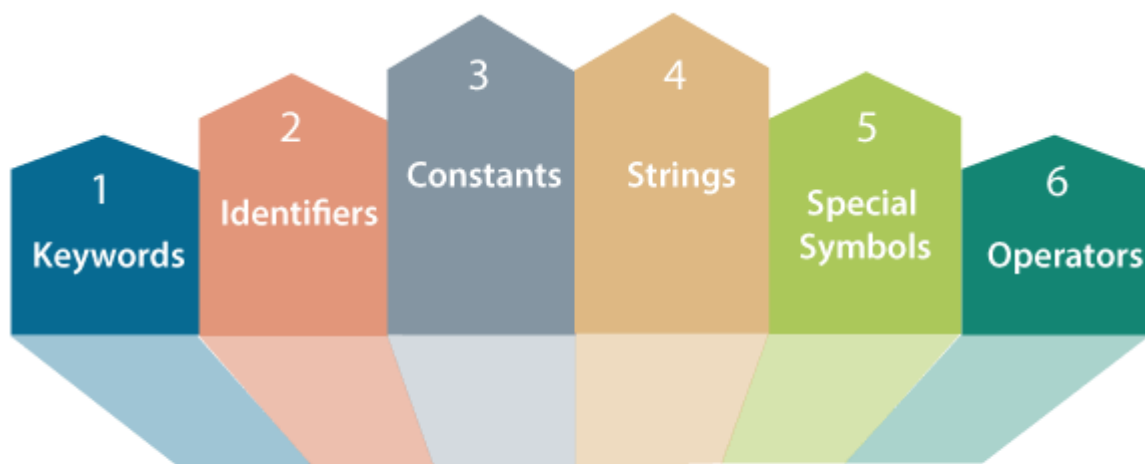


Tokens in C:

Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a [program in C language](#).

Classification of tokens in C

Tokens in [C language](#) can be divided into the following categories:



Classification of C Tokens

- Keywords in C
- Identifiers in C
- Strings in C
- Operators in C
- Constant in C
- Special Characters in C

Let's understand each token one by one.

Keywords in C

[Keywords in C](#) can be defined as the **pre-defined** or the **reserved words** having its own importance, and each keyword has its own functionality. Since keywords are the pre-defined words used by the compiler, so they cannot be used as the variable names. If the keywords are used as the variable names, it means that we are assigning a different meaning to the keyword, which is not allowed. C language supports 32 keywords given below:

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

Identifiers in C

[Identifiers in C](#) are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet. Identifiers cannot be used as keywords. Rules for constructing identifiers in C are given below:

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

Strings in C

[Strings in C](#) are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

```
char a[10] = "javatpoint"; // The compiler allocates the 10 bytes to the 'a' array.
```

```
char a[] = "javatpoint"; // The compiler allocates the memory at the run time.
```

```
char a[10] = {'j','a','v','a','t','p','o','i','n','t','\0'}; // String is represented in the form of characters.
```

Operators in C

[Operators in C](#) is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

Unary Operator

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

Binary Operator

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Conditional Operators
- Assignment Operator
- Misc Operator

Constants in C

A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

There are two ways of declaring constant:

- Using const keyword
- Using #define pre-processor

Types of [constants in C](#)

Constant

Example

Integer constant

10, 11, 34, etc.

Floating-point constant

45.6, 67.8, 11.2, etc.

Octal constant	011, 088, 022, etc.
Hexadecimal constant	0x1a, 0x4b, 0x6b, etc.
Character constant	'a', 'b', 'c', etc.
String constant	"java", "c++", ".net", etc.

Special characters in C

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets []:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ():** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.