University of Toronto
csc148F—Introduction to Computer Science, Fall 1997

Second Test — Horton's 10:00 section

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: ______________________

First Name: ______________________

Student # : ______________________

Tutor (circle one): Karen Reid Sasha Jovicic Brian Benwell Steve Myers

1.	________	/ 9
2.	________	/ 10
3.	________	/ 9
4.	________	/ 8
Total	________	/ 36

Question 1

[9 marks in total]

Consider the following fragment of Java code:

```
while (pos <= b) {
    sofar = sofar + A[pos];
    pos++;
}
```

(a) Give appropriate preconditions for this loop.

(b) Give appropriate postconditions for this loop.

First, give the most important postcondition — to do with the purpose of the code:

Now give any other postconditions:

(c) Give appropriate loop invariants for this loop.

First, give a loop invariant about the value of `sofar`:

Now give a loop invariant about the value of `pos`:

Question 2

[10 marks in total]

Consider the following Java code, and its specifications:

```
// Preconditions: n >= 1
int i = 1;
int ans = 1;
while (i != n) {        // Loop invariants: ans = i!  and  1 <= i <= n
    i++;
    ans = ans * i;
}
// Postconditions: ans = n!  and i = n+1
```

(a) [4 marks]

Let I refer to the loop invariants for the while-loop. Below is the outline for a proof that they are indeed loop invariants. Fill in the blanks to make the proof structure valid. *Do not* actually do the sub-proofs required in the base case and induction step.

BASE CASE: Prove that ______________________________

Let k be ____________________

INDUCTION HYPOTHESIS: Assume that ______________________________

INDUCTION STEP: Prove that ______________________________

INDUCTION CONCLUSION: The statements I are true at the beginning and ending of every iteration of the loop. That is, they are loop invariants.

Let us assume that the proof of the loop invariants has been completed. On the next page, we are going to prove the following statement:

if (the preconditions are true and the code terminates)
then at that point the postconditions will be true.

Remember that this is called proving "partial correctness", because it doesn't include proving that the code actually does terminate.

(b) [1 mark]
What sort of structure does this statement have? What sort of proof technique is thus appropriate?

(c) [5 marks]
Write the proof below. Use the proof technique you identified in part (b).

Question 3

[9 marks in total]

Consider the following Java method:

```
private static int mystery (int[] A, int first, int last) {
    System.out.println ("do " + first + " to " + last);
    int ans;
    if (first == last)
        ans = A[first];
    else {
        int m = (first + last) / 2;     // The '/' does integer division.
        int a = mystery (A, first, m);
        int b = mystery (A, m+1, last);
        if (a > b)
             ans = a;
        else
             ans = b;
    }
    System.out.println ("ans = " + ans);
    return ans;
 }
```

(a) [7 marks]
What output would result from the following statements, which run without crashing:

```
// This constructs an array of 5 elements containing the values given:
int[] A = {3, 9, 11, 4, 7};
System.out.println (mystery(A, 0, 4) + " is the final answer.");
```

(b) [2 marks]
If you were to prove the `mystery` method correct, you would need to specify its correct behaviour. Fill in the blank to complete the following specification:

Let $S(n)$ represent the following statement:
"If a call to mystery(list, p, q) is made, where $0 \leq p \leq q \leq$ list.length and $(q - p) = n$, then the call will return and

____.

Question 4

[8 marks in total]

The `Node` class, given here, can be used for a linked list of integers.

```
class Node {
    public int data;
    public Node next;

    // constructor: details omitted
}
```

Complete the method `smallestBigger`, begun below. It must return the smallest integer in the give list that is bigger than `n`, or return `n-1` if there is no integer greater than `n` in the list. Your solution **must** be recursive.

You may find Java's `Math.min(a,b)` method useful. It returns the minimum of the two given values.

```
public static int smallestBigger (Node front, int n) {
```