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 C121=MC121

PROGRAMMING

Friday 12 May 2000, 14:30
Duration: 120 minutes

Answer FOUR questions

Paper contains 6 questions

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

1 This question is about the implementation of a stock control system for a toy shop. The following declarations are to be used.

```
const maxi := 1000
type item:record
        code:string(5)
        description:string
        price:int           % selling price in pence
        quantity:int        % number in stock
        minquantity:int     % reorder level
        reorder:int         % reorder amount
        cost:int            % cost (p) when reordering
    end record
type stock:array 1..maxi of item
type codeQuantity:record
        code:string(5)
        quantity:int
    end record
```

The `code` field of `item` is a unique identifier for each type of item sold.

"D9711"	"LoveMe doll"	1495	57	40	20	968

represents that item D9711 (LoveMe doll) is sold for £14.95, there are 57 such items currently in stock, and when the stock level gets down to 40 or below, 20 more D9711 items will be ordered from the supplier at a cost of £9.68 each.

There is also a type `codeQuantityList` for lists of elements of type `codeQuantity`. The details of this list type are not required for the question.

a Write a Turing function `value` that takes an argument of type `stock` and returns the total value in pence of all items in `stock`. The value of a single item is its selling price.

b Write a procedure `incQuantity` that takes as arguments an argument of type `stock`, an item code (i.e. an argument of type **`string`**`(5)`) and an integer, `q` say, and increments the value of the `quantity` field of the item with code `c` by `q`. You may assume that there is at most one item in `stock` with a given code. The procedure should also return a boolean argument with value `true` if the quantity is incremented successfully and `false` if no item with the given code appears in the `stock` array.

c Write a function `reorderList` which takes an argument of type `stock` and returns a value of type `codeQuantityList` representing the items whose stock is at or below the minimum level and, for each such item, the quantity that needs to be re-ordered.

It is *not necessary* to provide an implementation of the list data type `codeQuantityList`. However, you should identify *clearly* the access functions and procedures that you employ.

d Write a procedure `updateStock` with two arguments, the first of type `stock` and the second of type `codeQuantityList`.
The `codeQuantityList` argument represents a delivery list: a list of records showing the items and quantities that have been received and are to be added to the current stock. `updateStock` should update the `quantity` fields of `stock` for each record in the delivery list (i.e., for each item on the delivery list, add the quantity to the current value of `quantity` in `stock`). The procedure should output an appropriate error message on screen for any record on the delivery list where the quantity value is not a positive integer or where the item code is not contained in array `stock`.

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

2 In a substitution cipher, the encrypted message (the 'ciphertext') is obtained by replacing every character of the original message (the 'plaintext') by its corresponding character in a given 'cipher alphabet'. For example:

plain alphabet:	`abcdefghijklmnopqrstuvwxyz`
cipher alphabet:	`JLPAWIQBCTRZYDSKEGFXHUONVM`

plaintext	`more paper please`
ciphertext	`YSGW KJKWG KZWJFW`

By convention, plaintext is written in lower case and ciphertext in upper case.

a Suppose that cipher alphabets are represented using the following Turing type:

```
    type cipher : array 'a'..'z' of 'A'..'Z'
```

Write a Turing procedure

```
  encryptFile( infile, outfile:string, subst:cipher )
```

that reads the text in file `infile` character by character, encrypts it using the cipher alphabet `subst`, and writes the ciphertext to file `outfile`.

Any upper case characters in the plaintext should be converted to lower case before encryption. Any other characters in the plaintext outside the range `'a'..'z'` should be written to the ciphertext without change.

The primitives in Turing for file handling are:

```
open:fileHandle, fileName, mode     where mode is get or put
get:fileHandle, items               put:fileHandle, items
close:fileHandle                    eof(fileHandle)
```

b A common way of producing a cipher alphabet is by means of a keyphrase. For example, to use `BORING LECTURE` as a keyphrase, ignore all spaces and repeated characters (`BORINGLECTU`), use this as the beginning of the cipher alphabet, and then continue with the remaining unused characters of the plain alphabet, in their correct order, starting where the keyphrase ends. For the example, the generated cipher alphabet would be as follows:

plain alphabet:	`abcdefghijklmnopqrstuvwxyz`
cipher alphabet:	`BORINGLECTUVWXYZADFHJKMPQS`

Write a Turing function

```
          keyphraseCipher( key:string ): cipher
```

that generates a cipher alphabet from the keyphrase `key`. You may assume that `key` is a non-empty string containing only characters in the range `'A'..'Z'`.

You may find it helpful to define a boolean function `inArray(c:char, sub:cipher, top:'a'..'z')` which returns `true` when character `c` is one of the elements in the array `sub` between `'a'` and `top`.

