```
1
2   void Car::Accelerate() {
3       this->speed++; // this will increase this car's
          speed.
4       this->fuel -= 0.5f; // this will decrease this
          car's fuel.
5   }
6
7   /*
8   in the assembly, this means - Load Effective Address
9   and it will take the address of the car object and it
      will place it in the register called 'ecx'.
10
11  This 'ecx' is then accessed inside the member
      functions to get hold of the object.
12
13  In the accelerate function, you will see that it moves
      the address from ecx into the this pointer.
14
15  So all the calls that access the members of the class
      will always first take the address of the object
      from the 'this' pointer.
16
17  To summarise, you don’t need to use the ‘this’ pointer
      manually
18  to access the members of the class because the
      compiler implicitly
19  uses it. if this pointer is optional, then WHERE do we
      use it?
20  */
21
22  // Case 1: Argument name same as attribute name
23  void Car::AddPassengers(int passengers)
24  {
25      // The presence of passengers as an argument will
          hide the member variable passengers. In order to
          distinguish the two, we can use the 'this'
          pointer.
26      this->passengers = passengers;
27  }
28  // Case 2: If we have a function that accepts a Car
      Object
```

```
29    void foo(const Car &car)
30    {
31        // If we need to invoke the foo function from a
          member function, so you need to invoke the foo
          function and pass the car object as an argument:
          so how would u get the car object in the member
          function? You can use it for the 'this 'pointer .
32    }
33    void foo_caller()
34    {
35        foo(*this); // since the function requires an
          object, we dereference the this pointer.
36    }
37    // Case 3: 'this' pointer in some member functions
        that need to return the current object to the caller
38    // Examples of such member functions are the prefix
        form of increment or decrement operator and
        assignment operator. We'll see these examples in
        operator overloading.
39
```