COMP2012H Object Oriented Programming and Data Structures
Spring Semester 2013
Midterm Exam Solution
March 26, 2013, 10:30-11:50am in Room 3598
Instructor: Chi Keung Tang

This is a **CLOSED-BOOK-CLOSED-NOTES** exam consisting of five (5) questions. Write your answer in the answer booklet provided.

---

1. **OO concepts** (5 points)

   (a) What is *polymorphism* in object-oriented programming?

   (b) Express in *less than 40 words* how the following concepts are related: dynamic binding, virtual functions, overriding, static binding, operator overloading, STL.

2. **Template and operator overloading** (10 points)

   You are given the following class template specification and a main program where its sample output is also given. Your tasks are

   (a) implement the four member functions; use member initializer as much as possible when defining constructors

   (b) overload `operator+` as a member function; `operator<<` and the prefix `operator++` as non-member function; you can assume the types to be instantiated `T1` and `T2` are primitive types (one example is shown in the `main` program) but *no friend function is allowed.*

   You must not implement functions other than the above to make the main program run. You can define all member functions within class template `Tuple` (which is probably easier with less syntax to bother), or outside of the class template.

```
template <typename T1, typename T2>
class Tuple {
public:
  Tuple ();
  Tuple (const T1& t1, const T2& t2);
  T1 get_ele1() const;
  T2 get_ele2() const;
  void set_ele(T1 v1, T2 v2);
private:
  T1 ele1;
  T2 ele2;
};

int main()
{
  Tuple<int,double> t1 (2, 4.5);
  Tuple<int,double> t2 (4, 3.8);
  Tuple<int,double> t3 = t1 + t2;
  cout << t3 << endl;
  ++t3;
  cout << t3 << endl;
  return 0;
}
// output is
// (6, 8.3)
// (7, 9.3)
```

3. **STL and iterators** (5 points)

   Complete the following program using STL algorithms `copy` and `sort` and iterators. A total of only three statements is needed.

```
#include <stdlib.h>
#include <vector.h>
#include <string>
#include <algorithm>
#include <iostream>
using namespace std;

typedef                           istream_iterator_char;   // complete the declaration

int main() {
  vector<char> v;
  istream_iterator_char start (cin);
  istream_iterator_char end;
  back_insert_iterator<vector<char> > dest (v);

  // use stl's copy to read characters from cin and copy them to v
  // use stl's sort to sort v
  // use stl's to copy the sorted v to cout, each character followed by a space
}
```

4. **Order of Construction and Destruction** (5 points)

   What is the output of running the following program?

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

class Bulb {
public:
   Bulb() { cout << "B" << endl;}
   ~Bulb() { cout << "~B" << endl;}
};

class Lamp {
   Bulb bulb;
public:
   Lamp() { cout << "l" << endl;}
   ~Lamp() { cout << "~l" << endl;}
};
```

```
class Room {
public:
   Room() { cout << "R" << endl;}
   ~Room() { cout << "~R" << endl;}
private:
   Lamp l;
};

class Living_Room : public Room {
public:
   Living_Room() { cout << "L" << endl;}
   ~Living_Room() { cout << "~L" << endl;}
private:
   Bulb *b;
   Lamp l;
};

int main ()
{
  static Lamp ll;
  { Room r; }
  Living_Room lr;
  return 0;
}
```

5. **Abstract Base Class and Inheritance** (10 points)

   Code up the abstract base class `BasePoint` and the derived class `Figure2P` so that the program below compiles and runs. A sample run is also included.

   Use member initializer whenever possible and the highest level of information hiding (i.e., must have no public data members).

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

class BasePoint {
  // your definitions here
};
class Figure1P : public BasePoint {
  int p1;
public:
  Figure1P(int px, int py, int r):p1(r),BasePoint(px, py) {}
  virtual void info() {
    BasePoint::info();
    cout << "property 1: p=" << p1 << endl;
  }
};
class Square : public Figure1P {
public:
  Square(int px, int py, int r):Figure1P(px, py, r) {};
  virtual string type() {
    return "square";
  }
};
class Circle : public Figure1P {
public:
  Circle(int px, int py, int r):Figure1P(px, py, r) {}
  virtual string type() {
    return "circle";
  }
};
class Figure2P : public Figure1P {
  // your definitions here
};
class Rectangle : public Figure2P {
public:
  Rectangle(int px, int py, int w, int h):Figure2P(px, py, w, h) {}
  virtual string type() {
    return "rectangle";
  }
};
class Oval : public Figure2P {
public:
  Oval(int px, int py, int w, int h):Figure2P(px, py, w, h) {};
  virtual string type() {
    return "oval";
  }
};
int main(void) {
  BasePoint **objs = new BasePoint*[5]; // allocate space for 5 BasePoint pointers,
                                        // they may be used to store derived classes
  // creating objects
  objs[0] = new Circle(7, 6, 55);
  objs[1] = new Rectangle(12, 54, 21, 14);
  objs[2] = new Square(19, 32, 10);
  objs[3] = new Oval(43, 10, 4, 3);
  objs[4] = new Square(3, 41, 3);
  bool flag=false;
  do {
    cout << endl << "We have 5 objects with numbers 0..4" << endl;
    cout << "Enter object number to view information about it " << endl;
    cout << "Enter any other number to quit " << endl;
    char onum;
    cin >> onum;
    flag = ((onum >= '0')&&(onum <= '4'));
    if (flag)
      objs[onum-'0']->info();
  } while(flag);
  // freeing memory
   for(int i=0;i<5;i++) delete objs[i];
   delete [] objs;
}
```

