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

Second Test — Horton's 10:00 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: _____

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++;
}
```

There are several good solutions to this question. Below two are shown. Their answers cannot be mixed and matched though!

(a) Give appropriate preconditions for this loop.

Solution (1) is:
0 <= b < A.length; sofar = 0; and pos = 0

Solution (2) is:
0 <= b < A.length; sofar = has a value; and
pos has some value a such that 0 <= a <= b.

(b) Give appropriate postconditions for this loop.

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

Solution (1) is:
sofar = the sum of the elements A[0] to A[b] inclusive.

Solution (2) is:
sofar has been incremented by the sum of the elements A[a] to A[b] inclusive.

Now give any other postconditions:

pos = b+1

(c) Give appropriate loop invariants for this loop.

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

Solution (1) is:
sofar = the sum of the elements A[0] to A[pos-1] inclusive

Solution (2) is:
sofar has been incremented by
the sum of the elements A[a] to A[pos-1] inclusive

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

Solution (1) is:
0 <= pos <= b+1

Solution (2) is:
a <= pos <= b+1

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
```

(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 `I is true at the beginning of the first iteration.`

Let k be `an arbitrary integer >= 1`

INDUCTION HYPOTHESIS: Assume that `I is true at the beginning of the kth iteration.`

INDUCTION STEP: Prove that `I is true at the end of the kth iteration`
`(and thus at the beginning of the (k+1)st iteration).`

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.

CONTINUED

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

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

Assume that (the preconditions are true and the code terminates).

- Since the code terminates, we know that at that point the while-loop's condition must be false. That is, i = n. Thus the second post-condition is true.

- The loop invariant I was true at the end of the last iteration (by definition of a loop invariant), so it must still be true now, since nothing has happened other than to exit the loop.

- Thus ans = i!

- Since ans = i! and i = n, by substition, ans = n! Thus the first post-condition is true.

- So both post-conditions are true.

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

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.");
```

THE OUTPUT IS:

```
do 0 to 4       \
do 0 to 2     \ |
do 0 to 1   \ | |
do 0 to 0 \ | | |
ans = 3   / | | |
do 1 to 1 \ | | |
ans = 9   / | | |        Bracketing on the right side indicates which
ans = 9     / | |        output results from the same call to mystery.
do 2 to 2 \   | |
ans = 11  /   | |
ans = 11     /  |
do 3 to 4   \   |
do 3 to 3 \ |   |
ans = 4   / |   |
do 4 to 4 \ |   |
ans = 7   / |   |
ans = 7     /   |
ans = 11       /
11 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

`the value returned is the maximum value in A between positions p and q inclusive.`

Note that S(n) should have said q is strictly less than list.length.

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 given 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) {


ANSWER:

  if (front == null)
          return n-1;
  else {

          int rest = smallestBigger (front.next, n);
          if (front.data > n && rest > n)
                  return Math.min(front.data, rest);
          else if (front.data > n)  // and rest <= n
                  return front.data;
          else  // front.data <=n; don't know about rest
                  return rest;
  }
}
```