面向对象程序设计
Object-Oriented Programming(OOP)

Classes ,Inheritance, Polymorphism

A Survey of Programming Techniques

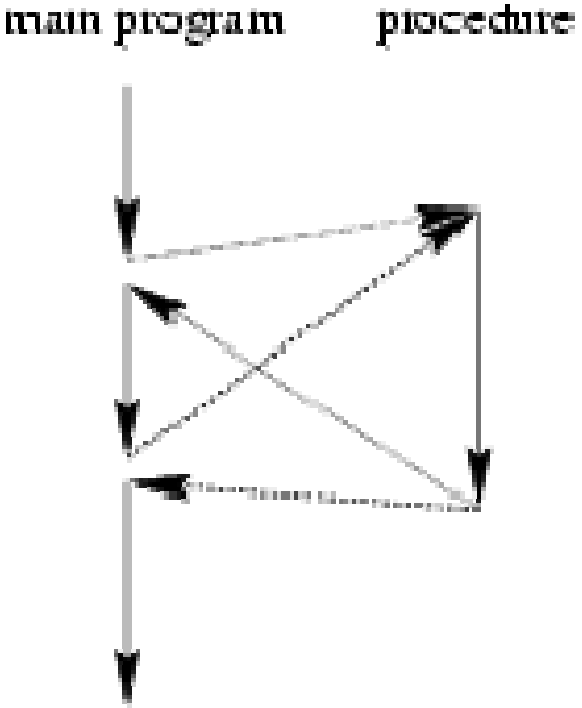
- Unstructured programming,
- procedural programming,
- modular programming and
- object-oriented programming.

Unstructured Programming

- 仅包含一个main主程序,直接操作全局数据global data 。
- 如果同样的“*语句序列*”用在不同的位置，则需要复制这个序列。
- 抽取这些序列，并命名它们为子过程(*procedures*)/函数，并提供调用和返回值的机制。

