Data Types in C++

Last Updated : 10 Oct, 2025

Data types specify the type of data that a variable can store. Whenever a variable is defined in C++, the compiler allocates memory for that variable based on the data type with which it is declared. Please note that every may require a different amount of memory.

Below is an example of integer data type.

```
#include <iostream>
using namespace std;

int main() {

    // Creating a variable to store integer
    int var = 10;

    cout << var;
    return 0;
}
```

Output

```
10
```

Explanation: In the above code, we needed to store the value **10** in our program, so we created a variable **var**. But before **var**, we have used the keyword '**int**'**.** This keyword is used to define that the variable **var** will store data of type **integer**.
Let's see how to use some primitive data types in C++ program.

1. Character Data Type (char)

The **character data type** is used to store a single character. The keyword used to define a character is **char**. Its size is 1 byte, and it stores characters enclosed in single quotes (' '). It can generally store upto 256 characters according to their ASCII codes.

```
#include <iostream>
using namespace std;
```

```
int main()
{

    // Character variable
    char c = 'A';
    cout << c;

    return 0;
}
```

Output

```
A
```

2. Integer Data Type (int)

Integer data type denotes that the given variable can store the integer numbers. The keyword used to define integers is **int.** Its size is **4-bytes** (for 64-bit) systems and can store numbers for binary, octal, decimal and hexadecimal base systems in the range from **-2,147,483,648** to **2,147,483,647.**

```
#include <iostream>
using namespace std;

int main()
{

    // Creating an integer variable
    int x = 25;
    cout << x << endl;

    // Using hexadecimal base value
    x = 0x15;
    cout << x;

    return 0;
}
```

Output

```
25

21
```

To know more about different base values in C++, refer to the article - Literals in C++

3. Boolean Data Type (bool)

The **boolean data type** is used to store logical values: **true(1)** or **false(0)**. The keyword used to define a boolean variable is **bool**. Its size is 1 byte.

```
#include <iostream>
using namespace std;

int main()
{

    // Creating a boolean variable
    bool isTrue = true;

    cout << isTrue;
    return 0;
}
```

Output

```
1
```

4. Floating Point Data Type (float)

Floating-point data type is used to store numbers with decimal points. The keyword used to define floating-point numbers is **float**. Its size is 4 bytes (on 64-bit systems) and can store values in the range from **1.2e-38** to **3.4e+38.**

```
#include <iostream>
using namespace std;

int main()
{

    // Floating point variable with a decimal value
    float f = 36.5;
    cout << f;

    return 0;
}
```

Output

```
36.5
```

5. Double Data Type (double)

The **double data type** is used to store decimal numbers with higher precision. The keyword used to define double-precision floating-point numbers is **double**. Its size is 8 bytes (on 64-bit systems) and can store the values in the range from **1.7e-308** to **1.7e+308**

```
#include <iostream>
using namespace std;

int main()
{

    // double precision floating point variable
    double pi = 3.1415926535;
    cout << pi;

    return 0;
}
```

Output

```
3.14159
```

6. Void Data Type (void)

The **void data type** represents the absence of value. We cannot create a variable of void type. It is used for pointer and functions that do not return any value using the keyword **void**.

Type Safety in C++

C++ is a **strongly typed language**. It means that all variables' data type should be specified at the declaration, and it does not change throughout the program. Moreover, we can only assign the values that are of the same type as that of the variable.

If we try to assign **floating point** value to a boolean variable, it may result in data corruption, runtime errors, or undefined behaviour.

```
#include <iostream>
using namespace std;

int main()
{

    // Assigning float value to isTrue
    bool a = 10.248f;
```

```
    cout << a;

    return 0;
}
```

Output

```
1
```

As we see, the floating-point value is not stored in the bool variable **a.** It just stores 1. This type checking is not only done for fundamental types, but for all data types to ensure valid operations and no data corruptions.

Data Type Conversion

Type conversion refers to the process of changing one data type into another compatible one without losing its original meaning. It's an important concept for handling different data types in C++.

```
#include <iostream>
using namespace std;

int main()
{
    int n = 3;
    char c = 'C';

    // Convert char data type into integer
    cout << (int)c << endl;

    int sum = n + c;
    cout << sum;
    return 0;
}
```

Output

```
67

70
```

Size of Data Types in C++

Earlier, we mentioned that the size of the data types is according to the 64-bit systems. Does it mean that the size of C++ data types is different for different computers?

Actually, it is partially true. The size of C++ data types can vary across different systems, depending on the architecture of the computer (e.g., 32-bit vs. 64-bit systems) and the compiler being used. But if the architecture of the computer is same, then the size across different computers remains same.

We can find the size of the data type using **sizeof** operator. According to this type, the range of values that a variable of given data types can store are decided.

```
#include <iostream>
using namespace std;

int main()
{

    // Printing the size of each data type
    cout << "Size of int: " << sizeof(int) << " bytes" << endl;
    cout << "Size of char: " << sizeof(char) << " byte" << endl;
    cout << "Size of float: " << sizeof(float) << " bytes" << endl;
    cout << "Size of double: " << sizeof(double) << " bytes";

    return 0;
}
```

Output

```
Size of int: 4 bytes
Size of char: 1 byte
Size of float: 4 bytes
Size of double: 8 bytes
```

Data Type Modifiers

Data type modifiers are the keywords used to change or give extra meaning to already existing data types. It is added to primitive data types as a prefix to modify their size or range of data they can store. There are 4 type modifiers in C++: **short, long, signed** and **unsigned.**

For Example, defining an **int** with **long** type modifier will change its size to 8 bytes:

```
int var1; // 4 bytes
long int var2; // 8 bytes
```