SCC.111 Software Development – Lecture 26: Composition and OO Case Study

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

Introduction

- Last lecture, we looked at:
  - Libraries
  - Namespaces

- Today we're going to look at a non-trivial example of an OO program.
- Take in a few more OO concepts along the way
  - Objects as function parameters
  - C++ references and initializer lists
  - Composition

Objects as Function Parameters 1

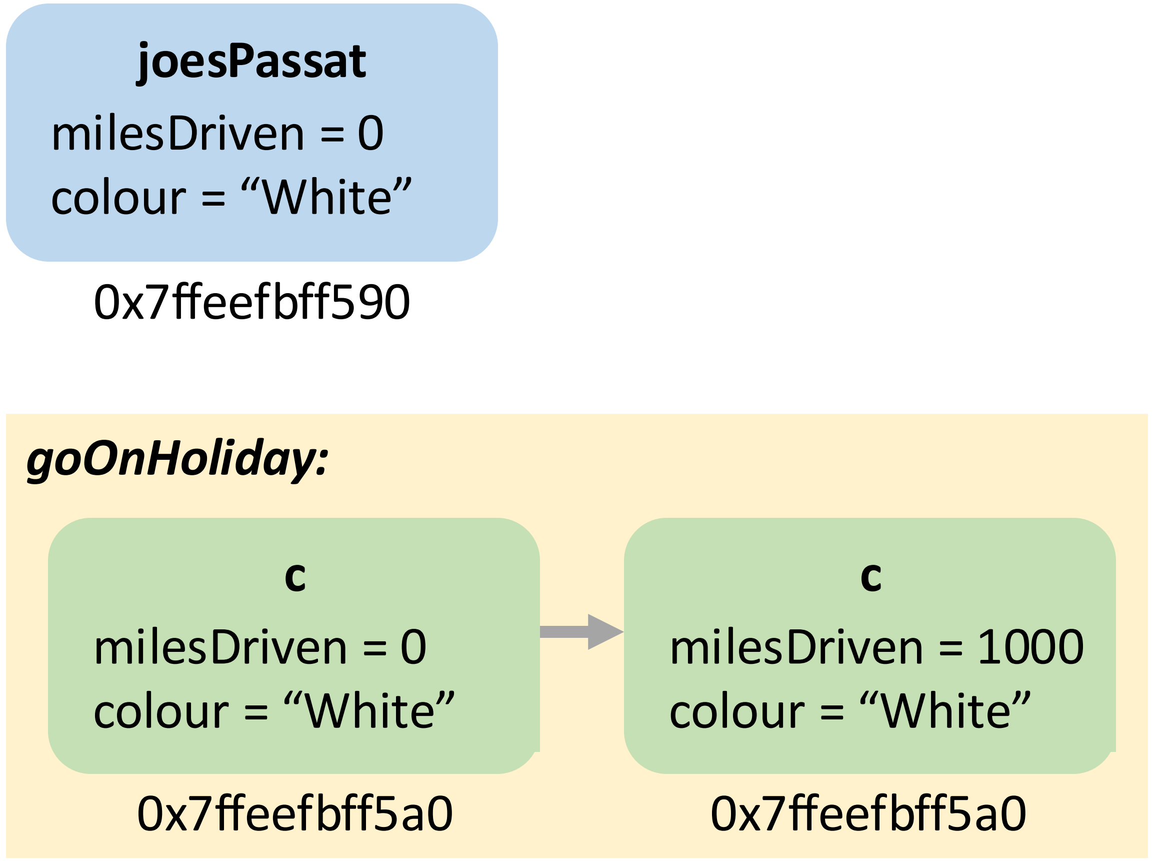
In C++, objects can be passed as arguments and returned from a function the same way we pass and return any other variable.

What mileage will this code print?

