ECE 462 Exam 2

08:10-09:20AM, October 30, 2009

I certify that I will not receive nor provide aid to any other student for this exam.

Signature:

You must sign here. Otherwise, the exam is not graded.

Individual programming assignments (IPAs) are due on November 20. Outcomes 1 and 2 are evaluated by IPAs.

This exam is printed **double sides**.

Write your answers next to the questions. If you need more space, you can use the two blank pages.

This is an *open-book, open-note* exam. You can use any book or note or program printouts.

Please turn off your cellular phone **now** .

Two outcomes are tested in this exam. To pass each outcome, you must receive 50% of the points.

- Outcome 5: an understanding of the difference between function overloading and function overriding.
- Outcome 6: an ability to overload operators in C++.

There are 10 questions in this exam. If a program has a syntax error, **point out** which line causes the error. If there are multiple errors, you need to write only one of them.

If a question has a numeric answer, you can write the *procedure* without the final result. For example, you can write "1 + 2" instead of "3".

Contents

Total Score:
Passed Outcomes:

This page is blank. You can write answers here.

This page is blank. You can write answers here.

1 Java Overloading (outcome 5, 1 point)

Write down the **sequence** of the called functions. Please remember that in a squence, the **order** is important. If class `X`'s function `f` is called, write `X's f`. If `f` is overloaded, also write the arguments, for example `X's f(int)`.

Explain your answers.

Answer:

B's f1()

D1's f1(int)

D2's f1(String)

D1's f1(double)

D2's f1(String)

```
class Base
{
    public void f1()         { }
    public void f1(int v)    { }
    public void f1(double v) { }
    public void f1(String s) { }
}

class Derived1 extends Base
{
    public void f1(int v)    { }
    public void f1(double v) { f1("fall-2008"); } // <--- ATTENTION
}

class Derived2 extends Derived1
{
    public void f1()         { }
    public void f1(String s) { }
}

class outcome51 {
    public static void main( String[] args ) {
        Base obb = new Base();
        Base obd1 = new Derived1();
        Derived1 obd2 = new Derived2();

        obb.f1();
```

```
        obd1.f1(3);
        obd2.f1("ece462");
        obd2.f1(12.9);
    }
}
```

2 C++ Overloading (outcome 5, 1 point)

Write down the **sequence** of the called functions. Please remember that in a squence, the **order** is important. If class `X`'s function `f` is called, write `X's f`. If `f` is overloaded, also write the arguments, for example `X's f(int)`.

Explain your answers.

Answer:

Y1(XB)

Y1(XB)

Y2(XB)

Y2(XB)

Y2(XD1)

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

class XDerived1: public XBase
{
};

class XDerived2: public XDerived1
{
};

class YBase
{
public:
  virtual void f1()                    { }
```

```
  virtual void f1(double v)              { }
  virtual void f1(string s)              { }
  virtual void f1(const XBase * xo)      { }
  virtual void f1(const XDerived2 * xd2) { }
};

class YDerived1: public YBase
{
public:
  virtual void f1(int v)                 { }
  virtual void f1(const XBase * xo)      { }
  virtual void f1(const XDerived1 * xd1) { }
};

class YDerived2: public YDerived1
{
public:
  virtual void f1(const XBase * xo)      { }
  virtual void f1(const XDerived1 * xd1) { }
  virtual void f1(const XDerived2 * xd2) { }
};

int main(int argc, char * argv[])
{
  XBase * xobb = new XBase;
  XBase * xobd2 = new XDerived2;
  XDerived1 * xod1d2 = new XDerived2;

  YBase * yobd1 = new YDerived1;
  YBase * yobd2 = new YDerived2;
  YDerived1 * yod1d2 = new YDerived2;

  yobd1 -> f1(xobb);
  yobd1 -> f1(xobd2);
  yobd2 -> f1(xobd2);
  yobd2 -> f1(xod1d2);
  yod1d2 -> f1(xod1d2);

  delete xobb;
  delete xobd2;
  delete xod1d2;
  delete yobd1;
  delete yobd2;
  delete yod1d2;
```

```
  return 0;
}
```

3 Java Overloading (outcome 5, 1 point)

Write down the **sequence** of the called functions. Please remember that in a squence, the **order** is important. If class `X`'s function `f` is called, write `X's f`. If `f` is overloaded, also write the arguments, for example `X's f(int)`.

Explain your answers.

Answer:

syntax error: bd1.f1("ece462")

```
class Base
{
    public void f1(double v) { }
}

class Derived1 extends Base
{
    public void f1(String v) { }
    public void f1(char v)   { }
    public void f1(int v)    { }
}

class Derived2 extends Derived1
{
    public void f1(double v) { }
    public void f1(String v) { }
}

class outcome53 {
    public static void main( String[] args ) {
        Base bd2 = new Derived2();
        Base bd1 = new Derived1();
        Derived1 d1d1 = new Derived1();
        Derived1 d1d2 = new Derived2();

        bd2.f1(3.14159);
        bd1.f1("ece462");
        d1d1.f1(2009);
        d1d2.f1("purdue");
```

```
        d1d2.f1('C');
    }
}
```

4 C++ Virtual Function (outcome 5, 1 point)

Write down the **sequence** of the called functions. Please remember that in a squence, the **order** is important. If class `X`'s function `f` is called, write `X's f`. If `f` is overloaded, also write the arguments, for example `X's f(int)`.

Explain your answers.

Answer:

B()

B(string)

D1(int)

D1()

D2(string)

