Data Type 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.
- Each data type requires different amounts of memory and has some specific operations which can be performed over it.
- The data type is a collection of data with values having fixed values, meaning as well as its characteristics.

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

Type	Size (bytes)	Format Specifier
`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`

The data types in C can be classified as follows:

Types	Description
Primary Data Types	Primary data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.
User Defined Data Types	The user-defined data types are defined by the user himself.
Derived Types	The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.

Primary Data Types in C

- Primary data types are also known as the fundamental data types because they are pre-defined or they already exist in the C language.

- All the other types of data types (derived and user-defined data types) are derived from these data types. Primary data types in C are of 4 types: int, char, float, and double.
- Here are the five primitive or primary data types that one can find in C programming language:
  - **Integer –** We use these for storing various whole numbers, such as 5, 8, 67, 2390, etc.
  - **Character –** It refers to all ASCII character sets as well as the single alphabets, such as 'x', 'Y', etc.
  - **Double –** These include all large types of numeric values that do not come under either floating-point data type or integer data type.
  - **Floating-point –** These refer to all the real number values or decimal points, such as 40.1, 820.673, 5.9, etc.

Various keywords are used in a program for specifying the data types mentioned above. Here are the keywords that we use:

Keyword Used	**Data Type**
int	Integer
float	Floating-point
char	Character
double	Double

The size of every data type gets defined in bytes/ bits. Also, these data types can hold a very wide range of values.

User Defined Data Type in C

- The UDT (User-Defined Data Type) is a typical data type that we can derive out of any existing data type in a program.
- We can utilise them for extending those built-in types that are already available in a program, and then you can create various customized data types of your own.

- The data types originally present in a program may not offer a wide variety of functions.
- But despite the various basic as well as derived data types present in the C language, there is a special feature using which we can define custom data types of our own, on the basis of our needs.

Types of User-Defined Data Types in C

The C language allows a feature known as the type definition. This basically allows a programmer to provide a definition to an identifier that will represent a data type which already exists in a program. The C program consists of the following types of UDT:

- Structures
- Union
- Typedef
- enum

Derived Data Type in C

The derived data types are basically derived out of the fundamental data types. A derived data type won't typically create a new data type – but would add various new functionalities to the existing ones instead.

Types of Derived Data Types in C

The C language supports a few derived data types. These are:

- **Arrays –** The *array* basically refers to a sequence (ordered sequence) of a finite number of data items from the same data type sharing one common name.
- **Function –** A *Function in C* refers to a self-contained block of single or multiple statements. It has its own specified name.
- **Pointers –** The *Pointers in C* language refer to some special form of variables that one can use for holding other variables' addresses.
- **Unions –** The unions are very similar to the structures. But here, the memory that we allocate to the largest data type gets reused for all the other types present in the group.

- **Structures –** A collection of various different types of data type items that get stored in a contagious type of memory allocation is known as *structure in C.*

*Note – In some of the situations, we can also call the structures and unions to be UDT (User-defined Data Types).

Data Type Modifiers in C Programming Language

- There are basically four types of modifiers for all data types used in C language.
- We use these along with all the basic data types for categorising them further.
- For instance, if we say that there is a chocolate bar on the table, the person we are speaking to will know that a chocolate bar is present on the table.
- But if we get more specific and say that there is a dark chocolate bar on the table or a milk chocolate bar on the table, it will become much clearer and more specific, to the person who is listening.
- In a very similar manner, the modifiers in C language help in making the primary or primitive data types much more specific.

Here are a few modifiers:

★ short
★ long
★ unsigned
★ Signed

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.

Example

```
// C Program to print size of
// different data type in C
#include <stdio.h>

int main()
{
int size_of_int = sizeof(int);
int size_of_char = sizeof(char);
int size_of_float = sizeof(float);
int size_of_double = sizeof(double);

printf("The size of int data type : %d\n", size_of_int);
printf("The size of char data type : %d\n",
size_of_char);
printf("The size of float data type : %d\n",
size_of_float);
printf("The size of double data type : %d",
size_of_double);

return 0;
}
```

Output:

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