C - DATA TYPES

In the C programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows:

S.N.	Types and Description
1	**Basic Types:** They are arithmetic types and consists of the two types: a integer types and b floating-point types.
2	**Enumerated types:** They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
3	**The type void:** The type specifier *void* indicates that no value is available.
4	**Derived types:** They include a Pointer types, b Array types, c Structure types, d Union types and e Function types.

The array types and structure types are referred to collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see basic types in the following section, whereas, other types will be covered in the upcoming chapters.

Integer Types

Following table gives you details about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions *sizeof*type yields the storage size of the object or type in bytes. Following is an example to get the size of int type on any machine:

```
#include <stdio.h>
#include <limits.h>

int main()
{
   printf("Storage size for int : %d \n", sizeof(int));

   return 0;
}
```

When you compile and execute the above program it produces the following result on Linux:

```
Storage size for int : 4
```

Floating-Point Types

Following table gives you details about standard floating-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. Following example will print storage space taken by a float type and its range values:

```
#include <stdio.h>
#include <float.h>

int main()
{
   printf("Storage size for float : %d \n", sizeof(float));
   printf("Minimum float positive value: %E\n", FLT_MIN );
   printf("Maximum float positive value: %E\n", FLT_MAX );
   printf("Precision value: %d\n", FLT_DIG );

   return 0;
}
```

When you compile and execute the above program, it produces the following result on Linux:

```
Storage size for float : 4
Minimum float positive value: 1.175494E-38
Maximum float positive value: 3.402823E+38
Precision value: 6
```

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

S.N.	Types and Description
1	**Function returns as void** There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void. For example **void exit *intstatus*;**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand*void*;**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example a memory allocation function **void *malloc*size*$_t$*size*;** returns a pointer to void which can be casted to any data type.

The void type may not be understood to you at this point, so let us proceed and we will cover these concepts in the upcoming chapters.