References in C++

Last Updated : 26 Jul, 2025

In C++, a **reference** works as an alias for an existing variable, providing an alternative name for it and allowing you to work with the original data directly.

Example:

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

int main() {
    int x = 10;

    // ref is a reference to x.
    int& ref = x;

    // printing value using ref
    cout << ref << endl;

    // Changing the value and printing again
    ref = 22;
    cout << ref;

    return 0;
}
```

Output

```
10

22
```

Explanation: In this program, **ref** is a reference to the variable **x**, meaning **ref** is just another name for **x**. When the value of **ref** is modified, it directly changes the value of **x**, since both **ref** and **x** refer to the same memory location.
If you've worked with pointers before, references might feel similar, they both refer to the same memory. However, references provide a simpler, more readable way to alias a variable without using *.

To see how references differ from pointers, check out: Pointers vs References.

Syntax

& is used to create a reference to a variable as shown:

T ***&ref*** *= var;*

Here, **ref** refers to the variable named **var** which is of type **T.**

It is important to keep in mind the following things about reference.

- It is necessary to initialize the reference at the time of creation.
- A reference cannot refer to another variable once it is declared.

Applications

There are multiple applications for references in C++, a few of them are mentioned below:

1. Passing Arguments by Reference

References are commonly used in function arguments to allow modification of the original variable passed to the function. They are also more efficient for large data structures since no copies are made.

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

void modifyValue(int &x) {

    // Modifies the original variable
    x = 20;
}

int main() {
    int a = 10;

    // Pass a by reference
    modifyValue(a);

    cout << a;
    return 0;
}
```

Output

```
20
```

Explanation: In this example, the function **modifyValue()** modifies the value of a directly by using a reference to it. No copy of a is made, and the original variable is modified.

2. Returning Reference from Functions

In C++, functions can return references to variables. This is especially useful when you need to return large data structures or want to allow direct modifications to a variable inside a function.

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

int& getMax(int &a, int &b) {

    // Return the larger of the two numbers
    return (a > b) ? a : b;
}

int main() {
    int x = 10, y = 20;
    int &maxVal = getMax(x, y);

    // Modify the value of the larger number
    maxVal = 30;
    cout << "x = " << x << ", y = " << y;
    return 0;
}
```

Output

```
x = 10, y = 30
```

Explanation: Here, the function **getMax()** returns a reference to the larger of two integers. Since **maxVal** is a reference, changing **maxVal** also modifies the value of **y**.
Note: Keep in mind not to return a reference to local variables as they are destroyed once the function returns some value.

3. Modify Data in Range Based Loops

The range based for loops generally

```
#include <iostream>
```

```cpp
#include <vector>
using namespace std;

int main() {
    vector<int> vect{ 10, 20, 30, 40 };

    // We can modify elements if we
    // use reference
    for (int& x : vect) {
        x = x + 5;
    }

    // Printing elements
    for (int x : vect) {
        cout << x << " ";
    }
    return 0;
}
```

Output

```
15 25 35 45
```

Limitations of References

1. Once a reference is created, it cannot be later made to reference another object; it cannot be reset. This is often done with pointers.
2. References cannot be NULL. Pointers are often made NULL to indicate that they are not pointing to any valid thing.
3. A reference must be initialized when declared. There is no such restriction with pointers.

Due to the above limitations, references in C++ cannot be used for implementing data structures like Linked List, Tree, etc. In Java, references don't have the above restrictions and can be used to implement all data structures. References being more powerful in Java is the main reason Java doesn't need pointers.

Advantages of using References

1. **Safer**: Since references must be initialized, wild references like wild pointers are unlikely to exist. It is still possible to have references that don't refer to a valid location

2. **Easier to use:** References don't need a dereferencing operator to access the value. They can be used like normal variables. The '&' operator is needed only at the time of declaration. Also, members of an object reference can be accessed with the dot operator ('.'), unlike pointers where the arrow operator (->) is needed to access members.

Together with the above reasons, there are a few places like the copy constructor argument where a pointer cannot be used. Reference must be used to pass the argument in the copy constructor. Similarly, references must be used for overloading some operators like ++.