CS202: Programming Systems

Week 6: Multiple inheritance
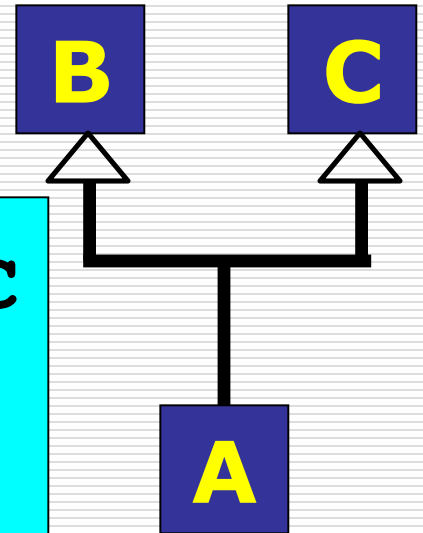
11/2016

CS202 – What will be discussed?

- ☐ Multiple inheritance
- ☐ Diamond problem
- ☐ Virtual inheritance

Multiple inheritance

☐ When a class has 2 or more direct base classes, it is called *multiple inheritance*.

☐ For example

```
class A: public B, public C
{
     ...
};
```

B   **C**

A

Multiple inheritance

- ☐ Data members and operations from B and C will be inherited to class A similarly to *single inheritance* mentioned last time.

- ☐ Virtual functions work as usual

Example

```cpp
class B {
    void draw();
};
class C {
    void cal();
};
class A: public B,
         public C
{
    void process();
};
```

```cpp
void doSth(A& a)
{
    // B::draw()
    a.draw();
    // C::cal()
    a.cal();
    // A::process()
    a.process();
}
```

Dynamic binding

```cpp
class B {
    virtual void draw() = 0;
};
class C {
    virtual void cal() = 0;
};
class A: public B, public C
{
    void draw(); //override B::draw()
    void cal(); //override C::cal()
};
```

Function name clash: ambiguity

☐ Overload resolution is not applied across different class scopes. It means function ambiguities from different base classes are not resolved based on function signatures.

`process() process()`

```
int main()
{

    A a;
    a.process();    //error:ambiguous
    a.B::process(); // OK
    a.C::process(); // OK
}
```

B   C

A
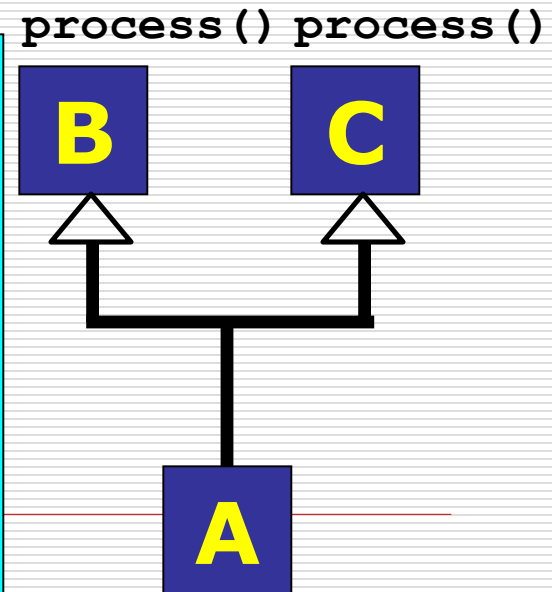
using keyword

□ If the use of the same name in different base classes is deliberately and the user would like to choose the function based on its signature

➔**using** declaration can bring the functions into a common scope.

Function name clashes!!!

```cpp
class B {
    void process(int);
};
class C {
    void process(double);
};
class A: public B, public C {...};

void doSth() {
    A a;
    a.process(10); //Error: ambiguous!
};
```

```cpp
class B {
    void process(int);
};
class C {
    void process(double);
};
class A: public B, public C {
    using A::process;
    using B::process;
    void process (char);
};


void doSth(A& a) {
    a.process(10);        // B::process(int)
    a.process('a');       // A::process(char)
    a.process(5.2);       // C::process(double)
};
```
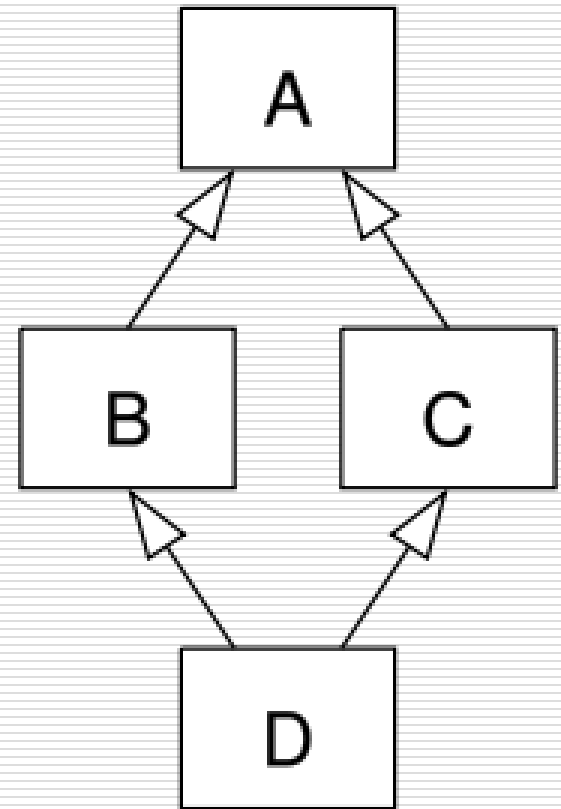
Replicated based class

☐ With the ability of specifying more than one base class, there may be a chance of having the same base class more than once.

Diamond problem!

```
class A {...};
class B: public A
{...};
class C: public A
{...};
class D: public B,
        public C
{...};
```

Replicated based class

```
void doSomething(D* p)
{
    p->process();  // error: ambiguous
    p->A::process();  // error: ambiguous
    p->B::process();  // ok
    p->C::process();  // ok
    //  ...
}
```

Virtual base class

```
class A {...};
class B: public virtual A
{...};
class C: public virtual A
{...};
class D: public B, public C
{...};
```

☐ D has only 1 **class A**