**What is Expressions ? Type of Expression**

An expression is a combination of variables constants and operators written according to the syntax of C language. In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Some examples of C expressions are shown in the table given below.

|  |  |
| --- | --- |
| **Algebraic Expression** | **C Expression** |
| a x b – c | a \* b – c |
| (m + n) (x + y) | (m + n) \* (x + y) |
| (ab / c) | a \* b / c |
| 3x2 +2x + 1 | 3\*x\*x+2\*x+1 |
| (x / y) + c | x / y + c |

**We’ll be covering the following topics in this tutorial:**

* [Evaluation of Expressions](https://ecomputernotes.com/what-is-c/types-and-variables/what-is-expressions-type-of-expression#Evaluation_of_Expressions)
* [Precedence in Arithmetic Operators](https://ecomputernotes.com/what-is-c/types-and-variables/what-is-expressions-type-of-expression#Precedence_in_Arithmetic_Operators)
* [Type conversions in expressions](https://ecomputernotes.com/what-is-c/types-and-variables/what-is-expressions-type-of-expression#Type_conversions_in_expressions)
* [Order](https://ecomputernotes.com/what-is-c/types-and-variables/what-is-expressions-type-of-expression#Order)
* [Category](https://ecomputernotes.com/what-is-c/types-and-variables/what-is-expressions-type-of-expression#Category)
* [Operator](https://ecomputernotes.com/what-is-c/types-and-variables/what-is-expressions-type-of-expression#Operator)
* [Associativity](https://ecomputernotes.com/what-is-c/types-and-variables/what-is-expressions-type-of-expression#Associativity)

**Evaluation of Expressions**

Expressions are evaluated using an assignment statement of the form

**Variable = expression;**

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before evaluation is attempted.  
**Example of evaluation statements are**

x = a \* b – c  
y = b / c \* a  
z = a – b / c + d;  
The following program illustrates the effect of presence of parenthesis in expressions.

main ()

{

**float** a, b, c x, y, z;

a = 9;

b = 12;

c = 3;

x = a – b / 3 + c \* 2 – 1;

y = a – b / (3 + c) \* (2 – 1);

z = a – ( b / (3 + c) \* 2) – 1;

printf ("x = %f",x);

printf ("y = %f",y);

printf ("z = %f",z);

}

**output**

x = 10.00  
y = 7.00  
z = 4.00

**Precedence in Arithmetic Operators**

An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C.

High priority \* / %

Low priority + -

**Rules for evaluation of expression**

• First parenthesized sub expression left to right are evaluated.  
• If parenthesis are nested, the evaluation begins with the innermost sub expression.  
• The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.  
• The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.  
• Arithmetic expressions are evaluated from left to right using the rules of precedence.  
• When Parenthesis are used, the expressions within parenthesis assume highest priority.

**Type conversions in expressions**

**Implicit type conversion**

C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance. This automatic type conversion is know as implicit type conversion  
During evaluation it adheres to very strict rules and type conversion. If the operands are of different types the lower type is automatically converted to the higher type before the operation proceeds. The result is of higher type.  
**The following rules apply during evaluating expressions**

All short and char are automatically converted to int then  
1. If one operand is long double, the other will be converted to long double and result  will be long double.  
2. If one operand is double, the other will be converted to double and result will be double.  
3. If one operand is float, the other will be converted to float and result will be float.  
4. If one of the operand is unsigned long int, the other will be converted into unsigned  long int and result will be unsigned long int.  
5. If one operand is long int and other is unsigned int then  a. If unsigned int can be converted to long int, then unsigned int operand will be converted as such and the result will be long int.  
…..b. Else Both operands will be converted to unsigned long int and the result will be unsigned long int.  
6. If one of the operand is long int, the other will be converted to long int and the result will be long int.  
7. If one operand is unsigned int the other will be converted to unsigned int and the result will be unsigned int.

