CHAPTER 3 *(DAY 1 After noon)*

OPERATORS AND PRECEDENCE **(Chapter Reading)**
OPERATORS AND OPERANDS (Reading)
PRECEDENCE OF OPERATION (Reading)
PRECEDENCE OF ARITHMETIC EXPRESSION (Reading)
PRE AND POST INCREMENT/DECREMENT (Reading)
PRECEDENCE OF C OPERATORS (Reading)
TRUTH TABLES FOR COMPOUND CONDITION (Reading)

CHAPTER – 3

OPERATORS AND PRECEDENCE (Reading)

OPERATORS AND OPERANDS

Based on number of **operands**, which have values, **operators** are categorized into three groups. ***Unary operators*** need only one **operand**; ***binary operators*** need two **operands**, while ***ternary operators*** need three **operands** to carry out the desired operation on the **operands**.

Unary operators: unary +, unary -, !, ++, --, ~

Binary operators: arithmetic +, arithmetic -, *, /, %, =, &&, | |
and relational operators

C provides the special **Unary Operator *sizeof*** to determine the size in bytes of an array or any other data type during program compilation. When applied to the name of an array the ***sizeof*** operator returns the total number of bytes in the array as an integer. Operator ***sizeof*** can be applied to any variable, type, or constant. When applied to a variable name or a constant, the number of bytes used to store the specific type of variable or constraint is returned. The number of bytes used to store any particular data type may vary between systems. Use *sizeof* **operator** to determine the number of bytes used to store the data types.

C provides the ***conditional operator*** (**? :**) which is closely related to the **if/else** structure. The **conditional operator** is *C*'s only ***ternary operator*** takes three operands. The **operands** with the **conditional operator** form a ***conditional expression***. The first operand is a condition, the second **operand** is the value for the entire **conditional expression** if the condition is true, and the third **operand** is the value for the entire **conditional expression** if the condition is false. The values in a **conditional expression** can also be actions to be executed.

PRECEDENCE OF OPERATION

Mathematical Operators in order of precedence:

Unary + -	right to left
( )	left to right
* / %	left to right
+ -	left to right

Relational Operators in order of precedence:

< <= > >=	left to right
== !=	left to right
=	right to left

Logical Operators in order of precedence:

!	right to left
&&	left to right
| |	left to right
? :	right to left

Bit wise Operators in order of precedence:

&	bit wise AND
|	bit-wise OR
^	bit-wise exclusive OR
<<	left shift
>>	right shift
~	one’s complement

Increment Operators in order of precedence:

++ --	right to left

PRECEDENCE OF ARITHMETIC EXPRESSION

Parenthesis take highest precedence of operation.

y = m * x + b

is not equal to m (x + b),
with the precedence of operation the statement becomes
(m * x) + b

```
       a + b + c + d + e
m =   ---------------------------
                5
```

Cannot be assumed as: a + b + c + d + e/5

With the precedence of operation the expression is evaluated as:

(a + b + c + d + e) / 5

Similarly, z = pr%q+w/x – y
is evaluated in the order of:

```
z  =  p * r % q + w / x  –  y
        1   2   4   3   5     ---- order of execution
```

The following expression
a * (b + c) + c * (d - e) / f

cannot be evaluated as:
a * b + c + c * d - e / f

the correct evaluation is:

```
a * (b + c) + c * (d - e) / f
  3    1    6   4   2    5    ----- order of execution
```

PRE AND POST INCREMENT/DECREMENT

int a = 0, b = 5;

a = a +1;
This can be written in post increment statement as
a++;
or
a +=1;
b = a++; value of b is 1 where as value of a is 2.
b = ++a; value of b is 3 and value of b is 3.

Similarly,
b = b -1;
Can be written as
b--;
or
b - =1;
b = ++a; value of b is 4, value of a is 4.
b = a--; value of b is 4, value of a is 3.
b--;
b = --a; value of b is 2, value of a is 2.

int a =5, b = 5, c = 5, d = 5;

