Lecture 17

- Virtual functions cont.
- Inheritance and
  - Constructors
  - Destructors
  - Assignments

Virtual Syntax & Semantics

- Default implementation in base class:
  - **virtual <type> <func>(<params>) { ... }**
  - Signals the compiler to create a **virtual function table** and to add a **virtual function pointer** to each object that derives from this class
- **Abstract virtual function:** derived classes must provide implementation
  - **virtual <type> <func>(<params>) = 0;**
  - The presence of abstract virtual functions marks class as being abstract
  - **Abstract classes** can't be instantiated
    (e.g. **Shape x;** or **new Shape;** is illegal)

Assignments Across Class Hierarchy

- class Y : **public** X {...}
- Y inherits **data and function** members from X
- **Public inheritance**
  - **"is a"** relationship
  - public and protected X members visible in Y
- X a; Y b;
  - Assignments: a = b; or b = a; **meaningful?**
  - How to implement Y assignment operator and copy constructor?
- X *pa; Y *pb;
  - Assignments: pa = pb; or pb = pa; **meaningful?**

Object Assignment

- class Y : **public** X {...}
- X a; Y b;
- **a = b; // OK – but slicing!**
  - assignment operator is called with reference to b
  - X-parts of b are copied to a, Y parts are lost
- **b = a; // not OK**
  - Y can contain **more data** than X
  - How to fill the rest?
- Y assignment op. and copy constructor can make use of X operators (see next example)

Pointer Assignment

- class Y : **public** X {...}
- X a; Y b;
- X *pa; Y *pb;
- **pa = &b;   or   pa = pb; // OK**
  - pa now points to b respectively *pb
  - information about Y is **lost** when accessing *pa
- **pb = &a;   or   pb = pa;   // not OK**
  - *pb is object of type Y
  - again, where would the additional data come from?

Inheritance & Constructors

```
class X {
public:
   X(int a_=0) { ... }
};

class Y : public X {
public:
   Y() { /* X() is called here */ ... }
   Y(int b_) : X(b_) { ... } // explicit X(int) call
};
```

- Base class constructors, copy constructors, and assignment operators are **not inherited**!
- Derived class **constructor calls the base-class constructor first** to initialize base class members
- If ommitted, the default derived class constructor is the base class constructor

Destructors & Inheritance

```
struct X {    // struct = class ... public:
   int *p;
   X()   { p = new int[100]; }
   ~X()  { delete [] p; }
};

struct Y : public X {
   int *q;
   Y() { /*X() called here*/ q = new int[200]; }
   ~Y(){ delete [] q; /* ~X() called here*/ }
};
```

- are called in reverse order of constructor calls
- Derived class destructor ~Y() calls base class destructor ~X() at the end
- ~Y() only deals with resources allocated in Y!
  ~X() takes care of the rest

Virtual Destructors

```
class X {
public:
   X() { ... }
   /* should have been virtual! */ ~X() { ... }
   virtual void foo() { ... }
};

class Y : public X {
public:
   Y() { ... }
   ~Y() { ... }
   virtual void foo() { ... }
};

X *px = new Y; // calls Y() -which calls X() first-OK
px->foo();     // calls Y::foo() - OK
delete px;     // only calls ~X(), but not ~Y()!!!
```

- In base classes destructors must be declared virtual!

Reusing Base Class Operators

```
struct X {
   int x;
   X() { x = 0; }
};

struct Y : public X {
   int y;
   Y() { y = 0; }

   Y(const Y &a) : X(a) {  // X copy constructor, copy X-part
      y = a.y;             // copy Y-part
   }

   Y &operator=(const Y &a) {
      X::operator=(a);  // X assignment operator, copy X-part
      y = a.y; return *this;  // copy Y-part & return
   }
};
X a, *pa;
Y b;
a = b;   // a.x = b.x; b.y not copied (object “slicing”)
pa = &b; // OK, *pa is object of type X. Y-parts not visible
```

Example

```
class Shape {  // abstract base class
private:
   int color;
public:
   virtual void draw(Screen *s) const = 0;
   virtual ~Shape() {}  // needs to be virtual!
   int get_color() const { return color; }
   void set_color(int c) : color(c) {}
};

class Circle : public Shape {
private:
   int x,y,r;
public:
   Circle(int a, int b, int c) x(a),y(b),r(c) {}
   void draw(Screen *s) const { ... }
};
```

Inheritance Tips. Watch out ...

- **Declare destructors virtual** if the class may be used as a base class
- Check whether **(this == &x)** holds in assignment operators. If yes, return ***this** right away
- **Base-class copy constructors are not automatically called** in derived class copy constructors (use “: X(...) {“)
- In the derived class assignment operator call base-class X operator **explicitely**: X::operator=(source);
- **Don't call virtual functions in constructors**
  - data in derived classe not initialized yet