以前看到虚函数觉得很神奇，为什么就能实现多态了呢。我自己曾设想，要实现运行时多态，应该让对象的某个部分始终指向一个固定的地址，子类继承的时候，就修改这个地址的内容。这样，父类和子类都是到同一个固定地址去读取内容，在运行时就能表现不同行为。

在看了《深度探索c++对象模型》之后，发现思路是类似的。在对象中，有一个指针指向一张虚函数表，里面按照次序存放了每一个虚函数，当子类继承的时候，即到虚函数表的指定位置去修改函数地址。当我们通过父类指针来操作一个子类的时候，调用虚函数，都是通过虚函数表+固定的偏移，这样运行期多态便实现了。

在深度《深度》这本书中，虚函数表大多放在了对象的末端，不知道现在的编译器是什么样的，实际做个实验。

实验环境：VC2013 Express

class Parent {

public:

int parent;

Parent() : parent(10) {}

virtual void a() { cout << "Parent::a()" << endl; }

virtual void b() { cout << "Parent::b()" << endl; }

virtual void c() { cout << "Parent::c()" << endl; }

};

class Child : public Parent {

public:

int child;

Child() :child(100) {}

virtual void a() { cout << "Child::a()" << endl; }

virtual void b\_child() { cout << "Child::b\_child()" << endl; }

virtual void c\_child() { cout << "Child::c\_child()" << endl; }

};

class GrandChild : public Child{

public:

int grandchild;

GrandChild() :grandchild(1000) {}

virtual void a() { cout << "GrandChild::a()" << endl; }

virtual void b\_child() { cout << "GrandChild::b\_child()" << endl; }

virtual void c\_grandchild() { cout << "GrandChild::c\_grandchild()" << endl; }

};

int main()

{

typedef void(\*func)();

GrandChild grandchild;

int \*\*vtable = (int \*\*)&grandchild;

for (int i = 0; (func)vtable[0][i] != nullptr; i++)

{

auto pfunc = (func)vtable[0][i];

cout << " ["<<i<<"] ";

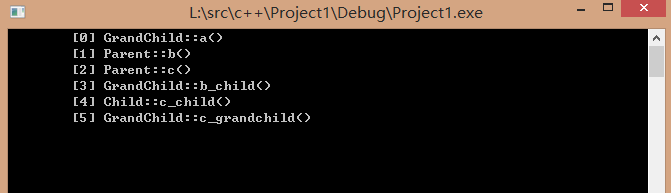
pfunc();

}

return 0;

}

结果显示



确实，虚函数表指针在对象起始处，并看到了对应项被覆盖。