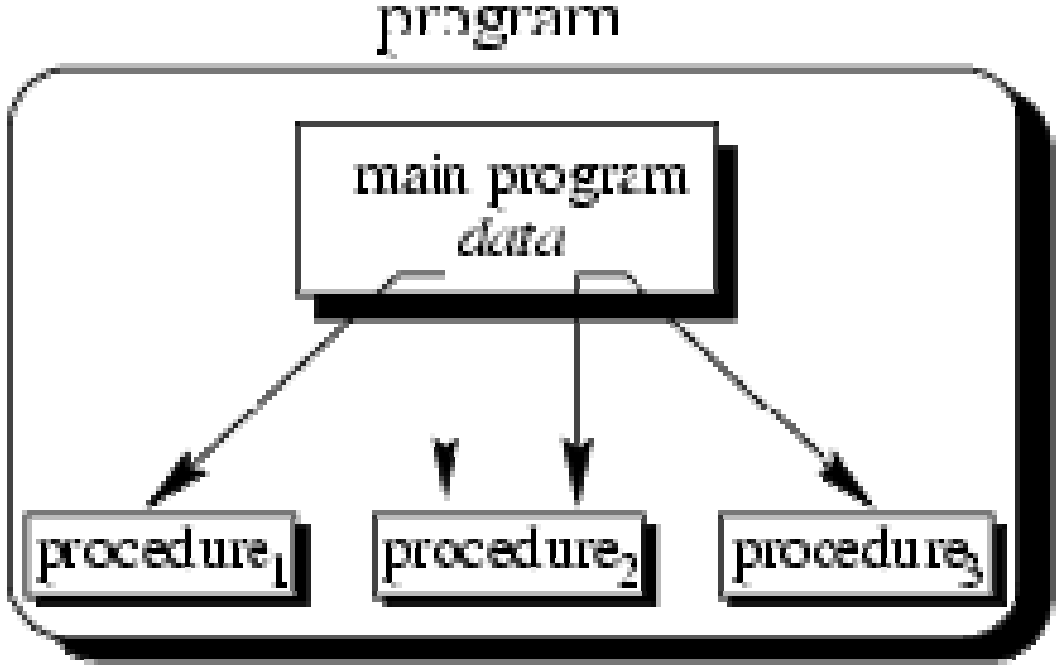
Procedural Programming

- 过程Procedure（函数、方法）：一个接一个执行的指令序列。
- 一个过程的某个语句调用其他过程，在被调用过程结束后的位置，主调过程的控制流继续执行。



Procedural Programming

- 程序Program：就是一系列过程调用。
- 主程序负责将数据传递给每具体的过程，数据被该过程处理，改过程结束后将返回结果。这种调用关系可以用一个调用层次图表示。



