**Chapter-4: Operators Supported or Used in ‘C’ Language**

‘C’ is richer than other computer languages in the matter of operators. ‘C’ has a number of operators, which can be categorized in the following categories: -

1. **Arithmetic Operators**
2. **Assignment Operators**
3. **Relational OperatorsLogical Operators**
4. **Bitwise Operators**
5. **Special Operators**
6. **Arithmetic Operators**

* **Simple Arithmetic Operators**

+ Plus **🡪 Addition or Summation**

– Minus **🡪 Subtraction or Negative**

\* Asterisk **🡪 Multiplication or Product**

/ Forward Slash **🡪 Division**

%Modulus **🡪 Remainder**

* **Increment/Decrement Operators**

++Plus Plus **🡪 Increment by value 1**

--Minus Minus  **🡪 Decrement by value 1**

1. **Assignment Operators**

* **Simple Assignment Operator**

**=** Equal To **🡪 Assignment**

* **Multiple Assignment Operator**

**=** Equal To **🡪 Multiple Assignment**

* **Shortcut or Compound Operator**

**+=** Plus Equal **🡪 Addition then Assignment**

**-+** Minus Equal **🡪 Minus then Assignment**

**\*=** Asterisk Equal **🡪 Product then Assignment**

**/=** Slash Equal **🡪 Division then Assignment**

**%=** Modulus Equal **🡪 Remainder then Assign**

1. **Relational or Comparison Operators**

**<** Less Than

**>** Greater Than

**<=** Less Than or Equal To

**>=** Greater Than or Equal To

== Equal To

!= Not Equal To

1. **Logical Operators**

**&&** Double Ampersand **🡪 Logical AND**

**||** Double Vertical Bar **🡪 Logical OR**

**!** Exclamation  **🡪 Logical NOT**

1. **Bitwise Operators**

**<<** Double Less Than **🡪 Left Shift**

**>>** Double Greater Than **🡪 Right Shift**

**&** Ampersand **🡪 Bitwise AND**

**|** Vertical Bar **🡪 Bitwise OR**

**~** Tilde **🡪 Bitwise NOT**

**^** Carrot  **🡪 Bitwise XOR**

1. **Special Operators**

**&** Ampersand **🡪 Address**

**\*** Asterisk **🡪 Value@Address**

**,** Comma **🡪 Separator**

**?** Question Mark **🡪 Conditional**

**()** Parenthesis  **🡪 Precedence**

**[]** Square Bracket **🡪 Array**

**.** Dot **🡪 Element**

=>Arrow  **🡪 Element**

**sizeof()** sizeof **🡪 size in bytes**

**(type)** Type Cast **🡪 Type Casting**

**Expression: -**

A combination of at least two **operands** and an **operator** is referred to as an **expression**.

For example: -



**3 + 2**





**3 + 2** or  **a + b 🡪 An Expression**

**3 + 2 \* 5** or **a + b \* c - d 🡪 An Expression**

**a++** or **++a 🡪 Not An Expression**

**b--** or **--b 🡪 Not An Expression**

**Kinds of Operators According to the nature and number of operands: -**

* 1. **Unary Operators**
  2. **Binary Operators**
  3. **Ternary Operators**

**Unary Operators**

These are those operators, which require **only one operand** to work. These include: -

**++ 🡪 a++** or **++a**

**-- 🡪 a--** or **--a**

**- 🡪 -3** or **-a**

**Binary Operators**

These are those operators, which require **two operands** to work. These include: -

**+ 🡪 3+2** or **a+b**

**- 🡪 3-2** or **a-b**

**\* 🡪 3\*2 or a\*b**

**/ 🡪 3/2 or a/2**

**% 🡪 3%2 or a%b**

**< 🡪 3<2 or 3>2**

**<= 🡪 2<=2 or 3>=2**

**== 🡪 2==2 or 3==2**

**!= 🡪 3!=2 or a!=2**

**= 🡪 a=3 or a=2**

**Ternary Operator**

This is that operator, which requires **three operands** to work. This includes: -

**? 🡪 3>2 ? 1 : 0**

**🡪 3 < 2 ? 0 : 1**

**Simple Arithmetic Operators**

Simple Arithmetic Operators are used in ‘C’ language to perform **mathematical calculations** or **expression evaluation.** For examples: **-**

