1. 继承成员访问 (无虚函数)

2. 虚函数与多态

1. 继承成员访问-无虚函数(1)

(1) 问题描述

用假设A、B是2个类 (A、B可以相同)

```
B  b;
A  *p = (A *)&b;   或者  A  &q = *(A *)&b
```

需要使用p或q去访问函数成员f( )和数据成员 k：

```
p->f( );   或者  q.f( ) ;
p->k;      或者  q.k;
```

1. 继承成员访问-无虚函数(2)

(2) 整体原则

f( )和 k必须在类A中是可以访问的。

(a) 在A中查找是否定义了f()和k，若定义了，则判断它们是否可以访问(如果可以访问则访问之、否则报错)；若A中没有定义f()和k，转(b) 。

(b) 在A的最近祖先类(父类)中做与(a)一样的操作，如此类推。

1. 继承成员访问-无虚函数 (3)

情形1：A和B没有继承关系

```
class A {
protected:
    int m;
    int e( ) { return 0; }
public:
    int k, n;
    int f( ) { return 1; }
    int g( ) { return 2; }
    int h( ) { return 3; }
} a1;

class B {
public:
    int m, n;
    int f( ) { return 4; }
    int g( ) { return 5; }
    int h( ) { return 6; }
    int e( ) { return 7; }
} b1;
```

```
int main( ) {
    A &a = *(A *)&b1;
    int m = a.m;    //error
    int n = a.n;    //语法正确，结果错误或崩溃
    int k = a.k;    //语法正确，结果错误
    a.f( );         //语法正确，A::f()，可能崩溃
    a.g( );         //语法正确，A::g()，可能崩溃
    a.h( );         //语法正确，A::h()，可能崩溃
    a.e( );         //error
};
```

1. 继承成员访问-无虚函数 (4)

情形2：A和B有继承关系

```
class A {
protected:
    int m;
public:
    int n, k;
    int f( ) { return 1; }
    int g( ) { return 2; }
    int h( ) { return 3; }
} a1;
```

```
class B: public A {
    int n;
    int h( ) { return 4; }
public:
    int m, i;
    int f( ) { return 5; }
} b1;
```

```
class C: public B {
    int k, j;
public:
    int m, n, p;
    int h( ) { return 6; }
    int e( ) { return 7; }
} c1;
```

```
int main( ) {
    B *b = (B *)&c1;
    int m = b->m;  //B::m
    int n = b->n;  //error
    int k = b->k;  //A::k
    int p = b->p;  //error
    b->e( );       //error
    b->f( );       //B::f( )
    b->g( );       //A::g( )
};
```

```
int main( ) {
    C *c = (C *)&b1;
    int m = c->m;  //语法正确，可能崩溃
    int n = c->i;  //B::i
    int n = c->j;  //error
    int k = c->k;  //error
    m = c->e( );   //C::e( )，可能崩溃
    m = c->f( );   //B::f( )
    n = c->g( );   //A::g( )
    k = c->h( );   //C::h( )，可能崩溃
};
```

2. 虚函数与多态 (1)

(1) 虚函数定义

用virtual定义的成员函数(虚函数必须是类的实例成员函数,即有this指针的函数)。

```
class A {
    int k;
public:
    int f( ) { return k; }
    virtual int g( ) { return 1; }
    virtual static int h( ); //error, 不是实例成员函数
    A(int k) { this->k = k; }
};
```

2. 虚函数与多态 (2)

(2) 虚函数作用

在类的继承链中，实现动态多态。

- 用基类对象指针（引用)指向类型派生类对象，通过这个基类指针(引用)去调用虚函数，就可实现动态多态（基类和派生类中定义了函数原型相同的虚函数，运行时确定调用哪一个函数)。
- 虚函数只有在继承关系时才起作用。

2. 虚函数与多态 (3)

(3) 虚函数的继承性

- 一旦基类定义了虚函数，即使没有virtual声明，所有派生类中原型相同的非静态成员函数自动成为虚函数。
- **构造函数**构造对象的类型是确定的，不需根据类型表现出多态性，故**不能定义为虚函数**。
- 析构函数可通过基类指针(引用)调用，基类指针指向的对象类型可能是不确定的，因此**析构函数可定义为虚函数**。

2. 虚函数与多态 (4)

(4) 虚函数的多态性 (1)

假定如下的继承关系：

… ← C0 ← C1 ← … Ck ← … ← Cn

即，C0是祖先类，Cn是子孙类。

有如下的语句：

```
Ck  c;
C0  *p = (C0 *)&c;
p->f( );
int  k = p->i;
```

或者：

```
C0 &q = c;
q.f( );
int  k = q.i;
```

2. 虚函数与多态 (5)

(4) 虚函数的多态性 (2)

对于语句p->f()，编译器将会做如下工作：

(a)沿着C0到祖先类的方向，查找变量f()的定义（只要发现f()的定义，就不再继续查找)，若没有找到f()或者找到f()但不能访问，则报错。否则(找到f()且f()可以访问)：判断f()是否为虚函数，若不是虚函数则直接调用f()；否则 (f()是虚函数)，转(b);

(b) f()是虚函数：沿着Ck (p实际指向的对象的类)到C0的方向，查找虚函数f()，只要发现某个类Cm (0<=m<=k)中重定义了f() (即使Cm::f()是private属性)，则调用Cm::f()。

2. 虚函数与多态 (6)

(4) 虚函数的多态性 (3)

数据成员没有虚特性 (没有多态性)

对于语句 p->i，编译器沿着C0到祖先类的方向，查找变量i的定义 (只要发现i的定义，就不再继续查找)，如果找到i的定义并且i可以访问，则访问之(成功)；否则 (没有找到i或者找到i但不能访问)，报错。

The end.