Paper Number(s): E1.8
E2.18 (E2.7A)

IMPERIAL COLLEGE LONDON

Corrected Copy

DEPARTMENT OF ELECTRICAL AND ELECTRONIC
EXAMINATIONS 2005

EEE Part II / ISE Part I: MEng, BEng and ACGI

SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES

Tuesday 24th May 2005 2:00pm

There are THREE questions on this paper.

Question 1 is compulsory and carries 40% of the marks.

Answer Question 1 and EITHER Question 2 (carrying 60%) or Question 3 (carrying 60%).

This exam is **open book**

Time allowed: 1:30 hours.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible:

First Marker(s): Shanahan, M.P.
Second Marker(s): Demiris, Y K.

Information for Invigilators:

Students may bring any written or printed aids into the exam.

Information for Candidates:

Marks may be deducted for answers that use unnecessarily complicated algorithms.

The Questions

1. Assume the existence of the following data type `TList`, and assume that `TList` has the standard set of access procedures `Empty`, `First`, `Rest`, and `Add`.

```
type
  TList = ^TLink;
  TLink =
  record
    First : integer;
    Rest : TList;
  end;
```

(a) Suppose `L` is a variable of type `TList`. Consider the following lines of code.

```
L := Empty;
Add(3,L);
Add(5,L);
Add(7,L);
Add(9,L);
```

After the execution of this code, what is the value of the following expression?

```
First(Rest(L)) + First(L)
```

[8]

(b) After the execution of the code in (a) what is the value of `L^.Rest^.Rest^.First`? [8]

(c) Consider the following iterative procedure.

```
function F1(L : TList): integer;
begin
  T := 0;
  while L <> Empty do
  begin
    T := T + First(L);
    L := Rest(L);
  end;
  return T;
end;
```

Write a recursive function called `F2` that computes the same thing as `F1`. [8]

(d) Consider the following recursive procedure.

```
function F3(L : TList): integer;
begin
  if L = Empty
  return 0
  else begin
    M := F3(Rest(L))
    if First(L) > M
    then return First(L)
    else return M;
  end;
end;
```

What does `F3` compute? [8]

(e) Write an iterative procedure `F4` that computes the same thing as `F3`. [8]

2. (a) Define a *dynamic* Pascal data type `TOrg` capable of representing the management structure of an organisation. Assume that every member of the organisation except one (the CEO) has exactly one manager, and allow any manager to have arbitrarily many people working directly under them. Use a tree structure in which each node is a string representing a person's name. The CEO will occupy the root of the tree. The leaves of the trees will be occupied by the workers. Everyone who occupies a non-leaf node will be a manager.

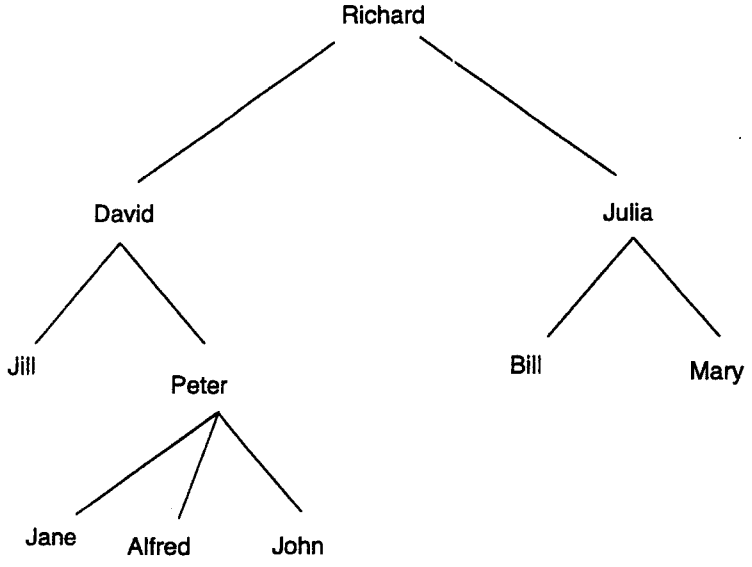

Figure 2.1

Fig. 2.1 shows an example tree. The CEO is Richard. Jill, Jane, Alfred, John, Bill, and Mary are workers. Richard, David, Peter, and Julia are managers. [12]

(b) Using the data structure you defined in (a) write a function `Boss` that takes a tree of type `TOrg` and two names `A` and `B` as arguments and returns True if `A` is `B`'s immediate boss. In Fig. 2.1, for example, Peter is Jane's immediate boss and David is Peter's immediate boss. You may assume all names in the tree are unique. [24]

(c) Using the data structure you defined in (a), write a function `Efficiency` that computes the proportion of workers to managers in a given tree. In Fig. 2.1 there are 6 workers and 4 managers, so the function would return 6/4 or 1.5. [24]

3. (a) Assuming `N` is declared as an integer constant, write a function `FactArray` that returns an array `A` of length `N` such that, for any `i` between 1 and `N`, `A[i]` is equal to the the factorial of `i`. Ensure that your algorithm is not unnecessarily inefficient. Recall that, for m > 0, the factorial of m is m * (m–1) * (m–2) * ... * 1. In terms of `N`, how many multiplications will your procedure perform? [18]

