**C FUNDAMENTALS**

**C Keywords and Identifiers**

In this tutorial, you will learn about keywords; reserved words in C programming that are part of the syntax. Also, you will learn about identifiers and how to name them.

**Character set**

A 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

C accepts both lowercase and uppercase alphabets as variables and functions.

**Digits**

0 1 2 3 4 5 6 7 8 9

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

**White space Characters**

Blank space, newline, horizontal tab, carriage return and form feed.

**C Keywords**

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. 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](https://www.programiz.com/c-programming/c-variables-constants) of type int (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.

|  |  |  |  |
| --- | --- | --- | --- |
| C Keywords | | | |
| 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 |

All these keywords, their syntax, and application will be discussed in their respective topics. However, if you want a brief overview of these keywords without going further, visit [List of all keywords in C programming](https://www.programiz.com/c-programming/list-all-keywords-c-language).

## C Identifiers

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

Identifiers must be unique. They are created to give a unique name to an 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 naming identifiers

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.
3. You cannot use keywords like int, while etc. as identifiers.
4. There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.

You can choose any name as an identifier if you follow the above rule, however, give meaningful names to identifiers that make sense.

# C Variables, Constants and Literals

In this tutorial, you will learn about variables and rules for naming a variable. You will also learn about different literals in C programming and how to create constants.

**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](https://www.programiz.com/c-programming/c-keywords-identifier)). Variable names are just the symbolic representation of a memory location. For example:

int playerScore = 95;

Here, playerScore is a variable of int type. Here, the variable is assigned an integer value 95.

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

char ch = 'a';

// some code

ch = 'l';

**Rules for naming a variable**

1. A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
2. The first letter of a variable should be either a letter or an underscore.
3. There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.

**Note:** You should always try to give meaningful names to variables. For example: firstName is a better variable name than fn.

C is a strongly typed language. This means that the variable type cannot be changed once it is declared. For example:

int number = 5; // integer variable

number = 5.5; // error

double number; // error

Here, the type of number variable is int. You cannot assign a floating-point (decimal) value 5.5 to this variable. Also, you cannot redefine the data type of the variable to double. By the way, to store the decimal values in C, you need to declare its type to either double or float.

Visit this page to learn more about [different types of data a variable can store](https://www.programiz.com/c-programming/c-data-types).

**Literals**

Literals are data used for representing fixed values. They can be used directly in the code. For example: 1, 2.5, 'c' etc.

Here, 1, 2.5 and 'c' are literals. Why? You cannot assign different values to these terms.

**1. Integers**

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

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

For example:

Decimal: 0, -9, 22 etc

Octal: 021, 077, 033 etc

Hexadecimal: 0x7f, 0x2a, 0x521 etc

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

**2. Floating-point Literals**

A floating-point literal is a numeric literal 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. Characters**

A character literal is created by enclosing a single character inside single quotation marks. For example: 'a', 'm', 'F', '2', '}' etc.

**4. Escape Sequences**

Sometimes, it is necessary to use characters that 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 sequences are used.

| 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 |

For example: \n is used for a newline. The backslash \ causes escape from the normal way the characters are handled by the compiler.

**5. String Literals**

A string literal is a sequence of characters enclosed in double-quote marks. For example:

"good" //string constant

"" //null string constant

" " //string constant of six white space

"x" //string constant having a single character.

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

**Constants**

If you want to define a variable whose value cannot be changed, you can use the const keyword. This will create a constant. For example,

const double PI = 3.14;

Notice, we have added keyword const.

Here, PI is a symbolic constant; its value cannot be changed.

const double PI = 3.14;

PI = 2.9; //Error

# C Data Types

In this tutorial, you will learn about basic data types such as int, float, char etc. in C programming.

In C programming, data types are declarations for variables. This determines the type and size of data associated with variables. For example,

int myVar;

Here, myVar is a variable of int (integer) type. The size of int is 4 bytes.

## Basic types

Here's a table containing commonly used types in C programming for quick access.

| Type | Size (bytes) | Format Specifier |
| --- | --- | --- |
| int | at least 2, usually 4 | %d, %i |
| char | 1 | %c |
| float | 4 | %f |
| double | 8 | %lf |
| short int | 2 usually | %hd |
| unsigned int | at least 2, usually 4 | %u |
| long int | at least 4, usually 8 | %ld, %li |
| long long int | at least 8 | %lld, %lli |
| unsigned long int | at least 4 | %lu |
| unsigned long long int | at least 8 | %llu |
| signed char | 1 | %c |
| unsigned char | 1 | %c |
| long double | at least 10, usually 12 or 16 | %Lf |

### int

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

We can use int for declaring an integer variable.

int id;

Here, id is a variable of type integer.