Join at menti.com | use code 6136 6044

```
#include "Car.h"

void goOnHoliday(Car c)
{
    c.drive(1000);
}

int main()
{
    Car joesPassat((char *)"White");

    goOnHoliday(joesPassat);

    joesPassat.show();
}
```

Objects as Function Parameters 1



```
#include "Car.h"

void goOnHoliday(Car c)
{
    c.drive(1000);
}

int main()
{
    Car joesPassat((char *)"White");
    goOnHoliday(joesPassat);

    joesPassat.show();
}
```

Objects as Function Parameters 2

In C++, objects can be passed as arguments and returned from a function the same way we pass and return any other variable.

- C++ is a pass-by-value language. When using an **object** as a function parameter, we are therefore:
    - Creating a new object instance, with identical attributes.
    - Interacting with that copy inside the function, independently of the actual variable that was passed.

Why is this a good default behaviour?

```
#include "Car.h"

void goOnHoliday(Car c)
{
    c.drive(1000);
}

int main()
{
    Car joesPassat((char *)"White");

    goOnHoliday(joesPassat);

    joesPassat.show();
}
```

Pointers to Objects…

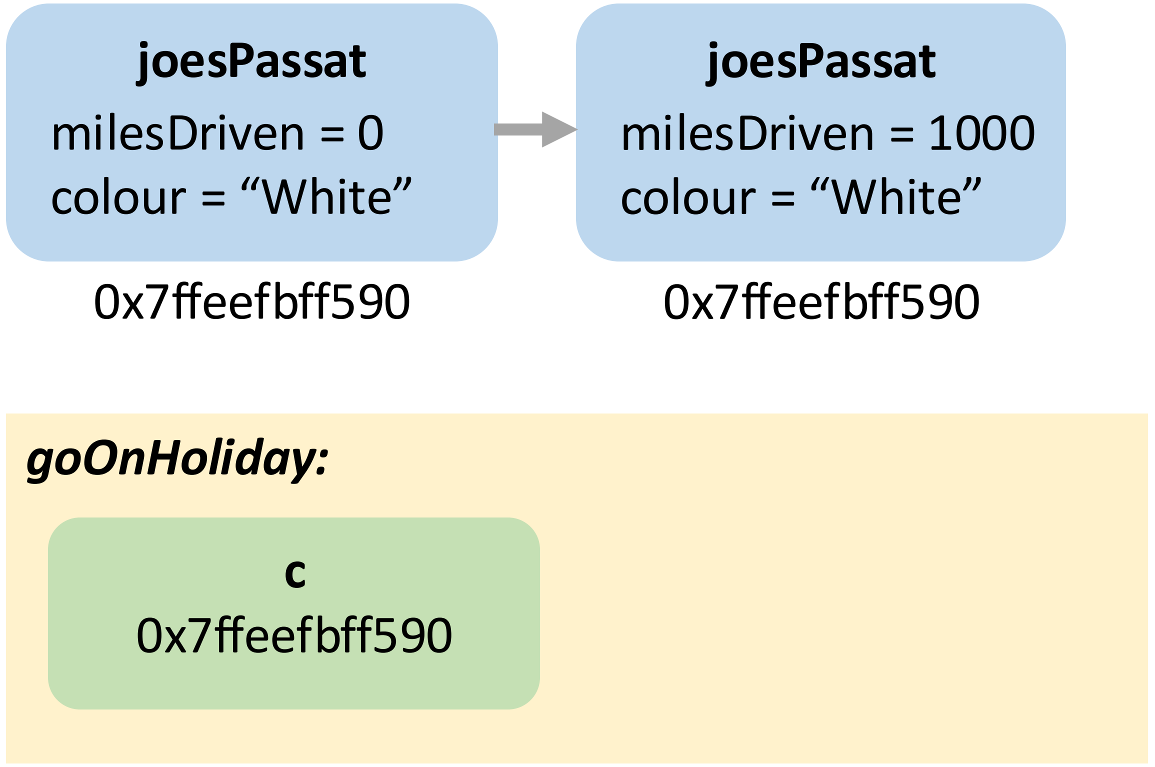
We can pass a pointer to an object, just like we can any other variable…

What mileage will this code print?

