ECE 462 Written Assignment 2

Due: 12:20PM, October 20, 2010

Name:

PUID:

1 `friend` in C++ (outcome 5)

What does `friend` mean in C++?

Answer: *allows a class to access private attributes and methods*

2 Overload << Operator (outcome 5)

Please overload the << operator to print the attributes of class `Person`.

Answer:

```
friend ostream & operator << (ostream & os, const Person & p)
```

```
#include <iostream>
#include <string>
using namespace std;

class Person
{
private:
  string p_name;
  string p_address;
public:
  Person(string n, string a)
  { p_name = n; p_address = a; }
  // >>>>>
  // overload << operator
  // <<<<<
  {
    os << "name: " << p.p_name << endl;
    os << "address: " << p.p_address << endl << endl;
    return os;
```

```
  }
};
int main(int argc, char * argv[])
{
  Person p1("John", "123 Main Street");
  Person p2("Amy", "567 First Avenue");
  Person p3("Tom", "873 North Street");
  Person p4("Mary", "952 South Second Street");
  cout << p1;
  cout << p2;
  cout << p3;
  cout << p4;
  /* output:

  name: John
  address: 123 Main Street

  name: Amy
  address: 567 First Avenue

  name: Tom
  address: 873 North Street

  name: Mary
  address: 952 South Second Street

  */
  return 0;
}
```

3 Overload – Operator (outcome 5)

Please overload the – operator to change the direction of a vector object.

Answer:

```
  Vector operator - ()
  {
    return Vector (-v_x, -v_y, -v_z);
  }
```

```
#include <iostream>
#include <string>
using namespace std;

class Vector
{
private:
```

```
  double v_x;
  double v_y;
  double v_z;
public:
  Vector(double x, double y, double z)
  { v_x = x; v_y = y; v_z = z; }
  // >>>>>
  // overload - (negation) operator
  // <<<<<
  void print()
  { cout << "(" << v_x << "," << v_y << "," << v_z << ")" << endl; }
};
int main(int argc, char * argv[])
{
  Vector v1(2, 3.1, 5.7);
  Vector v2 = -v1;
  Vector v3(-0.8, 0.4, 7.5);
  v1.print();
  v2.print();
  v3.print();
  v3 = -v1;
  v3.print();
  /* output:
     (2,3.1,5.7)
     (-2,-3.1,-5.7)
     (-0.8,0.4,7.5)
     (-2,-3.1,-5.7)
  */
  return 0;
}
```

4 Overload * Operator (outcome 5)

Please overload the * operator between a Vector object and double.

Answer:

```
Vector operator * (const Vector & v, double d)
{
  return Vector (v.getX() * d, v.getY() * d, v.getZ() * d);
}
```

```
#include <iostream>
#include <string>
using namespace std;
// do not change the Vector class
class Vector
{
```

```
private:
  double v_x;
  double v_y;
  double v_z;
public:
  // do not change the public methods
  Vector(double x, double y, double z)
  { v_x = x; v_y = y; v_z = z; }
  void print()
  { cout << "(" << v_x << "," << v_y << "," << v_z << ")" << endl; }
  double getX() const { return v_x; }
  double getY() const { return v_y; }
  double getZ() const { return v_z; }
};
// >>>>>
// overload operator * for multiplication between a Vector object
// and double. return a Vector object
// <<<<<
int main(int argc, char * argv[])
{
  Vector v1(2, 3.1, 5.7);
  Vector v2 = v1 * 0.5;
  v1.print();
  v2.print();
  /* output:
     (2,3.1,5.7)
     (1,1.55,2.85)
  */
  return 0;
}
```

5 Overload $=, <, >$ Operators (outcome 5)

What is the output of this program?

Answer:

```
no match for 'operator>' in 'p3 > p4'
```

```
#include <iostream>
#include <string>
using namespace std;

class Person
{
private:
  string p_name;
  string p_address;
public:
```

```
  Person(string n, string a)
  { p_name = n; p_address = a; }
  bool operator < (const Person & p)
  { return (p_name < p.p_name); }
  bool operator == (const Person & p)
  { return (p_name == p.p_name); }
  // operator > is not provided
};
int main(int argc, char * argv[])
{
  Person p1("John", "123 Main Street");
  Person p2("Amy", "567 First Avenue");
  Person p3("Tom", "873 North Street");
  Person p4("Mary", "952 South Second Street");
  if ((p1 < p2) &&
      (p3 > p4))
    { cout << "team 1 won" << endl; }
  return 0;
}
```

6 Overload = Operator (outcome 5)

Please overload the = (assignment) operator for `Vector`.

