

UNIVERSITY OF TORONTO  
FACULTY OF APPLIED SCIENCE AND ENGINEERING  
APS105F — Computer Fundamentals  
Midterm Test #2, November 13th, 2000  
Examiners - John Carter, G. Lemieux, James MacLean

Duration: 6:10–8:00 PM

- This is a “closed book” test; no aids are permitted.
- No electronic or mechanical computer devices are permitted.
- Write your answer in the space that follows each question.
- If necessary, use the back of the pages for rough work.
- This test has 8 pages.
- You must use the Java programming language to answer programming questions.
- You may assume that the `Stdin` methods are available:  
`getInt()`, `getFloat()`, `getDouble()`, `getString()`, and `getChar()`.

Circle your lecture section:    L0101    or    L0102    or    L0103

Name \_\_\_\_\_ **SAMPLE SOLUTIONS** \_\_\_\_\_

Student Number \_\_\_\_\_

ECF Login \_\_\_\_\_

**MARKS**

Question #	1	2	3	4	5	6	<b>TOTAL</b>
Value	14	10	10	15	10	16	<b>75</b>
Mark							

1. [14 Marks] Circle the correct answer for each of the following:

- (a)  **True** or  **False**: Every element in an array must be of the same type.
- (b)  **True** or  **False**: In Java, one cannot create an array of objects, only an array of references to objects.
- (c)  **True** or  **False**: An instance variable of a class can only be modified by a mutator method.
- (d)  **True** or  **False**: The declaration `int[][] table = new table[5][];` is valid. This question contained a typo. It was intended to be `int [][] table = new int[5][];`. Either answer for this question is therefore acceptable.
- (e)  **True** or  **False**: Multiple constructors for an object can be created provided they each have a different return type.
- (f)  **True** or  **False**: `a.length()` invokes a method to find the length of array `a`.
- (g)  **True** or  **False**: For a sequential search to work correctly, the list must be sorted.
- (h)  **True** or  **False**: Methods that have two or more recursive calls are always slow.
- (i)  **True** or  **False**: Recursion requires 1) a base case and 2) a way of writing the problem as a simpler version of the same problem.
- (j)  **True** or  **False**: The declaration `int[] list = new int[10];` automatically initializes the elements of `list`.
- (k)  **True** or  **False**: A binary search works well on linked lists because they are easier to keep in order and can be of unlimited length.
- (l)  **True** or  **False**: `'a' < 'A' < 'b' < 'B'` is a valid lexicographic ordering in Java.
- (m)  **True** or  **False**: Given `n` elements to sort, any sorting algorithm must compare every possible pair of elements.
- (n)  **True** or  **False**: Array indexes always start at 0 in Java.

## 2. [10 Marks]

In mathematics, a matrix is said to be *symmetrical* if the matrix is square and, for all possible values of  $i$  and  $j$ , the element in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column is equal to the element in the  $j^{\text{th}}$  row and  $i^{\text{th}}$  column. As an example, the matrix shown below is symmetrical.

$$\begin{pmatrix} 5 & 2 & 7 & 0 \\ 2 & 4 & 9 & 1 \\ 7 & 9 & 6 & 3 \\ 0 & 1 & 3 & 4 \end{pmatrix}$$

Complete the definition of a method `isSymmetrical` so that it returns `true` if and only if the elements of an array form a symmetrical matrix. In your method, you may assume that the array is rectangular but no other assumptions should be made.

```
public static boolean isSymmetrical (int[][] a)
{
    // sol'n shown below only loops over 1/2 the off-diagonal
    // elements, comparing them to the other half.
    // A solution where i,j loop over all possible values is
    // less efficient, but correct.
    if (a.length != a[0].length) return false ; // not square
    for (int i = 0; i < a.length; i++)
        for (int j = 0; j < i; j++)
            if (a[i][j] != a[j][i]) return false ;
    return true ;
}
```

## 3. [10 Marks]

Complete the definition of a recursive method `fromBase` that has two integer parameters: the first parameter,  $n$ , is a non-negative number and the second parameter,  $b$ , is the base in which  $n$  is written. The method should return the base 10 value of the number. As an example, since  $32_5 = 17$ , a call to the method of the form `fromBase(32, 5)` should return 17. You should assume that  $2 \leq b \leq 10$  and that  $n \geq 0$ . Note that the method must be recursive. No credit will be given for a non-recursive version.

```
public static int fromBase (int n, int b)
{
    if (n < b)
        return n ;
    else
        return n % b + b * fromBase(n/b, b);
}
// Bonus: An even better solution might check that n % b < b
```

## 4. [15 Marks]

A Blip is a single dot at location  $x,y$  which moves with constant velocity components  $v_x, v_y$  across a computer screen of size WIDTH by HEIGHT. (Note: all values are integers). Furthermore, Blips obey the following constraints:

- The range of values for  $x, v_x$  is 0 to WIDTH-1 while the range of values for  $y, v_y$  is 0 to HEIGHT-1.
- When a Blip is created, the location and velocity values are randomly chosen from within their valid range.
- A Blip moves by adding the velocity components to the  $x$  and  $y$  location. If the Blip moves off the screen, it appears on the other side by using modulo arithmetic.
- If two Blips collide by sharing the same location, they lose their velocity and become stationary.

You must write two things:

- a Blip class that obeys the above constraints, and
- a main method (a screensaver!) that moves 100 blip objects around.

The main method should operate in an infinite loop that must first move all the blips before checking if any of them collide.

