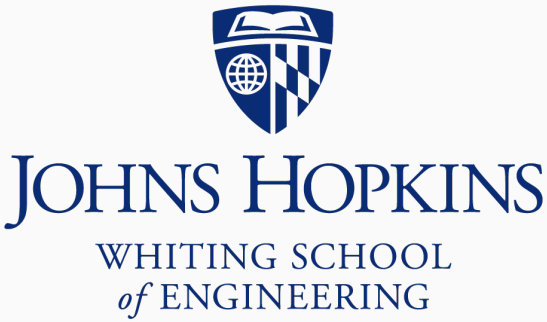
Bitwise operators

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Bitwise operators

We saw that integers can be used as *boolean* values with 0 = false and non-0 = true

Also saw logical operators for combining booleans

Operator	Function	Example	Result
`&&`	Both true (*AND*)	`1 && 0`	false (0)
`||`	Either true (*OR*)	`1 || 0`	true (non-0)
`!`	Opposite (*NOT*)	`!(1 || 0)`	false (0)

Bitwise operators

Saw that integer types consist of bits

Each bit could be considered a boolean true/false value

Binary:	0	0	1	1	0	1	0	1
Place value:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$2^5 + 2^4 + 2^2 + 2^0 = 32 + 16 + 4 + 1 = 53$$

Bitwise operators

Bitwise operators performs a function across all bits in integer operands, treating them as boolean true/false values

Bitwise AND (&) performs logical AND (&&) across all bits:

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

Bitwise operators

```c
#include <stdio.h>
int main() {
    int a = 12;
    int b = 25;
    printf("%d & %d = %d\n", a, b, a & b);
    return 0;
}
```

```
$ gcc bitwise_and.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
12 & 25 = 8
```

Bitwise operators

Since `int`s are 32-bit (4-byte) values, this is a more accurate picture:

```
  00000000000000000000000000001100  = 12
& 00000000000000000000000000011001  = 25

  --------------------------------
  00000000000000000000000000001000  = 8 (In decimal)
```

Bitwise operators

Bitwise OR (|) performs logical OR (||):

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

Bitwise operators

```c
#include <stdio.h>
int main() {
    int a = 12;
    int b = 25;
    printf("%d | %d = %d\n", a, b, a | b);
    return 0;
}
```

```
$ gcc bitwise_or.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
12 | 25 = 29
```

Bitwise operators

`x << n` shifts bits of `x` the left `N` positions

`N` 0s are "shifted in" at right-hand side

`N` bits "fall off" left-hand side

```
25 = 00011001 (In Binary)

Bitwise left-shift of 25 by 5 positions (25 << 5)
     11001
      << 5
----------
1100100000 = 800 (In decimal)
```

Bitwise operators

```c
#include <stdio.h>
int main() {
    int a = 25;
    int b = 5;
    printf("%d << %d = %d\n", a, b, a << b);
    return 0;
}
```

```
$ gcc bitwise_lshift.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
25 << 5 = 800
```

Bitwise operators

Similar for bitwise right shift (>>)

```
25 = 00011001 (In Binary)

Bitwise right-shift of 25 by 4 positions (25 >> 4)
  00011001
      >> 4
  --------
  00000001  = 1
```

Bitwise operators

```c
#include <stdio.h>
int main() {
    int a = 25;
    int b = 4;
    printf("%d >> %d = %d\n", a, b, a >> b);
    return 0;
}
```

```
$ gcc bitwise_rshift.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
25 >> 4 = 1
```

Bitwise operators

```c
#include <stdio.h>
int main() {
    int num = 53;
    char bin_str[33] = {'\0'};
    int tmp = num;
    for(int i = 0; i < 32; i++) {
        if((tmp & 1) != 0) {     // least significant bit set?
            bin_str[31-i] = '1'; // prepend 1
        } else {
            bin_str[31-i] = '0'; // prepend 0
        }
        tmp >>= 1;               // shift right by 1
    }
    printf("%d in binary: %s\n", num, bin_str);
    return 0;
}
```

Bitwise operators

```
$ gcc bitwise_convert.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
53 in binary: 00000000000000000000000000110101
```