Recitation 04

Pointers, Arrays, and Strings

Preface

Preparing for Recitation 04

Pulling the fourth assignment

- Inside your recitations repository in your VM
  - Run **git pull upstream master**
  - Make sure you have an r04 directory
  - You should have an in-class directory and a for-homework directory inside r04

Today's agenda

- We will discuss in recitation
  - Pointers
  - Arrays
  - Strings
- What you will do tonight
  - R04
    - Finish a program that counts the number of characters in the command line arguments passed to it

Pointers

A variable that stores a memory address

xkcd.com/138/

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 10
  - *ptr = 5; //this sets a to 5

Pointer types

- Why do we need pointer types?
  - Without it, making mistakes like de-refencing 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 next char after ptr
    - If ptr is an int*, then ptr + 1 is the 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
  - That 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
  - See ex1 in the r04 directory
- If you want to pass a reference, you must use pointers
  - Then the function can modify the variable by dereferencing the pointer
  - See ex2

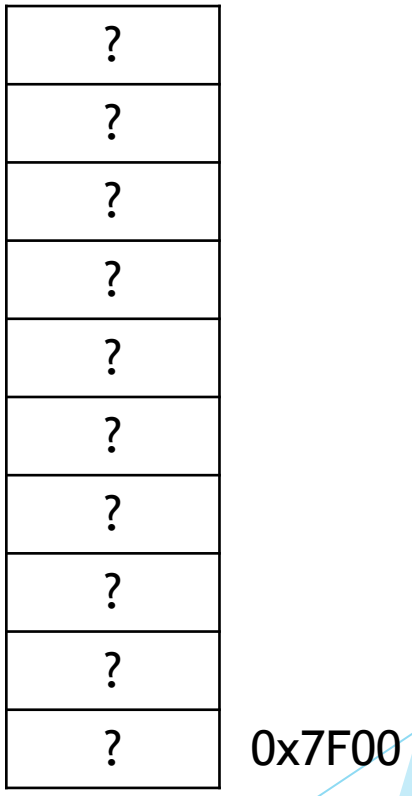
Arrays

Contiguous, homogenous data

What are arrays?

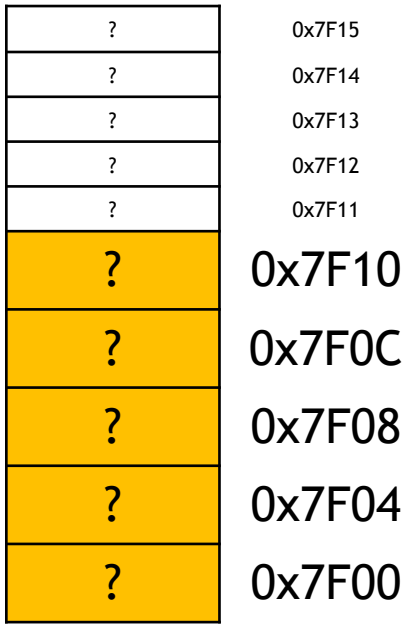
- Basically, they are chunks of memory that hold a number of elements of the same data type
- This memory is contiguous, that is, the elements are all touching
- You can define an int array like this
  - int my_array[5];
  - This will make an array of 5 ints, or 20 bytes
  - You can initialize the array 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 element of my_array to 5

Defining an array



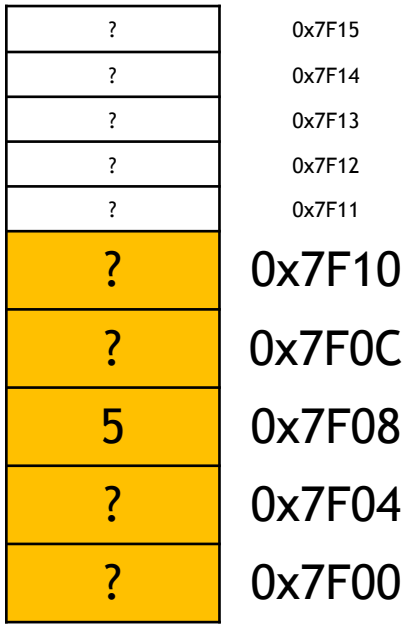
Defining an array

- int arr[5];
- The value of a an array is the address of its first element
  - The value of arr is 0x7F00
- The compiler keeps information about an array's size and type



