❖ **What is Operator ?**

- Whatever symbol we will used for perform some operation that's called a operators.
- It is used to perform certain mathematical or logical manipulation.
- C supports several kinds of operators.
- An operator is a symbol that tell to computer that to perform mathematical or logical operation.
- An operator is used in program to perform the operation on variable or data values.
- C operator can be classified as under.
  1) Arithmetic operator
  2) Relational operator
  3) Logical operator
  4) Assignment operator
  5) Increment and Decrement operator
  6) Conditional operator
  7) Special operator
  8) Bitwise operator

1. Arithmetic operators:

- Integer, floating point and double precision numbers can be added, subtracted, divided or multiplied using arithmetic operators.

Operators	Meaning
+	Addition - Plus
-	Subtraction - Minus
*	Multiplication
/	Division
%	Modular Division

- Integer arithmetic operator

  a=15, b=10

  a + b = 25
- Real arithmetic operator

  a=15.5, b=9.5

  a + b=19.5
- Mix mode arithmetic operator

  a=10, b=9.5

  a + b=19.5

2. Relational operators

- We often compare two quantities and depending on their relation take certain decisions.
- Ex. a < b or 1 < 20
- Containing relational operator is termed as a relational expression. The value of relation expression (0 or 1) 0 for false and 1 for true.

Operators	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

Ex. 5 <= 10 True

5 <- 10 False

3. Logical Operators

- Logical operators are used to combine two or more relation.
- The logical operators are called Boolean operators.

Operators	Meaning
&&	Logical AND
| |	Logical OR
!	Logical NOT

- Ex. a > b && x==10
- It is a combination of two relation expression.
- The expression has a value 1 (Boolean value) If the both expression a > b and x==10 are true. If either is false than the whole expression is false.

4. Assignment operators

- Assignment operators are used to assign result of an expression to a variable.
- C has a set of shorthand assignment operators of the form.
- Variable operator = expression
- Ex. y = y + 5 is y+ = 5

5. Increment and decrement operators

- Increment operator + +, + + adds 1 to the operand
- Decrement operator - -, - - subtract 1 to the operand
- ++m is equivalent to m+1 = m

6. Conditional operators

```
#include<stdio.h>
int main()
{
int num;     // conditional operator example
printf("Enter the Number : ");
scanf("%d",&num);
flag = ((num%2==0)?1:0);
if(flag==0)
  printf("\nEven");
else
  printf("\nOdd");
}
```

Example:- (p == 0) ? (p += 1) : (p += 2)

7. Special operators

- Below are some of special operators that C language offers.

S.no	Operators	Description
1	&	This is used to get the address of the variable. Example : &a will give address of a.
2	*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
3	Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

8. Bitwise operators

- A bitwise operator works on each bit of data. Bitwise operators are used in bit level programming.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

- Bitwise operator is advance topic in programming . Learn more about bitwise operator in C programming.

❖ Arithmetic Expressions

- An arithmeticexpression contains only arithmetic operators and operands. We know that the arithmetic operators in C language include unary operators (+ - ++ --), multiplicative operators (* / %) and additive operators (+ -). The arithmetic operands include integral operands (various int and char types) and floating-type operands (float, double and long double).

- Expressions in C can get rather complicated because of the number of different types and operators that can be mixed together.
- First, a bit of terminology. Expressions in C are built from combinations of operators and operands, so for example in this expression x = a+b*(-c)
- we have the operators =, + * and -. The operands are the variables x, a, b and c. You will also have noticed that

parentheses can be used for grouping sub-expressions such as the -c. Most of C's unusually rich set of operators are either binary operators, which take two operands, or unary operators, which take only one.

