Multiple Inheritance

pm_jat@daiict.ac.in

Text: Chapter11 and 21, Big C++

Multiple Inheritance

- Inheritance discussed so far is Single Inheritance
- If a class has only one super class, then it is Single Inheritance
- C++, also support Multiple Inheritance, i.e., when a class has more than one parent class

Multiple Inheritance

- Some of examples are-
  - **Faculty** could be Alumnus and Employee in DIICTian scenario
  - **Head-Engineering**, needs to be Manager and Engineer both
  - A **CustomerEmployee** would be Employee (a Person too), and Customer (a Person too)– forms diamond inheritance

Here is how we have multiple inheritance in C++

```
class C : public A, public B {

}
```

- In this case, C inherits from A and B both … “public”

Example: Multiple Inheritance

```
class A {
public:
   A() { x = 0; }
   A(int _x) { x = _x; }
   int getX() { return x; }
protected:
   int x;
};

class B {
public:
   B() { y = 0; }
   B(int _y) { y = _y; }
   int getY() { return y; }
protected:
   int y;
};

class C : public A, public B {
public:
   C() { z = 0; }
   C(int _x, int _y, int _z) : A(_x), B(_y) { z = _z; }
   int getZ() { return z; }
protected:
   int z;
};
```

Consider Example given

- What methods class C has?
- What is their visibility in class C?
- What data members class C has?
- What is their visibility in class C?

Example: Multiple Inheritance

```
class A {
public:
   A() { x = 0; }
   A(int _x) { x = _x; }
   int getX() { return x; }
protected:
   int x;
};

class B {
public:
   B() { y = 0; }
   B(int _y) { y = _y; }
   int getY() { return y; }
protected:
   int y;
};

class C : public A, public B {
public:
   C() { z = 0; }
   C(int _x, int _y, int _z) : A(_x), B(_y) { z = _z; }
   int getZ() { return z; }
protected:
   int z;
};
```

```
int main()
{
    C c(11,21,31);
    cout << "C:" << endl;
    cout << c.getX() << endl;
    cout << c.getY() << endl;
    cout << c.getZ() << endl;
}
```

Issues in Multiple Inheritance : Name ambiguity

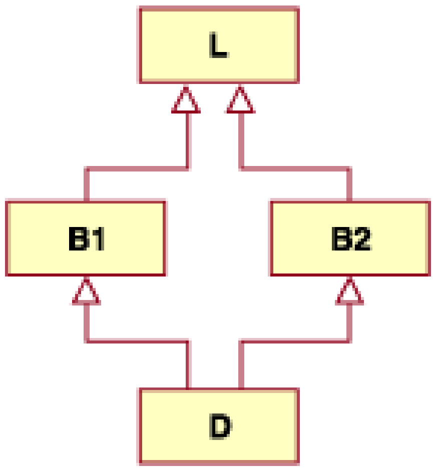
```
class A {
public:
   A() { ax = 0; }
   A(int x) { ax = x; }
   int getX() { return ax; }
protected:
   int ax;
};

class B {
public:
   B() { bx = 0; }
   B(int _bx) { bx = _bx; }
   int getX() { return bx; }
protected:
   int bx;
};

class C : public A, public B {
public:
   C() { c = 0; }
   C(int _ax, int _bx, int _cx) : A(_ax), B(_bx) { c = _cx; }
   int getC() { return c; }
protected:
   int c;
};
```

Base classes A and B of C both has getX() method

```
int main()
{
    C c(11,21,31);
    cout << "C:" << endl;
    //cout << c.getX() << endl; //has ambiguity
    cout << c.A::getX() << endl;
    cout << c.B::getX() << endl;
    cout << c.getC() << endl;
}
```