Answer:

```
  Vector & operator = (const Vector & v)
  {
    if (this != & v)
      {
        delete [] v_element;
        v_size = v.v_size;
        v_element = new double[v_size];
        for (int ecnt = 0; ecnt < v_size; ecnt ++)
          { v_element[ecnt] = v.v_element[ecnt]; }
      }
    return * this;
  }
```

```
#include <iostream>
#include <string>
using namespace std;

class Vector
{
private:
  double * v_element;
  int v_size;
```

```
public:
  Vector(int s, double * elem)
  {
    v_size = s;
    v_element = new double[s];
    for (int ecnt = 0; ecnt < s; ecnt ++)
      { v_element[ecnt] = elem[ecnt]; }
  }
  // copy constructor
  Vector(const Vector & v)
  {
    v_size = v.v_size;
    v_element = new double[v_size];
    for (int ecnt = 0; ecnt < v_size; ecnt ++)
      { v_element[ecnt] = v.v_element[ecnt]; }
  }
  // >>>>>
  // overload = (assignment) operator
  // <<<<<
  void print()
  {
    for (int ecnt = 0; ecnt < v_size; ecnt ++)
      { cout << v_element[ecnt] << " "; }
    cout << endl;
  }
};
int main(int argc, char * argv[])
{
  double elem1[] = {1.1, 2.3, -5.3, 0.7, 9.2, 0.05, 3.3};
  double elem2[] = {2.1, 1.3, 4.3, 0.07, 5.2, 0.95, 2.3, 7.4, 8.3};
  int size1 = sizeof(elem1) / sizeof(double);
  int size2 = sizeof(elem2) / sizeof(double);
  Vector v1(size1, elem1);
  Vector v2 = v1;
  Vector v3(size2, elem2);
  v1 = v3;
  v1.print();
  v2.print();
  v3.print();
  /* output:
     2.1 1.3 4.3 0.07 5.2 0.95 2.3 7.4 8.3
     1.1 2.3 -5.3 0.7 9.2 0.05 3.3
     2.1 1.3 4.3 0.07 5.2 0.95 2.3 7.4 8.3
  */
  return 0;
}
```

7 Overload Postfix ++ Operator (outcome 5)

Please overload the postfix ++ operator for `Vector`.

Answer:

```
const Vector Vector::operator ++ (int)
{
  Vector oldV = * this;
  v_x ++;
  v_y ++;
  v_z ++;
  return oldV;
}
```

```
#include <iostream>
#include <string>
using namespace std;
class Vector
{
private:
  int v_x;
  int v_y;
  int v_z;
public:
  Vector(int x, int y, int z)
  { v_x = x; v_y = y; v_z = z; }
  // >>>>>
  // postfix ++
  // <<<<<
  Vector & Vector::operator ++ ()   // prefix ++
  {
    v_x ++;
    v_y ++;
    v_z ++;
    return * this;
  }
  friend ostream & operator << (ostream & os, const Vector & v)
  {
    os << "(" << v.v_x << "," << v.v_y << "," << v.v_z << ")" << endl;
    return os;
  }
};
int main(int argc, char * argv[])
{
  Vector v1(2, 3, 5);
  cout << v1 << endl;
  cout << v1 ++ << endl;
  cout << ++ v1 << endl;
  /* output:
     (2,3,5)
```

```
     (2,3,5) <---- same as the previous one
     (4,5,7)
  */
  return 0;
}
```

8 Overloading and Overriding (outcome 6)

Which statement is correct?

Answer: C

A. In Java, if a function is overloaded, it cannot be overridden in a derived class.

B. Overloaded functions in C++ must use primitive types; objects cannot be used as parameters in overloaded functions.

C. In C++, if a function is overloaded, it can still be overridden in a derived class.

D. In Java, overloaded functions can be distinguished by both the argument types and the return types.

E. In Java, overloaded functions cannot use objects as parameters.

9 Virtual and Overloaded Function in C++ (outcome 6)

What is the output of this program?

