Mid-term Exam I for Programming Methodologies
Instructor: Kyuseok Shim
You are not going to win at everything in life. Go out there and do your best.
When it is over, congratulate the winner – if it's not you!
– By Gail Denvers's father (Gail won a gold medal at the 1992 Olympic) –

1. Let $f_1(n) = O(T(n))$ and $f_2(n) = O(T(n))$. Show that $f_1(n) + f_2(n) = O(T(n))$. [5 points]

2. Explain three usages of `static` in C++. [10 points]

   SOLUTION: See the lecture notes.

3. We are interesting in a function that returns true when two given strings are exactly same. Let us consider the following C++ code comparing two string. What kind of problem do you expect? How can you fix it? The problem can be fixed by changing only two characters from the code. [5 points]

```
bool compareString (char* str1, char* str2) {
    char* ptr1 = str1;
    char* ptr2 = str2;

    while (*ptr1 != '\0' && *ptr2 != '\0') {
        if (*ptr1 != *ptr2) return false;
        if (*ptr1 != '\0') ptr1++;
        if (*ptr2 != '\0') ptr2++;
    }

    return true;
}
```

   SOLUTION: The strings such as "abc" and "abcd" are considered the same. We need to change && as || to fix.

4. Consider the following declaration.

   - What is the expected output? Explain how the content of array `var` changes how the pointer `p` changes in the function boo. [10 points]
   - Do you see any problem in this code? If you do, how will you fix it? [5 points]

```
#include <iostream>

class mymath {
    private:
        int* foo (int *a);
    public:
        mymath ();
        void boo ();
        int var[10];
};

mymath::mymath () {
    for (int i=0; i<10; i++) var[i] = 1;
}

int* mymath::foo (int *a) {
    *(a+1) = *a + *a;
    return a+1;
}

void mymath::boo () {
    int *p = var;
    for (int i=0; i<10; i++) p = foo (p);
}

int main () {
    mymath m;
    m.boo();
    std::cout << m.var[9] << std::endl;
    return 0;
}
```

SOLUTION: 512. 2^i **is stored in var[i]. We should make int var[11] instead of int var[10]**

5. Consider the following declaration.

```
class Polygon {
private:
  double width;
  double height;

protected:
  double area;

public:
  double GetWidth() { return width; }
  double GetHeight() { return height; }
  Polygon(double w, double h) { width = w; height = h; }
  ~Polygon() {}
};
```

(a) Describe the access specifier "protected" in C++. Provide the difference from "private." [5 points]

(b) Make Triangle and Rectangle classes which inherit from the above Polygon class. You should include following methods: (1) the appropriate constructor for each classes, (2) the function which computes area and set the area (`setArea`), (3) the function which returns area (`getArea`). Don't use the width, height, area member variables explicitly in the child class. [5 points]

SOLUTION: (a) See the lecture notes.
(b)

```
class Triangle : public Polygon {
    public:
       void SetArea() { area = GetWidth() * GetHeight() / 2; }
       double GetArea() { return area; }
       Triangle(double w, double h) : Polygon(w, h) { }
};
```

```
class Rectangle : public Polygon {
    public:
       void SetArea() { area = GetWidth() * GetHeight(); }
       double GetArea() { return area; }
       Rectangle(double w, double h) : Polygon(w, h) { }
};
```

6. Consider the following code. Complete the declaration and definition of = operator and + operator on the line 1), 2), 3) and 4). You should not use friend keyword. (= operator copies all integers in right-hand side array to left-hand side array. Be aware that you have to copy the contents of array, not the pointer. For example, if a is $< 1, 2, 3 >$, b should be $< 1, 2, 3 >$ after "b = a." + operator concatenates two array. It mean that $< 1, 2 > + < 10, 11, 12 >$ should be $< 1, 2, 10, 11, 12 >$.) [15 points]

```
class MyIntArray {
   private:
      int* array;
      int size;
   public:
      MyIntArray();
      MyIntArray(int* intarray, int n);

1)    ______________________    // declaration of = overloading

2)    ______________________    // declaration of + overloading };

MyIntArray::MyIntArray() {
    array = NULL;
    size = 0;
}

MyIntArray::~MyIntArray() {
    delete []array;
}

MyIntArray::MyIntArray(int n, int* a) {
    size = n;
    array = new int[size];
    for ( int i = 0; i < size; i++ ) array[i] = a[i];
}

3)  ________________________ // definition of = overloading

4) ________________________ // definition of + overloading
```

