**Bit Manipulation**

**Power of Bit Shifting**

* If we left shift (<<) a number by 1, then the number gets multiplied by 2. For example: Consider num: 5,

num << 1 modifies num to 2 (Note: Floating precision is discarded)

* If we right shift (>>) a number by 1, then the number gets divided by 2

For example: consider num: 5,

num >> 1 modifies num to 10

**NOTE**

* Applying bit shifting operations instead of performing multiply and division by 2 is more Performant and can be helpful in competitive cases where micro delay in execution can cause problems and havoc.
* Powerful Bit Technique of Performing **2 raised to the power x** :

1 << n

**C++ function to count the number of 1’s bits in an int**

Syntax: \_\_builtin\_popcount(x)

E.g

result = \_\_builtin\_popcount(5); // result will be 2

**Check Least Significant Bit is 1 or not**

* Num & 1 : Do **Bitwise and** on a number with integer 1 whose binary value is 0001. On Performing **Bitwise And** the output of a single bit evaluates to 1 if both bits are 1 only.
* For eg. 5 & 1 == 1

5 => 0101

1 => 0001

* Quite useful technique for Bit Masking

**Unsigned int 8-bit**

uint8\_t - Equivalent to unsigned char, Consume 1 byte

**Unsigned int 16-bit**

uint16\_t - Equivalent to unsigned short, Consume 2 bytes

**Unsigned int 32-bit**

uint32\_t - Equivalent to unsigned int, Consume 4 bytes

**Unsigned int 64-bit**

uint64\_t - Equivalent to unsigned long long, Consume 8 bytes