Section 1: Memory, Data, and Addressing

- Preliminaries
- Representing information as bits and bytes
- Organizing and addressing data in memory
- Manipulating data in memory using C
- Boolean algebra and bit-level manipulations

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
  - NOT: ~A = 1 when A is 0 and vice-versa
  - DeMorgan’s Law: ~(A | B) = ~A & ~B

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

~	
0	1
1	0

Manipulating Bits

- Boolean operators can be applied to *bit vectors*: operations are applied *bitwise*

```
  01101001       01101001       01101001
& 01010101     | 01010101     ^ 01010101     ~ 01010101
----------     ----------     ----------     ----------
  01000001       01111101       00111100       10101010
```

Bit-Level Operations in C

- **Bitwise operators &, |, ^, ~ are available in C**
  - Apply to any “integral” data type
    - `long, int, short, char`
  - Arguments are treated as bit vectors
  - Operations applied bitwise
- **Examples:**

```
char a, b, c;
a = (char)0x41;        // 0x41 -> 01000001_2
b = ~a;                //         10111110_2 -> 0xBE
a = (char)0;           // 0x00 -> 00000000_2
b = ~a;                //         11111111_2 -> 0xFF
a = (char)0x69;        // 0x41 -> 01101001_2
b = (char)0x55;        // 0x55 -> 01010101_2
c = a & b;             //         01000001_2 -> 0x41
```

Contrast: Logic Operations in C

- **Logical operators in C: &&, ||, !**
  - Behavior:
    - View 0 as “False”
    - Anything nonzero as “True”
    - Always return 0 or 1
    - Early termination (&& and ||)
- **Examples (char data type)**
  - `!0x41          -->  0x00`
  - `!0x00          -->  0x01`
  - `0x69 && 0x55   -->  0x01`
  - `0x00 && 0x55   -->  0x00`
  - `0x69 || 0x55   -->  0x01`
  - `p && *p++` (avoids null pointer access: null pointer = 0x00000000)
    short for: `if (p) { *p++; }`

Representing & Manipulating Sets

- **Bit vectors can be used to represent *sets***
  - Width *w* bit vector represents subsets of {0, …, *w*–1}
  - $a_j = 1$ if $j \in A$ – each bit in the vector represents the absence (0) or presence (1) of an element in the set

01101001	{ 0, 3, 5, 6 }
76543210	
01010101	{ 0, 2, 4, 6 }
76543210	

- **Operations**

&	Intersection	01000001	{ 0, 6 }
|	Union	01111101	{ 0, 2, 3, 4, 5, 6 }
^	Symmetric difference	00111100	{ 2, 3, 4, 5 }
~	Complement	10101010	{ 1, 3, 5, 7 }