Concepts of C

C Programming Language Tutorial

What is C Language?

C language Tutorial with programming approach for beginners and professionals, helps you to understand the C language tutorial easily. Our C tutorial explains each topic with programs.

The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways:

1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

1) C as a mother language

C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language,** and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts **like the array, strings, functions, file handling**, etc. that are being used in many languages like **C++, Java, C#,** etc.

2) C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel).** It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

3) C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem.**

A procedural language breaks the program into functions, data structures, etc.

C is a procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

A structured programming language is a subset of the procedural language. **Structure means to break a program into parts or blocks** so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

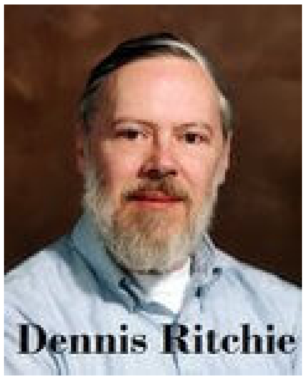
5) C as a mid-level programming language

C is considered as a middle-level language because it supports the feature of both low-level and high-level languages. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.

A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

History of C Language



History of C language is interesting to know. Here we are going to discuss a brief history of the c language.

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

Dennis Ritchie is known as the **founder of the c language.**

It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Features of C Language

C is the widely used language. It provides many **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

1) Simple

C is a simple language in the sense that it provides a structured approach (to break the problem into parts), the rich set of library functions, data types, etc.

2) Machine Independent or Portable

Unlike assembly language, c programs **can be executed on different machines** with some machine specific changes. Therefore, C is a machine independent language.

3) Mid-level programming language

Although, C is **intended to do low-level programming.** It is used to develop system applications such as kernel, driver, etc. It **also supports the features of a high-level language**. That is why it is known as mid-level language.

4) Structured programming language

C is a structured programming language in the sense that **we can break the program into parts using functions.** So, it is easy to understand and modify. Functions also provide code reusability.

5) Rich Library

C provides a lot of inbuilt functions that make the development fast.

6) Memory Management

It supports the feature of **dynamic memory allocation**. In C language, we can free the allocated memory at any time by calling the **free()** function.

7) Speed

The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

8) Pointer

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We **can use pointers for memory, structures, functions, array**, etc.

9) Recursion

In C, we **can call the function within the function.** It provides code reusability for every function. Recursion enables us to use the approach of backtracking.

10) Extensible

C language is extensible because it **can easily adopt new features**.

First C Program

Before starting the abcd of C language, you need to learn how to write, compile and run the first c program.

To write the first c program, open the C console and write the following code:

```
#include <stdio.h>
int main()
{
    printf("Hello C Language");
    return 0;
}
```

#include <stdio.h> includes the **standard input output** library functions. The printf() function is defined in stdio.h .

int main() The main() function is the entry point of every program in c language.

printf() The printf() function is **used to print data** on the console.

return 0 The return 0 statement, returns execution status to the OS. The 0 value is used for successful execution and 1 for unsuccessful execution.

printf() and scanf() in C

The **printf()** and **scanf()** functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

➢ printf() function

The **printf() function** is used for output. It prints the given statement to the console.

The syntax of printf() function is given below:

printf("format string",argument_list);

example : printf("Hello World! ");

The format string can be %d (integer), %c (character), %s (string), %f (float) etc.

➢ scanf() function

The **scanf() function** is used for input. It reads the input data from the console.

scanf("format string",argument_list);

example : scanf("%d",&a);

Program to print cube of given number

Let's see a simple example of c language that gets input from the user and prints the square of the given number.

```
#include<stdio.h>
int main()
{
   int num;
   printf("enter a number:");
   scanf("%d",&num);

   printf("cube of number is: %d ",num*num*num);

   return 0;
}
```

Variables in C

Variable is a name given to memory location; it is used to store data/values of different types".

Features

- Values of the variables may be changed during program execution.
- Variable name must be a valid identifier.
- Variables are containers to hold values.
- At a time one variable can hold maximum one value

Variable Declarations/Creation

Syntax : datatype variablename;

Ex: int x; float y; double d; char ch;

Variable Initialization

Variable initialization means storing some initial value to the variable.

Syntax : variable = value ;

Example1: int num; num = 10

Rules for defining the variable :

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Types of Variables in C

There are many types of variables in c :

- local variable
- global variable
- static variable
- automatic variable
- external variable

Local Variable

- A variable that is declared inside the function or block is called a local variable.
- It must be declared at the start of the block.

```
Example
void function1()
{
   int x=10;//local variable
}
```

You must have to initialize the local variable before it is used.

Global Variable

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

It must be declared at the start of the block.

```
int value=20;//global variable
void function1()
{
    int x=10;//local variable
}
```

Static Variable

A variable that is declared with the static keyword is called static variable.It retains its value between multiple function calls.

```
void function1()
{
   int x=10;//local variable
   static int y=10;//static variable
   x=x+1;
   y=y+1;
   printf("%d,%d",x,y);
}
```

Automatic Variable

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

```
void main()

{

   int x=10;//local variable (also automatic)

   auto int y=20;//automatic variable

}
```

External Variable

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern keyword**.

```
extern int x=10;//external variable (also global)
```

```
#include "myfile.h"
#include <stdio.h>
void printValue()
{
   printf("Global variable: %d", global_variable);
}
```

Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

There are the following data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Data Types	Memory Size	Range
char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	
double	8 byte	
long double	10 byte	
char	1 byte	−128 to 127

Keywords in C

A keyword is a **reserved word**. You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.

A list of 32 keywords in the c language is given below :

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

Identifiers: -

"A Name in C Program is called an identifier, it can be a variable name, a function name, a structure name, label etc. these are user defined words".

Uses: -

To give a name to entities like variable, function, labels, structures in C program etc.
Identifiers are used for identification purpose.

Rules for Defining an Identifier.

a. Allowed Characters are: -
   i. Alphabets: - Uppercase(A-Z) & Lowercase(a-z)
   ii. Digits: - (0-9)
   iii. Special Symbols: _ (Underscore), $ (Dollar)
b. Identifier should not start with digits.
c. Identifier should not be a keyword.
d. Identifiers are case sensitive.
e. Identifier name must be unique in a program.
f. There is no space between an identifier word.

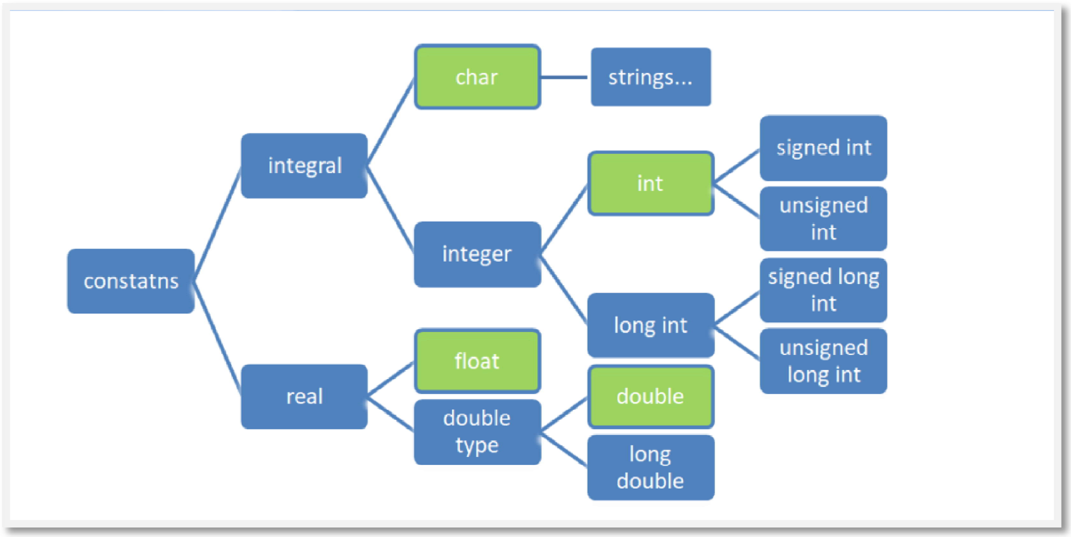
Constants:

Constant : It is value that cannot be changed/altered by the program during execution of the program.

These are fixed value in C. These are also called as literals.

Any value which is directly referred in a program is a constant.
Ex: 5, 44.55, ‘á’, “KLE”, ‘ ’

Constants are classified as:



List of Constants in C

Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

2 ways to define constant in C

There are two ways to define constant in C programming.

