UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2001

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

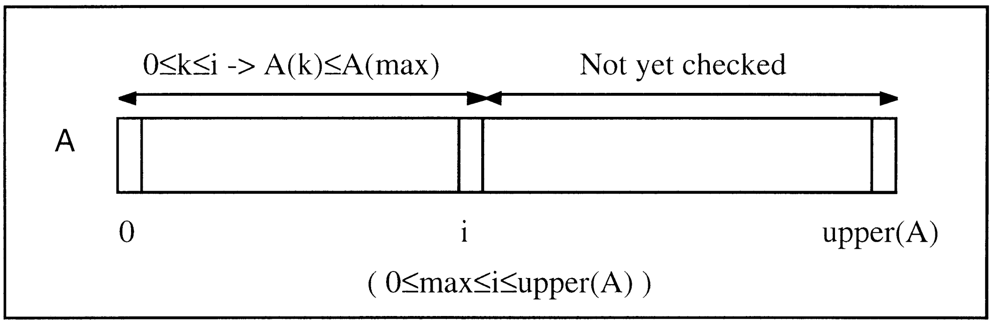
Thursday 3 May 2001, 16:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions
Calculators not required

1 Consider this specification of procedure Max to compute the *index* of the largest integer in an unordered array of integers by a simple linear search, together with its outline implementation using the Turing **loop** construct:

```
procedure Max(A:array 0..* of int,var max : int)
%pre: lower(A)=0
%post: 0≤max≤upper(A) &
%       ∀k:Nat(if k ≤ upper(A) then A(k)≤A(max))
var i:int := 0
max:=0
%above is the initialisation code to establish invariant (part b)
%invariant:  (part a)
%variant: upper(A)-i
loop exit when i ≥ upper(A)
    %variant >0
    i:=i+1
    %re-establish invariant  (parts c and d)
end loop
%post-condition established (part e)
end Max
```



a The diagram above shows the state of the computation at the start of an arbitrary iteration of the loop. Using it as a guide, write down a suitable invariant for the loop.

b Show that the initialisation code establishes the invariant.

c Write down the missing code to re-establish the invariant.

d Show carefully that the code given in part (c) reestablishes the invariant.

e Show that the post-condition is established when the end of the procedure is reached.

The five parts carry, respectively, 15%, 15%, 15%, 40%, 15%, of the marks.

2 This question is concerned with the Haskell function `in`:

```
in :: Char -> [Char] -> Bool
--pre: none
--post: in c xs <-> ∃m,n:[Char] (m ++[c]++n = xs)
in c [] = False
in c (x:y)
   | c == x      = True
   |otherwise   = in c y
```

a State the principle of list induction.

b Show by list induction on xs that for any c:Char and any ys:[Char]

`∀xs:[Char]((in c (xs ++ ys)) = (in c xs) || (in c ys))`

where `||` is the Haskell "or" operator.

You may use the following facts about ++,

for all lists xs and ys:[Char] and x:Char

```
(x: (xs ++ ys)) = (x:xs) ++ ys
[ ] ++ xs = xs
(x:xs) = [x] ++ xs
```

c Show by list induction on xs that `(in c xs)` satisfies its post-condition. That is, show `∀xs:[Char](∀c:Char`
`((in c xs)<->∃m,n:[Char]m++[c]++n=xs))`

Hint:
You may need to show the equivalence of `∃m,n:[Char]m++[c]++n=xt` and `∃m,n:[Char]m++[c]++n=(x:xt)` for $x \neq c$; use of a suitable diagram may help for this.

The three parts carry, respectively, 10%, 45% , 45% of the marks.

3 a The following Turing function Find computes the least index in the sorted array A of any value $\geq$x, if any.

function Find (x:**int**, A:**array** 0.. * **of int**):**int**
%pre: lower(A)=0 & A is sorted: ∀i,j:Nat(i≤j≤upper(A) -> A(i)≤A(j))
%post: ∀i:Nat(i<result -> A(i)<x) & ∀i:Nat(result ≤i<upper(A)+1-> A(i)≥x)
% & 0≤result≤upper(A)+1

Answer the following questions about Find.

i) Assuming that x is ≤A(upper(A)), draw a picture to illustrate the properties of the post-condition.

ii) Use the pre- and post-condition of Find to show carefully that Find(x,A) ≤ Find(x+1,A). (**Hint**: Use proof by contradiction.)

iii) How many times does x occur in A if Find(x,A)=Find(x+1,A)? (There is no need for a proof.)

b The procedure Divide is specified below.

procedure Divide(**var** A: **array** 0 .. * **of** 1..3, S,R:**int**, x:1..3, **var** K:**int**)
%pre: lower(A)=0 & 0 ≤S<R≤upper(A)+1
%post: A is a rearrangement of A0 (A0 is the original array)
% & ∀k:Nat(S≤k<K -> A(k)<x) & ∀k:Nat(K≤k <R -> A(k)≥x)
% & S≤K≤R &∀k:Nat(k<S -> A(k)=A0(k))
% & ∀k:Nat(R≤k <upper(A)+1 -> A(k)=A0(k))

i) Describe in English the properties of the post-condition of Divide.

ii) Let C be any array of integers in the range 1..3. Explain how you could use the procedure Divide (at most 2 calls) to sort C into ascending order.

iii) Justify your answer to part bii) by using the properties of the post-condition of Divide.

The two parts carry, respectively, 40%, 60% of the marks.

4 a The following tail recursive Haskell function isPrefix checks whether xs is a prefix of xt:

```
isPrefix :: [Char] -> [Char] -> Bool
--post z=true iff xs is a prefix of xt, where z = isPrefix xs xt
isPrefix [] xt       = True
isPrefix (x:xs) []   = False
isPrefix (x:xs) (y:xt)
    | x /=y          = False
    | otherwise      = isPrefix xs xt
```

i) The Turing function TisPrefix is supposed to mimic the Haskell function isPrefix. Complete the code and invariant at the lines marked (C) and (I). (Assume all characters in A and B are non-null.)

```
function TisPrefix(A,B:array 0 .. * of char(1)):boolean
%pre: lower(A)=lower(B)=0
%post: result  = isPrefix A(0 to upper(A)+1) B(0 to upper(B)+1)
var i:=0
loop
% variant = upper(A)+1-i
%invariant in terms of isPrefix      (I)
exit when i >upper(A) or i >upper(B)
%rest of code ...                         (C)
end loop
if i > upper(A) then result true else result false end if
end TisPrefix
```

ii) Show carefully that the invariant is maintained by the loop code.

b Given the specified procedure Swap, the following code fragment is supposed to find, in variable *s*, the smaller of the 2 integers *x* and *y*.

```
          (x=x0 & y=y0)
if x> y
    then Swap(x,y)          (*1)  (x≤y & x=y0 & y=x0)
    else                    (*2)
end if
                            (*3)
s:=x                        (*4)  (s=x0 & x0≤y0) or (s=y0 & y0≤x0)

procedure Swap(var X,Y:int)
    %post: (X=Y0 & Y=X0)
```

i) Assuming that the procedure Swap is as specified, add appropriate mid-conditions at the places marked (*2) and (*3) in the given code.

ii) Show carefully that the mid-conditions at (*1) and (*4) are true when they are reached and that the value of *s* produced by the code is the smaller of the given integers *x* and *y*.

The two parts carry, respectively, 55% , 45% of the marks.

