Lab10
Inheritance

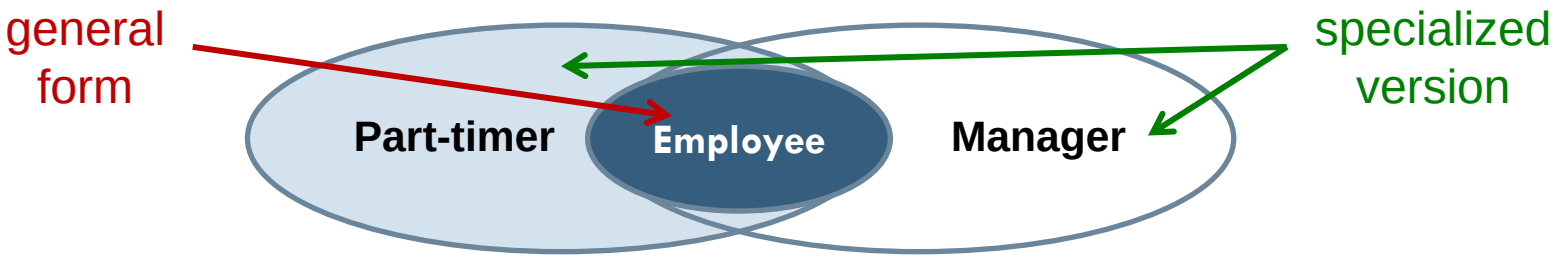
UEE 1303
Department of Electrical and Computer Engineering, Institute of Electronics
National Chiao Tung University

Outline

- Inheritance basics
  - Derived class
  - Protected qualifier
  - Redefining member functions
- Programming with Inheritance
  - Assignment operators and copy constructors
  - Destructors in derived classes
- Exercise

Derived Class (1/3)

- What's inheritance?
  - Purpose: Object-Oriented Programming
  - Provide abstraction dimension
    - Define general form of class, then specialized versions inherit properties of general class



    - Add new/modify base's functionality for it's appropriate use

Derived Class (2/3)

- Base class
  - "General" class from which others derive
- Derived class
  - New class
  - Automatically has base class's
    - Member variables
    - Member functions
  - Can add additional member functions and variables

Derived Class (3/3)

- Example
  - Base Class
    - Each **Employee** has its name and will be paid
  - An Employee may be…
    - **SalariedEmployee**
    - **HourlyEmployee**
    - **Manager**
  - Each is "subset" of employees

Base Class: Employee

- Class definition is just like what we done before
  - Example

```
class Employee {
        string first_name, family_name;
        char middle_initial;
        Date hiring_date;
        short department;
    public:
        string full_name();
    // …
};
```

Derived Classes: Manager

- Derived class interface only lists new or "to be redefined" members
  - Example:

```
class Manager: public Employee {          // public inheritance
    Employee* group[100];                 // people managed
    short level;
    // …
};
```

- Manager also has all the variables (first_name, department, …) which belong to Employee

Pointer's Conversion

- Because a Manager is an Employee, we can make the Employee's pointer point to a Manager object
  - Example (upcasting)

```
Manager m;
Employee *pe= &m;                    // "OK," public inheritance
```

- But the inverse conversion (downcasting) will cause a damage (An Employee is not a Manager certainly)

Access Controls

- The same access control rules still apply in a inheritance relationship
  - Generally, data member won't be public (packaging)
  - Use interface to access these data members
    - Example

```
void Manager::print() const {
cout<< "Name is " << full_name() << endl;
            // OK, if full_name() is a public member (interface)
}
void Manager::print() const {
cout<< "Name is " << family_name<< endl;
            // ERROR, if family_name is a private member
}
```

Belongs to Employee::

Belongs to Employee::

Constructor

- Constructor (ctor) of a derived class is responsible to call ctors for its base classes (and its own non-static data members)
  - Example

```
Manager::Manager(string& name, short dep, int lvl)
: Employee(name, dep), level(lvl) { // … }
                       // Initialize base and non-static data members
```

Belongs to Manager::

Belongs to Employee::

- Initialize base and non-static data members using their corresponding ctors
  - Default ctor for base/derived should be well defined

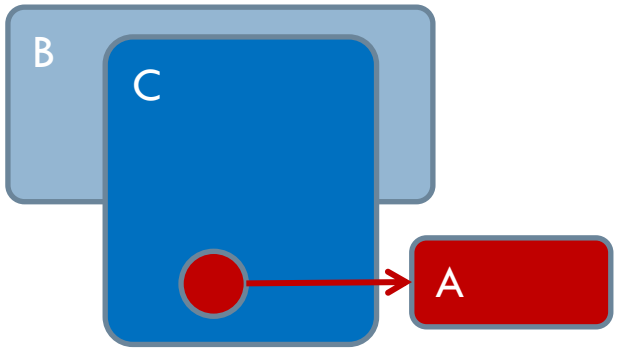
Execution Order of Constructor

```
struct A {
   A() { cout<< “ctorA” << endl; }
   ~A() { cout<< “dtorA” << endl; } };

struct B {
   B() { cout<< “ctorB” << endl; }
   ~B() { cout<< “dtorB” << endl; } };

struct C : public B {
   A a;
   C():B(), a() { cout<< “ctorC” << endl; }
   ~C() { cout<< “dtorC” << endl; } };

int main() {
   C c;
   return 0; }
```



Output:
========
ctor B
ctor A
ctor C
dtor C
dtor A
dtor B

Copy Ctor & Assignment Operator

- Copy ctors and copy assignment operators are never inherited (should be rewrited)
  - Example

```
struct C : public B {
A a; int d; int* pi;
C(int n1=0, int n2=0, int n3=0) : B(n1), a(n2), d(n3)
    pi = new int[10]; for(int i=0; i<10; ++i) pi[i] = i; }
C(const C& c) : B(c), a(c.a), d(c.d) {  // c is also of type B
    pi = new int[10]; for(int i=0; i<10; ++i) pi[i] = c.pi[i]; }
C& operator=(const C& c) {
    B::operator=(c); a = c.a; d = c.d; int* tmp= new int[10];
    for(int i=0; i< 10; ++i) tmp[i] = c.pi[i];
    delete[] pi; pi = tmp; return *this; }
~C() { delete[] pi; }
};
```

Call B's constructor

Call A and B's copy constructor

Rewrite copy ctor

Rewrite Assignment operater

Protected Members

- Protected members (data and functions)
  - Its name can be used by member functions and friends of the class only in which it is declared, and by member functions and friends of classes derived from this class

```
Class B {
        int b_priv;
    protected:
        void b_prot();
    public:
        void b_pub(); };

class D : public B {
    public:
        void d_func(); };
```

```
void D::d_func() { // D is derived from B
    b_priv= 1;      // error
    b_prot();       // ok
    b_pub();        // ok
    // … }

void func(B& b) {  // a global function
    b.b_priv= 1;    // error
    b.b_prot();     // error
    b.b_pub();      // ok
    // … }
```

Different Kind of Inheritance

- Like a member, a base class can be declared private, protected, or public
  - class X : public B { / * … */ }; // public inheritance
    - Public inheritance models is "is-a" relationship
  - class Y : protected B { / * … */ }; // protected inheritance
  - class Z : private B { /* … */ }; // private inheritance
    - Both model are "is-implemented-in-terms-of" relationship

Member in base class	Type of Inheritance		
	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	no access	no access	no access

Is-a vs. Has-a

- "Is-a" relationship is modeled by public inheritance
  - class Manager : public Employee { /* … */ };
    - It says a Manager **is an** Empoyee
- "Has-a" relationship is modeled through composition
  - Also called layering
  - class Employee {
    string first_name, family_name;
    // … };
    - It says every Employee **has a** first_name and a family_name

Lab10 Exercise

- Input: Given an employee list of a company
  - This list may contain 3 kinds of employees
    - Part-timer (P), Manager (M), and Chairman (C)
  - File format
    - Total number of employees
    - [Name] [Title] [Years of service]
- Output: Print out information of employees according to their salaries in descending order
  - Format
    - [Name] [Years of service] [Salary]

Salary Formula

- For each kind of employee, their base salary is
  - Base salary of employee (BSE): 20000/month
  - Part-timer: BSE + 1,000 * years of service
  - Manager: BSE + 15,000 + 5,000 * years of service
  - Chairman: manager’s payment + 50,000

Constraints

- Build up 3 derived classes (Parttimer, Manager, and Chairman) by yourself, and the base class Employee is given
- Parttimer and Manager is derived from Employee, and Chairman is derived from Manager

Input Example

```
10
ADAR      C     8
Terry     M     4.7
Carl      M     3.7
Shan      P     1.4
Peter     P     3.3
Sali      M     5.5
Anita     P     3.2
Mindy     P     3
Mia       P     3.8
Kate      P     3
```

Output Example

```
ADAR 8 125000
Sali 5.5 62500
Terry 4.7 58500
Carl 3.7 53500
Mia 3.8 23800
Peter 3.3 23300
Anita 3.2 23200
Mindy 3 23000
Kate 3 23000
Shan 1.4 21400
```