

Beginning C Programming for Engineers

R. Lindsay Todd

Rensselaer Polytechnic Institute

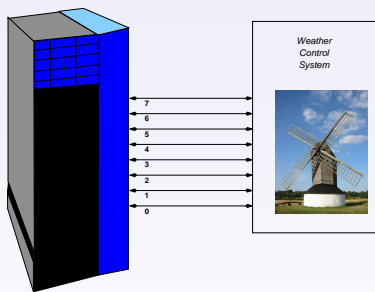
Lecture 6: Bit Operations

Outline

- 1 Why we need bit operations
- 2 Place Value
 - Binary
 - Octal
 - Hexadecimal
- 3 Conversions
- 4 Bitwise operators
 - Basic operations
 - Masking bits
 - Making “holes”
 - Setting bits
 - Masking and replacing
 - Toggling bits
 - Shifting bits
 - Bit operations and arithmetic

Why we need bit operations

Bytes, Nybbles, and Bits



- Everything is represented internally using *bits*.
- A *char* is stored in a unit of storage called a *byte*, usually 8 bits. Both *int* and *float* are often 32 bits. A *long* is usually 32 or 64 bits.
- C has bitwise operators that operate on integers as bit patterns.

Bits	Meaning
0–4	Wind speed
5–7	Wind direction
8–15	Temperature

Place Value

Place Value Numerals

- A *numeral* represents a number using some notation.
- Normal *place value* numerals: each *place* represents a power of ten. E.g.,
$$1492 \mapsto 1 \cdot 10^3 + 4 \cdot 10^2 + 9 \cdot 10^1 + 2 \cdot 10^0$$
$$108 \mapsto 1 \cdot 10^2 + 0 \cdot 10^1 + 8 \cdot 10^0$$
- Digits range from zero to nine (ten minus one). In general, for an *n* digit number $d_{n-1}d_{n-2} \dots d_0$, the value is

$$d_{n-1} \cdot 10^{n-1} + d_{n-2} \cdot 10^{n-2} + \dots + d_0 \cdot 10^0$$

Binary Numerals

- We can use other *bases* besides 10. For a base b , we need a set of b digits, one for each value from 0 to $b - 1$.
- Base two (binary) only uses digits from $\{0, 1\}$.

$$1011_{\text{two}} \mapsto 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ = 11_{\text{ten}}$$

- *Bits* are binary digits. A bit can be represented electronically with an on/off switch.
- There are no binary constants in C!

c

Octal Numerals

- For base eight (octal numerals), the digits are $\{0, 1, 2, 3, 4, 5, 6, 7\}$. E.g.,

$$127_{\text{eight}} \mapsto 1 \cdot 8^2 + 2 \cdot 8^1 + 7 \cdot 8^0 = 87_{\text{ten}} \\ 19_{\text{eight}} \mapsto \text{No!}$$

- In C, octal integer constants are written by writing a leading zero.

Octal examples

```
int a = 017; /* 1*8 + 7 = 15 */
int b = 0234; /* 2*64 + 3*8 + 4 = 156 */
int c = 019; /* Compilation error! */
```

c

Hexadecimal Numerals

- Base sixteen (hexadecimal) needs non-arabic digits. We use the set of digits:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

$$10F_{\text{sixteen}} \mapsto 1 \cdot 16^2 + 0 \cdot 16^1 + F \cdot 16^0 = 271_{\text{ten}} \\ 18_{\text{sixteen}} \mapsto 1 \cdot 16^1 + 8 \cdot 16^0 = 24_{\text{ten}}$$

- In C, hexadecimal integer constants are written by writing a leading 0x.

Hexadecimal examples

```
int a = 0x11; /* 1*16 + 1 = 17 */
int b = 0xA0; /* 10*16 + 0 = 160 */
int c = 0xAB; /* 10*16 + 11 = 171 */
```

c

I/O in Octal and Hexadecimal

- With `printf`, the `%o` format may be used to format *any* int value as octal.
- With `printf`, the `%x` and `%X` formats may be used to format *any* int as hexadecimal.
- The `%i` format can be used with `scanf` to read values as if they were C constants.

