

Programming with C I

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

Binary

- 🛡️ Computers represent everything as bits
- 🛡️ Recall: a byte is 8 bits
- 🛡️ Int: 4 bytes (32 bits)
- 🛡️ **What's the largest int we can represent?**

$2^{32} - 1$

(unsigned)

Hexadecimal (base 16)

- Binary takes up a lot of space
- Hexadecimal takes few digits but can easily be converted to binary (and vice versa)
 - Hex uses digits 0-9 and a-f
 - 1 hex digit = 4 bits
- 0000 0000 0000 0001 1101 0011 0101 1011
- 1d35b

Format ints

- %d for decimal
- %x for hex

Assign ints

- 0b for binary (ex: 0b11011 is 27)
- 0x for hex (ex: 0x83fa9 is 540585)

Bitwise Operators

 You know logical operators...&&,||,!

 We will now learn &,|,~,^,<<,>>

 These operate at the bit level

Table

&

a	b	a & b
1	1	1
0	0	0
0	1	0
0	0	0

Table

|

a	b	a b
1	1	1
0	0	0
0	1	1
0	0	0

Table

^

a	b	$a \wedge b$
1	1	0
1	0	1
0	1	1
0	0	0

Table

~

a	$\sim a$
1	0
0	1

Operators on multiple bits

AND

```
  0110
& 1100
----
  0100
```

OR

```
  0110
| 1100
----
  1110
```




XOR

```
  0110
^ 1100
----
  1010
```

NOT

```
~ 1100
----
  0011
```

Bitmasks

-  We often want to manipulate or isolate specific bits from a collection
-  A **bitmask** is a bit pattern that achieves this
-  We can use and/or create bitmasks using bitwise operators

Example: CSCI courses

 **Array of ints vs. storing bits**

 **Bitmasks**

- Setting bits to 1 with |
- Setting bits to 0 with &
- “Masking off” unwanted bits

 **But how do we mask an arbitrary position?**

<< and >>

 << k shifts x left by k

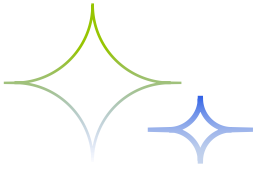
00110111 << 2 results in 11011100

01100011 << 4 results in 00110000

10010101 << 4 results in 01010000

 x >> k shifts x right by k

 Careful with unsigned ints for >>



THE END

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu