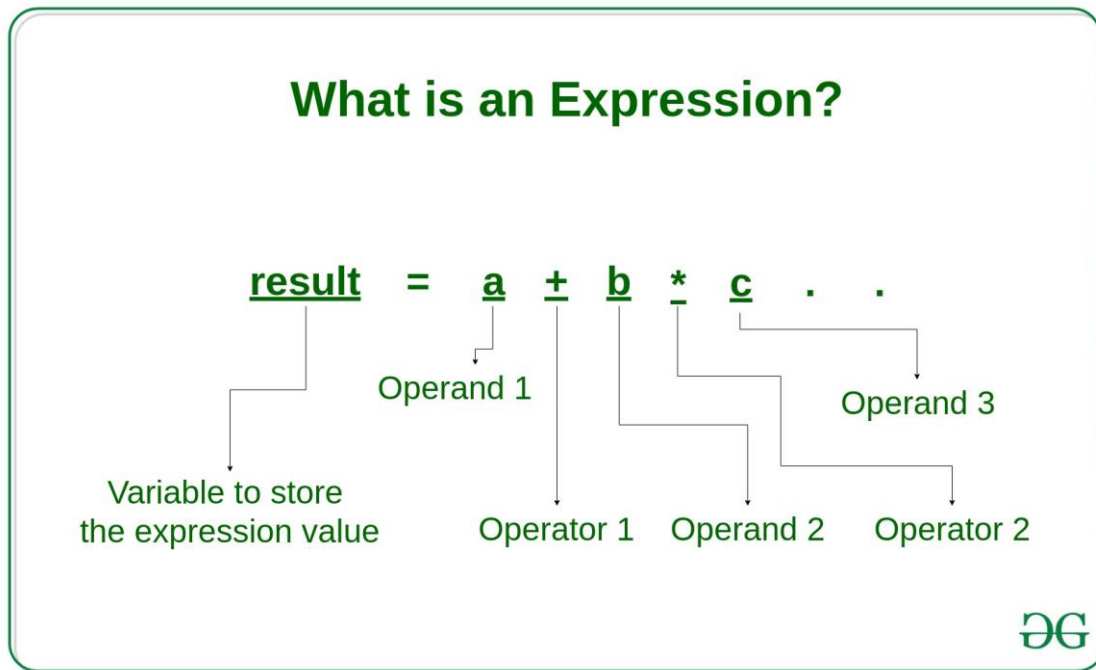


What is an Expression and What are the types of Expressions?

Expression: An expression is a combination of operators, constants and variables. An expression may consist of one or more operands, and zero or more operators to produce a value.



Example:

`a+b`

`c`

`s-1/7*f`

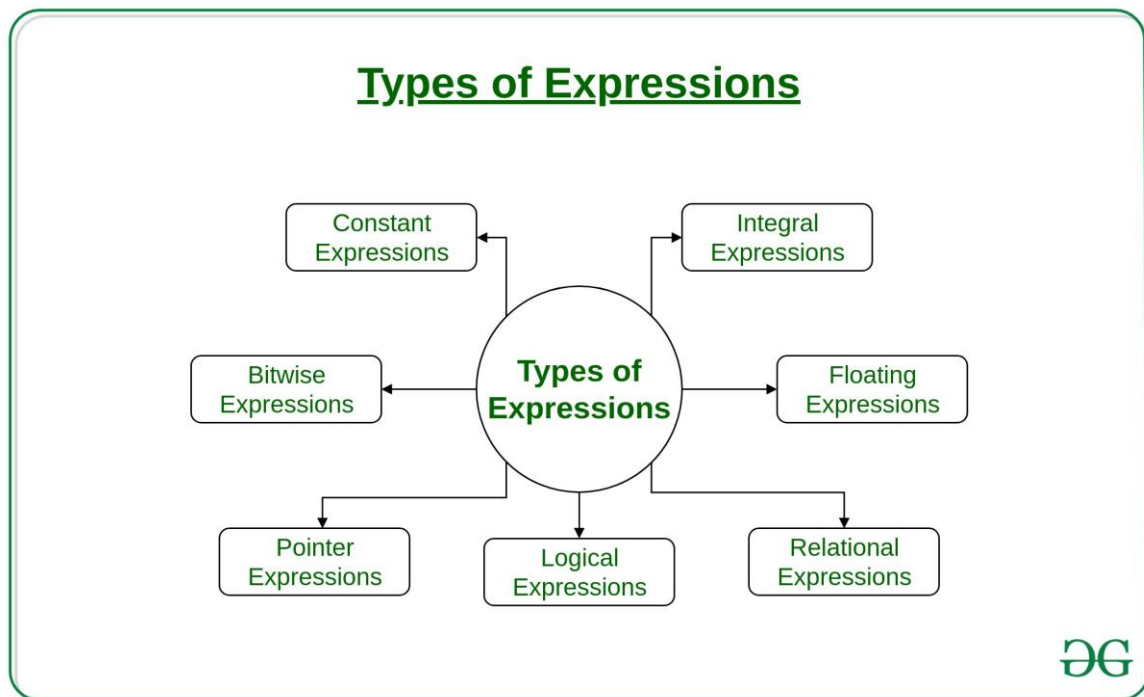
`.`

`.`

Etc

Types of Expressions:

Expressions may be of the following types:



- **Constant expressions:** Constant Expressions consists of only constant values. A constant value is one that doesn't change.
Examples:
5, 10 + 5 / 6.0, 'x'
- **Integral expressions:** Integral Expressions are those which produce integer results after implementing all the automatic and explicit type conversions.
Examples:
x, x * y, x + int(5.0)
where x and y are integer variables.
- **Floating expressions:** Float Expressions are which produce floating point results after implementing all the automatic and explicit type conversions.
Examples:
x + y, 10.75
where x and y are floating point variables.
- **Relational expressions:** Relational Expressions yield results of type bool which takes a value true or false. When arithmetic expressions are used on either side of a relational operator, they will be evaluated first and then the results compared. Relational expressions are also known as Boolean expressions.
Examples:

$x \leq y, x + y > 2$

- **Logical expressions:** Logical Expressions combine two or more relational expressions and produces bool type results.

Examples:

$x > y \ \&\& \ x == 10, x == 10 \ || \ y == 5$

- **Pointer expressions:** Pointer Expressions produce address values.

Examples:

$\&x, ptr, ptr++$

where x is a variable and ptr is a pointer.

- **Bitwise expressions:** Bitwise Expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits.

Examples:

$x \ll 3$

shifts three bit position to left

$y \gg 1$

shifts one bit position to right.

Shift operators are often used for multiplication and division by powers of two.

Note: An expression may also use combinations of the above expressions. Such expressions are known as **compound expressions**.

Example 1:

$10 - 3 \% 8 + 6 / 4$

$\boxed{}$

$10 - 3 + 6 / 4$

$\boxed{}$

$10 - 3 + 1$

$\boxed{}$

$7 + 1$

$\boxed{}$

8

Example 2:

17 - 8 / 4 * 2 + 3 - ++a

17 - 8 / 4 * 2 + 3 - 6

17 - 2 * 2 + 3 - 6

17 - 4 + 3 - 6

13 + 4 - 6

16 - 6

10

Assume that the value of $a = 5$, $b = 6$, $c = 8$, $d = 2$, and $e = 1$. Using these values, let us try to evaluate and find the result of the following expression using the above tables.

Expression:

$a + b \ \&\& \ b / c \% d * e \ || \ 5 \leq 0 \ != \ a + b - d$

Solution:

$5 + 6 \ \&\& \ 6 / 8 \% 2 * 1 \ || \ 5 \leq 0 \ != \ 5 + 6 - 2$ (substitute values)

$5 + 6 \ \&\& \ 0 \% 2 * 1 \ || \ 5 \leq 0 \ != \ 5 + 6 - 2$ (6 / 8 evaluated)

$5 + 6 \ \&\& \ 0 * 1 \ || \ 5 \leq 0 \ != \ 5 + 6 - 2$ (0 % 2 evaluated)

$5 + 6 \ \&\& \ 0 \ \ 5 \leq 0 \ != \ 5 + 6 - 2$	($0 * 1$ evaluated)
$11 \ \&\& \ 0 \ \ 5 \leq 0 \ != \ 5 + 6 - 2$	($5 + 6$ evaluated)
$11 \ \&\& \ 0 \ \ 5 \leq 0 \ != \ 11 - 2$	($5 + 6$ evaluated)
$11 \ \&\& \ 0 \ \ 5 \leq 0 \ != \ 9$	($11 - 2$ evaluated)
$11 \ \&\& \ 0 \ \ 0 \ != 9$	($5 \leq 0$ evaluated)
$11 \ \&\& \ 0 \ \ 1$	($0 \ != 9$ evaluated)
$0 \ \ 1$	($11 \ \&\& \ 0$ evaluated)
1	($0 \ \ 1$ evaluated)

In simple words, when an expression involving multiple operators is given, firstly try to evaluate the operators in an order starting with the highest precedence (or priority) to the lowest. If you come across an expression that contains operators with the same priority or precedence, apply associativity rule (L to R) or (R to L) to figure out which one to evaluate first and continue the process until you yield a final result.

