261

Chapter 9

The C++ Programming Language

Object Oriented Programming (inheritance)

262

The Personnel Example

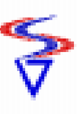
- **Suppose we want to computerize our personnel records...**
- **We start by identifying the two main types of employees we have:**

The magic of M$

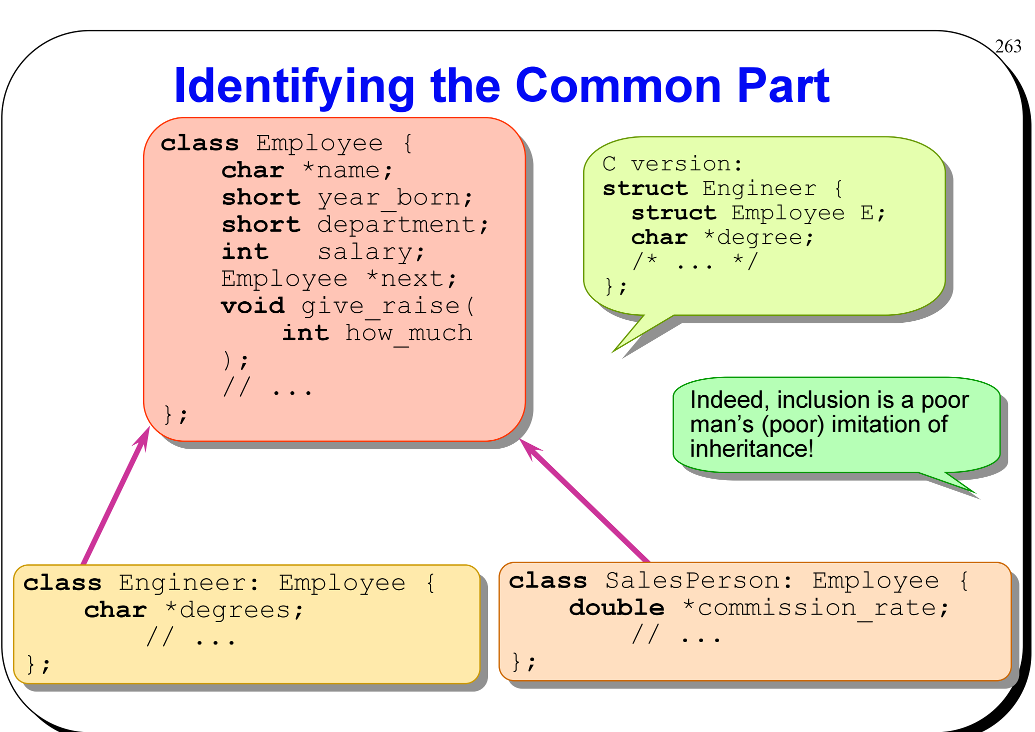
```
struct Engineer {
     Engineer *next;
     char *name;
     short year_born;
     short department;
     int   salary;
     char *degrees;
     void give_raise(
          int how_much
     );
     // ...
};
```

```
struct SalesPerson {
SalesPerson *next;
char *name;
short year_born;
short department;
int   salary;
double *comission_rate;
void give_raise(
int how_much
);
// ...
};
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only



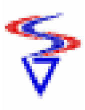
264

Inheritance

- **The *Behaviour* and the *Data* associated with a subclass are an extension of the properties of the superclass.**
- `Engineer` and `SalesPerson` extend, each in its own way, the data and behaviour of `Employee`.
- Identifying the `Employee` abstraction, helps us define more types:

```
class Manager: Employee {
    char *degrees;
        // ...
};
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only

265

Inheritance Mechanism

- A class may *inherit* from a *base class*, in which case the inheriting class is called a *derived* class.
- A class may *override* inherited *methods* (member functions), but not inherited *data*.
- *A Hierarchy* of related classes, that share code and data, is created.
- Inheritance can be viewed as an incremental refinement of a base class by a new derived class.

Base class

Derived class

Derived class

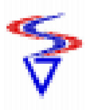
Arrows are always from a derived class to the base class, since the derived class knows about the base class, but not vice-versa.

266

The Benefits of Inheritance

- **Software Reusability.** Inheritance allows you to modify or extend a package somebody gave you without touching the package's code
  - Least important !
  - **Saves programmer time:**
  - **increase reliability, decrease maintenance cost, and,**
  - **if code sharing occurs, smaller programs.**
- **Consistency of Interface.** An overridden function must have the same return type:
  - More important !
  - **Saves user time: Easier learning, and easier integration.**
  - **guarantee that interface to similar objects is in fact similar.**
- **Polymorphism.** Different objects behave differently as a response to the same message.
  - Most important !

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only

267

Inheritance Vector Example

Here is a simple version of a `Vector` class that implements an unchecked array of integers:

```
class Vector {
    int len, *buff;
public:
    Vector(int l): len(l), buff(new int[l]){}
    ~Vector()                { delete[] buff; }
    int size() const         { return    len; }
    int& operator[](int i) { return buff[i]; }
};
```

```
int main()
{
     Vector v(10);
     // Populate the vector, and then use it.
     v[5] = v[6] + 5;
     // ..
}
```

268

A Checked Vector

- We may also need a vector whose bounds are checked on every reference.
- This is done by defining a new derived **class** `CheckedVector` that inherits the characteristics of the base **class** `Vector`, and overrides some of its methods:

```
class CheckedVector: public Vector {
public:
   CheckedVector(int size): Vector(size) {}
   int& operator[](int);   // A primitive overriding
};

int& CheckedVector :: operator[](int i)
{
  if (0 > i || i >= size())
     error();               // Exceptions :-(
  return Vector::operator[](i);
}
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion
For personal use only

269

Points Of Interest

- When a **class** inherits from a **class/struct**, the **public** keyword should be used, as in

```
class CheckedVector: public Vector {
...
```

(Don't ask why at this time.)

- Constructors are not Inherited. If we haven't defined a constructor for CheckedVector, then the compiler would try to generate an empty default constructor and **fail** in doing so since there is no default constructor of the base class Vector.

- The construction (initialization) of the ***sub-object*** of the inherited class Vector must be done in the ***initialization list*** of the constructor of the derived class CheckedVector

```
...
CheckedVector(int size): Vector(size) {}
...
```

270

More Points of Interest

- CheckedVector overrides the array reference function of Vector.

```
...
int& CheckedVector :: operator[](int i)
....
```

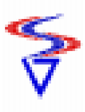
- Since CheckedVector doesn't override the size function of Vector, the following calls the inherited function

```
...
if (0 > i || i >= size())
....
```

- An overridden function can be called as follows:

```
...
return Vector :: operator[] (i);
....
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only