**Explicit Conversion**

Many times there may arise a situation where we want to force a type conversion in a way that is different from automatic conversion.  
Consider for example the calculation of number of female and male students in a class  
         female\_students  
Ratio =  —————  
          male\_students  
Since if female\_students and male\_students are declared as integers, the decimal part will be rounded off and its ratio will represent a wrong figure. This problem can be solved by converting locally one of the variables to the floating point as shown below.

**Ratio = (float) female\_students / male\_students**

The operator float converts the female\_students to floating point for the purpose of evaluation of the expression. Then using the rule of automatic conversion, the division is performed by floating point mode, thus retaining the fractional part of the result. The process of such a local conversion is known as explicit conversion or casting a value. The general form is (type\_name) expression

**Operator precedence and associativity**

Each operator in C has a precedence associated with it. The precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of these levels. The operators of higher precedence are evaluated first. The operators of same precedence are evaluated from right to left or from left to right depending on the level. This is known as associativity property of an operator.

**The table given below gives the precedence of each operator.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Order** | **Category** | **Operator** | **Operation** | **Associativity** |
| 1 | Highest precedence | ( ) [ ] ? : : . | Function call | L ? R Left to Right |
| 2 | Unary | ! ~ + – ++ – – & \* Size of | Logical negation (NOT) Bitwise 1’s complement Unary plus Unary minus Pre or post increment Pre or post decrement Address Indirection Size of operant in bytes | R ? L Right -> Left |
| 3 | Member Access | .\* ?\* | Dereference Dereference | L ? R |
| 4 | Multiplication | \* / % | Multiply Divide Modulus | L ? R |
| 5 | Additive | + – | Binary Plus Binary Minus | L ? R |
| 6 | Shift | << >> | Shift Left Shift Right | L ? R |
| 7 | Relational | < <= > >= | Less than Less than or equal to Greater than Greater than or equal to | L ? R |
| 8 | Equality | == != | Equal to Not Equal to | L ? R |
| 9 | Bitwise AAND | & | Bitwise AND | L ? R |
| 10 | Bitwise XOR | ^ | Bitwise XOR | L ? R |
| 11 | Bitwise OR | | | Bitwise OR | L ? R |
| 12 | Logical AND | && | Logical AND | L ? R |
| 14 | Conditional | ? : | Ternary Operator | R ? L |
| 15 | Assignment | = \*= %= /= += -= &= ^= |= <<= >>= | Assignment Assign product Assign reminder Assign quotient Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift | R ? L |
| 16 | Comma | , | Evaluate | L ? R |

**Managing Input and Output Operations**

Here you will learn about Single character input output, String input and output, Formatted Input For Scanf, Input specifications for real number, Input specifications for a character, Printing One Line, Conversion Strings and Specifiers, Specifier Meaning.

**Introduction**

One of the essential operations performed in a C language programs is to provide input values to the program and output the data produced by the program to a standard [output device](https://ecomputernotes.com/fundamental/input-output-and-memory/list-various-input-and-output-devices). We can assign values to variable through assignment statements such as x = 5 a = 0 ; and so on. Another method is to use the Input then scanf which can be used to read data from a key board. For outputting results we have used extensively the function printf which sends results out to a terminal. There exists several functions in ‘C’ language that can carry out input output operations. These functions are collectively known as standard Input/Output Library. Each program that uses standard input / out put function must contain the statement.  
# include < stdio.h >  at the beginning.

**Single character input output:**

The basic operation done in input output is to read a characters from the standard input device such as the keyboard and to output or writing it to the output unit usually the screen. The getchar function can be used to read a character from the standard input device. The scanf can also be used to achieve the function. The getchar has the following form.  
Variable name = getchar:

Variable name is a valid ‘C’ variable, that has been declared already and that possess the type char.

**Example program :**

#include<stdio.h> // assigns stdio-h header file to your program

**void** main() // Indicates the starting point of the program.

{

**char** C; // variable declaration

printf("Type one character:"); // message to user

C = getchar(); // get a character from key board and Stores it in variable C.

Printf ("The character you typed is = %c", C); // output

} // Statement which displays value of C on// Standard screen.

