COMP152H Object Oriented Programming and Data Structures
Spring Semester 2011
Midterm Exam
March 22, 2011, 9:00-10:20am 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 30 words* how the following concepts are related: dynamic polymorphism, dynamic binding, virtual functions, overriding, static polymorphism, template.

2. **Abstract Base Class** (5 points)

   The following shows the implementation of three classes derived from `Animal`. The `main` program and its output are also shown. Code up the `Animal` base class. Use member initializers whenever possible when you define constructors.

```
class Cat: public Animal {
public:
    Cat(std::string strName) : Animal(strName) { }
    const char* Speak() { return "Meow"; }
};
class Dog: public Animal {
public:
    Dog(std::string strName) : Animal(strName) { }
    const char* Speak() { return "Woof"; }
};
class Cow: public Animal {
public:
    Cow(std::string strName) : Animal(strName) { }
    const char* Speak() { return "Moo"; }
};

int main()
{
    // Animal animal;   cannot declare Animal object; compilation error
    Animal *animal_arr[3];
    Cow cCow("Betsy"); Dog dDog("Lucky"); Cat cCat("Tom");
    animal_arr[0] = &cCow; animal_arr[1] = &dDog; animal_arr[2] = &cCat;
    for (int i=0; i<3; i++)
      std::cout << animal_arr[i]->GetName() << " says "
                << animal_arr[i]->Speak() << "... ";
}
The output is
Betsy says Moo...  Lucky says Woof...  Tom says Meow...
```

3. **Inheritance** (10 points)

You are given `toy.h` and a main program in the following. Your task is to code up `toy.cpp` so that the main program runs and produces the output on the orders of construction and destruction. Note that a dynamic array is allocated when an object is instantiated in both the base class and derived class. Your implementation must not have memory leak or dangling pointers, and you must use member initializer when defining constructors as much as possible.

```
// toy.h
class B {
public:
  B(int n = 10);   // initialize a dynamic array iarr of size n
  virtual ~B();    // should deallocate memory
private:
  static int bcnt;
  const int MAX_iSIZE;  // maximum size of iarr
  int *iarr;
};
class D : public B {
public:
  D(int n = 10);   // initialize a dynamic array carr of size n
  virtual ~D();    // should deallocate memory
private:
  static int dcnt;
  const int MAX_cSIZE;  // maximum size of carr
  char *carr;
};
// main program
int main()
{
  B** b;
  b = new B*[2];
  for (int i=0;i<2;i++) b[i] = new D(10);
  for (int i=0;i<2;i++) delete b[i];
  delete b;
  return 0;
}
The output is
B constructed: 1
D constructed: 1
B constructed: 2
D constructed: 2
D destructed: 2
B destructed: 2
D destructed: 1
B destructed: 1
```

4. **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 and `operator<<` as a 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).

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 about), or outside of the class template.

```
template <typename T1, typename T2>
class Tuple {
public:
  Tuple ();     // assume default values of data members to be 0
  Tuple (const T1& t1, const T2& t2);
  // specify the function prototype for overloading the '+' operator
  T1 get_ele1() const;
  T2 get_ele2() const;
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;
  return 0;
}
// output is
// (6, 8.3)
```

5. **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<int> istream_iterator_int;
int main() {
  vector<int> v;
  istream_iterator_int start (cin);
  istream_iterator_int end;
  back_insert_iterator<vector<int> > dest (v);
  // use stl's copy to read integers 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 number followed by end of line
  return 0;
}
```

SOLUTIONS TO MIDTERM

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

   (a) Polymorphism in object-oriented programming means that we can work with objects without knowing their precise type at compile time; in C++, the same function can respond differently depending on the actual object being processed.

   (b) In dynamic polymorphism, dynamic binding is necessary, which is implemented using virtual functions in an inheritance setting to achieve overriding. To achieve static polymorphism, template is used in C++. (29 words)

2. **Abstract Base Class** (5 points)

```
class Animal
{
private:
    std::string m_strName;

public:
    Animal(std::string strName)
        : m_strName(strName)
    {
    }

    std::string GetName() { return m_strName; }
    virtual const char* Speak() = 0;
};
```

3. **Inheritance** (10 points)

```
int B::bcnt = 0;
int D::dcnt = 0;

B::B(int n) : iarr(new int [n]), MAX_iSIZE(n) {
  B::bcnt++;
  printf ("B constructed: %d\n", B::bcnt);
}

B::~B() {
  delete [] iarr;
  printf ("B destructed: %d\n", B::bcnt);
  B::bcnt--;
}

D::D(int n) : B(n), carr(new char [n]), MAX_cSIZE(n) {
  D::dcnt ++;
  printf ("D constructed: %d\n", D::dcnt);
```

```
}

D::~D() {
  delete [] carr;
  printf("D destructed: %d\n", D::dcnt);
  D::dcnt --;
}
```

4. **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; }
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;
}
```

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

```
  // use stl's copy to read integers 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 number followed by end of line
  copy (v.begin(), v.end(), ostream_iterator<int>(cout, "\n"));
```