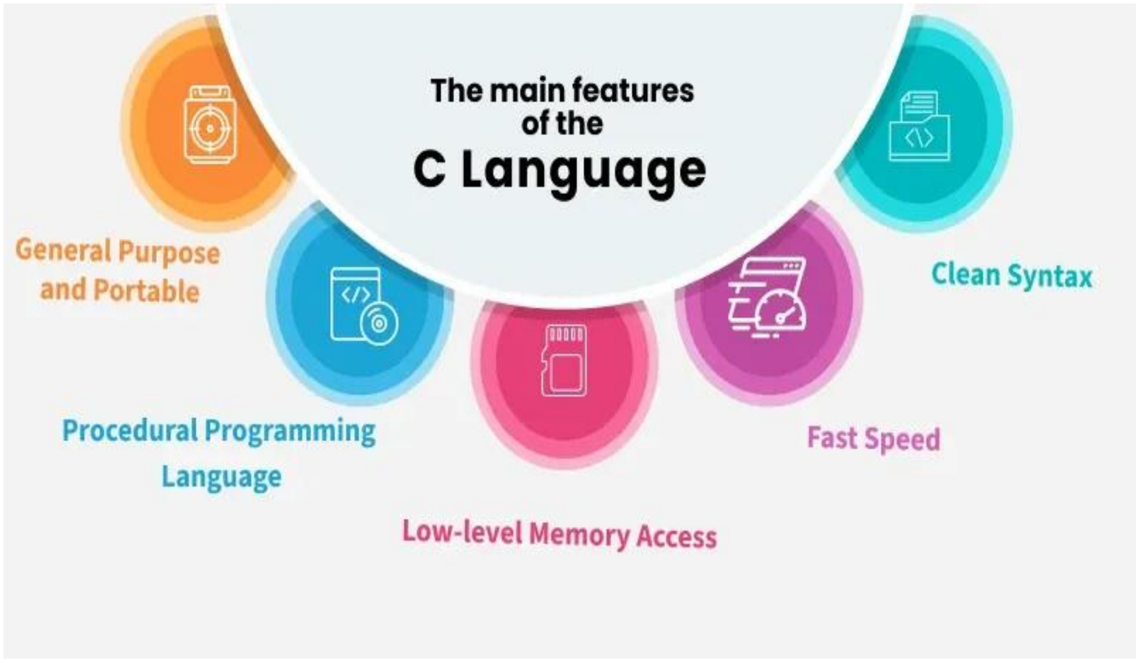
Introduction to C

C is a general-purpose procedural programming language initially developed by **Dennis Ritchie in 1972** at **Bell Laboratories of AT&T** Labs. It was mainly created as a system programming language to write the UNIX operating system.

Most Important Features of C Language:

1. Procedural Language
2. Fast and Efficient
3. Modularity
4. Statically Type
5. General-Purpose Language
6. Rich set of built-in Operators
7. Libraries with Rich Functions
8. Middle-Level Language
9. Portability
10. Easy to Extend



1. Procedural Language

In a procedural language like C step by step, predefined instructions are carried out. C program may contain more than one function to perform a particular task. New people to programming will think that this is the only way a particular programming language works. There are other programming paradigms as well in the programming world. Most of the commonly used paradigm is an object-oriented programming language.

2. Fast and Efficient
Newer languages like Java, python offer more features than C programming language but due to additional processing in these languages, their performance rate gets down effectively. C programming language as the middle-level language provides programmers access to direct manipulation with the computer hardware but higher-level languages do not allow this. That's one of the reasons C language is considered the first choice to start learning programming languages. It's fast because statically typed languages are faster than dynamically typed languages.

3. Modularity
The concept of storing C programming language code in the form of libraries for further future uses is known as modularity. This programming language can do very little on its own most of its power is held by its libraries. C language has its own library to solve common problems.

4. Statically Type
C programming language is a statically typed language. Meaning the type of variable is checked at the time of compilation but not at run time. This means each time a programmer types a program they have to mention the type of variables used.

5. General-Purpose Language
From system programming to photo editing software, the C programming language is used in various applications. Some of the common applications where it's used are as follows:

- Operating systems: Windows, Linux, iOS, Android, macOS
- Databases: PostgreSQL, Oracle, MySQL, MS SQL Server, etc.

6. Rich set of built-in Operators
It is a diversified language with a rich set of built-in operators which are used in writing complex or simplified C programs.

7. Libraries with Rich Functions
Robust libraries and functions in C help even a beginner coder to code with ease.

8. Middle-Level Language
As it is a middle-level language so it has the combined form of both capabilities of assembly language and features of the high-level language.

9. Portability
C language is lavishly portable as programs that are written in C language can run and compile on any system with either no or small changes.

10. Easy to Extend
Programs written in C language can be extended means when a program is already written in it then some more features and operations can be added to it.