271

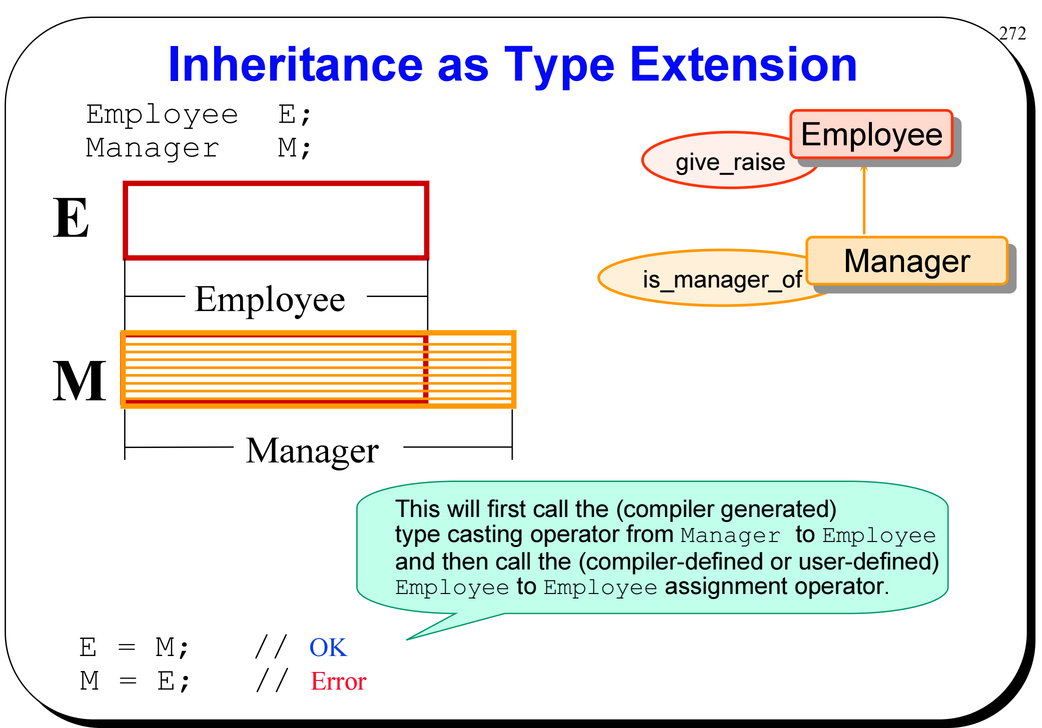
Data Hiding and Derived Classes

- `private` members of the base class are not accessible to the derived class
  - **Otherwise, privacy is completely compromised by an inherited type.**
- `public` members of the base class are accessible to anyone
- `protected` members of the base class are accessible to derived classes only

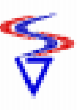
```
class Vector {
    protected:
        int len, *data;
    public:
    // ...
};
```

Protected data members make derived classes addicted

272



Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only

273

Is-A Relationship

A derived class is a (more specialized) version of the base class:

- Manager is an Employee
- CheckedVector is a Vector.
- LandVehicle is a Vehicle.

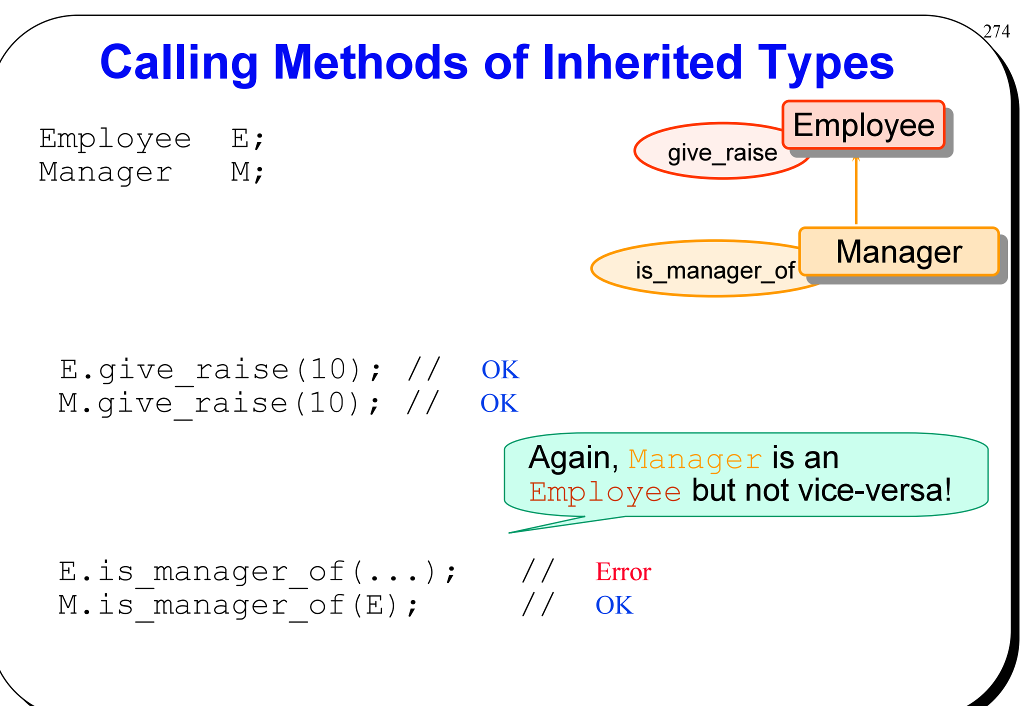
Thus, any function taking the class B as an argument, will also accept class D derived from B.

```
class Complex { ... };
Complex operator +(const Complex& c1, const
            Complex& c2) { ... }

class BoundedComplex: public Complex {
     ...
} z1, z2;
Complex c = z1 + z2;
```

The last statement calls `operator+(Complex,Complex)` using type conversion.

