CSE 165/ENGR 140
Intro to Object Orient Program

Lecture 23 – Multiple Inheritance (2)
Announcement

¬ Lab #9
  ◦ Due 4/23 at 11:59PM
¬ Final on Thursday (5/7)
  ◦ Last lecture of semester
  ◦ Review and quiz on Tuesday (5/5)
¬ Final project is out (in UCMCROPS)
  ◦ Due date: 5/11 (Monday) at 11:59PM
  ◦ Presentation date: 5/12 (Tuesday) at 3:00PM
¬ Reading assignment:
  ◦ Ch. 3 and 6 (Vol. 2)
Multiple Inheritance

- Useful to solve cases where containers are designed to work with objects that must derive from a given base class.
- Can be achieved using Templates.
- Can add capabilities to other classes (mixing classes).
class A { int x; };

class B : public A { int y; };

class C : public B { int z; };

int main() {
    cout << "sizeof(A) == " << sizeof(A) << endl;
    cout << "sizeof(B) == " << sizeof(B) << endl;
    cout << "sizeof(C) == " << sizeof(C) << endl;
    C c;
    A* ap = &c;
    B* bp = &c;
    cout << "&c == " << &c << endl;
    cout << "ap == " << static_cast<void*>(ap) << endl;
    cout << "bp == " << static_cast<void*>(bp) << endl;
    C* cp = static_cast<C*>(bp);
    cout << "cp == " << static_cast<void*>(cp) << endl;
    cout << "bp == cp? " << boolalpha << (bp == cp) << endl;
    cp = 0;
    bp = cp;
    cout << bp << endl;
}
class A { int x; };

class B { int y; };

class C : public A, public B { int z; };

int main() {
    cout << "sizeof(A) == " << sizeof(A) << endl;
    cout << "sizeof(B) == " << sizeof(B) << endl;
    cout << "sizeof(C) == " << sizeof(C) << endl;
    C c;
    A* ap = &c;
    B* bp = &c;
    cout << "&c == " << &c << endl;
    cout << "ap == " << static_cast<void*>(ap) << endl;
    cout << "bp == " << static_cast<void*>(bp) << endl;
    C* cp = static_cast<C*>(bp);
    cout << "cp == " << static_cast<void*>(cp) << endl;
    cout << "bp == cp? " << boolalpha << (bp == cp) << endl;
    cp = 0;
    bp = cp;
    cout << bp << endl;
}
Duplicate Subobjects

class Top {
   int x;
public:
   Top(int n) { x = n; }
};

class Left : public Top {
   int y;
public:
   Left(int m, int n) : Top(m) { y = n; }
};

class Right : public Top {
   int z;
public:
   Right(int m, int n) : Top(m) { z = n; }
};

class Bottom : public Left, public Right {
   int w;
public:
   Bottom(int i, int j, int k, int m) : Left(i, j), Right(i, k) { w = m; }
};

int main() {
   Bottom b(1, 2, 3, 4);
   cout << sizeof b << endl; }
The inheritance diagram of the previous example is expected to have a "diamond shape":

Duplicate Subobjects
But what happens is a duplication of “Top”, so the diagram actually seen as this:
Virtual Base Classes

- Can we avoid duplication of a base class?
  - Can we have a “true diamond inheritance”?
    - YES, with “virtual base classes” it is possible to define the hierarchy below without duplicating the contents of the Top class!
Virtual Base Classes Ex 1/2

class Top {
public:
    int x;
    Top (int n) { x = n; }
    virtual ~Top() {} // Top has to be a Polymorphic type (a virtual method has
                     //to exist)
    friend ostream& operator<<(ostream& os, const Top& t) { return os<<t.x; }
};

class Left : virtual public Top { // note the virtual keyword in this line
public:
    int y;
    Left (int m, int n) : Top(m) { y = n; }
};

class Right : virtual public Top { // note the virtual keyword in this line
public:
    int z;
    Right (int m, int n) : Top(m) { z = n; }
};
class Bottom : public Left, public Right {
public:
    int w;

    Bottom (int i, int j, int k, int m) : 
        // Top constructor must be given here (-1 and -2 are not used): 
        Top(i), Left(-1, j), Right(-2, k) { w = m; }

friend ostream& operator<<(ostream& os, const Bottom& b) { 
    return os << b.x << ',' << b.y << ',' << b.z << ',' << b.w;
}
};

int main() {
    Bottom b(1, 2, 3, 4);
    cout << b << endl;
    b.Left::x = 100;
    // Left::x and Right::x are the same, let's verify:
    cout << b.Left::x << " : " << b.Right::x << endl;
}

Output:
1,2,3,4
100 : 100
class Top {
public:
    int x;
    Top (int n) { x = n; }
    friend ostream& operator<<(ostream& os, const Top& t) { return os<<t.x; }
};

class Left : public Top { // no virtual keyword here!
public:
    int y;
    Left (int m, int n) : Top(m) { y = n; }
};

class Right : public Top { // no virtual keyword here!
public:
    int z;
    Right (int m, int n) : Top(m) { z = n; }
};
class Bottom : public Left, public Right {
public:
    int w;

    Bottom (int i, int j, int k, int m) :
        // The 2 Top instances are constructed by Left and Right:
        Left(-1, j), Right(-2, k) { w = m; }

friend ostream& operator<<(ostream& os, const Bottom& b) {
    // The following would give error: Ambiguous access to x!
    // return os << b.x << ',' << b.y << ',' << b.z << ',' << b.w;
    return os << b.Left::x <<'':''<< b.Right::x
        << ',' << b.y << ',' << b.z << ',' << b.w;
}

int main() {
    Bottom b(1, 2, 3, 4);
    cout << b << endl;
    b.Left::x = 100; // Here we only modify "one of the x members"
    cout << b.Left::x << " : " << b.Right::x << endl;
}
Virtual Base Classes

- It is the responsibility of the most derived class to initialize the virtual base class.

- All virtual base class subobjects are initialized first, in top-down, left-to-right order.
  - Non-virtual base classes are then initialized after that.

- The scope resolution operator can always be used to disambiguate a call/access.
  - when the disambiguation is not clear, the compiler should give an error.
"Mixed hierarchies" are also possible:
Virtual Base Classes

Example of G class in the diagram below:

```cpp
class G : public E, public F
{
    M m;
    public:
    G(const string& s)
        : B("from G"), C("from G"), E("from G"),
        F("from G"), m("in G") {
        cout << "G " << s << endl;
    }
};
```
Summary

- Self-deletion in one branch of the inheritance tree works as expected.
- Pointer conversion works as expected, similarly to single inheritance.
- In normal multiple inheritance, subobjects are duplicated in “diamond inheritance cases”.
- A virtual base class removes duplication in “diamond inheritance cases”.
- The two different types of inheritance can be mixed in a hierarchy.