The putchar function which in analogus to getchar function can be used for writing characters one at a time to the output terminal. The general form is putchar (variable name);  
Where variable is a valid C type variable that has already been declared Ex: Putchar ( ); Displays the value stored in variable C to the standard screen.

**Program shows the use of getchar function in an interactive environment.**

#include <stdio.h> // Inserts stdio.h header file into the Program

**void** main() // Beginning of main function.

{

**char** in; // character declaration of variable in.

printf ("please enter one character"); // message to user

in = getchar(); // assign the keyboard input value to in.

putchar(in); // out put 'in' value to standard screen.

}

**String input and output:**

The gets function relieves the string from standard input device while put S outputs the string to the standard [output device](https://ecomputernotes.com/fundamental/input-output-and-memory/list-various-input-and-output-devices). A strong is an array or set of characters.  
The function gets accepts the name of the string as a parameter, and fills the string with characters that are input from the keyboard till newline character is encountered. (That is till we press the enter key). All the end function gets appends a null terminator as must be done to any string and returns.  
The puts function displays the contents stored in its parameter on the standard screen.  
The standard form of the gets function is gets (str)//Here str is a string variable.  
The standard form for the puts character is puts (str) //Where str is a string variable.  
Eample program (Involving both gets and puts)

#include <stdio.h>

Void main()

{

**char** s[80];

printf ("Type a string less than 80 characters:");

gets (s);

printf ("The string types is:");

puts(s);

}

**Formatted Input For Scanf:**

The formatted input refers to input data that has been arranged in a particular format. Input values are generally taken by using the scanf function. The scanf function has the general form.

Scanf("control string", arg1, arg2, arg3....argn);

The format field is specified by the control string and the arguments arg1, arg2,....argn specifies the address of location where address is to be stored.

The control string specifies the field format which includes format specifications and optional number specifying field width and the conversion character % and also blanks, tabs and newlines.  
The Blanks tabs and newlines are ignored by compiler. The conversion character % is followed by the type of data that is to be assigned to variable of the assignment. The field width specifier is optional. The general format for reading a integer number is  %d.  
Here percent sign (%) denotes that a specifier for conversion follows and x is an integer number which specifies the width of the field of the number that is being read. The [data type](https://ecomputernotes.com/java/data-type-variable-and-array/explain-data-types-in-java) character d indicates that the number should be read in integer mode.  
**Example :**

scanf ("%3d %4d", &sum1, &sum2);

If the values input are 175 and 1342 here value 175 is assigned to sum1 and 1342 to sum 2. Suppose the input data was follows 1342 and 175.  
The number 134 will be assigned to sum1 and sum2 has the value 2 because of %3d the number 1342 will be cut to 134 and the remaining part is assigned to second variable sum2. If floating point numbers are assigned then the decimal or fractional part is skipped by the [computer](https://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer).  
To read the long integer [data type](https://ecomputernotes.com/java/data-type-variable-and-array/explain-data-types-in-java) we can use conversion specifier % ld & % hd for short integer.

**Input specifications for real number:**

Field specifications are not to be use while representing a real number therefore real numbers are specified in a straight forward manner using %f specifier. The general format of specifying a real number input is: Scanf (“%f “, &variable);

**Example:**

Scanf ("%f %f % f", &a, &b, &c);

With the input data321.76, 4.321, 678

The values321.76 is assigned to a , 4.321 to b & 678 to C.

If the number input is a double data type then the format specifier should be %lf instead of %f**.**

**Input specifications for a character.**

Single character or strings can be input by using the character specifiers.The general format is

%c or %s

Where C and S represents character and string respectively and x represents the field width. The address operator need not be specified while we input strings

Example : Scanf ("%c %c", &ch, name);

Here suppose the input given is a, Robert then a is assigned to ch and name will be assigned to Robert.

