Shallow Copy vs Deep Copy

In Object-Oriented Programming (OOP), **shallow copy** and **deep copy** refer to two different ways of copying objects. They determine how the data members (particularly references or pointers) of the object are copied.

1. Shallow Copy

A **shallow copy** creates a new object but copies references to the original object's memory addresses rather than duplicating the objects themselves. This means that if the original object contains references (like pointers, arrays, or objects), the shallow copy will point to the same objects as the original. Changing a referenced object from either the original or the copy will affect both, because they share the same memory address.

Key Points:

- Only copies the top-level structure.
- Does not create copies of nested objects (if any).
- Both the original and copied objects share the same nested objects in memory.

Example:

cpp

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

class ShallowCopyExample {
public:
    int* data;

    // Constructor
    ShallowCopyExample(int value) {
        data = new int(value);  // Dynamically allocate memory
    }

    // Shallow copy constructor (compiler-generated by default)
    ShallowCopyExample(const ShallowCopyExample& other) {
        data = other.data;  // Just copy the pointer
    }

    // Destructor
    ~ShallowCopyExample() {
        delete data;  // Free the memory
    }
};

int main() {
    ShallowCopyExample obj1(10);
    ShallowCopyExample obj2 = obj1;  // Shallow copy

    cout << "Original: " << *(obj1.data) << endl;  // 10
    cout << "Copy: " << *(obj2.data) << endl;      // 10

    // Change obj1's data
```

```
    *(obj1.data) = 20;

    cout << "After modifying obj1:" << endl;
    cout << "Original: " << *(obj1.data) << endl;  // 20
    cout << "Copy: " << *(obj2.data) << endl;      // 20 (shared memory)

    return 0;
}
```

Output:

```yaml
Original: 10
Copy: 10
After modifying obj1:
Original: 20
Copy: 20
```

In this example, both `obj1` and `obj2` share the same `data`. Modifying `data` in one affects the other because they are pointing to the same memory.

---

2. Deep Copy

A **deep copy** creates a completely independent copy of an object, including duplicating all objects it references. This means that changes to the copied object will not affect the original object, as they occupy different memory locations.

Key Points:

- Copies both the object and any nested objects or references.
- The original and the copy do not share any data members (memory addresses).
- Changes to the deep-copied object will not affect the original.

Example:

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

class DeepCopyExample {
public:
    int* data;

    // Constructor
    DeepCopyExample(int value) {
        data = new int(value);  // Dynamically allocate memory
    }

    // Deep copy constructor
    DeepCopyExample(const DeepCopyExample& other) {
        data = new int(*(other.data));  // Create a new copy of the data
    }

    // Destructor
```

```
    ~DeepCopyExample() {
        delete data;  // Free the memory
    }
};

int main() {
    DeepCopyExample obj1(10);
    DeepCopyExample obj2 = obj1;  // Deep copy

    cout << "Original: " << *(obj1.data) << endl;  // 10
    cout << "Copy: " << *(obj2.data) << endl;      // 10

    // Change obj1's data
    *(obj1.data) = 20;

    cout << "After modifying obj1:" << endl;
    cout << "Original: " << *(obj1.data) << endl;  // 20
    cout << "Copy: " << *(obj2.data) << endl;      // 10 (independent memory)

    return 0;
}
```

Output:

```yaml
Original: 10
Copy: 10
After modifying obj1:
Original: 20
Copy: 10
```

In this example, `obj1` and `obj2` have their own separate memory for `data`, so changes to one do not affect the other.

Summary of Differences:

Shallow Copy	Deep Copy
Copies only references to the original object's data.	Copies the entire object, including all nested objects.
The copied object shares references with the original object.	The copied object is completely independent of the original.
Changes to shared data will affect both objects.	Changes to one object do not affect the other.
Faster, since it only copies references.	Slower, since it creates new copies of everything.

When to Use:

- **Shallow Copy** is generally used when you don't need to modify the referenced objects and just need another reference to the original object.
- **Deep Copy** is used when you need a complete, independent copy of the object and its nested data, especially if you expect changes to happen to the copied object without affecting the original.

Let me know if you need further clarification!