Final Exam for Programming Methodologies (Spring of 2008)
Instructor: Kyuseok Shim

1. We would like to compute $T(n)$ where $T(n) = 4T(n/2) + n$ and $T(1) = 1$.

   (a) Compute $T(n)$ by iterating the recurrence. [5 points]
   SOLUTION: $T(n) = n + 4T(n/2) = n + 2n + 4n + ... + 4^{\log n}T(1) = \theta(n^2)$

   (b) Show $T(n) = O(n^2)$ by mathematical induction. [10 points]
   SOLUTION: Assume $T(k) \leq c_1k^2 - c_2k$ for $k < n$
   $T(n) = 4T(n/2) + n \leq 4(c_1(n/2)^2 - c_2(n/2)) + n = c_1n^2 - 2c_2n + n = c_1n^2 - c_2n - (c_2n - n) \leq c_1n^2 - c_2n$ if $c_2 \geq 1$

2. Consider the fibonacci numbers below:

$$fib(n) = fib(n-1) + fib(n-2)$$

   where $fib(0) = fib(1) = 1$. Note that we can show $(3/2)^n \leq fib(n) < (5/3)^n$ for $n > 4$.
   Provide the recurrence equation of time complexity $T(n)$ for the recursive function below and discuss the characteristics of $T(n)$. [5 points]

```
long fib(int n) {
   if (n <= 1)
      return 1;
   else
      return fib(n-1) + fib(n-2)
}
```

   SOLUTION: $T(n) = T(n-1) + T(n-2) + 2$. The running time of this algorithm grows exponentially.

3. Consider the following code. Find all compilation errors and give some explanations briefly. [5 points.]

```
int x,y;
const int* a = &x;
int* const b = &x;
const int* const c = &x;

a = &y;
*a = 3;
b = &y;
*b = 1;
c = &y;
*c = 3;
```

SOLUTION:

```
*a = 3;   ( a is nonconstant pointer to constant data. so, trying
to modify constant data is error)

b = &y;   ( b is constant pointer to nonconstant data. trying to
modify constant pointer is error)

c = &y;   ( c is constant pointer to constant data. trying to
modify constant pointer is error)

c = *y;    ( trying to modify constant data is error)
```

4. There are two other usages of const qualifier. One is (i) (const is placed after function parameters). Another one is (ii) (const is placed before return type). Explain each usage of const qualifier. [5 points.]

```
(i) myClass& myClass::myFunc(int a) const { }
(ii) const myClass& myClass::myFunc(int a) { }
```

SOLUTION: If a const qualifier is placed after function parameters, we cannot modify values of class member variables(objects). and If const is placed before the return type, the returned object is considered as constant object. so we cannot modify the returned object (or objects members).

5. Consider the following C++ class:

```
class TopClass {
protected:
    int i;
public:
    int j;
};
```

State whether each of the following class definition is okay or not. If not, explain why. [10 points]

(a)
```
class A {
public:
  void foo() { TopClass b; b.i = b.j;}
};
```

(b)
```
class A: public TopClass {
};
class B: public A {
public:
  void foo() { i = j; }
};
```

SOLUTION: (a) NOT OKAY. (b) OKAY.

6. Show the output of the following code and explain why you get the output. [10 points.]

```
#include <iostream>

#include <iostream>
class AnInteger {
    int k;
  public:
    AnInteger() { cout << "#1\n"; }
    AnInteger(const AnInteger &i) { k = i.k; cout << "#2\n"; }
    AnInteger(int n) { k = n; cout << "#3\n"; }
    AnInteger operator + (const AnInteger &i) const {
      cout << "#4\n";
      return AnInteger();
    }
};
int main() {
  AnInteger x = 3 + 4;
  AnInteger y = x + 3;
}
```

SOLUTION:
#3
#3
#4
#1

7. Consider the following code in main(). The member function fly() returns negative number if its object cannot fly. Otherwise, it prints the characteristics of its object. f() of `Eagle` prints "Eagle flies high.". fly() of `Pigeon` writes "Pigeon flies low.". However, Bigbird is a Sesame Street character who cannot fly. Thus, fly() returns negative number and will prints "Cannot fly!".

```
int main() {

  Bird *b[4];
  b[0] = new Eagle();
  b[1] = new Pigeon();
  b[2] = new Bigbird();
  b[3] = new Pigeon();

  for (int i = 0; i<4; i++)
     if (b[i]->fly() < 0) {
        cout << "Cannot fly!\n";
        exit(1);
```

```
    }
}
```

The following is the output of the above main program.

```
Eagle flies high.
Pigeon flies low.
Cannot fly!
```

Instead of the above main program, we want to use an exception (i.e. try-catch) in C++. In this case, the return type of fly() becomes void and fly() will throw an exception if the object cannot fly. Provide C++ classes and main program so that the output of the above code is as follows: [10 points.]

SOLUTION:

```
#include <iostream>

class NotFly { };

class Bird {
  public:
    virtual void fly() = 0;
};

class Eagle : public Bird {
  public:
    void fly() {
      cout << "Eagle flies high.\n";
    }
};

class Pigeon : public Bird {
  public:
    void fly() {
      cout << "Pigeon flies low.\n";
  }
};

class Bigbird : public Bird {
  public:
    void fly() {
      throw NotFly(); }
};

int main() {

   Bird *b[4];
   b[0] = new Eagle();
```

```
  b[1] = new Pigeon();
  b[2] = new Bigbird();
  b[3] = new Pigeon();

  try{
     for (int i = 0; i<4; i++)
        b[i]->fly();
  }

  catch (NotFly &a ) {
     cout << "Cannot fly!\n";
  }
}
```

8. Consider the following code.

```
class a {
   public:
       a() {
          cout << "a()" << endl;
          arr = new int [2];
       }
       ~a() {
          cout << "~a()" << endl;
          delete arr;
       }
   private:
      int *arr;
};

class b : public a {
   public:
      b() {
         cout << "b()" << endl;
         foo = new int [2];
      }
      ~b() {
         cout << "~b()" << endl;
         delete foo;
      }
   private:
      int *foo;
};

int main () {
   a *tmp = new b;
   delete tmp;
}
```

(a) Show the output of the above code. Explain the order of calling for each constructor/deconstructor. [5 points.]

(b) Is there any problem in the above code? Fix the code for all the problems that you found. [5 points.]

SOLUTION:
(a)

```
 a()
 b()
~a()
```

(b) We should change `delete arr` **as** `delete [] arr` **and change** `delete foo` **as** `delete [] foo`**. We also should put virtual in front of the destructor of class a.**