Size of Data Types in C

When programming in C, it is important to understand the size of data types. The Size of Data Types in C is helpful in understanding the amount of memory required to store the particular data type.

In C, the size of data types can vary depending on the implementation and the architecture being used. This article helps you understand and learn more about the Size of Data Types in C and their use while programming.

Data Types And Size of Data Types in C

In C, data types can be classified into three main categories: Primary, Derived and Floating Point Data Types. Primary data types in C Programming are the basic building blocks of data and are used to represent fundamental data objects. Derived data types are constructed from the primary data types. And the floating point data types represent real numbers. Read on to know more about the Data types and Size of Data Types in C.

Primary Data Types

C has four fundamental data types: char, int, float, and double. These data types are used to store basic values such as characters, integers, floating-point numbers, and double-precision floating-point numbers.

The following are the Primary Data Types in C:

- Character (char)
- Integer (int)
- Floating Point (float)
- Double Floating Point (double)
- Void (void)
- Boolean (_Bool)
- Wide Character (wchar_t)

Size Of Primary Data Types

The size of primary data types in C is platform-dependent, which means it may vary from one system to another. However, the standard size of primary data types in C is defined by the C99 standard.

Data Type	Size (in bytes)	Range	Precision
char	1	-128 to 127 (signed) or 0 to 255 (unsigned)	~7 decimal digits
unsigned char	1	0 to 255	~15 decimal digits
short	2	-32,768 to 32,767 (signed) or 0 to 65,535 (unsigned)	Varies
unsigned short	2	0 to 65,535	
int	4	-2,147,483,648 to 2,147,483,647 (signed)	
unsigned int	4	0 to 4,294,967,295	
long	4 or 8	-2,147,483,648 to 2,147,483,647 (signed) (32-bit systems) or -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (signed) (64-bit systems)	
unsigned long	4 or 8	0 to 4,294,967,295 (32-bit systems) or 0 to 18,446,744,073,709,551,615 (64-bit systems)	
long long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	
unsigned long long	8	0 to 18,446,744,073,709,551,615	

float	4	~7 decimal digits of precision	
double	8	~15 decimal digits of precision	
long double	Varies	Varies depending on the implementation	
bool	1	true (non-zero) or false (zero)	

Char Size

The char data type is used to represent character data. It is a 1-byte data type, which means it can hold a character from the ASCII character set.

Short Integer Size

The short int data type is used to represent small integers. It is a 2-byte data type, which means it can hold values between -32,768 to 32,767.

Long Integer Size

The long int data type is used to represent large integers. It is a 4-byte data type, which means it can hold values between -2,147,483,648 to 2,147,483,647.

Difference between the range for Unsigned and Signed Data Types

The range of signed and unsigned types is different because of the way negative numbers are represented. In signed types, negative numbers are represented using either the 1’s complement or 2’s complement. In unsigned types, there is no sign bit, which means all bits are used to represent positive values.

1’S Complement

In the 1’s complement representation, the sign bit is flipped to represent a negative number. For example, the 1’s complement representation of -5 would be 1111 1010 (in binary).

2’S Complement

In the 2’s complement representation, the sign bit is flipped and 1 is added to represent a negative number. For example, the 2’s complement representation of -5 would be 1111 1011 (in binary).

Floating Point Data Types

The floating-point data types are used to represent real numbers with fractional parts. They are designed to provide a compromise between range and precision, allowing for the representation of a wide range of values, including both very large and very small numbers, with a certain degree of decimal accuracy.

Size Of Floating Point Data Types

The size of floating point data types in C can vary across programming languages and platforms, but they generally fall into a few standard sizes.

1. Single-precision floating-point: Single-precision floating-point numbers, often referred to as "floats," typically occupy 32 bits (4 bytes) of memory. They provide a good balance between precision and storage efficiency, offering around 7 decimal digits of accuracy.
2. Double-precision floating-point: Double-precision floating-point numbers, commonly known as "doubles," use 64 bits (8 bytes) of memory. Doubles offer higher precision compared to floats, providing roughly 15 decimal digits of accuracy. They are commonly used when higher precision is required, such as in scientific and financial calculations.
3. Extended-precision floating-point: Some programming languages or libraries provide extended-precision floating-point types that offer even higher precision than doubles. These types can occupy 80 bits, 128 bits, or even more memory, depending on the implementation. They are generally used in specialized applications that demand extremely high precision, such as certain numerical computations or simulations.

Data Type	**Size (in bytes)**	**Approximate Range**	**Precision**
float	4	Approximately ±1.2 × 10^−38 to ±3.4 × 10^38	~7 decimal digits
double	8	Approximately ±2.3 × 10^−308 to ±1.7 × 10^308	~15 decimal digits
long double (varies)	Varies	Varies	Varies

The Floating Data Types in C are stored in a normalized or denormalized form.

Normalized Form

The normalized form represents the number in scientific notation using a mantissa and exponent. The mantissa signifies the important digits of a number, whereas the exponent indicates the placement of the decimal point.

De-Normalised Form

The denormalized form represents the number in by a mantissa only, and the exponent is zero. The denormalized form is used to represent numbers close to zero.

Derived Data Types

Derived data types, also known as user-defined data types, vary in size depending on the specific implementation and programming language. The size of a derived data type in C is determined by the underlying data elements and their respective sizes.

Size Of Derived Data Types

In general, the size of a derived data type in C is calculated by summing the sizes of its constituent data elements, accounting for any padding or alignment requirements imposed by the compiler.

For example, let's consider a simple struct in the C programming language:

```
struct Person {

    char name[50];

    int age;

    float height;

};
```

In this case, the size of the Person struct would typically be calculated as the sum of the sizes of its individual members:

```
sizeof(struct Person) = sizeof(char[50]) + sizeof(int) + sizeof(float)

                      = 50 + 4 + 4 (assuming 4-byte integers and floats)

                      = 58 bytes
```

It's important to note that the size of a derived data type in C can also be affected by factors such as memory alignment requirements, padding between members, and compiler-specific optimizations. These factors may cause the actual size of a derived data type to be larger than the sum of its individual members.

Understanding the Size of Data Types in C is crucial for efficient memory management and optimal performance in programming. The size of data types varies depending on the programming language, platform, and compiler being used. However, it's important to be aware that the actual Size of Data Types in C may vary on different systems.