Printing One Line: printf();

The most simple output statement can be produced in C’ Language by using printf statement. It allows you to display [information](https://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) required to the user and also prints the variables we can also format the output and provide text labels. The simple statement such as Printf (“Enter 2 numbers”);  
Prompts the message enclosed in the quotation to be displayed.  A simple program to illustrate the use of printf statement:-

#include<stdio.h>

main()

{

printf ("Hello!");

printf ("Welcome to the world of Engineering!");

}

Output:

Hello! Welcome to the world of Engineering.

Both the messages appear in the output as if a single statement. If you wish to print the second message to the beginning of next line, a new line character must be placed inside the quotation marks.

**For Example :**

printf ("Hello!\n"); OR

printf ("\n Welcome to the world of Engineering");

**Conversion Strings and Specifiers:**

The printf () function is quite flexible. It allows a variable number of arguments, labels and sophisticated formatting of output. The general form of the printf ( ) function is .

Syntax: Printf ("conversion string", variable list);

The conversion string includes all the text labels, escape character and conversion specifiers required for the desired output. The variable includes all the variable to be printed in order they are to be printed. There must be a conversion specifies after each variable.

**Specifier Meaning**

%c – Print a character  
%d – Print a Integer  
%i – Print a Integer  
%e – Print float value in exponential form.  
%f – Print float value  
%g – Print using %e or %f whichever is smaller  
%o – Print actual value  
%s – Print a string  
%x – Print a hexadecimal integer (Unsigned) using lower case a – F  
%X – Print a hexadecimal integer (Unsigned) using upper case A – F  
%a – Print a unsigned integer.  
%p – Print a pointer value  
%hx – hex short  
%lo – octal long  
%ld – long

**Decision Making – Branching**

In this tutorial you will learn about C Programming – Decision Making, Branching, if Statement, The If else construct, Compound Relational tests, Nested if Statement, The ELSE If Ladder, The Switch Statement and The GOTO statement.

**Branching Introduction:**

The C language programs presented until now follows a sequential form of execution of statements. Many times it is required to alter the flow of the sequence of instructions. C language provides statements that can alter the flow of a sequence of instructions. These statements are called control statements. These statements help to jump from one part of the program to another. The control transfer may be conditional or unconditional.

**if Statement:**

The simplest form of the control statement is the If statement. It is very frequently used in decision making and allowing the flow of program execution. The If structure has the following syntax.

if(condition)

{

statement;

}

The statement is any valid C language statement and the condition is any valid C language expression, frequently logical operators are used in the condition statement. The condition part should not end with a semicolon, since the condition and statement should be put together as a single statement. The command says if the condition is true then perform the following statement or If the condition is fake the [computer](https://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) skips the statement and moves on to the next instruction in the program.

Example program/\* Calculate the absolute value of an integer \*/

#include<stdio.h> //Include the stdio.h file

void main() // start of the program

{

int numbers; // Declare the variables

printf ("Type a number:"); // message to the user

scanf ("%d", &number); // read the number from standard input.

if (number < 0) // check whether the number is a negative number

number = - number; // If it is negative then convert it into positive.

Printf ("The absolute value is %d \n", number); // print the value

}

The above program checks the value of the input number to see if it is less than zero. If it is then the following program statement which negates the value of the number is executed. If the value of the number is not less than zero, we do not want to negate it then this statement is automatically skipped. The absolute number is then displayed by the program, and program execution ends.

**The If else construct:**

if (condition)

Program statement 1;

else

Program statement 2;

The syntax of the If else construct is as follows:-  
The if else is actually just on extension of the general format of if statement. If the result of the condition is true, then program statement 1 is executed, otherwise program statement 2 will be executed. If any case either program statement 1 is executed or program statement 2 is executed but not both when writing programs this else statement is so frequently required that almost all programming languages provide a special construct to handle this situation.

// Program find whether a number is negative or positive

#include <stdio.h> //include the stdio.h header file in your program

void main() // Start of the main

{

int num; // declare variable num as integer

printf("Enter the number"); //message to the user

scanf("%d", &num); // read the input number from keyboard

if(num<0) // check whether number is less than zero.

Printf ("The number is negative"); // If it is less than zero then it is negative.

else // else statement.

