UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

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

Friday 14 May 2004, 10:00
Duration: 90 minutes
(Reading time 5 minutes)

*Answer THREE questions*

Paper contains 4 questions
Calculators not required

1 a For the function `hdiv`, show by induction on `n` that for all $n \geq 0$:

$$2 \times (\texttt{hdiv n}) = \begin{cases} \texttt{n} & : \text{if n is even} \\ \texttt{n} - 1 & : \text{if n is odd} \end{cases}$$

```
hdiv :: Int -> Int
-- Pre-condition: n >= 0
hdiv n
   | n < 2      = 0
   | otherwise = 1 + (hdiv (n-2))
```

b Prove by induction that `helios n` terminates for all $n > 0$. (You may use the result from part (a) as necessary).

```
helios :: Int -> Int
-- Pre-condition: n > 0
helios 1 = 0
helios n
   | mod n 2 == 0  = helios (n + 1)
   | otherwise     = helios (hdiv n)
```

c Given the following definition of `IntTree` and the two functions `joinTree` and `countEnds`, prove by induction on `ts` that:

for all `ts` :: `IntTree`, for all `vs` :: `IntTree`:

$$\texttt{countEnds (joinTree ts vs)} = (\texttt{countEnds ts}) + (\texttt{countEnds vs}) - 1$$

```
data IntTree = ITnode IntTree Int IntTree | ITempty

countEnds :: IntTree -> Int
countEnds ITempty = 1
countEnds (ITnode lhs i rhs)
      = (countEnds lhs) + (countEnds rhs)

joinTree :: IntTree -> IntTree -> IntTree
joinTree ITempty vs = vs
joinTree (ITnode lhs i rhs) vs
      = (ITnode lhs i (joinTree rhs vs))
```

*The three parts carry, respectively, 30%, 30%, and 40% of the marks.*

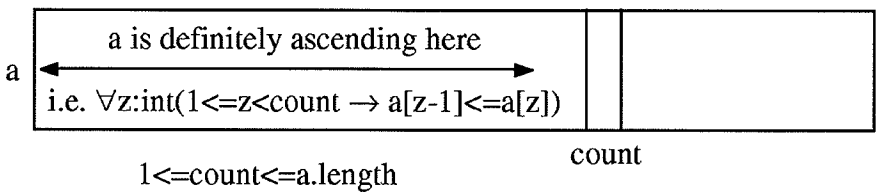
2 This question concerns the method `peak`, which returns the first position in a given sequence of integers at which the elements start to decrease, if any, and the length of the sequence otherwise. The method is specified and given below. You may assume that the array `a` is unchanged by the method.

```
int peak(int [] a) {
//Pre: a.length >= 1
//Post: ((1<=r<a.length & a[r-1]>a[r]) or r=a.length) &
//        (A)z:int(1<=z<r -->a[z-1]<=a[z])
//   i.e. a is in ascending order up to r-1 and decreases to a[r]
//          or a does not decrease at all
int count = 1;
while ((count<a.length) && (a[count-1]<=a[count])) {
   //Invariant see part (b)
   //Variant: a.length - count
   count++;}
return count;}
```

a What results are obtained for the following arrays?
Explain your answer with reference to the postcondition of `peak`.

$$\texttt{a} = [1]$$
$$\texttt{a} = [1, -2]$$
$$\texttt{a} = [1, 3, -2, 2, -3]$$

b Using the diagram below as a guide, write down a suitable invariant for `peak`.




c Show that the invariant is established before the first iteration of the while loop.

d Show that the postcondition is established *at the end* of the method.

e Show carefully that the invariant is re-established by the while loop.