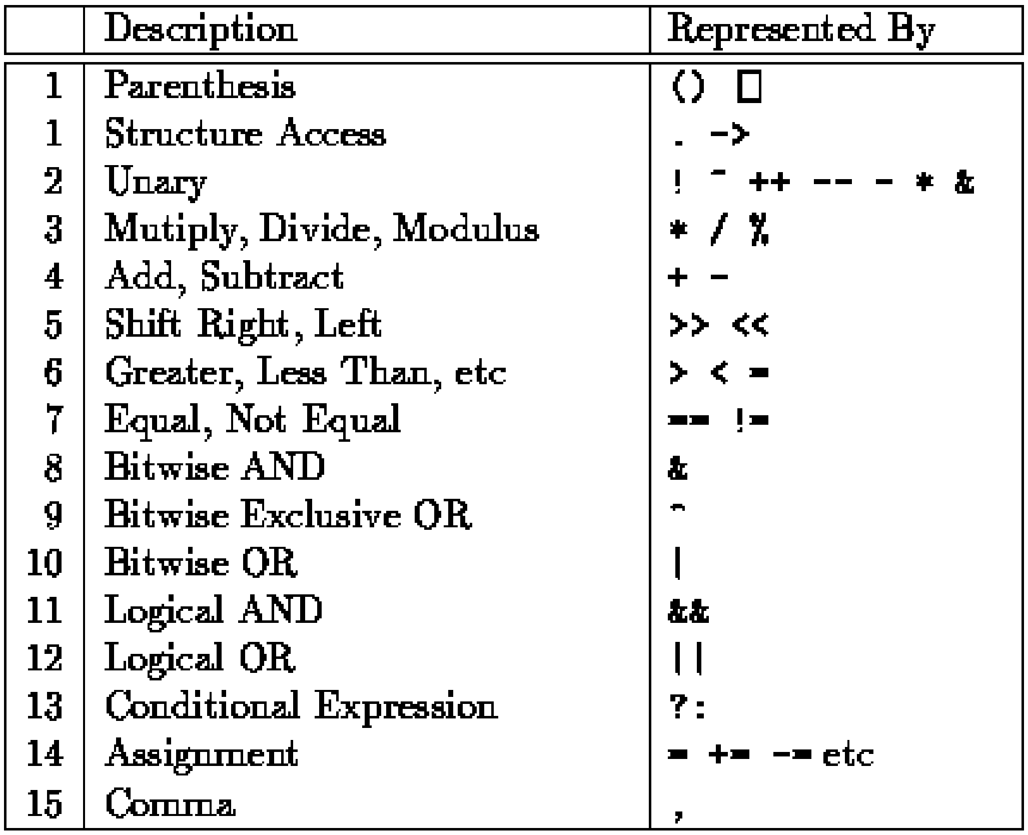
printf (“value of a: %d, and value of b: %d. \n”, a++, --b);
printf (“values: a is %d, b is %d, c is %d, d: %d. \n”, a++, --b, c--, ++d);

The output is:

value of a: 5, and value of b: 4.
values: a is 6, b is 3, c is 5, d: 6.

PRECEDENCE OF C OPERATORS

The following table shows the precedence of operators in C. Where a statement involves the use of several operators, those with the lowest number in the table will be applied first.



	Description	Represented By
1	Parenthesis	() []
1	Structure Access	. ->
2	Unary	! ~ ++ -- - * &
3	Mutiply, Divide, Modulus	* / %
4	Add, Subtract	+ -
5	Shift Right, Left	>> <<
6	Greater, Less Than, etc	> < =
7	Equal, Not Equal	== !=
8	Bitwise AND	&
9	Bitwise Exclusive OR	^
10	Bitwise OR	|
11	Logical AND	&&
12	Logical OR	||
13	Conditional Expression	?:
14	Assignment	= += -= etc
15	Comma	,

TRUTH TABLES FOR COMPOUND CONDITIONS

Truth table for && (and) operator:

Bool 1	**oper**	**Bool 2**	**result**
True	&&	True	True
True	&&	False	False
False	&&	True	False
False	&&	False	False

Truth table for | | (or) operator:

Bool 1	**oper**	**Bool 2**	**result**
True	||	True	True
True	||	False	True
False	||	True	True
False	||	False	False

Truth table for ! (not) operator:

Bool	**oper**	**result**
True	!	False
False	!	True

Operators and **operands** can be compounded; the result will take place in the order of precedence on different **operators**. Parentheses need to be used when in doubt or required results are to be obtained. **Boolean operators** can be applied similar to **arithmetic operators**. Best way to get the result of compounded **Boolean operators** and **operands** is to break down the conditions based on the order of precedence and apply the result with the next **operator** and **operand**.