CSE 3430 AU 21 LAB 1

Due: Monday, November 1, 2021 by 11:30 p.m. on Carmen

Objectives:

· Pointers

· Statically allocated arrays (Part 1)

· Dynamic memory allocation (Dynamically allocated arrays) (Part 2)

· Deallocation of dynamically allocated memory (Part 2)

· Switch-case statements

· Functions: pass-by-reference, pass-by-value, parameters, return values and reusability

REMINDERS and GRADING CRITERIA:

- This is an individual lab. You should make a lab1 folder/directory in your cse3430 folder/directory on stdlinux (assuming you followed the instructions for the earlier homework assignment); if you did not make a cse3430 directory before, you should do that now). After logging in to stdlinux using FastX v3, run these commands (assuming you created a cse3430 directory earlier; otherwise, run the first command below also):

  ```
  $mkdir cse3430     [Only necessary if you did not make this directory before]
  $cd cse3430
  $mkdir lab1
  $cd lab1
  ```

- This lab has two parts. Please see the description below.

- **VERY IMPORTANT:** I understand that the lab description seems long, but it is meant to be very clear. Please read and follow the instructions carefully. When software is written in the outside world, written specifications are always used, so getting used to following written specifications is a very useful (and marketable!) skill! By doing this lab, you will acquire some useful knowledge, and a number of skills which are both useful and marketable.

- Grading:

  Comments and code formatting: 10%

  Part 1 output: 50%

  Part 2 output: 40%

- **Note that the lab is due at 11:30 pm.** You should aim to always hand an assignment in on time or early. If you are late (even by a minute – or heaven forbid, less than a minute late), you will receive 75% of your earned points for the designated grade as long as the assignment is submitted by 11:30

pm the following day, based on the due date given above. If you are more than 24 hours late, you will receive a zero for the assignment and your assignment will not be graded at all.

- You have 13 days to do the lab, which is plenty of time if you start early and ask questions early if you need help. Do not wait more than a day or two to start the lab! Part 1 involves more code, but the code is easier to write. You should be finished with Part 1 in 6 to 7 days, at most. That leaves 6 to 7 days to do Part 2, which involves less code, but the code is more challenging to write. Asking questions about Part 1 4 or 5 days before the lab is due means you are behind, and it may be difficult to finish on time!

- Any lab submitted that does not compile – without errors - and run **WILL RECEIVE AN AUTOMATIC GRADE OF ZERO**. If your code generates warnings, you will lose credit for each warning generated, depending on the warning. No exceptions will be made for this rule - to achieve even a single point on a lab, your code must minimally build (compile to an executable without errors) on stdlinux and execute on stdlinux without crashing (that also means no segmentation faults, which is an example of "crashing"), using the following command:

  For Part 1: $gcc -ansi -pedantic lab1p1.c -o lab1p1

  For Part 2: $gcc -ansi -pedantic lab1p2.c -o lab1p2

NOTE VERY CAREFULLY: It is YOUR RESPONSIBILITY to make sure that your code compiles (using the compilation command shown above) before submitting it. ***If you modify your code in any way*** after it compiles successfully, **you must compile again after making any changes to make sure there are no errors.** We have had students in the past who added comments or made what they thought was a "minor" change, but did not recompile and retest before submitting. **This can result in disaster, and must not be done!** This is often the result of waiting too late to start and/or not working steadily enough on the lab assignment. Be sure you do not do things this way; be careful and be thorough! Although the requirements may seem harsh to some students, your manager in a work environment will have requirements at least this strict; if they do not, the business loses customers or loses money or closes, and not only does your manager lose their job, everyone else may as well.

- You should not add features to your program that are not required; this makes it more difficult for the grader to test whether the program meets the requirements, and that is what your score depends on.
- To create a C source code file, you can use any of the text editors available on stdlinux. I recommend you use either emacs or gedit. Here is how to create a source file for Part 1 with gedit:

  $gedit lab1p1.c

  Or if you want to use emacs:

  $emacs lab1p1.c

  The first time you use one of the above commands, the file will be created by the operating system, and the file will be empty. You can type your source code into the file and save it. After the first time you run the command, save the file, and close it, every time you run the command after that will just reopen the file that was already created, and you will be able to edit it and resave it.

