# ECE 462 Midterm Exam 3

# 10:30-11:20AM, November 16, 2007

## 1 Multiple Thread and Synchronization

### 1.1 Multithread

Which statement is correct?

*Answer: B*

A. A multithread program always produces the same result if the program executes on the same computer.

B. In Java, `x += y;` is **not** an atomic operation.

C. In C++, if a class implements the `Runnable` interface, the class must override the `run` method.

D. It is possible to cause deadlock even if there is only one program running and the program has only one thread.

E. In Java, if a `synchronized` method calls another `synchronized` method, deadlock always occurs.

### 1.2 Thread in Java

Which statement is correct?

*Answer: E*

A. Calling the `run` method of a thread object (for example `obj.run()`) will invoke the object's `start` method.

B. Creating a thread (calling `new`) makes the thread execute immediately in Java. In C++ (with Qt), a thread executes only after the method `start` is called.

C. A C++ (with Qt) thread can be created by instantiating a class that is derived (`extends`) from `Runnable`.

D. If a thread is created (`new`) first, the thread always finishes first.

E. After creating a thread, call `start` to start the thread's execution.

## 1.3 Thread Execution Order

What is the output of this program?

- If the program cannot compile, write "cannot compile".
- If the program causes run-time exception,write "exception".
- If the program *may* cause deadlock, write "deadlock".
- If the program enters an infinite loop, write down the first 4 lines of the output.

*Answer: f1 f2*

```
// outcome 8, question 3
class C1 extends Thread {
    synchronized void f1() {
        System.out.println("f1");
        f2();
    }
    synchronized void f2() {
        System.out.println("f2");
    }
    public void run() {
        f1();
    }
}

class J8Q3 {
    public static void main( String[] args )
    {
        C1 cobj = new C1();
        cobj.start();
    }
}
```

## 1.4 Performance Improvement

Suppose 90% of a program can be perfectly parallelized using multiple threads. The remaining 10% must execute sequentially. If the program has 9 threads running on a machine with 16 processors, what is the speedup? Please write the formula; you do not have to calculate the numeric result.

*Answer:* $\frac{1}{1-0.9+\frac{0.9}{9}} = \frac{1}{0.1+0.1} = \frac{1}{0.2} = 5$

## 1.5 Improve Thread Efficiency

Which method is likely to improve thread efficiency?

*Answer: B*

A. Do not synchronize. Allow threads to modify shared data freely.

B. Reduce the size of critical sections and use synchronization only when necessary.

C. Use as few threads as possible.

D. Use condition variables (such as `QWaitCondition`) instead of mutex lock (such as `QMutex`).

E. Use `wakeAll` instead of `wait` when a waiting condition becomes true.

## 1.6 Threads and Object

Which is **not** a possible output of this program?

*Answer: D*

```
// Outcome 8, Question 6
#include <QThread>
#include <QMutex>
#include <QWaitCondition>
#include <cstdlib>
#include <iostream>
#include <ctime>
#include <stdlib.h>
#include <sys/time.h>
using namespace std;

void keepBusy() {
  double howLongInMillisec = 0.8 *(random() % 100) ;
  // cout << howLongInMillisec << endl;
  int ticksPerSec = CLOCKS_PER_SEC;
  int ticksPerMillisec = ticksPerSec / 1000;
  clock_t ct = clock();
  while ( clock() < ct + howLongInMillisec * ticksPerMillisec ) {
  }
}

class Account{
private:
  int balance;
public:
  Account() { balance = 100; }
  void deposit( int dep ) {
    int currentbalance = balance;
    keepBusy();
    int newbalance = currentbalance + dep;
    balance = newbalance;
  }
  int getBalance() {
    return balance;
  }
};


class Depositor : public QThread {
private:
  Account * saving;
  int amount;
public:
  Depositor(Account * sacct, int n) {
    saving = sacct;
```

```
    amount = n;
  }
  void run() {
    saving -> deposit(amount);
  }
};

int main()
{
  /* to set the random seed */
  struct timeval tp;
  gettimeofday(&tp, NULL);
  srandom(tp.tv_usec);
  /* actual program starts here */
  Account * sacct = new Account;
  Depositor* depositors[3];
  depositors[0] = new Depositor(sacct, 1);
  depositors[1] = new Depositor(sacct, 2);
  depositors[2] = new Depositor(sacct, 3);
  for ( int i=0; i < 3; i++ ) {
    depositors[i]->start();
  }
  for ( int i=0; i < 3; i++ ) {
    depositors[i]->wait();
  }
  cout << sacct -> getBalance() << endl;
}
```

A. 101

B. 102

C. 103

D. 100

E. 105

# 2 Overloading and Overriding

## 2.1 Overloading in C++ and Java

Which statement is correct?

*Answer: A*

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

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

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

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.

## 2.2 Overloading in C++ and Java

Which statement is correct?

*Answer: E (12.9)*

A. In C++, an integer can be promoted to string automatically.

B. In C++, only primitive types can distinguish which version of the overloaded function. Objects cannot decide the overloaded function.

C. In Java, an overloaded function does not have to be a method in any class.

D. In Java, if a function is overloaded, a derived class must override all versions of the overloaded function.

E. In C++, a constructor can be used to convert two user-defined classes even though these two classes do not form base-derived relationship.

## 2.3 Overloading in Java

Which statement is correct?

*Answer: C*

A. In Java, a program can compile if ambiguity occurs in deciding which version of an overloaded function to choose. This program will cause run-time exception.

B. In Java, a user can give "hint" during program execution to decide which version of an overloaded function to choose by specifying the input at run-time.

C. In Java, if an exact match is unavailable, an object of class X may match to an object of class Y, here X is derived from Y (`class X extends Y`).

D. In Java, overloading must be distinguished using objects; primitive types are **not** allowed.

E. In Java, overloaded functions can have the same input parameters (numbers and types) and be distinguished by return types only. In C++, overloaded functions can be distinguished only by input parameters (numbers and types), not by return types.

## 2.4 Overloading in Java

What is the output of this program? If the program cannot compile, write "cannot compile".

*Answer: Df1 Bf2 Bf3*

```
// outcome 3, question 4
class Base {
    public void foo() {
        System.out.println( "Bf1" );
    }
    public void foo( int i ) {
        System.out.println( "Bf2" );
    }
    public void foo( int i, int j ) {
        System.out.println( "Bf3" );
    }
}

class Derived extends Base {
    public void foo() {
        System.out.println( "Df1" );
    }
}

class J3Q4 {
    public static void main( String[] args )
    {
        Derived d = new Derived();
        d.foo();
        d.foo( 3 );
        d.foo( 3, 4 );
    }
}
```

## 2.5 Overloading and Overriding and Class Hierarchy in C++

Which line of this program causes compile-time error? If there are multiple answers, you need to answer only one. If the program has no compile-time error, write 0.

*Answer: Line 9*

```
In member function ‘void Base::bar(int)’:

9:  error:  no matching function for call to ‘Base::foo(int&)’

10:  note:  candidates are:  virtual void Base::foo()
```

```
 1 // outcome 3, question 5
 2 #include <iostream>
 3 using namespace std;
 4
 5 class Base {
 6 public:
 7   Base() { }
 8   void bar() { foo(); }
 9   void bar(int x) { foo(x); }
10   virtual void foo() { cout <<"Bf1"<< endl;}
11 };
12
13 class Derived : public Base {
14 public:
15   Derived() { }
16   ~Derived(){}
17   virtual void foo() { cout << "Df1" << endl; }
18   virtual void foo(int x) { cout <<"Df2"<< endl;}
19 };
20
21 int main() {
22   Base* p = new Derived;
23   p->bar();
24   delete p;
25   return 0;
26 }
```

## 2.6 Overloading and Overriding

What is the output of this program? If the program cannot compile, write “cannot compile”.

*Answer: Bf (15.15)*

```
// outcome 3, question 6
class Base {
    public Base() {}
    private void foo(){ // notice ‘‘private’’
        System.out.println("Bf" );
    }
    public void bar() { foo(); }
```

```
}

class Derived extends Base {
    private void foo() {
        System.out.println( "Df" );
    }
    public Derived() {
    }
}

class J3Q6 {
    public static void main( String[] args ) {
        Base p = new Derived();
        p.bar();
    }
}
```

# 3 Template Classes and the STL Library in C++

## 3.1 Container Classes

Which statement is correct?

*Answer: E*

A. An element can be inserted anywhere in a queue.

B. Once a vector object is created, the last element cannot be deleted.

C. In a map, both the values and the keys must be unique. Thus, it must be a one-to-one mapping.

D. Inserting an element at the front of a vector usually takes the same amount of as inserting an element at the end. This is especially true if the vector has many elements.

E. An element can be inserted anywhere in a list.

## 3.2 Container Classes

Which statement is correct?

*Answer: A*

A. Elements can be inserted or removed anywhere in a vector.

B. Elements can be inserted only at the end of a list and removed at the beginning of a vector.

C. In a Java list, duplicate elements are **not** allowed.

D. In a C++ vector, duplicate elements are **not** allowed. The elements must be unique.

E. In a C++ map, elements can be inserted or removed only at the beginning.

## 3.3 Template in C++

Which statement is correct?

*Answer: A*

A. A stack can be efficiently implemented using a list.

B. A list can be efficiently implemented using a vector.

C. A vector can be efficiently implemented using a queue.

D. A list can be efficiently implemented using a set.

E. A set can be efficiently implemented using a stack.

### 3.4 C++ Set and Class Hierarchy

What is the output of this program? If the program cannot compile, write "cannot compile".

*Answer: B0 D1 B2 (different orders acceptable, no duplicates)*

```
// outcome 5, question 4
#include <iostream>
#include <string>
#include <set>
using namespace std;

class BaseC
{
protected:
  int b_val;
public:
  BaseC(int val): b_val(val) {}
  virtual void print() { cout << "B" << b_val << endl; }
};

class DerivedC: public BaseC
{
public:
  DerivedC(int val): BaseC(val) {}
  virtual void print() { cout << "D" << b_val << endl; }
};
int main()
{
  set<BaseC*> bset;
  BaseC * bobj[3];
  bobj[0] = new BaseC(0);
  bobj[1] = new DerivedC(1);
  bobj[2] = new BaseC(2);
  bset.insert(bobj[0]);
  bset.insert(bobj[1]);
  bset.insert(bobj[2]);
  bset.insert(bobj[2]);
  bset.insert(bobj[1]);
  bset.insert(bobj[0]);
  typedef set<BaseC*>::const_iterator CI;
  for (CI iter = bset.begin();
       iter != bset.end();
       iter++)
    {
      (* iter) -> print();
    }
  return 0;
}
```

### 3.5 Container and Iterator

Replace /* here */ by declaring (and defining) an iterator.

*Answer: ListIterator iter = animals.listIterator();*

```
// outcome 5, question 5
import java.util.*;
class J5Q5 {
    public static void main( String[] args )
    {
        List<String> animals = new ArrayList<String>();
        animals.add( "cheetah" );
        animals.add( "lion" );
        animals.add( "cat" );
        animals.add( "fox" );
        animals.add( "cat" );
        /* here */ /* ---------- */
        while ( iter.hasNext() ) {
            System.out.println( iter.next()  );
            /* output: cheetah lion cat fox cat */
        }
    }
}
```

### 3.6 C++ Template

What is the output of this program? If the program cannot compile, write "cannot compile".

*Answer: 15 h ece462 6.5*

```
// outcome 5, question 6
#include <iostream>
#include <string>
using namespace std;
template <class Txxx, class T2> class Container2
{
public:
  Container2(Txxx t1in, T2 t2in): c2_t1(t1in), c2_t2(t2in)
  { /* nothing */ }
  Txxx getT1(void) { return c2_t1; }
  T2 getT2(void) { return c2_t2; }
private:
  Txxx c2_t1;
  T2 c2_t2;
};
int main(void)
{
  Container2<int, char> obj1(15, 'h');
  cout << obj1.getT1() << " " << obj1.getT2() << endl;

  Container2<string, float> obj2("ece462", 6.5);
  cout << obj2.getT1() << " " << obj2.getT2() << endl;
  return 0;
}
```

# 4 Inheritance and Polymorphism

## 4.1 Inheritance and Polymorphism

Which statement is correct?

*Answer: A*

A. In C++, polymorphism is achieved by using virtual functions.

B. Polymorphism allows objects to send messages to each other.

C. An object of the derived class "has an" object of the base class.

D. Encapsulation means an object's behavior may change based on whether this object is created for the base class or a derived class at run-time.

E. In Java, polymorphism is achieved using abstract classes.

## 4.2 Inheritance and Class

Which statement is correct?

*Answer: C*

A. A base class must be an abstract class.

B. Inheritance means an object (`classX objx`) can be created for its own (`classX`) class or a base class (`class Y` and classX extends classY).

C. A derived class is more specific than a base class.

D. Encapsulation means derived classes must **not** be abstract.

E. In C++, if a class has one or more virtual functions, the class is abstract.

## 4.3 Inheritance, Interface, and Implementation

*Answer: A*

Which statement is correct?

A. An abstract C++ class must have at least one method that is pure virtual.

B. If a Java class has a virtual function, this class is abstract.

C. If a class has no virtual function, this class cannot have derived classes.

D. If a C++ class is abstract, all methods must be pure virtual.

E. If class X is derived from class Y (`class X extends Y`), class Y is more specific than X.

## 4.4 Inheritance and Polymorphism

What is the output of this program? If the program cannot compile, write "cannot compile".

*Answer: XC YC YD XD*

```
// Outcome 1, Question 4
#include <iostream>
using namespace std;

class X {
public:
  X() {
    cout << "XC" << endl;
  }
  ~X() {
    cout << "XD" << endl;
  }
};

class Y : public X {
public:
  Y() {
    cout << "YC" << endl;
  }
  ~ Y() {
    cout << "YD" << endl;
  }
};

int main()
{
  Y* yptr = new Y;
  delete yptr;
  return 0;
}
```

## 4.5 Inheritance and Interface in Java

In Java, `Runnable` is an interface. How do you create a class that has the properties of this interface?