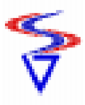
274

Calling Methods of Inherited Types

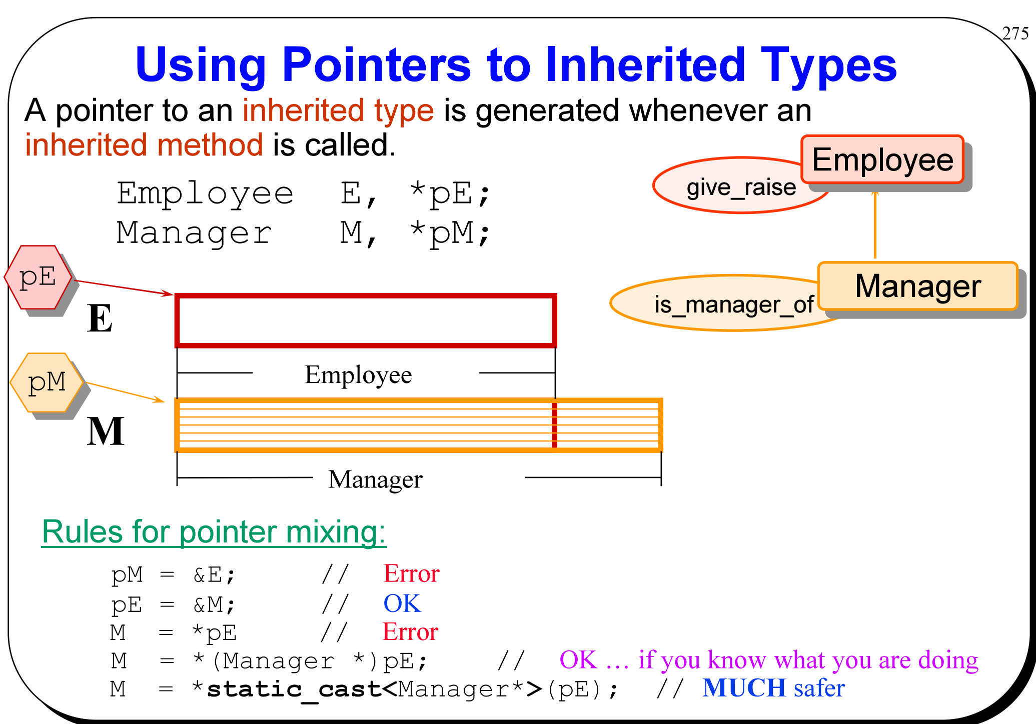


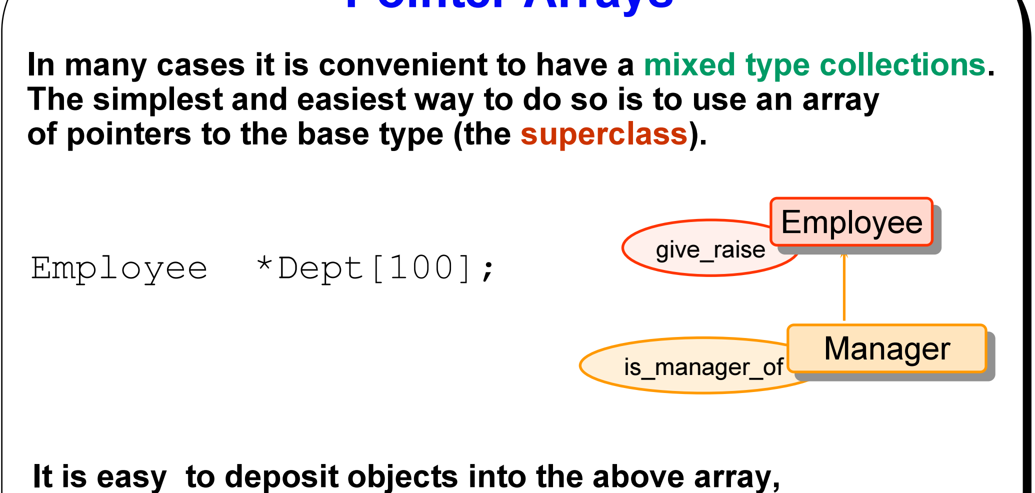

```
Employee   E;
Manager    M;

E.give_raise(10); //   OK
M.give_raise(10); //   OK

E.is_manager_of(...);    //   Error
M.is_manager_of(E);      //   OK
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

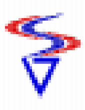


For personal use only

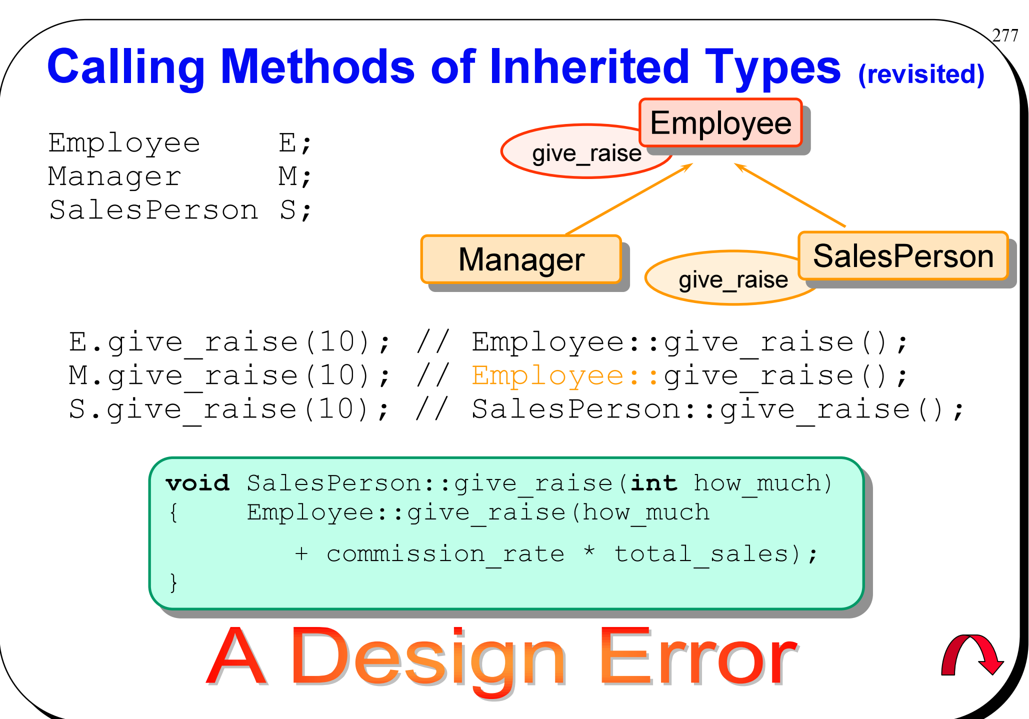


