This fifth programming project will give you experience writing programs that involve for loops and arrays. This programming project will use the same development process as in Programming Projects #2, #3 and #4, and will be subject to the same rules and grading criteria, along with some new criteria.

This specification will be less detailed than for previous programming projects. YOU ARE EXPECTED TO KNOW HOW TO PERFORM BASIC TASKS WITHOUT HAVING TO BE TOLD EXPLICITLY, based on your experience with previous programming projects.

To get full credit on this programming project, you **MUST** use for loops and dynamically allocated arrays appropriately.

**NOTE:** The next programming project (#6) will be BASED ON PP#5, and you will use your program for PP#5 as the starting point.

**IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!**

For this project, you are **ABSOLUTELY FORBIDDEN** to have any user-defined functions. (We’ll use those in PP#6.)

**I. PROJECT DESCRIPTION**

You’ve been hired to develop statistics software. Specifically, your software will calculate various statistics, the same statistics as in the statistics program in PP#3, as well as some new statistics (described below).

In each individual run of your software, you will input two lists of numbers, and these two lists will have the same length, a length that will be input (and idiotproofed) at runtime, just before inputting the lists. For each of the two lists of numbers, you will need to calculate the following statistics: the taxicab norm as in PP#3, the Euclidean norm (also known as the square norm) as in PP#3, and an additional norm known as the *max norm* (see page 5). In addition, for the two lists of numbers together, you will calculate an additional statistic known as the *dot product* (see page 5).

**HOW TO COPY-AND-PASTE IN PUTTY**

In PP#5, you’ll find that using copy-and-paste will be **EXTREMELY** helpful.

Here’s how to copy-and-paste in PuTTY:

1. Using your mouse (or laptop touchpad), position the mouse cursor at the very left of the text that you want to copy.
2. Hold down the left mouse button.
3. To copy, drag the mouse cursor over the text that you want to copy, which will highlight it. The text is now copied (so no need to Ctrl-C or anything).
4. Position the green text cursor where you want to paste.
5. To paste, (single) click the right mouse button.

**NOTE:** This method only works with PuTTY.
II. PROGRAM DESCRIPTION

Write a program to calculate the statistics from input data. The body of the program MUST be broken into THREE subsections, rather than the usual four subsections (there WON’T be a greeting subsection):

1. an input subsection;
2. a calculation subsection;
3. an output subsection.

Because of how data will be input (see below), THERE WON’T BE A GREETING SUBSECTION.

You are ABSOLUTELY FORBIDDEN to have:

- ANY calculations in the input subsection (and the only outputs should be idiotproofing error messages);
- ANY inputs or outputs in the calculation subsection;
- ANY inputs or calculations in the output subsection.

That is, the three subsections MUST BE COMPLETELY SEPARATE, and MUST BE CLEARLY LABELED.

For this project, if blocks can occur in any subsection of the execution section (body) of the program, and the same is true of for loops.
A. ARRAY DECLARATIONS

You \textbf{MUST} use \textbf{DYNAMIC} memory allocation and deallocation for \textbf{ALL} arrays. (See “Array Lesson 2,” slides #26-33.) Any statically allocated arrays will be \textbf{SEVERELY PENALIZED}.

Therefore, \textbf{ALL} arrays \textbf{MUST} be declared as \textbf{POINTERS}. For example:

\begin{verbatim}
float* input_value1 = (float*)NULL;
\end{verbatim}

B. INPUT SUBSECTION

The program will take its input from a data file, rather than from a user typing live at the keyboard (see part \textbf{III}, \textbf{INPUT DATA FILES}, below).

The input data will be in the following format:

1. a single length, which is shared by both of the lists of numbers (for example, if you input a length of 5, then the first list will have length 5 and the second list will also have length 5).
2. for each element in the two lists:
   (a) the value of that element of the first list;
   (b) the value of that element of the second list.

Several such data files will be provided, each representing an individual run. You should determine how to input the data \textbf{BY EXAMINING THE INPUT DATA FILES} (see \textbf{HOW TO FIND AND EXAMINE THE INPUT DATA FILES}, below).