**3+2 🡪 5**

**2+3 🡪 5**

**3-2 🡪 1**

**2-3 🡪 -1**

**3\*2 🡪 6**

**2\*2 🡪 4**

**2\*2\*2 🡪 8**

**3/2 🡪 1**

**3%2 🡪 1**

**5/2 🡪 2**

**5%2 🡪 1**

**10/2 🡪 5**

**10%2 🡪 0**

**5.0/2 🡪 2.5**

**5/2.0 🡪 2.5**

**5.0/2.0 🡪 2.5**

**2/5 🡪 0**

**2/5.0 🡪 0.4**

**2.0/5 🡪 0.4**

**2.0/5.0 🡪 0.4**

**5.0%2 🡪 Error [**Hint**: Fractional division never gives remainder]**

**5%2.0 🡪 Error**

**5.0%2.0 🡪 Error**

**“A”+”A” 🡪 Error**

**“A”+65 🡪 Error**

**‘A’+65 🡪 130 [**Hint**: ASCII of ‘A’ is 65, Hence 65+65 🡪 130]**

**‘A’-65 🡪 0**

**‘A’\*2 🡪 130**

**‘A’/2 🡪 32**

**‘A’%2 🡪 1**

**printf(“Operator”)+scanf(“%d %d”,&var1,&var2) 🡪 10**



**Explanation:-**

**printf(“Operator”) 🡪 8**

**scanf(“%d %d”,&var1,&var2) 🡪 2**

**Hence,**

**Printf(“Operator”)+scanf(“%d %d”,&var1,&var2)**

**🡺 8 + 2**

**🡺 10**

**‘A’ + printf(“Hello”) 🡪 70**

**20 + scanf(“%d %d %d %d %d”,&v1,&v2,&v3,&v4,&v5) 🡪 25**

**‘A’ + printf(“Hello”) - 20 + scanf(“%d%d”,&v1,&v2) 🡪 52**

**10+10 🡪 20 [**Consideration**:- Result is Decimal Number]**

**10+010 🡪 18 [**Hint**:- Octal 010 is 8 in decimal]**

**10+0x10 🡪 26 [**Hint**: - Hexa 0x10 is 16 in decimal]**

**10+0X10 🡪 26 [**Hint**: - Hexa 0X10 is also 16 in decimal]**

**10+0xa 🡪 20 [**Hint**: - Hexa 0xa is 10 in decimal]**

**10+0XA 🡪 20 [**Hint**: - Hexa 0Xa is also 10 in decimal]**

**10+010+0x10 🡪 34 [10+8+16 🡪 34]**

**5+3\*2 🡪 30**

**(5+3)\*2 🡪 16**

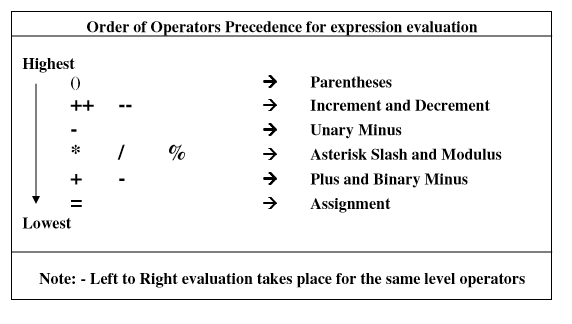
**5+3\*2+3+5 🡪 19**

**(5+3)\*(2+3+5) 🡪 80**

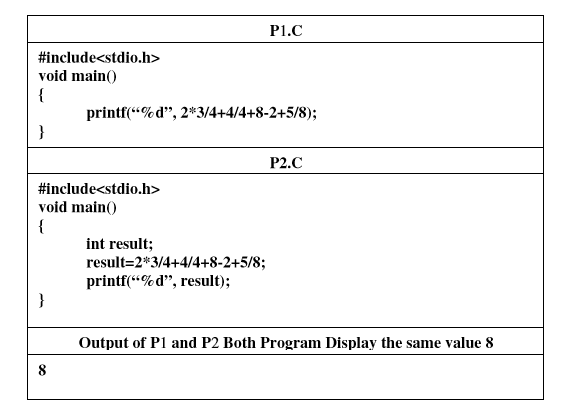
**2\*3/4+4/4+8-2+5/8 🡪 8**

**Note: -**

* We can use **parentheses ()** to alter the order of evaluation.
* The order of precedence can be **high** with the help of **() parentheses**

****

**Note-2:** We can have the result of an expression in a variable with the help of assignment operator **=** . Also if we want to display it, printf() can be used.



