Object-Oriented Programming Review

Object-Oriented Programming

Object-Oriented Programming languages vary but generally all support the following features:

Data Abstraction (encapsulation) – Define types with an interface and hidden implementation.

Inheritance – Classes can inherit data or behaviour.

Polymorphism – allows you create a hierarchy of classes with a common interface.

C++ Revision

Classes have a public interface (methods only) and a private implementation containing all data members.

```
class Point {
public:
   Point(int a, int b=0): x(a), y(b) {}
   void draw() const;
private:
   int x, y;
};
```

C++ Revision

Constructors are used to create and initialise a new object. Every constructor should initalise every data member.

A constructor with 1 parameter defines an implicit conversion unless the constructor is declared explicit.

```
Point p = 3;  // Point p(3, 0);
p = 4;        // p = Point(4)
```

C++ Revision

If you don't write any constructors, the compile will supply a default constructor.

```
Point(): x(), y() {}
```

The compiler also supplies default destructor, copy constructor and assignment operator.

```
~Point() {}
Point(const Point& p): x(p.x), y(p.y) {}
Point& operator=(const Point& p) {
   x = p.x; y = p.y; return *this;
}
```

C++ Revision

C++ classes can be referred to in three different ways. Each have implications when assigning variables to each other:

```
Point p(5, 10); // Actual instance
Point *p1 = &p; // p1 is a pointer to p
Point &p2 = p;  // P2 is a reference to p

Point p3 = p;   // Copy Constructor
p3 = p;         // Assignment operator
Point *p4 = p1; // p1 and p3 both point to p
Point &p5 = p2; // (Same as Point p3 = p;)
```

C++ Revision

Methods which are defined in the class are `inline`. Functions longer than a few lines should not be defined here.

If the method has default arguments, they should only be specified once.

What is the efficiency of passing instances, references, pointers?

C++ Revision

Methods have access to private members of other objects of the same class.

A function or class can be declared as a friend.

A friend has access to private data members.

C++ Static

Static data members are shared by all objects of a class. It is initialised in a declaration outside the class.

```
class Counter {
public:
   Counter() {++count;}
   ~Counter() {--count;}
   Counter(const Counter& c) {++count;}
   Counter& operator=(const Counter& c) {return *this;}
   static int getCount() {return count;}
private:
    static int count;
};
int Counter::count = 0;
```

C++ Operators

Operators may be overloaded for classes as either a global function with one parameter for each operand or as a method of the left-hand operand.

```
Point operator+(const Point &a, const Point &b) {
   return Point(a.getX()+b.getX(), a.getY()+b.getY());
}

Point Point::operator+(const Point &b) {
   return Point(x+b.x, y+b.y);
}
```

C++ Inheritance

Inheritance allows a derived class to inherit all the members of a base class except the constructors, destructor and assignment operator.

```
class Derived: public Base { ... };
class Duck: public Bird { ... };
class Square: public Rectangle { ... };
```

C++ Inheritance

Important terms you should know:

- pubic, private, protected base classes.
- Abstract and nonabstract base classes
- Virtual and nonvirtual member functions

C++ Polymorphism

Polymorphism class hierarchies consist of an interface (abstract base class) and a set of derived implementations.

Typically the base class has no data members or code and derived classes do not add any new public methods (except constructors).

C++ Overriding Methods

A method overriding a base method must have the same parameters, return type and const-ness.

An overridden method must be virtual (in base class) if the correct version is to be called through a base pointer or reference.

C++ Programming

Generally you should not create objects of an abstract class or interface.

The abstractness of a base class can be enforced by either using protected constructors or pure (=0) virtual methods.

A class with overridden methods should have a public virtual destructor, even if it does nothing.

Example

```
class Complex{
public:
  Complex(double real, double imaginary=0):
           _real(real),_imaginary(imaginary){}
  void operator+(Complex other){
    _real = _real + other._real;
    _imaginary = _imaginary + other._imaginary;
  }
  void operator <<(ostream os){
    os <<"("<<_real<<","<<_imaginary<<")";
  }
  Complex operator++(){
    ++_real;
    return *this;
  }
  Complex operator++(int){
    Complex temp = *this;
    ++_real;
    return temp;
  }
private:
  double _real,_imaginary;
};
```

Example

```
class Complex{
public:
   explicit Complex(double real, double imaginary=0):
                    mReal(real),mImaginary(imaginary){}
   Complex& operator+=(const Complex& other){
     mReal += other.mReal;
     mImaginary += other.mImaginary;
     return *this;
   }
   Complex& operator++(){
     ++mReal;
     return *this;
   }
   const Complex operator++(int){
     Complex temp(*this);
     ++*this;
     return temp;
   }
private:
   double mReal, mImaginary;
};

ostream& Print(ostream& os, Complex a) const {
   return os <<  ...
}
```