LECTURE NOTE-9

INCREMENT AND DECREMENT OPERATOR

In C, ++ and – are called increment and decrement operators respectively. Both of these operators are unary operators, i.e, used on single operand. ++ adds 1 to operand and – subtracts 1 to operand respectively. For example:

Let a=5 and b=10

a++; //a becomes 6

a--; //a becomes 5

++a; //a becomes 6

--a; //a becomes 5

When i++ is used as prefix(like: ++var), ++var will increment the value of var and then return it but, if ++ is used as postfix(like: var++), operator will return the value of operand first and then only increment it. This can be demonstrated by an example:

```
#include <stdio.h>
int main()
{
 int c=2,d=2;
printf("%d\n",c++); //this statement displays 2 then, only c incremented by 1 to 3.
Printf("%d",++c);   //this statement increments 1 to c then, only c is displayed.
Return 0;
}
```

Output

```
2
4
```

Conditional Operators (? :)

Conditional operators are used in decision making in C programming, i.e, executes different statements according to test condition whether it is either true or false.

Syntax of conditional operators;

conditional_expression?expression1:expression2

If the test condition is true (that is, if its value is non-zero), expression1 is returned and if false expression2 is returned.

Let us understand this with the help of a few examples:

```
int x, y ;

scanf ( "%d", &x ) ;

y = ( x> 5 ? 3 : 4 ) ;
```

This statement will store 3 in y if x is greater than 5, otherwise it will store 4 in y.

The equivalent if statement will be,

```
if ( x > 5 )

    y = 3 ;

else

    y = 4 ;
```

Misc Operators:

There are few other operators supported by c language.

Operator	Description	Example
sizeof()	It is a unary operator which is used in finding the size of data type, constant, arrays, structure etc.	sizeof(a), where a is integer, will return 4.

&	Returns the address of a variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* &sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<<>>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right

Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right