Introduction

* C Keywords & Identifier
* C Variables & Constants
* C Data Types
* C Input/Output
* C Operators
* Basic Examples

Flow Control

* if...else Statement
* C for Loop
* while & do...while Loop
* break and continue
* switch Statement
* Decision Examples

Functions

* Functions Introduction
* User-defined Function
* Function Types
* Recursion in C
* Variable Scope
* Function Examples

Arrays

* C Arrays Introduction
* Multi-dimensional Arrays
* C Arrays & Functions
* Strings in C
* String Functions
* Array Examples

C Pointers

* C Pointers Introduction
* Pointers & Arrays
* Pointers & Functions
* Memory Management
* Pointer Examples

Structure & File

* Structure Introduction
* Structures & Pointers
* C Structure & Functions
* Unions in C
* Structure Examples
* Files Handling

Additional Resources

* C Examples
* Library Function in C

# C Programming Keywords and Identifiers

Reserved words in C programming that are part of the syntax. Also, you will learn about identifiers and proper way to name a variable.

## Character set

Character set is a set of alphabets, letters and some special characters that are valid in C language.

### Alphabets

<="">

Uppercase: A B C ................................... X Y Z

Lowercase: a b c ...................................... x y z

### Digits

0 1 2 3 4 5 6 7 8 9

### Special Characters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Special Characters in C Programming | | | | |
| , | < | > | . | \_ |
| ( | ) | ; | $ | : |
| % | [ | ] | # | ? |
| ' | & | { | } | " |
| ^ | ! | \* | / | | |
| - | \ | ~ | + |  |

**White space Characters**

blank space, new line, horizontal tab, carriage return and form feed

## C Keywords

Keywords are predefined, reserved words used in programming that have a special meaning. Keywords are part of the syntax and they cannot be used as an identifier. For example:

int money;

Here, int is a keyword that indicates 'money' is a variable of type integer.

<="">

As C is a case sensitive language, all keywords must be written in lowercase. Here is a list of all keywords allowed in ANSI C.

|  |  |  |  |
| --- | --- | --- | --- |
| Keywords in C Language | | | |
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | return | union |
| continue | for | signed | void |
| do | if | static | while |
| default | goto | sizeof | volatile |
| const | float | short | unsigned |

Along with these keywords, C supports other numerous keywords depending upon the compiler.

All these keywords, their syntax and application will be discussed in their respective topics. However, if you want a brief overview on these keywords without going further, visit list of all keywords in C programming.

## C Identifiers

Identifier refers to name given to entities such as variables, functions, structures etc.

Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program. For example:

int money;

double accountBalance;

Here, money and accountBalance are identifiers.

Also remember, identifier names must be different from keywords. You cannot use int as an identifier because int is a keyword.

### Rules for writing an identifier

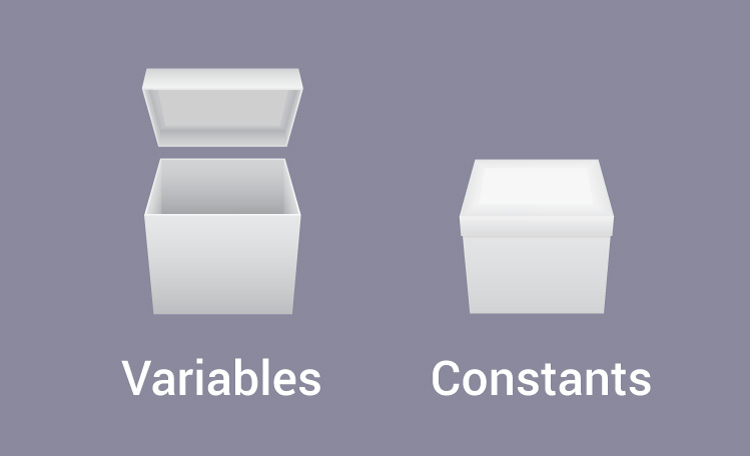
1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore.
3. There is no rule on length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler.

### Good Programming Practice

You can choose any name for an identifier (excluding keywords). However, if you give meaningful name to an identifier, it will be easy to understand and work on for you and your fellow programmers.

# C Programming Constants and Variables

In this tutorial, you will learn about variables, rules for naming a variable, constants and different type of constants in C programming.