Note: `Math.random()` returns a random double value  $x$  in the interval  $0 \leq x < 1$ .

```
class Blip
{
    private int x, y, velx, vely;
    private static final int WIDTH = 1024;
    private static final int HEIGHT = 768;

    public Blip()
    {
        x = (int)(Math.random() * WIDTH);
        y = (int)(Math.random() * HEIGHT);
        velx = (int)(Math.random() * WIDTH);
        vely = (int)(Math.random() * HEIGHT);
    }

    public void move()
    {
        x = (x + velx) % WIDTH;
        y = (y + vely) % HEIGHT;
    }

    private void crash()
    {
        velx = vely = 0;
    }

    private void checkCrash( Blip other )
    {
        if( x == other.x && y == other.y ) {
            crash();
            other.crash();
        }
    }

    public static void main(String[] args)
```

```

{
    Blip[] blips = new Blip[100];

    while( true ) {

        // move all the blips
        for( int i=0; i < blips.length; i++ ) {
            if( blips[i] == null ) blips[i] = new Blip();
            else blips[i].move();
        }

        // check for collisions
        for( int i=0; i < blips.length; i++ ) {
            for( int j=i+1; j < blips.length; j++ ) {
                // NB: i != j
                blips[i].checkCrash( blips[j] );
            }
        }
    }
}

```

### 5. [10 Marks]

Given the definitions below of the components in a linked list, complete the definition of the `split()` method below to divide one linked list into two smaller ones. The original list keeps the front part, and the `split()` method returns a new linked list containing the back part.

The splitting point is found as follows. The head of the new linked list will be the first `Item` in the original list whose value matches the key parameter. If no match exists, the original list must remain intact and `split()` will return an empty list.

*Your solution may not create or delete any `Items`, nor may it use arrays.*

```

class Item
{
    public int value ;
    public Item next ;
}

class List
{
    private Item head;

    public List split(int key)
    {
        List newList = new List(); // create new, empty list

        Item current = head ; // start search at beginning of list
        Item last     = null ;

        boolean found = false ;

        while (!found && current != null)
        {

```

```

if (current.value == key) // found item!
{
    found = true ;

    newList.head = current ; // make Item referenced by 'current' first
                            // element in list 2
    if (current == head) // list 1 will be empty
        head = null ;
    else
        last.next = null ; // make Item referenced by 'last' the
                            // last element in list 1
}
else // keep searching
{
    last = current ;
    current = current.next ;
}
}

return newList ;
}
}

```

## 6. [16 Marks]

Suppose that an array is known to contain only integers in the range  $0 \leq i \leq 5$ . For each of the following operations on such an array,

- i) indicate whether it is easier to implement the operation if the array is known to be sorted in non-decreasing order, and
- ii) complete the definition of a method that will perform the operation efficiently on a sorted array.

- (a) Find and return the minimum value in the array.

```

public static int minimum (int[] list)
{
    // assume list != null and list.length != 0
    return list[0];
}

```

This operation *is* easier given a sorted array.

- (b) Compute and return the mean (average) of the values stored in the array.

```

public static double mean (int[] list)
{
    int sum = 0 ;

    for (int i=0; i < list.length; i++)
        sum += list[i] ;

    // assume list.length != 0
    return (double)sum/list.length ;
}

```

This operation does *not* benefit from the array being sorted.

- (c) Create and return an array containing the number of occurrences of each value in the original array. For example, if the original array contained the values {2, 3, 5, 1, 0, 1, 2, 1} then the method should produce an array containing {1, 3, 2, 1, 0, 1}.

```
public static int[] frequency (int[] list)
{
    int [] freq = new int[6] ; // know we only need account for 0 to 5

    for (int i = 0; i < list.length; i++)
        freq[list[i]]++ ;

    return freq ;
}
```

This operation (as implemented) does *not* benefit from a sorted array. Now, if you note that if the array is sorted that you need to only count until you see the first 5, then number of 5's is just the number of items remaining in the list. To get credit for this interpretation (*i.e.* that the operation is more efficient if sorted), you must explain why.

```
// Alternative solution
public static int[] frequency (int[] list)
{
    int [] freq = new int[6] ; // know we only need account for 0 to 5

    for (int i = 0; i < list.length && list[i] < 5; i++)
        freq[list[i]]++ ;
    freq[5] = list.length - i ;

    return freq ;
}
```

- (d) Compute and return the mode. A mode is a value that appears more frequently than any other. You may assume that the list contains only one mode.

```
public static int mode (int[] list)
{
    int [] freq = frequency(list); // get freq dist'n from 'list'

    int mode = 0 ; // init with an element from the freq array

    for (int i = 1; i < freq.length; i++)
        if (freq[i] > freq[mode])
            mode = i ;

    return mode ;
}
```

As written, this operation does *not* benefit from a sorted array. However, as pointed out in the previous part, other implementations might.

Note that an improvement is possible here: as the loop iterates, if the current value of mode can be shown to be large enough that it cannot be exceeded by counting the numbers remaining in the list then you can terminate the loop early. (A good way to

determine this is to keep track of the total number of list elements accounted for so far.)

```
// Alternative solution
public int mode(int [] list)
{
    int freq = new int[6] ;

    int i = 0 ;
    int mode = 0 ;

    while (i < list.length && list[i] < 5)
    {
        final int thisNumber = list[i++] ;
        freq[thisNumber]++ ;
        if (freq[thisNumber] > freq[mode])
            mode = thisNumber ;
        int remaining = list.length - i ;
        if (freq[mode] > remaining)
            return mode ;
    }
    freq[5] = list.length - i ;
    if (freq[5] > freq[mode])
        mode = 5 ;

    return mode ;
}
```