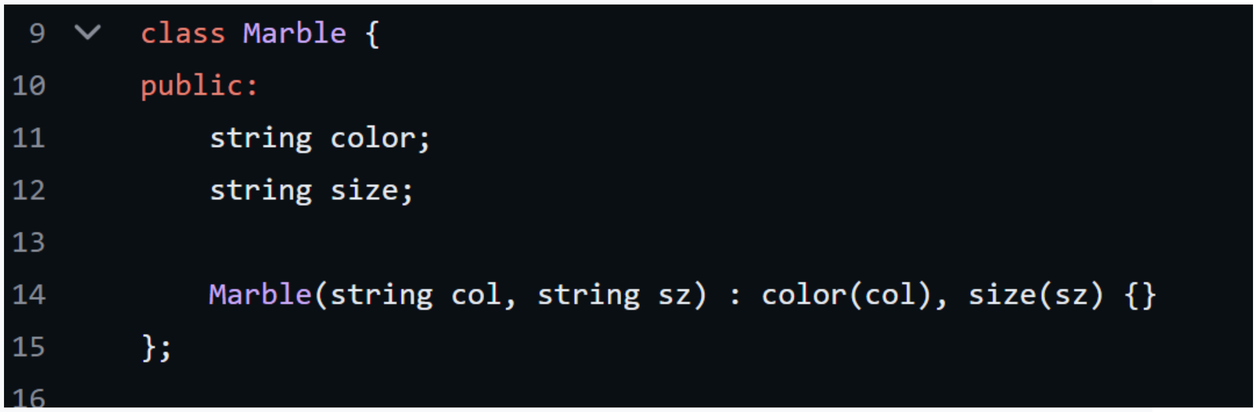
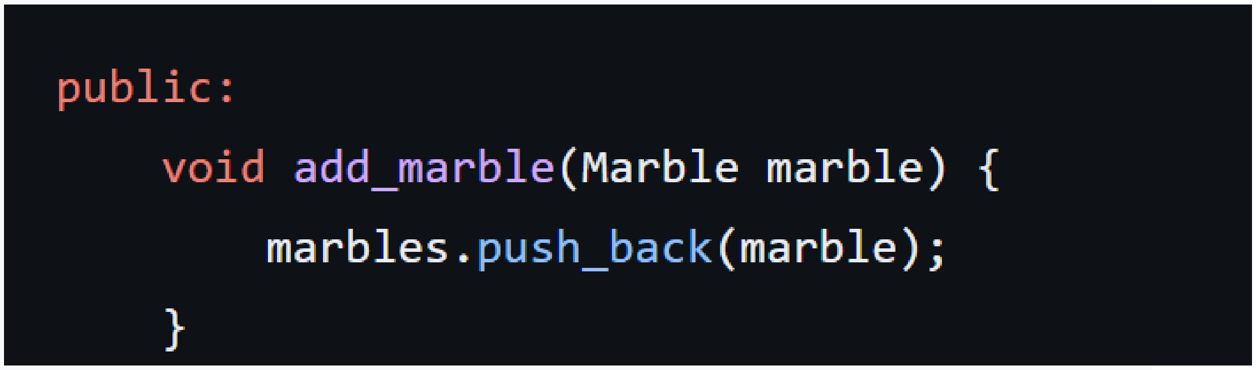
1. some way of representing marbles (what makes up a marble in this program?),



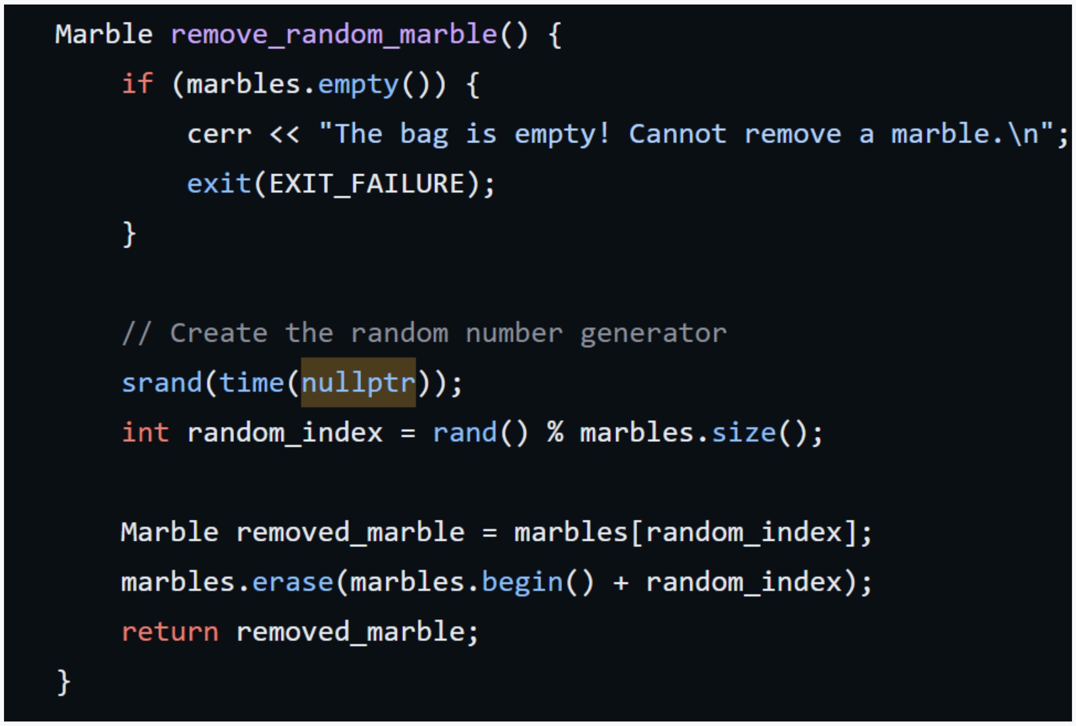
```cpp
class Marble {
public:
    string color;
    string size;

    Marble(string col, string sz) : color(col), size(sz) {}
};
```

