Jaime Williams
lab 8

items marked in red were corrected.

2) Use the provided address from #1 as the base address of the array. Based on the code below, create an educated guess that clearly outlines what you believe will happen as each line is executed. In your explanation clearly explain what is happening, don't just give memory addresses or values. If you only provide memory addresses or values you will receive 0 points for this problem. Your guesses will be clearly labeled in the PDF file. You must provide the line of code and then the explanation. NOTE: Where I provide the comment // reset ptr no explanation is required

	Type	Value	Description
Correct guess ptr++;	int *	A13C	increments ptr 4 bytes
Correct guess printf("*ptr %i\n", *ptr); ✓	int	400	Value is from arr[2]
Correct guess printf("ptr %p\n", ptr);	int *	A13C	the second element at the arr
Correct guess *++ptr;	int	600	advances ptr to next arry element
✗ printf("*++ptr %i\n", *ptr);	int	600 800	~~advances~~ ptr then prints
✗ printf("ptr %p\n", ptr);	int*	A140 ~~A144~~	prints pointer address
		800	
✗ *ptr++;	int	1000	advances pointer
✗ printf("*ptr++ %i\n", *ptr);	int	~~800~~	~~Advances ptr~~ then prints it
✗ printf("ptr %p\n", ptr);	int *	~~A14C~~	prints ptr address
		A144	
ptr = arr; // reset ptr			I think this is zero arr is not initialized
// fun with printf repeat last couple of commands		400	
✗ printf("*++ptr %i\n", *++ptr);	int	~~600~~ A13C	adv ptr 2 twice
✗ printf("ptr %p\n", ptr);	int*	~~A140~~	prints ptr address
✗ printf("*ptr++ %i\n", *ptr++);	int	400 ~~600~~	~~advances~~ ptr twice
✗ printf("ptr %p\n", ptr);	int*	~~A148~~	prints ptr address
		A13C	
ptr = arr; // reset ptr			arr still @ zero
			ptr points @ 2nd element in arr
Correct guess *ptr += 1;	int	0+1=1 200	prints what is in arr[1]
✗ printf("*ptr %i\n", *ptr);	int	~~400~~	address of arr[1]
✗ printf("ptr %p\n", ptr);	int*	~~A13C~~ A138	~~adv ptr twice~~ then print
✗ printf("*(ptr+1) %i\n", *(ptr+1));	int	~~A140~~	adds
		401	
✗ ptr = arr; // reset ptr		201	

Jaime Williams
lab 8

	Type	Value	Description
200		400	
✗ *(arr+2) = *ptr+100;	int	~~300~~	arr[2] gets 300
✗ printf("*(arr+2) %i\n", *(arr+2));	int	~~201~~ ~~800~~	~~advanced arr~~ twice prints Int @ that location
Correct Guess ptr = arr + 5;	int*	A14C	sets ptr to arr[5]
✗ printf("*ptr %i\n", *ptr);	int	~~1000~~ 1200	prints the int held in arr[5]
Correct guess printf("ptr %p\n", ptr);	int *	A14C	prints address of ptr
~~Correct guess~~ ptr = arr; // reset ptr			
Correct guess arr[2] = *(ptr + 5);	int	1200 ~~1200~~	Sets integer val at arr[2] to 1200
" " printf("arr[2] %i\n", arr[2]);	int*	~~A140~~	prints address of arr[2]
this is correct		~~A160~~	~~arr[2]~~ 10 + 10 = ~~arr[12]~~ address
✗ ptr = (arr + 10);	int *	~~A184~~	prints printer address
✗ printf("ptr %p\n", ptr);	int *	~~A184~~	random value since
✗ printf("*ptr %i\n", *ptr);	int	~~A?160~~	we went A100B
		13451480	

3) Edit the C file
   a) Add the code from problem #2 to your C file
   b) Compile and execute your C file - capture the output
   c) In the PDF clearly state the line of code, your guess and what the result was. If you guessed correctly then state – correct guess, otherwise clearly explain the incorrect guess.
   d) Explain how the value for *ptr was deteremined based on

```
ptr = (arr + 10);
printf("ptr %p\n", ptr);
printf("*ptr %i\n", *ptr);
```

arr = 10 elements + 10 elements — this enables pgm to calculate address for 20th element = arr[19]
↳ this pulls the int out of the memory location

TO TURN IN:

A zip file containing:

- Your PDF file
- Your C file

You better know the naming scheme.

CSCD 240

NOTE: Your answers, for all problems, will be saved in a file named cscd240Lab8pointers.pdf for all problems

NOTE: Your C file will be named cscd240Lab8.c

1) Type in, compile and execute the following code.

```
#include <stdio.h>
int main()
{                    0  1  2  3  4  5  6  7  8  9
        int arr[] = { 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000};
        int *ptr = arr;

        /* This gives us an idea of the memory map */
        printf("sizeof(ptr) %ld\n", sizeof(ptr) );
        printf("sizeof(arr[0]) %ld\n", sizeof(arr[0]) );

        printf("arr %p\n", arr);
        printf("ptr %p\n", ptr);

        printf("arr[1] %p\n", &arr[1]);
        printf("arr[9] %p\n", &arr[9]);
        printf("&ptr %p\n", &ptr);
        /* end memory map */
        return 0;

}// end main
```

cs.linux memMap
0x7fffff5A7B5b0

0		5	B5C4
1	B5B4	6	B5C8
2	B5B8	7	B5CC
3	B5BC	8	B5D0
4	B5C0	9	B5D4

This code will provide a base address for arr and ptr.

Use the address of 0x7fff3d830280 for the starting location of the array.
Use the address of 0x7fff3d8302a8 for the location of ptr.

Answer /complete the following

a) What is the size of ptr? 4 8
b) What is the size of arr[0]? 4 4
c) Draw a memory map that shows the memory locations of each element of the array and of ptr.

arr[0]	0x BF8C A138	arr[5]	A14C
[1]	A13C	[6]	A150
[2]	A140	[7]	A154
[3]	A144	[8]	A158
[4]	A148	[9]	A15C