CSE31: Lab #4 – Memory in C (part 2)

Overview

In this lab, we will continue to investigate some of the interesting facts about how C interacts with memory. You will continue to practice the use of pointers to access memory.

You can refer to chapter 5 and 6 of K&R for references on pointers.

Getting started

Before we begin any activities, create a directory (Lab_4) inside the CSE31 directory we created during Lab #1. You will save all your works from this lab here.

(Exercise) How type-casting work with memory

From lecture, we've learned that an array is stored in a contiguous memory space. The address of each element of an array is assigned by the computer (operating system)

TPS (Think-Pair-Share) activity 1 Paired with the classmate sitting next to you and do the following tasks (25 minutes):

1. Open ***memCast.c,*** compile and run the program. What do you expect the program to print? (**%x** in **printf** allows an integer to be printed in **Hex** format).
2. **Before changing the code**, what do you expect the program to print if you print ***four_ints[0]*** again at the end of the program?
3. Insert a print statement to print out ***four_ints[0]*** at the end of the program and verify your answer from (2).
4. Now add a print statement to the program so it will print out ***four_ints[1]***. What does it print? Are you surprised (or lost) by the results?
5. Let's study the code carefully and investigate what happened. No, the memory did not go crazy.
    a. How many array(s) were allocated in this program?

b. Are ***four_ints*** and ***four_c*** pointing to the same location?

c. Verify your answer of (b) by printing out the values of ***four_ints*** and ***four_c***.

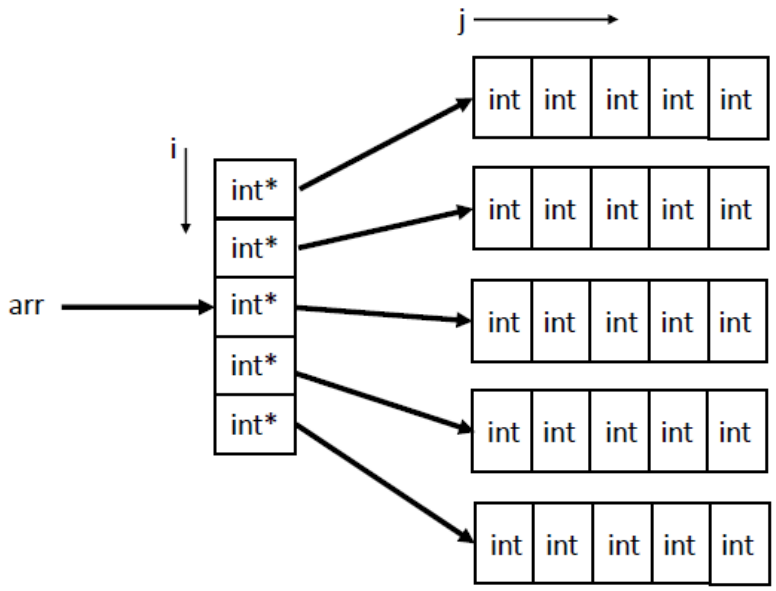
6. Write a loop to print out the addresses (int Dec) and values (in Hex) of all the elements of ***four_ints***. What is the difference in addresses between two consecutive elements? Discuss what this difference means.
7. Use a piece of paper to draw the layout of the array **horizontally** so that the smallest address begins from the **RIGHT-hand-side**. Indicate the **address** and **value** of each element based on the results of (6). You can draw it digitally.
8. Now, write a loop to print out the same information of ***four_c*** as you did in (6). What is the difference in addresses between two consecutive elements? Discuss what this difference means.
9. Use the same piece of paper (or file) from (7) to draw a similar structure of ***four_c***.
10. By comparing the layout of the array pointed by the two pointers, what do you observe in terms of how C accesses memory when the index of an array is incremented?
11. Submit your answers from this activity and the completed ***MemCast.c***
12. Your TA will "invite" one of you randomly to share what you have discussed.

(Exercise) 2D arrays with malloc

In C, a 1-dimensional array can be declared as a pointer pointed to a dynamically allocated memory location:

```
int* array = (int*)malloc(n * sizeof(int)); // n is the size of array
```

We can also use a **double pointer (**)** to construct a 2D array. In fact, a 2D array is just a multiple rows of 1D arrays. You can also view it as an array of arrays:



In the example above, each row is an array of *int*. Since each row is an array of *int*, we need an ***int**** pointing to each row. As a result, we need to have an array of *int** to point at different rows of *int*.

TPS (Think-Pair-Share) activity 2 Paired with the same classmate and do the following tasks (25 minutes):

1. Open ***Array2D.c***. This program will create a ***n x n*** array of int. Explain what line #8 does.
2. Since every array must be allocated in the memory before it can be used, we need to allocate the rows one by one. To do this, we need to be able to access the pointers from the first array (pointed by arr). Assuming i is the index of that array, how do we access the i^{th} value of the array?
3. Without using **array notations ([])**, Insert code to complete **allocating all the rows and initialize all contents to be 0**. You code should work with different values for ***n***. **Hint:** if ***j*** is the index of each row, how do you access ***arr[i][j]*** in pointer notation?
4. To verify whether you have created your array correctly, we need a function to print out the array. The function ***printArray*** has been declared. It takes in both the ***array*** to be printed and ***size*** of array. Why do we need to pass the size as an argument?
5. Complete ***printArray*** so it prints out the **content** and **layout** of an array correctly.

6. Now, let's modify the content of the array. Insert code to make the array into a diagonal matrix that looks like this (**again, do not limit the size to 5 only**):

1	0	0	0	0
0	2	0	0	0
0	0	3	0	0
0	0	0	4	0
0	0	0	0	5

7. Call ***printArray*** to print out your new array and verify result.
8. Submit your answers from this activity and the completed ***Array2D.c***.

(Assignment 1, individual) Matrix multiplication

Now we know how 2D arrays work. Let's put them in practical use (you must not use any array notions ([]) in this assignment):

1. Open ***MatrixMult.c***. and define in ***main()*** two ***n x n*** matrices (arrays) using ***malloc***.
2. Implement ***printArray*** function so you can call it to print a 2D array.
3. In ***main()***, call ***printArray*** to print out the 2 arrays you have defined in (1).
4. Implement ***matMult*** so it multiplies the 2 input arrays and return the resulting array. **Pay attention to the input arguments and return type**.
5. In ***main()***, call ***matMult*** to multiply the 2 arrays you have defined in (1).
6. In ***main()***, call ***printArray*** to print out the resulting array you receive in (5).
7. You need to declare any variables that are necessary.

What to submit

When you are done with this lab assignments, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your ***MemCast.c***, ***Array2D.c***, ***MatrixMult.c***, answers to the TPS activities in a text file, and a list of Collaborators.
- Your assignment is closed **7 days after this lab is posted** (at 11:59pm).
- You must demo your submission to your TA **within 14 days** (preferably during next lab so you will have a chance to make correction and re-submit/re-demo.)