

**Using Constructors and Destructors in Derived Classes**

- Knowing that a derived-class inherits its base-class' members... When an object of a derived class is instantiated, the base-class' constructor must be called...
- A *base-class initializer* can be provided in the derived class constructor to call the base-class constructor explicitly...

```
Circle::Circle(double r, int a, int b) : Point(a,b) {
    ...
};
```

Otherwise base class's default constructor called implicitly...

- Base-class *constructor and base-class assignment operators* are **not inherited** by derived-class... however derived-class can call base-class constructor and assignment operator...

```
class ABase {
public:
    ABase(int a) { x = a; }
    void setX(int a) { x = a; }
    void operator =(int a) { x = a; }
    void operator --(int i) {x -= i;}
    void operator ++(int i) {x += i;}
private:
    int x;
};

class ADerived : public ABase {
public:
    ADerived(int a, int b) : ABase(b){y = a;}
    void operator ++(int i) {y += i;}
    void operator =(ABase &aB) {
        aB = y; // Called not inherited...base
    }
private:
    int y;
};

void main() {
    ABase basObj(10);
    ADerived drvObj(20,30);

    basObj = 50;
    drvObj = basObj;

    basObj +=10;
    basObj -=2;
    drvObj +=10;

    drvObj -=2;
    drvObj.setX(10);
    drvObj = 100;
}
```

Syntax Error... base-class assignment operators are not inherited by derived-class

**Implicit Derived-Class Object to Base-Class Object Conversion**

- Even though a derived-class object is *a* base-class object, the *derived-class type* and the *base-class type* are *different*...
  - *Derived-class object can be treated as a base-class object*...
    - Derived class has members corresponding to all of the base class's members... Thus, *base-class can be assigned a derived-class*...
    - Needless to say, that derived-class may have more members than the base-class object...
- *Base-class object cannot be treated as a derived-class object*...
  - Would leave additional derived class members undefined... Thus *derived-class cannot be assigned a base-class*...
  - Obviously, *assignment operator* can be overloaded to allow such an assignment...

```
#include <iostream.h>

class ABase {
public:
    ABase(int a) { x = a; }
    int getX() const {return x;}

protected:
    int x;
};

class ADerived : public ABase {
public:
    ADerived(int a, int b) : ABase(b){
        y = a;
    }
    void operator =(ABase &aB) {
        y = aB.getX();
    }
private:
    int y;
};

void main() {
    ABase basObj(10);
    ADerived drvObj(20,30);

    drvObj = basObj;
    basObj = drvObj;
}
```

Syntax Error, without the assignment operator overloaded...

- Mixing base and derived class pointers and objects
  - Referring to a base-class object with a base-class pointer → allowed...
  - Referring to a derived-class object with a derived-class pointer → allowed...
  - Referring to a derived-class object with a base-class pointer
    - allowed for base-class members... The derived-class object is an object of its base-class as well...
    - Syntax error for derived class members...
  - Referring to a base-class object with a derived-class pointer → Syntax error... The derived-class pointer must first be cast to a base-class pointer...

```
#include <iostream.h>
class ABase {
public:
    ABase(int a) { x = a; }
    void printX() const {cout << x <<endl;}
protected:
    int x;
};
class ADerived : public ABase {
public:
    ADerived(int a, int b):ABase(b){y = a;}
    void printY() const {cout << y <<endl;}
private:
    int y;
};

void main() {
    ABase *basPtr, basObj(10);
    ADerived *drvPtr, drvObj(20,30);

    basPtr = &basObj;
    basPtr ->printX();
    drvPtr = &drvObj;
    drvPtr->printY();
    drvPtr->printX();
    drvPtr->printY();

    basPtr = &drvObj;
    basPtr->printX();
    basPtr->printY();
    drvPtr = &basObj;
    drvPtr = static_cast<ADerived *>(&basObj);
    drvPtr->printX();
    drvPtr->printY();
}
```

Syntax Error...

## Problems with Multiple Inheritance

