# **CS241 Homework 2, Winter 20**

Arrays and Pointers

**Due:** Monday of Week 8, Feb 24.

***Please answer all the questions and submit your work through Blackboard. You can also print out a hard copy and turn in your hard copy in class.***

1. (10 points) Please indicate whether the following statements with a question mark are allowed or not allowed in a program of ANSI C. If not allowed, please explain why.
   1. double a[100];

a++; ?

**Yes, this is allowed. It increments the array, but not the values in the array**

* 1. int a[100];

int \* p = a;

p++; ?

**Yes this is allowed. It increments the address that the array is stored at.**

* 1. #define N 100

int a[N]; ?

**Yes this is allowed, it creates an array of size 100**

* 1. float a[100];

float \*f = a[0]; ?

**No, this is not allowed. You can’t compare a float array to a float pointer.**

* 1. int a[100];

int \* p = &a;

**This is not allowed, because we are trying to initialize an int pointer from an int array pointer.**

1. (10 points) As we know that when you pass an array with the array name to a function, the address of this array is copied over to the function. Any changes to this array inside the function will be reflected in the calling environment. This is recognized as “call by reference” in C. For this question, however, we would like to do the opposite. You are asked to write a function that takes an array of integers and the size of this array (an integer) as its arguments. Inside this function, each element of this array is increased by 10 (a[i] += 10;). Finally, the modified array is returned to the calling function and the data can be used by the calling function. However, the changes made inside this function **cannot** affect the array (that was passed over to the function) in the calling function. I would call this function as “pass array by value”. Attach your code for the answer to this question.

**Int AddTen(int array[], int size)**

**{**

**Int temp[] = malloc(sizeof(int));**

**For(int i=0; i<size; i++)**

**{**

**Temp[i] = array[i] + 10;**

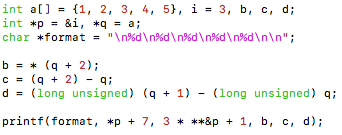
**}**

**Return temp;**

**}**

**I’d like to say that as far as I’ve learned in C, I’ve never “returned an array”. I always return the memory that the array is stored at. And I thought that you CAN’T return an array in C.**

1. (10 points) What will get printed out when the following code is executed? Please explain your answer. Answers without correct explanations will not receive full credits.



**10**

1. “format” is the string that we’re printing out, and we’re just going to tack on integers to it.
2. \*p points to what’s inside of i. adding 7 to what’s inside of I gives us 10!
3. 3 \* \*\*&p can be reduced to 3 \* \*p—the \*& cancel eachother out. So now we’re taking 3 times whatever’s in the pointer of p, which is 3. 3\*3 is 9, plus 1 = 10
4. \*(q+2) moves us 2 indices into the array, giving us 3
5. (q+2)-q literally just resolves to that -> 2
6. I don’t know this one. And I’m not going to even attempt a fake getaround answer :D

**10**

**3**

**2**

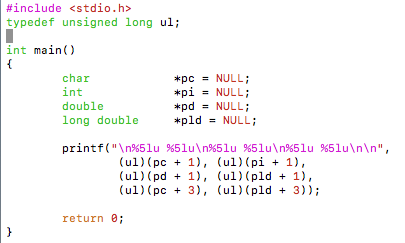
**4**

1. (10 points) A general guideline to define an array is that if you know the size of the array at compile time, define it as an array of fixed size; otherwise, define a pointer, allocate memory for the array at run time when the size of the array is known, and make the pointer points at the beginning of the allocated memory. The latter is recognized as run time allocation. Please discuss the advantages and disadvantages of run time allocation.

**The advantages of run time allocation are pretty substantial. This allows you to accept user input, or accept input from files/outside sources while dynamically allocating memory. You’ve now opened yourself up to a whole plethora of different projects!**

**The disadvantages of run time allocation, however, is that it takes more time to complete. Also, this memory needs to be freed by the developer when it’s done, which results in more bugs that are difficult to find.**

1. (12 points) The value of a pointer is a memory location, which can be interpreted as an integer as seen in a previous question. If you want to see addresses printed as decimal numbers rather than hexadecimals, it is usually safe to cast address as an **unsigned long** and use the **%lu** format descriptor. What will get printed out when the following code is executed? Please explain your answer. Answer without correct explanation will not receive full credits for this question.



