# Chapter 15
# More Inheritance

- Reading: pp. 869-906
- Good Problems to Work: pp. 877-878 15.2, 15.3; pp. 883-884 15.4, 15.6 C, D; pp. 895-896 15.7, 15.8

## Key Terminology

- Private, Protected, Public class members
- Derived class access of Base class members
- Inheritance
  - Constructor call order
  - Destructor call order
- Base Access Specifiers
- What derived classes inherit

## Protected Members

- Until now, we've been working with two access specifications:
  - private
  - public
- Another access specification is:
  - protected

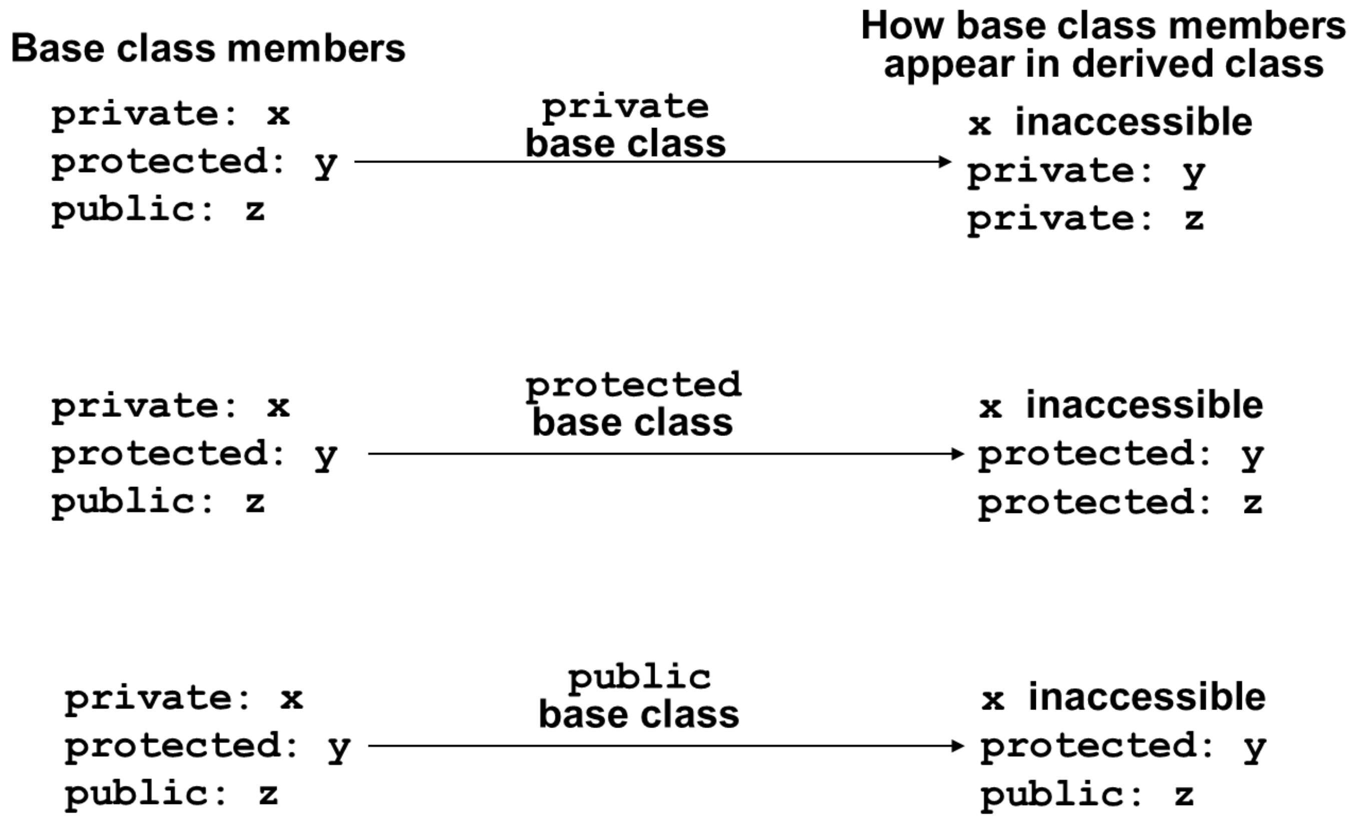
## Protected Members

- Recall from our Employee example that an Employee class contained two private members: mName, mSSN.
- HourlyEmployee was derived from Employee and thus could not directly access private Employee members
- Protected members of a class are like private members, except that derived classes may access protected members directly
- We will not use protected members in this class

# Base Access Specifications

- Recall that HourlyEmployee was publicly derived from Employee
- The base access specification is given by
  - `class HourlyEmployee: public Employee`
- We could also use private or protected
  - `class HourlyEmployee : public Employee`
  - `class HourlyEmployee : protected Employee`
  - `class HourlyEmployee : private Employee`
  - `class HourlyEmployee : Employee`
- The default access specification is private

## Base Access Specifiers



| Base class members | | How base class members appear in derived class |
|---|---|---|
| `private: x`<br>`protected: y`<br>`public: z` | `private` base class $\longrightarrow$ | `x` inaccessible<br>`private: y`<br>`private: z` |
| `private: x`<br>`protected: y`<br>`public: z` | `protected` base class $\longrightarrow$ | `x` inaccessible<br>`protected: y`<br>`protected: z` |
| `private: x`<br>`protected: y`<br>`public: z` | `public` base class $\longrightarrow$ | `x` inaccessible<br>`protected: y`<br>`public: z` |

## Derived Class Inherits?

- A derived class inherits every base class member except:
  1. any constructors
  2. destructor
  3. operator= members
  4. any friends

## Type Compatibility

- Objects of a derived class can be used wherever objects of a base class object are expected
- Rules for pointers and objects:
  - A derived class pointer can always be assigned to a base class pointer
  - A type cast is required to perform the opposite assignment
    - This could cause an ERROR!!!

## Example

```
class Base
{
  public:
    int i;
    Base(int k) {i = k;}
};
class Derived : public Base
{
  public:
    double d;
    Derived(int k, double g) : Base(k) { d = g;}
};
```

## Which are allowed?

- Base *pb = new Base (5);
- Derived *pd = new Derived (6, 10.5);
- Base *pb1 = pd;
- Base *pb2 = new Derived (7, 11.5);
- Derived *pd1 = static_cast<Derived *>(pb1);
- cout << pd1->d;
- pd = static_cast<Derived *>(pb);
- cout << pd->d;

## What is the Output?

```
class Base
{
  protected:
    int baseVar;
  public:
    Base(int val = 2) { baseVar = val; }
    int getVar() { return baseVar; }
};
class Derived : public Base
{
  private:
    int deriVar;
  public:
    Derived(int val = 100) { deriVar = val; }
    int getVar() { return deriVar; }
};
int main()
{
  Base *pObject;
  Derived object;
  pObject = & object;
  cout << pObject->getVar() << endl;
  return 0;
}
```