UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2002

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 MC125=C121R

PROGRAMMING (JMC)

Wednesday 1 May 2002, 14:00
Duration: 120 minutes

Answer FOUR questions

Paper contains 6 questions
Calculators not required

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

1 This question is about the Eight Queens Problem (EQP), which can be described as follows: "find an arrangement for eight queen pieces on an otherwise empty chess board, so that no queen attacks any other".

A queen attacks another piece which is in the same row, or the same column or on the same diagonal on the board. A chess board has 64 squares, arranged as an 8X8 grid. The individual squares in the grid may be represented by a pair of coordinates, (x, y), representing (row, column) with $0 \leq x, y \leq 7$.

There are many possible arrangements of eight queen pieces on the board, of which only a few are solutions to the EQP. Your task is to supply methods to test an arrangement of eight queens to determine whether it is a solution to the EQP, as follows:
You are given the class definitions:

```
class position{
      int xpos;
      int ypos;
      }

class chessBoard{
      char[8][8] b;
      }

class positionList{
      position queenpos[8];
      }
```

Write the following methods in Kenya or Java:

a
```
chessBoard initialiseBoard(){
    //returns a chessBoard, all positions set to ' ' }

positionList findQueens(ChessBoard b) {
    //assert: 8 positions of b are set to 'Q' the rest
    //to ' '.
    //returns a positionList of the (x,y) coordinates
    //of the 'Q's in b
    }
```

b
```
boolean isSameRow(position c, position d) {
    //returns true if queens at positions c and d are
    //in the same row of the board
    }

boolean isSameCol(position c, position d) {
    //returns true if queens at positions c and d are
    //in the same column of the board
    }
```

```
boolean isSameDiag(position c,position d) {
        //returns true if queens at positions c and d are
        //on the same diagonal of the board.
        }
```

c Using your answers to parts b and c, or otherwise, write

```
boolean noAttack(positionList p) {
    //takes a positionList  p of positions of 8 queens
    //returns true if no queen in p attacks any other
    }
```

d Using your answers to parts a and b, or otherwise, write

```
boolean isEQPSolution(chessBoard b) {
    //takes a chessBoard b with 8 positions set to
    // 'Q', the rest to ' '
    //returns true if this is a solution to the EQP.
    }
```

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

2 In this question, you are asked to write methods in Kenya/Java to sort an array of integer values, by two methods, as specified in parts a and b.

You are given the following class definition:

```
class sortedArray {     //the first n positions of the
    int n;              //array s contain n integers,
    int [100] s;        //sorted in increasing order
    }
```

a Insertion Sort: write the following method in Kenya or Java:

```
void insertionSort(int item, sortedArray a) {
    //takes a sortedArray of integers, a, and a new
    //integer, item
    //returns updated a, with item inserted into the
    //correct position.
    }
```

b Merge Sort: write the following method in Kenya or Java.
Note: producing a merged sorted array from two arrays which are separately sorted consists of processing the two arrays in parallel, and copying the values in the correct sorted sequence.

```
sortedArray mergeSort(sortedArray a,b) {
    assert a.n + b.n<100: merged list is too long
    //returns a sortedArray containing the items of
    //a and b, in increasing order.
    }
```

The two parts carry, respectively, 50%, 50% of the marks.

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

3a i) Describe what the Abstract Data Type (ADT) *List* is, and give the *List*'s access procedure headers with post-conditions, also specifying the exception cases.

ii) Consider the ADT *Stack*. Give its access procedures' headers with the post-conditions, also specifying the exception cases.

iii) Give the axioms that any implementation of the ADT *Stack* must satisfy.

iv) Explain the difference between the ADTs *List* and *Stack*.

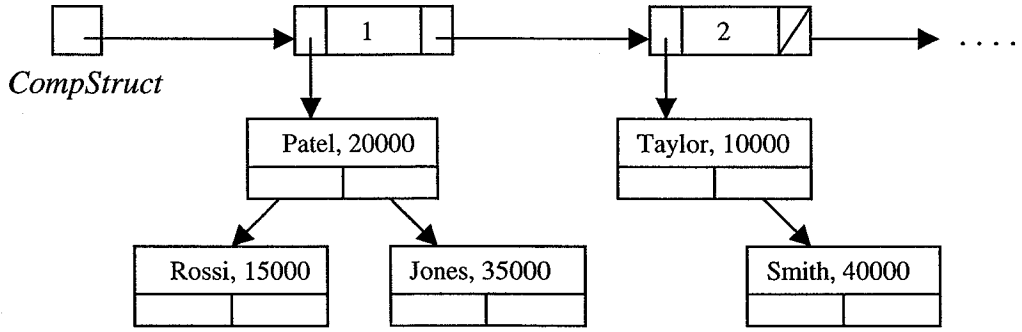
b Assume the availability of an ADT *List* with its own access procedures, and the availability of an ADT *Stack* with its own access procedures Assume the order of access in a list to be from the first to the last item in the list. Write the Java code for the following two high-level access procedures:

i) *Stack* ListtoStack(*List* L)
//pre: L is a list
//post: returns a stack containing the items of L, with the access order identical to
//that of L; L is left unchanged

ii) *List* ReverseStacktoList(*Stack* S)
//pre: S is a stack
//post: returns a list containing the items of S, where the access order of L is the
//reverse of the access order of S.

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

4 The internal organization (*CompStruct*) of a company is composed of ten different sectors, uniquely numbered from 1 to 10. People employed within the same sector are organized into a tree structure, sorted on the basis of their salaries. Within the same sector, employees have all different salaries. For each employee the organization includes his/her surname and salary. The diagram below gives a partial example of such an organization. In sector 1 the employees are Rossi with the lower salary 15000, Jones with salary 20000 and Patel with higher salary 35000. In sector 2, the employees are Taylor with salary 10000 and Smith with higher salary 40000.



a. Assume *CompStruct* to be the definition of an ADT. Give the Java type declarations for this ADT *CompStruct* in full, including the type declarations of all the data structures needed to implement it. Do not encapsulate the declaration of an employee into a separate type.

b. Assume now the existence of a standard Binary Search Tree ADT. Give the Java type declaration for the ADT *CompStruct* using this standard ADT.

c. Write the Java code for the *CompStruct*'s access procedure *EmployeeName* specified below. To do so, use the Java type declaration you have given to question (b) and assume the existence of the Binary Search Tree's access procedure String retrieve(int sal). This procedure returns the name of the employee in a given tree, whose salary is equal to sal.

String EmployeeName(int sectnum, int sal)
//pre: *sectnum* is a sector number, and *sal* is the salary of an employee in this sector.
//post: returns the surname of the employee in the given *sectnum*, whose salary is *sal.*

Give also the implementation of any auxiliary method or other access procedure of *CompStruct* that might be needed.

The three parts carry, respectively, 30%, 20%, 50% of the marks.

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

5a Implement a class *CarRentalCustomer*. Class *CarRentalCustomer* inherits class *Customer* and has the following fields/attributes:

(i) "renting", states whether "this" customer is currently renting a car;

(ii) "rentedCars", an array of type RentalCar for this customer;

(iii) "surname", the customer's surname (inherited from Customer);

(iv) "firstName", the customer's first name (inherited from Customer);

(v) "id", a numerical customer id;

(vi) "rentalTime", an array of the number of rental days for each rented car; and

(vii) "creditCardDetails", a primitive type containing details of the customer's credit card(s).

b Write a suitable constructor method for class *CarRentalCustomer*; it initializes ALL fields and has a first name, surname, and list of type int -- the array for rental times -- as parameters; however, first name and surname are to be initialized by class Customer; you may assume that customers rent no more than 20 cars at a time;

c Write methods for *CarRentalCustomer*:

(i) "status", an abstract method that returns a description of the customer's importance to the rental company, such as `platinum', `advantage gold', etc;

(ii) "deleteCar", an abstract method that deletes a specific RentalCar -- its only parameter -- from rentedCars, if possible;

(iii) "addCar", an abstract method that adds a specified RentalCar to rentedCars, if possible;

(iv) "getBillingInformation", returns the customer's credit card details.

Finally: Where would you put these methods within class CarRentalCustomer? What changes, if any, do you have to make to the code written in the previous part?

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

6a Explain the crucial consequences of

(i) declaring a primitive type (e.g. int) as final;

(ii) declaring a class as final;

(iii) storing a class in folder F and file C;

(iv) declaring a field to be private;

(v) declaring a field to be *protected.*

b Write a Java method *LastEvenIfAny* that has as parameter an integer array and returns the rightmost even entry in that array, provided it exists. Otherwise, it returns -1. For example [12, 4, 3, 8, 9, -2, 5] returns -2 and [3, -7, 11] returns -1. Make sure that your code returns -1 if the array is "empty".

c Consider the following Java source code:

```
public interface ITransaction {
      public double calcTotal();
}

public class RentalTransaction implements ITransaction {
      private double amount;
      public void RentalTransaction(int price) {
            amount = price;
      }
      public int calcTotal(double discount) {
            return amount - discount;
      }
}

public class AuctionTransaction {
      private double amount;
      public AuctionTransaction(double price) {
            amount = (int)price;
      }
      public abstract int calcTotal(double heftyFee) {};
}

public class TransactionRunner {
      public static void main(String[] args) {
            RentalTransaction booking = new RentalTransaction(586);
            ITransaction iBooking = (ITransaction)booking;
            if (iBooking instanceof RentalTransaction) {
                  double myTotal = iBooking.calcTotal(1.0);
            }
            AuctionTransaction auction = new AuctionTransaction(12000000.35);
            double myBill = auction.calcTotal();
      }
}
```

Identify eight errors in this source code that the compiler will detect and complain about. Explain the nature of these errors in each instance that you identify.

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