Sample run

```
We have 5 objects with numbers 0..4
Enter object number to view information about it
Enter any other number to quit
0

figure: circle
position: x=7, y=6
property 1: p=55

We have 5 objects with numbers 0..4
Enter object number to view information about it
Enter any other number to quit
1

figure: rectangle
position: x=12, y=54
property 1: p=21
property 2: p=14

We have 5 objects with numbers 0..4
Enter object number to view information about it
Enter any other number to quit
2

figure: square
position: x=19, y=32
property 1: p=10

We have 5 objects with numbers 0..4
Enter object number to view information about it
Enter any other number to quit
3

figure: oval
position: x=43, y=10
property 1: p=4
property 2: p=3

We have 5 objects with numbers 0..4
Enter object number to view information about it
Enter any other number to quit
4

figure: square
position: x=3, y=41
property 1: p=3

We have 5 objects with numbers 0..4
Enter object number to view information about it
Enter any other number to quit
5
```

SOLUTIONS TO MIDTERM

1. **OO concepts** (5 points)

   (a) Working with objects without knowing their precise type; at compile or run time.

   (b) Dynamic binding is necessary for dynamic polymorphism, which is implemented in C++ using virtual functions in inheritance to get overriding done. Static binding achieves static polymorphism: STL and operator overloading are examples in C++. (34 words)

2. **Template and operator overloading** (10 points)

```
template <typename T1, typename T2>
class Tuple {
public:
  Tuple () : ele1(0), ele2(0) {}
  Tuple (const T1& t1, const T2& t2): ele1(t1), ele2(t2) {}
  Tuple operator+ (const Tuple& t) const
    { return Tuple (ele1 + t.ele1, ele2 + t.ele2); }
  T1 get_ele1() const { return ele1; }
  T2 get_ele2() const { return ele2; }
  void set_ele(T1 v1, T2 v2) { ele1 = v1; ele2 = v2; }
private:
  T1 ele1;
  T2 ele2;
};


/*
// if defined outside of the class
template <typename T1, typename T2>
Tuple<T1,T2> Tuple<T1,T2>::operator+ (const Tuple<T1,T2> &t) const
    { return Tuple<T1,T2> (ele1 + t.ele1, ele2 + t.ele2); }
*/


template <typename T1, typename T2>
ostream& operator<< (ostream& os, const Tuple<T1,T2> &t)
{
  os << "(" << t.get_ele1() << ", " << t.get_ele2() << ")" ;
  return os;
}

template <typename T1, typename T2>
Tuple<T1,T2> operator++ (Tuple<T1,T2> &t)
{
   t.set_ele (t.get_ele1()+1, t.get_ele2()+1);
   return t;
}
```

3. **STL and iterators** (5 points)

```
  // use stl's copy to read characters from cin and copy them to v
  copy (start, end, dest);
  // use stl's sort to sort v
  sort(v.begin(), v.end());
  // use stl's to copy the sorted v to cout, each character followed by a space
  copy (v.begin(), v.end(), ostream_iterator<char> (cout, " "));
```

4. **Order of Construction and Destructio** (5 points)

```
B
l
B
l
R
~R
~l
~B
B
l
R
B
l
L
~L
~l
~B
~R
~l
~B
~l
~B
```

5. **Abstract Base Class and Inheritance** (10 points)

```
class BasePoint {
  int x, y; //position
public:
  BasePoint(int px, int py):x(px),y(py) {}
  virtual string type() = 0;
  virtual void info() {
    cout << endl << "figure: " << type() << endl;
    cout << "position: x=" << x << ", y=" << y << endl;
  }
};
class Figure2P : public Figure1P {
  int p2;
public:
  Figure2P(int px, int py, int w, int h):p2(h),Figure1P(px, py, w) {}
  virtual void info() {
    Figure1P::info();
    cout << "property 2: p=" << p2 << endl;
  }
};
```