[例] 员工平均薪水统计. 可分解为下述子任务：

1. 计算有多少员工

2. 获取每个员工的薪水

3. 计算薪水之和

4. 总薪水/员工数

“计算薪水之和”又可分解为下述步骤：

1. 获取每个员工记录

2. 得到该员工的薪水

3. 将该员工薪水加到总薪水上

4. 处理下一个员工

“获取每个员工记录”又可分解为下述步骤：

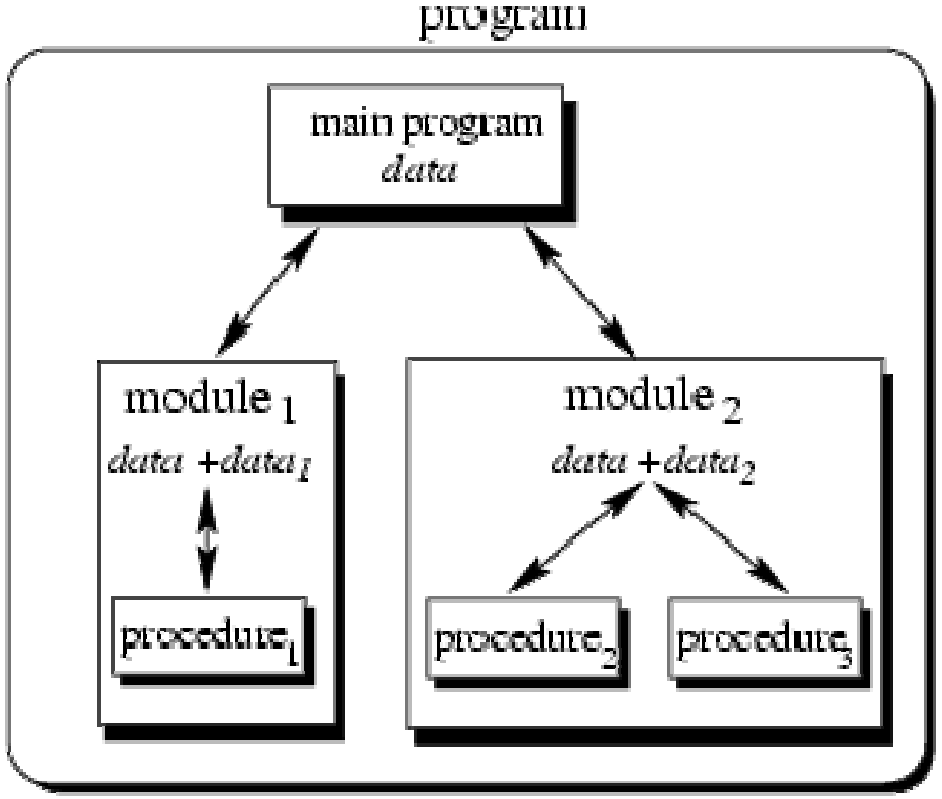
1. 打开员工文件

2. 定位到该员工记录

3. 读取该员工记录

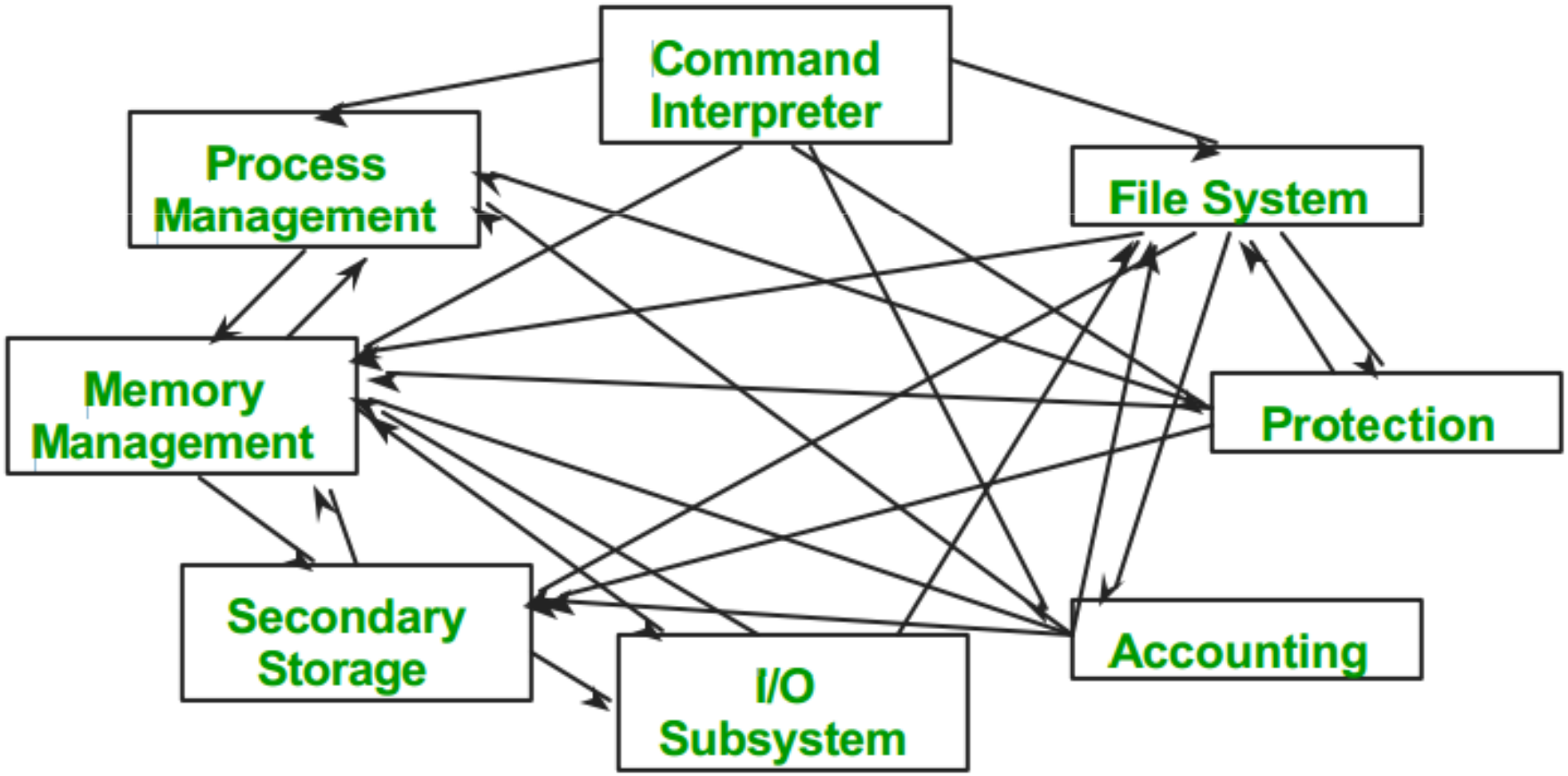
Modular Programming

- 相关功能的procedures被分组为独立的 模块 *modules*.
- 一个程序现在就不是单一的，而是分解为多个更小模块，这些模块通过过程调用交互作用，构成一个完整程序。



A MODULAR OPERATING SYSTEM

- It is clear what modules an OS should provide
- Not so clear how to hook them together (well)…