C Program to Print "Hello World"

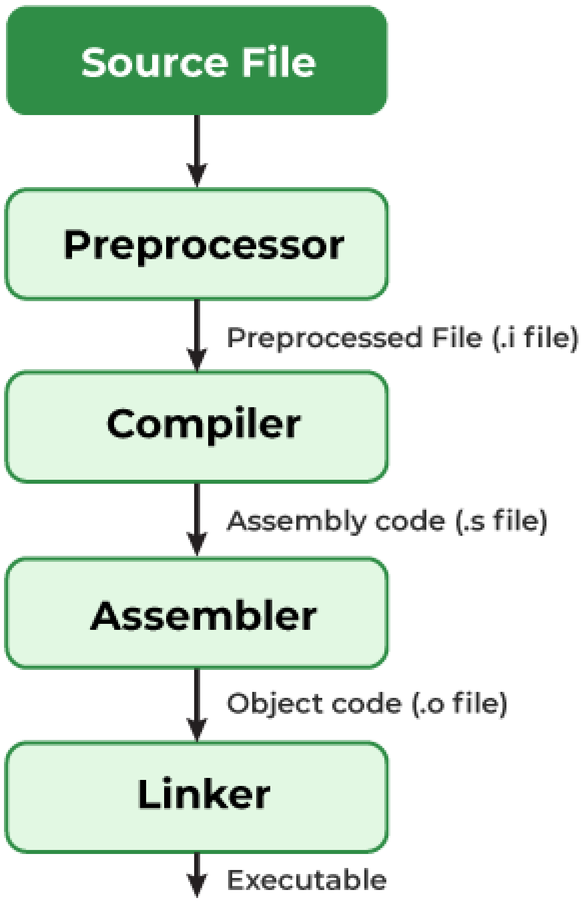
```
#include <stdio.h>
int main()
{
   printf("Hello World");
   return 0;
}
```

Output:

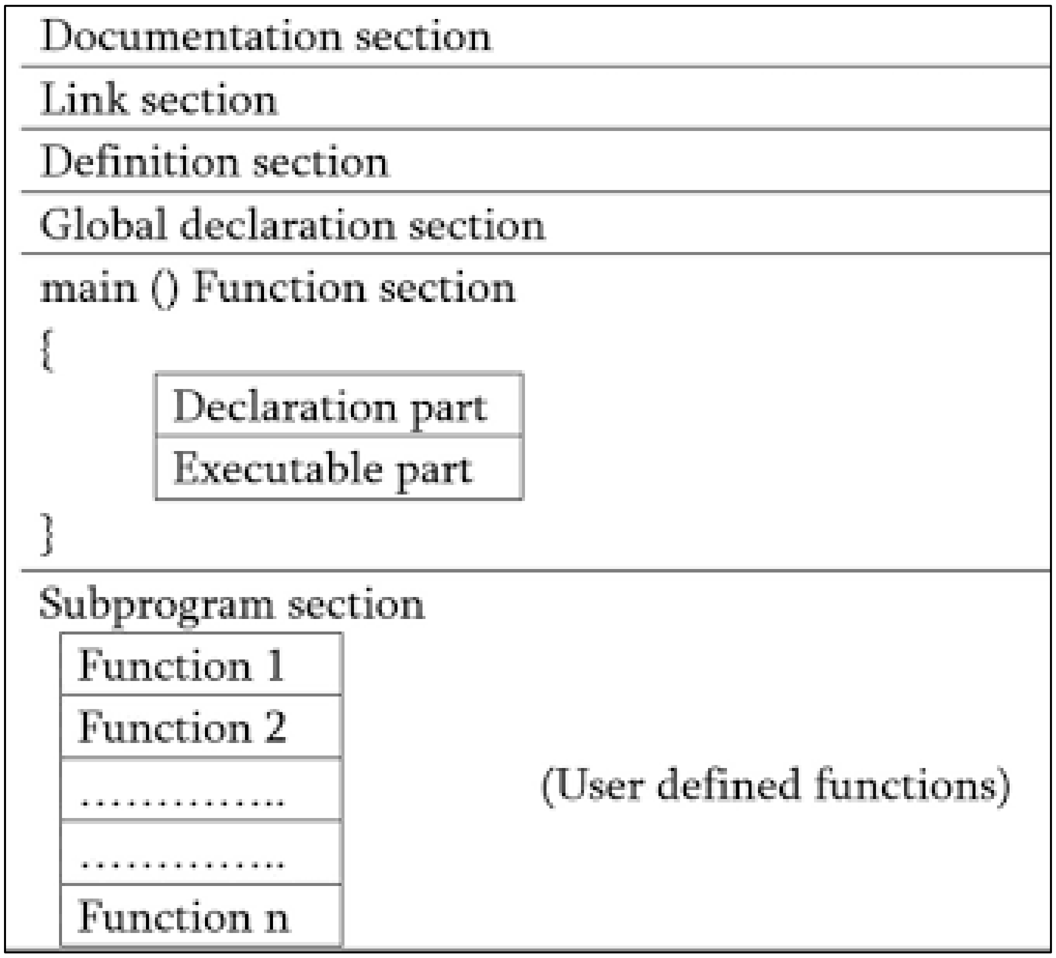
```
Hello World
```

Compiling a C Program

The compilation is the process of converting the source code of the C language into machine code. As C is a mid-level language, it needs a compiler to convert it into an executable code so that the program can be run on our machine.
The C program goes through the following phases during compilation:



Structure of C Program



The basic structure of a C program typically consists of the following sections:

- **Documentation Section:**

This section is used for comments, providing information about the program, such as the author, date, and a brief description of its purpose. Comments are ignored by the compiler but are crucial for code readability and maintenance.

- **Link Section (Preprocessor Directives):**

This section includes preprocessor directives, primarily #include statements. These directives instruct the C preprocessor to include header files, which contain declarations for functions and macros from system libraries (like stdio.h for standard input/output) or user-defined libraries.

- **Definition Section:**

This section is used to define symbolic constants using the #define directive. These constants are replaced by their defined values during preprocessing.

- **Global Declaration Section:**

Here, global variables and function prototypes (declarations of functions that will be defined later in the program) are declared. Global variables are accessible throughout the entire program.

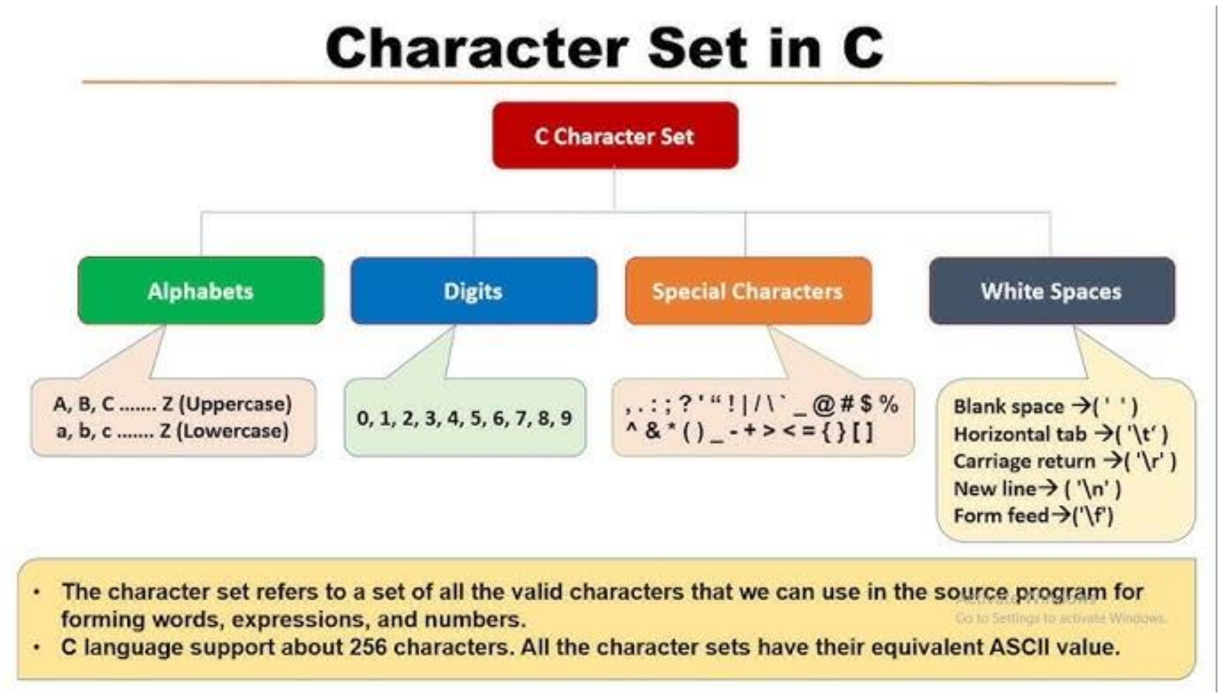
- **main() Function:**

This is the heart of every C program, as execution always begins here. The main() function contains the primary logic of the program and can call other functions. It typically returns an integer value (0 for successful execution).

- **Subprogram Section (User-Defined Functions):**

This section contains the definitions of user-defined functions that are called from the main() function or other user-defined functions. These functions encapsulate specific tasks, promoting modularity and reusability.

Not all sections are mandatory in every C program, but understanding this structure is fundamental for writing well-organized and maintainable C code.

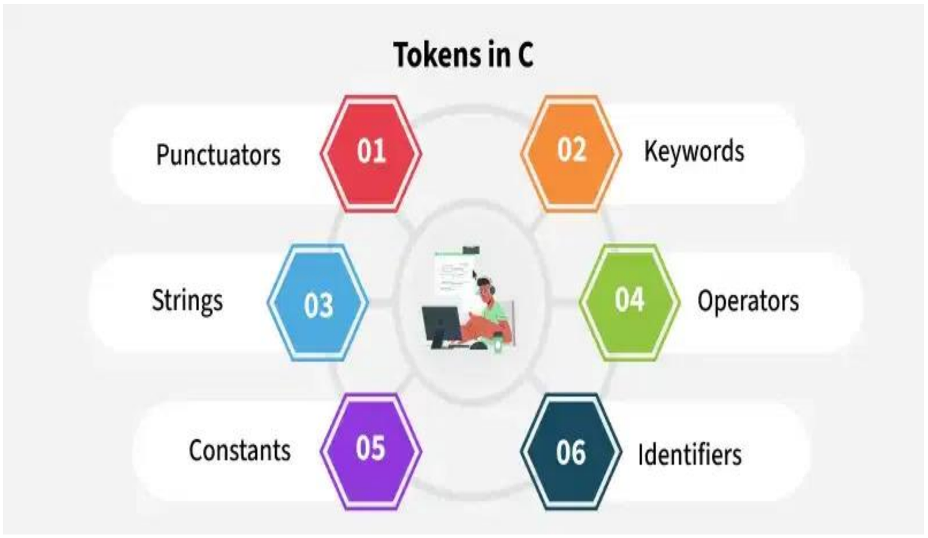


- The character set refers to a set of all the valid characters that we can use in the source program for forming words, expressions, and numbers.
- C language support about 256 characters. All the character sets have their equivalent ASCII value.

Tokens in C

In C programming, tokens are the smallest units in a program that have meaningful representations. Tokens are the building blocks of a C program, and they are recognized by the C compiler to form valid expressions and statements. Tokens can be classified into various categories, each with specific roles in the program.