```
#include <iostream>
using namespace std;
/*
 *
 * ATTENDION: Some functions are virtual; some are not.
 *
 */
class Base
{
public:
  void f1()                 { }
  void f1(string s)         { }
  virtual void f1(int v)    { }
};

class Derived1: public Base
{
public:
  virtual void f1()         { }
  virtual void f1(int v)    { }
```

```
};

class Derived2: public Derived1
{
public:
  void f1(string s)         { }
  virtual void f1(int v)    { }
};

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

  Base * bd1 = new Derived1;
  Base * bd2 = new Derived2;
  Derived1 * d1d1 = new Derived1;
  Derived1 * d1d2 = new Derived2;
  Derived2 * d2d2 = new Derived2;

  bd1 -> f1();
  bd2 -> f1("ece");
  d1d1 -> f1(462);
  d1d2 -> f1();
  d2d2 -> f1("purdue");

  delete bd1;
  delete bd2;
  delete d1d1;
  delete d1d2;
  delete d2d2;

  return 0;
}
```

5 Overload << Operator (outcome 6, 1 point)

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

Answer:

```
friend ostream & operator << (ostream & os, const Person & p)
{
  os << "name: " << p.name << endl;
  os << "address: " << p.address << endl << endl;
  return os;
```

```
  }
```

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

class Person
{
private:
  string name;
  string address;
public:
  Person(string n, string a)
  { name = n; address = a; }
  // >>>>>
  // overload << operator
  // <<<<<
};
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;
  return 0;
}
```

6 Overload - Operator (outcome 6, 1 point)

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;
}
```

7 Overload = Operator (outcome 6, 1 point)

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;
  }
  virtual ~Vector()
```

```
  {
    delete [] v_element;
  }
};
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;
}
```

8 Overload Prefix ++ Operator (outcome 6, 1 point)

Please overload the **prefix** ++ operator for `Vector`.

Answer:

```
Vector & operator ++ ()
{
  v_x ++;
  v_y ++;
  v_z ++;
  return * this;
}
```

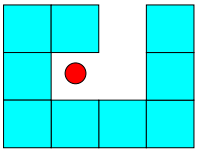
```
#include <iostream>
#include <string>
using namespace std;
class Vector
{
```

```
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; }
  // >>>>>
  // prefix ++
  // <<<<<
  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;
  return 0;
}
```

9 Piece Generation, 1 point

(This question was announced in Blackboard on October 28.)

GPA1 requires that you generate all possible pieces without holes. A hole is defined as an empty square that is enclosed by filled squares in all four directions. An example is shown here; the empty square marked by the circle is enclosed in all four directions.

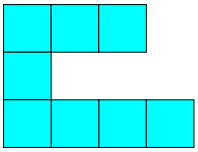


One solution to generate the pieces is by *partitioning* an integer into the sum of smaller integers and using each integer as the number of squares for a row. The following is an algorithm to generate pieces:

```
  Algorithm: to generate n-square pieces without holes
```

```
step 1
     partition n into sum of positive integers.
     for example, 7 = 4 + 3 = 2 + 2 + 3 = 3 + 2 + 1 + 1 = ...

step 2
     Each number represents the number of squares in the same row
     4 + 3 means there are two rows; the first row has 4 squares;
     the second row has 3 squares

     3 + 2 + 1  + 1 means there are four rows, with 3, 2, 1, and 1
     squares in the rows

step 3
     shift the rows
     4 + 3 can have different relative positions, such as
     SSSS        SSSS          SSSS
     SSS        SSS          SSS

     3 + 2 + 2 can have several relative positions, such as
     SSS         SSS         SSS
     SS         SS          SS
     SS         SS         SS

     The first row is shifted to the maximum amount. This amount is
     determined by the following rows.  If the next row has r squares,
     the first row may be shifted up to r - 1 squares.

step 4
     generate all possible pieces by shifting the rows

step 5
     eliminate duplicates
```

As you already know from GPA1, the number of pieces grows rapidly as the number of squares increases. To generate pieces with many squares, we are going to restrict the number of squares in each row to be 3, 4, 5, or 6. The following is an **invalid** piece because the second row has only one square.



Explain how to compute the **number** of partitions for n as sums of only 3, 4, 5, or 6. For example, 12 has **five** partitions: 12 = 3 + 3 + 3 + 3 = 4 + 4 + 4 = 3 + 4 + 5 = 3 + 3 + 6 = 6 + 6.

Choose ONLY ONE of the two definitions: (1) Different orders are considered as different partitions. For example,

```
3 + 4 + 5
4 + 5 + 3
5 + 4 + 3
```

are counted as three different partitions.

(2) Different orders are considered as the same partitions;

```
3 + 4 + 5 and
5 + 3 + 4
```

are counted as only one partition.

Please indicate which definition you choose.

10 UML and GPA 3, 1 point

Draw an appropriate UML diagram according to the following description:

In an online Super Tetris game, a rotation operation involves four objects: `Player`, `Playfield`, `Piece` and `Server`. When rotation is triggered, `Player` calls the `rotate` function of `Playfield` and waits for a boolean-typed return value. `Playfield` calls the `rotate` function of `Piece` to rotate 90 degree clockwise, and then send a string "rotate" to the `Server` and wait until `Server` returns "OK". Finally, `Playfield`'s `rotate` function returns true to `Player`.