CS 3721: Programming Languages Lab
Lab #13: Multiple Inheritance and Abstract Classes vs. Interfaces

**Due date:** Today. At the end of the the lab.

Goals of this lab:

- Be able to translate programs written in Java using interfaces into C++ programs using multiple inheritance and abstract classes.

**Multiple Inheritance:** In C++, a class can be derived from more than one base class. This is known as multiple inheritance. It is a technique in which a derived class inherits the members of two or more base classes. An Example:

```cpp
class A { /* ... */ };
class B { /* ... */ };
class C { /* ... */ };

class X : public A, public B, public C
{ /* ... */ };
```

The class X is derived from the classes A, B, and C.

Java was designed without multiple inheritance. However, In Java, by making use of Java interfaces we can solve most problems that are commonly solved using multiple inheritance in C++.

**Abstract Classes vs. Interfaces:** In lab11, we had discussed pure abstract classes in C++. The following is a translation from Java interface to a pure abstract class in C++:

```java
//Java: //C++
public interface A { => class A {
    void f();
    public:
        virtual void f() = 0;
};
```

1. **Translation from Java to C++**. Today, we’re going to translate a simple Java program in which inheritance and interfaces are used into a C++ program in which multiple inheritance and abstract classes are used.

Following is the Java program that is going to be translated into a C++ program. You can find the source code on the course web page.

```java
public interface B {
    void h();
}

public interface C {
    void i();
}
```
public class A {
    public void f() {
        System.out.println("A's f()");
    }

    public void g() {
        System.out.println("A's g()");
    }
}

public class D extends A implements B, C {
    public void g() {
        System.out.println("D's g()");
    }

    public void h() {
        System.out.println("D's h()");
    }

    public void i() {
        System.out.println("D's i()");
    }

    public static void main(String[] args) {
        System.out.println("A viewed as an A");
        A a = new A();
        a.f();
        a.g();

        System.out.println("D viewed as a D");
        D d = new D();
        d.f();
        d.g();
        d.h();
        d.i();

        System.out.println("D viewed as an A");
        A da = new D();
        da.f();
        da.g();

        System.out.println("D viewed as a B");
        B db = new D();
        db.h();
    }
}
System.out.println("D viewed as a C");
C dc = new D();
dc.i();
}
}

You may want to run the program and see the output.
The following is the C++ version of the above program. You are supposed to fill in the empty slots.

#include <iostream>

using namespace std;

class B {
public:
    virtual void h() = 0;
};

// Write the corresponding abstract class C
 cigaret
 cigaret
 cigaret
 cigaret

class A {
public:
    virtual void f() {
        cout << "A's f()" << endl;
    }
    // Write the corresponding function g()
 cigaret
 cigaret
 cigaret
 cigaret

class D : cigaret , cigaret , cigaret {
public:
    void g() {
        cout << "D's g()" << endl;
    }
    // Write the corresponding method h()
 cigaret
 cigaret
 cigaret
 cigaret
}
// Write the corresponding method i()

int main()
{
    cout << "A viewed as an A" << endl;
    A *a = new A();
a->f();
a->g();

cout << "D viewed as a D" << endl;
D *d = new D();
// Call the corresponding methods through
// pointer d


cout << "D viewed as an A" << endl;
A *da = new D();
// Call the corresponding methods through
// pointer da


cout << "D viewed as a B" << endl;
B *db = new D();
// Call the corresponding method through
// pointer db


cout << "D viewed as a C" << endl;
C *dc = new D();
// Call the corresponding method through
// pointer dc

}

• What is the output of your C++ program?