UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2000

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
BEng Honours Degree in Mathematics and Computer Science Part I
MEng 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

Monday 8 May 2000, 16:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions

1a Let a Haskell datatype to represent signed lists of numbers be defined as follows:

```
data Seq  =  Nothing | Pos Int Seq | Neg Int Seq
```

State a principle of structural induction for `Seq`.

b Consider the following Haskell functions

```
add :: Seq -> Int
add Nothing = 0
add (Pos x s) = (add s) + x
add (Neg x s) = (add s) - x

negate :: Seq -> Seq
negate Nothing    = Nothing
negate (Pos x s) = Neg x (negate s)
negate (Neg x s) = Pos x (negate s)
```

Prove by structural induction that for all `s` of type `Seq`,

```
add s  =  - add (negate s)
```

c Let the function f be defined as follows:

```
f :: Int -> Int -> Int
--pre: both arguments are non-negative
f x y    | x==0          = y
         | y==0          = x
         | otherwise     = f x (y-1) + f (x-1) (y+2)
                                     + f (x-1) (y+3)
```

Using either double induction or well-founded induction over a suitable clearly-stated ordering, show that for all integers `x,y ≥ 0,  f x y` terminates and returns a number with the same parity (odd or even) as `x+y`.

The three parts carry, respectively, 20%, 40%, 40% of the marks.

2 This question asks you to develop a Turing procedure

```
procedure Replace(x, y: int, var A: array 0..* of int)
```

that replaces every occurrence of `x` in the array `A` by `y`.

a Write down a formal pre- and post-condition for `Replace`, in logic.

b Write the body of `Replace`, using a `loop` construct (not a `for` loop). Include a loop variant and invariant as comments.

c Show that the loop code re-establishes the loop invariant. Remember to check that all array accesses are legal.

d Now suppose that we are given a *function*

```
function Freplace(x,y:int, var A:array 0..* of int):int
```

`Freplace` has the same effect on A as `Replace`, and also returns a result defined informally by:

r is the largest index `i` between `lower(A)` and `upper(A)` such that `A(i)≠A0(i)`, and **r** = `upper(A)+1` if there is no such index.

i) Write down a formal post-condition for `Freplace` in logic.

ii) With brief justification, explain what the value of `ans` will be, after running the following program fragment:

```
Replace(x, y, A)
ans := Freplace(x, y, A)
```

The four parts carry, respectively, 15%, 30%, 30%, 25% of the marks.

3a Consider the following recursive function oddsum.

```
oddsum :: Int -> Int
--pre: argument is non-negative

oddsum n    | n==0          = 0
            | otherwise     = 2*n - 1 + oddsum (n-1)
```

i) Write down a tail-recursive function trOddsum with an accumulating parameter that, when called with suitable arguments, serves to calculate oddsum. You may write trOddsum in either Haskell or Turing.

ii) What arguments must you give the trOddsum function to calculate oddsum n?

iii) Prove by induction on n that with the arguments specified in a(ii), trOddsum does calculate oddsum. Warning: you will have to formulate an inductive hypothesis to handle arbitrary values of the accumulating parameter.

b Write the Turing code of a function lpOddsum that calculates trOddsum by using a loop, without recursion. Do not forget to include a pre-condition, post-condition, loop variant, and loop invariant as comments.

c i) Show that the loop code in lpOddsum that you wrote in part c re-establishes the invariant.

ii) Use this to show that lpOddsum does produce the same result as trOddsum when given any arguments that meet its pre-condition.

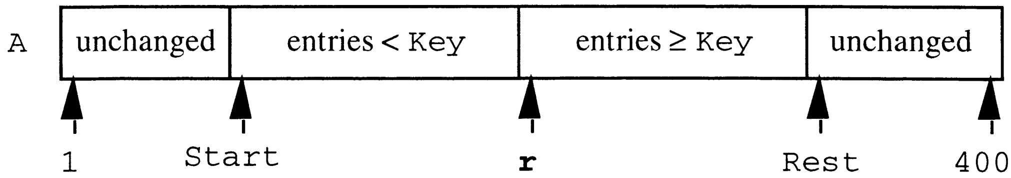
4 In this question, you are to develop a Turing procedure to sort an array `A` of integers, with `lower(A)=1` and `upper(A)=400`. Your procedure will use Quicksort and will need to sort regions of `A`, from `Start` up to but not including `Rest`. Here is the program header:

```
procedure Sort(var A:array 1..400 of int; Start,Rest:int)
```

You are given a function

```
Partition(A,Start,Rest,Key)
```

that does a crude "midwives" sort of `A` and returns an **`int`** value; the diagram below shows how `A` looks after `Partition` has terminated, and the result **`r`** returned:



You may use a library procedure `Swap` to interchange two elements of `A`.

a Write down the pre-conditions and post-conditions of `Sort` and `Partition`.

b Write the code of `Sort`. Include a recursion variant.

c Outline an argument by course-of-values induction on the recursion variant to show that `Sort` meets its post-condition. You may assume that `Partition` meets its post-condition when called with arguments meeting its pre-condition.

d Prove using natural deduction that for any array `A:1 to 400 of` **`int`** meeting the post-condition of Sort, the following sentence is true:

$$A(1) = A(400) \rightarrow \forall i,j{:}\mathrm{int}(1 \le i \le 400 \wedge 1 \le j \le 400 \rightarrow A(i)=A(j)).$$

The four parts carry, respectively, 30%, 20%, 30%, 20% of the marks.

