer values in the files **lab5a.dat** and **lab5b.dat**

Each file consists of test grades.

The **output** printf statements are provided in lab5.c, function print\_all.

**DEFINED OUTPUT APPEARANCE**:

Print statements are included in the code, and require no changes, except for your name.

Your Name. Lab 5.

Stats for lab5s.dat: // using the **sample** data file

Average grade = 51.857143

Grade Distribution:

Category 100 - 0

Category 90 - 1

Category 80 - 1

Category 70 - 0

Category 60 - 1

Category 50 - 0

Category 40 - 1

Category 30 - 1

Category 20 - 1

Category 10 - 1

Category 0 - 0

**WHAT DOES IT MEAN “To put the code in its own file”:**

Type: vim class\_stats.c

*Inside of the file, type:*

// Your name

// Lab 5, class\_stats function

#include “lab5.h”

void class\_stats( int grades[], int number\_of\_grades, double \*average, int distribution[])

{

*and then keep typing the needed code according to the algorithm.*

🡪 more on next page

**REMINDERS**:

* You should look at all the provided files and get an idea of what code each file contains.
* Remember to put your name and Lab 5 in the comment header of your function, in your makefile, and in the output.
* You should examine the data file and confirm the correctness of the answer produced by your program.
* Before you do your final run, you need to make a change in lab5.c.

In lab5.c, near the top, you will see this code:

const char \*FILENAME[] = /\* array of the sample data file name \*/

{"lab5s.dat", NULL };

//const char \*FILENAME[] = /\* array of the data file names \*/

// {"lab5a.dat", "lab5b.dat", NULL };

**You will need to remove the left most comment marks off of one pair of lines and onto the other pair to look like this:**

//const char \*FILENAME[] = /\* array of the sample data file name \*/

// {"lab5s.dat", NULL };

const char \*FILENAME[] = /\* array of the data file names \*/

{"lab5a.dat", "lab5b.dat", NULL };

**ALGORITHM DEVELOPMENT - Pseudo code**:

/\*-------------------------------------------------------------------------\*/

main /\* main is given to you as ***lab5.c***  \*/

Print your name for output.

for loop through the two data files

| Call function **get\_data** Will open \*.dat files, mentioned in lab5.h.

| Call function **class\_stats** You write this one.

|\_ Call function **print\_all**

🡪 more on next page

/\*-------------------------------------------------------------------------\*/

/\* This code will reside in a file **class\_stats.c** \*/

/\* In a *separate file from the other functions* \*/

// You need to do a #include of **lab5.h**

void class\_stats(int grades[], /\* input, array that holds data \*/

int number\_of\_grades, /\* input, actual size of the file \*/

double \*average, /\* output, average of the array \*/

int distribution[]) /\* output, distribution count \*/

Initialize **\*average** to zero.

for loop from 0 to less than **DIST\_SIZE**

|\_ Set each element of the **distribution** array to 0

for loop from zero to < number\_of\_grades, incrementing by one

| add the current value from the grades array into \*average or sum

| compute the correct position in distribution array for current grade

| (Hence divide the current grade by ten, using integer division.)

| Add one to the count in the distribution array, based on the position

|\_ just figured above.

[***Example****: if grade is 66, the position would be 6 (because it is a*

*grade between 60 and 69) and distribution[6] would have a one added to it.*]

Compute the \*average, dividing either \*average or sum by the **number\_of\_grades**, putting the

answer back into **\*average**

return

/\*-------------------------------------------------------------------------\*/

**CREATING A MAKE FILE:**  Use the slides 13-14 of 5-UNIX as a reference. Also pasted at the end of this file.

lab5

lab5.o class\_stats.o

lab5.c lab5.h class\_stats.c lab5.h

* 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: press: **tab**, then type: **gcc**. Enter the names of all the functions again ending with “.o”. Add in **-o lab5** for 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: 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 -lm*

*lab5.o: lab5.c lab5.h*

*gcc -c lab5.c -lm*

*find\_two\_radii.o: find\_two\_radii.c lab5.h*

*gcc -c find\_two\_radii.c -lm*

**PREPARE YOUR FILE FOR GRADING:**

Make sure your output is **NOT** printing lab5**s**.txt, but output from lab5a.dat & lab5b.dat

When all is well and correct,

Type: **script StudentName\_lab5.txt** [Script will keep a log of your session.]

Type: **touch lab5.h** to force a recompilation

Type: **make** to compile and link the code

Type: **lab5** to run the program to show the output of the program

(or whatever name you used for the executable, I used *grades*)

Type: **exit** to leave the script session

**Turn in your finished files. No Zip Files please.**

Go to Canvas and turn in:

1. makefile
2. class\_stats.c
3. lab5.h
4. lab5.c
5. your script session (StudentName\_lab5.txt).

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Helpful slides:**

**Slide 13:**

*/\* Second pass at a makefile: \*/*

*/\* Look at its contents. We have no* ***p2.h*** *but it is included in light italics*

*to show where it would be placed. \*/*

*>****cat makefile***

# Your Name

power2: power2.o compute.o p2.h

gcc power2.o compute.o -o power2 -lm

power2.o: power2.c p2.h

gcc -c power2.c

compute.o: compute.c p2.h

gcc -c compute.c

/\* Run make using our new makefile \*/

[bielr@athena ~/csc60]68> **make**

gcc -c power2.c

gcc -c compute.c

gcc power2.o compute.o -o power2 -lm

[bielr@athena ~/csc60]69>

**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 (That is a capital I.)

$ = 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.