**TPS activity 1**

**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).**

**It prints** 2.

**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?**

**It prints** 41414141  **because it is printing four bytes of int.**

**3. Insert a print statement to print out four\_ints[0] at the end of the program and verify your answer from (2).**

**printf("%x\n", four\_ints[0]);** //prints 41414141

**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?**

**It prints out** 2. **I am not surprised, we can see that** four\_ints[0] **moves** 4 bytes**(because of int have four) to next index** four\_ints[1] **where the next** 2 **value was stored from the for loop.**

**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?**

**Two were allocated when we declared:**

int four\_ints[4]; //array of integers

char\* four\_c; //array of characters

**b. Are four\_ints and four\_c pointing to the same location?**

**Yes both** four\_c **and** four\_ints **point to the same location which is** 0x7ffe499a5b00

**c. Verify your answer of (b) by printing out the values of four\_ints and four\_c.**

**printf("%p\n", four\_c);** // = 0x7ffe499a5b00

**printf("%p\n", four\_ints);**// = 0x7ffe499a5b00

**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.**

**for(int i = 0; i < 6; i++){//tps 6 prints all addresses**

**printf("Address: %d\n", four\_ints[i]);**

**printf("Hexadecimal: %x\n", four\_ints[i]);**

**}**

Address: 0x7ffd87be96a0

Hexadecimal: 41414141

Address: 0x7ffd87be96b0

Hexadecimal: 2

Address: 0x7ffd87be96c0

Hexadecimal: 2

Address: 0x7ffd87be96d0

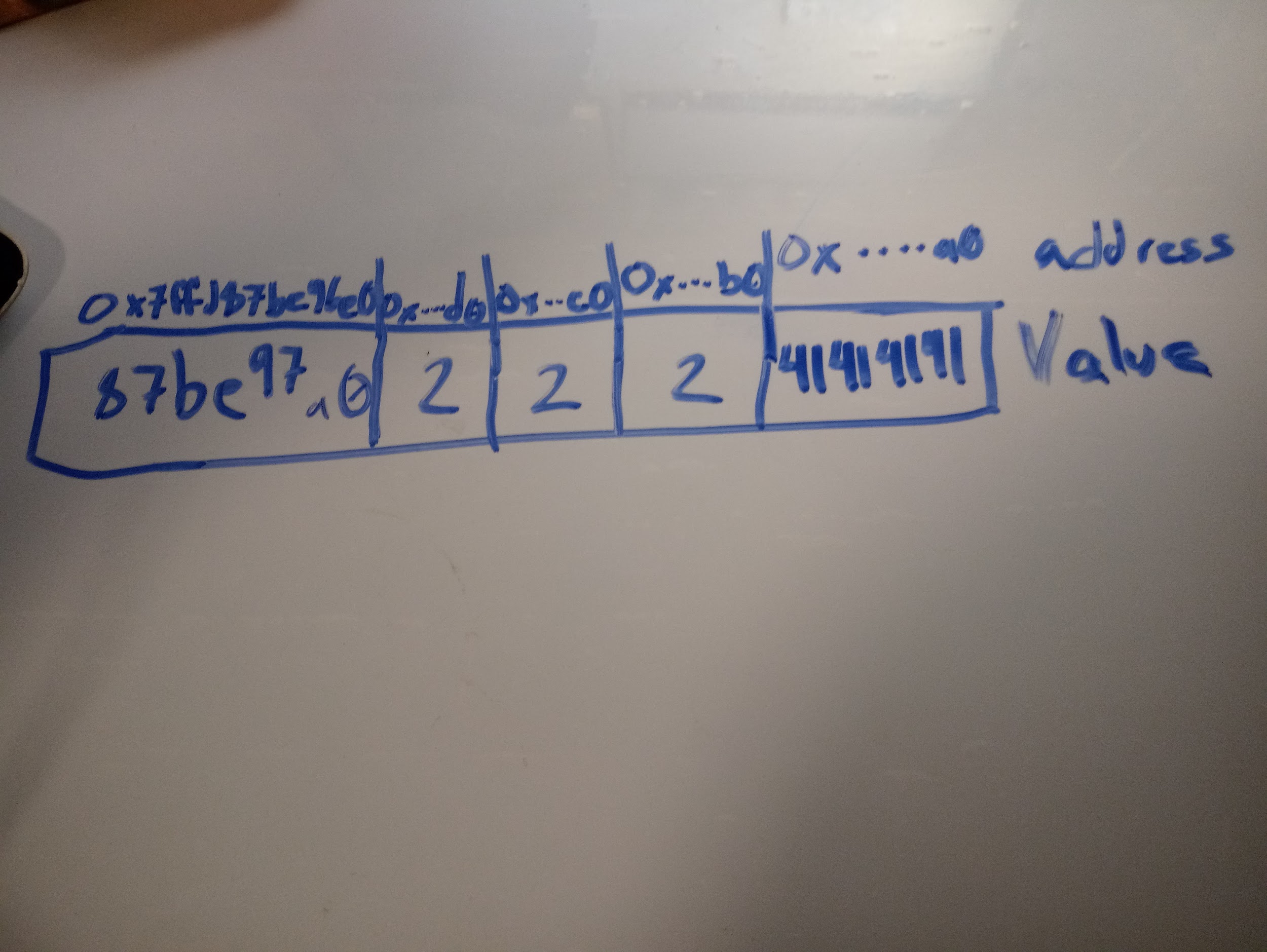
Hexadecimal: 2

Address: 0x7ffd87be96e0

Hexadecimal: 87be97a0

This changes by 4 bytes each time, since it is an integer.

**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.**

for(int i = 0; i < 4; i++){//tps prints all addresses for four\_c

printf("%p\n", &four\_c[i] );

printf("%c\n", four\_c[i]);

