Shallow and Deep Copy:

Documents having pics is example of Shallow Copy

**Copy Constructor**

**Definition:** A copy constructor initializes a new object as a copy of an existing object. The default copy constructor performs a shallow copy, which copies the values of data members directly.

class MyClass {

public:

int value;

MyClass(int x) : value(x) {

std::cout << "Parameterized Constructor called\n";

}

// Copy Constructor

MyClass(const MyClass& other) : value(other.value) {

std::cout << "Copy Constructor called\n";

}

};

int main() {

MyClass obj1(10); // Parameterized Constructor

MyClass obj2 = obj1; // Copy Constructor

return 0;

}

#include <iostream>

class ShallowCopyClass {

public:

int\* data;

ShallowCopyClass(int value) {

data = new int(value);

}

// Shallow Copy Constructor

ShallowCopyClass(const ShallowCopyClass& other) : data(other.data) { }

~ShallowCopyClass() {

delete data;

}

};

int main() {

ShallowCopyClass obj1(10);

ShallowCopyClass obj2 = obj1; // Shallow Copy Constructor

\*obj2.data = 20; // Changes data for both obj1 and obj2

std::cout << \*obj1.data << " " << \*obj2.data << "\n"; // Both will print 20

return 0;

}

DEEP COPY:

1. **Shallow Copy vs. Deep Copy**

**1. Shallow Copy:**

* **Definition:** Copies the values of the object’s data members. If the data members include pointers, only the pointer values are copied, not the data they point to.
* **Real-Life Example:** Imagine you have a document with references to images (pointers). If you make a shallow copy of this document, both the original and the copy will point to the same images. Changes to the images will affect both documents.

**Example Code for Shallow Copy:**

cpp

Copy code

class ShallowCopyClass {

public:

int\* data;

ShallowCopyClass(int value) {

data = new int(value);

}

// Shallow Copy Constructor

ShallowCopyClass(const ShallowCopyClass& other) : data(other.data) { }

~ShallowCopyClass() {

delete data;

}

};

int main() {

ShallowCopyClass obj1(10);

ShallowCopyClass obj2 = obj1; // Shallow Copy Constructor

\*obj2.data = 20; // Changes data for both obj1 and obj2

std::cout << \*obj1.data << " " << \*obj2.data << "\n"; // Both will print 20

return 0;

}

**Drawbacks:** If you delete or modify the data through one object, it affects the other object, potentially leading to undefined behavior (e.g., double deletion).

**2. Deep Copy:**

* **Definition:** Creates a copy of the data members, including dynamically allocated memory. Each object gets its own copy of the data.
* **Real-Life Example:** Consider a photo album with each album containing its own set of photos. If you create a deep copy of an album, the new album will have its own copies of the photos, so changes to the photos in one album won't affect the other.

**Example Code for Deep Copy:**

cpp

Copy code

class DeepCopyClass {

public:

int\* data;

DeepCopyClass(int value) {

data = new int(value);

}

// Deep Copy Constructor

DeepCopyClass(const DeepCopyClass& other) {

data = new int(\*other.data); // Allocate new memory and copy the value

}

~DeepCopyClass() {

delete data;

}

};

int main() {

DeepCopyClass obj1(10);

DeepCopyClass obj2 = obj1; // Deep Copy Constructor

\*obj2.data = 20; // Changes only obj2's data

std::cout << \*obj1.data << " " << \*obj2.data << "\n"; // Prints 10 and 20

return 0;

}

**Benefits of Deep Copy:**

* **Independence:** Objects do not share the same data, preventing unintended side effects from modifications or deletions.
* **Safety:** Avoids issues related to double deletion and data corruption.

1. **Summary**

* **Shallow Copy:** Efficient but can lead to issues with shared resources. Suitable for simple objects without dynamic memory or pointers.
* **Deep Copy:** More resource-intensive but prevents issues related to shared resources. Ideal for objects that manage dynamic memory or resources.

In your interview, you can emphasize the need for deep copying when dealing with dynamic resources to ensure data integrity and independence between objects.

Good question! Let's clarify how `obj2` is created and why `\*` is used in the context of modifying `data`.

### \*\*Creation of `obj2`\*\*

In the provided example:

```cpp

ShallowCopyClass obj1(10); // 'obj1' is created on the stack

ShallowCopyClass obj2 = obj1; // 'obj2' is created on the stack as a shallow copy of 'obj1'

```

- \*\*Stack vs. Heap:\*\* Both `obj1` and `obj2` are created on the stack, not on the heap. The creation on the stack means that these objects are automatically managed and will be destroyed when they go out of scope (i.e., when the function ends).

### \*\*Understanding `\*` and Pointer Usage\*\*

- \*\*Pointer Member:\*\* The `data` member of `ShallowCopyClass` is a pointer (`int\*`). This means it holds the memory address of an integer, not the integer value itself.

- \*\*Accessing Pointer Data:\*\* To access or modify the value pointed to by a pointer, you use the dereference operator `\*`.

Here's why `\*` is used in `\*obj2.data = 20;`:

1. \*\*Pointer Dereferencing:\*\* `obj2.data` is a pointer to an `int`. To access or modify the integer value that `data` points to, you need to dereference the pointer using `\*`.

2. \*\*Modification:\*\* `\*obj2.data = 20;` modifies the integer value that `data` points to. Since `obj2` and `obj1` share the same pointer (because of the shallow copy), changing the value through `obj2` also affects `obj1`.

### \*\*Detailed Example with Explanation:\*\*

```cpp

class ShallowCopyClass {

public:

int\* data;

ShallowCopyClass(int value) {

data = new int(value); // Allocate memory and initialize

}

// Shallow Copy Constructor

ShallowCopyClass(const ShallowCopyClass& other) : data(other.data) { }

~ShallowCopyClass() {

delete data; // Potential issue with double deletion

}

};

int main() {

ShallowCopyClass obj1(10); // 'obj1' is created with data = 10

ShallowCopyClass obj2 = obj1; // 'obj2' is a shallow copy of 'obj1'

\*obj2.data = 20; // Modify the value pointed to by obj2.data, which also affects obj1.data

std::cout << \*obj1.data << " " << \*obj2.data << "\n"; // Prints 20 20, since both point to the same location

return 0;

}

```

### \*\*Key Points:\*\*

- \*\*`obj1` and `obj2`\*\* are both stack-allocated objects.

- \*\*`data`\*\* is a pointer, so `\*data` is used to access the integer value it points to.

- \*\*Shallow Copy\*\* causes `obj1.data` and `obj2.data` to point to the same integer, so changes through one affect the other.

### \*\*Why is `\*` Used?\*\*

- \*\*Pointer to Value:\*\* `data` is a pointer, and `\*` is needed to access or modify the value stored at the memory address `data` points to.

- \*\*Shared Data:\*\* Since `data` is shared between `obj1` and `obj2`, any modification through one object reflects in the other, demonstrating the shallow copy’s limitations.

By understanding and explaining this, you show that you grasp pointer usage and the implications of shallow copying in C++.