ARRAYS

CPTS 121 L25 LAB 8

WHAT IS AN ARRAY?

- A contiguous section of memory
- A data structure to organize N elements for some type T
    - A data structure is a way of organizing memory

```
int myArr[15]; // N = 15, T = int
```

- The code will reserve (15*sizeof(int)) = 60 bytes (assume sizeof(int) = 4)
- We access each element by using [] operator
- If we use the name, it acts a pointer, but it is NOT a pointer itself.
    - myArr is 0x0f234c4385, but myArr is not a pointer, it is a data structure

HOW DO WE ACCESS EACH ELEMENT?

- We can use the [] operator or pointer notation *(ADDR [+ OFFSET])

int myArr[15];

… init myArr …

Set Value	Get Address
myArr[10] = 10;	&myAddr[10]
*(myArr + 10) = 10;	myArr + 10;

Important: myArr stores a memory address!

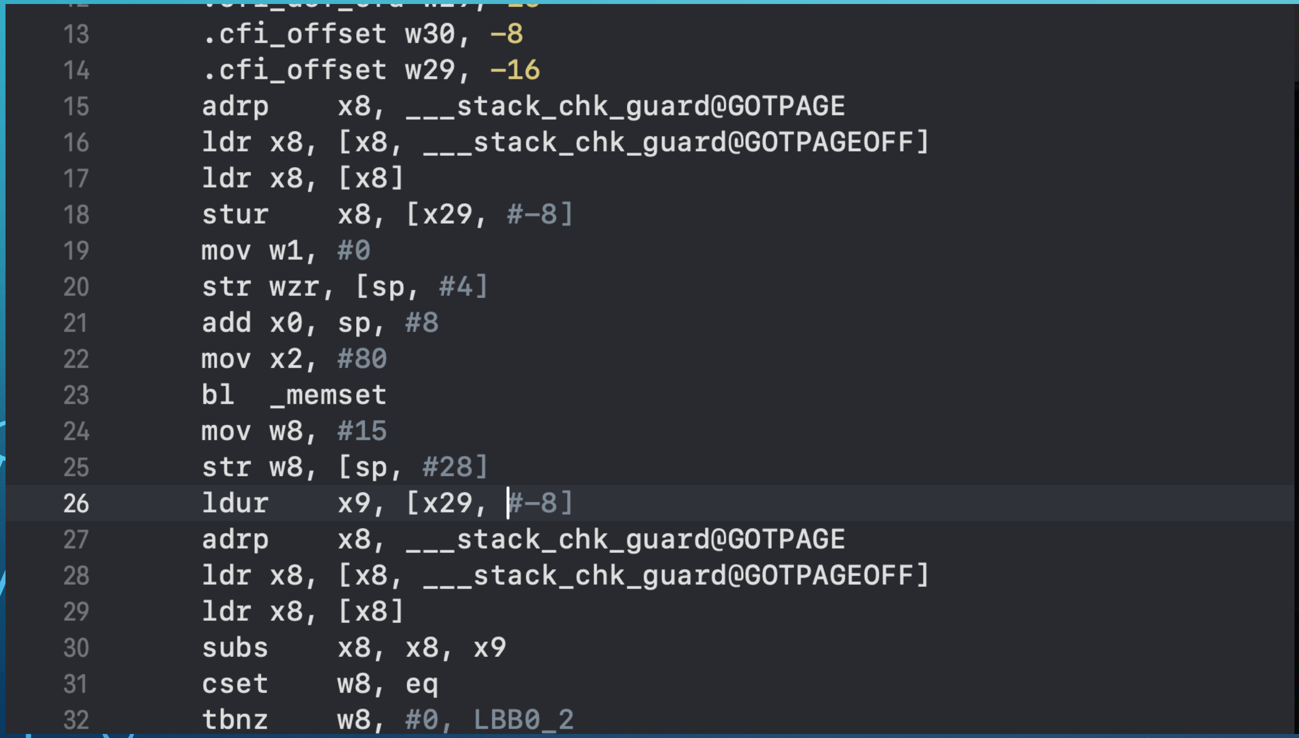
WHAT ACTUALLY HAPPENS WHEN WE READ AT AN INDEX?

sp is stack pointer (where we are working in memory for the call stack)