Object-Oriented Programming

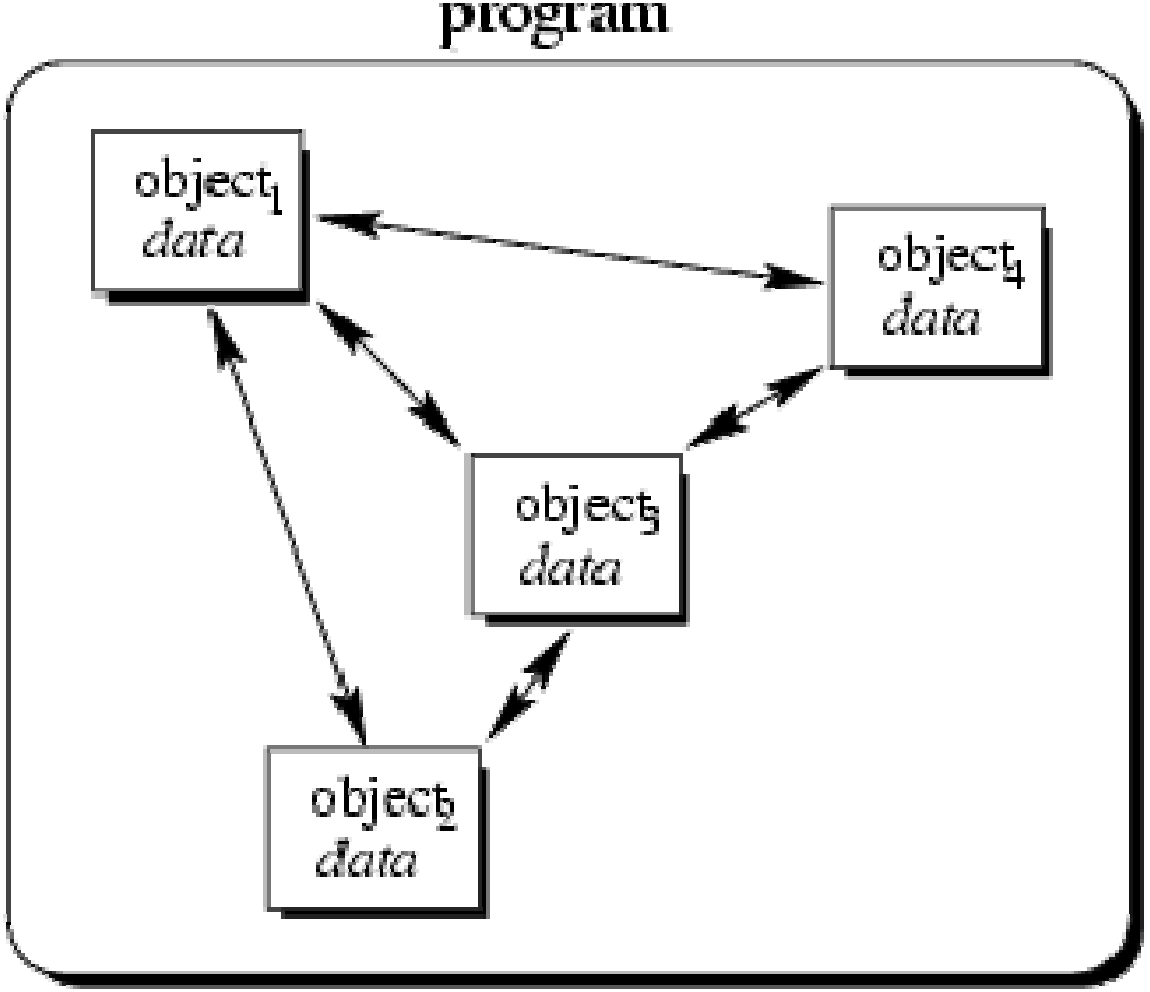
- 过程式程序设计
  - 分解为任务与子任务
  - 用函数实现各个任务
  - 指示计算机依次执行任务
- 对于很多任务和大数据量的问题，过程式设计的程序会很复杂，难以维护。
- 比如制造一个汽车，汽车的每个部件都有不同的变量和信息，使得很难将和某个部件如车轮相关的代码都组合在一起。大量变量和函数及其关系很难理清楚！

