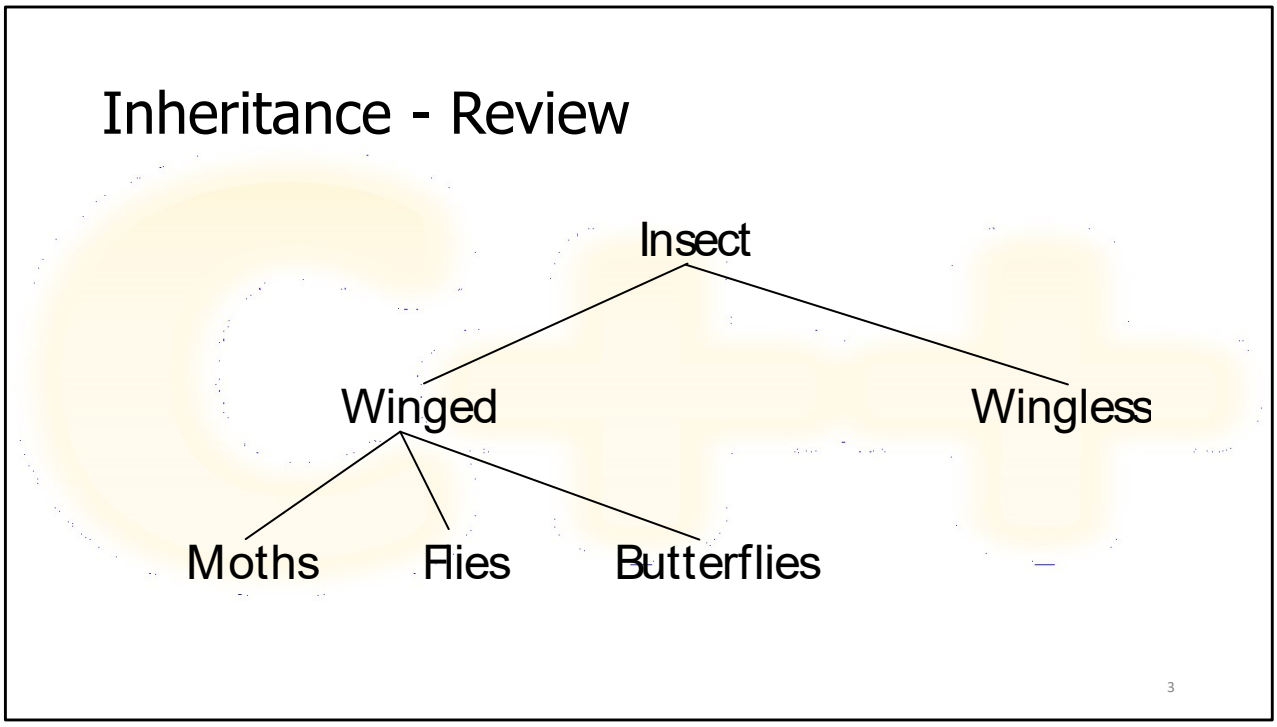
# Inheritance

# Contents

- Objectives
  - To be able to implement single class inheritance hierarchies
- Contents
  - Inheritance - Review
  - Deriving Classes
  - Public, Private & Protected Inheritance
  - Protected Access
  - Base Class Initialisation
  - Multiple Inheritance




Classes can be organised into hierarchies. The shared characteristics are located at the top of the tree and are inherited by members of the tree beneath them. Both the data and methods of a class hierarchy are inherited. All flies will therefore inherit the data and methods of winged insects. Applying this principle to program design allows for code re-use and structured software.

Inheritance defines new classes of objects in terms of extending existing classes. In other words an 'is a kind of' relationship. The butterfly is a kind of insect, however it is an extended insect. Butterflies have wings.

