Paper Number(s): **E2.7A**

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE
UNIVERSITY OF LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2002

EEE PART II: B.Eng., M.Eng. and ACGI

PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING

Friday, 7 June 2:00 pm

Corrected Copy

There are THREE questions on this paper.

Answer TWO questions.

This exam is OPEN BOOK.

Time allowed: 1:30 hours.

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:

None.

QUESTION ONE

Here is the type definition for a binary tree of strings.

```
TTree = ^TNode;
TNode =
record
     Node : string;
     Left : TTree;
     Right : TTree;
end;
```

To answer the following questions, you can assume the existence of access procedures for the type `TTree` called `Empty`, `Left`, `Right`, and Root with the obvious meanings.

a) Write a Pascal function that takes a binary tree and returns the difference between the number of nodes in its left sub-tree and the number of nodes in its right sub-tree. For example, given the tree of Fig. 1 below, the function would return 2, since there are 5 nodes in the left sub-tree and 3 in the right sub-tree.

[6]

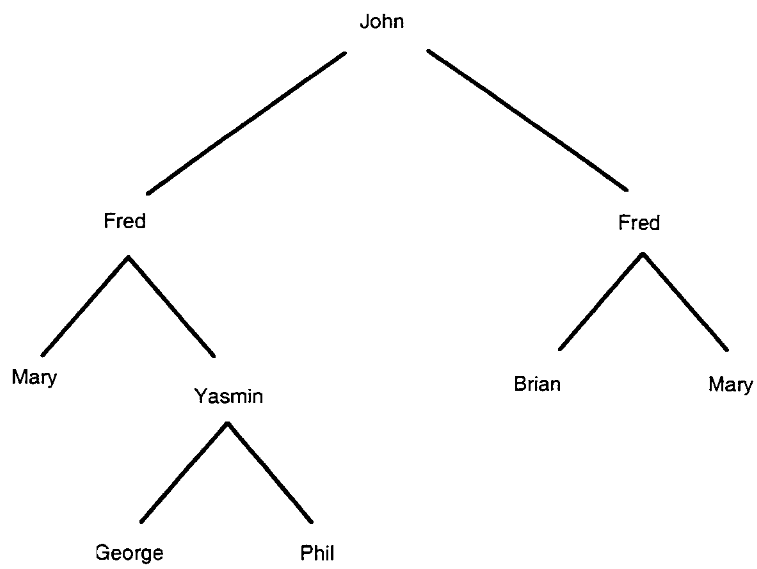

Fig. 1

b) Write a Pascal function that returns the height of a binary tree. This corresponds to the number of nodes in the longest branch of the tree from root to leaf. For example, the height of the tree in Fig. 1 is 4, the longest branch being the one that stretches from John to Phil (or George).

[7]

c) A binary tree is *symmetrical* if its left sub-tree is a mirror image of its right sub-tree. For example, the tree in Fig. 1 above is not symmetrical. But the tree in Fig. 2 below is symmetrical.

Continued on next page

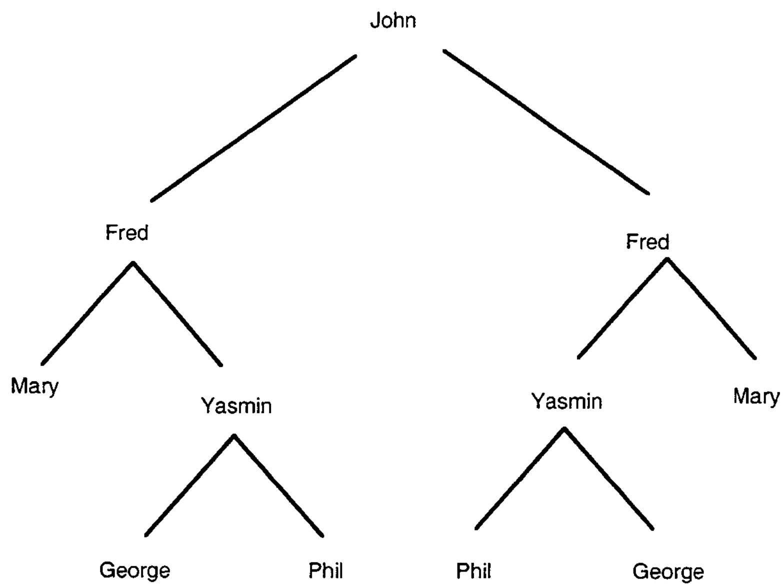

Fig. 2

Write a Pascal function that takes a binary tree and returns True if the tree is symmetrical and False if it is not.

