ESC101 : Fundamental of computing

II Mid-semester Exam **9:30-10:30AM** **1 October 2008**

SOLUTION with marking scheme

Name : **Roll No. :**

Section :

Instructions :

1. There are three double sided sheets in the question paper. There are total **five** questions. Please report immediately if your question paper does not meet these requirements.
2. You have to write answer to each question **only** in the space provided within or after the question.
3. Avoid overwriting. You will be provided with rough sheets. You need not return the rough sheet.
4. For the questions which require filling in the blanks, you have to write only in the blanks provided in the question. You can't add any new instruction or remove already existing instruction.

	Q1	Q2	Q3	Q4	Q5
Marks					

Q.1 (marks=2,1,2,2) We want to design a class **Date** with the following requirement.

- An object of this class should have information about the day, month, and year. Once an object of this class is created, no method outside the class should be allowed to change it.
- There should be a constructor which receives suitable arguments to create a **Date** object.
- There should be suitable methods in this class such that any method outside the class can retrieve month or year of a **Date** object.
- There should be a non-static method `Is_earlier_than(Date Q)` which would return true if the current date (the object on which we invoke this method) comes *earlier* than the date referenced by `Q`, and false otherwise. For example, as we all know, the date 30/11/2002 comes earlier than 12/1/2004.

Give a complete description of class based on the requirement given above.

```
public class Date
{
   private int day;
   private int month;
   private int year;

   Date(int d, int m, int y)
   {
     this.day = d;
     this.month = m;
     this.year = y;
   }

   public int getMonth()
   {  return this.month; }

   public int getYear()
   {  return this.year;  }

   public boolean Is_earlier_than(Date Q)
   {
      if(this.year<Q.year) return true;
      if(this.year = Q.year && this.month<Q.month) return true;
      if(this.year = Q.year && this.month=Q.month && this.day < Q.day) return true;
      else return false;
   }
}
```

Marking scheme : 2 marks for `attributes`, 1 mark for the `constructor`, one mark for `getMonth()`, one mark for `getYear()`, and 2 marks for the method `Is_earlier_than(Date Q)`.
Note : There are other solutions also possible for the method `Is_earlier(Date Q)`. For example,

```
  public boolean Is_earlier_than(Date Q)
  {  long x = this.day + this.month*100 + this.year*10000;
     long y = Q.day + Q.month*100 + Q.year*10000;
     if(x<y) return true;
     else return false
  }
```

There is no magic in choosing numbers 100 and 10000 in the solution. But not all numbers will work. Just ponder over it.

Q.2 (marks(5))
Consider the following class

```
public class Pair
{
     public int first;
     public int second;

     public Pair(int i,int j)
     {   first = i;
         second = j;
     }
}
```

What will be the output of the following program which can access the above class.

```
class example
{   public static void f(Pair P, int i)
    {
      P.first  = P.first + i;
      P.second = P.second + i;
      P = new Pair(3,4);
      System.out.println("P is : ("+P.first+","+P.second+")");
    }
    public static void f(Pair P, Pair Q)
    {
      P.first = P.first+Q.first;
      P.second= P.second+Q.second;
      Q=P;
    }

    public static void main(String args[])
    {
        Pair A = new Pair(1,2);
        Pair B = new Pair(10,20);
        Pair C = A;
        f(C,1);
        System.out.println("A is : ("+A.first+","+A.second+")");
        f(C,B);
        System.out.println("A is : ("+A.first+","+A.second+")");
        System.out.println("B is : ("+B.first+","+B.second+")");
        System.out.println("C is : ("+C.first+","+C.second+")");
    }
}
```

Answer :

1. P is : (3,4)
2. A is : (2,3)
3. A is : (12,23)
4. B is : (10,20)
5. C is : (12,23)

Marking scheme : Each correct answer carries 1 mark. **No partial credits**.

Q.3 (marks = 3,3)
Write the body of the following methods.

(a) In a positive integer `n`, critical digit is the right most digit of `n` which is smaller than the digit on its right. Index of critical digit is defined as : the number of digits to the right of the critical digit plus one. For example, for $n = 12349870$, the critical digit is 4 and its index is 5, whereas for $n = 987654$, the critical digit does not exists. The following method should return the index of the critical digit of n, if it exists, and -1 otherwise.

```
public static int index_of_critical_digit(int n)
{
    int current_digit=n%10;
    n = n/10;
    int index=1;
    boolean flag=true;
    while(n>0 && flag)
    {
       int prev_digit = current_digit;

       current_digit  = n%10;

       n=n/10;

       index = index+1;

       if(current_digit < prev_digit)

             flag = false;
    }

    if(flag==false) return index;

    else return -1;
}
```