Program: numbers.c

```
#include <stdio.h>

int main() {
    int val;
    printf("Enter value: ");
    scanf("%i", &val);
    printf("Decimal: %d, %i\n", val, val);
    printf("Octal: %o\n", val);
    printf("Hex x X: %x, %X\n", val, val);
    return 0;
}
```

Results

```
Enter value: 167
Decimal: 167, 167
Octal: 247
Hex x X: a7, A7
```

c

Bit Patterns

- Bit patterns can be represented using octal or hexadecimal, e.g.

	1	1	0	1	0	1	1	0
0	3			2			6	
0x	d				6			

- Each octal digit corresponds to a pattern of three bits, since $2^3 = 8$. There are 8 possible patterns of 3 bits.
- Hexadecimal digits correspond to a pattern of four bits, since $2^4 = 16$. There are 16 possible patterns of 4 bits.

Bit Pattern Conversions

Octal	Bit pattern	Decimal
0	0 0 0	0
1	0 0 1	1
2	0 1 0	2
3	0 1 1	3
4	1 0 0	4
5	1 0 1	5
6	1 1 0	6
7	1 1 1	7

Hexadecimal	Bit pattern	Decimal
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
A	1 0 1 0	10
B	1 0 1 1	11
C	1 1 0 0	12
D	1 1 0 1	13
E	1 1 1 0	14
F	1 1 1 1	15

Bitwise operators

The bitwise operators operate on each corresponding bit of their operands. For each bit, x and y:

x	y	AND x&y	OR x y	XOR x^y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Additionally, the *complement* operator, `~`, negates each bit.

x	~x
0	1
1	0

Examples: Bit Operations

```
unsigned char x = 37, y=246;
```

x	0	0	1	0	0	1	0	1	37
y	1	1	1	1	0	1	1	0	246
x&y	0	0	1	0	0	1	0	0	36
x y	1	1	1	1	0	1	1	1	247
x^y	1	1	0	1	0	0	1	1	211
~x	1	1	0	1	1	0	1	0	218

- Don't confuse `&` and `|` with the logical `&&` and `||` operators.

```
37 && 246 ↪ 1
37 || 246 ↪ 1
```

- There are also `&=`, `|=`, and `^=` operators.

Masking

- The AND (&) operator can be used to mask (select) particular bits from a pattern of bits.
- Example: Select the wind speed (lowest 5 bits):

x	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
0x1F	0	0	0	1	1	1	1	1
x & 0x1F	0	0	0	b ₄	b ₃	b ₂	b ₁	b ₀

Program: maskbits.c

```
#include <stdio.h>

int main() {
    int x;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Wind speed: %X\n",
        (x & 0x1F));
    return 0;
}
```

Results

```
Enter number: 0x450E
Wind speed: E
```

Making "holes"

- The complement (~) operator can sometimes help us with making bit masks.
- Example: Select everything except the wind direction (bits 5–7), setting bits 5–7 to 0:

x	b ₈	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
0xE0	0	1	1	1	0	0	0	0	0
~0xE0	1	0	0	0	1	1	1	1	1
x & ~0xE0	b ₈	0	0	0	b ₄	b ₃	b ₂	b ₁	b ₀

Setting Bits

- The OR (|) operator can be used to specifically set particular bits.
- Example: Set the lowest 5 bits of a number:

x	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
0x1F	0	0	0	1	1	1	1	1
x 0x1F	b ₇	b ₆	b ₅	1	1	1	1	1

Program: setbits.c

```
#include <stdio.h>

int main() {
    int x;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Lowest 5 bits set: %X\n",
        (x | 0x1F));
    return 0;
}
```

Results

```
Enter number: 0x4503
Lowest 5 bits set: 451F
```

Masking and Replacing

- Sometimes we need to mask, then set, to get the desired result.
- Example: Transfer wind speed (5 lowest bits) from machine y to x.

x	x ₇	x ₆	x ₅	x ₄	x ₃	x ₂	x ₁	x ₀
~0x1F	1	1	1	0	0	0	0	0
x & ~0x1F	x ₇	x ₆	x ₅	0	0	0	0	0
y	y ₇	y ₆	y ₅	y ₄	y ₃	y ₂	y ₁	y ₀
y & 0x1F	0	0	0	y ₄	y ₃	y ₂	y ₁	y ₀
(x & ~0x1F) (y & 0x1F)	x ₇	x ₆	x ₅	y ₄	y ₃	y ₂	y ₁	y ₀

