CSC / CPE 357

Systems Programming

Integer Data Types in C

```
#include <stdint.h>
```

	Signed	Unsigned	Maximum Value (Unsigned)
?-Bit Integer (*at least* 16 bits according to C specification)	`int`	`unsigned int`	?
8-bit Integer	`int8_t`	`uint8_t`	`UINT8_MAX` ($2^8 - 1$)
16-bit Integer	`int16_t`	`uint16_t`	`UINT16_MAX`
32-bit Integer	`int32_t`	`uint32_t`	`UINT32_MAX`
64-bit Integer	`int64_t`	`uint64_t`	`UINT32_MAX`

Number Representation in C

- In C, values can be specified in octal or hexadecimal instead of decimal.
- Leading `0` (zero) on an integer constant means octal
  - `int i = 037; // octal 37, decimal 31`

- Leading 0x or 0X means hexadecimal
  - `int i = 0x1f; // hexadecimal 1f, decimal 31`

- Octal and hexadecimal constants may also be followed by:
  - `L` for long
  - `U` for unsigned
  - `0XFUL` is an unsigned long constant with value 15 decimal.

Handy Conversion Tool

Bitwise Operators in C

C provides six operators for bit manipulation; these may only be applied to integral operands: `char`, `short`, `int`, and `long` (`signed` or `unsigned`)

Symbol	Operation
&	bitwise AND
|	bitwise OR
^	bitwise XOR
~	bitwise NOT
<<	left shift
>>	right shift

Bitwise vs Logical Operators

- Easy to confuse two different sets of operators in C:
  - Bitwise operators: `&` and `|`
  - Logical operators: `&&` and `||` (logical AND, logical OR)
    - Left-to-right evaluation of a truth value
- If x is 1 and y is 2:
  - `x & y` is zero
  - `x && y` is one

Bitwise AND

The bitwise AND operator & is often used to mask bits

```
n = n & 0177;  // octal 177, dec 127, hex 0x7f, bin 01111111
```

sets to zero all but the low-order 7 bits of n.

		A & B
0	1	0
0	0	0
1	1	1
1	0	0

Bitwise OR

The bitwise OR operator `|` is used to turn bits on:

```
x = x | SET_ON;
```

sets to one in `x` the bits that are set to one in `SET_ON`.

		A | B
0	1	1
0	0	0
1	1	1
1	0	1

Bitwise XOR

The bitwise exclusive OR operator `^` sets a one in each bit position where its operands have different bits, and zero where they are the same.

		A ^ B
0	1	1
0	0	0
1	1	0
1	0	1

Bitwise NOT

The unary operator ~ yields the one's complement of an integer -- converts each 1-bit into a 0-bit and vice versa.

For example:

```
x = x & ~077
```

sets the last six bits of x to zero

Shift Operators

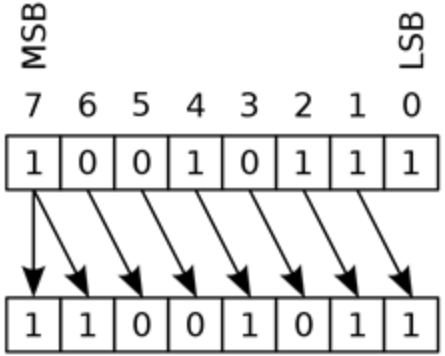
- The shift operators << and >> perform left and right shifts
  - Number of bit positions given by the right operand, must be non-negative.

- For example, `x << 2` shifts the value of x to the left by two bits
  - Vacated bits filled with zero
  - `x << 2` is equivalent to multiplication by 4

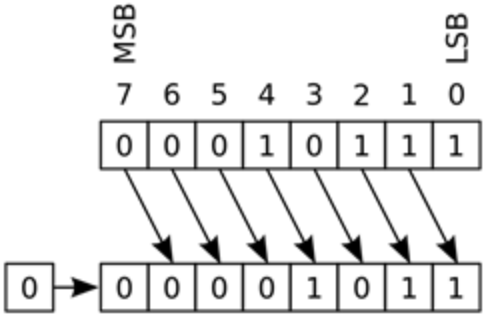
- Right shifting *unsigned* value always fits the vacated bits with zero.
- Right shifting *signed* value is implementation-dependent:
  - Fill with bit signs ("arithmetic shift") on some machines/compilers (x86/gcc)
  - Fill with 0 ("logical shift") on others

