Lecture 1 Notes 9/3

Tuesday, September 3, 2013 7:45 PM

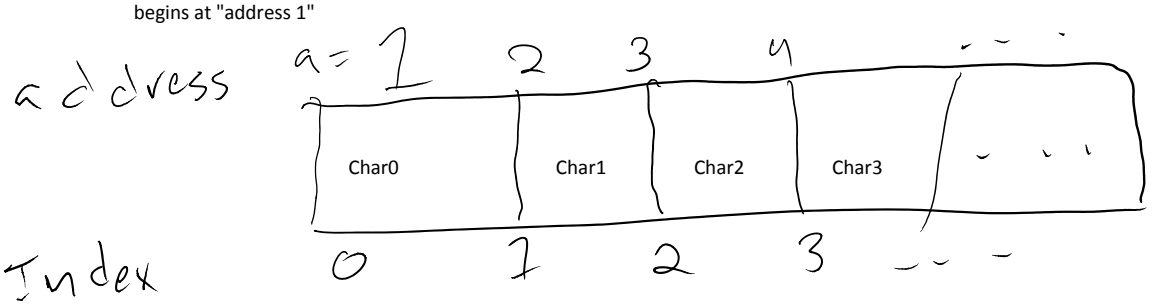
CS61: System Programming and Machine Organization

- GIT:
  - NOTE: **Lectures rely on code, and it is important to have the code in front of you so as not to fall behind. If you have any problems cloning this repository please ask your TFs for help.**
  - git clone git://code.seas.harvard.edu/cs61/cs61-lectures.git
  - Repository containing lecture code
- Course Websites
  - Wiki: http://cs61.seas.harvard.edu/wiki/2013/Home
    - Contains course resources and syllabus
  - Feedback page: http://cs61.seas.harvard.edu/feedback/
- Topics of the course
  - 1) How do computers run programs?
  - 2) How do we make programs robust?
    - Capable of performing as you desire, even with weird inputs, attacks, conflicts, etc.
    - EX: Feedback tool stopped working when a student opened hundreds of thousands of connections.
      - What was the problem?
        - Server ran out of a resource, specifically file descriptors. Only can open finite files, after that server crashes.
      - Possible Solutions
        - MAC addresses to ID unique users, only one connection each; however, MAC are single hop, not part of routing; MACs are spoofable
        - Limit usage per user, require captcha.
        - Still potentially vulnerable: pay people to break captchas.
          - Improved robustness by authenticating users.
  - 3) How can we make programs fast (performance)?
    - Not just algorithmic - using properties of the machine
- Example: Program to add two numbers
  - EX: int sum(int a, int b){ return a + b; }
  - changed return type of function to unsigned
    - Unsigned represents only positives, but still seems to work.
  - changed to char* sum(char* a, int b){ return &a[b]; }
    - Using pointer arithmetic - adds b to the address and returns the address, in effect adding the two numbers.

What is an Array:
An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

From <http://www.cplusplus.com/doc/tutorial/arrays/>

Suppose "a" = 1. Because "a" is a pointer to an array of char's (char *) we could say that the array "a" begins at "address 1"



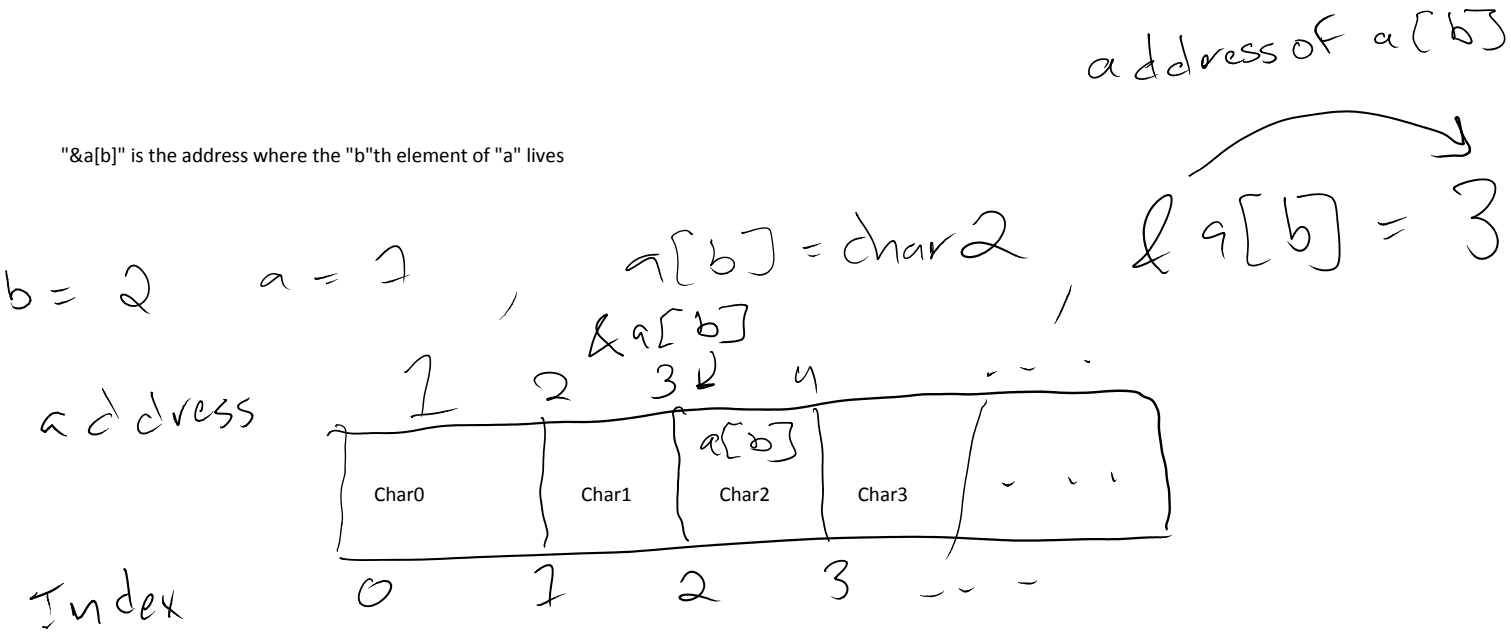
"a" = Index 0, Address 1 (the index is actually ambiguous but you should note the index and the address are different. The index is respective to the position in the array and the address is the location in memory).