Emacs has some nice features which many students like. **PLEASE NOTE:** *When you save files created with gedit, you will get an error related to saving metadata for the file from stdlinux, but you can ignore it.* This will not usually cause any problems.

LAB DESCRIPTION

This lab has two parts, Part 1 and Part 2. For Part 1, we will write code to do calculations on data in a single statically allocated array (as described in slide set B-5), and we will read input for the program from a file using redirection of the input as described in class slide set B-3 on Input and Output in C. In Part 2, the data will consist of a data set of an *unknown* size, so we will need to use dynamic allocation of memory for the array (as described in slide set B-6). For Part 2, again, we will read input for the program from a file using redirection of the input as described in slide set B-3. Much of the code that you write for Part 1, you will be able to reuse for Part 2, but you will also need to write some new code for Part 2 (see the description below). See the instructions below on how to make a copy of your source code file for Part 1 to a source code for Part 2, so that you can then modify the code for Part 2.

Part 1.

DATA SET CALCULATOR **Mandatory filename: lab1p1.c**

PROBLEM:
The user of your program will use it to do some elementary calculations for a single floating point data set of size 8; the user will enter the data for the data set as input to the program (we will read it from a file using redirection of input; see below). Since the number of data sets is known (one) and the size of the data set is also known (8), you can use a static array to store the values. As we have seen in lecture, a static array in C to handle this problem can be declared as follows:

```
float dataSet[8];
int dataSetSize = 8;
```

You should declare this array in main. You should also declare a variable to hold the size of the array (that is, the number of elements in the array), as shown above. When you call any function in the program that will need to access the array or that will call another function which needs to access the array, you will have to pass both dataSet and dataSetSize to the function as parameters.

- In writing your code, you should remember the principle of reusability. In general, each of your functions should do ***a single thing***. If there are several tasks clearly relating to doing a single thing, they can be put into the same function, but if you have any doubt, separate the tasks into different

functions. By writing code in this way, functions are more reusable, and this is a very important property. See the class slide set B-7 C Software Design Principles.

- Related to reusability is the principle that ***in main, only variable declarations and calls to other functions should appear. THE main FUNCTION SHOULD HAVE NO CODE OTHER THAN THIS.***

- You should first call a function to print a single prompt to the user to enter the data for the data set. Your program needs to read the user input and store the float values in the data set in the array that you declared in main, as shown above. The user will enter all 8 of the floating point values on a single line (but remember that scanf will skip white space characters when you call it to read numeric values such as floats). If you do not completely understand this description jump to the bottom of this file and check out the example data for Part 1. You will be given a file on Carmen called lab1p1in to test and debug your program code, so after writing and compiling your code as shown above, you should run it as follows using redirection:

```
$lab1p1 < lab1p1in
```

- After reading the values in the data set into the array, you should call another function which should **repeatedly do the following thing (put this in a loop in a function):**

Prompt the user to choose one of the following options for a calculation to perform on the data set (ask the user to enter one of the six numbers, followed by enter):

1) Find the minimum value.
2) Find the maximum value.
3) Calculate the sum of all the values.
4) Calculate the average of all the values.
5) Print the values in the data set.
6) Exit the program.

Your code should print a blank line at the end of the menu (to separate it from the following output). You can put code to print this menu of choices in the same function with a function that has a switch-case statement, and also put a call to scanf before the switch-case to get the user's choice of option. After the user selects one of the six options, your program should use a switch-case statement to call a function to perform the necessary calculation, or terminate the program. The calculation function called should not also be used to print the result; the result should be printed out by a call to printf which is not in the calculation function (you can do this as part of the switch-case code), except for operation 5, which prints the values in the data set (in other words, for operation 5, the values in the array should be printed by the function which is called to print the array). The program should output the result with an appropriate message which specifies the calculation which was performed, for example:

```
The maximum value in the data set is: 569.45
```

The results for options 1, 2, 3, and 4 should be printed out as floating point values with 2 digits of precision, followed by a blank line to separate the output from following output, and the result for option 5 should be to output a label, such as Data set: on one line, followed by the values in the data set in the

order in which they were input, each on a separate line starting on the line after the label, with two digits of precision for each value. Also print a blank line after all of the values, to separate the output for this operation from any following output.