\textbf{IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!}

Because of how the data will be input, \textbf{YOU WON'T PROMPT THE USER FOR THE INPUTS} (see \textbf{HOW THE DATA WILL BE INPUT}, below).

You \textbf{MUST} store the input data in appropriate one-dimensional arrays. You are \textbf{ABSOLUTELY FORBIDDEN} to use multidimensional arrays in PP#5.

C. ALLOCATING ARRAYS

You \textbf{MUST} use \textbf{DYNAMIC} memory allocation and deallocation for \textbf{ALL} arrays. (See “Array Lesson 2,” slides #26-33.) Any statically allocated arrays will be \textbf{SEVERELY PENALIZED}.

Therefore, \textbf{ALL} arrays \textbf{MUST} be declared as \textbf{POINTERS}.

\textbf{IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!}

Note that \textbf{ALL} of the arrays \textbf{MUST} be allocated, at runtime, in the execution section, \textbf{IMMEDIATELY AFTER INPUTTING AND IDIOTPROOFING THE LENGTH OF THE ARRAYS}.

In other words, once you have input and idiotproofed the length of the arrays, you \textbf{MUST IMMEDIATELY} allocate both of the arrays.

After allocating each array, the program \textbf{MUST IMMEDIATELY} check whether the array was allocated successfully, and if not, the program \textbf{MUST} output a suitable, \textbf{UNIQUE} error message and then \textbf{MUST EXIT}.

For details on dynamically allocating and deallocating arrays, see the lecture slide packet “Array Lesson 2,” slides 26-33.
D. IDIOTPROOFING

YOU MUST IDIOTPROOF ANY input that needs idiotproofing, to make sure that it has an appropriate value. YOU are responsible for figuring out all of the possible cases of idiocy that could come up. ALL IDIOTPROOFING MUST BE COMPLETED BEFORE ANY CALCULATIONS ARE PERFORMED; that is, idiotproofing belongs in the input subsection.

Note that, for this programming project, you are ABSOLUTELY FORBIDDEN to use while loops for your idiotproofing; that is, upon detecting idiocy, the program MUST EXIT.

Idiotproofing error messages MUST be clear, complete English sentences that COMPLETELY AND UNAMBIGUOUSLY state the nature of the error. Thus, EACH ERROR MESSAGE MUST BE UNIQUE. For example, an error message might be:

ERROR: You cannot have a list length of -3.

E. CALCULATION SUBSECTION

In the calculation subsection, the program MUST calculate the following values, in the following order:

- for each list of numbers:
  1. their taxicab norm;
  2. their Euclidean (square) norm;
  3. their max norm;

- for the combination of the two lists:
  1. their dot product.

In any for loop in the calculation subsection, you MUST calculate EXACTLY ONE kind of result; that is, you are ABSOLUTELY FORBIDDEN to calculate multiple kinds of results in a single for loop.

For example, the for loop that calculates the taxicab norm CANNOT also calculate the Euclidean (square) norm.

However, within a particular for loop, you may choose to calculate temporary scalar variables representing various subexpressions.
Max Norm and Dot Product

Given a list of \( n \) real numbers
\[ x_1, x_2, \cdots, x_n \]
the max norm\(^1\), which is the \( p \)-norm with \( p \) of infinity, is a real number calculated as the maximum of all the absolute values of the values in the list:
\[
\|X\|_\infty = \max_{i=1}^n |x_i| = \max(|x_1|, |x_2|, \cdots, |x_n|)
\]

The dot product\(^2\) of two lists of the same length is a real number calculated as the sum of the pairwise products of the values of corresponding elements in the two lists:
\[
X \cdot Y = \sum_{i=1}^n x_i \cdot y_i = x_1 \cdot y_1 + x_2 \cdot y_2 + \cdots + x_n \cdot y_n
\]

How to Find the Maximum Absolute Value in an Array

1. Start by assuming that the maximum absolute value in the array is the absolute value of the first element of the array.
2. Loop over all of the elements in the array:
   (a) compare the absolute value of each element in the array to what you currently have for the maximum absolute value found so far;
   (b) if the absolute value of the current element of the array is greater than what you currently have for the maximum absolute value found so far, then the maximum absolute value found so far becomes the absolute value of the current element of the array.

