Bit Manipulation

CS 2060

Prof. Jonathan Ventura

Bitwise Operators

- Bitwise operators allow us to test and manipulation the bits in variables.
- 0b indicates a binary number, e.g. 0b0100 is the binary string 0100.
- The bits are numbered from right to left, so that the right-most bit is the "first" bit.
- For example, the third bit of 0100 is a 1.

Bitwise operators

Operator	Description
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR (XOR)
<<	left shift
>>	right shift
~	complement

Table: Bitwise operators in C

AND and OR

- Bitwise AND and OR make a bit string by comparing the individual bits in two bit strings.
- Bitwise AND (&) outputs a bit string with a 1 for each bit which is a 1 in both input strings, and 0 otherwise:

```
char a = 0b00001111;
char b = 0b11111000;
char c = a & b;      // = 0b00001000
```

- Be careful not to confuse & and &&!
- & is bitwise AND (returns a bit string).
- && is logical AND (returns 0 or 1).

AND and OR

- Bitwise OR (|) outputs a bit string with a 1 for each bit which is a 1 in *either* input string, and 0 otherwise:

```
char a = 0b00001111;
char b = 0b11111000;
char c = a | b;      // = 0b11111111
```

- Again, don't confuse | and ||!
- | is bitwise OR (returns a bit string).
- || is logical OR (returns 0 or 1).

Bit flags

- Bitwise AND and OR are useful for making strings of binary flags.
- Each bit indicates an option which can be set true or false.

- For example, in an image processing application, we might use bit flags to indicate filter options:
  - The first bit could indicate whether the image should be flipped vertically
  - The second bit could indicate whether the image should be flipped horizontally
  - The third bit could indicate whether the image should be resized
  - And so on...

Bit flag example

```
enum FilterOptions {
    FLIP_VERTICAL = 1,   // = 0b0001
    FLIP_HORIZONTAL = 2, // = 0b0010
    RESIZE = 4,          // = 0b0100
    // ...
}
```

Bit flag example

```
int processImage( Image *im, char options )
{
    // Test option flags using logical AND
    if ( options & FLIP_VERTICAL )
        // do vertical flip
    if ( options & FLIP_HORIZONTAL )
        // do horizontal flip
    if ( options & RESIZE )
        // do resize
}
```

Bit flag example

```
// Specify options using logical OR
char options = ( FLIP_VERTICAL | RESIZE ); // = 0b0101
processImage( im, options );
```

Bit Shifting

- The bit shift operators *shift* a bit string right or left by a specified number of bits:

```
char a = 0b00001111;// = 15
char b = a << 1;    // = 0b00011110 = 30
char c = b << 3;    // = 0b11110000 = 240
char d = a >> 2;    // = 0b00000011 = 3
```

- Shifting to the left by n bits is the same as multiplying by $2n$.
- Shifting to the right by n bits is the same as dividing by $2n$.

Bit Shifting

- Bit shift is useful to make a bit string with a 1 at a specific bit:

```
char a = 1 << 4;     // = 0b00010000
```

- We can use this to test whether a particular bit is one or zero:

```
// test whether fifth bit of flags is 1
if ( flags & ( 1 << 4 ) )
{
    // ...
}
```

Bit Shifting

- We can also use bit shifting to look at portion of a bit string.
- Here we print each byte of a 4-byte integer:

```
unsigned int a = rand();
printf( "bytes in %u:\n", a );
for ( int i = 0; i < 4; i++ )
{
    // get first byte of a
    unsigned char b = a & 255;
    printf( "byte %d: %d\n, b );
    a >>= 8; // shift down to get next byte
}
```

Complement

- The complement operator ~ flips zeros to ones and ones to zeros:

```
char a = 0b00001010; // = 10
char b = ~a;         // = 0b11110101 = 245
```

- As you might guess from the example, $\sim a = 255 - a$ in 8-bit unsigned arithmetic.