Arithmetic Expressions

$$6*2/(2+1 * 2/3 + 6) + 8 * (8/4)$$

Evaluation of expression	Description of each operation
$6*2/(2+1 * 2/3 + 6) + 8 * (8/4)$	An expression is given.
$6*2/(2+2/3 + 6) + 8 * (8/4)$	2 is multiplied by 1, giving value 2.

$6*2/(2+0+6) + 8 * (8/4)$	2 is divided by 3, giving value 0.
$6*2/ 8+ 8 * (8/4)$	2 is added to 6, giving value 8.
$6*2/8 + 8 * 2$	8 is divided by 4, giving value 2.
$12/8 +8 * 2$	6 is multiplied by 2, giving value 12.
$1 + 8 * 2$	12 is divided by 8, giving value 1.
$1 + 16$	8 is multiplied by 2, giving value 16.
17	1 is added to 16, giving value 17.

Relational Expressions

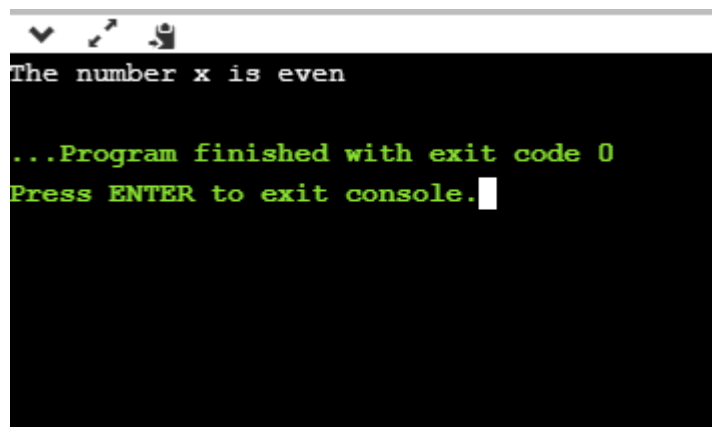
- A relational expression is an expression used to compare two operands.
- It is a condition which is used to decide whether the action should be taken or not.
- In relational expressions, a numeric value cannot be compared with the string value.
- The result of the relational expression can be either zero or non-zero value. Here, the zero value is equivalent to a false and non-zero value is equivalent to true.

Relational Expression	Description
$x\%2 == 0$	This condition is used to check whether the x is an even number or not. The relational expression results in value 1 if x is an even number otherwise results in value 0.
$a!=b$	It is used to check whether a is not equal to b. This relational expression results in 1 if a is not equal to b otherwise 0.
$a+b == x+y$	It is used to check whether the expression "a+b" is equal to the expression "x+y".
$a>=9$	It is used to check whether the value of a is greater than or equal to 9.

Let's see a simple example:

```
1. #include <stdio.h>
2. int main()
3. {
4.
5.     int x=4;
6.     if(x%2==0)
7.     {
8.         printf("The number x is even");
9.     }
10.    else
11.    printf("The number x is not even");
12.    return 0;
13.}
```

Output



```
The number x is even

...Program finished with exit code 0
Press ENTER to exit console.
```

Logical Expressions

- A logical expression is an expression that computes either a zero or non-zero value.
- It is a complex test condition to take a decision.

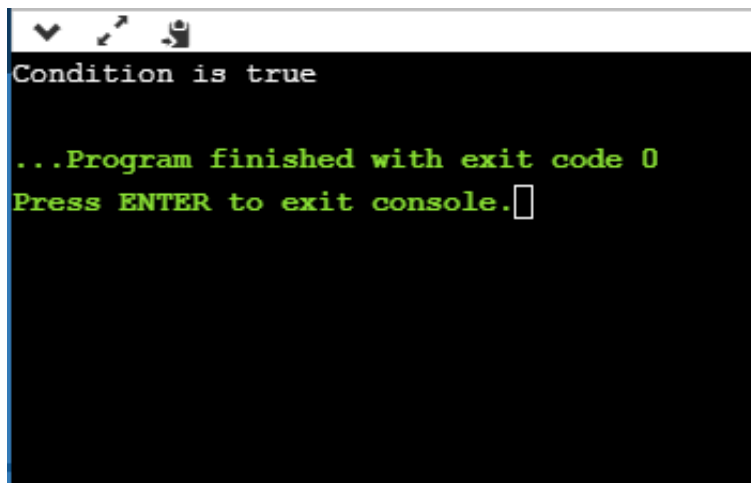
Let's see some example of the logical expressions.