❖ **Rules of Operator Precedence**

- C applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those in algebra:
- Operators in expressions contained within pairs of parentheses are evaluated first. Parentheses are said to be at the "highest level of precedence." In cases of nested, or embedded, parentheses, such as
- ((a + b) + c) the operators in the innermost pair of parentheses are applied first.
- Multiplication, division and remainder operations are applied next. If an expression contains several multiplication, division and remainder operations, evaluation proceeds from left to right. Multiplication, division and remainder are said to be on the same level of precedence.
- Addition and subtraction operations are evaluated next. If an expression contains several addition and subtraction operations, evaluation proceeds from left to right. Addition and subtraction also have the same level of precedence, which is lower than the precedence of the multiplication, division and remainder operations.
- The assignment operator (=) is evaluated last.
- The rules of operator precedence specify the order C uses to evaluate expressions.1 When we say evaluation proceeds from left to right, we're referring to the associativity of the operators. We'll see that some operators associate from right to left. Figure 2.7 summarizes these rules of operator precedence for the operators we've seen so far.

❖ **Fig. 1 Precedence of arithmetic operators.**

- Sample Algebraic and C Expressions

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the *innermost* pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*/%	MultiplicationDivision Remainder	Evaluated second. If there are several, they're evaluated left to right.
+-	AdditionSubtraction	Evaluated third. If there are several, they're evaluated left to right.
=	Assignment	Evaluated last.

Example: -

```
#include <stdio.h>
int main (void)
{
    int   a = 25;     int   b = 2;     float c = 25.0;     float d = 2.0;
    printf ("6 + a / 5 * b = %i\n", 6 + a / 5 * b);
    printf ("a / b * b = %i\n", a / b * b);
    printf ("c / d * d = %f\n", c / d * d);
    printf ("-a = %i\n", -a);
    return 0;
}
```

Program Output

```
6 + a / 5 * b = 16
a / b * b = 24
c / d * d = 25.000000
-a = -25
```

❖ **Type casting is** a way to convert a variable from one data type to another data type.

- For example, if you want to store a long value into a simple integer then you can type cast long to int. You can convert values from one type to another explicitly using the cast operator as follows:
- (type_name) expression
- Consider the following example where the cast operator causes the division of one integer variable by another to be performed as a floating-point operation:

```
#include <stdio.h>
main()
{
        int sum = 17, count = 5;   double mean;
        mean = (double) sum / count;
        printf("Value of mean : %f\n", mean );
}
```

- When the above code is compiled and executed, it produces the following result:
- Value of mean : 3.400000
- It should be noted here that the cast operator has precedence over division, so the value of sum is first converted to type double and finally it gets divided by count yielding a double value.
- Type conversions can be implicit which is performed by the compiler automatically, or it can be specified explicitly through the use of the cast operator. It is considered good programming practice to use the cast operator whenever type conversions are necessary.

❖ Math Functions

- The math.h header defines various mathematical functions and one macro.
- All the functions available in this library take double as an argument and return double as the result.
- **Below we list some common math functions**
- double pow (double x, double y) -- Compute x raised to the powery.
- double sin(double x) -- Compute sine of angle in radians.
- double sinh(double x) - Compute the hyperbolic sine of x.
- double sqrt(double x) -- Compute the square root of x.
- double exp(double x -- Compute exponential of x
- double ceil(double x) -- Get smallest integral value that exceeds x.
- double cos(double x) -- Compute cosine of angle in radians.

❖ Managing Input and Output Operators

❖ Reading and Writing Single Char

- The ch = getchar() function and problems associated with input stream
- The isalpha(), isdigit(), isalnum(), islower(), isupper(), isprint(), isspace(), ispunct(), etc. functions and ctype.h file
- The putchar(<char>) function
- putchar('\n') jumps to next line

❖ The getchar() & putchar() functions

- The int getchar(void) function reads the next available character from the screen and returns it as an integer.

- This function reads only single character at a time. You can use this method in the loop in case you want to read more than one characters from the screen.
- The int putchar(int c) function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen.