Answer:

16 = 10 Derived2::f1(int) + 1 Base::f1 + 5 Derived2::f2

```
#include <iostream>
using namespace std;
int gvalue = 0;
class Base
{
public:
  void f1() { gvalue += 1; } // <--- not virtual
  virtual void f2() { gvalue += 2; }
};

class Derived1: public Base
{
public:
  void f1() { gvalue += 3; }
  void f2() { gvalue += 4; }
};
```

```
class Derived2: public Derived1
{
public:
  void f1(int v) { gvalue += v; }
  void f2() { gvalue += 5; }
};

int main(int argc, char * argv[])
{
  Derived2 dobj;
  dobj.f1(10);
  Base * bobj = new Derived2;
  bobj -> f1();
  bobj -> f2();
  cout << gvalue << endl;
  return 0;
}
//
```

10 Overload and Override in Java (outcome 6)

What is the output of this program?

Answer:

18 = 10 Derived2.f1(int) + 3 Derived1.f1 + 5 Derived2.f2

```
class Base
{
    public static int gvalue = 0;
    public void f1() { gvalue += 1; }
    public void f2() { gvalue += 2; }
}

class Derived1 extends Base
{
    public void f1() { gvalue += 3; }
    public void f2() { gvalue += 4; }
}

class Derived2 extends Derived1
{
    public void f1(int v) { gvalue += v; }
    public void f2() { gvalue += 5; }
}

class outcome731 {
    public static void main( String[] args ) {
```

```
        Derived2 dobj = new Derived2();
        dobj.f1(10);
        Base bobj = new Derived2();
        bobj.f1();
        bobj.f2();
        System.out.println(bobj.gvalue);
    }
}
//
```

11 Overload using Return Types (outcome 6)

What is the output of this program?

Answer:

```
outcome741.java:9: f1() in Derived1 cannot override f1() in Base;
attempting to use incompatible return type

found   : java.lang.String
required: int
    public String f1() { return "Derived1"; }
                  ^
outcome741.java:10: f2() in Derived1 cannot override f2() in Base;
attempting to use incompatible return type
found   : int
required: java.lang.String
    public int f2() { return 4; }
               ^
outcome741.java:15: f1() in Derived2 cannot override f1() in Derived1;
attempting to use incompatible return type
found   : void
required: java.lang.String
    public void f1() { System.out.println("Derived2.f1"); }
                ^
outcome741.java:16: f2() in Derived2 cannot override f2() in Derived1;
attempting to use incompatible return type
found   : void
required: int
    public void f2() { System.out.println("Derived2.f2"); }
                ^
outcome741.java:22: 'void' type not allowed here
        System.out.println(dobj.f1());
                                   ^
outcome741.java:23: 'void' type not allowed here
        System.out.println(dobj.f2());
                                   ^
6 errors
```

```
class Base
{
    public int f1() { return 1; }
    public String f2() { return "Base"; }
}

class Derived1 extends Base
{
    public String f1() { return "Derived1"; }
    public int f2() { return 4; }
}

class Derived2 extends Derived1
{
    public void f1() { System.out.println("Derived2.f1"); }
    public void f2() { System.out.println("Derived2.f2"); }
}

class outcome731 {
    public static void main( String[] args ) {
        Derived2 dobj = new Derived2();
        System.out.println(dobj.f1());
        System.out.println(dobj.f2());
    }
}
//
```

12 Overload Resolution (outcome 6)

What is the output of this program?

Answer:

```
Y1(XB)
Y1(XB)
YB(XD2)
Y1(XB)
Y1(XB)
YB(XD2)
Y2(XB)
Y2(XB)
Y2(XD1)
Y1(int)
```

```
#include <iostream>
using namespace std;
class XBase
{
```

```
};

class XDerived1: public XBase
{
};

class XDerived2: public XDerived1
{
};

class YBase
{
public:
  virtual void f1()                       { cout << "YB()" << endl; }
  virtual void f1(double v)               { cout << "YB(double)" << endl; }
  virtual void f1(string s)               { cout << "YB(string)" << endl; }
  virtual void f1(const XBase * xo)       { cout << "YB(XB)" << endl; }
  virtual void f1(const XDerived2 * xd2) { cout << "YB(XD2)" << endl; }
};

class YDerived1: public YBase
{
public:
  virtual void f1(int v)                  { cout << "Y1(int)" << endl; }
  virtual void f1(const XBase * xo)       { cout << "Y1(XB)" << endl; }
  virtual void f1(const XDerived1 * xd1) { cout << "Y1(XD1)" << endl; }
};

class YDerived2: public YDerived1
{
public:
  virtual void f1(const XBase * xo)       { cout << "Y2(XB)" << endl; }
  virtual void f1(const XDerived1 * xd1) { cout << "Y2(XD1)" << endl; }
  virtual void f1(const XDerived2 * xd2) { cout << "Y2(XD2)" << endl; }
};

int main(int argc, char * argv[])
{
  XBase * xobb = new XDerived2;
  XBase * xobd1 = new XBase;
  XDerived2 * xobd2 = new XDerived2;

  YBase * yobb = new YDerived1;
  YBase * yobd1 = new YDerived1;
  YDerived1 * yobd2 = new YDerived2;

  yobb -> f1(xobb);
  yobb -> f1(xobd1);
  yobb -> f1(xobd2);

  delete xobd1;
```

```
  xobd1 = new XDerived1;
  yobd1 -> f1(xobb);
  yobd1 -> f1(xobd1);
  yobd1 -> f1(xobd2);

  delete xobb;
  xobb = new XBase;

  yobd2 -> f1(xobb);
  yobd2 -> f1(xobd1);
  yobd2 -> f1(xobd2);

  yobd2 -> f1(2);
  // do not worry about delete here
  return 0;
}
//
```