Types of Tokens in C



1. Punctuators

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose. Some of these are listed below:

- Brackets[]: Opening and closing brackets are used as array element references. These indicate single and multidimensional subscripts.
- Parentheses(): These special symbols are used to indicate function calls and function parameters.
- Braces{}: These opening and ending curly braces mark the start and end of a block of code containing more than one executable statement.
- Comma (,): It is used to separate more than one statement like for separating parameters in function calls.
- Colon(:): It is an operator that essentially invokes something called an initialization list.
- Semicolon(;): It is known as a statement terminator. It indicates the end of one logical entity. That's why each individual statement must be ended with a semicolon.
- Asterisk (*): It is used to create a pointer variable and for the multiplication of variables.
- Assignment operator(=): It is used to assign values and for logical operation validation.
- Pre-processor (#): The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.
- Dot (.): Used to access members of a structure or union.
- Tilde(~): Bitwise One's Complement Operator.

2. Keywords

Keywords are reserved words that have predefined meanings in C. These cannot be used as identifiers (variable names, function names, etc.). Keywords define the structure and behavior of the program **C** language supports **32** keywords such as int, for, if, ... etc.

Note: *The number of keywords may change depending on the version of C you are using. For example, keywords present in ANSI C are 32 while in C11, it was increased to 44. Moreover, in the latest c23, it is increased to around 54.*

3. Strings

Strings are nothing but an array of characters ended with a null character ('\0'). This null character indicates the end of the string. Strings are always enclosed in double quotes. Whereas a character is enclosed in single quotes in C.

4. Operators

Operators are symbols that trigger an action when applied to C variables and other objects. The data items on which operators act are called operands.

5. Constants

Constants are fixed values used in a C program. These values do not change during the execution of the program. Constants can be integers, floating-point numbers, characters, or strings.

6. Identifiers

Identifiers are names given to variables, functions, arrays, and other user-defined items. They must begin with a letter (a-z, A-Z) or an underscore (_) and can be followed by letters, digits (0-9), and underscores.

C Variables

A variable in C is a named piece of memory which is used to store data and access it whenever required. It allows us to use the memory without having to memorize the exact memory address.

To create a variable in C, we have to specify a name and the type of data it is going to store in the syntax.

Syntax:

```
data_type name;
```

Example:

```
int num;
char letter;
float decimal;
```

In C, every variable must be declared before it is used. We can also declare multiple variables of same data type in a single statement by separating them using comma as shown:

Syntax:
data_type name1, name2, name3, ...;

Example:
int a, b, sum;

Rules for Naming Variables in C

- A variable name must only contain **letters**, **digits**, and **underscores**.
- It must **start with an alphabet** or an **underscore** only. It cannot start with a digit.
- **No white space** is allowed within the variable name.
- A variable name must **not** be any reserved word or **keyword**.
- The name must be unique in the program.
- Only first 31 characters are significant.

C Variable Initialization

Once the variable is declared, we can store useful values in it. The first value we store is called initial value and the process is called **Initialization**. It is done using assignment operator **(=)**.

int num;
num = 3;
Or
int num = 3;

Accessing Variables

The data stored inside a C variable can be easily accessed by using the variable's name.

```
#include <stdio.h>
int main()
{
   // Create integer variable
   int num = 3;
   // Access the value stored in variable
   printf("%d", num);
   return 0;
}
```

Output:
3

Changing Stored Values

We can also update the value of a variable with a new value whenever needed by using the assignment operator =.
Example:

```
#include <stdio.h>
int main()
{
   // Create integer variable
```

```
    int n = 3;

    // Change the stored data
    n = 22;

    // Access the value stored in variable
    printf("%d", n);
    return 0;
}
```

Output:

```
22
```

Memory Allocation of C Variables

When a variable is **declared**, the compiler is told that the variable with the given name and type exists in the program. But no memory is allocated to it yet. Memory is allocated when the variable is **defined**.
Most programming languages like C generally declare and define a variable in the single step. For example, in the above part where we create a variable, variable is declared and defined in a single statement.
The size of memory assigned for variables depends on the type of variable. We can check the size of the variables using **sizeof operator.**

Example:

```
#include <stdio.h>
int main()
{
    int num = 22;
    printf("%d bytes", sizeof(num));
    return 0;
}
```

Output:

```
4 bytes
```

Variables are also stored in different parts of the memory based on their **storage classes.**

Scope of Variables in C

We have told that a variable can be accessed anywhere once it is declared, but it is partially true. A variable can be accessed using its name anywhere in a specific region of the program called its **scope**. It is the region of the program where the name assigned to the variable is valid.
A scope is generally the area inside the **{} curly braces.**
Example:

```
// num cannot be accessed here

int main() {

    // num cannot be accessed here
    {
        // Variable declaration
        int num;
    }
    // Cannot be accessed here either
    return 0;
}
```

Constants in C

C also provides some variables whose value cannot be changed. These variables are called **constants** and are created simply by prefixing **const** keyword in variable declaration.

Syntax:

const data_type name = value;

Constants must be initialized at the time of declaration.

Difference between Constant and Variable in C

Constant	Variables
A constant is a variable or value that cannot be altered once defined.	A variable is a name associated with some memory location.
A constant is used to hold the fixed values which we can retrieve later but cannot change.	A variable is used to hold some value that can be changed according to the requirement.
The constants are generally stored in the text segment as they are read-only	The variables are stored inside a data segment, heap, or stack depending on the environment it is declared in.
We can only assign a value to the constant while defining it.	We can assign value to the variable anytime.
A constant can be defined by using **#define** or **const** keyword.	A variable can only be defined using the standard variable definition syntax.

Example: #define pi 3.14
const int pi = 3.14;
	Example: int var = 25;
	var = 10;

Difference between Keyword and Identifier in C

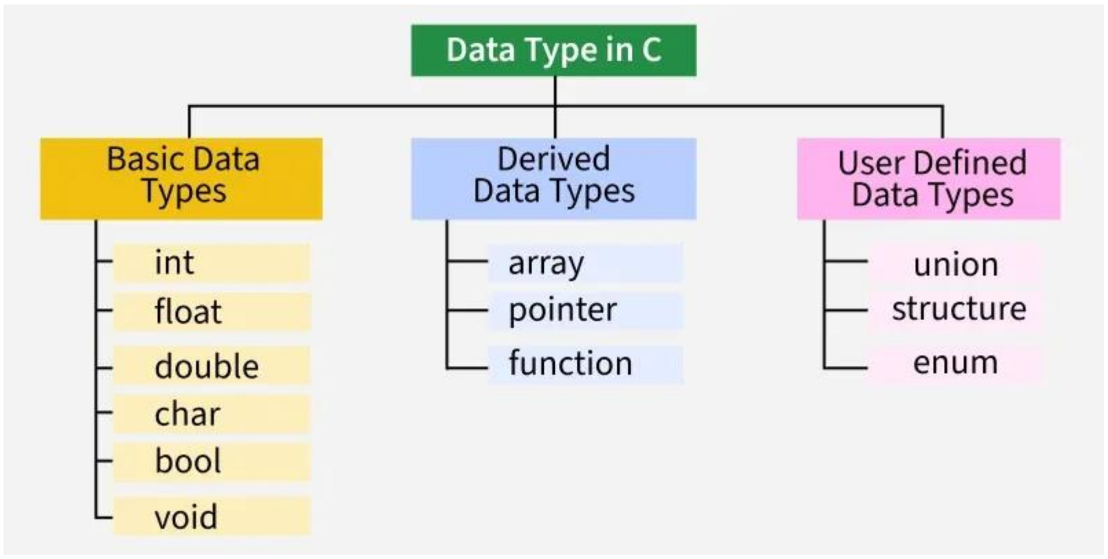
Parameters	Keywords	Identifiers
Definition	Keywords are predefined word that gets reserved for working program that have special meaning and cannot get used anywhere else.	Identifiers are the values used to define different programming items such as variables, integers, structures, unions and others and mostly have an alphabetic character.
Use	Specify the type/kind of entity.	Identify the name of a particular entity.
	It always starts with a lowercase letter.	First character can be a uppercase, lowercase letter or underscore.
Rules of Definition	A keyword should be in lower case and can only contains alphabetical characters.	An identifier can be in upper case or lower case and can consist of alphabetical characters, digits and underscores.
Purpose	They help to identify a specific property that exists within a computer language.	They help to locate the name of the entity that gets defined along with a keyword.
Examples	int, char, if, while, do etc.	Test, count1, high_speed, etc.

Data Types in C

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc.

Example:
int number;

C is a statically type language where each variable's type must be specified at the declaration and once specified, it cannot be changed.



Integer Data Type

The integer datatype in C is used to store the integer numbers (any number including positive, negative and zero without decimal part). Octal values, hexadecimal values, and decimal values can also be stored in int data type in C.

- Range: -2,147,483,648 to 2,147,483,647
- Size: 4 bytes
- Format Specifier: %d

Format specifiers are the symbols that are used for printing and scanning values of given data types.

Example:

We use **int keyword** to declare the integer variable:

int val**;**

We can store the integer values (literals) in this variable.

```
#include <stdio.h>
int main()
{
   int var = 22;
   printf("var = %d", var);
   return 0;
}
```

Output:

```
var = 22
```

The integer data type can also be used as:

1. **unsigned int:** It can store the data values from zero to positive numbers, but it can’t store negative values
2. **short int:** It is lesser in size than the int by 2 bytes so can only store values from -32,768 to 32,767.
3. **long int:** Larger version of the int datatype so can store values greater than int.
4. **unsigned short int:** Similar in relationship with short int as unsigned int with int.

The size of an integer data type is compiler dependent. We can use <u>sizeof operator</u> to check the actual size of any data type.

Character Data Type

Character data type allows its variable to store only a single character. The size of the character is 1 byte. It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

- Range: (-128 to 127) or (0 to 255)
- Size: 1 byte
- Format Specifier: %c

```
#include <stdio.h>
int main()
{
   char ch = ‘A’;
   printf(“ch = %c”, ch);
   return 0;
}
```

Output:

```
ch = A
```

Float Data Type

In C programming, float data type is used to store single precision floating-point values. These values are decimal and exponential numbers.

- **Range:** 1.2E-38 to 3.4E+38
- **Size:** 4 bytes
- **Format Specifier:** %f

```
#include <stdio.h>
int main()
{
   float val = 12.45;
   printf(“val = %f”, val);
   return 0;
}
```

Output:

```
val = 12.450000
```

Double Data Type

The double data type in C is used to store decimal numbers (numbers with floating point values) with double precision. It can easily accommodate about 16 to 17 digits after or before a decimal point.

- **Range:** 1.7E-308 to 1.7E+308
- **Size:** 8 bytes
- **Format Specifier:** %lf

```
#include <stdio.h>
int main()
 {
    double val = 1.4521;
    printf("val = %lf", val);
    return 0;
}
```

Output:

```
val = 1.452100
```

Void Data Type

The void data type in C is used to indicate the absence of a value. Variables of void data type are not allowed. It can only be used for **pointers** and **function return type and parameters**.