O After your program outputs the result of the operation, **it should prompt the user again to select an option for a calculation**, until the user selects option 6 to exit the program. Therefore, the function with the switch-case statement which gets the user's choice of calculation should put the switch-case statement inside a loop, which will terminate when the user chooses option 6. Also, when the user chooses option 6 to exit, the function should return to main (but no return statement should be used), which should return 0, and the program will terminate.

CONSTRAINTS:

· Program structure should be:
  -Any necessary #include directives for library header files (stdio.h and stdlib.h)
  -Declarations of all functions in the program (except main)
  -Definition of main
  -Definitions of all other functions (the order does not matter)

·Requirements for main:
  -main should only do the following:
    -Declare variables that are needed by other functions in the program. You will need the following variables in main:
      float dataSet[8];
      int dataSetSize = 8;
    -Call other functions
    -NOTE: Reading of input, getting user choice of calculation to perform, and calculations should NOT be done in main; all of these should be done in functions separate from main.

· You can use a statically declared array for Part 1, as explained above, because the number of arrays is known, and the size of the array is known when the code is written.

· Remember that C programs should exhibit the Unix/Linux feature of reusability of functions; each function should do only one well-defined thing (or perhaps two things that are so closely related that they would ***always*** be done together).

· Use **a separate function** to do each type of calculation (some of these functions might call other ones for example, the code for option 4 might call the function for option 3).

· Use a function to get the input from the user about the float values in the data set.

· Also use a separate function to get the user's choice of the calculation to perform. Use a different function for each of the 5 calculation options, other than option 6, which just returns to main.

· Be sure to document what each function does. Documentation is a necessary part of software for organizations. You should put a brief comment before each function explaining what the function does (not how it does it), and a comment for each block of code documenting what the code in the block does (not how it does it).

· Even though you will have a number of functions, all code must be in a single file named lab1p1.c

· Your program should be able to work if it receives input by redirection of stdin to a file (use the test input file provided to verify this, and when testing your program). The grader will grade your lab using redirection of input, so you should make sure your lab works when it is executed this way.

· Be sure to follow the software design principles in C identified in the class slide set B-7 C Software Design Principles posted on Carmen.

- You should pay attention to ***the order*** in which you write code for the functions in the program. This is a critical part of writing, testing, and debugging software. We always want to write and test functions on which other functions depend before writing and testing the functions which depend on them. For the program for Part 1, the function which reads input from the input file into the array should be written first, along with the function to print out the elements of the array (see the description of that function below). DO NOT write the code for any other functions before you write, test, and debug these first two functions. For other functions, until you are ready to write them, you can use "stub" functions. This is always done by writers of C code in the outside world. A stub function is a function definition with a return type and parameter names and types, but a code block which is empty – with no statements. Here is an example:

```
int sumArray (int *array, int size) {
   /* no code, because it is a stub function */
}
```

If this function is called in the program, it will immediately return, and it will DO NOTHING. Thus, any calls to the stub function will not interfere with testing and debugging any other functions that this sumArray function depends on. Of course, as mentioned above, you also need to have a declaration of this function at the top of your code file after your #include directives, as mentioned above under "Constraints" related to program structure.

Part 2.

DATA SET CALCULATOR - MODIFIED **Mandatory filename: lab1p2.c**

IMPORTANT: DO NOT work on Part 2 until you have completed your code for Part 1 and tested it thoroughly! Failing to follow this advice will make the work MUCH harder, and will make it much more likely that you will not be able to complete Part 2 correctly!

After you finish and test Part 1 thoroughly, you should copy your C source code file for Part 1 to a source file for Part 2, using the following command in your cse3430 lab1 directory on stdlinux:

```
$cp lab1p1.c lab1p2.c
```

This will copy the source code file lab1p1.c to a new file named lab1p2.c, but you will still have lab1p1.c in your files (it will not be deleted, only copied). The code for Part 1 will work correctly for Part 2, except for the modifications described below.

ALSO IMPORTANT: Do not start writing code for Part 2 till you have watched the Zoom video on Part 2, called Lab 1 Part 2, which is linked in the Zoom video links.pdf file on Carmen.