# indicates a constant

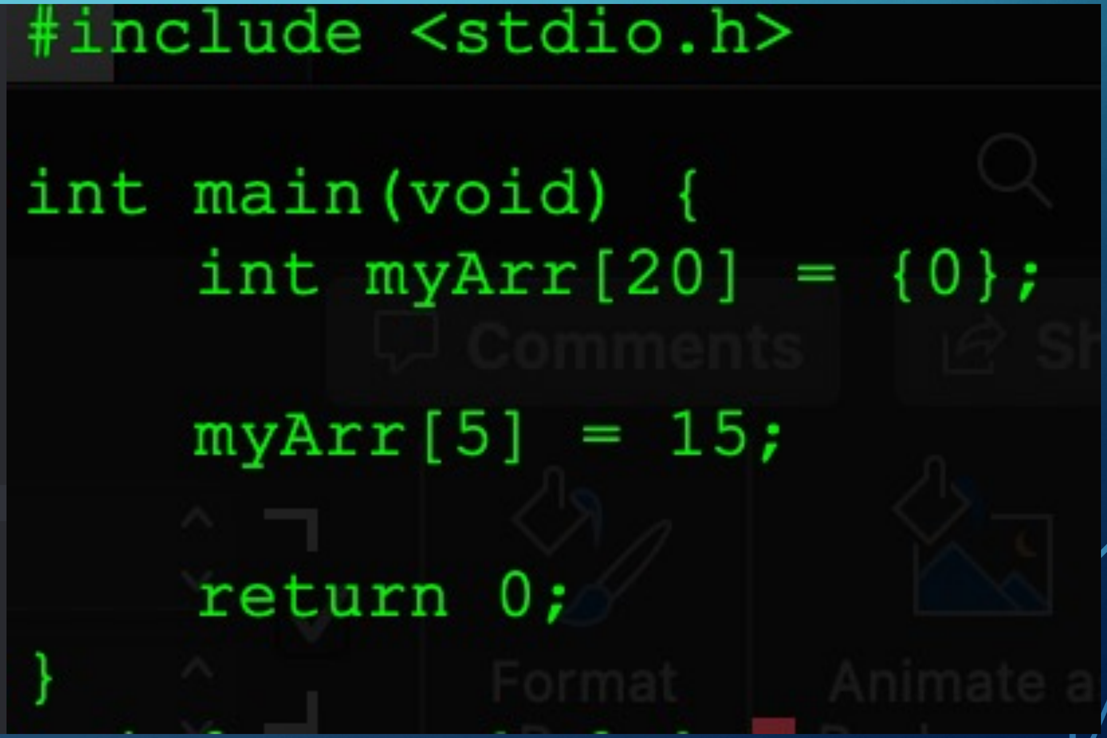
Line 25: We store w8 into sp at offset 28 ()

As we can see, we say myArr[5], but it is using the Offset #28, (#28 / 4 bytes) = 7. It is using 7 due to Internal offsets. Assuming it says #20, this shows that the Computer knows the size of the type we are working with And is working with different offsets than what we do in C And other HLLs.



```
13        .cfi_offset w30, -8
14        .cfi_offset w29, -16
15        adrp    x8, ___stack_chk_guard@GOTPAGE
16        ldr x8, [x8, ___stack_chk_guard@GOTPAGEOFF]
17        ldr x8, [x8]
18        stur    x8, [x29, #-8]
19        mov w1, #0
20        str wzr, [sp, #4]
21        add x0, sp, #8
22        mov x2, #80
23        bl  _memset
24        mov w8, #15
25        str w8, [sp, #28]
26        ldur    x9, [x29, #-8]
27        adrp    x8, ___stack_chk_guard@GOTPAGE
28        ldr x8, [x8, ___stack_chk_guard@GOTPAGEOFF]
29        ldr x8, [x8]
30        subs    x8, x8, x9
31        cset    w8, eq
32        tbnz    w8, #0, LBB0_2
```

Assembly



```c
#include <stdio.h>

int main(void) {
    int myArr[20] = {0};

    myArr[5] = 15;

    return 0;
}
```

C

HOW DO WE PASS THIS TO A FUNCTION?
(NOT BEST, USE ONLY WHEN WE KNOW THE LENGTH)

```
int sum(int inputArr[5]) {
    int accum = 0;
    for (int i = 0; i < 5; i += 1) {
        accum += inputArr[i];
    }
    return accum;
}
```

HOW DO WE PASS THIS TO A FUNCTION? (WRONG WAY, DO NOT DO THIS!)

```
int sum(int * inputArr) {
    int accum = 0;
    for (int i = 0; i < sizeof(inputArr); i += 1) {
        accum += inputArr[i];
    }
    return accum;
}
```

PROBLEMS WITH PASSING IT AS IN SLIDE 6

- We will go through the array the same # of times as sizeof(int*) has bytes
  - i.e., 8 iterations if sizeof(int*) is 8 bytes. N iterations = N bytes
- If we modify the function to rely on a null terminator (i.e., NULL or '\0'), we may run in an infinite loop when that is missing.
- We have undefined behavior if we go past the size of the array.
  - i.e., The length (size) of the array is 3, but sizeof() returns 8. In the 5 out of bound positions, we may encounter: a fatal error, useless values, and/or corrupt other memory locations (depends if we are reading or writing)
  - CWE 170 - https://cwe.mitre.org/data/definitions/170.html
- (Assume we copy to another array)
  - CWE 120 - https://cwe.mitre.org/data/definitions/120.html **Buffer Copy without Checking Size of Input**

HOW DO WE PASS THIS TO A FUNCTION? (MOST PROPER WAY)

```
int sum(int * inputArr, size_t size) {
    int accum = 0;
    for (size_t index = 0; index < size; index += 1) {
        accum += inputArr[index];
    }
    return accum;
}
```

WHY WE PREFER THE FUNCTION ON SLIDE 8

- It allows us to stop at the length of the array
- We do not rely on reserved values (i.e., all values are valid)
  - If we reserve values, then we run the chance of terminating prematurely
- NOTE: We should handle strings with a length as well.
  - While we can rely on strings to be terminated with a null character, we cannot rely that the input we are working with is a string. That is, someone may have passed in a character array without the null terminator.

REMINDERS

- PA 5 (Yahtzee) Due Wednesday October 25, 2023
- Quiz 6 Due Monday October 23, 2023
- Look over the review materials I have posted on GitHub. They go over all topics that have been covered in addition to some you will cover in the coming weeks. I will be updating it as I have time to come up with more questions.
- Remaining:
  - Q6 (Arrays), Q7 (Arrays and Strings), Q8 (Recursion & More Pointers), Q9 (*Not sure*)
  - PA 5 (Yahtzee), PA 6 (Battleship), PA 7 (Poker)
  - Lab 9 (Strings), Lab 10 (Structures), Lab 11, Lab 12, **Lab 13 (Very Similar to Final)**
  - Exam 2, Lab Final, Written Final (Essentially Exam 3)