Printf ("The number is positive"); //If it is more than zero then the given number is positive.

}

In the above program the If statement checks whether the given number is less than 0. If it is less than zero then it is negative therefore the condition becomes true then the statement The number is negative is executed. If the number is not less than zero the If else construct skips the first statement and prints the second statement declaring that the number is positive.

**Compound Relational tests:**

C language provides the mechanisms necessary to perform compound relational tests. A compound relational test is simple one or more simple relational tests joined together by either the logical AND or the logical OR operators. These operators are represented by the character pairs && // respectively. The compound operators can be used to form complex expressions in C.

**Syntax:**

if (condition1 && condition2 && condition3)

if (condition1 || condition2 || condition3)

The syntax in the statement ‘a’ represents a complex if statement which combines different conditions using the and operator in this case if all the conditions are true only then the whole statement is considered to be true. Even if one condition is false the whole if statement is considered to be false.  
The statement ‘b’ uses the logical operator or (//) to group different expression to be checked. In this case if any one of the expression if found to be true the whole expression considered to be true, we can also uses the mixed expressions using logical operators and and or together.

**Nested if Statement**

The if statement may itself contain another if statement is known as nested if statement.

**Syntax:**

if (condition1)

if (condition2)

statement-1;

else

statement-2;

else

statement-3;

The if statement may be nested as deeply as you need to nest it. One block of code will only be executed if two conditions are true. Condition 1 is tested first and then condition 2 is tested. The second if condition is nested in the first. The second if condition is tested only when the first condition is true else the program flow will skip to the corresponding else statement.

#include<stdio.h> //includes the stdio.h file to your program

main ( ) //start of main function

{

int a,b,c,big; //declaration of variables

printf ("Enter three numbers"); //message to the user

scanf ("%d %d %d", &a, &b, &c); //Read variables a,b,c,

if (a>b) // check whether a is greater than b if true then

if(a>c) // check whether a is greater than c

big = a; // assign a to big

else big = c; // assign c to big

else if (b>c) // if the condition (a>b) fails check whether b is greater than c

big = b; // assign b to big

else big = c; // assign C to big

printf ("Largest of %d,%d&%d = %d", a,b,c,big);

}//print the given numbers along with the largest number.

In the above program the statement if (a>c) is nested within the if (a>b). If the first If condition, If (a>b) is true only then the second if statement if (a>b) is executed. If the first if condition is executed to be false then the program control shifts to the statement after corresponding else statement.

/\* Example program using compound if else construct \*/

/\* This program determines if a year is a leap year \*/

#include<stdio.h> //Includes stdio.h file to your program

void main() // start of the program

{

int year, rem\_4, rem\_10, rem\_400; // variable declaration

printf("Enter the year to be tested"); // message for user

scanf ("t.d", & year); // Read the year from standard input.

rem\_4 = year % 4; //find the remainder of year – by 4

rem\_100 = year % 100; //find the remainder of year – by 100

rem\_400 = year % 400; // find the remainder of year – by 400

if ((rem\_4 == 0 && rem\_100!= 0) //rem\_400 = = 0)

//apply if condition 5 check whether remainder is zero

printf ("It is a leap year, \n") ; // print true condition

else

printf ("No. It is not a leap year. \n"); //print the false condition

}

The above program checks whether the given year is a leap year or not. The year given is divided by 4,100 and 400 respectively and its remainder is collected in the variables rem\_4, rem\_100 and rem\_400. A if condition statements checks whether the remainders are zero. If remainder is zero then the year is a leap year. Here either the year – y 400 is to be zero or both the year – 4 and year – by 100 has to be zero, then the year is a leap year.

**The ELSE If Ladder**

When a series of many conditions have to be checked we may use the ladder else if statement which takes the following general form.

If (condition1)

Statement1;

**else** **if** (condition2)

Statement2;

**else** **if** (condition3)

Statement3;

**else** **if** (condition-n)

Statement-n;

**else**

Default statement;

Statement-x;

This construct is known as if else construct or ladder. The conditions are evaluated from the top of the ladder to downwards. As soon on the true condition is found, the statement associated with it is executed and the control is transferred to the statement – x (skipping the rest of the ladder. When all the condition becomes false, the final else containing the default statement will be executed.