}

0x7ffef9661060

A

0x7ffef9661061

A

0x7ffef9661062

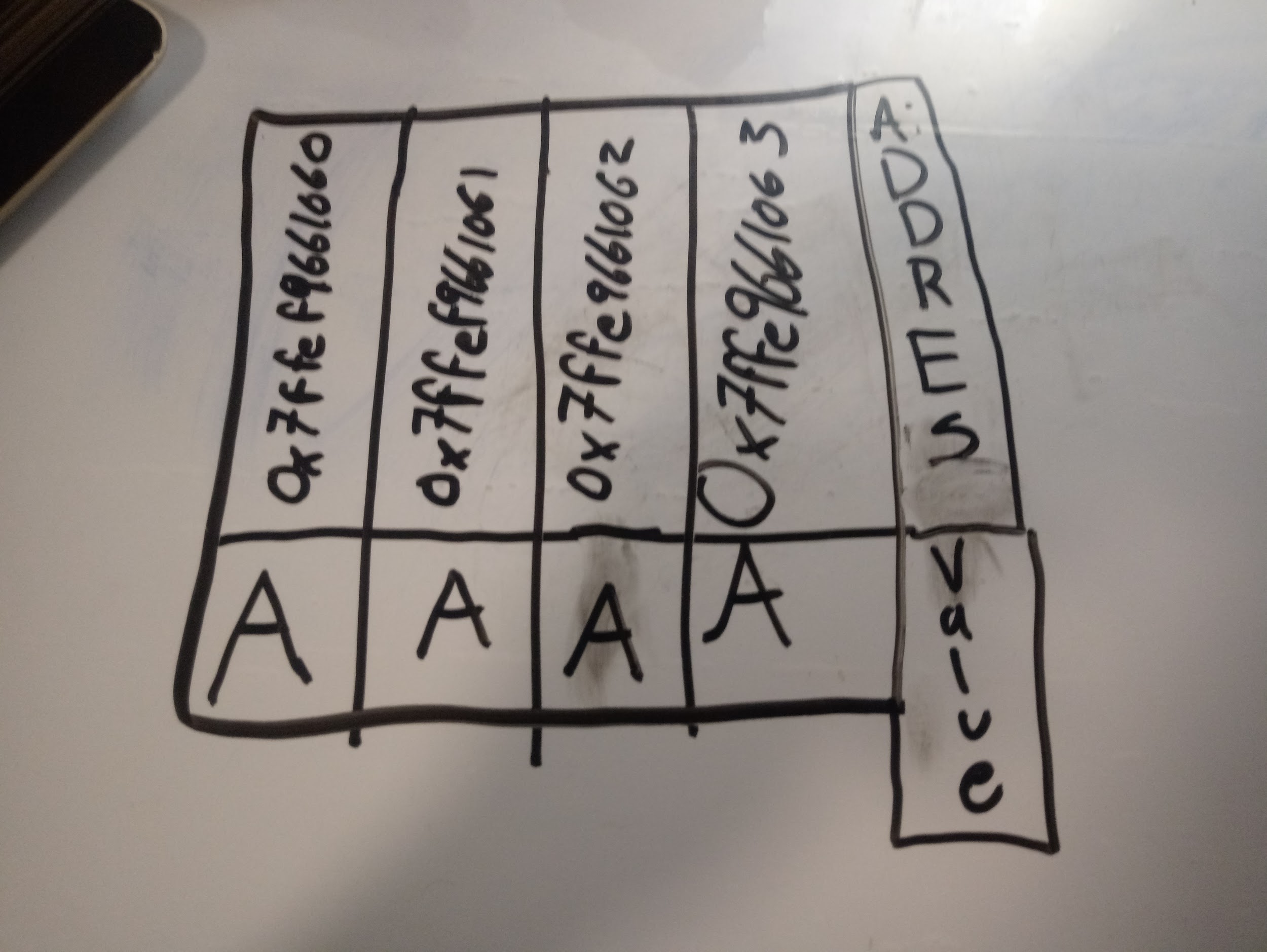
A

0x7ffef9661063

A

The addresses change by 1 bits because it is a char loop

**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?**

**You can specifically access memory in whatever way you instruct your code to, it can be bits, bytes or whatever datay type you want to.**

**TPS (Think-Pair-Share) activity 2**

**1. Open Array2D.c. This program will create a n x n array of int. Explain what line #8 does.**

**It is allocating memory in the heap to store a pointer of a pointer the size of** n **which equals** 5

**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?**

**\*(arr + i)**

**arr[i][j] \*(\*(arrr+1)+a)**

**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?**

**You need a for loop that works with n.**

for(int i = 0; i < n; i++){

\*(arr+i) = (int\*)malloc(n\*sizeof(int\*));//allocate memory

for(int j = 0; j < n; j++){

\*(\*(arr + i) + j) = 0;//column

}//end for loop 2

}//end for loop 1

**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?**

**To print the amount of values in the array; so that it will fully print the entire thing.**

**5. Complete printArray so it prints out the content and layout of an array correctly.**

**void printArray(int \*\* array, int size){**

**int j, i;**

**for( int i = 0; i < size; i++){**

**for( int j = 0; j < size; j++){**

**printf("%d", \*(\*(array + i) + j));//print**

**}//end for loop 2**

**printf("\n");**

**}//end for loop 1**

**}**

**Output:**

**00000**

**00000**

**00000**

**00000**

**00000**

**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):**

**for( int i = 0; i < n; i++){**

**for( int j = 0; j < n; j++){**

**if(j == i)**

**\*(\*(arr + i) + j) = i +1 ;**

**}//end for loop 2**

**printf("\n");// breaks**

**}//end for loop 1**

**7. Call printArray to print out your new array and verify result.**

**10000**

**02000**

**00300**

**00040**

**00005**