*The five parts carry, respectively, 15%, 10%, 15%, 25%, and 35% of the marks.*

3 This question concerns the method `partition`, given below, which re-orders a portion of an array of integers by means of swapping array elements, according to the following postcondition (where **r** is the result):

$$perm(a0, a) \wedge start \leq \mathbf{r} \leq rest$$
$$\wedge \forall i : int(start \leq i < \mathbf{r} \rightarrow a[i] < x)$$
$$\wedge \forall i : int(\mathbf{r} \leq i < rest \rightarrow a[i] \geq x)$$
$$\wedge \forall i : int((0 \leq i < start \vee rest \leq i < a.length) \rightarrow a0[i] = a[i])$$

For the remainder of the question, you may assume that the properties $\forall i : int((0 \leq i < start \vee rest \leq i < a.length) \rightarrow a0[i] = a[i])$ and $perm(a, a0)$ are satisfied by the method.

```
int partition (int [] a, int x, int start, int rest) {
// Pre: 0<=start<=rest<=a.length
// Post:as given in the question
int greyStart = start ; int bigStart = rest;
// Variant: bigStart-greyStart
while (bigStart > greyStart) {
   if (a[greyStart] >= x) {
      swap(a, greyStart, bigStart-1); bigStart--; }
      //swaps a[greyStart] with a[bigStart-1]
   else greyStart++;}
return bigStart;}
```

a i) Draw a diagram to show a snapshot of the state of `partition` at the start of an arbitrary iteration of the while loop.

ii) Use the diagram to give a suitable invariant for the while loop.

b Show carefully that the invariant is re-established by the while loop *in the case a swap is made.*

c Let `a` be an array containing integers in the range $[0, 100]$ (i.e. $\geq 0$ and $\leq 100$). The method `partition` could be used to partition `a` into three parts, elements in the range $[0, 34)$ (i.e. $\geq 0$ and $< 34$), elements in the range $[34, 67)$ and elements in the range $[67, 100]$.

i) Give code statements that achieve this partitioning, using `partition`, such that the start indices of the ranges $[34, 67)$ and $[67, 100]$ are, respectively, stored in `int` variables `p` and `q`.

ii) Use the postcondition of `partition` to justify carefully why your code achieves the required result.

*The three parts carry, respectively, 25%, 35%, and 40% of the marks.*

4a Let a be an array of length 20. With respect to the *array-as-list* representation:

Which elements of a are referred to by a(5 to 10)?
Express a in this representation.

b A Java method `rotateL` that rotates an array of integers one place to the left is specified below and an implementation using a while loop is also outlined.

```
void rotateL(int [] a) {
//Pre: a0.length>0   //Post: a=a0(1 to a0.length)++a0[0]
   int s=0; int store=a[0];
   while (s<a.length-1) {//Variant: a.length-s-1
   //Inv: 0<=s<a.length & a0(1 to s+1) = a(0 to s) &
   //a0(s+1 to a0.length)=a(s+1 to a.length) & a.length=a0.length
   //Loop code and finalisation code you fill it in}
```

i) Give the loop code and finalisation code of `rotateL`.

ii) Show the invariant is true just before the first iteration of the while loop. Show that the post-condition is established by the end of `rotateL`.

c The Haskell function `reduce` can be used to find the number of times `v` is contained in `t` (i.e. `div t v`) and is given below in terms of the tail recursive `trReduce`, which uses repeated subtraction. A corresponding Java implementation `jReduce` could use a while loop and an outline is given below.

```
reduce::Int->Int->Int
--pre: t>=0 & v>0
reduce t v = trReduce 0 t v
trReduce::Int->Int->Int->Int
--pre: t>=0 & v>0
trReduce n t v
  |t<v   = n
  |t>=v  = trReduce (n+1) (t-v) v
```

```
int jReduce(int t, int v)
{//Pre same as for trReduce
{//Post: r = trReduce 0 t0 v
   int count;   //Initialisation Code-- you fill it in
   while // Invariant, Loop Test and code - you fill it in
   // Finalisation code - you fill it in}
```

i) Complete the code of `jReduce`, including the initialisation, loop test, loop code and finalisation code.

ii) Give an invariant for `jReduce` using `trReduce t, t0, v` and `count`. Show the invariant of `jReduce` is re-established by the while loop code.

*The three parts carry, respectively, 10%, 40%, and 50% of the marks.*