MODIFIED PROBLEM:
The user of your program will use it to do some elementary floating point calculations for a single data set of an UNKNOWN SIZE. The data set consists of a list of floating point values, but the

number of values in the data set will also be specified by the user in the input to the program, so you do not know in advance how many values there will be in the data set. THEREFORE, YOU CANNOT USE A STATIC ARRAY, as you did in Part 1, to store these values. As we will see in lecture, a static array in C is one declared as follows:

```
int array[8];
```

This kind of array can only be used in C when the length or size of the array, 8 in this example, is known when writing the code. ***If the size of the array is not known while writing the code, the kind of array declared above, a static array, CANNOT be used***; instead, A DYNAMIC ARRAY must be used, and the space to store the elements of the array must be allocated at run time (after the program starts running). This must be done by calling a library function, malloc or calloc, to allocate the space. These functions and their use are covered in the class slides (B-6 C Pointers Part 2.pptx). The variables we will declare instead in main for the modified problem are these:

```
float *dataSetPtr;
int dataSetSize;
```

Be sure to watch the Zoom video on Lab 1 Part 2 to understand why these variables are declared this way in main.

- In writing your code, you should remember the principle of reusability. In general, each of your functions should do a single thing. If there are several tasks clearly relating to doing a single thing, they can be put into the same function, but if you have any doubt, separate the tasks into different functions. By writing code in this way, functions are more reusable, and this is a very important property. See the class slide set B-7 C Software Design Principles.

- Related to reusability is the principle that **in main, only variable declarations and calls to other functions should appear. The main function should have no code other than this.**

- First, you should prompt the user to enter the size of the data set. The user will enter an integer greater than or equal to 1 to indicate the size of the data set. Your program should have **a separate function** to get the size of the data set (that is, you cannot put code to do this in main; you should call the function you write to do this from main). This function must be declared to return void (it cannot return an int, or any other type of value). Remember the principle that the main function in a C program should only have declarations of variables and calls to other functions; no other code should appear in main (no loops, no if or if-else or else-if, no switch-case, etc.).

- After calling the function to get the size of the data set, your program should call a separate function to dynamically allocate an array with the right amount of memory to hold the elements of the array. This function also must return void (it cannot return a pointer or any other type of value).

- You should then call a function to print a single prompt for the user to enter values for the data set; you should be able to reuse the same function you wrote for Part 1 to get the float input for the data set. You can assume that the user will enter the input as described in Part 1, so you do not need to do error checking for this.

NOTICE HOW WE ARE REUSING CODE WE WROTE BEFORE FOR PART 1 HERE: Once the user enters the size of the data set and malloc or calloc is called to allocate space for the float data set, the function which you wrote for Part 1 to read the float data into the float array can be called to read the float values from input and store them in the data set. You do not need to, and YOU SHOULD NOT, write code to do this again; instead, reuse the function you wrote to do this before by calling it!

- After getting the values in the data set from the user input, you should call another function which should **repeatedly do the following two things (put this in a loop in a function):**

Prompt the user to choose one of the following options for a calculation to perform on the data set chosen by the user (ask the user to enter one of the six options, followed by enter):

1) Find the minimum value.
2) Find the maximum value.
3) Calculate the sum of all the values.
4) Calculate the average of all the values.
5) Print the values in the data set.
6) Exit the program.

You can reuse the code you wrote for Part 1 for this function. Each of the calculation functions should work the same way as for Part 1 (so you can and should reuse the calculation functions you wrote for Part 1), for example:

The maximum value in data set 1 is: 569.45

The results for options 1, 2, 3, and 4 should be printed out as floating point values with 2 digits of precision, followed by a blank line to separate the output from following output, and the result for option 5 should be to output a label, such as Data set: on one line, followed by the values in the data set in the order in which they were input, each on a separate line starting on the line after the label, with two digits of precision for each value. Also print a blank line after all of the values, to separate the output for this operation from any following output.

- After your program outputs the result of the operation, **it should prompt the user again to select another option for a calculation, till the user selects operation 6, which should cause the function to call a function to free the dynamically allocated memory for the array (or, if you prefer, the function can return to main, and main can call a function to free the dynamically allocated memory for the array).**

- The calculation functions for Part 1 should all be reused.

- Also, for Part 2, since memory was dynamically allocated, you need to write a function to deallocate, or free, this memory before the program terminates (see the description above). When the user chooses option 6 to exit, this function to free all dynamically allocated memory should be called (or the program can return to main and the function can be called from there).