Object-Oriented Programming

- 管理复杂任务的有效方法是将自完善的(self-sufficient)、模块化代码片段进行分别打包！
- 人们是以对象(Objects)方式思考世界的：方向盘、踏板、车轮是如何相互作用的。
- OOP允许程序员撇开细节考察作为简洁的、自完善盒子的对象抽象功能及对象间的相互作用关系。允许程序员通过对象的相互通信构建一个清晰的、易维护的程序架构！

Object-Oriented Programming

- 程序由一组具有自主功能的、交互的对象构成。
- 对象间通过传递消息交互协作完整任务。



封装(Encapsulation)-类(class)与对象 (object)

- 现实世界中，对象(Object)描述具有状态(State)和行为(behavior)的事物。例如，手机有开和关状态、有显示游戏或播放音乐行为。
- 对象式语言中的对象(Object)是类似的，具有状态(State)和行为(behavior) 。 State由一些域构成，叫做变量(variables)或属性(proporities), behavior包含一组methods(或Functions)。
- 类(class)则是创建同类型的不同具体对象的蓝图或模具或模板Tempalte. 即一个类用于创建具体对象，这个类定义了这些对象的states 或 behaviors. 或者说类规定了该类型的对象的成员变量和成员方法有哪些。

封装(Encapsulation)-类(class)与对象 (object)

- 类定义了同类型对象的数据(也称变量或属性)和函数(也称方法)成员，其访问修饰有public和private等，其中private的成员不能被外部访问，外部只能调用类的public成员。这些public成员就构成了该类对外部的接口(Interface).
- 类(对象)如同一个盒子，其中只有暴露在盒子外的接口可被外部看到的。
- 面向对象程序由一些对象构成，每个对象可完成其特有的功能，对象之间通过发送消息来交换信息或通知其他对象完成某个动作。

  -向其他对象发送消息,是通过调用该对象的方法完成的。

封装(Encapsulation)-类(class)与对象 (object)

```
class circle {
    double r;
  public:
     circle(double r_=0) { r =r _; }
     double area(){
        return 3.1415*r*r;
    }
};

  circle  C(3.5);

  double A = C. area();
```

定义一个类: circle

定义circle类的对象: C

向对象 C发送消息area()，该消息返回结果A

继承(Inheritance)-派生类

- 继承使得我们可以定义相关类之间的层次结构。
- 假设编写一个车辆库存的管理程序，其中有小汽车(Cars)和卡车(trucks)，我们可能会定义两个类Car 和truck，为此，我们可能不得不复制两个类共同的特性。
- C++中我们可以将公共的代码放在一个叫做Vehicle的类中。

```
class Vehicle {
protected:
    string license;
    int year;
public:
    Vehicle(const string &myLicense, const int myYear)
        : license(myLicense), year(myYear) {}
    const string getDesc() const
        {return license + " from " + stringify(year);}
    const string &getLicense() const {return license;}
    const int getYear() const {return year;}
};
```

继承(Inheritance)-派生类

```
class Vehicle {
protected:
    string license;
    int year;
public:
    Vehicle(const string &myLicense, const int myYear)
        : license(myLicense), year(myYear) {}
    const string getDesc() const
        {return license + " from " + stringify(year);}
    const string &getLicense() const {return license;}
    const int getYear() const {return year;}
};
```