Join at menti.com | use code 7700 3489

```
#include "Car.h"

void goOnHoliday(Car *c)
{
    c->drive(1000);
}

int main()
{
    Car joesPassat((char *)"White");

    goOnHoliday(&joesPassat);

    joesPassat.show();
}
```

Pointers to Objects…



```
#include "Car.h"

void goOnHoliday(Car *c)
{
    c->drive(1000);
}

int main()
{
    Car joesPassat((char *)"White");
    goOnHoliday(&joesPassat);

    joesPassat.show();
}
```

Pointers to Objects…

We can pass a pointer to an object, just like we can any other variable…

- When using a **pointer** as a function parameter we:
    - Explicitly pass the **memory location** (address) of the variable by value as a parameter.
    - Dereference that pointer to access the variable's data in memory.
    - Permit **reassignment** of the pointer to point to a **different memory location**, should we wish to.

```
#include "Car.h"

void goOnHoliday(Car *c)
{
    c->drive(1000);
}

int main()
{
    Car joesPassat((char *)"White");

    goOnHoliday(&joesPassat);

    joesPassat.show();
}
```

References to Objects…

In C++, we can pass references to an object too!

- When using a **reference** as a function parameter we:
  - Create an "alias" for the same variable. The compiler treats the variable and reference as 100% equivalent.
  - Indicate the function will take a reference by the **&** symbol in the parameter list.
  - Implicitly create and dereference the reference ☺

References never permit reassignment to a different variable… references are immutable.

References must always refer to something. They are not permitted to be NULL.

What mileage will this code print?

```
#include "Car.h"

void goOnHoliday(Car &c)
{
    c.drive(1000);
}

int main()
{
    Car joesPassat((char *)"White");

    goOnHoliday(joesPassat);

    joesPassat.show();
}
```

References anywhere…1

We can use references anywhere we use a variable

- Parameter lists
- Local variables
- Global variables
- Class attributes

```cpp
#include "Car.h"

void goOnHoliday(Car &c)
{
    Car &sameCar = c;
    c.drive(1000);
}
```

Are these code samples legal C++?

References anywhere…2

We can use references anywhere we use a variable

- Parameter lists
- Local variables
- Global variables
- Class attributes

```
#include "Car.h"

void goOnHoliday(Car &c)
{
    Car &sameCar;
    sameCar = c;
    c.drive(1000);
}
```

Are these code samples legal C++?

References anywhere…3

We can use references anywhere we use a variable

- Parameter lists
- Local variables
- Global variables
- Class attributes

```cpp
#include "Car.h"

Car &sameCar;

void goOnHoliday(Car &c)
{
    sameCar = c;
    c.drive(1000);
}
```

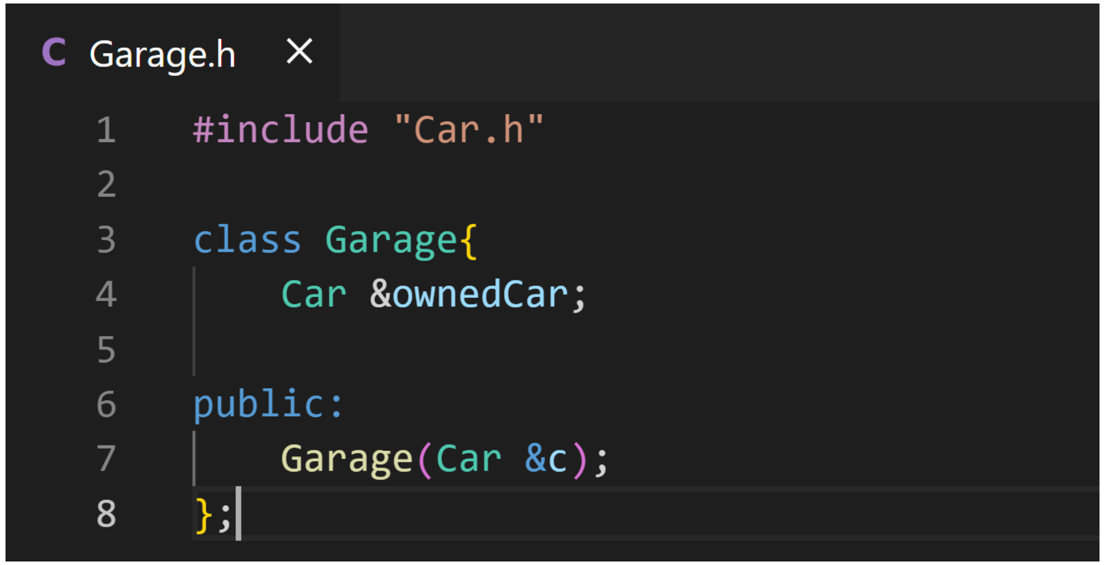
Are these code samples legal C++?