## Variables

In programming, a variable is a container (storage area) to hold data.

<="">

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

int playerScore = 95;

Here, playerScore is a variable of integer type. The variable is assigned value: 95.

The value of a variable can be changed, hence the name 'variable'.

In C programming, you have to declare a variable before you can use it.

### Rules for naming a variable in C

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore. However, it is discouraged to start variable name with an underscore. It is because variable name that starts with an underscore can conflict with system name and may cause error.
3. There is no rule on how long a variable can be. However, only the first 31 characters of a variable are checked by the compiler. So, the first 31 letters of two variables in a program should be different.

C is a strongly typed language. What this means it that, the type of a variable cannot be changed.

Visit this page to learn more about different types of data a variable can store.

## Constants/Literals

A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5, "C programming is easy", etc.

As mentioned, an identifier also can be defined as a constant.

const double PI = 3.14

Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.

<="">

Below are the different types of constants you can use in C.

### 1. Integer constants

An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

* decimal constant(base 10)
* octal constant(base 8)
* hexadecimal constant(base 16)

For example:

Decimal constants: 0, -9, 22 etc

Octal constants: 021, 077, 033 etc

Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

### 2. Floating-point constants

A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

-2.0

0.0000234

-0.22E-5

**Note:**E-5 = 10-5

### 3. Character constants

A character constant is a constant which uses single quotation around characters. For example: 'a', 'l', 'm', 'F'

### 4. Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequence is used.

For example: \n is used for newline. The backslash ( \ ) causes "escape" from the normal way the characters are interpreted by the compiler.

| Escape Sequences | |
| --- | --- |
| Escape Sequences | Character |
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \\ | Backslash |
| \' | Single quotation mark |
| \" | Double quotation mark |
| \? | Question mark |
| \0 | Null character |

### 5. String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

"good" //string constant

"" //null string constant

" " //string constant of six white space

"x" //string constant having single character.

"Earth is round\n" //prints string with newline

### 6. Enumeration constants

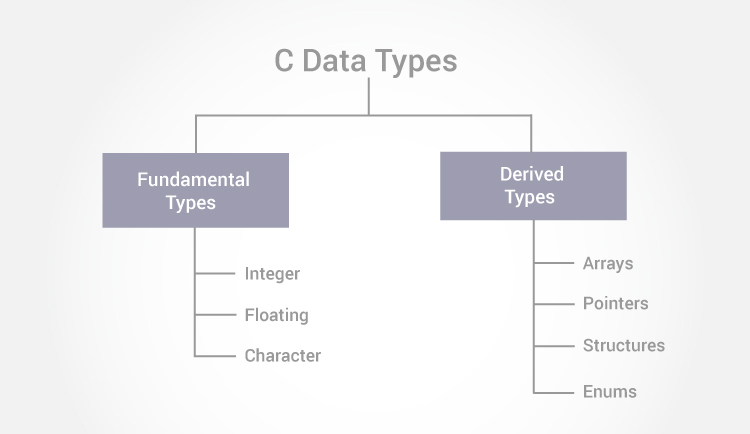
Keyword enum is used to define enumeration types. For example:

enum color {yellow, green, black, white};

Here, color is a variable and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively. For more information, visit page: C Enumeration.

# C Programming Data Types

In this tutorial, you will learn about data types and how to declare a variable in C programming.



In C programming, variables or memory locations should be declared before it can be used. Similarly, a function also needs to be declared before use.

<="">

Data types simply refers to the type and size of data associated with variables and functions.

## Data types in C

1. Fundamental Data Types
   * Integer types
   * Floating type
   * Character type
2. Derived Data Types
   * Arrays
   * Pointers
   * Structures
   * Enumeration

This tutorial will focus on fundamental data types. To learn about derived data types, visit the corresponding tutorial.

**Note: %d – for Integer**

**%l – for Long**

**%f – float**

**%lf – for double (%.2lf – indicates 2 decimal position)**

**%c – for character**

## int - Integer data types

Integers are whole numbers that can have both positive and negative values but no decimal values. Example: 0, -5, 10

In C programming, keyword int is used for declaring integer variable. For example:

int id;

Here, id is a variable of type integer.