Code: Mask & Set

Program: maskset.c

```
#include <stdio.h>

int main() {
    int x, y;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Replacement bits: ");
    scanf("%i", &y);
    printf("With new bits: %X\n",
        (x & ~0x1F) | (y & 0x1F));
    return 0;
}
```

Results

Enter number: 0x1B49
 Replacement bits: 0x03
 With new bits: 1B43

Toggling Bits

- The exclusive or operation can be used to “toggle” bits, that is, reverse their sense from true to false, and vice versa.
- Example: Toggle bits 4 and 6.

x	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
0x50	0	1	0	1	0	0	0	0
x ^ 0x50	b ₇	\bar{b}_6	b ₅	\bar{b}_4	b ₃	b ₂	b ₁	b ₀

Program: toggle.c

```
#include <stdio.h>

int main()
{
    int x;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Toggle bits 4, 6: %X\n",
        (x ^ 0x0050));
    return 0;
}
```

Results

Enter number: 0xFF
 Toggle bits 4, 6: AF

Shift Operations

```
unsigned char x = 37;
```

x	0	0	1	0	0	1	0	1	37
x << 1	0	1	0	0	1	0	1	0	74
x << 2	1	0	0	1	0	1	0	0	148
x << 3	0	0	1	0	1	0	0	0	40
x >> 1	0	0	0	1	0	0	1	0	18
x >> 2	0	0	0	0	1	0	0	1	9
x >> 3	0	0	0	0	0	1	0	0	4

- Shift operators move bit patterns either left or right.
- With **unsigned int** numbers, shifting left is equivalent to multiplying by 2 (until bits shift off the left end); shifting right is equivalent to dividing by 2.

Example: Shifts

Program: bits.c

```
/* Extract wind direction. */

#include <stdio.h>

int main() {
    int i;

    printf("Enter number: ");
    scanf("%i", &i);
    printf("i=%x\n", i);
    printf("i>>5=%x\n", i>>5);
    printf("Bits 5-7: %x\n",
        (i >> 5) & 07);
    return 0;
}
```

Bits	Meaning
0-4	Wind speed
5-7	Wind direction
8-15	Temperature

Results

Enter number: 0x90c5
 i=90c5
 i>>5=486
 Bits 5-7: 6

Bit Patterns and Powers of 2

Some interesting bit patterns:

$2^1 - 1$	0	0	0	1	1
$2^2 - 1$	0	0	1	1	3
$2^3 - 1$	0	1	1	1	7
$2^4 - 1$	1	1	1	1	15

In general, $2^n - 1$ is represented in binary as a numeral with n digits all "1".

c

numbers.c

```
#include <stdio.h>

int main() {
    int val;
    printf("Enter value: ");
    scanf("%i", &val);
    printf("Decimal: %d, %i\n", val, val);
    printf("Octal: %o\n", val);
    printf("Hex x X: %x, %X\n", val, val);
    return 0;
}
```

maskbits.c

```
#include <stdio.h>

int main() {
    int x;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Wind speed: %X\n",
          (x & 0x1F));
    return 0;
}
```

setbits.c

```
#include <stdio.h>

int main() {
    int x;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Lowest 5 bits set: %X\n",
          (x | 0x1F));
    return 0;
}
```

maskset.c

```
#include <stdio.h>

int main() {
    int x, y;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Replacement bits: ");
    scanf("%i", &y);
    printf("With new bits: %X\n",
           (x & ~0x1F) | (y & 0x1F) );
    return 0;
}
```

toggle.c

```
#include <stdio.h>

int main()
{
    int x;
    printf("Enter number: ");
    scanf("%i", &x);
    printf("Toggle bits 4, 6: %X\n",
          (x ^ 0x0050));
    return 0;
}
```

bits.c

```
/* Extract wind direction. */

#include <stdio.h>

int main() {
    int i;

    printf("Enter number: ");
    scanf("%i", &i);
    printf("i=%x\n", i);
    printf("i>>5=%x\n", i>>5);
    printf("Bits 5-7: %x\n",
           (i >> 5) & 07);
    return 0;
}
```