(b) Rather than using pointers, it is possible to use an array to represent a binary tree. Fig. 3.1 shows one way to represent the tree of characters in of Fig. 3.2. The root is the 1st element of the array. Each node occupies three array elements: the first element is the character, the second is the location in the array of the left sub-node, and the third is the location of the third sub-node. An empty sub-tree is denoted by 0 (analogous to the nil pointer).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
B	4	7	A	0	0	D	10	13	C	0	0	E	0	0	

Figure 3.1

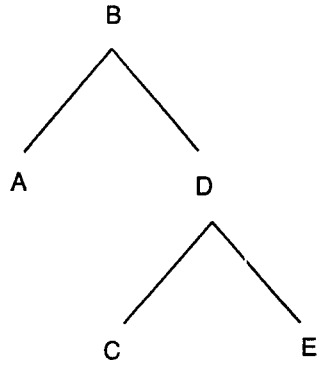

Figure 3.2

Write a procedure `PrintTree` that takes an `N`-element array of integers as an argument and carries out an in-order (left-node-right) traversal of the corresponding binary tree of characters, printing each character on the console as it goes. Assume that the tree is represented in the `N`-element array according to the above scheme. You may assume that characters are represented by their ASCII equivalents and that there is a function `Char` that takes an integer and returns the corresponding ASCII character. [24]

(c) Using the same representational scheme as in (b), write a function `SumTree` that takes an `N`-element array of integers as an argument and returns the sum of the ASCII codes of all the characters stored in the tree. Assume that unused locations of the array are filled with zeros. Note: there is an easy way to do this and a hard way. You will gain most marks for doing it the easy way. [18]

2005

Model Answers

1 (a) [New theoretical application]

16

(b) [New theoretical application]

5

(c) [New theoretical application]

```
function F2(L : TList): integer;
begin
  if L = Empty
  return 0
  else return(First(L)+F2(Rest(L));
end;
```

(d) [New theoretical application]

F3 computes the maximum integer in the list L.

(e) [New theoretical application]

```
function F4(L : TList): integer;
begin
  M := 0;
  while L <> Empty do
  begin
    if First(L) > M
    then M := First(L);
    L := Rest(L);
  end;
  return M;
end;
```

2 (a) [New theoretical application]

```
type
  TOrg = ^TNode;
  TNode =
  record
    Name : string;
    Manages : TSubs;
  end;
  TSubs = ^TLink;
  TLink =
  record
    First : TOrg;
    Rest : TSubs;
  end;
```

(b) [New theoretical application]

```
function Boss(T: TOrg;
  A : string; B : string): boolean;
var L : TSubs;
begin
  if T = nil
  return false
  else begin
    if (T^.Name = A) and member(B,T^.Manages)
    then return True
    else begin
      L := T^.Manages;
      while L <> nil do
      begin
        if Boss(L^.First,A,B)
        then return True
        else L := L^.Rest;
      end;
      return False;
    end;
  end;
end;

function member(N : string; L : TSubs): boolean;
begin
  if L = nil
  then return False
  else if N = L^.First
  then return True
  else return member(N,L^.Rest);
end;
```

(c) [New theoretical application]

```
function Efficiency(T : TOrg): real;
var N, M : integer;
begin
  N := Workers(T);
  M := Managers(T);
  return N/M;
end;
```

```
function Workers(T : TOrg): integer;
var L : TSubs; N : integer;
begin
  if T = nil
  then return 0
  else if T^.Manages = nil
  then return 1
  else begin
    L := T^.Manages;
    N := 0;
    while L <> nil do
    begin
      N := N + Workers(L^.First);
      L := L^.Rest;
    end;
    return N;
  end;
end;
```

```
function Managers(T : TOrg): integer;
var L : TSubs; N : integer;
begin
  if T = nil
  then return 0
  else if T^.Manages = nil
  then return 0
  else begin
    L := T^.Manages;
    N := 1;
    while L <> nil do
    begin
      N := N + Managers(L^.First);
      L := L^.Rest;
    end;
    return N;
  end;
end;
```

3. (a) [New theoretical application]

```
function FactArray: array[1..N] of integer;
var I, T : integer;
  A : array[1..N] of integer;
begin
  T := 1;
  for I := 1 to N do
  begin
    T := T * I;
    A[I] := T;
  end;
end;
```

The procedure will perform N multiplications. Any student whose algorithm inefficiently computes N separate factorials will be given only 1 mark out of 4.

(b) [New theoretical application]

```
procedure PrintTree(T: array[1..N] of integer;
  I: integer);
begin
  if I <> 0
  then begin
    PrintTree(T, T[I+1]);
    writeln(Char(T[I]));
    PrintTree(T, T[I+2]);
  end;
end;
```

(c) [New theoretical application]

```
function SumTree(T: array[1..N]
  of integer): integer;
var I, J, S: integer;
begin
  S := 0;
  for I := 1 to N do
  begin
    J := (3 * (I - 1)) + 1;
    S := S + T[J];
  end;
  return S;
end;
```