The "&" symbol in C returns the address (i.e. the location in memory) where the array is stored.

"a[b] "alone is the value stored at index "b" of the array that "a" points to.

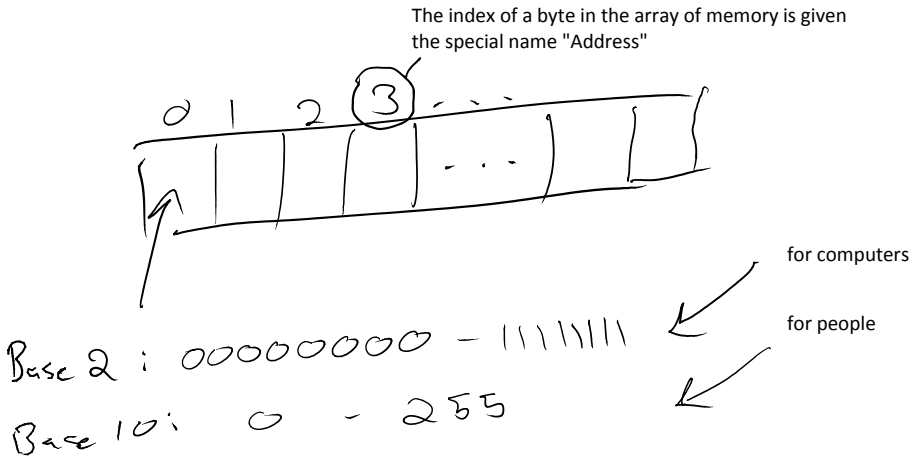
"a[b] "alone is the value stored at index "b" of the array that "a" points to.

