```cpp
// Lecture 50: Delegating Constructors
class Car {
private: // this is optional.
    float fuel{ 0 }; // Lecture 44 - Non-static Variables initialisers
    float speed{ 0 };
    int passengers{ 0 };
    int arr[5] = { 1,2,3 };
    char *p{}; // p will be default NULL.

public:
    Car();
    Car(float fuel);
    Car(float fuel, int passengers);
    ~Car();
//
};
Car::Car(){
    fuel = 0; speed = 0; passengers = 0;
}
Car::Car(float fuel){
    speed = 0; this->fuel = fuel; passengers = 0;
}
Car::Car(float fuel, int passengers)
{
    speed = 0; this->fuel = fuel; this->passengers = passengers;
}
// Notice that, the same code is repeated in different constructors. This is a source of bugs, because it is possible that in one of the constructors, the initialisation may be skipped or it may be incorrect.

// So we'll apply the delegating constructors concept of C++11 here:
// the common initialisation code will be kept in the most elaborate constructor.

// So consider the following implementation:
// Constructor 1
Car::Car():Car(0){
    std::cout << "ctor1"<< std::endl;
```

```
36    }
37    // Constructor 2
38    Car::Car(float fuel):Car(fuel, 0){
39        std::cout << "ctor2" << std::endl;
40    }
41    // Constructor 3
42    Car::Car(float fuel, int passengers)
43    {
44        speed = 0; this->fuel = fuel; this->passengers =
          passengers;
45        std::cout << "ctor3" << std::endl;
46    }
47    // In the following implementation,  ctor 1 delegates
        to ctor2, and ctor2 delegates to ctor3.
48    // When we call the default ctor (ctor1), what happens
        is that ctor2 will be invoked, which will inturn
        invoke ctor3.
49
50    // Labelling all the constructors as such, the program
        will hit line 34, and will go to line 38, and will
        go line 42, which will execute lines 43 to 46, and
        then it will go back to execute lines 39 to 40, and
        then execute lines 35 to 36.
51    // When the program sees a call to the other
        constructor, it will JUMP to the next constructor.
52    // Like recursion lol.
53    // This is why the initialisation code should always
        be kept in a single constructor.
54
```