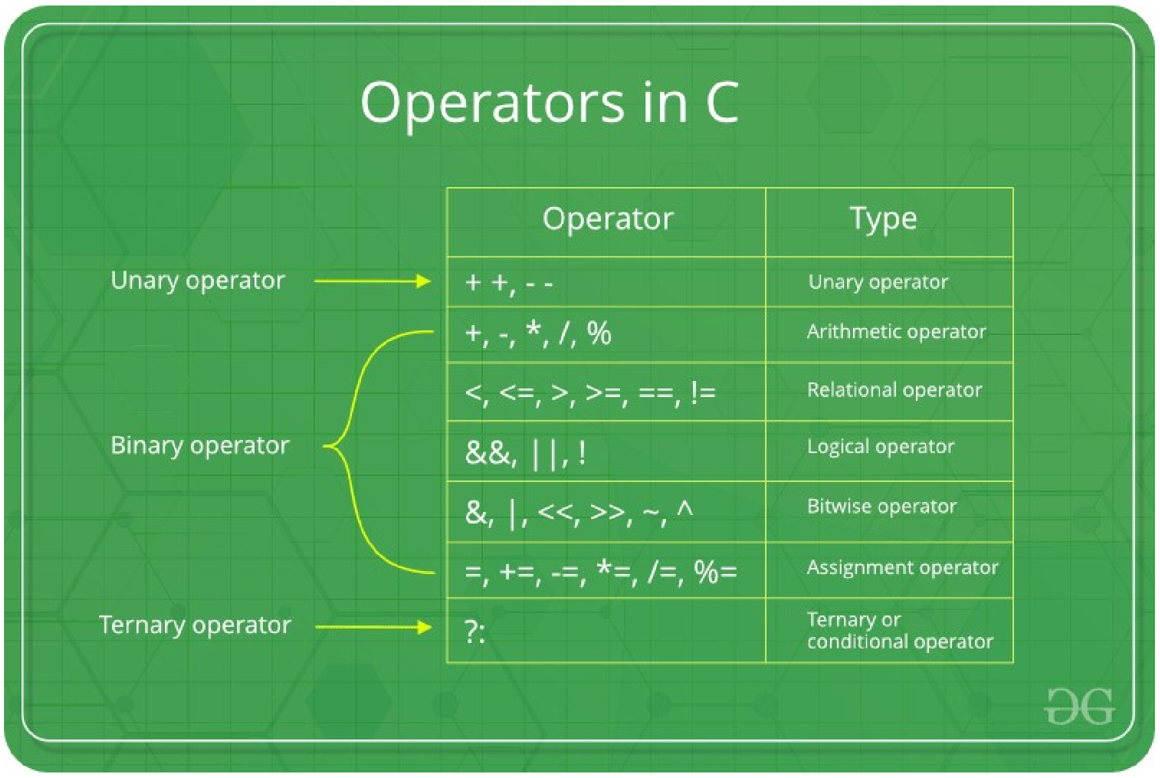
1. const keyword
2. #define preprocessor

C Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

1. Arithmetic Operators
2. Relational Operators
3. Shift Operators
4. Logical Operators
5. Bitwise Operators
6. Ternary or Conditional Operators
7. Assignment Operator
8. Misc Operator


Operators in C
Unary operator
Binary operator
Ternary operator

Operator	Type
+ +, - -	Unary operator
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
&&, ||, !	Logical operator
&, |, <<, >>, ~, ^	Bitwise operator
=, +=, -=, *=, /=, %=	Assignment operator
?:	Ternary or conditional operator

Precedence of Operators in C

- The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.
- Let's understand the precedence by the example given below :

  int value=10+20*10;

- The value variable will contain 210 because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below :

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Comments in C

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments
2. Multi-Line Comments

Single Line Comments

Single line comments are represented by double slash \\. Let's see an example of a single line comment in C.

```
#include<stdio.h>
int main()
{
   //printing information
   printf("Hello C");
   return 0;
}
```

Even you can place the comment after the statement. For example

```
printf("Hello C");//printing information
```

Multi Line Comments

Multi-Line comments are represented by slash asterisk * ... *\. It can occupy many lines of code, but it can't be nested. Syntax

```
/*
code
to be commented
*/
```

Let's see an example of a multi-Line comment in C.

```
#include<stdio.h>
int main()
{
   /*printing information
   Multi-Line Comment*/
   printf("Hello C");
           return 0;
}
```

C Format Specifier

The Format specifier is a string used in the formatted input and output functions. The format string determines the format of the input and output. The format string always starts with a '%' character.