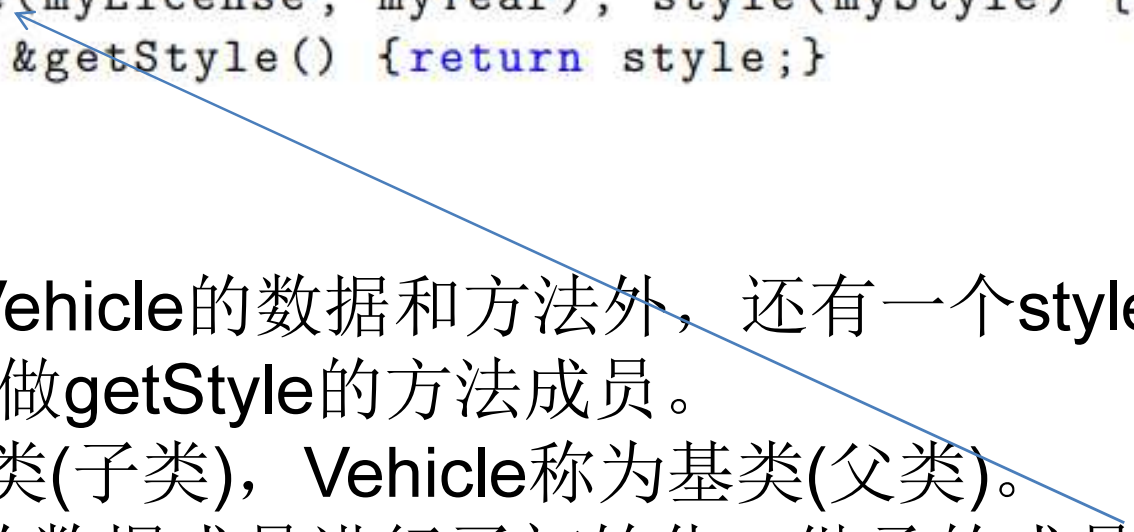
protected类似于private

初始化成员列表

说明某个地方有一个叫做stringify函数将数字转化为字符

继承(Inheritance)-派生类

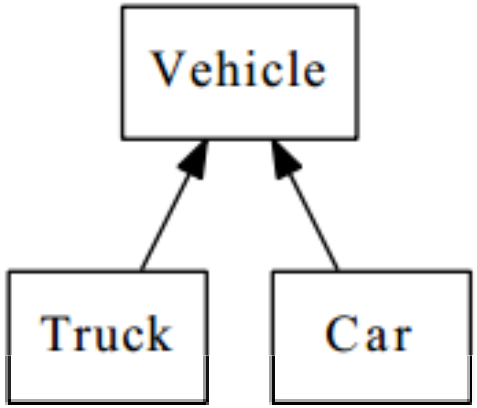
```
class Car : public Vehicle { // Makes Car inherit from Vehicle
    string style;

public:
    Car(const string &myLicense, const int myYear, const string
        &myStyle)
        : Vehicle(myLicense, myYear), style(myStyle) {}
    const string &getStyle() {return style;}
};
```



- Car除了具有Vehicle的数据和方法外，还有一个style的数据成员和一个叫做getStyle的方法成员。
- Car 称为派生类(子类)，Vehicle称为基类(父类)。
- Car只对派生的数据成员进行了初始化，继承的成员的初始化有基类构造函数完成！

继承(Inheritance)-派生类

- 类似的，可以定义派生类 truck. 这三个类构成一个层级关系。



Is-a 和 Has-a

- 一个类A依赖于另外一个类B的两种方式：

  1. 每个A类对象有一个B类对象。如每个Vehicle有一个string类的对象（license）。

  2. 每个A类对象也是一个B类对象。如每个Car类对象也是一个Vehicle对象。

- 继承(Inheritance) 用来定义Is-a关系，而Has-a关系是通过类的数据成员反应的。

重载方法(Overriding Methods)

- 我们也许需要对Car类定义定义不同于Vehicle的getDesc方法。此后，当对Car对象调用getDesc 时将调用这个重定义的getDesc方法。这种重定义基类同名函数的方法称为重载。

```
class Car : public Vehicle { // Makes Car inherit from Vehicle
    string style;
public:
    Car(const string &myLicense, const int myYear, const string
        &myStyle)
        : Vehicle(myLicense, myYear), style(myStyle) {}
    const string getDesc() // Overriding this member function
        {return stringify(year) + ' ' + style + ": " + license
            ;}
    const string &getStyle() {return style;}
};
```

