Наследования (продолжение)

Лекция 8

Множественное наследование

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

Спецификатор доступа распространяется только на один базовый класс; для других базовых классов начинает действовать принцип умолчания.

Множественное наследование

Класс не может появляться как непосредственно базовый дважды:

```
class C : public A, public A { ... };  – Er.!
```

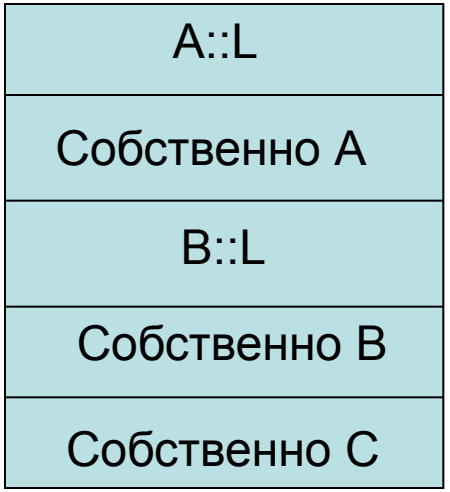
но может быть более одного раза непрямым базовым классом:

```
class L  { public:  int n; ... };

class A : public L { ... };

class B : public L { ... };

class C : public A, public B {
...
void f ();
... };
```



Здесь **решетка смежности** такая: L <-- A <-- C --> B --> L .

При этом может возникнуть неоднозначность из-за «многократного» базового класса.

Доступ к членам производного класса

void C::f () { ... n = 5; ...} // Er.! – неясно, чье n, но

void C::f () { ...A::n = 5; ...} // O.K.! , либо B::n = 5;

Имя класса в операции разрешения видимости (A или B) – это указание, в каком классе в решетке смежности искать заданное имя.

Продолжение предыдущего примера:

```
void g ( ) {
        C* pc = new C;
        L* pl = pc;          // Er.! – L  не является однозначным,
        pl = (L*) pc;        // Er.! – явное преобразование не помогает,
                              // но возможно:
        pl = (L*) (A*) pc;  // либо pl = (L*) (B*) pc; O.K.!
```

Базовый класс считается **доступным** в некоторой области видимости, если доступны его public-члены.

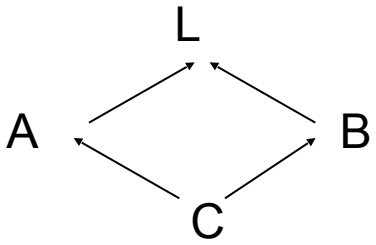
```
class B   { public:  int a; ... };
class D : private B { ... };

void g () {
        D* pd = new D;
        B* pb = pd;      // Er.! – в g() public-члены B, унаследованные
                         // D,  недоступны, такое преобразование
                         // может осуществлять только
                         // функция-член D, либо друзья D.
}
```

Виртуальные базовые классы.

```
class L   { public:  int  n ;  ... };
class A :  virtual public  L  { ... };
class B :  virtual public  L  { ... };
class C : public A, public B  { ... void f (); ... };
```

Теперь решетка смежности будет такой:



и теперь допустимо:

```
void C :: f () { ... n = 5; ...} // O.K.! – n в одном экземпляре

void g () {
        C* pc = new C;
        L* pl = pc;          // O.K.! – появилась однозначность.
}
```

Правила выбора имен в производном классе.

1 шаг: контроль **однозначности** (т.е. проверяется, определено ли анализируемое имя в одном базовом классе или в нескольких); при этом контекст не привлекается, совместное использование (в одном из базовых классов) допускается.

2 шаг: если однозначно определенное имя есть имя перегруженной функции, то пытаются **разрешить** анализируемый вызов (т.е. найти best-maching).

3 шаг: если предыдущие шаги завершились успешно, то проводится **контроль доступа**.

Неоднозначность из-за совпадающих имен в различных базовых классах.

```
class A {
  public:
     int a;
     void (*b) ( );
     void f ( );
     void g ( );  ...
};
```

```
class B {
     int a;
     void b ( );
     void h (char);
public:
     void f ( );
     int g;
     void h ( );
     void h (int);
  };
```

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

```
void gg (C* pc) {
      pc --> a = 1;       // Er.! – A::a  или  B::a
      pc --> b();  // Er.! – нет однозначности
      pc --> f ();        // Er.! – нет однозначности
      pc --> g ();        // Er.! – нет однозначности,
                          // контекст не привлекается!
      pc --> g = 1;       // Er.! – нет однозначности,
                          // контекст не привлекается!
      pc --> h ();        // O.K.!
      pc --> h (1);       // O.K.!
      pc --> h  ('a');    // Er.! – доступ в последнюю
                          // очередь
      pc --> A::a = 1;    // O.K.! – т.е. снимаем
                          //неоднозначность
                          // с помощью  операции «::»
}
```