Bitwise Operators Treat Numbers as Bits

Declare: `int A = 120;  int B = 42;`

```
/* A = 0x00000078, B = 0x0000002A
using C's notation for hexadecimal. */
```

Then…

`A & B` evaluates to `40  0x00000028`

```
    0000 0000 0000 0000 0000 0000 0111 1000
AND 0000 0000 0000 0000 0000 0000 0010 1010
    0000 0000 0000 0000 0000 0000 0010 1000
```

Apply AND to pairs of bits.

Bitwise Operators Treat Numbers as Bits

Declare: `int A = 120;  int B = 42;`

```
/* A = 0x00000078, B = 0x0000002A
using C's notation for hexadecimal. */
```

Then…

`A & B`	evaluates to	`40`	`0x00000028`
`A | B`	evaluates to	`122`	`0x0000007A`
`~A`	evaluates to	`-121`	`0xFFFFFF87`
`A ^ B`	evaluates to	`82`	`0x00000052`

Left Shift by N Multiplies by 2^N

Shifting left by N bits adds **N** 0s on right.
- It's like **multiplying by 2^N**.
- **N** bits lost on left! (**Shifts can overflow.**)

Declare: `int A = 120;/* 0x00000078 */`
`unsigned int B = 0xFFFFFF00;`

Then…

`A << 2`	evaluates to	`480`	`0x000001E0`
`B << 4`	evaluates to	`(<B!)`	`0xFFFFF000`

Right Shift by N Divides by 2^N

A question for you: **What bits appear on the left when shifting right?**

Declare: `int A = 120;/* 0x00000078 */`

`A >> 2` evaluates to `30  0x0000001E`

What about `0xFFFFFF00 >> 4`?

Is `0xFFFFFF00` equal to

`-256` (/16 = `-16`, so insert 1s)? or equal to

`4,294,967,040` (/16 = `268,435,440,` insert 0s)?

