EL2310 – Scientific Programming

Lecture 14: Inheritance and Polymorphism in C++

Hakan Karaoguz (hkarao@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 14: Inheritance and Polymorphism in C++

- Announcements
- Wrap Up
- Inheritance
- Polymorphism

Last time

- Classes in Depth
- Overloading of Functions and Operators

Today

- Inheritance
- Polymorphism

Announcements

- This week's schedule:
  - 5 October → @ Teknikringen 14 Plan 5 room 523
  - 10 October → Help Session
- C++ project will be announced this week

Lecture 14: Inheritance and Polymorphism in C++

Announcements

Wrap Up

Inheritance

Polymorphism

Source and header files

Header file ex A.h:

```
// Preprocessor guards
#ifndef A_H
#define A_H
class A{
public:
  A();
private:
  int m_X;
}; // Don’t forget the
semicolon!!
#endif
```

Source file ex A.cpp:

```
#include "A.h"
A::A(){
m_X=0;
}
```

Setters and Getters

- In order to modify/access to the private data members of a class, we use *set* and *get* functions.

A.h:

```
Class A {
 public:
  A(); // Constructor
  void setm_X(int x);
  int getm_X();
 private:
  int m_X;
};
```

A.cpp:

```
A::A(){m_X = 0;}
void A::setm_X(int x) {
  m_X = x;
}
int A::getm_X() {
  return m_X;
}
```

keyword `const`

- To make some functions, data members and objects as ”read-only”
- `const` function type:
- Ex: `void fcn(int arg) const;`
- `const` data members:
- Ex: `const int m_X;`
- `const` data members cannot be modified by assignment.
- `const` objects:
- Ex: `const A a;`
- `const` objects can only use `const` member functions
- *constructors* cannot be `const`!! But they can be used to initialize `const` objects

Static members

- A `static` member (data/function) is the same across all objects.
- It's a special member of a *class* that can be accessed even if there is no object of that class!:
- Ex: `int A::m_Counter = 0;` if `m_Counter` is a static data member of class `A`
- `static` member functions cannot be `const`. Because `const` member functions only work for the object that it operates.

Lecture 14: Inheritance and Polymorphism in C++

Announcements

Wrap Up

Inheritance

Polymorphism

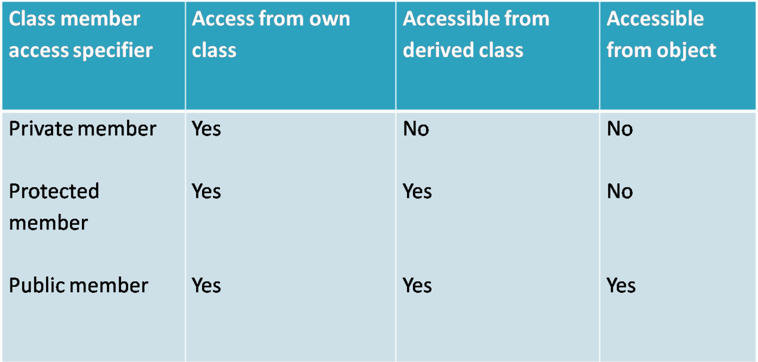
Inheritance

- Inheritance is a way to show a relation like “is a”
- Ex: a Car is a Vehicle
- A Car inherits many of its properties from being a vehicle
- These same properties could be inherited by a Truck or a Bus
- Syntax:
  ```
  class Car :  public Vehicle
  ```
  specifies that Car inherits from Vehicle

Inheritance cont’d

- We call the `Vehicle` the *base class*
- A `Car` is a *derived class* of `Vehicle`
- A base class can have more than one derived classes (Bus, Truck)
- The aim of inheritance: Increase code reusability, sharing of similar functions

Inheritance cont’d



Class member access specifier	Access from own class	Accessible from derived class	Accessible from object
Private member	Yes	No	No
Protected member	Yes	Yes	No
Public member	Yes	Yes	Yes

Task1

- Create a Bus class that is derived from base class Vehicle
- Add a function and a private variable for the Bus class that returns the number of decks of the bus (single or double decker)

Accessing methods of inherited class from Base Class (Downcasting) (Not recommended!!)

-
  ```
  class Vehicle
  {
  public:
    void drive();
  }
  class Car:  public Vehicle
  {
  public:
    void openTrunk();
  }
  ```
- `Vehicle *v = new Car();`
- `v->drive();` runs drive() from the Vehicle part of the Car
- `v->openTrunk();` NOT POSSIBLE!
- But: `((Car *)v)->openTunk();` WORKS!

Lecture 14: Inheritance and Polymorphism in C++

Announcements

Wrap Up

Inheritance

Polymorphism

Polymorphism

- Program in general rather than specific
- The word means having many forms
- C++ polymorphism means that a call to a member function will cause a different function to be executed

`virtual` functions

- What if we want the object know what it “really” is and run the correct `drive()` method?
- Declare the method with the keyword `virtual`

  ```
  class Vehicle {
    virtual void drive();
  }
  class Car:  public Vehicle {
    virtual void drive();
  }
  ```
- `Vehicle *v1 = new Vehicle();`
- `Vehicle *v2 = new Car();`
- `v1->drive();` runs drive() from the Vehicle
- `v2->drive();` runs drive() from the Car

Interfacing: Abstract class

- In C++, abstract classes provides interfaces
- Not to be confused with data abstraction
- To make a class abstract : declare at least one of its functions as pure ”virtual” function.
- A pure virtual function is specified by placing ”= 0”

-
  ```
  class Car
  {
  public:
    virtual double getNrWheels() = 0; // pure virtual
  function
  private:
    double NrWheels
  };
  ```
- Virtual functions cannot have overloaded versions in derived classes!

Abstract class

- Abstract classes cannot be instantiated
- Purpose : A base classes which could be inherited in other classes
- Inherited classes have to overload each of the virtual functions in the base class
- Meaning: B (inherits the base class A) supports the interface provided by A.

Task2

- Modify the `Vehicle` and the `Car` classes such that the getNumberofWheels function becomes pure virtual function and the Vehicle class becomes an abstract class
- Do not forget that the derived class should implement its own function in this case.