Shift Operators: Arithmetic vs Logical

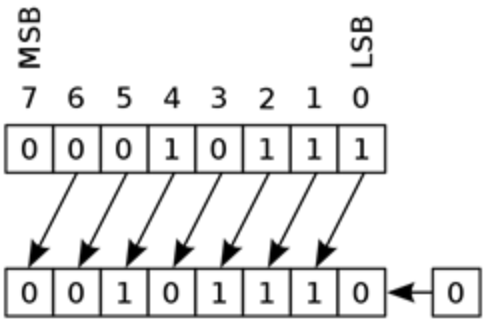
- **Arithmetic shift**: sign bit copied to preserve sign
- **Logical shift**: zeros are shifted in to replace the discarded bits



Arithmetic Right Shift



Logical Right Shift



Left Shift (Logical and Arithmetic are Identical)

Images from Wikimedia Commons

Negative Numbers

- In the C programming language, signed integers are represented using **Two's Complement**.

- Most significant bit of the number (also called MSB) is used as the sign
  - 0 for positive numbers
  - 1 for negative numbers.

- Remaining bits are used to store the number's value.

- 0 is considered a positive number

Two's Complement

- We create a Two's Complement representation of a negative number by inverting all bits (recall the ~ operator) of the positive value, then adding 1.

- For example, binary representation of -47 (decimal):
  - Binary representation of *positive* 47: `00101111`
  - Flip all bits: `11010000`
  - Add 1: `11010001`

- This representation limits our range of values
  - Maximum positive value signed `int8_t` can hold is: $2^7 - 1 = 127$
  - Minimum value: -2^7 (since 0 is considered positive)

Bit Masking

Bit masking refers to operations like the following:

- Set a specific bit to a value (0 or 1)
- Clear a specific bit, or a sequence of bits
- Invert the values of all bits

Example: File Permissions

Recall `chmod` (change file permissions):

```
#include <sys/stat.h>

int chmod(const char *pathname, mode_t mode);
```

The `mode` argument is a bitmask. To change permissions, there are several defined constants that can be applied along with bitwise operators.

Constant	Description	Effect on regular file	Effect on directory
S_ISUID	set-user-ID	set effective user ID on execution	(not used)
S_ISGID	set-group-ID	if group-execute set, then set effective group ID on execution; otherwise, enable mandatory record locking (if supported)	set group ID of new files created in directory to group ID of directory
S_ISVTX	sticky bit	control caching of file contents (if supported)	restrict removal and renaming of files in directory
S_IRUSR	user-read	user permission to read file	user permission to read directory entries
S_IWUSR	user-write	user permission to write file	user permission to remove and create files in directory
S_IXUSR	user-execute	user permission to execute file	user permission to search for given pathname in directory
S_IRGRP	group-read	group permission to read file	group permission to read directory entries
S_IWGRP	group-write	group permission to write file	group permission to remove and create files in directory
S_IXGRP	group-execute	group permission to execute file	group permission to search for given pathname in directory
S_IROTH	other-read	other permission to read file	other permission to read directory entries
S_IWOTH	other-write	other permission to write file	other permission to remove and create files in directory
S_IXOTH	other-execute	other permission to execute file	other permission to search for given pathname in directory

Figure 4.26 Summary of file access permission bits

Example: The Lonely Integer

Given an array of integer values, where all elements except one occur twice, find the unique element, the so-called lonely integer.

For example, if `L={1,2,3,3,8,1,9,2,9}` the unique element is 8, because the rest of the elements come in pairs.

Example: getbits()

`getbits(x,p,n)` returns the n-bit field of x that begins at position p.

Assume:

- bit position 0 is at the right end
- n and p are positive values, within range

For example, `getbits(x,4,3)` returns the three bits in positions 4, 3 and 2 from the right