13 Overloading with Promotion (outcome 6)

What is the output of this program?

Answer:

```
Bd
D2s
D1i
D1i
D2s
```

```
class Base
{
    public void f1(double v) { System.out.println("Bd"); }
}

class Derived1 extends Base
{
    public void f1(String v) { System.out.println("D1s"); }
    public void f1(char v)   { System.out.println("D1c"); }
    public void f1(int v)    { System.out.println("D1i"); }
}

class Derived2 extends Derived1
{
    public void f1(double v) { System.out.println("D2d"); }
    public void f1(String v) { System.out.println("D2s"); }
}
```

```
class outcome761 {
    public static void main( String[] args ) {
        Base bd1obj = new Derived1();
        bd1obj.f1(3.14159);

        Derived2 d2d2obj = new Derived2();
        d2d2obj.f1("ece462");
        d2d2obj.f1(10);

        Derived1 d1d2obj = new Derived2();

        d1d2obj.f1(1110);
        d1d2obj.f1("ece462");
    }
}
//
```

14 Virtual Function in C++ (outcome 6)

What is the output of this program?

Answer:

```
B()
D1(int)
B(string)
B(double)
D1(int)
```

```
#include <iostream>
using namespace std;

class Base
{
public:
  void f1()                 { cout << "B()" << endl; } // <--- not virtual
  void f1(string s)         { cout << "B(string)" << endl; }
  virtual void f1(int v)    { cout << "B(int)" << endl; }
  virtual void f1(double v) { cout << "B(double)" << endl; }
};

class Derived1: public Base
{
public:
  void f1()                 { cout << "D1()" << endl; }
  virtual void f1(int v)    { cout << "D1(int)" << endl; }
};
```

```
class Derived2: public Derived1
{
public:
  void f1(string s)         { cout << "D2(string)" << endl; }
  virtual void f1(int v)    { cout << "D2(int)" << endl; }
};

int main(int argc, char * argv[])
{

  Base * obb = new Derived1;
  Base * obd1 = new Derived1;
  Base * obd2 = new Derived2;

  obb  -> f1();
  obd1 -> f1(3);
  obd2 -> f1("ece462");
  obd1 -> f1(46.2);
  obb  -> f1('c');
  // do not worry about delete here
  return 0;
}
//
```

15 Overloading in Java (outcome 6)

What is the output of this program?

Answer:

```
473 = 1 + 16 + 64 + 128 + 8 + 32 + 64 + 128 + 32
```

```
class Base
{
    protected static int val = 0; // <--- static
    public void f1()         { val += 1; }
    public void f1(int v)    { val += 2; }
    public void f1(double v) { val += 4; }
    public void f1(String s) { val += 8; }
    public int getVal() { return val; }
}

class Derived1 extends Base
{
    public void f1(int v)    { val += 16; }
    public void f1(double v) { f1("fall-2008"); val += 32; }
}

class Derived2 extends Derived1
```

```
{
    public void f1()         { val += 64; }
    public void f1(String s) { f1(); val += 128; }
}

class outcome781 {
    public static void main( String[] args ) {
        Base obb = new Base();
        Base obd1 = new Derived1();
        Base obd2 = new Derived2();

        obb.f1();
        obd1.f1(3);
        obd2.f1("ece462");
        obd1.f1(26.4);
        obd2.f1(12.9);
        System.out.println(obb.getVal());
    }
}
//
```