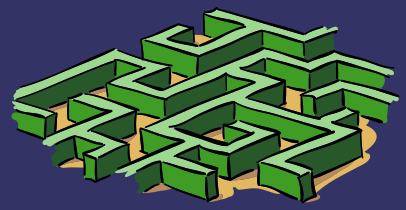
'C' Bitwise Operators

Relevance to 'C' of bitwise applications

Syntax and expressions

Example getter and setter functions

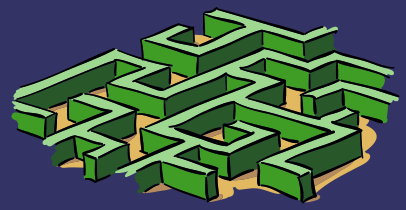
Eratothene's prime number sieve



Relevance to 'C' of Bitwise Applications

'C' was designed to write system software as an alternative to assembler: compilers, kernels, device drivers, interpreters, relational database engines, virtual machines.

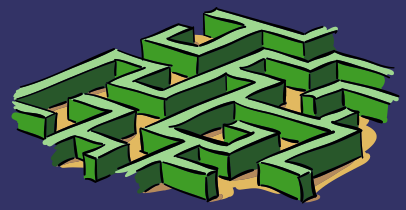
So this language needs access to raw hardware and individual bit values. Coding designed for specific hardware design features will be non portable.



Portability Constraints

Different microprocessors organise integer and floating point data differently, e.g. big endian or little endian, two's or one's complement, location and size of exponent, sign bit and mantissa.

Device drivers for different hardware implement different instruction sets.



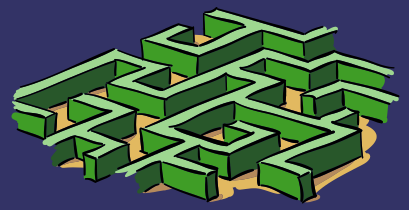
Operator Summary

Summary of Bitwise Operators

operator	coding	notes
left shift	<<	shifts LHS left by no. of bits in RHS
right shift	>>	shifts LHS right by no. of bits in RHS
bitwise AND	&	returns parrallel and of LHS and RHS input bits
bitwise OR	|	returns parrallel or of LHS and RHS input bits
exclusive OR	^	returns 1 if LHS and RHS bits the same, 0 if different
twos complement	~	Unary NOT operator. Returns 0 for RHS 1 and 1 for 0.

Bitwise Operators

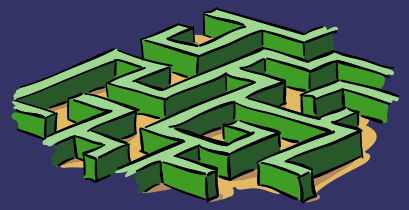
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
>>	Shift Right
<<	Shift left
&=	Bitwise AND Assignment
|=	Bitwise OR Assignment
^=	Bitwise XOR Assignment
>>=	Shift Right Assignment
<<=	Shift Left Assignment



Bitwise Operators

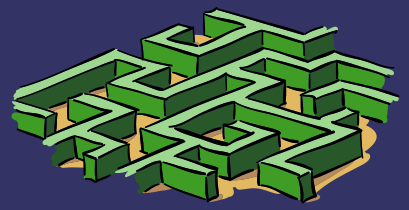
Applied to integer type – long, int, short, byte and char.

A	B	A | B	A & B	A ^ B	! A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0



Bitwise Operators

EXAMPLE	MEANING	EXPLANATION	EXAMPLE	RESULT
~	Bitwise unary NOT	This operator is used to invert all the bits	~42	213
&	Bitwise AND	Produce a 1 bit if both operands are also 1 otherwise 0	2 & 7	2
|	Bitwise OR	either of the bits in the operands is a 1, then the resultant bit is a 1 otherwise 0	2 | 7	5
^	Bitwise exclusive OR	if exactly one operand is 1, then the result is 1. Otherwise, the result is zero	2 ^ 7	7



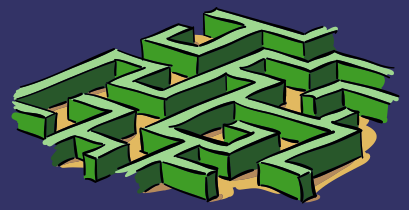
Bitwise Operators

EXAMPLE	MEANING	EXPLANATION	EXAMPLE	RESULT
>>	Shift right	The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.	7>>2	1
<<	Shift left	The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times.	2 << 2	8
|=	Bitwise OR assignment	either of the bits in the operands is a 1, then the resultant bit is a 1 otherwise 0	a=2 a&=2	a=2
&=	Bitwise AND assignment	if exactly one operand is 1, then the result is 1. Otherwise, the result is zero	a=2 a|=2	a=2

Bitwise Operators

- bitwise operators operate on individual bits of integer (int and long) values.
- If an operand is shorter than an int, it is promoted to int before doing the operations.
- Negative integers are stored in two's complement form.

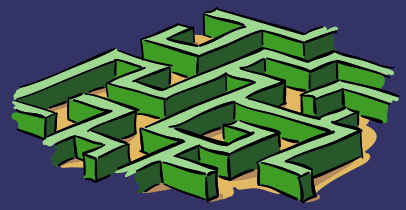
For example, -4 is:
1111 1111 1111 1111 1111 1111 1111 1100.



In-place operators

Inplace versions of operators exist which modify the LHS operand in place rather than returning the result for a seperate assignment, e.g. a >>= b performs a right shift of b bits directly on a .

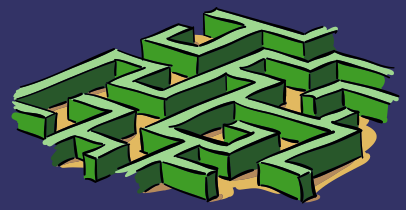
These work in the same manner as += , = and *= in-place operators compared to + and * .