Logical Expressions	Description
<code>(x > 4) && (x < 6)</code>	It is a test condition to check whether the x is greater than 4 and x is less than 6. The result of the condition is true only when both the conditions are true.
<code>x > 10 y < 11</code>	It is a test condition used to check whether x is greater than 10 or y is less than 11. The result of the test condition is true if either of the conditions holds true value.
<code>! (x > 10) && (y = 2)</code>	It is a test condition used to check whether x is not greater than 10 and y is equal to 2. The result of the condition is true if both the conditions are true.

Let's see a simple program of "&&" operator.

```
1. #include <stdio.h>
2. int main()
3. {
4.     int x = 4;
5.     int y = 10;
6.     if ( (x < 10) && (y > 5))
7.     {
8.         printf("Condition is true");
9.     }
10.    else
11.    printf("Condition is false");
12.    return 0;
13.}
```


Output



A screenshot of a console window with a black background and white and green text. The window has a title bar with standard OS icons (minimize, maximize, close) on the left. The text inside the console reads: "Condition is true" in white, followed by "...Program finished with exit code 0" and "Press ENTER to exit console." in green. A white cursor is visible at the end of the last line.

```
Condition is true

...Program finished with exit code 0
Press ENTER to exit console.
```

Mixed Mode Expression And Type Conversions

When the variables and constants of different types are mixed in an expression ,they are all converted to the same type .

The compiler converts all operands up to the type of the largest operand , which is called type promotion .

Generally , the lower type of the operator is promoted to higher type operand and the result will be of the higher type .

The ordering the data type is :

char < int < long < float < double .

Also , an unsigned value out ranks the corresponding signed type .

Conversion Rules :

The following are the rules applicable to the arithmetic operations between two operators having different data types .

- If char and short int values are used as operands , the char operand is automatically elevated to int .
- If float and double values are used as operands , the float operand is automatically elevated to double .
- If int and float values are used as operands , the int operand is automatically elevated to float .
- If float and double values are used as operands ,the float is automatically elevated to double .
- If long and unsigned int are used as operands , both the operands are automatically elevated to unsigned long.

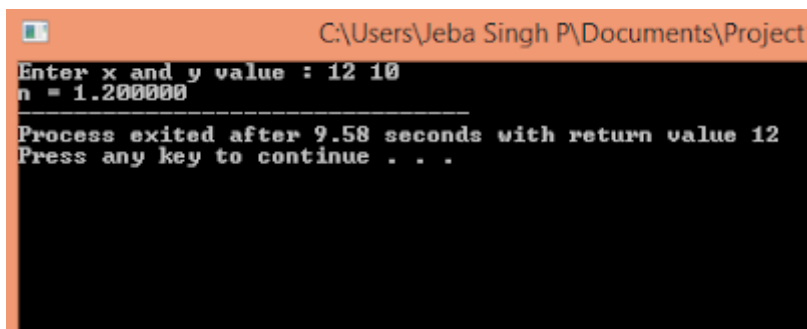
Type Casting :

- To convert the value of an expression to a different type ,the expression must be preceded by the name of the desired data type , enclosed in parenthesis ; (data type) expression .
- This type of conversion is known as type casting . The operators with in C are grouped hierarchically according to their precedence (i.e, order of evaluation).
- Operations with higher precedence are carried out before operations having a lower precedence . The natural order of evaluation can be altered through the use of paranthesis.
- The arithmetic operators * , / and % fall in to precedence group , + and – falls in to another . The first group has a higher precedence than the second one .
- The order in which consecutive operations with in the same precedence group are carried out is known as associately .with in each of the precedence groups described above , the associativity is from left to right.

Example Program:

```
[*] Untitled14.c
1  #include<stdio.h>
2  int main()
3  {
4      int x,y;
5      float n;
6      printf("Enter x and y value : ");
7      scanf("%d%d",&x,&y);
8      n=(float)x/y;
9      printf("n = %f",n);
10 }
11
```

Output :



```
C:\Users\Jeba Singh P\Documents\Project
Enter x and y value : 12 10
n = 1.200000
-----
Process exited after 9.58 seconds with return value 12
Press any key to continue . . .
```