C++ Pass by Value vs Pass by Reference

In this presentation, we will explore pass by value and pass by reference, comparing their characteristics, benefits, and drawbacks, and using code examples to illustrate.

by Michael Hanna

The Basics: Introduction to Pass by Value and Pass by Reference

1

Pass by Value

A technique of passing arguments to a function by creating a copy of the original value to ensure independent change.

2

Pass by Reference

A technique of passing arguments to a function by passing a reference (memory address) of the original value to the function.

3

Key Differences

Pass by reference can modify the original value, while pass by value cannot. Pass by reference is generally faster and more memory-efficient.

Pass by Value Explained

Definition

Pass by value creates a copy of the actual parameter's value, which is then passed to the function. The value of the original parameter is not modified.

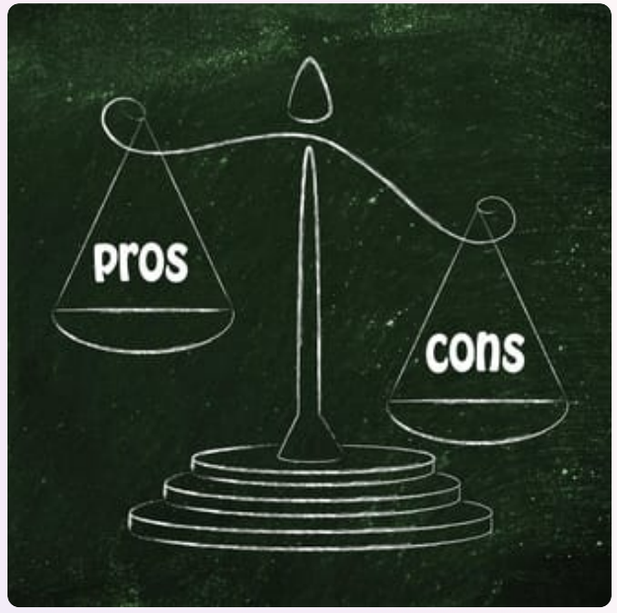
How it Works

The function creates its own variable, separate from the original, and assigns the value of the argument to it. The function modifies this new variable without changing the original.

Use Case

Pass by value is useful when you only need to read the value of the parameter and not modify it. It is also safer for multi-threaded environments.

Pros and Cons of Pass by Value Approach



Pros

- Simplicity: pass by value is easy to understand and use
- No side effects: changes made to arguments in the function don't impact the calling code
- Predictability: the value of the argument is fixed and cannot change

Cons

- Memory overhead: every argument passed by value creates an independent copy
- Performance impact: copying large objects can be slow
- Not suitable for large data structures or expensive copies

Pass by Reference Explained

Definition

Pass by reference allows the function to modify the value of the original parameter by directly accessing its memory address.

How it Works

The function declares a pointer parameter that points to the memory address of the original argument. Any changes to the pointer's value modify the original value directly.

Use Case

Pass by reference is useful when you need to update the original value of the parameter or when you want to avoid copying large data.

Pros and Cons of Pass by Reference Approach

1 Pros

Efficiency: Pass by reference avoids the overhead of copying values, reducing memory usage and improving performance.

2 Cons

Side Effects: Changes made to parameters through references persist beyond the scope of the function, requiring care to avoid unintended side effects.

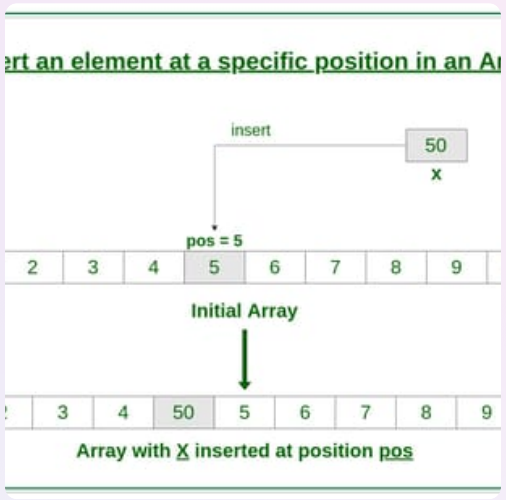
3 Additional Responsibility:

Handles references must manually manage memory, which can be tricky and error-prone.

C++ Pass by Value vs Pass by Reference: Examples to Compare

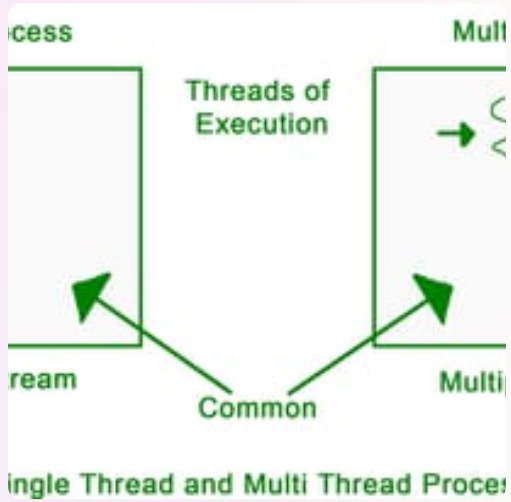
Function to Swap Two Values

We can either use pointers (passing by reference) or temporary variables (passing by value) to swap two values in a function.



Function to Find Maximum Element in an Array

Using pointers, we can modify the maximum value. Pass by value creates a copy of the array, which can be expensive.



Function for Multi-threading

Pass by reference is preferred over pass by value when it comes to multi-threaded environments where access to shared objects needs to be synchronized.

Reference, Pointer and Constant?

Passing by value is the default for primitive types (int, char, …). The non-primitive data (user-defined) types are pass by reference by default.

How about **pointers** and **constant** qualifiers?

```
struct Object {
  int i;
};
void sample(Object* o) { // 1
  o->i++;
}
void sample(Object const& o) { // 2
  cout << "Called #2\n";
}
void sample(Object & o) { // 3
  o.i++;
}
void sample1(Object o) { // 4
  o.i++;
}
```

```
int main() {
  Object obj = { 10 };
  Object const obj_c = { 10 };

  sample(&obj);  // calls #1
  sample(obj);   // calls #3
  sample(obj_c); // calls #2
  sample1(obj);  // calls #4
}
```