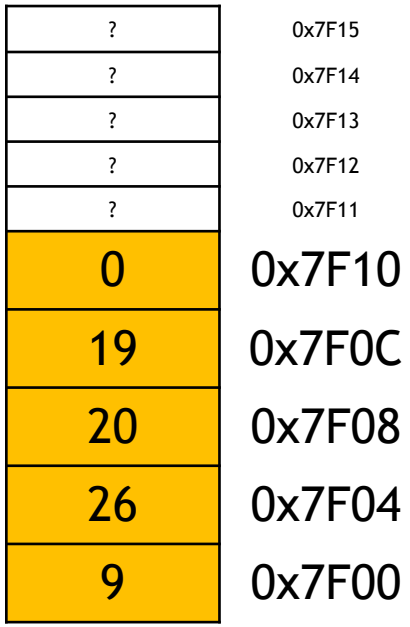
Indexing an array

- 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



Initializing an array

- int arr[5] = {9, 26, 20, 19, 0};



Arrays and functions

- Array names act as pointers to the array's first element
  - Key difference being compile-time size information and immutability (you can't reassign an array name)
- To use a function with an array, we use pointers
  - But then is there a problem?
- See ex3 and ex4

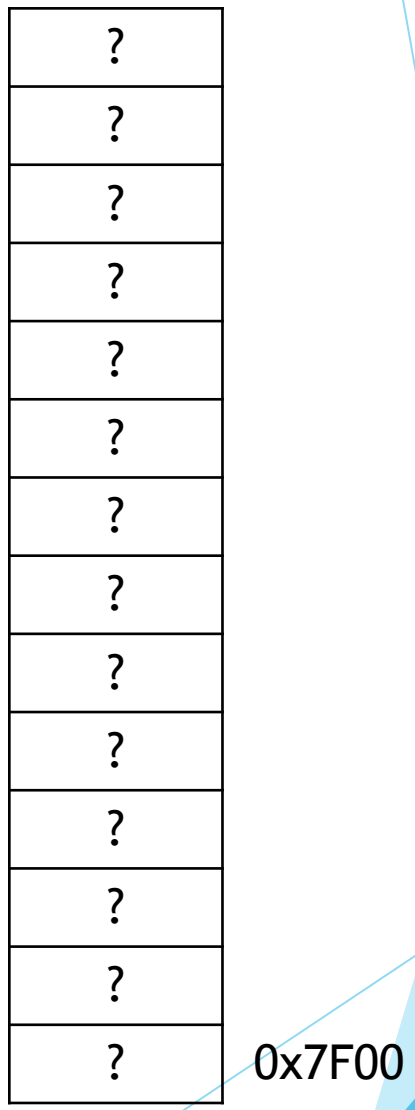
Strings

Arrays of chars

What are strings?

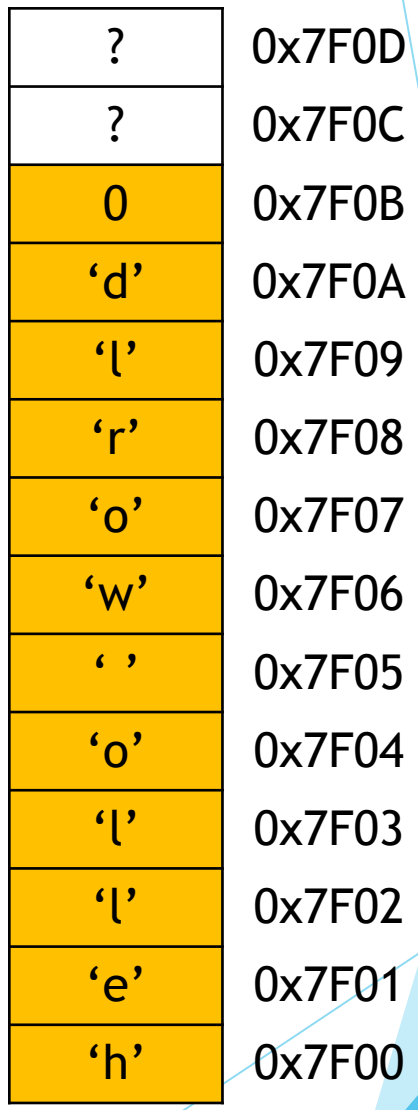
- They are arrays of the type char, which is typically one byte
- Char literals are in single quotes ‘’
- String literals are in double quotes “”
- Unlike other arrays, strings have a way of knowing the length even at runtime
  - Strings are stored with the last byte set to 0
  - C strings are called “null terminated”
  - So you can find the length by looping over the string, keeping a counter, and stopping when you find a char equal to zero
  - There is also a standard library function for this, strlen
- See ex5

Defining a string



Defining a string

- char *arr = “hello world”;
- The literal “hello world” includes the null-terminator.



What was that nonsense last week about argv and argc?

- Programs take in a number of arguments when run from the command line
- git is a program you are familiar with, it takes a few arguments
- If you run git add main.c, then there are 3 arguments
  - git
  - add
  - main.c

What was that nonsense last week about argv and argc?

- To see these arguments, the function main is allowed to take two arguments
- These are typically called argc, for “argument count”, and argv, for “argument vector”
- argv is an array of pointers to strings
- argv has argc many elements
- See ex6