SOLUTION:

```
1) MyIntArray& operator=(const MyIntArray&);

2) MyIntArray operator+(const MyIntArray&) const;

3) MyIntArray& MyIntArray::operator=(const MyIntArray& m) {
    if ( size != m.size) {
       delete []array;
       size = m.size;
       array = new int[size];
    }
    for ( int i = 0; i < size; i++ ) array[i] = m.array[i];
    return *this;
}

4) MyIntArray MyIntArray::operator+(const MyIntArray& m) const {
    MyIntArray res;
    int i;
    res.size = size + m.size;
    res.array = new int[res.size];
    for (i = 0; i < size; i++ ) res.array[i] = array[i];
    for (int j = 0; j < m.size; j++, i++ ) res.array[i] = m.array[j];
    return res;
}
```

7. Consider the following program. Explain the behavior of the code and point out the bug. Convert the code into correct one. [5 points]

```
class Shim {
    int *s;
  public:
    Shim() { s = new int; *s = 1; }
    ~Shim() { delete s; }
};

int main() {
  Shim x;
  Shim y = x;
  return 1;
}
```

SOLUTION: The default copy constructor called by the statement "Shim y = x;" just copies each field separately. So x.s and y.s point to the same object, and the same dynamically created int will be deleted by the two destructors.

8. The following function tests whether the given array of integers contain at least one non-null value. It terminates as soon as a non-null value is found. In the worst case, the cost is the size of the array. Assuming that the probability of being null value for every array element is exactly 0.2. What is the average time complexity of the following procedure? [10 points]

```
bool testNonNull(int *a, int n) {
  for (int i = 0; i < n; i++)
    if (a[i] != 0)
      return true;
  return false;
```

SOLUTION: The probability is 0.2 for the loop to terminate after the first iteration. $0.8 * 0.2$ to terminate after the second iteration. Thus, we have

$$0.2\sum_{i=1}^{n} 0.8^{i-1} i \leq 0.2\frac{1}{1-0.8^2} = O(1)$$

9. Stooge sort stoogesort(int *L, int i, int j) is a recursive sorting algorithm that is defined as follows:

   (a) If the current list subset has at most one value, exit the procedure immediately.

   (b) If the current list subset has two values and the value at the end is smaller than the value at the start, swap them and exit the procedure.

   (c) If there are 3 or more elements in the current list subset, then:

      - Stooge sort the initial 2/3 of the list
      - Stooge sort the final 2/3 of the list
      - Stooge sort the initial 2/3 of the list again else:

   (d) exit the procedure

- Provide C++ code for Stooge sort function `void stoogesort(int *L, int i, int j)`. [5 points]
- Prove the Stooge sort always sort correctly for a sequence of n numbers in the array L by mathematical induction. [10 points]
- Formulate the recurrence equation for time complexity $T(n)$ and calculate $T(n)$. [10 points]

SOLUTION:
(1)

```
void stoogesort(int *L, int i, int j)
     if (i >= j) return;
     if L[j] < L[i] {
        int t = L[i];
        L[i] = L[j];
        L[j] = t;
     }
     if j - i > 1 {
         int t = (j - i + 1)/3
         stoogesort(L, i  , j-t);
         stoogesort(L, i+t, j  );
         stoogesort(L, i  , j-t);
     }
```

(2) Basis case: For n=1 the Stooge sort sorts correctly.

Induction hypothesis: We assume that Stooge sort sorts any sequence of size less than n correctly (including any sequence of size).

Induction step: Note that we have three regions (of equal size): region A from i to i+k, region B from i+k to j-k, and region C from j-k to j. After the first recursive call, every value in A is less than or equal to any number in B. After the second recursive call), each number in B is less than or equal to any number in C. Thus C will now have the largest numbers in sorted order. After the third call is finished, every number in A is less than or equal to any number in B; thus B will contain the maximum numbers (sorted) from A and B. Therefore, at the end A, B, and C are sorted and the elements of A are less than or equal to the elements of B which in turn are less than or equal to the elements of C.

(3) $T(n) = 1 + 3T(2n/3)$. **Thus,** $T(n) = \sum_{i=1}^{\log^{n}_{3/2} -1} = n^{2.71}$