Example:

```
void fun(int a, int b){
    // function body
}
```

where function **fun** is a void type of function means it doesn't return any value.

Size of Data Types in C

The size of the data types in C is dependent on the size of the architecture, so we cannot define the universal size of the data types. For that, the C language provides the **sizeof()** operator to check the size of the data types.

```
#include <stdio.h>
int main()
{
    printf("The size of int: %d\n", sizeof(int));
    printf("The size of char: %d\n", sizeof(char));
    printf("The size of float: %d\n", sizeof(float));
    printf("The size of double: %d", sizeof(double));
    return 0;
}
```

Output:

```
The size of int: 4
The size of char: 1
The size of float: 4
The size of double: 8
```

Different data types also have different ranges up to which can vary from compiler to compiler. List of ranges along with the memory requirement and format specifiers on the 64-bit GCC compiler.

Data Type	Size (bytes)	Range	Format Specifier
short int	**2**	**-32,768 to 32,767**	**%hd**
unsigned short int	**2**	**0 to 65,535**	**%hu**
unsigned int	**4**	**0 to 4,294,967,295**	**%u**
int	**4**	**-2,147,483,648 to 2,147,483,647**	**%d**
long int	**4**	**-2,147,483,648 to 2,147,483,647**	**%ld**
unsigned long int	**4**	**0 to 4,294,967,295**	**%lu**
long long int	**8**	**-(2^63) to (2^63)-1**	**%lld**
unsigned long long int	**8**	**0 to 18,446,744,073,709,551,615**	**%llu**
signed char	**1**	**-128 to 127**	**%c**
unsigned char	**1**	**0 to 255**	**%c**
float	**4**	**1.2E-38 to 3.4E+38**	**%f**
double	**8**	**1.7E-308 to 1.7E+308**	**%lf**

Data Type	**Size (bytes)**	**Range**	**Format Specifier**
long double	**16**	**3.4E-4932 to 1.1E+4932**	**%Lf**

Note: The long, short, signed and unsigned are datatype modifier that can be used with some primitive data types to change the size or length of the datatype.

Assignment Statement

In C programming, an assignment statement is used to assign (store) a value into a variable.

General Syntax

variable = expression;

- variable → a valid C variable (left-hand side must always be a variable, not a constant or expression).
- = → the assignment operator.
- expression → can be a constant, another variable, or a valid arithmetic/logic expression.

Examples

```
int x;
x = 10;            // assigning constant
x = x + 5;         // assigning result of expression
float y = 3.14;    // assignment with declaration
char ch = 'A';     // assigning character constant
```

Types of Assignment Statements

Simple Assignment

```
a = 20;
b = a;
```

Multiple Assignment

```
x = y = z = 100;   // assigns 100 to x, y, and z
```

Compound Assignment (using shorthand operators)

```
a += 5;   // same as a = a + 5
b -= 2;   // same as b = b - 2
c *= 3;   // same as c = c * 3
d /= 4;   // same as d = d / 4
e %= 2;   // same as e = e % 2
```

Rules for Assignment Statement

- The left-hand side (LHS) must be a single variable (not a constant or expression).
  ✅ x = 10;
  ❌ 10 = x; (invalid)
  ❌ x + y = 5; (invalid)
- The right-hand side (RHS) can be a constant, variable, or expression.
  ✅ sum = a + b;
- Assignment is performed right to left (RHS is evaluated first, then stored in LHS).

Example Program:

```
#include <stdio.h>
int main()
{
    int a, b, c;
    a = 5;        // simple assignment
    b = a + 10;   // expression assignment
    c = b;        // variable assignment
    c += 2;        // compound assignment

    printf("a = %d, b = %d, c = %d", a, b, c);
    return 0;
}
```

Output:

```
a = 5, b = 15, c = 17
```

Constants in C

In C programming, const is a keyword used to declare a variable as constant, meaning its value cannot be changed after it is initialized. It is mainly used to protect variables from being accidentally modified, making the program safer and easier to understand. These constants can be of various types, such as integer, floating-point, string, or character constants.

Types of Constants in C

Type of Constants	Data type	Example of Data type
Integer Constants	int	23, 738, -1278, etc.
	unsigned int	2000u, 5000U, etc.
	long int, long long int	325647 l, 1245473940L
Floating Point Constants / Real Constants	double	500.987654321
	float	20.987654F
Octal constant	int	Example: 013 /*starts with 0 */
Hexadecimal constant	int	Example: 0x90 /*starts with 0x*/
character constants	char	Example: ‘X’, ‘Y’, ‘Z’
string constants	char	Example: “PQRS”, “ABCD”

More About Types of Constants in C

Integer Constants

It can be an octal integer or a decimal integer or even a hexadecimal integer. We specify a decimal integer value as a direct integer value, while we prefix the octal integer values with ‘0’. We also prefix the hexadecimal integer values with ‘0x’.

The integer constant used in a program can also be of an unsigned type or a long type. We suffix the unsigned constant value with ‘u’ and we suffix the long integer constant value with ‘l’. Also, we suffix the unsigned long integer constant value using ‘ul’.
Examples,
55 —> Decimal Integer Constant
0x5B —> Hexadecimal Integer Constant
023 —> Octal Integer Constant
68ul —> Unsigned Long Integer Constant
50l —> Long Integer Constant
30u —> Unsigned Integer Constant

Floating Point Constants / Real Constants

This type of constant must contain both the parts- decimal as well as integers. Sometimes, the floating-point constant may also contain the exponential part. In such a case when the floating-point constant gets represented in an exponential form, its value must be suffixed using ‘E’ or ‘e’.
Example:
We represent the floating-point value 3.14 as 314E-2 in its exponent form.