F. OUTPUT SUBSECTION

In the program’s output subsection, you **MUST** output the following:
- the (shared) length of the lists;
- each of the items calculated, in the same order as in the calculation subsection.
Each output **MUST** be accompanied by helpful explanatory text; for example, the OUTPUT might look like this:
The max norm of the first list is 44.75000.

You may output these quantities in any format that you like, as long as the meaning of the quantities is **CLEARLY EXPLAINED** in the outputs. You are welcome to use format descriptors on your placeholders (for example, "%10.5f"), but you **AREN’T** required to use them.

G. DEALLOCATING ARRAYS

At the end of the program, after the output subsection, you **MUST** deallocate each of the arrays that were allocated in the input subsection, using a `free` statement for each, and then nullifying the pointer, like so:
```
free(input_value1);
input_value1 = (float*)NULL;
```
The deallocations **MUST** occur in the **OPPOSITE ORDER** from the allocations; that is, whichever array was allocated first **MUST** be deallocated last, and so on.

For details, see the lecture slide packet “Array Lesson 2,” slides 26-33.

\(^1\)https://en.wikipedia.org/wiki/Matrix_norm#Max_norm
\(^2\)https://en.wikipedia.org/wiki/Dot_product
III. INPUT DATA FILES

A. HOW TO FIND AND EXAMINE THE INPUT DATA FILES

The input files for your runs can be found on ssh.ou.edu in the directory `~neem1883/CS1313pp5`. You can find the names of all of the data files using the `ls` command:

```
ls  ~neem1883/CS1313pp5
```

The directory contains several data files; some are actual data and some are idiotproofing test files. You **MUST** perform the runs in alphabetical order.

Actual (non-idiotproofing) test files have file names beginning with the prefix `actual_`

Idiotproofing test files have file names beginning with the prefix `idiot_`

You **SHOULD CLOSELY EXAMINE** (but not change) the contents of each of the data files using `nano`:

```
nano   ~neem1883/CS1313pp5/actual_2.txt
```

B. HOW THE DATA WILL BE INPUT

For this programming project, **YOU WON’T PROMPT THE USER FOR THE INPUTS**, because there won’t be a user as such. Instead, the inputs will come from a file. To get the inputs from the file, you’ll use a command like this at the Unix prompt:

```
big_statistics < ~neem1883/CS1313pp5/actual_2.txt
```

This use of a file is referred to as *redirecting input*. The less than symbol `<` indicates that the input will come from the file named `actual_2.txt`. In other words, as far as the program is concerned, the file will appear to be a user typing at the keyboard, and the program will accept input from the file exactly as if that input were being typed at the keyboard by a real live user. Thus, you **MUST** write your `scanf` statements exactly as if a user were going to be typing the data at the keyboard, but without the user needing to be prompted.

However, because there isn’t actually a real live user, it isn’t necessary to greet the user nor to prompt for inputs; the data file won’t understand the prompts anyway, so to speak.

Your run commands **MUST** look like this example:

```
big_statistics < ~neem1883/CS1313pp5/idiot_1.txt
```

This means, “run the executable named `big_statistics`, redirecting input from the file named `idiot_1.txt` that’s in the directory named `~neem1883/CS1313pp5`.”
IV. RUNS

Run this program several times, using the several different input files that are available (see below). The runs MUST be in alphabetical order according to the input file names.

The order of the runs in your script file MUST be:

- all actual files, in alphabetical order, followed by
- all idiot files, in alphabetical order.

V. ADDITIONAL GRADING CRITERIA

All grading criteria for Programming Projects #2, #3 and #4 apply.

1. Declaration order:
   In the declaration section, the order of declarations MUST be:
   (a) named constants: float scalars followed by int scalars;
   (b) variables, in the following order:
      i. arrays: float arrays followed by int arrays;
      ii. scalars: float scalars followed by int scalars.

2. Block closes of for statements:
   ALL block closes associated with for statements MUST be followed, on the same line, by a space, a comment open, a space, the keyword for, a space, the counter variable, a space, and a comment close. For example:

   ```c
   for (element = first_element; element < number_of_elements; element++) {
      scanf("%f %f", &input_value1[element], &input_value2[element]);
   } /* for element */
   ```

3. Indenting for statements and their associated block closes:
   For a given for loop, the for statement and its associated block close MUST be indented identically, and this indentation amount MUST be appropriate with respect to their position within the program.