Check the following example:

```
#include <stdio.h>int main( ){   int c;
printf( "Enter a value :");   c = getchar( );
printf( "\nYou entered: ");   putchar( c );
return 0;}
```

❖ **The gets() & puts() functions**

- The char *gets(char *s) function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF.
- The int puts(const char *s) function writes the string s and a trailing newline to stdout.

```
#include <stdio.h>int main( )
{
   char str[100];
   printf( "Enter a value :");
   gets( str );
   printf( "\nYou entered: ");
   puts( str );
   return 0;
}
```

❖ **Formatted Input**

- Scanf(<control string>,arg1,arg2,agr3…)
- Control string specifies the field format in which the data is to be entered
- arg1, arg2 etc specifies the locations or addresses where the data is to be stored
- Control string contains field specifications which directs the interpretation of input
- Control String may include
- Field or format specs consisting of conversion char (%), a data type specifier, and optional width field
- Blanks, tabs and newlines
- %wd for integers, %f for float, %lf for double,
- * as width for skipping a field
- %ws or %wc for char strings
- Scanf returns no of items read

Code	Meaning	Code	Meaning
%c	Single Char	%i	Decimal, hex or octal int
%d	Decimal Integer	%o	Octal integer
%e	Floating point	%s	String
%f	Floating point	%u	Unsigned int
%g	Floating point	%x	Hexadecimal int
%h	Short int	%[/^..]	String of words

❖ **Points for Scanf**

- All args except CS, must be pointer to vars
- Format specs should match the args in order

- Input data items to be separated by spaces and match the variables receiving in order
- Invalid mismatch will terminate scanf
- Scanf ignores line boundaries
- Unread data items are part of next read
- W should be large enough

❖ **Formatted Output**

- printf("control" string, arg1,arg2…)
- Control string may contain following
- Chars to be printed on the screen as it is
- Format specs for each item to be displayed
- Escape sequences chars such as \n, \t etc
- The args should match the specs in number order and type
- Form of format spec is %w.p type-specs

❖ Format Spec for Integers

Format	**Output**
printf("%d",9876);	9876
printf("%6d",9876);	_ _9876
printf("%2d",9876);	9876
printf("%-6d",9876);	9876_ _
printf("%06d",9876);	009876

❖ **Real Number Output**

- %w.p f or %w.p e is the format spec
- w is minimum number of position used to display the value
- p is number of digits displayed after decimal point

- If E is used Output will contain E
- If g or G, it picks up shorter f or e(E)
- The value, when printed, is rounded to p decimal places and printed right justified in w columns
- Leading blanks and trailing zeroes will appear as necessary
- The default precision is 6 decimal places
- w >= p +7 for e or E
- printf ("%.*f", width, precision, number)

❖ **Format Specs for Real Numbers**

Format	Output
printf("%7.4f",y)	98.7654
printf("%7.2f",y)	_ _ 98.77
printf("%-7.2f",y)	98.77_ _
printf("%f",y)	98.7654
printf("10.2e",y)	_ __ 9.88e+01
printf("%11.4e",-y)	-9.8765e+01
printf("%-10.2e",y)	9.88e+01_ _
printf("%e",y)	9.876540e+01

Printing Characters

- %w.ps for strings where w is the field width of display and p is no of chars to be printed
- It is printed right justified !
- %wc can print right justified in width of w columns
- Default value of w is 1, if precedes with – sign, it prints the data right justified

Printing of Chars and Strings

Specification	Output
%s	New Delhi 110001
%20s	_ _ _ _ New Delhi 110001
%20.10s	_ _ _ _ _ _ _ _ _ _ New Delhi
%.5s	New D
%-20.10s	New Delhi _ _ _ _ _ _ _ _ _ _ _ _ _
%5s	New Delhi 110001

Output Format Flags

Flag	Meaning
-	Output left justified within the field.
+	Signed numeric item will be preceded by +/-
0	Causes leading zeros to appear
#(with o or x)	Octal and hex items to be preceded with 0 or 0x
# (with e f or g)	Causes a decimal point to be present in all floating point numbers. In g prevents truncation of trailing zeros