The commonly used format specifiers in printf() function are:

Format specifier	Description
%d or %i	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.
%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.
%X	It is used to print the hexadecimal unsigned integer, but %X prints the alphabetical characters in uppercase such as A, B, C, etc.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
%e/%E	It is used for scientific notation. It is also known as Mantissa or Exponent.
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%ld	It is used to print the long-signed integer value.

Escape Sequence in C

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

It is composed of two or more characters starting with backslash \. For example: \n represents new line.

List of Escape Sequences in C

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

Escape Sequence Example

```
#include<stdio.h>
int main()
{
    int number = 50;
    printf("You\nare\nlearning\n\'c\' language\n\"Do you know C language\"");
    return 0;
}
```

Output :

```
You
are
learning
'c' language
"Do you know C language"
```

ASCII value in C

What is ASCII code?

The full form of ASCII is the **American Standard Code for information interchange**. It is a character encoding scheme used for electronics communication. Each character or a special character is represented by some ASCII code, and each ascii code occupies 7 bits in memory.

In C programming language, a character variable does not contain a character value itself rather the ascii value of the character variable.

The ascii value represents the character variable in numbers, and each character variable is assigned with some number range from 0 to 127. For example, the ascii value of 'A' is 65.

In the above example, we assign 'A' to the character variable whose ascii value is 65, so 65 will be stored in the character variable rather than 'A'.

Let's understand through an example.

We will create a program which will display the ascii value of the character variable.

```
#include <stdio.h>
int main()
{
   char ch;    // variable declaration
   printf("Enter a character : ");
   scanf("%c",&ch);  // user input

   printf("\n The ascii value of the ch variable is : %d", ch);
   return 0;
}
```

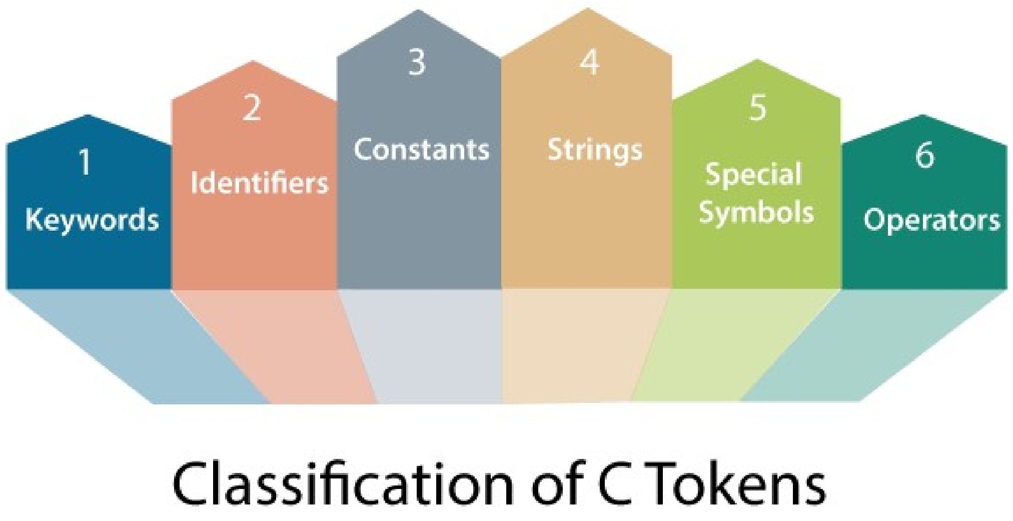
```
Output :
Enter a character : g
The ascii value of the ch variable is : 103
```

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:



- 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:

1. Arithmetic Operators
2. Relational Operators
3. Shift Operators
4. Logical Operators
5. Bitwise Operators
6. Conditional Operators
7. Assignment Operator
8. 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.

C Control Statement : Control statements are used to control the flow of execution since C language is asequential programming language.

If we want to **execute or skip** some parts/statements of the program as per requirement, then we can go for control statements.

1. Decision Making Statements: simple if, if-else, nested if-else, else-if ladder

2. Selection Statements: switch statement

3. Looping or Iterative Statements: for, while, do-while

4. Jump or Branching Statements: break, continue, return, goto

1) Decision Making :

Based Condition/Expression Result we can take decision to execute parts of the program.