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

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

3 a Define the Abstract Data Type *Binary Search Tree*, and describe the additional properties possessed by a Binary Search Tree which is also an *AVL-Tree*.

b Compare binary search tree and AVL-Tree data structures for ease of maintenance and for performance in retrieving an item of data. Illustrate your answer with an example in which the tree structure is built using the following sequence of integer values: 50, 20, 10, 30, 80, 70, 5, 15.

c List the function headers and pre- and post-conditions for access procedures for the Abstract Data Type AVL-Tree.

d Write in Turing

i) the data declarations for a dynamic implementation of the ADT AVL-Tree.

ii) the code for the higher-level function:

```
function IsAVL(T:BSTree) : boolean
%pre: takes a binary search tree, T
% post: returns TRUE if T satisfies the structure requirement for an
%           AVL-Tree
```

You may assume availability of the standard access procedures for the Abstract Data Type Binary Search Tree. Any further functions or procedures must be written in full.

The four parts carry, respectively, 25%, 25%, 20%, 30% of the marks.

4 This question is about the implementation of an Abstract Data Type *Family* . A Family contains individuals of type Person, which holds details of surname, first name and date of birth.

The access procedures have been defined as follows:

```
function NewFamily(A:Person) : Family
%post: returns a new family structure containing only Person A.

procedure AddChild(P,C:Person, var F:Family)
%post: adds child C of person P in Family F.
%      Gives a message if P is not found in F.

procedure AddSibling(P,S:Person, var F:Family)
%post: adds sibling S of person P in Family F.
%      gives a message if P is not found in F.

function Children(P:Person, F:Family) :List
% pre:   takes a Person P and a Family  F
% post:   returns a list of P's children in F

function Siblings(P:Person, F:Family) : List
% pre:   takes a Person P and a Family F
% post: returns a list of P's siblings in F
```

(Note that a sibling is either a brother or a sister).

a Give the data declarations in Turing for an implementation of the ADT Family.

b Write the Turing code for the access procedures defined above.

You may assume the availablity of the ADT List with data type Person, and standard access procedures. Any additional functions must be given in full.

c Explain the working of your answer to part b. This may be by use of examples or by careful commenting of the code.

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

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

5 a Describe the basic principles of object-oriented programming and outline its major difference from the principle of procedural programming.

b An **library_account** class has the following methods:

setaccount(X:Int, Y:Int) :	set an account with number X and the value of Y as the number of books X is allowed to borrow
borrow(Y:Int) :	borrow the number Y of books if the borrowing is within the limitation otherwise report error
return_back(Y:Int) :	return the number Y of books.

i) write an OOT program *library_account.tu* to implement the class

ii) write an OOT program to set up two accounts x1 and x2 which can borrow y1 and y2 books respectively and then make the following transactions :

1) x1 borrows N books (N<y1)

2) x2 borrows M books (M<y2)

3) x1 or x2 returns books to equal the number of books they have borrowed.

iii) use inheritance to implement a new class **library_accountnew** by extending the class **library_account** with a new method of :

getbalance(var X : int) : print the current number of books X is allowed to borrow

The two parts carry, respectively, 25% and 75% (25% for each subpart) of the marks.

6 Let **Face** be a class implemented by the program "face.tu" and **happyface**, implemented by the program "hf.tu", is a subclass of the **Face** class. The following procedure in "face.tu" implements the function of drawing a face with eyes on the screen:

```
procedure drawFace
      % Draw head as an oval
      Draw.FillOval(x, y, headRadius, headRadius, faceColour)
      %Draw right and left eyes
      const eyeRadius := headRadius div 10
      const eyeY := y + headRadius div 5
      const eyeToRight := headRadius div 5
      Draw.FillOval(x+eyeToRight, eyeY, eyeRadius, eyeRadius, black)
      Draw.FillOval(x-eyeToRight, eyeY, eyeRadius, eyeRadius, black)
end drawFace
```

where the procedure exported from the OOT's predefined *Draw* module:

```
Draw.FillOval(x, y, xRadius, yRadius, c)
```

draws a filled oval on the screen centred at (x, y) with colour c .

a) Define a new class **sleepyface** which is the subclass of the **Face** class with a new method "sleep" drawing a pair of closed eyes in a face. You may use the following procedure exported from the OOT's predefined *Draw* module to draw a line (as a closed eye) from position of (x1, y1) to the position of (x2, y2) with color c :

```
Draw.Line(x1, y1, x2, y2, c)
```

b) Given a pointer p declared as var p : ^Face, write the possible dynamic classes of p. What type declaration for q would make the assignment q : = p legal ?

c) A generic stack class implemented by the program *GenericStack* contains pointers to objects of any class. Write a program to store three different faces from the different classes (**Face**, **happyface** and **sleepyface**) and then draw the face objects located by the pointers in the stack on the screen

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

End of paper