1. **4**
2. This pointer holds a character, 1 byte, so NULL + 1 \* 1 = 1
3. This pointer holds an int, 4 bytes, so NULL + 4 \* 1=4
4. This pointer holds a double, so NULL + 8 \* 1 = 8
5. This pointer holds a long double, so NULL + 16 \* 1 = 16
6. This is the same as the first, but adds 3 characters to NULL instead of just the 1 -> NULL + 1 \* 3 = 3
7. This is the same as the fourth, but adds 3 long doubles to NULL instead of just the 1 -> NULL + 16 \* 3 = 48

**8 16**

**3 48**

1. (10 points) What will get printed out when the follow piece of code is executed. Explain your answer. Answer without correct explanation will not receive full credits for this question.

../../../../../../../../Desktop/Screen%20Shot%202019-02-13%20

**hello!**

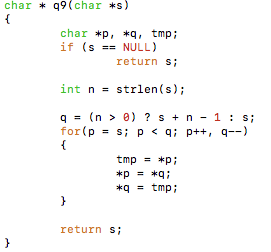
1. Gets the character in the “1” index, which is “h”
2. Gets the pointer to the 0 + 1 index of “def”, which is “e”
3. The code \*”abc” points to the first character, ‘a’. add 11 to this and you get the letter “l”
4. Same as the first one, “1” index, which is “l”
5. Same as the third one, points to character ‘g’, add 8 to get “o”
6. (10 points) Write a function to implement the C [strcmp()](https://www.programiz.com/c-programming/library-function/string.h/strcmp) function that is defined in <string.h> file and modify it as follows: When the two strings have the same length, do exactly the same as the C strcmp() does; when the two string have different lengths, the longer string is always greater than the shorter string. Attach your code for the solution of this question. **Hint:** Check out the definition of strcmp() function from the link: <https://www.programiz.com/c-programming/library-function/string.h/strcmp>.



1. (8 points) Please complete the following table:

|  |  |  |
| --- | --- | --- |
| char \*p[2][3] = {“abc”, “defg”, “hi”, “jklmno”, “pqrstuvw”, “xyz”; | | |
| Expression | Equivalent Expression | Value |
| \*\*\*p | p[0][0][0] | ‘a’ |
| \*\*p[1] | p[0][0][1] | ‘b’ |
| \*\*(p[1] + 2) | p[1][0][3] | ‘i’ |
| \*(\*(p + 1) + 1)[7] | error | ? |
| (\*(\*(p + 1) + 1))[7] | p[2][3][7] | W |
| \*(p[1][2] + 2) | p[1][2][2] | ‘i’ |

1. (10 points) What does the code shown below do?



**I feel like this is a trick question. We pass in a string ‘s’, and then do a bunch of stuff to strings ‘p’ and ‘q’ that appear to be swapping them, and then we just pass back string ‘s’…but WITHOUT doing anything to string ‘s’. So really all this particular code does is create a memory issue because we aren’t allocating memory for p and q, and then passes back the exact same string we initially passed TO it…**

1. (10 points) Study each of the following pieces of code. If nothing wrong in the code, simply answer it and indicate what will get printed out when this piece of code is executed. If something wrong, please point it out and explain what is wrong (again, answer without correct explanation won’t receive full credit).
   1. char s[] = “abc”;

char \* p = s;

\*p = ‘A’;

print(“%s\n”, s);

**Abc**

* 1. char str[] = “abc”;

const char \*p = str;

\*(p + 1) = ‘B’;

printf(“%s\n”, p);

**We told p to be a constant character string, then tried to modify it. So this is incorrect.**

* 1. int x = 8, n = 6;

const int \* const p = &x;

x = n;

printf(“%d\n”, \*p);

**6**

* 1. char s[] = “abc”;

const char \*p = s;

\*s = ‘A’;

printf(“%s\n”, p);

**Abc**

* 1. char \* p = “abc”, \* q = “abc”;

printf(“%d\n”, p == q);

**1**