2. a way to add new marbles into the bag (how do we interact with marbles and add them into the bag?),



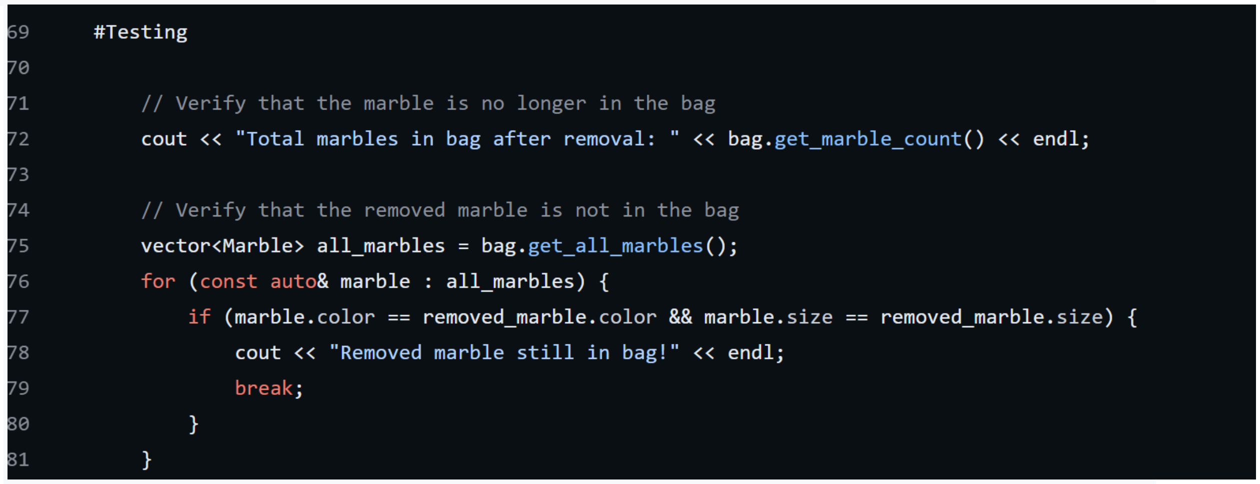
```cpp
public:
    void add_marble(Marble marble) {
        marbles.push_back(marble);
    }
```

3. a way to remove a marble out of the bag (perhaps a random marble taken out of the bag?),



```cpp
Marble remove_random_marble() {
    if (marbles.empty()) {
        cerr << "The bag is empty! Cannot remove a marble.\n";
        exit(EXIT_FAILURE);
    }

    // Create the random number generator
    srand(time(nullptr));
    int random_index = rand() % marbles.size();

    Marble removed_marble = marbles[random_index];
    marbles.erase(marbles.begin() + random_index);
    return removed_marble;
}
```

4. a few ways that we could use to show that our implementation should be working correctly (tests), (perhaps you start with an empty bag, put a marble in with some known values, then you pull the marble out and verify that it has the same values, perhaps you try to pull a marble out of an empty bag, perhaps you try to add 3 billion marbles... maybe you do not have to go that high or maybe your solution is smart enough to deal with this 🙂)



```
#Testing

    // Verify that the marble is no longer in the bag
    cout << "Total marbles in bag after removal: " << bag.get_marble_count() << endl;

    // Verify that the removed marble is not in the bag
    vector<Marble> all_marbles = bag.get_all_marbles();
    for (const auto& marble : all_marbles) {
        if (marble.color == removed_marble.color && marble.size == removed_marble.size) {
            cout << "Removed marble still in bag!" << endl;
            break;
        }
    }
```