CONSTRAINTS:

· Program structure should be:

-Any necessary #include directives for library header files (stdio.h and stdlib.h)

-Declarations of all functions in the program (except main)

-Definition of main

-Definitions of all other functions (the order does not matter) ·Requirements for main:

-main should only do the following:

-Declare variables that are needed by other functions in the program. You will need the following variables in main:

```
float *dataSetPtr = NULL; /*pointer to data set */
int dataSetSize = 0; /* variable to hold size of data set */
```

-Call other functions

-NOTE: Reading of input, dynamic allocation of memory, getting user choice of data set and calculation to perform, calculations, and freeing of dynamically allocated memory should NOT be done in main; all of these should be done in functions separate from main

· You cannot use a statically declared array for Part 2, as explained above, because the size of the array is unknown when the code is written.

· Your code should work correctly for a data set of any size (including size 1), (including size just 1 and up to the limits of available memory, of course), and these numbers are not known in advance. If your code works for the test data shown below, you should be fine, but explicitly test data with a data set of size 1 (we may do this in grading the lab).

· Remember that C programs should exhibit the Unix/Linux feature of reusability of functions; each function should do only one well-defined thing (or perhaps two things that are so closely related that they would always be done together).

· Use **a separate function** to do each type of calculation (some of these functions might call other ones for example, the code for option 4 might call the function for option 3).

· Use a function to get the input from the user about the size of the data set.

· Use a separate function to allocate space for the data set, and a separate function to read in the values in the data set.

· Also use a separate function to get the user's choice of the calculation to perform. Use a different function for each of the 6 calculation options, including a separate function to deallocate all dynamically allocated memory, as described above.

· Be sure to document what each function does. Documentation is a necessary part of software for organizations. You should put a brief comment before each function explaining what the function does (not how it does it), and a comment for each block of code documenting what the code in the block does (not how it does it).

· Even though you will have a number of functions, all code must be in a single file named lab2.c

· Your program should be able to work whether it receives input from the command line (e.g. keyboard) or via redirected (stdin) input from a file. Note: Since input from a file is being accomplished via redirection of stdin from the command line, no changes to your code should be necessary. Testing it both ways would be a great way to verify this.

· Be sure to follow the Software Design Principles in C identified in the slide set (slide set B-7) posted on Carmen.

GUIDANCE ON DOING THE LAB

-Make sure you are comfortable with the concepts in the two slides set on pointers (B-5 C Pointers Part 1 before writing Part 1, and B-6 C Pointers Part 2, before writing Part 2 of the lab) before you start writing code for each part of the lab! Also, before you start Part 2, be sure to watch the Zoom video on Lab 1 Part 2 linked in the Zoom video links.pdf on Carmen.

LAB SUBMISSION

LAB SUBMISSION

You should submit the files for your lab assignment on Carmen, as follows:

- First, make sure that you are in the stdlinux directory where your Lab 1 files are.

- To zip your files into a folder, use:

    $zip lab1.zip lab1p1.c lab1p2.c

- Run firefox from stdlinux:

    $firefox https://carmen.osu.edu/#

- Log in to Carmen using your name.number and password, and select this course in Carmen.

- Choose the Lab 1 assignment on Carmen, and submit your lab1.zip file.

Be sure to submit the following files (see above): lab2.zip

You should zip the folder with your files before submitting it.

Do not submit executable versions of your programs, lab1p1 or lab1p2 (WARNING: YOU WILL GET NO CREDIT); only submit C source code files! Be sure you use the command shown above to create your lab1.zip file with your C source code files.

See below for a sample data set inputs to the programs for Part 1 and Part 2. You can download these from Carmen into your cse3430/lab1 directory!

Sample Data Set Input for Part 1: provided on Carmen as the file **lab1p1in** (This includes user responses to the prompts to select an operation to perform)

34.5 569.45 335.2 193.4 74.39 122.45 413.2 89.32
1
2
3
4
5
6

Sample Data Set Input for Part 2: provided on Carmen as the file lab1p2in (This includes user responses to the prompts to select an operation to perform)

NOTE: 1st line is number of data sets; 2nd through 7th lines are size of data set followed by float values for data set; remaining lines are choice of data set followed by choice of calculation to perform.

```
8
34.5 569.45 335.2 193.4 74.39 122.45 413.2 89.32
1
2
3
4
5
6
```