1. 不能，static成员函数没有this参数，而虚函数有隐含的this参数，故不能有virtual static之类的说明。
2. 构造函数构造对象类型确定，无多态可言，不能定义为虚函数；析构函数可以通过父类指针调用，而父类指针指向的对象类型不确定，故析构函数可以定义为虚函数。
3. （1）p->h()错误，p为父类对象指针p，而h()不是其函数成员，同理cout << p->y;cout << p->z;也错误，p没有y和z这两个成员；

（2）同上，cout << q->z; 错误；

最终结果如下表

|  |  |
| --- | --- |
| p->f(); | A::f() |
| p->g(); | C::g() |
| cout << p->a; | 0 |
| cout << p->x; | 1 |
| q->f(); | C::f() |
| q->g(); | C::g() |
| q->h(); | B::h() |
| cout << p->a; | 0 |
| cout << q->x; | 11 |
| cout << q->y; | 12 |

1. class A的可访问成员及访问权限如下：

**private:**

**int a;**

**protected:**

**int b;**

**public:**

**int c;**

**~A( );**

class B的可访问成员及访问权限如下：

**private:**

**int a;**

**A::c;**

**~A( );**

**protected:**

**int b;**

**A::b;**

**public:**

**int c, d;**

class C的可访问成员及访问权限如下：

**private:**

**int a;**

**protected:**

**int b, e;**

**A::b;**

**~A( );**

**public:**

**int g;**

**A::c;**

struct D的可访问成员及访问权限如下：

**private:**

**int a;**

**protected:**

**int b, f;**

**public:**

**int e, g;**

**A::b;**

**A::c;**

**B::b;**

**B::c;**

**B::d;**

**C::b;**

**C::e;**

**C::g;**

**~A( );**

1. （1）构造函数不能是虚函数，virtual A( ) { }错误；

（2）A中有纯虚函数，不能使用抽象类的对象，class A{}a;错误，A\* p = new A;也错误；

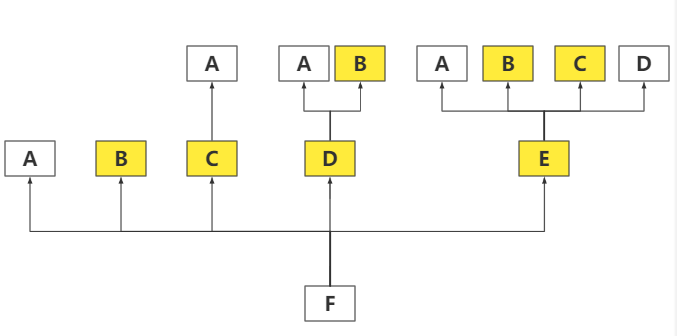
（3）long f()应改为返回值为int类型，函数的原型和基类函数名称和参数类型相同时，返回类型相同才能自动成为虚函数；

（4）B为抽象类，不能定义b，B\* p = new B;也错误；

（5）f(A,B)中A和B均为抽象类，不能为其产生实参；

（6）A为抽象类，A g(B&)不能返回A类对象。

1. 分析：对于前五行代码，只需注意先构造虚基类，再构造直接基类，接着构造数据成员，最后调用自身构造函数即可。而最后一行代码，作出派生树如下：



其中黄色方框代表虚基类，箭头指向表示由派生类指向基类，则按照派生树，上述顺序和同名虚基类（如图中的B和C）仅构造一次的要求，先构造第一层，构造B——‘B’，构造C——’AC’，构造A——‘A‘，构造D——‘AD’，构造E——‘ABAD’（B和C无需构造，先构造直接基类A，再构造数据成员D），构造C和E——直接使用上面已经得到的结论，调用自身构造函数——‘F‘。得到最终结果如下：

|  |  |
| --- | --- |
| A a; | A |
| B b; | B |
| C c; | AC |
| D d; | BAD |
| E e; | BACABADE |
| F f; | BACAADABADEACBACABADEF |