You can declare multiple variable at once in C programming. For example:

int id, age;

The size of int is either 2 bytes(In older PC's) or 4 bytes. If you consider an integer having size of 4 byte( equal to 32 bits), it can take 232 distinct states as: -231,-231+1, ...,-2, -1, 0, 1, 2, ..., 231-2, 231-1

Similarly, int of 2 bytes, it can take 216 distinct states from -215 to 215-1. If you try to store larger number than 231-1, i.e,+2147483647 and smaller number than -231, i.e, -2147483648, program will not run correctly.

## float - Floating types

Floating type variables can hold real numbers such as: 2.34, -9.382, 5.0 etc. You can declare a floating point variable in C by using either float or double keyword. For example:

float accountBalance;

double bookPrice;

Here, both accountBalance and bookPrice are floating type variables.

In C, floating values can be represented in exponential form as well. For example:

float normalizationFactor = 22.442e2;

### Difference between float and double

The size of float (single precision float data type) is 4 bytes. And the size of double (double precision float data type) is 8 bytes. Floating point variables has a precision of 6 digits whereas the precision of double is 14 digits.

## char - Character types

Keyword char is used for declaring character type variables. For example:

char test = 'h'

<="">

Here, test is a character variable. The value of test is 'h'.

The size of character variable is 1 byte.

## C Qualifiers

Qualifiers alters the meaning of base data types to yield a new data type.

### Size qualifiers

Size qualifiers alters the size of a basic type. There are two size qualifiers, long and short. For example:

long double i;

The size of float is 8 bytes. However, when long keyword is used, that variable becomes 10 bytes.

Learn more about long keyword in C programming.

If you know that the value of a variable will not be large, short can be used.

### Sign qualifiers

Integers and floating point variables can hold both negative and positive values. However, if a variable needs to hold positive value only, unsigned data types are used. For example:

// unsigned variables cannot hold negative value

unsigned int positiveInteger;

There is another qualifier signed which can hold both negative and positive only. However, it is not necessary to define variable signed since a variable is signed by default.

An integer variable of 4 bytes can hold data from -231 to 231-1. However, if the variable is defined as unsigned, it can hold data from 0 to 232-1.

It is important to note that, sign qualifiers can be applied to int and char types only.

### Constant qualifiers

An identifier can be declared as a constant. To do so const keyword is used.

const int cost = 20;

The value of cost cannot be changed in the program.

### Volatile qualifiers

A variable should be declared volatile whenever its value can be changed by some external sources outside the program. Keyword volatile is used for creating volatile variables.

# C Programming Input Output (I/O): printf() and scanf()

This tutorial focuses on two in-built functions printf() and scanf() to perform I/O task in C programming. Also, you will learn to write a valid program in C.

C programming has several in-built library functions to perform input and output tasks.

<="">

Two commonly used functions for I/O (Input/Output) are printf() and scanf().

The scanf() function reads formatted input from standard input (keyboard) whereas the printf() function sends formatted output to the standard output (screen).

## Example #1: C Output

#include <stdio.h> //This is needed to run printf() function.

int main()

{

printf("C Programming"); //displays the content inside quotation

return 0;

}

**Output**

C Programming

**How this program works?**

* All valid C program must contain the main() function. The code execution begins from the start of main() function.
* The printf() is a library function to send formatted output to the screen. The printf() function is declared in "stdio.h" header file.
* Here, stdio.h is a header file (standard input output header file) and #include is a preprocessor directive to paste the code from the header file when necessary. When the compiler encounters printf() function and doesn't find stdio.h header file, compiler shows error.
* The return 0; statement is the "Exit status" of the program. In simple terms, program ends.

## Example #2: C Integer Output

#include <stdio.h>

int main()

{

int testInteger = 5;

printf("Number = %d", testInteger);

return 0;

}

**Output**

Number = 5

Inside the quotation of printf() function, there is a format string "%d" (for integer). If the format string matches the argument (testInteger in this case), it is displayed on the screen.

## Example #3: C Integer Input/Output

#include <stdio.h>

int main()

{

int testInteger;

printf("Enter an integer: ");

scanf("%d",&testInteger);

printf("Number = %d",testInteger);

return 0;

}

**Output**

Enter an integer: 4

Number = 4

The scanf() function reads formatted input from the keyboard. When user enters an integer, it is stored in variable testInteger.

