UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2003

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
BSc Honours Degree in Mathematics and Computer Science Part I
MSci Honours Degree in Mathematics and Computer Science Part I
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER C141=MC141

REASONING ABOUT PROGRAMS

Tuesday 6 May 2003, 16:00
Duration: 90 minutes
(Reading time 5 minutes)

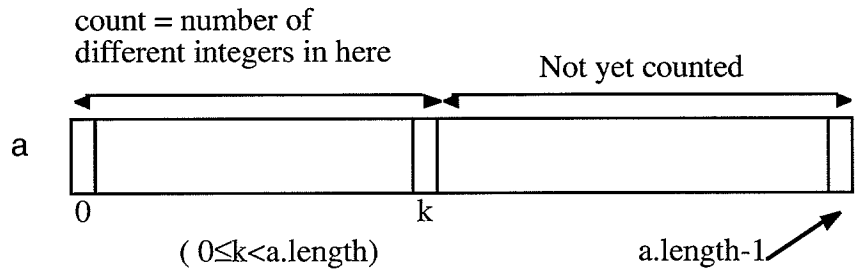
Answer THREE questions

Paper contains 4 questions
Calculators not required

1 Here is the outline of a Java method countElements, which counts the number of *different* integers that occur in *sorted* array a.

```
int countElements(int [] a) {
//pre: a is sorted in ascending order and non-empty
//i.e. a.length>0 & ∀i:Nat(i<a.length-1→ a[i]≤a[i+1])
//post:a=a0 &
//       r= #{j:Int | ∃i:Nat(i<a.length & a[i]=j)}
//i.e.r counts number of distinct integers that appear in a
  int k=0, count = 1;
  while (k<a.length-1) {
    //variant: a.length-k-1
    // invariant:                    part (a)
    if (a[k] < a[k+1]) count++;
    k++;
    }
  return count;
  }
```

a The following diagram details the state of the computation at the start of an arbitrary iteration of the while loop in terms of the relevant variables.



Using the diagram and the form of the postcondition as a guide, write down a suitable invariant..

b Show carefully that the code re-establishes the invariant.

c i) Show that the invariant is established at the start of the while loop.

ii) Show that the postcondition is achieved after the return.

The three parts carry, respectively, 20%, 45%, 35% of the marks.

2 The Haskell function `smallest` finds the smallest integer in a *non-empty* list of integers.

```
smallest :: [Int] -> Int
--pre ys ≠ [], where zs = smallest ys
smallest (x:xs)  = allmin x xs

allmin :: Int -> [Int] -> Int
--pre: none
allmin x []     = x
allmin x (y:yt) = allmin (min x y) yt
```

Show by *list induction* on *ys* the following properties of `allmin` and `smallest`:

a for all ys:[Int],
$\forall$ x, y:Int (`allmin (min x y) ys = min x (allmin y ys)`)

b for all **non-empty** ys: [Int]
$\forall$ x:Int (in(x, ys) $\rightarrow$ (`smallest ys`) $\leq$ x)

in(x,ys) means x is a member of the list ys.

(**Hints**: Use part (a); the base case is when ys has 1 element.)

In this question you may assume without proof the following facts
for all x, y, z:Int and xt:[Int]

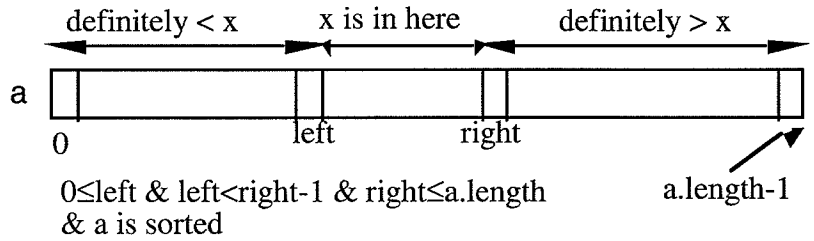
(1) `(min x y)` returns the smaller of x and y, or either if x=y;
(2) `(min x y)` $\leq$ x and `(min x y)` $\leq$ y;
(3) `min` is associative: `min x (min y z) = min (min x y) z`;
(4) if in(x, (y:xt)) then either x=y or in(x, xt);
(5) if x=y then in(x, [y]) and if x$\neq$y then $\neg$in(x, [y]).

