Josh Harshman
cscd240
Lab7

Lab 7

1.
a. What is the size of ptr?
-The size of the pointer variable ptr is 8.

b. What is the size of twod?
-The size of twod is 48 --maybe? sizeof(twod) does return 48.

c. What is the size of twod[0] and why?
-The size of twod[0] is 12 because it is a 4x3 array. 4x3=12.

d. What is the size of twod[0][0]?
-The size of twod[0][0] is 4.

e. What can you say about twod and towd[0] as it relates to the name of the array?
- twod is the name of the array and can be used as a pointer to the base address of the array. twod[0] is the value at the base address of the array so &towd[0] should have the same address as twod.

f. Draw a memory map that shows the memory locations of each element of the array and of ptr

Address		value		
0x7fffa46b3790	0	->	{0x200, 1}	{0x204, 2}
0x7fffa46b3794	10	->	{0x300, 11}	{0x304, 12}
0x7fffa46b3798	20	->	{0x400, 21}	{0x404, 22}
0x7fffa46b379c	30	->	{0x500, 31}	{0x504, 32}
…				
…				
0x7fff24a09cd8	ptr=0x7fffa46b3790			

2.

printf("twod + 3 is: %p\n", twod + 3);
Guess: Type is a pointer. The pointer is being incremented by 3 and then being printed. The memory address will be 0x7fffa46b379c
--Correct Guess
printf("*(*(twod + 1)) is: %d\n", *(*(twod + 1)));
Guess: Type will be a decimal. The base address of the array is being temporarily incremented by 1. It will print the value at index 1 of the array which is 10
--Correct Guess
printf("*twod + 1 is: %p\n", *twod+1);
Guess: Type will be a pointer. The array name (which is acting as a pointer) will be incremented by 1, to the next contiguous memory space. This will print the address of index 1 of the array. The address will be 0x7fffa46b3794
--Correct Guess
printf("*twod[2] is: %d\n", *twod[2]);
Guess: Type will be a decimal. The array name is being dereferenced at index 2. This will print the value of the twod array at index 2 which is 20.
--Correct Guess
printf("*(twod + 2) + 2 is: %p\n", *(twod + 2) + 2);
Guess: This will be a pointer. It will print address of the array at index [2][2]. Which according to my memory map above will be 0x404
--Correct Guess
printf("twod[1] is: %p\n", twod[1]);
Guess: This will be a pointer. It will print the address that twod[1] points to. The address will be 0x7fffa46b3794
--Correct Guess
printf("twod[1][2] is: %d\n", twod[1][2]);
Guess: This will be a decimal. It is resolving the sub rows and cols of the 2 dim array. It will print out the value held at the index [1][2] in the 2dim array twod. The value will be 12
--Correct Guess

printf("ptr %p\n", ptr);
Guess: This will be a pointer. Not to be confused with &ptr which is the address at which the actual pointer variable ptr is stored, this will print out the address which it points to which is the base address of the 2d array 0x7fffa46b3790
--Correct Guess
printf("twod [1] %p\n", twod [1]);
Guess: This will be a pointer. It will print the address that twod[1] points to. The address will be 0x7fffa46b3794
--Correct Guess

printf("ptr[1] %?\d", ptr[1]);
Guess: This will be a decimal. It will print out the value that ptr[1] points to which is 1
--Correct Guess
printf("ptr + 1 %p\n", ptr + 1);
Guess: This will be a pointer. This will increment the address that ptr points to, to the next memory address in contiguous memory. It will print the memory address that ptr+1 points to which is 0x200 (according to my memory map)
--Correct Guess
printf("*(ptr + 1) %d\n", *(ptr + 1));
Guess: This will be a decimal. It will increment the address that the pointer variable points to by 4 bytes and print the value held at that address. It will print the value at the memory address that ptr+1 points to which is 1. ptr[1] yields same result.
--Correct guess
printf("twod + 1 %p\n", twod+1);
Guess: Will be a pointer. Will print the address of the array at index 1 which is 0x7fffa46b3794
--Correct Guess
printf("*twod + 1 %p\n", *twod + 1);
Guess: WIll be a pointer. Will print the address of the array at index [0][1]. According to my memory map this will be 0x200
--Correct Guess
printf("ptr[8] %d\n", ptr[8]);
Guess: This will print a decimal. Since ptr points to the base address of the array, and the values are stored in contiguous memory, ptr sub 8 will print the value held at index 8 of the array which is 22.
--Correct Guess

3.