Note the '&' sign before testInteger; &testInteger gets the address of testInteger and the value is stored in that address.

## Example #3: C Floats Input/Output

#include <stdio.h>

int main()

{

float f;

printf("Enter a number: ");

// %f format string is used in case of floats

scanf("%f",&f);

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

return 0;

}

<="">

**Output**

Enter a number: 23.45

Value = 23.450000

The format string "%f" is used to read and display formatted in case of floats.

## Example #4: C Character I/O

#include <stdio.h>

int main()

{

char chr;

printf("Enter a character: ");

scanf("%c",&chr);

printf("You entered %c.",chr);

return 0;

}

**Output**

Enter a character: g

You entered g.

Format string %c is used in case of character types.

### Little bit on ASCII code

When a character is entered in the above program, the character itself is not stored. Instead, a numeric value(ASCII value) is stored.

And when we displayed that value using "%c" text format, the entered character is displayed.

## Example #5: C ASCII Code

#include <stdio.h>

int main()

{

char chr;

printf("Enter a character: ");

scanf("%c",&chr);

// When %c text format is used, character is displayed in case of character types

printf("You entered %c.\n",chr);

// When %d text format is used, integer is displayed in case of character types

printf("ASCII value of %c is %d.", chr, chr);

return 0;

}

**Output**

Enter a character: g

You entered g.

ASCII value of g is 103.

The ASCII value of character 'g' is 103. When, 'g' is entered, 103 is stored in variable var1 instead of g.

You can display a character if you know ASCII code of that character. This is shown by following example.

## Example #6: C ASCII Code

#include <stdio.h>

int main()

{

int chr = 69;

printf("Character having ASCII value 69 is %c.",chr);

return 0;

}

**Output**

Character having ASCII value 69 is E.

## More on Input/Output of floats and Integers

Integer and floats can be displayed in different formats in C programming.

### Example #7: I/O of Floats and Integers

#include <stdio.h>

int main()

{

int integer = 9876;

float decimal = 987.6543;

// Prints the number right justified within 6 columns

printf("4 digit integer right justified to 6 column: %6d\n", integer);

// Tries to print number right justified to 3 digits but the number is not right adjusted because there are only 4 numbers

printf("4 digit integer right justified to 3 column: %3d\n", integer);

// Rounds to two digit places

printf("Floating point number rounded to 2 digits: %.2f\n",decimal);

// Rounds to 0 digit places

printf("Floating point number rounded to 0 digits: %.f\n",987.6543);

// Prints the number in exponential notation(scientific notation)

printf("Floating point number in exponential form: %e\n",987.6543);

return 0;

}

**Output**

4 digit integer right justified to 6 column: 9876

4 digit integer right justified to 3 column: 9876

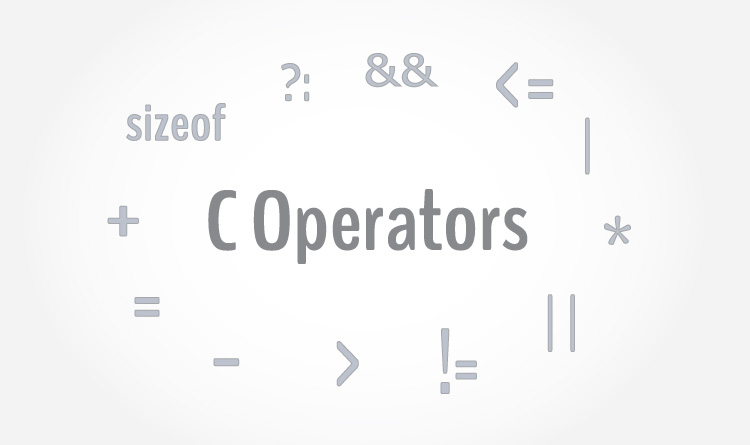
Floating point number rounded to 2 digits: 987.65

Floating point number rounded to 0 digits: 988

Floating point number in exponential form: 9.876543e+02

# C Programming Operators

C programming has various operators to perform tasks including arithmetic, conditional and bitwise operations. You will learn about various C operators and how to use them in this tutorial.



An operator is a symbol which operates on a value or a variable. For example: + is an operator to perform addition.