"&a[b]" is the address where the "b"th element of "a" lives

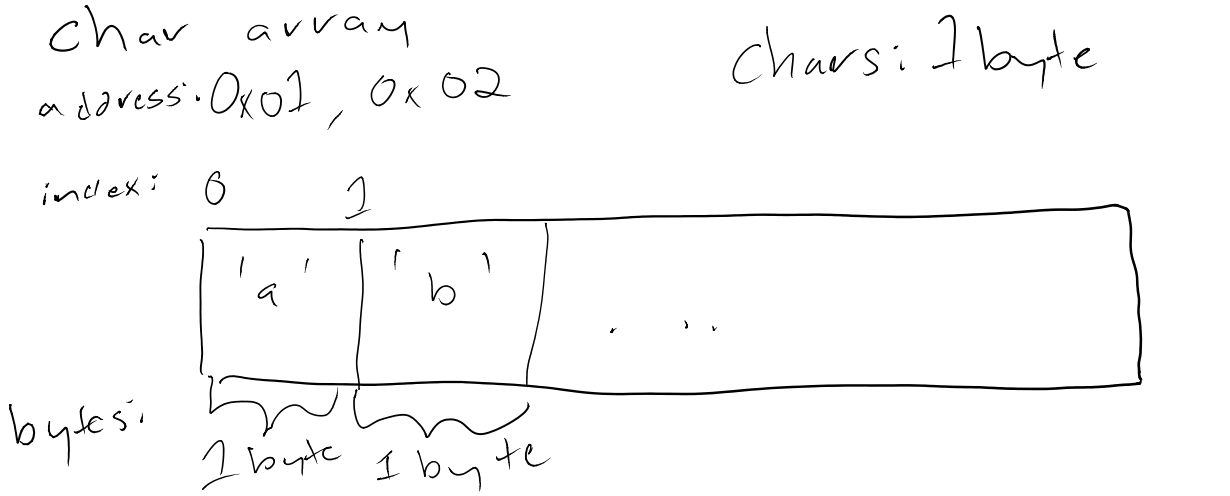


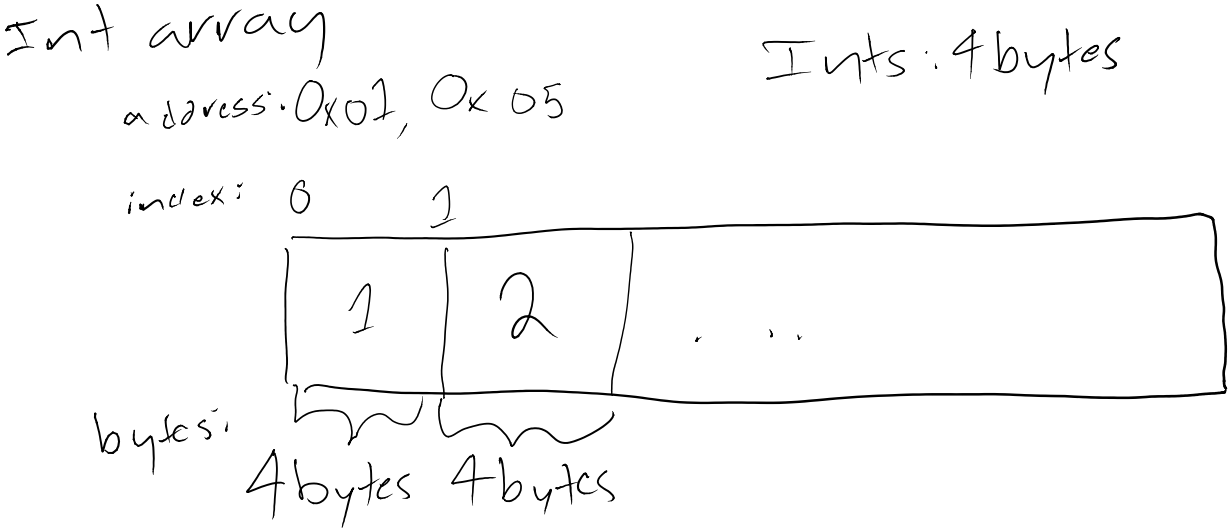
- Why doesn't it segfault?
  - Because "a" is set to a random integer we may not have access to the value in memory at the address of "a." When we access memory we are not allowed to we seg fault.
  - When we do not return the content "a[b]" but return the address "&a[b]" we never directly access the value in memory, so there is no segfault.
- If you actually look into a program, you see lots of random characters, many of which are ^@ which represents the byte 0.
- We can think of memory as being a huge array. In C terms: unsigned char memory[VERY_LARGE];
  - Simply an enormous array of bytes that can have any value between 0 and 255 (2^8-1) since a byte is 8 bits which are each 0 or 1

Memory: Huge Array of Bytes



- To run a program, we essentially pull the program off the disk into memory.
- Why is **int*** sum(**int*** a, int b){ return &a[b]; } different than **char*** sum(**char*** a, int b){ return &a[b]; }
  - "chars" are 1 byte in memory and "ints" are 4 bytes in memory. In a char array each element is separated by 1 byte, in an int array each element is separated by 4 bytes. (See picture below).
  - Let's look at how ints are stored in memory.
  - First byte of 1 = 0x1
  - First byte of 100 = 0x64 (Hexadecimal - 16*6 + 4 = 100)
  - First byte of 256 = 0: Too big for first byte, which can only hold up to 255
  - Ints are represented by 4 bytes on a 32-bit architecture.
    - Processor cannot understand ints over 2147483647 as a result.
    - 4 bytes = 32bits = 2^32 - 1





- Use hexdump to print out sum function: 8b 44 24 08 03 44 24 04 c3
  - If we use the unsigned version, the code remains exactly the same.
  - Same if we use the char* version.
  - Even though our C code was different, the actual compiled code is exactly the same!
  - But if we look at the int* version, we see a slight difference - another byte that multiplies by 4.
    - This explains our weird results. Instead of incrementing the address by 1, the size of a char, the program now increments by 4, the size of an int every time we increment the index of the array.
    - As a result, sum returns a + 4*b instead of a+b.
- Code is just numbers! We can just use an array containing the numbers 8b 44 … and use it to sum!
  - We can even use files or a picture!
  - Open hellokitty and use bytes at a specific place in the picture as the sum function.
    - Actually slightly cheated - edited a couple bytes in the hellokitty logo. Visually, just means a few pixels are changed.
  - Can we run anything? Well, if we run the code at address 13 in hello.jpg, we get a segfault. It gave us illegal instructions. Our computer is robust enough to shut it down and not let it run.

- Takeaways
  - All data is the same to the processor.
  - Types are an abstraction provided by the C language
  - The C compiler converts the C world that we think about when we are programming to the language of the actual processor
  - Changing the index of an array corresponds to changing the address of the array by the size of the elements in the array (measured in bytes)
  - Memory (like many things in your computer) is a finite resource and when you run out you're in trouble
  - CS61 is going to rock, if these concepts seem hard to you now do not worry they will be covered at length throughout the course.