[7]

QUESTION TWO

Consider the following Pascal program. (Note: the function `sqrt(x)` returns the square root of `x`, and the function `round(x)` rounds `x` to the nearest integer.)

```
program Compute;
const n = 100;
var
     i, j : integer;
     A : array [1..n] of boolean;
begin
     // Initialise array
     for i := 1 to n do
     begin
          A[i] := true;
     end;
     // Fill in array
     for i := 2 to round(sqrt(n)) do
     begin
          j := 2 * i;
          while j <= n do
          begin
               A[j] := false;
               j := j + i;
          end;
     end;
end.
```

a) Simulate the first three iterations of the second for loop, and show the contents of the first 15 elements of the array A after each iteration. (You can abbreviate true to T and false to F.)

[8]

b) What does the program do? Explain how it does it?

[8]

c) Suggest one way of improving the efficiency of the program.

[4]

QUESTION THREE

Here is the Pascal type declaration for a dynamic linked list of real numbers.

```
type
    TList = ^TLink;
    TLink =
    record
        First : real;
        Rest : TList;

    end;
```

a) Write a function `Middle` that takes a `TList` and returns the real number half way along that list. In other words, if the list is of length N, your function must return the $N/2^{th}$ element of the list. (If the list has an odd number of elements, return the $(N+1)/2^{th}$ element.) Use the following method. Count the number of elements N in the list, then start from the beginning of the list again and work along it until the $N/2^{th}$ element is reached.

[9]

b) Write a second version of `Middle` that uses the following method. Starting from the beginning of the list, work along it maintaining two pointers. The first pointer advances one element at a time, and the second pointer advances two elements at a time. When the second pointer reaches the end of the list, the first pointer will point to the element required.

[9]

c) In terms of loop iterations and/or recursive calls, which function is most efficient, and by how much? Explain your answer.

[2]

Model Answers

2002 E1 X E2.7A 1/5

QUESTION ONE

a)

```
function Difference(T : TTree): integer;
begin
     Difference :=
          Abs(Count(Left(T) - Count(Right(T)));
end;

function Count(T : TTree): integer;
begin
     if T = Empty
     then Count := 0
     else Count := Count(Left(T)) + Count(Right(T));
end;
```

b)

```
function Height(T : TTree): integer;
begin
     if T = Empty
     then Height := 0
     else begin
          L := Height(Left(T));
          R := Height(Right(T));
          if L > R
          then Height := L + 1
          else Height := R + 1;
     end;
end;
```

c)

```
function Symmetrical(T : TTree): boolean;
begin
     if (T = Empty) or Mirrors(Left(T),Right(T))
     then Symmetrical := true
     else Symmetrical := false;
end;
```

```
function Mirrors(T1, T2 : TTree): boolean;
var F : boolean;
begin
     if (T1 = Empty) and (T2 = Empty)
     then Mirrors := true
     else begin
          F := Mirrors(Left(T1),Right(T2));
          Mirrors :=
               F and Mirrors(Right(T1),Left(T2));
     end;
end;
```

QUESTION TWO

a)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
T	T	T	F	T	F	T	F	T	F	T	F	T	F	T
T	T	T	F	T	F	T	F	F	F	T	F	T	F	F
T	T	T	F	T	F	T	F	F	F	T	F	T	F	F

b) The program computes the first 100 prime numbers (using a simple version of the sieve of Eratosthenes). When the program terminates, the i^{th} element of A will be true if i is a prime number and false if it isn't. Initially all the elements of A are true (first for loop). The second for loop knocks out successive multiples of the natural numbers (by setting the corresponding element in A to false). First it eliminates multiples of 2, then multiples of 3, and so on. The algorithm only needs to go as far as multiples of $\sqrt{n}$ to get all the primes up to n.

c) The second for loop can be made more efficient by including a check to see whether A[i] is false. If so, all multiples of i will already be false, so there's no need to execute the while loop.

```
for i := 2 to round(sqrt(n)) do
begin
      if A[i] = true
      then begin
            j := 2 * i;
            while j <= n do
            begin
                  A[j] := false;
                  j := j + i;
            end;
      end;
end;
```

QUESTION THREE

a)

```
function Middle(L : TList): real
var N, M : integer;
begin
     N := Length(L);
     if N mod 2 = 0
     then N := N div 2
     else N := (N+1) div 2;
     if N <> 0
     then  begin
          M := 1;
          while M <> N do
          begin
               L := L^.Rest;
               M := M+1;
          end;
          Middle := L^.First;
     end
     else Middle := 0;
end;

function Length(L : TList): integer;
begin
     if L = nil
     then Length := 0
     else Length := Length(L^.Rest) + 1;
end;
```

b)

```
function Middle(L : TList): real;
begin
     if L = nil
     then Middle := 0
     else  begin
          Ptr := L^.Rest;
          while Ptr <> nil and Ptr^.Rest <>nil do
          begin
               Ptr := Ptr^.Rest^.Rest;
               L := L^.Rest;
          end;
          Middle := L^.First;
     end;
end;
```

c) The second function is more efficient. For a list of length n, it will execute n/2 iterations of the while loop. The first function executes n recursive calls to Length plus n/2 iterations of the while loop. So the first function will take around three times as long.