<="">

C programming has wide range of operators to perform various operations. For better understanding of operators, these operators can be classified as:

|  |
| --- |
| Operators in C programming |
| Arithmetic Operators |
| Increment and Decrement Operators |
| Assignment Operators |
| Relational Operators |
| Logical Operators |
| Conditional Operators |
| Bitwise Operators |
| Special Operators |

## C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

| Operator | Meaning of Operator |
| --- | --- |
| + | addition or unary plus |
| - | subtraction or unary minus |
| \* | multiplication |
| / | division |
| % | remainder after division( modulo division) |

### Example #1: Arithmetic Operators

// C Program to demonstrate the working of arithmetic operators

#include <stdio.h>

int main()

{

int a = 9,b = 4, c;

c = a+b;

printf("a+b = %d \n",c);

c = a-b;

printf("a-b = %d \n",c);

c = a\*b;

printf("a\*b = %d \n",c);

c=a/b;

printf("a/b = %d \n",c);

c=a%b;

printf("Remainder when a divided by b = %d \n",c);

return 0;

}

**Output**

a+b = 13

a-b = 5

a\*b = 36

a/b = 2

Remainder when a divided by b=1

The operators +, - and \* computes addition, subtraction and multiplication respectively as you might have expected.

In normal calculation, 9/4 = 2.25. However, the output is 2 in the program.

It is because both variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after decimal point and shows answer 2 instead of 2.25.

The modulo operator % computes the remainder. When a = 9 is divided by b = 4, the remainder is 1. The % operator can only be used with integers.

Suppose a = 5.0, b = 2.0, c = 5 and d = 2. Then in C programming,

a/b = 2.5 // Because both operands are floating-point variables

a/d = 2.5 // Because one operand is floating-point variable

c/b = 2.5 // Because one operand is floating-point variable

c/d = 2 // Because both operands are integers

## Increment and decrement operators

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

### Example #2: Increment and Decrement Operators

// C Program to demonstrate the working of increment and decrement operators

#include <stdio.h>

int main()

{

int a = 10, b = 100;

float c = 10.5, d = 100.5;

printf("++a = %d \n", ++a);

printf("--b = %d \n", --b);

printf("++c = %f \n", ++c);

printf("--d = %f \n", --d);

return 0;

}

**Output**

++a = 11

--b = 99

++c = 11.500000

++d = 99.500000

Here, the operators ++ and -- are used as prefix. These two operators can also be used as postfix like a++ and a--. Visit this page to learn more on how increment and decrement operators work when used as postfix.

## C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

| Operator | Example | Same as |
| --- | --- | --- |
| = | a = b | a = b |
| += | a += b | a = a+b |
| -= | a -= b | a = a-b |
| \*= | a \*= b | a = a\*b |
| /= | a /= b | a = a/b |
| %= | a %= b | a = a%b |

## Example #3: Assignment Operators

// C Program to demonstrate the working of assignment operators

#include <stdio.h>

int main()

{

int a = 5, c;

c = a;

printf("c = %d \n", c);

c += a; // c = c+a

printf("c = %d \n", c);

c -= a; // c = c-a

printf("c = %d \n", c);

c \*= a; // c = c\*a

printf("c = %d \n", c);

c /= a; // c = c/a

printf("c = %d \n", c);

c %= a; // c = c%a

printf("c = %d \n", c);

return 0;

}

**Output**

c = 5

c = 10

c = 5

c = 25

c = 5

c = 0

### C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

<="">

Relational operators are used in decision making and loops.

| Operator | Meaning of Operator | Example |
| --- | --- | --- |
| == | Equal to | 5 == 3 returns 0 |
| > | Greater than | 5 > 3 returns 1 |
| < | Less than | 5 < 3 returns 0 |
| != | Not equal to | 5 != 3 returns 1 |
| >= | Greater than or equal to | 5 >= 3 returns 1 |
| <= | Less than or equal to | 5 <= 3 return 0 |

### Example #4: Relational Operators

// C Program to demonstrate the working of arithmetic operators

#include <stdio.h>

int main()