4. Indenting inside for loops:
   For a given for loop, ALL statements INSIDE the for loop MUST be indented FOUR SPACES farther than the for statement and its associated block close. For example:

   ```c
   taxicab_norm1 = initial_sum;
   for (element = first_element; element < number_of_elements; element++) {
      taxicab_norm1 += fabs(input_value1[element]);
   } /* for element */
   ```
5. **Commenting** for loops:
   Each for loop **MUST** be preceded by a comment that describes what the loop as a whole does. For example:
   ```c
   /*
   * Calculate the taxicab norm of first list of input values.
   */
   taxicab_norm1 = initial_sum;
   for (element = first_element; element < number_of_elements; element++) {
       taxicab_norm1 += fabs(input_value1[element]);
   } /* for element */
   ```

6. **Commenting inside** for loops:
   A statement inside a for loop **MUST** be preceded by a comment that describes what the statement does. The comment **MUST** be properly indented, so that the asterisk of the comment lines up with the statement. For example:
   ```c
   for (element = first_element; element < number_of_elements; element++) {
     /*
     * Increase the taxicab norm for the first list
     * by the value of the current element.
     */
     taxicab_norm1 += fabs(input_value1[element]);
   } /* for element */
   ```

VIII. DEBUGGING VIA printf STATEMENTS
The best mechanism for debugging this program is to put in lots of printf statements that show where in the program the run currently is. For example:
```c
for (element = first_element; element < number_of_elements; element++) {
    printf("About to input data for element #%d:\n", element);
    scanf("%f %f", &input_value1[element], &input_value2[element]);
    printf("Done inputting data for element #%d:\n", element);
    printf("  input_value1[%d]=%f, input_value2[%d]=%f\n",
           element, input_value1[element],
           element, input_value2[element]);
} /* for element */
```

**IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT!!!**
Once you’ve completed debugging, you **MUST** delete **ALL** debugging printf statements. **EXTRANEOUS OUTPUTS WILL BE SEVERELY PENALIZED.**
IX. EXTRA CREDIT

You can receive an extra credit bonus of as much as 5% of the total value of PP#5 by doing the following:

1. Attend at least one CS1313 help session for at least 30 minutes, through Wed Nov 8.
2. During the help session that you attend, work on CS1313 assignments (ideally PP#5, but any CS1313 assignment is acceptable). **YOU CANNOT GET EXTRA CREDIT IF YOU DON’T WORK ON CS1313 ASSIGNMENTS DURING THE HELP SESSION.**
3. Before you leave the help session, fill out both halves of the form on the last page of this project specification and have the help session leader (instructor or TA) sign both halves. **THE FORM CANNOT BE SIGNED UNTIL IT IS COMPLETELY FILLED OUT.**
4. Attach the bottom half of the form to your PP#5 script printout, **AFTER** the script itself, and keep the top half for your own records.

**BONUS VALUE NOTICE:** Up through Wed Nov 1, the extra credit bonus will be worth 5% of the total value of PP#5, but from Mon Nov 6 through Wed Nov 8, the extra credit bonus will be worth only 2.5% of the total value of PP#5. That is, **YOU’LL GET TWICE AS MUCH EXTRA CREDIT DURING THE FIRST WEEK AS DURING THE SECOND WEEK.**
CS1313 PROGRAMMING PROJECT #5 BONUS REQUEST FORM

Name  ____________________________________________ Lab ____________
Help Session Date ________________
Help Session Time (Arrive) ________________ Help Session Time (Depart) ________________

Instructor Signature __________________________________________

Keep this copy for your records.

CS1313 PROGRAMMING PROJECT #5 BONUS REQUEST FORM

Name  ____________________________________________ Lab ____________
Help Session Date ________________
Help Session Time (Arrive) ________________ Help Session Time (Depart) ________________

Instructor Signature __________________________________________

Submit this copy.
In your submission, attach this copy **AFTER** your script file printout.
If you put this in the wrong place in your submission, then you **WON’T** get the extra credit.