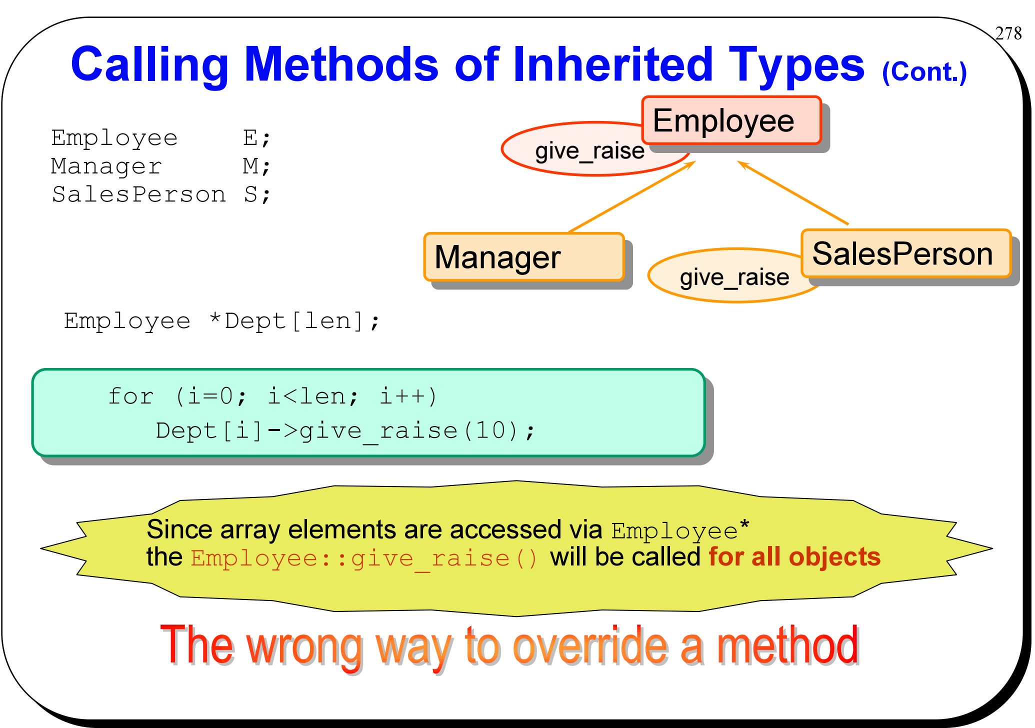


Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

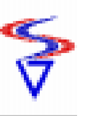


For personal use only





Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

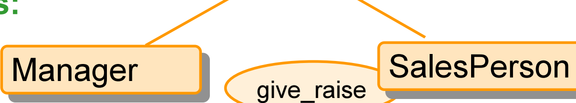


For personal use only

Determining the Object Type

Given a pointer of type `base*` (where `base` is a base class), how do we know the actual class of the object being pointed to ?

There are three solutions:



- **Ensure that only objects of a single type are always pointed to.**
- **Place a type field in the base class and update it in the constructor of the derived class.**
- **Use virtual functions.**

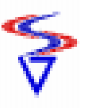
Polymorphism

Virtual functions give full control over the behavior of an object, even if it is referenced via a base class pointer (or reference).

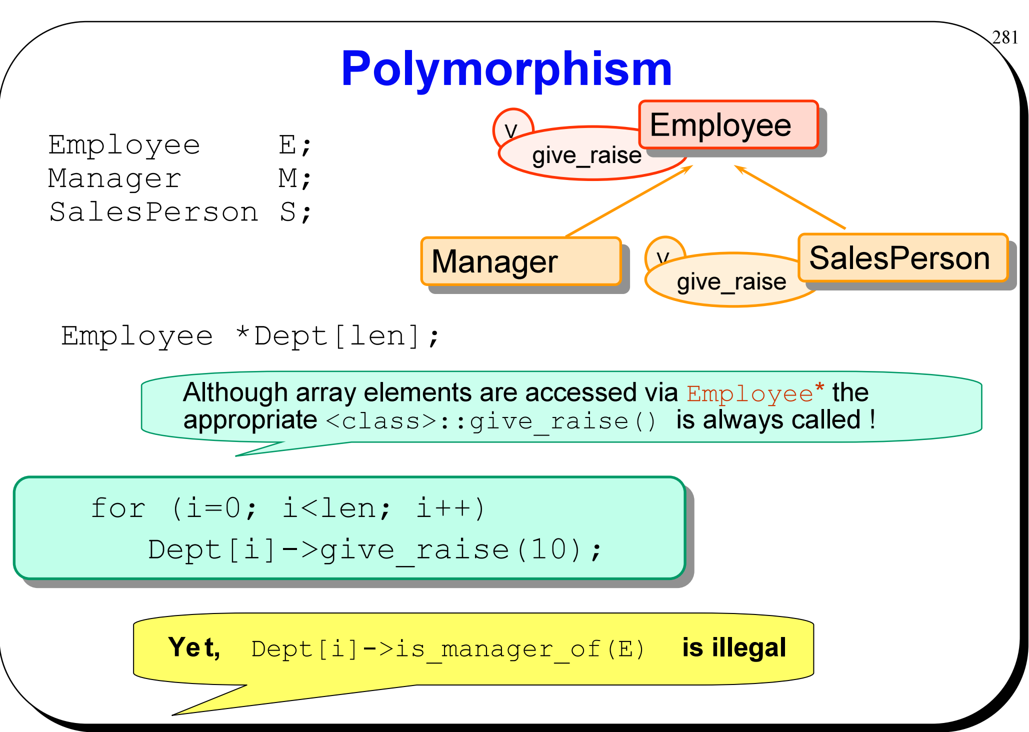
```
class Employee {
    ...
public:
    ...