You can declare multiple variables at once in C programming. For example,

int id, age;

The size of int is usually 4 bytes (32 bits). And, it can take 232 distinct states from -2147483648 to 2147483647.

### float and double

float and double are used to hold real numbers.

float salary;

double price;

In C, floating-point numbers can also be represented in exponential. For example,

float normalizationFactor = 22.442e2;

What's the 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.

**char**

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

char test = 'h';

The size of the character variable is 1 byte.

**void**

void is an incomplete type. It means "nothing" or "no type". You can think of void as **absent**.

For example, if a function is not returning anything, its return type should be void.

Note that, you cannot create variables of void type.

**short and long**

If you need to use a large number, you can use a type specifier long. Here's how:

long a;

long long b;

long double c;

Here variables a and b can store integer values. And, c can store a floating-point number.

If you are sure, only a small integer ([−32,767, +32,767] range) will be used, you can use short.

short d;

You can always check the size of a variable using the sizeof() operator.

#include <stdio.h>

int main() {

short a;

long b;

long long c;

long double d;

printf("size of short = %d bytes\n", sizeof(a));

printf("size of long = %d bytes\n", sizeof(b));

printf("size of long long = %d bytes\n", sizeof(c));

printf("size of long double= %d bytes\n", sizeof(d));

return 0;

}

**signed and unsigned**

In C, signed and unsigned are type modifiers. You can alter the data storage of a data type by using them:

* signed - allows for storage of both positive and negative numbers
* unsigned - allows for storage of only positive numbers

For example,

// valid codes

unsigned int x = 35;

int y = -35; // signed int

int z = 36; // signed int

// invalid code: unsigned int cannot hold negative integers

unsigned int num = -35;

Here, the variables x and num can hold only zero and positive values because we have used the unsigned modifier.

Considering the size of int is 4 bytes, variable y can hold values from -231 to 231-1, whereas variable x can hold values from 0 to 232-1.

**Derived Data Types**

Data types that are derived from fundamental data types are derived types. For example: arrays, pointers, function types, structures, etc.

# C Input Output (I/O)

In this tutorial, you will learn to use scanf() function to take input from the user, and printf() function to display output to the user.

**C Output**

In C programming, printf() is one of the main output function. The function sends formatted output to the screen. For example,

**Example 1: C Output**

#include <stdio.h>

int main()

{

// Displays the string inside quotations

printf("C Programming");

return 0;

}

**Output**

C Programming

How does this program work?

* All valid C programs must contain the main() function. The code execution begins from the start of the main() function.
* The printf() is a library function to send formatted output to the screen. The function prints the string inside quotations.
* To use printf() in our program, we need to include stdio.h header file using the #include <stdio.h> statement.
* The return 0; statement inside the main() function is the "Exit status" of the program. It's optional.

**Example 2: Integer Output**

#include <stdio.h>

int main()

{

int testInteger = 5;

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

return 0;

}

**Output**

Number = 5

We use %d format specifier to print int types. Here, the %d inside the quotations will be replaced by the value of testInteger.

**Example 3: float and double Output**

#include <stdio.h>

int main()

{

float number1 = 13.5;

double number2 = 12.4;

printf("number1 = %f\n", number1);

printf("number2 = %lf", number2);

return 0;

}

**Output**

number1 = 13.500000

number2 = 12.400000

To print float, we use %f format specifier. Similarly, we use %lf to print double values.

**Example 4: Print Characters**

#include <stdio.h>

int main()

{

char chr = 'a';

printf("character = %c", chr);

return 0;

}

**Output**

character = a

To print char, we use %c format specifier.

**C Input**

In C programming, scanf() is one of the commonly used function to take input from the user. The scanf() function reads formatted input from the standard input such as keyboards.

**Example 5: 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

Here, we have used %d format specifier inside the scanf() function to take int input from the user. When the user enters an integer, it is stored in the testInteger variable.

Notice, that we have used &testInteger inside scanf(). It is because &testInteger gets the address of testInteger, and the value entered by the user is stored in that address.

