COPS 8

13. IV. 2011

Inheritance

Def 1: Inheritance - property that allows us to define a new class (derived class) based on the code of another existing class (base class)

Def 2. If A and B are 2 classes, we say, that B inherits from A (B is derived from A) if :

- B contains all the members from A (variables + method)
- B may redefine the method from A
- B may add new methods.

C++ - supports inheritance and multiple inheritance

class B : < access_modifier > class A { ...

}

private protected public

members \ inheritance	private	protected	public
private	private	private	private
protected	private	protected	protected
public	private	protected	public

ex: ~~class~~ Position.h

```
#ifndef POS_H
#define POS_H

    class Position {
        protected int x,y ;

        public :
            Position( int x1 = 0, int y1 = 0) : x(x1),
                                                y(y1)

      Position (const Position &p) {x = p.x; y = p.y;}

      void translate( int dx, int dy) {x += dx;
                    y += dy;}

      char * to String() { char * aux new char [10]

   sprintf(aux, "( %d, %d ), x, y);

      return aux; }

  };

 class Point : public class Position
   {   bool visible;
       public:
         Point (int x1 = 0, int y1 = 0, bool vis = false):
                 :Position (x1, y1)
            {visible = vis }
Point (const Point &p) : Position(p)
        { visible = p.visible; } // a new edit member

  char * toString ()   <- overwritting ≠ overloading
                          redefining the behaviour for the first class
      { char * temp = new char [15]
        sprintf (temp, "(%d, %d, %d), x, y,
                        visible);
        return temp;
      }
 };
```

Rules for initialization and destruction of object in inheritance

1. Constructors and destructors are never inherited, you have to explicitly call them from the base classes

```
class D : public B1, public B2 ... public Bn {
          ~
          public
                B() : B1(), B2(), ..., Bn() { ... }
```

If a class doesn't have a constructor it won't be present in the constructor initializing list.

2. the assignment operator is not inherited, you have to explicitly call the assignments operators from the base classes; they will

3. If you don't provide an assignment operator, the compiler will automaticly call the assignments operators from the base classes and they perform a bit by bit copy.

4. If you don't provide a copy constructor the compiler will automaticly call the copy constructor from the base classes and they will perform a bit by bit copy.

Compatibility between derived classes and base classes

- automatic cast conversion - an object of the derived class may be used in any situation when an object of the base ~~cla~~ class is expected.
- & - a reference to the derived class may be used in any context when a reference to the base class is expected
- * - the same for pointers

⇒ the substitution principle

- pointers to methods - from the base class may be used in a context when a pointer to a method from the derived class is expected

Declaration: <return_type> (<classname>::*
*pMethod (<lfp>)

Initialization

- pMethod = &<className>::<methodName>
  char * (Point::* pPointFun)();
- pPointFun = &Point::toString

call

$(ob.*\underbrace{pPointFun}_{Method})(<Typ>)$

Point p1: (p1 * pPointFun)();

Test.cpp

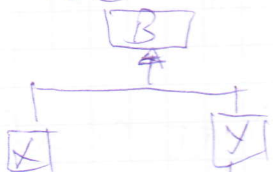
```
#include "Position.h"
#include <iostream>
using namespace std;
void print (Position p)
    { cout << p.toString() << endl; }

int main ()
{   Position p;
    Point cp;
    p = cp;
    print(cp)        //    (0,0)
    cout << cp.toString() << endl;   // (0,0,0)
    Position pos1 (5,5) * ppos;
    Point p1 (10,10,1) * ppoint;
    Point p2(p1) ;
    cout << p2.toString() << endl();
cout << pos1.toString() << endl;
ppos = &p2;
cout << ppos -> toString();
ppoint = (Point *) ppos;
cout << ppoint -> toString() << endl;
Position & pos2 = p1;
cout << pos2.toString() << endl;
char * (Position :: pfPos) ();
char * (Point :: pfPoint) ();
pfPos = & Position :: toString();
pfPoint = pfPos;
cout << (pos1. * pfPos) ();
cout << (p1. * pfPoint) ();
&Pos = pfPoint;   // error
pfPos = & Point :: toString;
pfPoint = & Point :: toString;
cout << (p1. * pfPoint) ();

 ...
}
```

Remark !!!

Automatic cast conversion between BC and DC is possible if

1) public inheritance

2) there is only one path from the DC in the BC

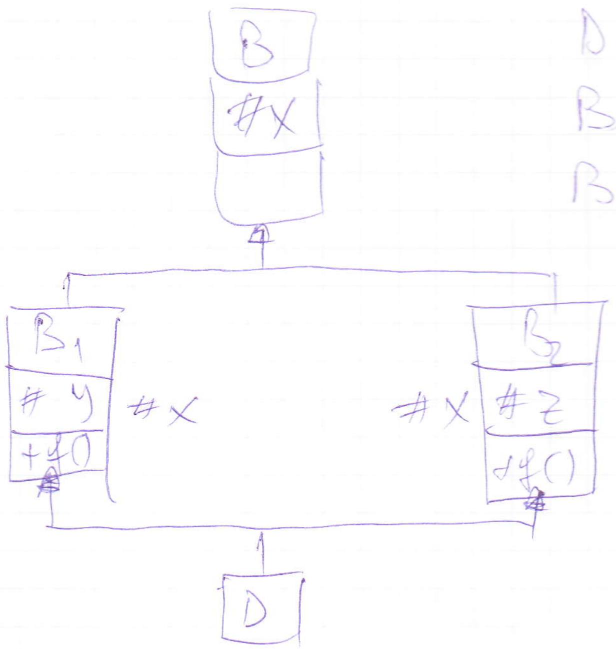


private / protected inheritance

```
class A { ... }
class B : private A { ... }
          /protected/

A a;
B b;
a = b ;   // error!
a = (A) b ;
```

Generalization / Specialization

Multiple inheritance - not recommended



D::x ?
B1::X
B2::X