Left and Right Shift Operators

The >> operator shifts a variable to the right and the << operator shifts a variable to the left. Zeros are shifted into vacated bits, but with signed data types, what happens with sign bits is platform dependant.

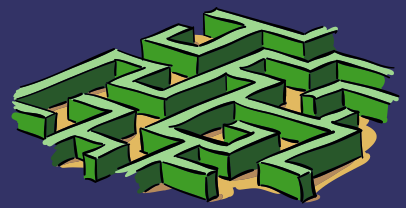
The number of bit positions these operators shift the value on their left is specified on the right of the operator. Uses include fast multiplication or division of integers by integer powers of 2, e.g. 2,4,8,16 etc.



Left and right shift example

```
#include <stdio.h>
int main(void){
        unsigned int a=16;
        printf("%d\t",a>>3); /* prints 16 divided by 8  */
        printf("%d\n",a<<3); /* prints 16 multiplied by 8 */
        return 0;
}
```

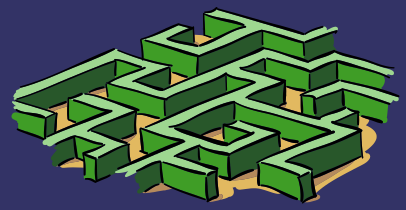
output: 2 128



Bitwise AND and inclusive OR

Single & and | operators (bitwise AND and OR) work differently from logical AND and OR (&& and ||). You can think of the logical operators as returning a single 1 for true, and 0 for false.

The purpose of the & and | bitwise operators is to return a resulting set of output 1s and 0s based on the boolean AND or OR operations between corresponding bits of the input.

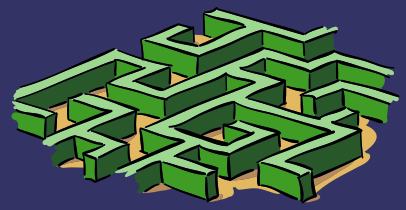


Bitwise AND/OR example

```
#include <stdio.h>
int main(void){
        unsigned char a='\x00',b='\xff',c;
        c='\x50' | '\x07'; /* 01010000 | 00000111 */
        printf("hex 50 | 07 is %x\n",c);
        c='\x73' & '\x37'; /* 01110011 & 00110111 */
        printf("hex 73 & 37 is %x\n",c);
        return 0;
}
```

Output:

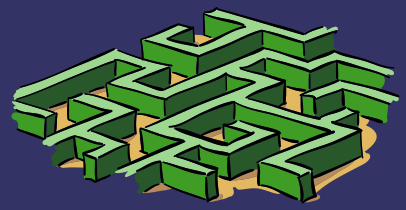
```
hex 50 | 07 is 57
hex 73 & 37 is 33
```



Bitwise exclusive OR operator

Symbol: ^

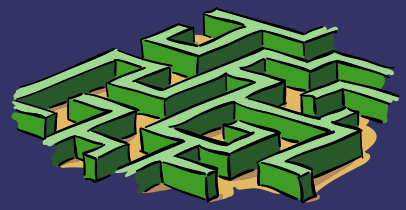
For each bit of output, this output is a 1 if corresponding bits of input are different, and the output is a 0 if the input bits are the same.



Bitwise complement operator ~

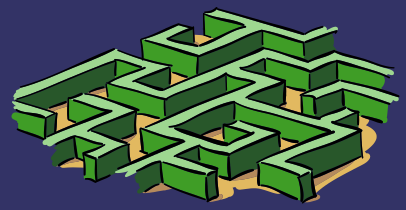
Symbol: ~

This is a unary operator in the sense that it works on a single input value. The bit pattern output is the opposite of the bit pattern input with input 1`s becoming output 0`s and input 0`s becoming output
1`s.



2's Complement Operator

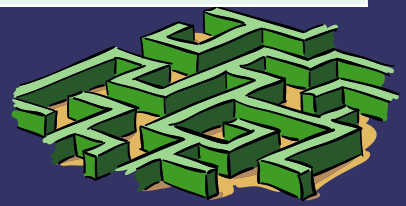
- Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.
- 35 = 00100011 (In Binary) Bitwise complement Operation of 35 ~ 00100011 _____ 11011100 = 220 (In decimal). Twist in bitwise complement operator in C Programming.
- The bitwise complement of 35 (~35) is -36 instead of 220, but why?



2’s Complement Operator

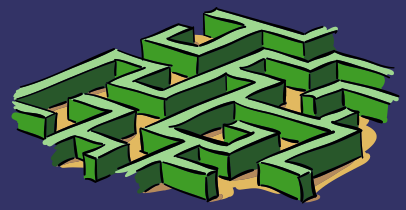
- For any integer **n**, bitwise complement of **n** will be -(n+1). To understand this, you should have the knowledge of 2's complement.
- Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1.
  For example:

Decimal	Binary	2's complement
0	00000000	-(11111111+1) = -00000000 = -0(decimal)
1	00000001	-(11111110+1) = -11111111 = -256(decimal)
12	00001100	-(11110011+1) = -11110100 = -244(decimal)
220	11011100	-(00100011+1) = -00100100 = -36(decimal)



2’s Complement Operator

- Note: Overflow is ignored while computing 2's complement. The bitwise complement of 35 is 220 (in decimal). The 2's complement of 200 is -36. Hence, the output is -36 instead of 220.
- Bitwise complement of any number N is -(N+1). Here's how:
- bitwise complement of N = ~N (represented in 2's complement form) 2'complement of ~N= -(~(~N)+1) = -(N+1)



2’s Complement Operator