University of Toronto
csc148—Introduction to Computer Science, Fall 1997

First Term Test — Horton's morning section

Solutions!

Duration: 50 minutes

Aids allowed: None

Make sure that your examination booklet has 6 pages (including this one). Write your answers in the spaces provided. Write legibly.

Family Name: ______________________ **Lecturer:** ______________________

First Name: ______________________ **Tutor:** ______________________

Student # : ______________________

1. ____________ / 7
2. ____________ / 4
3. ____________ / 14
4. ____________ / 10

Total ____________ / 35

Question 1

[7 marks in total]

(a) [2 marks]
Which of the following is an ADT and which is a data structure? Circle the appropriate answer for each.

sorted array:	ADT	$\boxed{\text{data structure}}$
unsorted array:	ADT	$\boxed{\text{data structure}}$
ordered set:	$\boxed{\text{ADT}}$	data structure
circularly-linked list:	ADT	$\boxed{\text{data structure}}$

(b) [2 marks]
What two things must one specify in order to define an ADT?
`One must specifiy (1) the data of which the ADT consists (and any necessary constraints on it), and (2) the operations that can be performed on the data.`

(c) [3 marks]
Briefly explain what information-hiding is.
`Information-hiding occurs when we design a program in a way that ''hides'' information about the implementation of a class from code that uses the class.`

Why is information-hiding a good thing?
`It allows client code to use a class without the programmer having to know how the class works. Furthermore, it prevents client code from becoming dependent upon implementation details that may change -- thus giving the class flexibility to change. Information-hiding also protects the implementation from harm by client code.`

How do we achieve information-hiding in a Java program?
`We use accessibility modifiers like ''private''.`

Question 2

[4 marks]

Consider the following method:

```
// Return true if s occurs in A, and false otherwise.
static boolean occursIn (String[] A, String s) {
    for (int i=0; i<A.length; i++) {
          // If find a match at position i, return true.
          if (                )
             return true;
    }
    // Didn't find a match anywhere, so return false.
    return false;
}
```

Consider the following two ways to fill in the missing condition:

1. A[i] == s
2. A[i].equals(s)
   Note that `equals` is a method in class `String` that returns true if two strings contain the same sequence of characters, and false otherwise.

(a) [3 marks]
Explain how the method's behaviour will differ depending on whether we use condition (1) or condition (2).

`Condition (1) compares the references. It is only true if A[i] and s both refer to exactly the same instance of a String. Condition (2), on the other hand, compares the actual contents of the Strings. It will be true even if A[i] and s are two different instances of String with the same sequence of characters.`

(b) [1 mark]
Suppose we were writing a program to read a newspaper story from a file, and check to see if any of the words in it are from a given vocabulary list, stored in an array of strings. Which version of this method would be appropriate to use? Circle one answer.

version with condition (1) $\boxed{\text{version with condition (2)}}$

Question 3

[14 marks in total]

In this question, you will use the `IntNode` class, given here:

```
class IntNode {
   public int num;
   public IntNode next;

   // Constructor
   public IntNode(int n) {
      num = n;
      next = null;
   }
}
```

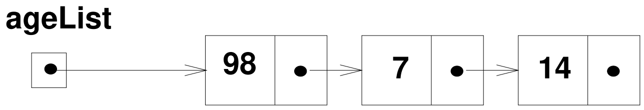
(a) [9 marks]
Complete the following Java method which attaches a new node containing the integer `n` to the end of the linked list referenced by `front`. Your method must also return a reference to the front of the revised list.

```
static IntNode append (IntNode front, int n) {


        IntNode newNode = new IntNode(n);
        IntNode temp = front;

        if (temp==null)
                return newNode;
        while (temp.next != null)
                temp = temp.next;
        temp.next = newNode;
        return front;
}
```

(b) [3 marks]
Using the `append()` method from part (a), write a fragment of Java code which will declare the variable `ageList` and create the linked list structure illustrated in the following diagram:



Don't worry about creating a method in which to put your code fragment. Assume that the append method is declared in the same class as your code fragment. Since it is declared `static`, it can be called directly, without naming a specific object whose `append` you are calling.

You can answer this question even if you have not completed part (a).

```
IntNode ageList = null;

ageList = append(ageList, 98);
ageList = append(ageList, 7);
ageList = append(ageList, 14);
```

(c) [2 marks]
We chose to have the append method return a reference to the front of the revised list. Was this necessary? Circle the appropriate answer:

$\boxed{\text{Yes}}$ No

Briefly explain your answer.

`The append method may be given an empty list to append to. In this case, the front reference has to change from null to being a reference to the new node. But because parameters in Java are pass-by-value, we can't change the value of the actual parameter by changing the formal parameter ''front''. We can only return a (possibly revised) front reference, and expect the caller to store it in the appropriate variable.`

Question 4

[10 marks in total]

Suppose you are writing a program to keep track of inventory in a warehouse. The following ADT would be useful: An **Inventory Item** consists of an identification number, and a count of the number of units currently in stock. The following operations can be performed on an Inventory Item:

- record arrival: Record the fact that a given number of units of the Inventory Item have arrived at the warehouse.
- record shipment: Record the fact that a given number of units of the Inventory Item have been shipped out of the warehouse.
- number of units: Return the number of units of the Inventory Item that are currently in stock.

Write a Java class for implementing this ADT. Marks will be awarded for good design.

```
class InventoryItem {
    private int itemID;
    private int inStock;

    public InventoryItem(int identifier) {
         itemID = identifier;
         inStock = 0;
    }
    public void recordArrival (int numUnits) {
         inStock = inStock + numUnits;
    }
    public void recordShipment (int numUnits) {
         inStock = inStock - numUnits;
    }
    public int numberOfUnits () {
         return inStock;
    }
}
```