The two parts carry, respectively, 50% , 50% of the marks.

3 Consider the following code that implements binary chop to find an element x *known to be present* in the sorted array a.

```
int binChopPresent( int [] a, int x) {
//Pre: ∃i:Nat(i<a.length & a[i]=x) & a is in ascending order
//Post: a=a0 & 0≤r<a.length & a[r]=x & ∀j:Nat(j<r→a[j]<x)
  int left, mid;
  if (a[0]==x) return 0; else left=0;
  int right = a.length;
  //Variant: right-left-2
  //Invariant:∀i:Nat(i≤left→a[i]<x & right≤i<a.length→a[i]>x)
  //              & 0≤left<right-1<a.length & a=a0
  while (right-left>2) {
    mid = (right+left) / 2;  //left<mid<right
    if (a[mid] < x) left = mid;
    else if (a[mid]>x) right = mid;
    else return mid;       // (*) (Part b)
     }
  return left+1;
  }
```

The diagram below shows the computation state of an arbitrary iteration of the while loop of binChopPresent.



a i) For the inputs x=1 and a= | 0 | 1 | 1 | 1 | 1 |

what value is returned by binChopPresent?

ii) What value *ought* to be returned for these inputs to make the postcondition true? *Briefly explain your answer.*

b i) Replace the statement "**return** mid" by a second **while** loop (i.e. at (*)), so that the code *will* always establish the postcondition.
Make sure to include suitable invariant.

ii) Explain why, when the code given in your answer to part (bi) is included in binChopPresent, the postcondition is achieved in all cases.

(You do not need to show that the invariant for either loop is reestablished.)

c Show *carefully* that the invariant of the first while loop is established just before the first while statement.

The three parts carry, respectively, 15%, 55%, 30% of the marks.

4a Warshall's algorithm can be used to compute whether or not a path exists between any two nodes in a graph. A variant of the algorithm, called shortestP, which computes *the number of arcs in the shortest path* between any two nodes in a given graph, is shown below. The input is a suitable encoding of the arcs of the graph in an array a of type int [] []. You should assume that the length of the shortest path from any node to itself is 0.

i Give a suitable precondition (in logic) and *justify your answer* carefully.

ii Give the missing code at (*).

iii Use the invariant and failed while test to show that the postcondition is achieved.

```
int [ ] [ ] shortestP(int [ ] [ ] a) {
  //pre:                              (to be filled in)
  //post: a=a0 & ∀i,j:Nat(i<a.length & j<a.length →
  //     r[i][j] = length of shortest path in a between i and j if
  //              one exists, otherwise r[i][j] = a.length)
 int [ ] [ ] p = (int [ ] [ ]) a.clone();  // copies a into p
 int n = 0;
  while (n<a.length) {
    //invariant : a=a0 & 0≤n≤a.length & a.length = p.length &
    //  p[i][j] = length of shortest path in a between i and j
    //  using transit nodes in {0,1... n-1} if one exists,
    //  otherwise p[i][j]=a.length.
    //variant:   a.length-n
        for (int i=0; i<a.length; i++)
          for (int j=0;  j<a.length; j++) {
            //                          (to be filled in (*))}
        n++;
     }
  return p;
  }
```

b The method isUnique, which determines whether or not all elements in *sorted array* a are different, is to be implemented by a "for loop". The specification and outline structure of the method are given:

```
boolean isUnique(int [ ] a) {
  //pre: ∀i:Nat (i<a.length-1 → a[i]≤a[i+1])
  //post: a=a0 & r<->¬∃i,j:Nat(i<j<a.length & a[i]=a[j])
  boolean unique = true;
  for (int k = 0; k < a.length-1; k++)
  //      (remaining code - to be filled in)
  }
```

i) Fill in the missing code, which will include the body of the for loop and any necessary finalisation after the loop has terminated.
Do not alter the given code.

ii) Show carefully that, for any input that satisfies the precondition, the method achieves the postcondition.

Parts a, bi and bii carry, respectively, 50%, 15% , 35% of the marks.

