Data Types in C++

All variables use data type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data they can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data type with which it is declared. Every data type requires a different amount of memory.

C++ supports a wide variety of data types and the programmer can select the data type appropriate to the needs of the application. Data types specify the size and types of values to be stored. However, storage representation and machine instructions to manipulate each data type differ from machine to machine, although C++ instructions are identical on all machines.

C++ supports the following data types:

- **Primary or Built-in or Fundamental data type**
- **Derived data types**
- **User-defined data types**

Data types in C++ are mainly divided into three types:

1. **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool, etc. Primitive data types available in C++ are:

- **Integer** - The integer data type is used to represent whole numbers in a computer program. It is typically used to store values that do not have a fractional component, such as the number of students in a class or the number of items in an order.

In C++, the integer data type is represented by the keyword '**int**'. Some more representation of Integer are **short, long , long long , unsigned short, unsigned int, unsigned long, unsigned long long** (unsigned means non negative number is to be stored).

- **Character** - The character data type is used to represent individual characters in a computer program. It is typically used to store values that represent letters, digits, and other symbols.

In C++, the character data type is represented by the keyword '**char**'. Some more representation of Character are **unsigned char, wchar_t, char8_t,char16_t, char_32,**

- **Boolean** - The Boolean data type is used to represent true/false values in a computer program. It is typically used to store values that can only be one of two options, such as yes/no, on/off, or true/false.

In C++, the Boolean data type is represented by the keyword '**bool**'.

- **Floating Point** - The floating point data type is used to represent real numbers in a computer program. It is typically used to store values that have a fractional component, such as 3.14 or 42.0.

In C++, the floating point data type is represented by the keyword '**float**'.

- **Double Floating Point** - Double floating point datatype, also known as double in C++, is a data type used to store decimal values with greater precision than a single floating point datatype. It uses twice as many bits to store a value and can represent a larger range of values than a single floating point.

In C++, the floating point data type is represented by the keyword '**double**'. "**long double"**

- **Valueless or Void** - The void data type in C++ is used to specify that a function or a pointer does not return a value. It can also be used as a placeholder for a data type when a function does not take any arguments,
- **Wide Character** - A wide character, also known as wchar_t in C++, is a data type used to represent a wide character, which is a character that can take up more than one byte of storage. It is commonly used to store characters from non-Latin scripts, such as Chinese or Arabic, which require more than one byte to represent a character.

2. **Derived Data Type**: The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

- **Function** - A function is a block of code that performs a specific task and may or may not return a value. In C++, functions can be defined using derived data types, which are data types that are derived from or based on the built-in data types.
- **Array** - An array is a collection of variables of the same data type stored in contiguous memory locations. Arrays in C++ are defined using square brackets and can have one or more dimensions.
- **Pointer** - A pointer is a variable that stores the memory address of another variable. Pointers in C++ are defined using the * operator and are used to manipulate data stored in memory directly.

- **Reference** - A reference in C++ is a variable that acts as an alias for another variable. It allows you to create a second name for a variable, which can be used to modify the original variable's value.

3. **Abstract or User-Defined Data Types**: These data types are defined by the user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:

- **Class** - A class in C++ is a user-defined data type that allows you to create objects with their own data members and member functions. It is a template for creating objects that define the attributes and behaviors of those objects.
- **Structure** - A structure in C++ is a composite data type that allows you to store multiple variables of different data types in a single unit. Structures are defined using the struct keyword and can contain variables, functions, and other data types.
- **Union** - A union in C++ is a derived data type that allows you to store multiple variables of different data types in the same memory location. Unlike a structure, which stores each variable in a separate memory location, a union stores all variables in the same location and allows only one variable to be accessed at a time.
- **Enumeration** - An enumeration in C++ is a user-defined data type that consists of a set of named constants called enumerators. It allows you to create a set of symbolic names for a set of integral values, which can make your code more readable and maintainable.

Enumerations are defined using the enum keyword and can be defined with or without a specified underlying type.

> 👉 **Typedef defined Datatype** - typedef is a keyword in C++ that allows you to create a new name for an existing data type. It can be used to give a data type a more descriptive or abbreviated name, which can make your code more readable and maintainable.