Chris Hainzl

Object-Oriented Programming (CMPS373)

Dr. D’Antonio

7 April 2023

HW 4

1. A bug in this code is that on the line A\* createA(int x = 0) { return new A(x); }, we cannot access the protected function A::A(int x) through a pointer or object. On that line, we are trying to call a protected function via creating another ‘A’ pointer, even though we are inside of the ‘B’ class, as opposed to the ‘this’ pointer. Since we are using another ‘A’ pointer and not the ‘this’ pointer, the accessibility of that other pointer is being checked. Therefore, we cannot access the protected function.

2. The output for this code is as follows:

A::f() // pa is an A pointer, so since D does not have a function f(), it calls on the function f() in class A (which D inherits).

D::h() // D inherits from A, and both classes have a function named h(). Since h() was declared as virtual in A, D::h() is called.

B::f() // pb is a B pointer, and B inherits from A. Since both A and B have a function named f() declared as void, A::f() is overridden and B::f() is called.

A::h() // There is no function in B named h(). Therefore, pb calls on A::h().

B::f() // pc is a C pointer, and C inherits from B. Since there is not a function in class C named f(), B::f() is called.

C::h() // Class C only inherits from B, which does not have a function named h(). Therefore, C::h() is called.

B::f() // pd is a D pointer, and D inherits from both C and A. Both classes D and B inherit from A, which is declared as a virtual parent. Class A has a function f() which is overridden in class B. Therefore, B::f() is called.

D::g() // Class D inherits from C and A, and only C has a function named g(). But since class D also has a function named g(), C::g() is overridden and D::g() is called.

D::h() // Classes C and A both have a function named h(). But since h() was declared virtual in class A, and declared void in both classes C and D, A::h() is ignored, C::h() is overridden, and D::h() is called.

3. a. The output for this code is as follows:

A Base is born

A Der is born

Base::func()

Der::vfunc()

A Base dies

b. No, this output does not seem correct. The way in which I would change the program so that the output is correct is by declaring both func() and the destructor in the ‘Base’ struct to be virtual. This change yields the following output, which seems to be correct:

A Base is born

A Der is born

Der::func()

Der::vfunc()

A Der dies

A Base dies