**Example 6: Float and Double Input/Output**

#include <stdio.h>

int main()

{

float num1;

double num2;

printf("Enter a number: ");

scanf("%f", &num1);

printf("Enter another number: ");

scanf("%lf", &num2);

printf("num1 = %f\n", num1);

printf("num2 = %lf", num2);

return 0;

}

**Output**

Enter a number: 12.523

Enter another number: 10.2

num1 = 12.523000

num2 = 10.200000

We use %f and %lf format specifier for float and double respectively.

**Example 7: 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

When a character is entered by the user in the above program, the character itself is not stored. Instead, an integer value (ASCII value) is stored.

And when we display that value using %c text format, the entered character is displayed. If we use %d to display the character, it's ASCII value is printed.

**Example 8: ASCII Value**

#include <stdio.h>

int main()

{

char chr;

printf("Enter a character: ");

scanf("%c", &chr);

// When %c is used, a character is displayed

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

// When %d is used, ASCII value is displayed

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

return 0;

}

Output

Enter a character: g

You entered g.

ASCII value is 103.

**I/O Multiple Values**

Here's how you can take multiple inputs from the user and display them.

#include <stdio.h>

int main()

{

int a;

float b;

printf("Enter integer and then a float: ");

// Taking multiple inputs

scanf("%d%f", &a, &b);

printf("You entered %d and %f", a, b);

return 0;

}

**Output**

Enter integer and then a float: -3

3.4

You entered -3 and 3.400000

**Format Specifiers for I/O**

As you can see from the above examples, we use

* %d for int
* %f for float
* %lf for double
* %c for char

Here's a list of commonly used C data types and their format specifiers.

| Data Type | Format Specifier |
| --- | --- |
| int | %d |
| char | %c |
| float | %f |
| double | %lf |
| short int | %hd |
| unsigned int | %u |
| long int | %li |
| long long int | %lli |
| unsigned long int | %lu |
| unsigned long long int | %llu |
| signed char | %c |
| unsigned char | %c |
| long double | %Lf |

# C Programming Operators

In this tutorial, you will learn about different operators in C programming with the help of examples.

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

C has a wide range of operators to perform various operations.

**C Arithmetic Operators**

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc 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**

// 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 the variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after the 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,

// Either one of the operands is a floating-point number

a/b = 2.5

a/d = 2.5

c/b = 2.5

// Both operands are integers

c/d = 2

**C 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**

// 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 prefixes. These two operators can also be used as postfixes like a++ and a--. Visit this page to learn more about how [increment and decrement operators work when used as postfix](https://www.programiz.com/article/increment-decrement-operator-difference-prefix-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**

// Working of assignment operators

#include <stdio.h>

int main()

{

int a = 5, c;

c = a; // c is 5

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

c += a; // c is 10

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

c -= a; // c is 5

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

c \*= a; // c is 25

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

c /= a; // c is 5

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

c %= a; // c = 0

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](https://www.programiz.com/c-programming/c-if-else-statement) and [loops](https://www.programiz.com/c-programming/c-for-loop).

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

**Example 4: Relational Operators**

// Working of relational operators

#include <stdio.h>

int main()

{

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

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

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

printf("%d > %d is %d \n", a, b, a > b);

printf("%d > %d is %d \n", a, c, a > c);

printf("%d < %d is %d \n", a, b, a < b);

printf("%d < %d is %d \n", a, c, a < c);

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

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

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

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

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

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

return 0;

}

**Output**

5 == 5 is 1

5 == 10 is 0

5 > 5 is 0

5 > 10 is 0

5 < 5 is 0

5 < 10 is 1

5 != 5 is 0

5 != 10 is 1

5 >= 5 is 1

5 >= 10 is 0

5 <= 5 is 1

5 <= 10 is 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](https://www.programiz.com/c-programming/c-if-else-statement).

| Operator | Meaning | Example |
| --- | --- | --- |
| && | Logical 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**

// 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) is %d \n", result);

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

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

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

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

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

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

result = !(a != b);

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

result = !(a == b);

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

return 0;

}

**Output**

(a == b) && (c > b) is 1

(a == b) && (c < b) is 0

(a == b) || (c < b) is 1

(a != b) || (c < b) is 0

!(a != b) is 1

!(a == b) is 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).

**C Bitwise Operators**

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc 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](https://www.programiz.com/c-programming/bitwise-operators) 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 a unary operator that returns the size of data (constants, variables, array, structure, etc).

**Example 6: sizeof Operator**

#include <stdio.h>

int main()

{

int a;

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));

return 0;

}

**Output**

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte