UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1997

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 Royal College of Science

PAPER MC1.7

TURING PROGRAMMING AND DATA STRUCTURES

Friday, May 2nd 1997, 10.00 - 11.30

Answer THREE questions

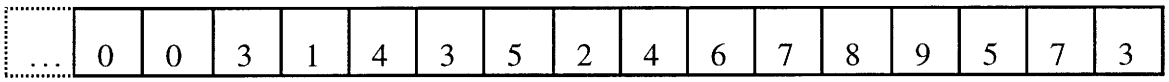
For admin. only: paper contains 4 questions

Section A *(Use a separate answer book for this Section)*

1 The standard type `int` in Turing cannot hold numbers as large as 2^{31}. This question is about the implementation of a type `bignum` which can be used when very large non-negative integers are required. The declarations below are to be used throughout this question. Although the elements of the `bignum` array type are given as `int`s you can assume that they are `int`s within the range

```
const top:= 100
type dignum : 1..top
type digit : 0..9
type bignum : array dignum of digit
```

So the number `31435246789573` will be held in a variable of type `bignum` like this:



a Write a function `zero` which returns a `bignum` with value `0`.

b Write a procedure `putnum` which puts a `bignum` on the screen. If the `bignum` is `0` put `0` on the screen. Otherwise your procedure must omit all leading zeros.

c Write a function `inc` which adds `1` to a `bignum`. If every digit of the `bignum` is a `9` then the function should return `0`. You may find it easier to write the function if you give it an extra argument, an index into the `bignum`.

d Write a function `add` which adds two `bignums`. You can assume that the two `bignums` to be added will result in a `bignum` that doesn't overflow. You may find it easier to first write a subsidiary procedure that has as its parameters: the two numbers to be added, the result so far, a carry digit and the current index.

e Write a function `fib` that takes as input a non-negative integer `n` and returns a `bignum` representation of the n^{th} Fibonacci number. You may use functions and procedures defined in earlier parts of this question if you so wish.

The five parts carry, respectively, 10%, 20%, 20%,40% and 10% of the marks.

2 This question is about the implementation of a stock control system. The declarations below are to be used throughout this question.

```
const maxi:= 10000
type item : record
                name : string(10)
                price : int %price in pence
                quantity : int %number in stock
                restock : int
            end record
type stocknum : 1..maxi
type stock : array stocknum of item
```

The `price` field contains the price of the `item` in pence. For a given item the `quantity` field contains the number of items available in stock. The `restock` field contains the minimum number of the item that needs to be maintained in stock. So

"milk"	92	50	40

is the record for milk. A single item of milk costs £0.92, there are 50 items of milk in stock and when the stock level gets down to 40, milk needs to be restocked.

a Write a function `value` which takes as its input parameter a variable of type `stock` and returns as its result the total value of all the items in stock in pence. The value of a single item is its price.

b Write a procedure `order` which takes as its input parameter a variable of type `stock` and prints on the screen the names of all items that need to be restocked.

c i Write a procedure `putitem` that prints the name of an item followed by its price in pounds and pence. So

`putitem(i)` would print `milk .92` , if item `i` is milk

ii Write a procedure `tillslip` which reads in stock codes and prints out an itemised receipt or tillslip which consists of the `name` of each item purchased, followed by the `price` in pounds and pence. The stock code for an item is just its index in the array of type `stock`. To terminate the list of items a `0` should be entered. The bottom of the tillslip should contain the total cost of the purchases. You can assume that the reading in of stock codes does not print any number on the screen and that an actual piece of paper is not printed, rather the tillslip appears on the screen.

iii How would you alter procedure `tillslip` to update the stock levels while it prints the slip?

The three parts carry, respectively, 25%, 25%, and 50% of the marks.

Turn over ...

Section B *(Use a separate answer book for this Section)*

3 a Define the Abstract Data Type *List* and give the function header and the pre-and post-conditions for the access procedures.

b Give *type declarations* for the ADT List in Turing for

i) a dynamic implementation

ii) a static implementation

c Draw diagrams showing how the list (a,b,c,d) is represented in the data structures given in your answers to part b, explaining your notation.

d Write the following higher level function in Turing

```
function Alternate (L : List) : List
% post : returns a list containing the first, third, fifth, ..... items of L.
```

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

4 An Abstract Data Type *Admissions* is to be set up for a secondary school, to deal with candidates applying for entry.

For each secondary school, an Admissions structure will be held containing the following data for each of its candidates: First name, surname, date of birth, primary school name, distance of primary school from secondary school.

The ADT is to help manage the process of following the declared order of priority for admission to the secondary school, which is as follows:

1) pupils from primary schools in order of distance from the secondary school, nearest first.

2) for pupils from the same primary school, in order of age, oldest first. Only pupils born during period 1 September 1985 and 31 August 1986 inclusive are to be considered.

The access procedures are as follows:

function NewSchool (S : SecSchool) : Admissions
% post : initialise a new Admissions structure for secondary school S

function AnyWaiting (A : Admissions) : boolean
% post: returns true if there are any candidates in A, false otherwise

procedure AddCandidate (C : Candidate, var A : Admissions)
% pre : takes a Candidate C and Admissions A
% post : adds C to A in the correct position

function NexttoOffer (A : Admissions) : Candidate
% pre: takes an Admissions structure A
% post: returns Candidate with highest priority

procedure OfferPlace (var A : Admissions)
% pre : takes Admissions A
% post: prints letter offering school place to highest priority candidate and removes him/her from A.

In this question, you are asked to implement the ADT Admissions dynamically. You may assume the availability of the standard ADTs *List* and *Queue;* other than the access procedures for *List* and *Queue*, other procedures should be given in full.

a Give an example of your data structures using diagrams. Explain any notation you use.

b Give data declarations in Turing code for your data structures.

c Write Turing code for the access procedures for ADT Admissions given above.

The three parts carry, respectively, 25%, 30% and 45% of the marks.

End of paper