Learning Objectives: Multilevel Inheritance

- **Define multilevel inheritance**
- **Create multilevel inheritance from a class with a base class**
- **Extend a class and override a function within multilevel inheritance**

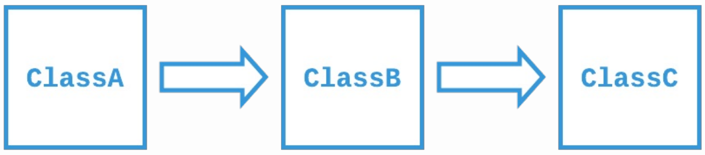
Multiple Inheritance

Multiple Inheritance

Multiple inheritance is a condition where a class inherits from more than one base class. C++ allows multiple inheritance for both associated and unassociated base classes; however, for this particular section, we will only go over multiple inheritance with **associated** base classes, meaning one base class is derived from another base class.

Multilevel Inheritance

Multiple inheritance with associated base classes is called **multilevel inheritance**. This is a condition where a class inherits from more than one base class, but each base class is associated with each other. The image below shows `ClassC` inheriting from `ClassB`, which in turn inherits from `ClassA`. This is an example of multilevel inheritance.



.guides/img/inheritance/MultiLevelInheritance

The classes `Carnivore` and `Dinosaur` are already defined. `Carnivore` is the base class for `Dinosaur`. Create the `Tyrannosaurus` class which is a derived class of `Dinosaur`. The constructor for `Tyrannosaurus` takes a string and two doubles and gets associated with the constructor from the `Dinosaur` class.

```
//add class definitions below this line

  class Tyrannosaurus : public Dinosaur {
    public:
      Tyrannosaurus(string d, double s, double w) : Dinosaur(d,
        s, w) {}
  };

//add class definitions above this line
```

Instantiate a `Tyrannosaurus` object with the appropriate arguments. This t-rex `tiny` is 12 meters tall, weighs 14 metric tons, and eats whatever it wants. Print the `size` attribute to make sure inheritance is working as expected.

```
//add code below this line

Tyrannosaurus tiny("whatever it wants", 12, 14);
cout << tiny.GetSize() << endl;

//add code above this line
```

challenge

Try these variations:

- Print the `weight` attribute with `cout << tiny.GetWeight() << endl;`
- Print the `diet` attribute with `cout << tiny.GetDiet() << endl;`

Extending & Overriding Functions

Extending a Class within Multilevel Inheritance

Multilevel inheritance works just like single inheritance except there are more than one derived class. Add the following code as class definitions in the text editor.

```
//add class definitions below this line

class ClassC : public ClassB {
  public:
    void Bonjour() {
      cout << "Bonjour" << endl;
    }
};

//add class definitions above this line
```

Instantiate a `ClassC` object to call the `Bonjour` function. Then use the scope resolution operator : to invoke the `Hello` function from both `ClassB` and `ClassA`.

```
  //add code below this line

  ClassC c;
  c.Bonjour();
  c.ClassB::Hello();
  c.ClassA::Hello();

  //add code above this line
```

challenge

Try this variation:

- Extend `ClassC` with the function `AuRevoir` that prints `Au revoir`. Then call this function in `main`.
  ▼ **Solution**

```
//add class definitions below this line

class ClassC : public ClassB {
  public:
    void Bonjour() {
      cout << "Bonjour" << endl;
    }

    void AuRevoir() {
      cout << "Au revoir" << endl;
    }
};

//add class definitions above this line

int main() {

  //add code below this line

  ClassC c;
  c.AuRevoir();

  //add code above this line

  return 0;

}
```

Overriding a Function within Multilevel Inheritance

Like extending a class, overriding a function works the same in multilevel inheritance as it does in single inheritance. Change `ClassC` so that it overrides the `Hello` function.

```
//add class definitions below this line

class ClassC : public ClassB {
  public:
    void Hello() {
      cout << "Hello from Class C" << endl;
    }
};

//add class definitions above this line
```

Now replace the call to `Bonjour` with a call to `Hello`.

```
  //add code below this line

  ClassC c;
  c.Hello();
  c.ClassB::Hello();
  c.ClassA::Hello();

  //add code above this line
```

Notice how calling the `Hello` function automatically defaults to the function within the object’s specified class. If you want to call the same function as specified from within another base class, simply use the scope resolution operator `:` as shown above.