    virtual void give_raise(int how_much) {...};
};
```

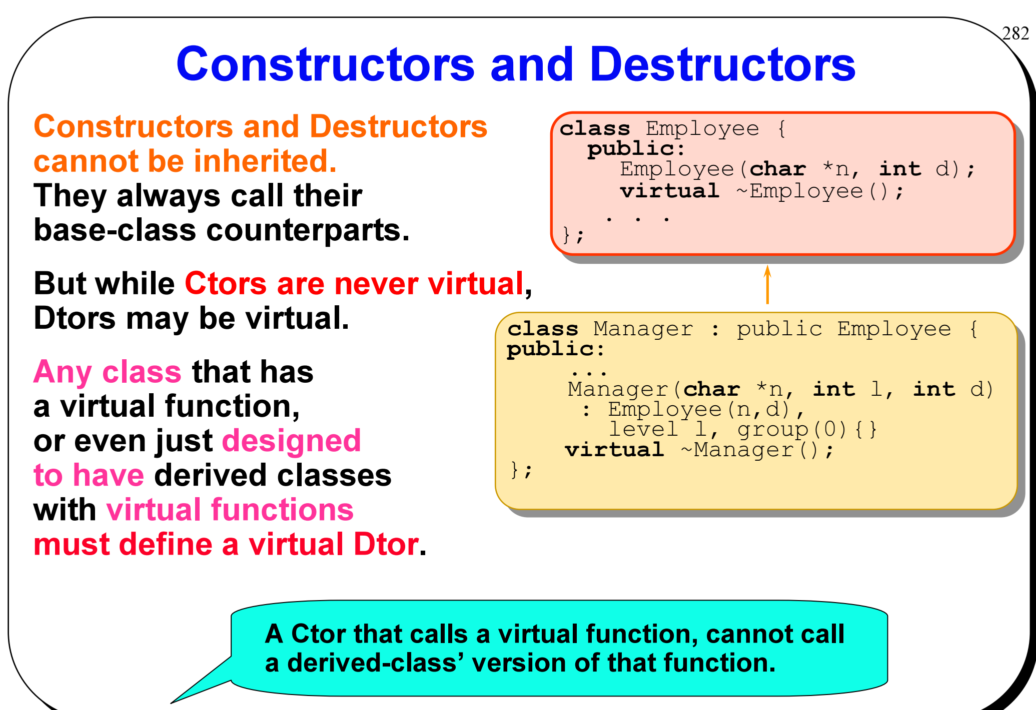
```
class SalesPerson: public Employee {
    ...
    double *commission_rate;
public:
    ...
    virtual void give_raise(int how_much) {...};
};
```

Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion

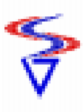


For personal use only





Copyright 1995 - 2003
C. Gotsman & Y.M. Kimchi
Computer Science Dept. Technion



For personal use only