CSSE 120 – Introduction to Software Development in C

Quiz 3 from Videos: ArrayConcepts ReadScores

Reading: arrayPatterns.pdf arrayGotchas.pdf

Name: _____ **Section:** _____ **Grade:** _____

Answer these questions while viewing the assigned videos. ***Not sure of an answer?*** Ask your instructor to explain ***at the beginning of the next class session***. You can then fill in your answer, still for full credit. (But no fair doing that unless you attempted the question first.)

Videos for this quiz: www.rose-hulman.edu/class/csse/binaries/C-Videos/session3/

Reading for this quiz:

- **Array Gotchas: items 1 through 5, from the above link, file arrayGotchas.pdf.**
- **Array Patterns: items 1 through 9, from the above link, file arrayPatterns.pdf.**

Video 1: Array Concepts (6 minutes, 13 seconds)

1. For this problem, the following elaboration from the video may be helpful:
   - Arrays in C are much like lists in Python. For arrays, however, it is helpful to distinguish two different concepts:
     - The ***capacity*** of the array – how many items the array can hold.
     - The ***size*** of the array – how many items are *currently* stored in the array (with the rest of the capacity of the array storing indeterminate garbage).

     For example, consider the code snippet to the right. There, the ***capacity*** of the array is ***4***, but its ***size*** at the end of the snippet is ***1***.

```
float x[4];
size = 1;
x[0] = 4582;
```

   The video uses ***size*** for both these concepts, but the following question is phrased more carefully, using the above definitions for ***capacity*** and ***size*** when referring to arrays. For *lists*, the two terms can be used interchangeably.

Here are some properties. Some are true for lists (but not arrays), some are true for arrays (but not lists), some are true for both and some are true for neither. For each property, circle the appropriate answer. Warning: Some of these are tricky, and some are NOT addressed by the video; bring your questions to class!

a.	Can contain data of different types.	Arrays	Lists	Both	Neither
b.	Have a fixed initial *capacity* that doesn't change.	Arrays	Lists	Both	Neither
c.	Store their own *size*.	Arrays	Lists	Both	Neither
d.	Store their own *capacity*.	Arrays	Lists	Both	Neither
e.	Automatically increase their capacity as needed.	Arrays	Lists	Both	Neither
f.	Use square brackets, like `scores[3]`, for access.	Arrays	Lists	Both	Neither

g. Can be mutated by statements like:
`scores[7] = 89;`
if their ***size*** is at least 8. Arrays Lists Both Neither

h. Allow insertion between existing elements
even if the array/list is at its full capacity. Arrays Lists Both Neither

i. Allow insertion between existing elements
if the array/list is not yet at its full capacity. Arrays Lists Both Neither

j. Are zero indexed . Arrays Lists Both Neither

k. If the ***size*** of the array/list called *scores* is exactly 4,
this statement is successful: `scores[4] = 99;` Arrays Lists Both Neither

l. Can be printed in their entirety by a single *print*
(for lists) or *printf* (for arrays) Arrays Lists Both Neither

2. Implement (here, on paper) the following function:

```
int count_bigger(float numbers[], int size, float threshold) {

/* Returns the number of numbers in the given array that are bigger
   than the given threshold, where the array has the given size.
*/
```

Video 2: Read Scores (a worked example of looping through an array) (5 minutes, 36 seconds)

Note: You can check out the project mentioned in this video as **Session27_C1_Arrays_ReadScores**

3. **True** or **False** (circle one): The pattern for looping through an array from beginning to end, doing something with each item in the array one by one as you go through the loop, is much the same in C with arrays as it is in Python with lists.

Reading: Array Patterns, from:

www.rose-hulman.edu/class/csse/binaries/C-Videos/session3/arrayPatterns.pdf

Read Patterns 1 through 9 in the above document, in enough detail to understand the pattern but without bogging down. Skim the remaining patterns very briefly for now.

Apply the ***Finding a Given Element*** pattern in the above document to implement the following function (here, on paper).

```
int has_negative(float numbers[], int size) {

/* Returns TRUE if the given array contains an element that is
   negative, else returns FALSE, where the array has the given size
   and where TRUE and FALSE have been defined to be 1 and 0,
   respectively.
*/
```

Reading: Array Gotcha's, from:

www.rose-hulman.edu/class/csse/binaries/C-Videos/session3/arrayGotchas.pdf

Read items 1 through 5 in the above document.

4. For each of the following, indicate whether it is True or False:

   When you access an array with a subscript whose value is outside of the bounds of the array:

   - ***The program may run but with a garbage value*** (often 0) for the array element. **True** **False**
   - ***The program may crash*** (because the space outside the array that was accessed is system memory). **True** **False**
   - ***Another variable in the program may change its value mysteriously!*** **True** **False**
   - ***Behavior may oscillate among the above, seemingly at random!*** **True** **False**

If you have other questions about these videos /reading, ask them at your next class session!