Initializer lists…

Another way to initialize variables, including references, in a class constructor

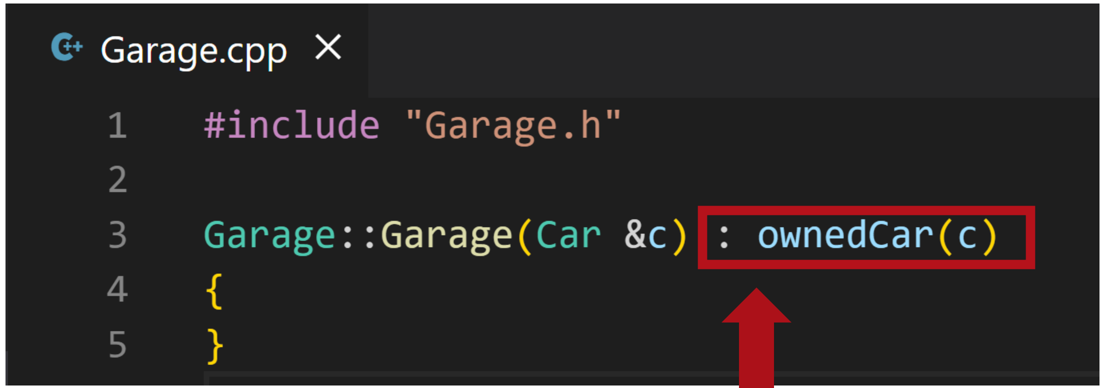
- It is quite common want to use a reference as an attribute of a class.
- Yet we have seen we can't create references without assigning them a value.
- Constructors are designed to allow the initialization of an object, so is the natural place to solve this.
- But do not guarantee all attributes are initialized before they are used.

Initializer lists provide this guarantee. Simply a list of the values to use to initialize attributes.



Garage.h

```
#include "Car.h"

class Garage{
    Car &ownedCar;

public:
    Garage(Car &c);
};
```



Garage.cpp

```
#include "Garage.h"

Garage::Garage(Car &c) : ownedCar(c)
{
}
```

Composition

Sometimes we want to group together lots of objects to make something even more awesome.

- Like a Dragon
- Like a Micro:bit

- In Object Oriented programming languages we can easily do this through **composition**.
- Simply create a class with attributes (variables) that make up the thing you want…

Then your new class implicitly has all their capabilities.

Case Study

Let's take a look at the C++ the makes up the micro:bit firmware…

- As an example of **composition**.
- Look out for the OO principles we've learned.
- Look out for the C++ we've discussed.

main.cpp

MicroBit.h / MicroBit.cpp

build

Summary

- Today we learned that:

  - Objects in C++ are passed by value
  - We can use pointers to objects, just like any other variable
  - C++ references provide a (slightly!) safer alternative
  - We can use composition to create new classes from object instances of others

  - **Real systems use these principles. This is not just an academic exercise.**