CSO-Recitation 05

CSCI-UA 0201-007

R05: Assessment 03 & Pointers & Arrays

Today’s Topics

- Assessment 03
- Pointers
- Arrays

Pointers

A variable that stores a memory address

What are pointers?

- They are variables that store addresses
  - Pointers can have different types, depending on what they point to
    - But they remain the same size – for us on a 64-bit system, 8 bytes (64 bits)

Type	Value	Address
int	an integer number	memory address
float	a floating point number	memory address
char	a character/byte	memory address
pointer	memory address	memory address

- If I want the value of a variable var -> var
- If I want the address of a variable var -> &var
- If var is a pointer, then I can get the value of the variable that var points to -> *var

What are pointers?

- They are variables that store addresses
  - Pointers can have different types, depending on what they point to
    - But they remain the same size – for us on a 64-bit system, 8 bytes (64 bits)
- Two primary operations
  - & - called “reference”
    - Gets the address of a variable / array element
    - You perform this to get the value for a pointer
  - * - called “de-reference”
    - Gets the value located at a memory address
    - You perform this on the pointer

How do you use pointers?

- Say you have a variable var
  - int var = 10;
- You can make a pointer called ptr using this code
  - int *ptr;
- ptr can be set to point to var with the reference operator
  - ptr = &var;
- The value of ptr is now the address of var, not its value
  - To get the value, de-reference:
    - *ptr //this equals to 10
    - *ptr = 5; // this sets var to 5

Pointer types

- Why do we need pointer types?
  - Without it, making mistakes like de-referencing a number by accident would be common
  - Without it, pointer arithmetic wouldn't work
- What is pointer arithmetic?
  - If you have a pointer called ptr, the value of ptr+1 is based off the type of ptr
    - If ptr is a char*, then ptr+1 is the memory address of next char after ptr
    - If ptr is an int*, then ptr+1 is the memory address of next int after ptr
  - ptr+n means "start at ptr, and go forward as many bytes as *n* copies of what ptr points to take up"

Function arguments and pointers

- In C, arguments are passed by value
  - Means that when you call a function, the arguments are copied from the caller to the function's stack frame
  - This means that if a function modifies one of its arguments, it is not modified for whoever called the function
- If you want to pass a reference, you must use pointers
  - Then the function can modify the variable by dereferencing the pointer

Arrays

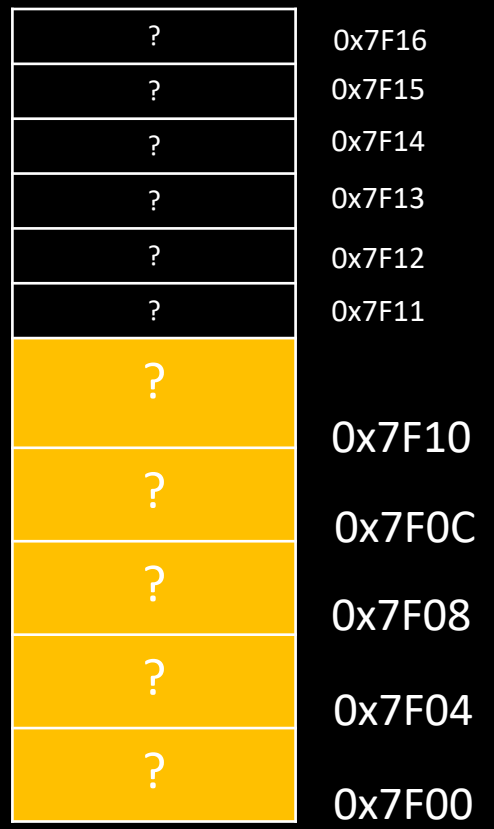
Contiguous, homogenous data

What are arrays?

- Basically, they are chunks of memory that hold a number of elements of the same data type
- This memory is contiguous, that is, the elements are all touching
- You can define an int array like this
  - int my_array[5];
  - This will make an array of 5 ints (20 bytes)
  - You can initialize the array as follows:
    - int my_array[5] = {1, 2, 3, 4, 5};
    - You can also set it to all zeroes using int my_array[5]={0};
- You can index with the [] operator
  - my_array[0] gets the first element of my_array
  - my_array[0] = 5 sets the first elelment of my_array to 5

Defining an array

- int arr[5];
- The value of an array is the address of its first element
  - The value of arr is 0x7F00
    - arr==&arr[0]
- Let a pointer points to the 1st element of this array
  - int *p = arr;
    - int *p = &arr[0];
- Array and pointer can be syntactically equivalent
  - *p == p[0], here also *p==arr[0]
  - *arr (==arr[0]) / *(arr+2) ==arr[2]



Pointer and array

- One difference between an array name and a
  - A pointer is a variable
    - p = arr; / p++; are legal
  - But an array name is not a variable..
    - cannot write things like arr++; / arr=p; (illegal)
- When an array name is passed to a function,
  - What it passed is the location of the initial element
  - So within the called function, this argument is a local variable, and so an array name parameter is a pointer, that is, a variable containing an address

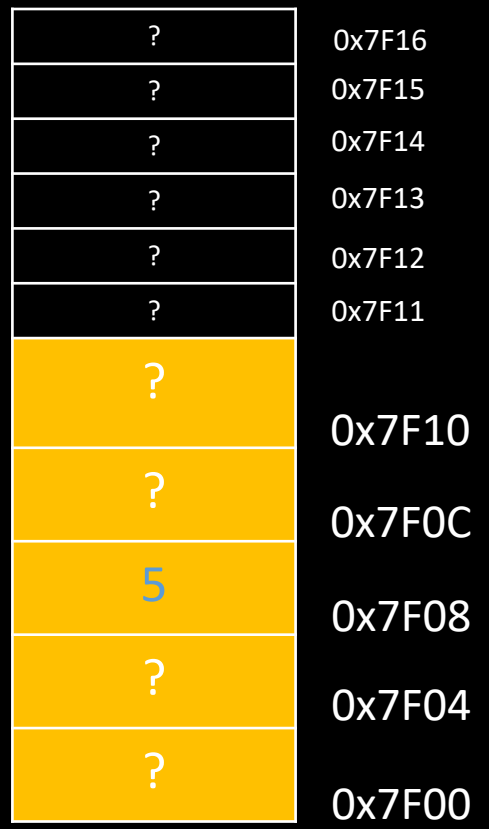
Pass array to function via pointer

```
// multiply every array element by 2
void multiply2(int *a, int n) {
   for (int i = 0; i < n; i++) {
      a[i] *= 2; // (*(a+i)) *= 2;
   }
}

int main() {
   int a[2] = {1, 2};
   multiply2(a, 2);
   for (int i = 0; i < 2; i++) {
       printf("a[%d]=%d", i, a[i]);
   }
}
```

Indexing an array

- int arr[5];
- Arrays can be index like so
  - arr[2] = 5;
  - This will set the third element of arr to 5
  - This is the same as *(arr + 2) = 5;
    - Which is to say, this is done by taking the value of arr, 0x7F00, and adding 2 to it according to pointer arithmetic
    - The size of int is 4, so we are going 8 bytes passed arr, 8 + 0x7F00 = 0x7F08



Arrays and functions

- Array names act as pointers to the array's first element
- To use a function with an array, we use pointers
  - But we need to also pass the number of elements in this array to function

Pointers to pointers (Pointer arrays)

- Since pointers are variable themselves, they can be stored in arrays just as other variables can
  - char *a[2];
- Let a pointer points to the 1st element of this array (of pointers)
  - char **p = &a[0]; / char **p=a;
- An array of pointers
- Think about what can this do?