Issues in Multiple Inheritance : Diamond Inheritance

- Class B1 and B2 inherits from L, and
- D inherits from B1 and B2, both
- Therefore, in D, L inherits twice
- It brings in some issues, consider example on next slide



```
class A {
public:
    A() { ax = 0; }
    A(int x) { ax = x; }
    int getAX() { return ax; }
    int getX() { return ax; }
protected:
    int ax;
};

class B : public A {
public:
    B() { bx = 0; }
    B(int _ax, int _bx)
         { ax = _ax; bx = _bx; }
    int getBX() { return bx; }
    int getX() { return ax+bx; }
protected:
    int bx;
};
```

```
class C : public A {
public:
    C() { cx = 0; }
    C(int _ax, int _cx)
         { ax = _ax; cx = _cx; }
    int getCX() { return cx; }
    int getX() { return ax+cx; }
protected:
    int cx;
};
```

```
class D : public B, public C {
public:
   D() { dx = 0; }
   D(int _ax, int _bx, int _cx, int _dx)
        : B(_ax, _bx), C(_ax, _cx)
      { dx = _dx; }
   int getDX() { return dx; }
   int getX() { return ax+bx+cx+dx; }
private:
   int dx;
};
```

- What data and function members C has?
- What is their visibility?

Issues in Multiple Inheritance : Diamond Inheritance

- Class D has
  - two copies of data ax
  - Ambiguous method names getX(), getAX()

- Two copies of same variable should be more critical

Issues in multiple inheritance

```
class A {
public:
   A() { ax = 0; }
   A(int x) { ax = x; }
   int getAX() { return ax; }
   int getX() { return ax; }
protected:
   int ax;
};

class B : public A {
public:
   B() { bx = 0; }
   B(int _ax, int _bx)
       { ax = _ax; bx = _bx; }
   int getBX() { return bx; }
   int getX() { return ax+bx; }
protected:
   int bx;
};

class C : public A {
public:
   C() { cx = 0; }
   C(int _ax, int _cx)
       { ax = _ax; cx = _cx; }
   int getCX() { return cx; }
   int getX() { return ax+cx; }
protected:
   int cx;
};
```

```
class D : public B, public C {
public:
   D() { dx = 0; }
   D(int _ax, int _bx, int _cx, int _dx)
         : B(_ax, _bx), C(_ax, _cx)
       { dx = _dx; }
   int getDX() { return dx; }
   int getX() { return ax+bx+cx+dx; }
private:
   int dx;
};

int main()
{
    D d(11,21,31,41);
    cout << "D:" << endl;
    cout << d.getAX() << endl;
    cout << d.getBX() << endl;
    cout << d.getCX() << endl;
    cout << d.getDX() << endl;
    cout << d.getX() << endl;
}
```

Virtual Inheritance

- C++ addresses this issue by allowing such base (being inherited multiple times) class to be virtual base class
- As a result all virtual occurrences of the class throughout the class hierarchy share one actual occurrence of it.
- Here is how we modify our intermediate classes B and C declarations to inherit class A as virtual base class

Issues in multiple inheritance

```
class A {
public:
   A() { ax = 0; }
   A(int x) { ax = x; }
   int getAX() { return ax; }
   int getX() { return ax; }
protected:
   int ax;
};
class B : virtual public A {
public:
   B() { bx = 0; }
   B(int _ax, int _bx)
       { ax = _ax; bx = _bx; }
   int getBX() { return bx; }
   int getX() { return ax+bx; }
protected:
   int bx;
};
class C : virtual public A {
public:
   C() { cx = 0; }
   C(int _ax, int _cx)
       { ax = _ax; cx = _cx; }
   int getCX() { return cx; }
   int getX() { return ax+cx; }
protected:
   int cx;
};
```

```
class D : public B, public C {
public:
   D() { dx = 0; }
   D(int _ax, int _bx, int _cx, int _dx)
         : B(_ax, _bx), C(_ax, _cx)
       { dx = _dx; }
   int getDX() { return dx; }
   int getX() { return ax+bx+cx+dx; }
private:
   int dx;
};

int main()
{
    D d(11,21,31,41);
    cout << "D:" << endl;
    cout << d.getAX() << endl;
    cout << d.getBX() << endl;
    cout << d.getCX() << endl;
    cout << d.getDX() << endl;
    cout << d.getX() << endl;
}
```

Multiple Inheritance is to be avoided

- It is more complex
- Not very commonly needed, therefore should be avoided

Thanks