1.1 simple if () statement: if the expression result is true then if body will execute, if the expression result is false then if body will not execute. If the condition returns false then the statements inside "if" are skipped.

Syntax :

```
if (Expression/Condition)
{
statement1; statement2;
}
Next_Statement;
```

Ex :

```
#include<stdio.h>
int main()
{
   int num; num = 101;
   if(num == 100)
   {
      printf("Yes num is equals to 100");
   }
   printf("\n I am out of if body...");

   return 0;
}
```

Output :

```
I am out of if body...
```

1.1 if else statement :

If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped.

If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.

Syntax :

```
if (Expression)
{
   statement1;
}
else
{
   statement2;

}
```

Ex :

```
#include<stdio.h>
int main()
{
   int a,b;
   a = 10;
   b = 20;

   if(a>b)
   {
      printf("\nMaximum = %d",a);
   }
   else
   {
      printf("\nMaximum = %d",b);
   }

   return 0;

}
```

Output :

```
Maximum = 20
```

1.2 nested if-else statement

Nesting means defining if-else body into another if-else body, Execution of innerif-else body depends on outer expression result.

Syntax :

```
if (Expression1)
{
   if (Expression2)
   {
   Statement1;
   }
   else
   {
   Statement2;
   }

}

else

{

   Statement3;

}
```

Ex :

```
#include<stdio.h>
int main()
{
   int a,b,c;
   printf("\nEnter a Number : ");
   scanf("%d%d%d",&a,&b,&c);

   if(a>=b && a>=c)
   {
      printf("%d is Maximum",a);
   }
   else if(b>=a && b>=c)
   {
       printf("%d is Maximum",b);
   }
   else
   {
       printf("%d is Maximum",c);
   }
}
```

Output :

```
Enter a Number : 10 20 30
30 is Maximum
```

else-if ladder/if-else-if ladder

else-if ladder is used to execute a block of code that is decided from multiple options/Conditions.

If any condition becomes true then the statements associated with that if body isexecuted, and the rest of the C else-if ladder is bypassed(skipped).

Syntax :

```
if (Expression1)
{
   statement1;
}
else if (Expression2)
{
   statement2;
}
else if (Expression3)
{
   statement3;
}
else if (Expression4)
{
   statement4;
}
………
else
{
   Statement n;
}
Next_Statement;
```

Ex :

```
#include<stdio.h>
int main()
{
   int age,temp=0;

      printf("\n\nEnter Your Age Here :");
      scanf("%d",&age);

      if(age>=18 && age<100)
      {
         printf("\n Congratulation !!!! You are Eligible for Voting ");
      }
      else if(age>=100)
      {
         printf("\nYou are Eligible But We Suggest Take a Rest ");
      }
      else
      {
         printf("\nYou Are Not Eligible or You are Not Mature ");

      }

  return 0;
}
```

Output :

```
Enter Your Age Here :20
Congratulation !!!! You are Eligible for Voting
```

2. Selection Statements: switch statement

Keywords: **switch, case, break, default.**

- switch statement is used to execute block of statements if any matching case isfound in the switch body, with respect switch variable/value.
- If any matching case is found then default case will execute.
- Break statement is used to exit/come out of from switch body.
- Switch works only with integers and characters values. Hence variables must be ofint or char type.

Syntax :

```
switch (Variable/value)
{
   case value1: statements;
         break;
   case value2: statements;
         break;
   case valueN: statements;
           break;
   default : statements;
}
```

Ex1:

```
Int x;
X=3;
switch(x)
{
          case 1: printf("\n Result = 1");
                    break;
          case 2: printf("\n Result = 2");
                    break;
          case 3: printf("\n Result = 3");
                    break;
          case 4: printf("\n Result = 4");
                    break;
          default: printf("\n No Matching Value");
                    break;
}
```

Output: Result = 3

C Program to check whether a given character is Vowel or Consonant.

```
#include<stdio.h>
  int main()
  {
      char ch;

      printf("\nEnter a Character : ");
      scanf("%c", &ch);

      switch(ch)
      {

          case 'a' :
          case 'A' :
          case 'e' :
          case 'E' :
          case 'i' :
          case 'I' :
          case 'o' :
          case 'O' :
          case 'u' :
          case 'U' : printf("\n %c is Vowel",ch);break;
          default : printf("\n %c is Consonant",ch);break;
      }

  }
```

Output:

```
Enter a Character:
A is Vowel
```