CSE351

Announcements:
- HW1 released later today

Today’s topics
- More on addresses/pointers
- Bit-level manipulations
  - Boolean algebra
  - Boolean algebra in C

What is memory, really?
- How do we find data in memory?
- What is an address?
- What is a pointer?

Arrays
- Arrays represent adjacent locations in memory storing the same type of data object
- E.g., int * array[128], allocated 512 adjacent locations in memory starting at 0x00000000
- Pointers to arrays point to a certain type of object
  - E.g., int * array[128];
  - array_ptr = &big_array[0];
  - array_ptr[0] = &big_array[3];
  - array_ptr[0] = &big_array[1];
  - array_ptr = &big_array[128];
  - In general: &big_array[i] is the same as (big_array + i)

Arrays
- Arrays represent adjacent locations in memory storing the same type of data object
- E.g., int * array_ptr;
  - array_ptr[0] = big_array;
  - array_ptr[3] = big_array[3];
  - array_ptr[1] = big_array[1];
  - array_ptr[128] = big_array[128];
  - In general: &big_array[i] is the same as (big_array + i)
General rules for C (assignments)

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory LOCATION
  - RHS must evaluate to a VALUE (could be an address)
  - E.g., `x` at location 0x04, `y` at 0x18
    - `int x, y;
x = y;` // get value at `y` and put it in `x`

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory LOCATION
  - RHS must evaluate to a VALUE (could be an address)
  - E.g., `x` at location 0x04, `y` at 0x18
    - `int x, y;
x = y;` // get value at `y` and put it in `x`

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory LOCATION
  - RHS must evaluate to a VALUE (could be an address)
  - E.g., `x` at location 0x04, `y` at 0x18
    - `int x, y;
x = y;` // get value at `y` and put it in `x`

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory LOCATION
  - RHS must evaluate to a VALUE (could be an address)
  - E.g., `x` at location 0x04, `y` at 0x18
    - `int x, y;
x = y;` // get value at `y` and put it in `x`

- **Left-hand-side = right-hand-side**
  - LHS must evaluate to a memory LOCATION
  - RHS must evaluate to a VALUE (could be an address)
  - E.g., `x` at location 0x04, `y` at 0x18
    - `int x, y;
x = y;` // get value at `y` and put it in `x`
Examine the Data Representations

**Code to print byte representation of data**
- C code to print unsigned char * creates byte array

```c
typedef unsigned char *pointer;
void show_bytes(pointer start, int len)
{
    for (i = 0; i < len; i++)
        printf(\"0x%p\", start[i]);
    printf(\"n\");
}
```

Some print directives:
- %x: Print pointer
- %a: Print hexadecimal
- \n: New line

**Representing Integers**

- **Decimal:** 12345
- **Binary:** 0010 0010 0011 1001
- **Hex:** 3039

**Representing Integers**

- **Decimal:** 12345
- **Binary:** 0010 0010 0011 1001
- **Hex:** 3039
Representing Integers

- int A = 12345;
- int B = -12345;
- long int C = 12345;

Decimal: 12345
Binary: 0001 0100 1011
Hex: 1 0 3 9

Two's complement representation for negative integers (covered later)

Representing Pointers

- int *p = &B;
- *p = 48;
- ~p

Sum P
IA32 P
x86-64 P

Representing strings

- A C-style string is represented by an array of bytes.
- Elements are one-byte ASCII codes for each character.
- A 0 value marks the end of the string.

Null-terminated Strings

- For example, “Harry Potter” can be stored as a 13-byte array.

Why do we put a 0, or null, at the end of the string?

Computing string length?

Compatibility

char s[10] = “12345”;

Byte ordering not an issue

Unicode characters – up to 4 bytes/character

- ASCII codes still work (leading 0 bit) but can support many characters in all languages in the world
- Java and C have libraries for Unicode (Java commonly uses 2 bytes/char)

Boolean Algebra

- Developed by George Boole in 19th Century
- Algebraic representation of logic
  - Encode “true” as 1 and “false” as 0
  - AND: A&B = 1 when both A is 1 and B is 1
  - OR: A|B = 1 when either A is 1 or B is 1
  - XOR: A^B = 1 when either A is 1 or B is 1, but not both
  - NOR: “A|B” when A and B are not true
  - DeMorgan’s Law: (A & B) = (~A) & (~B)
General Boolean Algebras

- Operate on bit vectors
  - Operations applied bitwise
    - 01010101
    - 01010101
    - 01010101
    - 01010101
  - All of the properties of Boolean algebra apply

- How does this relate to set operations?

Representing & Manipulating Sets

- Representation
  - Width w bit vector represents subsets of \( \{0, ..., w-1\} \)
  - \( \emptyset \rightarrow 0 \)
  - \( 01010101 \rightarrow 01100101 \)
  - \( 01010101 \rightarrow \{0, 2, 4, 6\} \)

- Operations
  - \& intersection
  - | union
  - ^ symmetric difference
  - ~ complement

Bit-Level Operations in C

- Operations &, |, ^, ~ are available in C
  - Apply to any “integral” data type
    - long, int, short, char, unsigned
  - View arguments as bit vectors
  - Arguments applied bit-wise

- Examples (char data type)
  - 0x11000000
  - 0x11111111
  - 0x080 & 0x55
  - 0x081 | 0x000
  - 0x081
  - 0x11000000
  - 0x081
  - 0x081

Contrast: Logic Operations in C

- Contrast to logical operators
  - &&, ||, !
    - View it as “Falsity”
    - Anything nonzero as “True”
    - Always return 0 or 1
    - Early termination

- Examples (char data type)
  - 0x041
  - 0x000
  - 0x001

  - (voids null pointer access, null pointer = 0x00000000)