**Explanation: - result = 2\*3/4+4/4+8-2+5/8; 🡪 result=8;**

**Expression Evaluation: -**

**6/4+4/4+8-2+5/8**

**🡺 1+4/4+8-2+5/8 [6/4 🡪 1]**

**🡺 1+1+8-2+5/8 [4/4 🡪 1]**

**🡺 1+1+8-2+0 [5/8 🡪 0]**

**🡺 2+8-2+0 [1+1 🡪 2]**

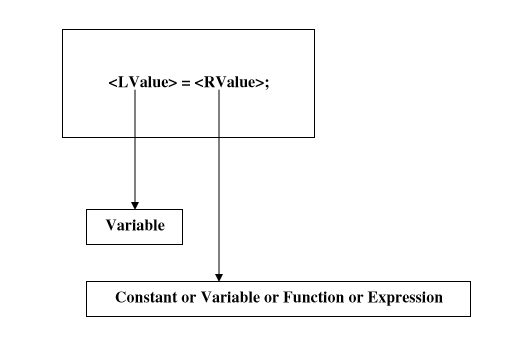
**🡺 10-2+0 [2+8 🡪 10]**

**🡺 8+0 [10-2 🡪 8]**

**🡺 8 [8+0 🡪 8]**

**Simple Assignment Operator: -**

Simple Assignment Operator **=** is used in the following format to assign a value to a specified variable.



**Explanation: - int var1,var2,var3,var4,var5,var6,var7,var8,var9,var10;**

**var**1**=’A’**

**var**2**=65;**

**var**3=**065;**

**var**4=**0x65;**

**var**5=**0X65;**

**var**6=**0XA;**

**var7=Var6;**

**var**8**=printf(“RValue”);**

**var**9**=5+3\*2-1;**

**var**10**=10 + 010 + ’A’ - 0XA;**

**Points To Be Noted Down: -**

* **LValue always denotes a variable on the left of the assignment operator =.**
* **RValue always denotes a constant or variable or function or expression on the right of the assignment operator =.**
* **No expression or function or constant allowed as LValue.**
* **The order of precedence of RValue is higher than LValue. Therefore, RValue is evaluated first and then return value is taken back assign to the LValue.**

**Explanations: - int a,b,c,d;**

**a = 500;**

**b = a;**

**c = a+b;**

**d=a+b+c;**

**a+b+c=d; 🡪 Error**

**500 = a; 🡪 Error**

**a = 10, 20, 30; 🡪 a=10; [**Hint**: - Comma Operator]**

**b = (10,20,30); 🡪 b=30; [Hint: - Comma Operator]**

**c = (printf(“First”),printf(“Second”)); 🡪 c = 6;**

**d = (printf(“First”),printf(“Second”),scanf(“%d%d%d”,&a,&b,&c));**

**🡪 d = 3;**

**Multiple Assignment Operator: -**

When we need to assign a common value to several variables, multiple assignment technique can be used in the following format: -

**Syntax: -**



**Explanation: - int a,b,c,d,e,f=50;**



* **a = 20;**
* **b = 20;**
* **c = 20; a=b=c=d=e=20;**
* **d = 20;**
* **e = 20;**
* **a=b=c=d=e=f;**
* **a=b=c=d=e+f;**
* **a=b=c+d;**
* **a=b+c\*d-e/f;**
* **a=b=c=printf(“RValue”);**
* **a=b+c=d; 🡪 Error**
* **a=b=c+d+d=e+f; 🡪 Error**
* **a=(b=c+d)+(d=e+f);**

**Shortcut or Compound Operators: -**

When we need to update the value of a variable, two operations take place. These includes: -

1. **Arithmetical + - \* / %**
2. **Assignment =**

**Syntax: -**



**Explanation: - int num=20;**

**Add 30 To num: -**

**num=num+30;**

**Multiply by 2: -**

**num=num\*2;**

**The Operations can be combined in the following format: -**

**+= 🡪 Addition then assignment**

**-+ 🡪 Subtraction then assignment**

**\*= 🡪 Multiplication then assignment**

**/= 🡪 Division then assignment**

**%= 🡪 Remainder then assignment**

Syntax:



Examples: -

**num+=30; 🡪 num=num+30;**

**num\*=2; 🡪 num=num\*2;**

**num%=2; 🡪 num=num%2;**

**Note: - Both the forms act the same internally.**

**Increment and Decrement Arithmetic Operators: -**

* **++ Increment Operator 🡪 Increase by value 1**
* **-- Decrement Operator 🡪 Decrease by value 1**

Increment and Decrement Operators are used to **update the value of a variable** by value one in the following format: -

**Syntax: -**



**Explanations: - int num=20;**

**To add 1 to the variable num: -**

**num++; or ++num; 🡪 num=num+1; 🡪 num=20+1; 🡪 num=21;**

**To subtract 1 from the variable num: -**

**num- -; or - -num; 🡪 num=num-1; 🡪 num=21-1;**

**🡪 num=20;**

**Note: - Here both the forms Postfix and Prefix act the same. But in the case of expression it acts differently.**

**Different Forms of Increment and Decrement Operators: -**

* **Postfix Form 🡪 <variable>++ and <variable>--**
* **Prefix Form 🡪 ++<variable> and --<variable>**

**Postfix Form: -** In this form an operator always comes before an operand. E.g. **++a, --b**

**Prefix Form: -** In this form an operator always comes after an operand. E.g. **a++, b--**

**Points To Be Noted Down: -**

* **If no expression form then different forms act same**
* **If expression form then different forms act differently**
* Expression form with increment and decrement operators means more operations have to be combined, in such case the prefix form the increment or decrement operation evaluated first then the other combined operation evaluated. In the case of postfix form other combined operations evaluated first and after back the increment or decrement operation evaluated.

**Explanation-1: - Different Form Same Action**

int num=20;

**Postfix Form: -**

num++; 🡪 num=num+1 🡪 num=20+1 🡪 num=21;

**Prefix Form: -**

++num; 🡪 num=num+1 🡪 num=20+1 🡪 num=21;

**Explanation-2: - Different Form Different Action**

**int n1=20,n2;**

**Postfix Form: -**

**n2=n1++; 🡪 Assignment then Increment 🡪 n2=n1 then n1++**

**🡺**

**🡪 n2=n1;**

**🡪 n2=20;**

**Now,**

**🡪 n1++;**

**🡪 n1=n1+1;**

**🡪 n1=20+1;**

**🡪 n1=21;**

**[Here, n2=20 and n1=21**]

**Prefix Form: -**

**n2=n1++; 🡪 Increment then Assignment 🡪 n1++ then n2=n1;**

**🡺**

**🡪 n1++**

**🡪 n1=n1+1;**

**🡪 n1=20+1;**

**🡪 n1=21;**

**Now,**

**🡪 n2=n1;**

**🡪 n2=21;**

**[Here, n2=21 and also n1=21]**

**Hence, we can say that if it is used in expression form, then the different forms act differently.**

**Explanation-3: -**

**int a=5,b=4,c=3, d=2, e;**

**Prefix Expression: -**

**e = ++a + --b - --c \* ++d;**

**🡺 ++a + --b - --c \* ++d;**

**🡪 ++a 🡪 a=6**

**🡪 --b 🡪 b=3**

**🡪 --c 🡪 c=2**

**🡪 ++d 🡪 d=3**

**Now,**

**++a + --b - --c \* ++d;**

**🡪 6 + --b - --c \* ++d;**

**🡪 6 + 3 - --c \* ++d;**

**🡪 6 + 3 - 2 \* ++d;**

**🡪 6 + 3 -2 \* 3;**

**🡪 6 + 3 - 6;**

**🡪 9 - 6;**

**🡪 3;**

**Then,**

**e=++a + --b - --c \* ++d; 🡪 e=3;**

**Hence,**

**a=6, b=3, c=2, d=3 and e=3**

**Postfix Expression: -**

**e = a++ + b++ - c-- \* d++;**

**🡺**

**a++ + b-- - c-- \* d++;**

**🡪 a + b - c \* d;**

**🡪 5 + 4 - 3 \* 2;**

**🡪 5 + 4 - 6;**

**🡪 9 -6;**

**🡪 3**

**Now,**

**e= a + b - c \* d; 🡪 e=3;**

**Then,**

**🡪 a++ 🡪 a=6**

**🡪 b-- 🡪 b=3**

**🡪 c-- 🡪 c=2**

**🡪 d++ 🡪 d=2**

**Hence,**

**a=6, b=3, c=2, d=3 and e=3**