{

int a = 5, b = 5, c = 10;

printf("%d == %d = %d \n", a, b, a == b); // true

printf("%d == %d = %d \n", a, c, a == c); // false

printf("%d > %d = %d \n", a, b, a > b); //false

printf("%d > %d = %d \n", a, c, a > c); //false

printf("%d < %d = %d \n", a, b, a < b); //false

printf("%d < %d = %d \n", a, c, a < c); //true

printf("%d != %d = %d \n", a, b, a != b); //false

printf("%d != %d = %d \n", a, c, a != c); //true

printf("%d >= %d = %d \n", a, b, a >= b); //true

printf("%d >= %d = %d \n", a, c, a >= c); //false

printf("%d <= %d = %d \n", a, b, a <= b); //true

printf("%d <= %d = %d \n", a, c, a <= c); //true

return 0;

}

**Output**

5 == 5 = 1

5 == 10 = 0

5 > 5 = 0

5 > 10 = 0

5 < 5 = 0

5 < 10 = 1

5 != 5 = 0

5 != 10 = 1

5 >= 5 = 1

5 >= 10 = 0

5 <= 5 = 1

5 <= 10 = 1

### C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

| Operator | Meaning of Operator | Example |
| --- | --- | --- |
| && | Logial AND. True only if all operands are true | If c = 5 and d = 2 then, expression ((c == 5) && (d > 5)) equals to 0. |
| || | Logical OR. True only if either one operand is true | If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1. |
| ! | Logical NOT. True only if the operand is 0 | If c = 5 then, expression ! (c == 5) equals to 0. |

### Example #5: Logical Operators

// C Program to demonstrate the working of logical operators

#include <stdio.h>

int main()

{

int a = 5, b = 5, c = 10, result;

result = (a = b) && (c > b);

printf("(a = b) && (c > b) equals to %d \n", result);

result = (a = b) && (c < b);

printf("(a = b) && (c < b) equals to %d \n", result);

result = (a = b) || (c < b);

printf("(a = b) || (c < b) equals to %d \n", result);

result = (a != b) || (c < b);

printf("(a != b) || (c < b) equals to %d \n", result);

result = !(a != b);

printf("!(a == b) equals to %d \n", result);

result = !(a == b);

printf("!(a == b) equals to %d \n", result);

return 0;

}

**Output**

(a = b) && (c > b) equals to 1

(a = b) && (c < b) equals to 0

(a = b) || (c < b) equals to 1

(a != b) || (c < b) equals to 0

!(a != b) equals to 1

!(a == b) equals to 0

**Explanation of logical operator program**

* (a = b) && (c > 5) evaluates to 1 because both operands (a = b) and (c > b) is 1 (true).
* (a = b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).
* (a = b) || (c < b) evaluates to 1 because (a = b) is 1 (true).
* (a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).
* !(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).
* !(a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false).

### Bitwise Operators

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

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

Visit bitwise operator in C to learn more.

## Other Operators

### Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

### The sizeof operator

The sizeof is an unary operator which returns the size of data (constant, variables, array, structure etc).

### Example #6: sizeof Operator

#include <stdio.h>

int main()

{

int a, e[10];

float b;

double c;

char d;

printf("Size of int=%lu bytes\n",sizeof(a));

printf("Size of float=%lu bytes\n",sizeof(b));

printf("Size of double=%lu bytes\n",sizeof(c));

printf("Size of char=%lu byte\n",sizeof(d));

printf("Size of integer type array having 10 elements = %lu bytes\n", sizeof(e));

return 0;

}

**Output**

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte

Size of integer type array having 10 elements = 40 bytes

### C Ternary Operator (?:)

A conditional operator is a ternary operator, that is, it works on 3 operands.

#### Conditional Operator Syntax

conditionalExpression ? expression1 : expression2

The conditional operator works as follows:

* The first expression conditionalExpression is evaluated at first. This expression evaluates to 1 if it's and evaluates to 0 if it's false.
* If conditionalExpression is true, expression1 is evaluated.
* If conditionalExpression is false, expression2 is evaluated.

#### Example #7: C conditional Operator

#include <stdio.h>

int main(){

char February;

int days;

printf("If this year is leap year, enter 1. If not enter any integer: ");

scanf("%c",&February);

// If test condition (February == 'l') is true, days equal to 29.

// If test condition (February =='l') is false, days equal to 28.

days = (February == '1') ? 29 : 28;

printf("Number of days in February = %d",days);

return 0;

}

**Output**

If this year is leap year, enter 1. If not enter any integer: 1

Number of days in February = 29