## Code without Objects

Suppose you were writing a program to model several cars. In your program, you want to keep track of each car's color and the distance the car has traveled, and you want to be able to increment this distance and print out the car's properties. You could write something like the code below to accomplish this:

In [ ]:
```
#include <iostream>
#include <string>
using std::string;
using std::cout;

int main()
{
    // Variables to hold each car's color.
    string car_1_color = "green";
    string car_2_color = "red";
    string car_3_color = "blue";

    // Variables to hold each car's initial position.
    int car_1_distance = 0;
    int car_2_distance = 0;
    int car_3_distance = 0;

    // Increment car_1's position by 1.
    car_1_distance++;

    // Print out the position and color of each car.
    cout << "The distance that the " << car_1_color << " car 1 has traveled is: " << car_1_distance << "\n";
    cout << "The distance that the " << car_2_color << " car 2 has traveled is: " << car_2_distance << "\n";
    cout << "The distance that the " << car_3_color << " car 3 has traveled is: " << car_3_distance << "\n";

}
```

Compile & Execute | Explain

*Loading terminal (id_8cijeu2), please wait...*

This works for the few cars that are defined in the program, but if you wanted the program to keep track of *many* cars this would be cumbersome. You would need to create a new variables for every car, and the code would quickly become cluttered. One way to fix this would be to define a `Car` class with those variables as attributes, along with a few class methods to increment the distance traveled and print out car data.

## Code with Objects

In the next cell, the code above has been rewritten with a `Car` class.

In [ ]:
```
#include <iostream>
#include <string>
using std::string;
using std::cout;

// The Car class
class Car {
  public:
    // Method to print data.
    void PrintCarData()
    {
        cout << "The distance that the " << color << " car " << number << " has traveled is: " << distance << "\n";
    }

    // Method to increment the distance travelled.
    void IncrementDistance()
    {
        distance++;
    }

    // Class/object attributes
    string color;
    int distance = 0;
    int number;
};

int main()
{
    // Create class instances for each car.
    Car car_1, car_2, car_3;

    // Set each instance's color.
    car_1.color = "green";
    car_2.color = "red";
    car_3.color = "blue";

    // Set each instance's number.
    car_1.number = 1;
    car_2.number = 2;
    car_3.number = 3;

    // Increment car_1's position by 1.
    car_1.IncrementDistance();

    // Print out the position and color of each car.
    car_1.PrintCarData();
    car_2.PrintCarData();
    car_3.PrintCarData();

}
```

Compile & Execute | See Explanation

*Loading terminal (id_0btk71p), please wait...*

This looks ok, and you have reduced the number of variables in `main`, so you might see how this could be more organized going forward. However, there is now a lot more code than you started with, and the `main` doesn't seem much more organzied. The code above still sets the attributes for each car after the car has been created.

### Adding a Constructor

The best way to fix this is to add a *constructor* to the `Car` class. The constructor allows you to instantiate new objects with the data that you want. In the next code cell, we have added a constructor for `Car` that allows the `number` and `color` to be passed in. This means that each `Car` object can be created with those variables.

In [ ]:
```
#include <iostream>
#include <string>
using std::string;
using std::cout;

class Car {
  public:
    void PrintCarData()
    {
        cout << "The distance that the " << color << " car " << number << " has traveled is: " << distance << "\n";
    }

    void IncrementDistance()
    {
        distance++;
    }

    // Adding a constructor here:
    Car(string c, int n)
    {
        // Setting the class attributes with
        // The values passed into the constructor.
        color = c;
        number = n;
    }

    string color;
    int distance = 0;
    int number;
};

int main()
{
    // Create class instances for each car.
    Car car_1 = Car("green", 1);
    Car car_2 = Car("red", 2);
    Car car_3 = Car("blue", 3);

    // Increment car_1's position by 1.
    car_1.IncrementDistance();

    // Print out the position and color of each car.
    car_1.PrintCarData();
    car_2.PrintCarData();
    car_3.PrintCarData();
}
```

Compile & Execute | Explain

*Loading terminal (id_xvhh213), please wait...*

This is now beginning to look better. The `main` is more organized than when we first started, although there is a little more code overall to accomodate the class definition. At this point, you might want to separate your class definition into it's own `.h` and `.cpp` files. We'll do that in the next concept!