差别化编程Programming by difference

- 继承仅仅允许我们重载已有方法或增加新方法，而不能移除基类的方法。

Access Modifiers and Inheritence

- 如果Vechile内的属性year 和license被定义为私有的，则我们将不能再起派生类如Car中访问这些属性；为了让派生类可以访问而外部不能访问基类属性，可将这些基类的属性定义为Protected.
- public用于指定派生类型。Class Car: public Vechilie 说明基类Vechilie的成员的访问性在派生类中保持不变。如基类的protected成员在派生类中仍然是protected的。

Access specifiers in the base class

	private	protected	public
private inheritance	The member is inaccessible.	The member is private.	The member is private.
protected inheritance	The member is inaccessible.	The member is protected.	The member is protected.
public inheritance	The member is inaccessible.	The member is protected.	The member is public.

多态(polymorphism)

- C++支持函数多态和类多态（ function polymorphism and class polymorphism）。多态指相同名字表达不一样的能力。
- 用一个指向基类的指针分别指向基类和派生类对象，并2次调用getDesc()函数输出，结果如何？

```
Car c(“CarLicense”,2001,”CarStyle”);
Vehicle  v (“License”,1900),* p = &v;
 p->getDesc();
 p = &m;
 p-> getDesc();
```

虚函数(Virtual Function)和多态(polymorphism)

```
class vehicle{
    …
    virtual const string getDesc ( ) {  … }
};
```

```
Car c(“CarLicense”,2001,”CarStyle”);
Vehicle  v (“License”,1900),* p = &v;
 p->getDesc();
 p = &m;
 p-> getDesc();
```

虚函数(Virtual Function)和多态(polymorphism)

- 通过指针或引用访问一个类对象的虚函数，将调用指针或引用指向的实际对象的相应虚函数。

多态性

- 包含虚函数的类也称为多态类。

纯虚函数与抽象基类

- 函数体=0的虚函数叫做纯虚函数。
- 包含一个纯虚函数的类叫做抽象基类。

```
// abstract class CPolygon
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area () =0;
};
```

纯虚函数与抽象基类

- 函数体=0的虚函数叫做纯虚函数。
- 包含一个纯虚函数的类叫做抽象基类。
- 不能定义抽象基类的变量(对象)！

```
// abstract class CPolygon
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area () =0;
};
```

纯虚函数与抽象基类

- 抽象基类用作接口，派生类必须实现纯虚函数。

```
class Polygon {
  protected:
    int width, height;
  public:
    Polygon (int a, int b) : width(a),
height(b) {}
    virtual int area (void) =0;
    void printarea()
      { cout << this->area() << '\n'; }
};
```

```
class Rectangle: public Polygon {
  public:
    Rectangle(int a,int b) :
Polygon(a,b) {}
    int area()
      { return width*height; }
};

class Triangle: public Polygon {
  public:
    Triangle(int a,int b) :
Polygon(a,b) {}
    int area()
      { return width*height/2; }
};
```

纯虚函数与抽象基类

- 抽象基类用作接口，派生类必须实现纯虚函数。

```
int main () {
  Polygon * ppoly1 = new Rectangle (4,5);
  Polygon * ppoly2 = new Triangle (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  delete ppoly1;
  delete ppoly2;
  return 0;
}
```

多继承Multiple inheritance

```
class One
{
  // class internals
};

class Two
{
  // class internals
};

class MultipleInheritance : public One, public Two
{
  // class internals
};
```

作业

- 1. 实现一个公司员工信息管理程序，员工包括雇员和经理两种人员，当然经理也是雇员。请定义至少下述三种类,并对雇员和经理类定义一个print虚函数，用以输出员工信息。

```
class Employee{
    //…
};
class Manager: public Employee{
    private: int level;
    //…
};
class Company{
    std::vector< Employee *> employees; //请自己查询vector用法
    //…
};
```