NetID: thouchi2

1. What will be the output of the following program?

```
class Animal {
    public:
        void saySomething() { cout << "I don't know what to say"; }
};

class Dog : public Animal {
    public:
        virtual void saySomething() { cout << "Woof! Woof!"; }
};

int main() {
    Animal* a;
    Dog d;
    a = &d;
    a->saySomething();
}
```

A. [Correct Answer] `"I don't know what to say"`
B. `"Woof! Woof!"`
C. None of the above
D. [Your Answer] Runtime Error
E. `"I don't know what to say Woof! Woof!"`

2. Consider the following class definitions:

```
class Restaurant{
public:
    int rate() const;
private:
    double rating;
};

class Chipotle: public Restaurant {
public:
    int rateBad();
};
```

Where could the assignment `rating = 3.0;` appear for the private variable `rating`?

A. The answer to this question cannot be determined from the given code.
B. Both `rate()` and `rateBad()` can make the assignment.
C. [Correct Answer] Neither `rate()` nor `rateBad()` can make the assignment.
D. [Your Answer] `rate()` can make the assignment, but `rateBad()` cannot.
E. `rateBad()` can make the assignment, but `rate()` cannot.

3. Suppose `class modPNG` contains exactly one pure virtual function whose name is `print`. Also suppose that `class flipImage` is a `public modPNG` that implements `print`.

Which of the following C++ statements will certainly result in a compiler error? Make sure to read **all** options carefully.

A. `modPNG * a; flipImage * b; a=b;`
B. `modPNG * a = new modPNG;`
C. [Correct Answer] Exactly two of the code options will result in a compiler error.
D. `modPNG a;`
E. [Your Answer] All three of the code options will result in a compiler error.

4. What will be the output of the following program?

```
class Base {
    public:
        virtual ~Base() { cout << "Base ";  }
};

class Derived : public Base {
    public:
        Auxilliary *a2;
        Derived() { a2 = new Auxilliary(); }
        virtual ~Derived() { cout<< "Derived "; delete a2; }
};

class Auxilliary {
   public:
       ~Auxilliary() { cout << "Auxilliary "; }
};

int main() {
    Base* b = new Derived;
    delete b;
}
```

A. `"Auxilliary Derived Base "`
B. `"Base "`
C. `"Base Auxilliary Derived "`
D. [Your Answer] `"Auxilliary Derived "`
E. [Correct Answer] `"Derived Auxilliary Base "`

Loading [Contrib]/a11y/accessibility-menu.js