An **operator** is a character that symbolizes an arithmetic or logical operation. The objects on which the operator works are called the **operands**. For example, in the C statement z = x + y; the operation is addition, the operator is the + sign and the operands are x and y.

Operators in C have an **order of precedence**, which means that they are always performed in a certain order, unless otherwise specified. For example, * has a higher precedence that +, so the statement z=2*3+1 will evaluate to 7, since2*3=6 and 6+1=7. The statement z=2*(3+1) will evaluate to 8, because 3+1=4 and 2*4=8. There is a table in Appendix B on pages 477-478 in your textbook that shows the order of precedence for all C operators.

Arithmetic Operators

Operator	Name	Example	Result
+	Addition	x=3+2;	x=5
-	Subtraction	x=3-2;	x=1
*	Multiplication	x=3*2;	x=6
/	Division	x=12/3; y=9.0/12.0; (for float y) z=9/12; (for int z)	x=4 y=0.75 **z=0 (Be careful! Integer division does not round up.)**
%	Modulus (or Remainder)	x=24%3; y=26%3;	x=0 (24/3=8, with no remainder) y=2 (26/3=8 with a remainder of 2)

Compound Operators (start with int x = 15;)

Operator	Example	Equivalent	Result
+=	x += 2;	x = x+2;	x=15+2=17
-=	x -=y;	x = x-y;	x=15-2=13
*=	x *=8;	x = x*8;	x=15*8=120
/=	x /= 2;	x = x/2;	x=15/2=7
%=	x %=2;	x = x%2;	x=15%2=1

Shift Operators

Operator	Name	Example	Result
>>	Right shift	x = 0x40; y = x >> 5;	Shift x 5 bits to the right. Fill with 0s. x= 0x40 = 01000000 y= 00000010 = 0x02
<<	Left shift	x = 0x01; y = x << 3;	Shift x 3 bits to the left. Fill with 0s. x= 0x01 = 00000001 y= 00001000 = 0x08

Relational Operators (always evaluates to a true or false condition)

Operator	Name	Example	Result
==	Equal to	if(x==3)statement;	Statement executes if () is true
!=	Not equal to	if(x!=3)statement;	
<	Less than	if(x<3)statement;	
>	Greater than	if(x>3)statement;	
<=	Less than or equal to	if(x<=3)statement;	
>=	Greater than or equal to	if(x>=3)statement;	

Unary Operators

Operator	Name	Example	Result
!	Logical negation	if(!x)statement;	Execute statement if x is false
~	Ones complement	leds = \b00110101; PAOUT = ~leds;	Invert all bits of leds and send new value to PAOUT.
-	Arithmetic negation	y=5; x= -y;	x= -5 (= minus five)
++	Increment: Post-increment	x=5; y=3*x++;	y=3*5=15 (multiply) x=6 (increment **after** multiply)
++	Increment: Pre-increment	x=5; y=3*++x;	Increment x **before** multiply y= 3*6=18
--	Decrement: Post-idecrement	x=5; y=3*x--;	y=3*5=15 (multiply) x=4 (decrement **after** multiply)
--	Decrement: Pre-decrement	x=5; y=3*--x;	Decrement x **before** multiply y= 3*4=12

Bitwise Logical Operators (starting with x=0x55; y=0x0F;)

Operator	Name	Example	Result
~	NOT	z = ~x;	x = 0x55 = 01010101 z = 10101010 = 0xAA
&	AND	z = x & y;	x = 0x55 = 01010101 y = 0x0F = 00001111 z = 00000101 = 0x05
|	OR	z = x | y;	x = 0x55 = 01010101 y = 0x0F = 00001111 z = 01011111 = 0x5F
^	XOR	z = x^y;	x = 0x55 = 01010101 y = 0x0F = 00001111 z = 01011010 = 0x5A

Logical Relational Operators (always evaluates to a true or false condition)

Operator	Name	Example	Result
&&	Logical AND	if(x && 0x0F)statement;	Statement executes if () is true (non-zero)
||	Logical OR	if(x || 0x0F)statement;	