a. Done

b. DeadDingos-MacBook-Air:Lab7 dingo$ ./a.out
sizeof(ptr) 8
sizeof(twod[0]) 12
sizeof(twod[0][0]) 4
twod 0x7fff5d9fbbc0
ptr 0x7fff5d9fbbc0
&twod[0] 0x7fff5d9fbbc0
&twod[0][0] 0x7fff5d9fbbc0
&twod[0][1] 0x7fff5d9fbbc4
&twod[0][2] 0x7fff5d9fbbc8
&twod[1] 0x7fff5d9fbbcc
&ptr 0x7fff5d9fbbb8

twod + 3 is: 0x7fff5d9fbbe4
((twod+1)) is: 10
*twod + 1 is: 0x7fff5d9fbbc4
*towd[2] is: 20
*(twod+2)+2 is: 0x7fff5d9fbbe0
twod[1] is: 0x7fff5d9fbbcc
twod[1][2] is: 12

ptr 0x7fff5d9fbbc0
twod[1] 0x7fff5d9fbbcc
ptr[1] 1
ptr + 1 0x7fff5d9fbbc4
*(ptr + 1) 1
twod+1 0x7fff5d9fbbcc
*twod+1 0x7fff5d9fbbc4
value: 1570749380
ptr[8] 22

c. Done. Refer to question 2

4. No, the code will not compile as it is not a valid printf statement. There are no arguments, only variable formatters. No closing parenthesis, and no semicolon.

5. Function added.
output:

0 1 2
10 11 12
20 21 22
30 31 32

6. Function added.
output:

2 1 0

12 11 10
22 21 20
32 31 30

7. Function added
output:

0 1 2
10 11 12
20 21 22
30 31 32

8. Function added
output:
2 1 0
12 11 10
22 21 20
32 31 30

9. Function added.
output:

2 1 0
12 11 10
22 21 20
32 31 30

My guess is that the warning about incompatible pointer types comes from passing a pointer into a function expecting a different type of pointer.

10. Function added
output:

0 1 2
10 11 12
20 21 22
30 31 32

11. In the expression *(twod[3]), the twod[3] portion of the statement is evaluated first because of order of precedence. This will evaluate to point to the equivalent position of twod[3][0]. The expression is then dereferenced to get the value at that index. If executed in this context, the value would return 30. On the other hand, in the expression (*towd)[3], the *twod portion of the statement is evaluated first because order of precedence. This dereferences the value that the base address. The rest of the expression is then evaluated and ends up returning a 10 as the value. This is because the memory the array is stored in is contiguous and hence one is able to iterate through it linearly in this context.

12. Yes you can pass the two dimensional array (twod), to a function with the prototype "void function7(int **twod, int rows, int cols)". The 2 dimensional array is an array of pointers to pointers already, but the compiler will generate incompatible pointer type warnings if you try to do this without casting to a pointer - to pointer - to pointer. In order to modify **, you need ***.

13. Much of the functionality stays the same when you pass an array by pointer or by square brackets. Technically the array name and a pointer to the array are the same thing, except the pointer is more flexible in that it is not a constant. When passing the pointer of a 2d array by first dereferencing the 2d array into a 1d pointer, the process of iterating through the array becomes a lot more linear in a function. This is in contrast to passing it by brackets in which the process for iterating through the 2d array requires 2 nested for loops. When passing by brackets however, you must give the number of columns so the function can know the width of the array. The number of rows and overall size can be omitted because the function does not need to allocate space for the array as it was already declared explicitly.

14.
DeadDingos-MacBook-Air:Lab7 dingo$ make
gcc cscd240Lab7.c
./a.out
sizeof(ptr) 8
sizeof(twod[0]) 12
sizeof(twod[0][0]) 4
twod 0x7fff59181b80
ptr 0x7fff59181b80
&twod[0] 0x7fff59181b80
&twod[0][0] 0x7fff59181b80
&twod[0][1] 0x7fff59181b84
&twod[0][2] 0x7fff59181b88
&twod[1] 0x7fff59181b8c
&ptr 0x7fff59181b78

twod + 3 is: 0x7fff59181ba4
((twod+1)) is: 10
*twod + 1 is: 0x7fff59181b84
*towd[2] is: 20
*(twod+2)+2 is: 0x7fff59181ba0
twod[1] is: 0x7fff59181b8c

```
twod[1][2] is: 12

ptr 0x7fff59181b80
twod[1] 0x7fff59181b8c
ptr[1] 1
ptr + 1 0x7fff59181b84
*(ptr + 1) 1
twod+1 0x7fff59181b8c
*twod+1 0x7fff59181b84
ptr[8] 22
0 1 2
10 11 12
20 21 22
30 31 32

2 1 0
12 11 10
22 21 20
32 31 30

0 1 2
10 11 12
20 21 22
30 31 32

2 1 0
12 11 10
22 21 20
32 31 30

2 1 0
12 11 10
22 21 20
32 31 30

0 1 2
10 11 12
20 21 22
30 31 32
```