/\* Example program using If else ladder to grade the student according to the following rules.

Marks Grade70 to 100, 60 to 69, 50 to 59, 40 to 49, 0 to 39

#include <stdio.h> //include the standard stdio.h header file

void main() //start the function main

{

int marks; //variable declaration

printf ("Enter marks\n"); //message to the user

scanf ("%d", & marks); //read and store the input marks.

If (marks < =100 && marks > = 70 )//check whether marks is lesss than 100 or greater

printf ("\n Distinction"); than 70 & print Distinction if condition is true

else if (marks > = 60) //else if the previous condition fails check

printf("\n First class"); whether marks is > 60 if true print statement

else if (marks > = 50) //else if marks is greater than 50 print

printf ("\n second class"); //Second class

else if (marks>=35) //else if marks is greater than 35 print

printf ("\n pass class"); //pass class

elseprintf ("Fail"); //If all condition fail apply default condition and print Fail

}

The above program checks a series of conditions. The program begins from the first if statement and then checks the series of conditions it stops the execution of remaining if statements whenever a condition becomes true.  
In the first If condition statement it checks whether the input value is lesser than 100 and greater than 70. If both conditions are true it prints distinction. Instead if the condition fails then the program control is transferred to the next if statement through the else statement and now it checks whether the next condition given is whether the marks value is greater than 60 If the condition is true it prints first class and comes out of the If else chain to the end of the program on the other hand if this condition also fails the control is transferred to next if statements program execution continues till the end of the loop and executes the default else statement fails and stops the program.

**The Switch Statement:**

Unlike the If statement which allows a selection of two alternatives the switch statement allows a program to select one statement for execution out of a set of alternatives. During the execution of the switch statement only one of the possible statements will be executed the remaining statements will be skipped. The usage of multiple If else statement increases the complexity of the program since when the number of If else statements increase it affects the readability of the program and makes it difficult to follow the program. The switch statement removes these disadvantages by using a simple and straight forward approach. The general format of the Switch Statement is :

Switch (expression)

{

Case **case**-label-1;

Case **case**-label-2;

Case **case**-label-n;

.......

Case **default**

}

When the switch statement is executed the control expression is evaluated first and the value is compared with the case label values in the given order. If the label matches with the value of the expression then the control is transferred directly to the group of statements which follow the label. If none of the statements matches then the statement against the default is executed. The default statement is optional in switch statement in case if any default statement is not given and if none of the condition matches then no action takes place in this case the control transfers to the next statement of the if else statement.

//Example switch program

//A program to stimulate the four arithmetic operations using switch.

#include<stdio.h> //include stdio.h header file to your program

main() //start of the main function

{

int num1, num2, result; //Declaration of variables

char operator;

printf ("Enter two numbers"); //Message to the user to enter 2 numbers

scanf ("%dy.d", &num1, &num2); //read and store the numbers.

Printf ("Enter an operator"); //message to the user to enter an operator

Scanf ("%c", &operator); //read and store the operator

Switch(operator) //start of the switch construct

{

case '+ : result = num1 + num2; //If the operator entered is ‘t’

break; add the 2 numbers & store it in result

case '-': result = num1 – num2; //If the operator entered is ‘-‘

break; subtract the numbers & store

case '\*': result = num1 \* num2; it in the result

break; //If the operator enter is (\*)

case '/'; if (num2! = 0) multiply the 2 numbers & show it in

result

result = num1 / num2; // if the operator is ‘/’ then

else check for division by zero if

{ the second number is zero

printf ("warning : division by zero \ n");

result = 0:

}

break;

default: printf ("\n unknown operator"); //If the operator entered result = 0 is none of the 4 operators print message

break; set the result value to zero.

printf ("t.d", result); print the result.

}

In the above program the break statement is need after the case statement to break out of the loop and prevent the program from executing other cases