Character Constants

The character constants are symbols that are enclosed in one single quotation. The maximum length of a character quotation is of one character only.
Example:
‘B’
‘5’
‘+’
Some predefined character constants exist in the C programming language, known as escape sequences. Each escape sequence consists of a special functionality of its own, and each of these sequences gets prefixed with a ‘\’ symbol. We use these escape sequences in output functions known as ‘printf()’.

String Constants

The string constants are a collection of various special symbols, digits, characters, and escape sequences that get enclosed in double quotations.
The definition of a string constant occurs in a single line:
“This is Cookie”
We can define this with the use of constant multiple lines as well:
” This\
is\
Cookie”
The definition of the same string constant can also occur using white spaces:
“This” “is” “Cookie”
All the three mentioned above define the very same string constant.

Rules of Constructing Constants in C

Here is how we construct these constants in a given program:

Integer Constants

- It must have at least one digit.
- There must be no decimal point.
- It does not allow any blanks or commas.
- An integer constant can be both negative or positive.
- We assume an integer constant to be positive if there is no sign in front of that constant.
- The allowable range for this type of constant is from -32768 to 32767.

Real Constants

Real constants can be written in **two forms**:

Fractional (Decimal) form

- This type of constant must consist of at least one digit before and after the decimal point.
- This type of constant can be both negative or positive.
- It does not allow any blanks or commas within.
- We assume constant to be positive if there is no sign in front of that constant.

Exponential (Scientific) form
Represented in the form:

mantissa e exponent
or
mantissa E exponent

- The **mantissa** is a real number (with or without a decimal point).
- The **exponent** must be an integer (positive or negative).
- No spaces are allowed between mantissa, e/E, and exponent.
  - ✅ Example: 3.14e2 (means $3.14 \times 10^2 = 314$)
  - ✅ Example: -0.45E-3 (means $-0.45 \times 10^{-3} = -0.00045$)

String and Character Constants

- This type of constant can be a single digit, a single alphabet, or even a single special symbol that stays enclosed within single quotes.
- The string constants get enclosed within double quotes.
- The maximum length of this type of constant is a single character.

Backslash Character Constants

- These are some types of characters that have a special type of meaning in the C language.
- These types of constants must be preceded by a backslash symbol so that the program can use the special function in them.

- Here is a list of all the special characters used in the C language and their purpose:

Meaning of Character	Backslash Character
Backspace	\b
New line	\n
Form feed	\f
Horizontal tab	\t
Carriage return	\r
Single quote	\’
Double quote	\”
Vertical tab	\v
Backslash	\\
Question mark	\?
Alert or bell	\a
Hexadecimal constant (Here, N – hexadecimal constant)	\XN
Octal constant (Here, N is an octal constant)	\N

Syntax:
const *data_type var_name = value;*

```
#include <stdio.h>
int main()
{
        // Defining constant variable
        const int a = 10;
        printf("%d", a);
        return 0;
}
```
Output:

```
10
```

Properties of Constant

1. Initialization with Declaration

We can only initialize the constant variable in C at the time of its declaration. If we do not initialize it at the time of declaration, it will store the garbage value that was previously stored in the same memory.

```
#include <stdio.h>
int main()
{
        // Not initializing a constant variable
        const int a;

        // printing value
        printf("%d", a);
  return 0;
}
```
Output:

```
0
```

2. Immutability

The constant variables in c are immutable after its definition, i.e., they can be initialized only once in the whole program. After that, we cannot modify the value stored inside that variable.

```
#include <stdio.h>
int main()
{
   // Declaring a constant variable
   const int a;
     // Initializing constant variable var after declaration
   a = 20;
   printf("%d", a);
   return 0;
}
```
Output:

```
In function 'main':
10:9: error: assignment of read-only variable 'a'
10 |     a = 20;
```

| ^

Constants Using #define

In C, the **#define directive** can also be used to define symbolic constants that do not require a data type. They are called **macros** and are replaced by their values at compile time.

Syntax:
#define *CONSTANT_NAME value*

Example:

```
#include <stdio.h>
#define PI 3.14
int main()
{
   printf("%.2f", PI);
   return 0;
}
```

Output:

```
3.14
```

const vs #define

Constants using const	Constants using #define
They are the variables that are immutable.	They are the macros that are replaced by their value.
They are handled by the compiler.	They are handled by the preprocessor.
Syntax: **const *type name* = *value*;**	Syntax: **#define *name value***

Formatted and Unformatted Input/Output functions in C

In C language, the Input/Output (I/O) functions are part of the standard library, and these functions are used for interacting with the user or other systems, to perform operations such as reading input and printing output.

These functions provide ways to read data from files and other input devices or write data to files or other output devices.

These input/output functions can be classified into two categories on the basis of how they handle the input/output data in terms of formatting or structure:

- **Formatted I/O Functions.**
- **Unformatted I/O Functions.**

Formatted I/O Functions

Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

Why are they called formatted I/O?
These functions are called formatted I/O functions because we can use format specifiers in these functions and hence, we can format these functions according to our needs.

List of some common format specifiers-

Format Specifier	Type	Description
%d	int/signed int	used for I/O signed integer value
%c	char	Used for I/O character value
%f	float	Used for I/O decimal floating-point value
%s	string	Used for I/O string/group of characters
%ld	long int	Used for I/O long signed integer value

printf()

printf() function is used in a C program to display any value like float, integer, character, string, etc. on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).