(b) The following method returns the index of the right most digit of `n` which is bigger than `c`. If no such digit exists in `n`, then it returns -1. For example, for `n=7192` and `c=6`, the method returns 2; for `n = 1324` and `c=0`, the method returns 1; whereas for `n=5216` and `c= 8`, it returns -1.

```
public static int index_of_right_most_digit_bigger_than_c(int c, int n)
{
    int index=1;

    while( n > 0 && n%10 <= c)
    {
         n = n/10;

         index = index + 1;
    }

    if(n>0) return index;

    else return -1;
}
```

Marking scheme : Correct answer to each blank carries 0.5 marks. There is no **partial credits** for any blank.

Q4. (marks = (7)) Consider a positive integer `n` of type `int`. The next higher permutation of `n` is the smallest integer greater than `n` which is formed by permuting the digits of `n`. For example, the next higher permutation for `n`=1209861 is 1210689, the next higher permutation for `n`=1421731 is 1423117. Note that next higher permutation may not exist for every number.

The method `next_higher_perm` described below prints next higher permutation of a positive integer `num`. It uses the methods designed in **Question 3** and the following methods whose action is obvious from their name.

- **public static int Power(int i)** : which returns 10^{i}.
- **public static int digit_at_index(int i, int n)** : which returns the digit of `n` at `i`th place from right. For example for `i=3`, and `n`=3276, it returns 2.
- **public static int reverse(int n)** : which returns reverse of given number n. For example, reverse of 1249 is 9421 and reverse of 4330 is 334.

Fill in the blanks.

```
public static void next_higher_perm(int num)
{
     int c_index = index_of_critical_digit(num);

     if(c_index == -1)

        System.out.println("Next higher permutation does not exist for "+num);

     else
     {
       int c_digit = digit_at_index(c_index,num);

       int b_index = index_of_right_most_digit_bigger_than_c(c_digit,num);

       int b_digit = digit_at_index(b_index , num);

       int upper = num/Power(c_index);

       int lower = num%Power(c_index-1);

       lower = lower - (Power(b_index-1))*(b_digit - c_digit);

       int next_perm;

       next_perm= upper*(Power(c_index)) + b_digit*(Power(c_index-1)) + reverse(lower);

       System.out.print("The next higher permutation for "+num+" is "+next_perm);
     }
}
```

Suggestion : try this question only after you have attempted all other.

Marking scheme : There are seven blanks, weightage of each blank is 1 mark. **No partial credits** for blanks **except** for the **4th blank**. In this blank, if you write `(b_index,num)`, you get 1 mark. But if you write `(b_index)` only you get 0.5 mark only.

Q.5 (marks = 5)

Recall the clever prisoner problem **discussed in great detail during a lecture** : Initially there are n prisoners. They stand in a line every morning according in the order which is fixed right on the first day. Every morning, the warden kills mth prisoner. The process goes on and finally the last prisoner left is allowed to escape. We want to solve a related problem now.

Given the number of prisoners `n`, the selection number `m`, and an integer `i`, we want to find out the day when prisoner who stands at `i`th position on the first day will be executed or will escape prison. For example for `n=4, m=11`, the prisoner with initial position `i=1` will be executed on 3rd day, prisoner with initial position `i=2` will be executed on 2nd day, prisoner with initial position `3` will be executed on 1st day, the prisoner with initial position `4` will escape from prison.

For your help, I am providing the method `Unlucky_position(j,m)` below which computes the position of the prisoner who is executed on the day when there are `j` prisoners and the selection number is `m`.

```
public static int The_Unlucky_Position(int m, int j)
    {   if(m%j==0)   return j;
        else         return m%j;
    }
```

Fill in the blanks of the method Execution_day(n,m). You may use the above method.

```
public static void Execution_day(int n, int m, int initial_pos)
{
    int i = initial_pos;
    boolean Alive = true;
    int count = 0;

    while(Alive == true && n>1 )
    {
        int j = The_Unlucky_Position(m,n);

        if(j==i)  Alive = false;

        if(j<i)  i = i-1 ;

        count = count+1;

        n = n-1;
    }

    if(Alive==false)
    System.out.println("Prisoner with initial position "+initial_pos+" will be executed on day"+count);
    else
    System.out.println("Prisoner with initial position "+initial_pos+" will escape from the prison");
    }
}
```

Marking scheme : The two blanks in the condition of `while` loop carry 0.5 marks each. Each of the remaining blanks has weightage of 1 mark. **No partial credits** for any blank.