*Answer:* `class X implements Runnable`

## 4.6 Inheritance and Interface in Java

In Java, if class X is derived from another class (`extends`), X can still inherit (`implements`) an interface. True or False?

*Answer: True (15.18)*

# 5 User-Defined Operator Overloading

## 5.1 Operator Overloading and Object-Oriented Programming

Which statement is correct?

*Answer: B*

A. User-defined operator overloading is supported in C++ and Java.

B. If operators < and == are overloaded, calling <= will **not** be automatically converted to calling < and ==.

C. A binary operator can be implemented as a member function, but a unary operator cannot be a member function.

D. In C++, an operator can be overloaded for user-defined objects only. In Java, an operator can be overloaded for primitive types only.

E. User-defined operator overloading is necessary for polymorphism.

## 5.2 Operator Overloading in C++

Which statement is correct?

*Answer: E*

A. User-defined operator overloading is necessary for encapsulation.

B. Operator overloading is necessary if a base class is abstract.

C. If an operator is overloaded, polymorphism is disabled.

D. The word `virtual` can be added in front of an overloaded operator.

E. If an overloaded operator is a member function, this operator can access private attributes of the class.

## 5.3 Operator Overloading as Member Functions

Which statement is correct?

*Answer: A*

```
// outcome 2, question 3
#include <iostream>
using namespace std;
class MyComplex {
  double re, im;
public:
  MyComplex( double r, double i = 0) : re(r), im(i) {}
  MyComplex operator+( MyComplex) const;
  MyComplex operator-( MyComplex) const;
  friend ostream& operator<< ( ostream&, const MyComplex& );
};

MyComplex MyComplex::operator+( const MyComplex arg ) const {
  double d1 = re + arg.re;
  double d2 = im + arg.im;
  return MyComplex( d1, d2 );
}

MyComplex MyComplex::operator-( const MyComplex arg ) const {
  double d1 = re - arg.re;
  double d2 = im - arg.im;
  return MyComplex( d1, d2 );
}

ostream& operator<< ( ostream& os, const MyComplex& c ) {
  os << "(" << c.re << ", " << c.im << ")" << endl;
  return os;
}

int main()
{
  MyComplex a(3, 4);
  MyComplex b(2, 9);
  MyComplex c = 4.7 + a; // x <-----
  MyComplex d = b + 9.2; // y <-----
  return 0;
}
```

A. Line x causes compile-time error.

B. Line y causes compile-time error.

C. Both x and y cause compile-time error.

D. There is no compile-time error in the program.

E. The program has compile-time error (one or multiple) but it is **not** caused by x or y.

## 5.4 Operator Overloading Restrictions

Which operator cannot be overloaded?

*Answer: D*

A. []

B. ++

C. >>

D. ::

E. ()

## 5.5 Overloading Operator

Please fill in the place marked by /* here */. If nothing is needed, please write *nothing*.

*Answer:* `if (this != &str)`

```
// outcome 2, question 5
#include <cstring>
#include <vector>
#include <iostream>
using namespace std;

class MyString {
  char* charArr;
  int length;
public:
  MyString() {charArr = 0; length = 0;}
  MyString( const char* ch ) {
    length = strlen( ch );
    charArr = new char[ length + 1];
    strcpy( charArr, ch );
  }

  MyString( const char ch ) {
    length = 1;
    charArr = new char[2];
    *charArr = ch;
    *(charArr + 1) = '\0';
  }
  ~MyString() { delete[] charArr; }

  MyString( const MyString& str ) {
    length=str.length;
```

```
    charArr = new char[length+1];
    strcpy( charArr, str.charArr );
  }

  MyString& operator=( const MyString& str ) {
    if (str.charArr == 0) {
      delete[] charArr;
      charArr = 0;
      length = 0;
      return *this;
    }
    /* here */ /* ------------------------------ */
    {
      delete[] charArr;
      charArr = new char[str.length + 1];
      strcpy(charArr, str.charArr );
      length = str.length;
    }
    return *this;
  }
};

int main()
{
  MyString s0;
  MyString s1( "hello" );
  MyString s2 = s1;
  return 0;
}
```

## 5.6 Overloading Operator <<

Please fill in the place marked by /* here */. If nothing is needed, please write *nothing*.

*Answer:* `return os;`

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

class MyComplex {
  double re, im;
public:
  MyComplex( double r, double i ) : re(r), im(i) {}
  MyComplex operator-() const;
  friend ostream& operator<< ( ostream&, const MyComplex& );
};

MyComplex MyComplex::operator-() const {
  return MyComplex( -re, -im );
}

ostream& operator<< ( ostream& os, const MyComplex& c ) {
  os << "(" << c.re << ", " << c.im << ")" << endl;
  /* here */ /* -------------------------------- */
}

int main()
{
  MyComplex c(3, 4);
  MyComplex z = -c;
  cout << z << endl;
  return 0;
}
```

# Contents