

## Quiz #1 Answers

1. Explain how these two declarations of a Connect-4 board produce different behaviors with the default copy operation. How can one of them cause odd behaviors in the code?

The default copy operator is a shallow copy. The first declaration actually has the memory for the board inside the objects so it is copied in a shallow copy. Of course, this code doesn't give us flexible sizing for the board. The second declaration uses pointers so only the pointer is in the objects, the memory for the board will be elsewhere. As a result, when the default copy operator is applied to this, you will get two objects of type Connect4 that both point to the same board memory. A move on one board will occur on the other one as well. This generally isn't what you want.

```
class Connect4 {
public:
    // Methods here
private:
    char board[7][6];
};

class Connect4 {
public:
    // Methods here
private:
    char **board;
};
```

2. What is an explicit constructor and why are they helpful in C++?

The explicit keyword is used to modify single argument constructors in C++. The only reason it is needed is because single argument constructors play a secondary role in C++, that of type caster. When a constructor exists for a class that takes a single argument of a certain type, it will be called any time an object of that type is cast to this class. This can cause problems with implicit casts though. C++ puts in implicit type conversions in places where one type is used and another type is expected if there is a conversion between the two types. The explicit keyword prevents C++ from using that constructor in those implicit type conversions.

Extra Credit: Think about the code we wrote for the Connect-4 board in class. If we wanted it to be able to use an arbitrarily sized board for play, what changes would have to be made? Describe both the changes and why we have to make them.

There are two ways to do this. One requires more changes than the other. One way is to start with what was done in problem one and make the board a pointer to a pointer and dynamically allocate it to a given size. This requires adding members that store the size of the array and using those as bounds in loops. With this route you also have to write a constructor that does the allocation and a destructor to deallocate that memory. The other approach would be to make the board a vector of vectors. With this approach the destructor isn't required, though a constructor would be nice just so that they could pass in the size and have the vectors sized. It also requires that the loop bounds be modified to use the vector size.