Syntax:

//to print variables

printf("Format Specifier", var1, var2, ..., varn);

//to print a string

printf("Enter the text which you want to display");

```
#include <stdio.h>
int main()
{
   int a =20;
   printf("%d", a);
   printf("\nThis is a string");
   return 0;
}
```

Output:

```
20
This is a string
```

scanf()

scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This is a pre-defined function declared in stdio.h(header file). In scanf() function we use &(address-of operator) which is used to store the variable value on the memory location of that variable.

Syntax:

scanf("Format Specifier", &var1, &var2, ..., &varn);

```
#include <stdio.h>
int main()
{
   int num1;
   printf("Enter a integer number: \n");
   scanf("%d", &num1);
   printf("You have entered %d", num1);
   return 0;
}
```

Output:

```
Enter a integer number:
```

```
20
You have entered 20
```

sprintf()

sprintf stands for **"string print"**. This function is similar to printf() function but this function prints the string into a character array instead of printing it on the console screen.

Syntax:

sprintf(array_name, "format specifier", variable_name);

```
#include <stdio.h>
int main()
{
   char str[50];
   int a = 2, b = 8, c, d;
   sprintf(str, "a = %d and b = %d", a, b);
   sscanf(str, "a = %d and b = %d", &c, &d);
   printf("c = %d and d = %d", c, d);
   return 0;
}
```

Output:

```
c = 2 and d = 8
```

Unformatted Input/Output Functions

Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype. These functions are used to read single input from the user at the console and it allows to display the value at the console.

Why they are called unformatted I/O?

These functions are called unformatted I/O functions because we cannot use format specifiers in these functions and hence, cannot format these functions according to our needs.

The unformatted I/O functions in C are discussed below:

getch()

getch() function reads a single character from the keyboard by the user but doesn't display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h(header file). getch() is also used for hold the screen.

Syntax:

getch();

```
#include <conio.h>
#include <stdio.h>
int main()
{
    char ch;
    printf("Enter any character: ");
    // Reads a character but not displays
    ch =getch();
    printf("\nEntered character is: %c", ch);
    return 0;
}
```

Output:

```
Enter any character:
Entered character is : s
```

getche()

getche() function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key. This function is declared in conio.h(header file).

Syntax:

getche();

```
#include <conio.h>
#include <stdio.h>
int main()
{
    char ch;
    printf("Enter any character: ");
    // Reads a character but not displays
    ch =getche();
    printf("\nEntered character is: %c", ch);
    return 0;
}
```

Output:

```
Enter any character: s
Entered character is: s
```

getchar()

The getchar() function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed. This function is declared in stdio.h(header file).

Syntax:

getchar();

```
#include <stdio.h>
int main()
{
   char ch;
   printf("Enter the character: ");
   ch = getchar();
   printf("Entered character is %c", ch);
   return 0;
}
```

Output:

```
Enter the character: r
Entered character is r
```

putchar()

The putchar() function is used to display a single character at a time by passing that character directly to it or by passing a variable that has already stored a character. This function is declared in stdio.h(header file).

Syntax:

putchar(variable_name);

```
#include <conio.h>
#include <stdio.h>
int main()
{
   char ch;
   printf("Enter any character: ");
   ch = getchar();
   putchar(ch);
   return 0;
}
```

Output:

```
Enter any character: A
A
```

gets()

gets() function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array. This function allows us to write space-separated texts or strings. This function is declared in stdio.h(header file).

Syntax:

//Declare a char type variable of any length

char str[length of string in number];

gets(str);

```
#include <conio.h>
#include <stdio.h>
int main()
{
   char name[50];
   printf("Please enter some texts: ");
   // Reading a line of character or a string
   gets(name);
   // Displaying this line of character or a string
   printf("You have entered: %s", name);
   return 0;
}
```

Output:

```
Please enter some texts: Welcome to the world of C
You have entered: Welcome to the world of C
```

puts()

In C programming puts() function is used to display a group of characters or strings which is already stored in a character array. This function is declared in stdio.h(header file).

Syntax:

puts(identifier_name);

```
#include <stdio.h>
int main()
{
   char name[50];
   printf("Enter your text: ");
   // Reads string from user
   gets(name);
   printf("Your text is: ");
   // Displays string
   puts(name);
   return 0;
}
```

Output:

```
Enter your text: Programming in C
Your text is: Programming in C
```

putch()

putch() function is used to display a single character which is given by the user and that character prints at the current cursor location. This function is declared in conio.h(header file).

Syntax:

putch(variable_name);

```
#include <stdio.h>
#include <conio.h>
int main()
{
   char ch;
   printf("Enter any character: ");
   // Reads a character from the keyboard
   ch = getch();
   printf("Entered character is: ");
   // Displays that character on the console
   putch(ch);
   return 0;
}
```

Output:

```
Enter any character: Entered character is: a
```

Formatted I/O vs Unformatted I/O

Formatted I/O functions	**Unformatted I/O functions**
These functions allow us to take input or display output in the user's desired format.	These functions do not allow to take input or display output in user desired format.
These functions support format specifiers.	These functions do not support format specifiers.
These are used for storing data more user friendly	These functions are not more user-friendly.
Here, we can use all data types.	Here, we can use only character and string data types.
printf(), scanf, sprintf() and sscanf() are examples of these functions.	getch(), getche(), gets() and puts(), are some examples of these functions.