## Deriving Classes

```
 1 #pragma once
 2
 3 class Person
 4 {
 5 public:
 6     Person(void);
 7     ~Person(void);
 8     void set_age(int a);
 9     void set_height(double h);
10     int get_age();
11     double get_height();
12 private:
13     int age;
14     double height;
15 };
```

```
 1 #pragma once
 2 #include "Person.h"
 3
 4 class Adult : public Person
 5 {
 6 public:
 7     Adult(void);
 8     ~Adult(void);
 9     void set_smoker(bool s);
10     bool is_smoker();
11 private:
12     bool smokes;
13 };
```

4

Inheritance defines new classes based directly on existing classes. Inherited classes are known as base classes. Inheriting classes are known as derived classes. A derived class can itself be used as a base class, and a base class may have many derived classes. The concept of type extension is used to add capabilities and features to a base type.

# Protected Access

| | public | protected | private |
|---|---|---|---|
| Class users | Yes | No | No |
| Derived classes | Yes | Yes | No |
| Base class | Yes | Yes | Yes |

5

Class users can access only public member functions and data. Since data members should always be declared private the laws of encapsulation are upheld. Protected items cannot be accessed by class users, but can be accessed either from within the base class or a derived class. Private class members may only be accessed from within the class in which they are declared.

## Public, Private & Protected Inheritance

class adult : ***public*** person

| Access in Base Class | Base Class Inherited as | Access in Derived Class |
|---|---|---|
| Public<br>Protected<br>Private | Public | Public<br>Protected<br>No access |
| Public<br>Protected<br>Private | Protected | Protected<br>Protected<br>No access |
| Public<br>Protected<br>Private | Private | Private<br>Private<br>No access |

6

In the previous example we used public inheritance:

class adult : ***public*** person

It is also possible to specify protected and private inheritance. The most common inheritance is public inheritance. Using public inheritance you have no access to base class private members, protected access to base class protected members etc.

This table shows how the inheritance accessor affects the accessors in the base class.

## Base Class Initialisation

```
 8 Adult:: Adult(int a, double h, bool s)
 9 : Person(a,h)
10 {
11     smokes = s;
12 }
```

7

Data members of base classes are initialised before those of derived classes. Constructors for the base class are called before those of any derived classes. The reverse is true when destroying objects. The destructors for derived classes are called first. This is true even though the base class may be a subset of the derived class where expansion is taking place.
It is not recommended to initialise base class members from within a derived class constructor. It is the base class's responsibility to do this.

There are a number of ways of initialising base class variables from a derived class.

You can perform straightforward assignment to the base class variables from within the derived class constructor. However, this method is potentially inefficient as the base class variable may already have been initialised when the base constructor was called. Also, this method requires the variable to be visible to the derived class. i.e. the data must be public (Very bad) or protected (still undesirable).

Another method which gets around the protected / public drawback of the previous method is to use accessor routines to set values in the base class. However, this does not get around the fact that a base class constructor will already have been called & may have initialised the variable once already. In addition, this method adds a function call overhead to the problem.

The best method of initialising the base class is to use a colon initialiser.

## Class Scoping Issues

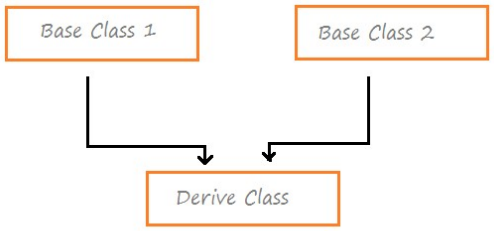
```
1 #include "Adult.h"
2 #include <iostream>
3 using namespace std;
4
5 void Adult::Display() const
6 {
7     Person::Display();
8     cout << "Smokes: " << smokes << endl;
9 }
```

8

The scope of a base class effectively surrounds that of a derived class. Therefore, a name in the derived class will effectively hide the identical name in the base class. If it is required to access the base class item the scope can be resolved explicitly.

# Multiple Inheritance

Multiple Inheritance (Aggregation) should be avoided.  Always prefer containment over aggregation.




Multiple Inheritance

9

# Exercise