C - Data Types

Data types in C 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. In this chapter, we will learn about **data types in C**. A related concept is that of "variables", which refer to the addressable location in the memory of the processor. The data captured via different input devices is stored in the computer memory. A symbolic name can be assigned to the storage location called variable name.

C is a statically typed language. The name of the variable along with the type of data it intends to store must be explicitly declared before actually using it.

C is also a strongly typed language, which means that the automatic or implicit conversion of one data type to another is not allowed.

The types in C can be classified as follows –

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (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 assign 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 collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

Integer Data Types in C

The following table provides the details of standard integer types with their 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	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

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.

Example of Integer Data Types

Given below is an example to get the size of various type on a machine using different constant defined in limits.h header file –

</> Open Compiler

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

int main(int argc, char** argv) {

   printf("CHAR_BIT    :   %d\n", CHAR_BIT);
   printf("CHAR_MAX    :   %d\n", CHAR_MAX);
   printf("CHAR_MIN    :   %d\n", CHAR_MIN);
   printf("INT_MAX     :   %d\n", INT_MAX);
   printf("INT_MIN     :   %d\n", INT_MIN);
   printf("LONG_MAX    :   %ld\n", (long) LONG_MAX);
```

```
   printf("LONG_MIN    :   %ld\n", (long) LONG_MIN);
   printf("SCHAR_MAX   :   %d\n", SCHAR_MAX);
   printf("SCHAR_MIN   :   %d\n", SCHAR_MIN);
   printf("SHRT_MAX    :   %d\n", SHRT_MAX);
   printf("SHRT_MIN    :   %d\n", SHRT_MIN);
   printf("UCHAR_MAX   :   %d\n", UCHAR_MAX);
   printf("UINT_MAX    :   %u\n", (unsigned int) UINT_MAX);
   printf("ULONG_MAX   :   %lu\n", (unsigned long) ULONG_MAX);
   printf("USHRT_MAX   :   %d\n", (unsigned short) USHRT_MAX);

   return 0;
}
```

Output

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

```
CHAR_BIT   :  8
CHAR_MAX   :  127
CHAR_MIN   :  -128
INT_MAX    :  2147483647
INT_MIN    :  -2147483648
LONG_MAX   :  9223372036854775807
LONG_MIN   :  -9223372036854775808
SCHAR_MAX  :  127
SCHAR_MIN  :  -128
SHRT_MAX   :  32767
SHRT_MIN   :  -32768
UCHAR_MAX  :  255
UINT_MAX   :  4294967295
ULONG_MAX  :  18446744073709551615
USHRT_MAX  :  65535
```

Floating-Point Data Types in C

The following table provides the details of 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 the macros that allow you to use these values and other details about the binary representation of real numbers in your programs.

Example Floating-Point Data Types

The following example prints the storage space taken by a float type and its range values –

</> Open Compiler

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

int main(int argc, char** argv) {

   printf("Storage size for float : %zu \n", sizeof(float));
   printf("FLT_MAX     :   %g\n", (float) FLT_MAX);
   printf("FLT_MIN     :   %g\n", (float) FLT_MIN);
   printf("-FLT_MAX    :   %g\n", (float) -FLT_MAX);
   printf("-FLT_MIN    :   %g\n", (float) -FLT_MIN);
   printf("DBL_MAX     :   %g\n", (double) DBL_MAX);
   printf("DBL_MIN     :   %g\n", (double) DBL_MIN);
   printf("-DBL_MAX     :  %g\n", (double) -DBL_MAX);
   printf("Precision value: %d\n", FLT_DIG );

   return 0;
}
```

Output

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

```
Storage size for float : 4
FLT_MAX     :  3.40282e+38
FLT_MIN     :  1.17549e-38
-FLT_MAX    :  -3.40282e+38
-FLT_MIN    :  -1.17549e-38
DBL_MAX     :  1.79769e+308
DBL_MIN     :  2.22507e-308
-DBL_MAX    : -1.79769e+308
Precision value: 6
```

Note: "sizeof" returns "size_t". The type of unsigned integer of "size_t" can vary depending on platform. And, it may not be long unsigned int everywhere. In such cases, we use "%zu" for the format string instead of "%d".

Earlier versions of C did not have Boolean data type. C99 standardization of ANSI C introduced _bool type which treats zero value as false and non-zero as true.

User-defined Data Types in C

There are two user-defined data types struct and union, that can be defined by the user with the help of the combination of other basic data types.

Struct Data Type

One of the unique features of C language is to store values of different data types in one variable. The keywords **struct** and **union** are provided to derive a user-defined data type. For example,

```
struct student   {
   char name[20];
   int marks, age;
};
```

Union Data Type

A union is a special case of struct where the size of union variable is not the sum of sizes of individual elements, as in struct, but it corresponds to the largest size among individual elements. Hence, only one of elements can be used at a time. Look at following example:

```
union ab  {
   int a;
```

```
   float b;
};
```

We shall learn more about structure and union types in a later chapter.

The void Data Type in C

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

Sr.No	Types & Description
1	**Function returns as void** There are various functions in C that do not return any value or you can say they return **void**. A function with no return value has the return type as **void**. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept 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.

Arrays Data Type in C

An array is a collection of multiple values of same data type stored in consecutive memory locations. The size of array is mentioned in square brackets []. For example,

```
int marks[5];
```

Arrays can be initialized at the time of declaration. The values to be assigned are put in parentheses.

```
int marks[ ]={50,56,76,67,43};
```

C also supports multi-dimensional arrays. To learn more about arrays, refer to the chapter on Arrays in C.

Pointers Data Type in C

A pointer is a special variable that stores address or reference of another variable/object in the memory. The name of pointer variable is prefixed by asterisk (*). The type of the pointer variable and the variable/object to be pointed must be same.

```
int x;
int *y;
y = &x;
```

Here, "y" is a pointer variable that stores the address of variable "x" which is of "int" type.

Pointers are used for many different purposes. Text string manipulation and dynamic memory allocation are some of the processes where the use of pointers is mandatory. Later in this tutorial, you can find a detailed chapter on Pointers in C.