E1.8
E2.7A

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2003

SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES

Monday, 9 June 2:00 pm

Time allowed: 1:30 hours

There are THREE questions on this paper.

Answer TWO questions.

Corrected Copy

This exam is OPEN BOOK

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

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

Information for Invigilators:

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

Information for Candidates:

None.

The Questions

1. Assume the existence of a data type `TList` for a linked list, with the standard set of access procedures `Empty`, `First`, `Rest`, and `Add`. Now consider the following procedure.

```
function Merge(List1, List2 : TList): TList;
  var List3: TList;
  begin
    List3 := Empty;
    while List1 <> Empty or List2 <> Empty do
    begin
      if List1 = Empty or
        First(List2) < First(List1)
      then begin
        List3 := Add(First(List2),List3);
        List2 := Rest(List2);
      end
      else begin
        List3 := Add(First(List1),List3);
        List1 := Rest(List1);
      end;
    end;
    Merge := List3;
  end;
```

(a) Suppose `L1` is the list `[Amber, Chris, Ellen]` and `L2` is the list `[Billy, Darren]`. Trace the execution of the procedure call `Merge(L1,L2)` by showing the values of `List1`, `List2`, and `List3` at the end of each iteration of the while loop. [10]

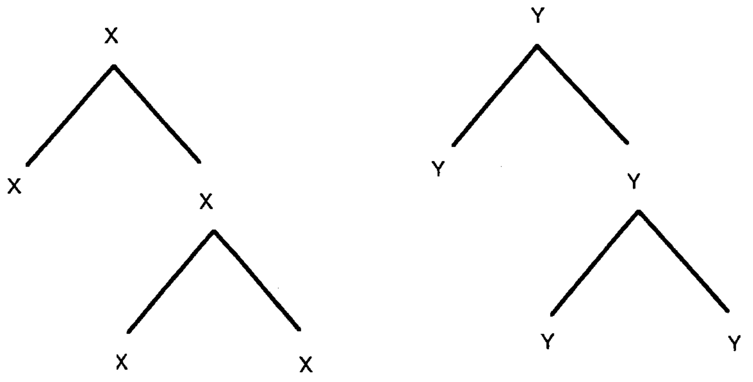
(b) What does the function do? How does it work? What conditions must `List1` and `List2` meet for the function to work correctly? [6]

(c) What difference, if any, would it make to the procedure if the arguments were declared as call-by-reference? [4]

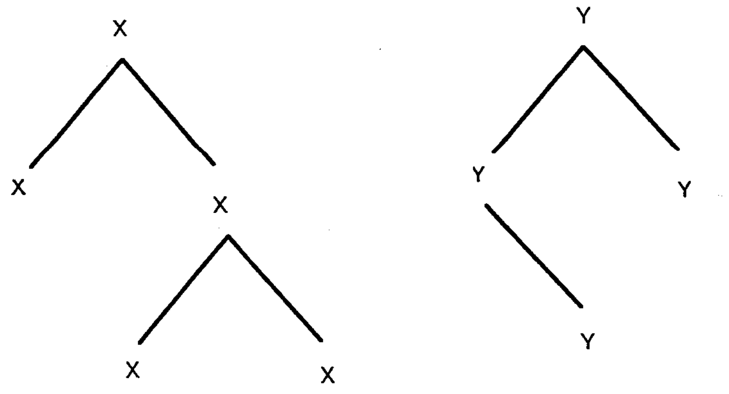
2. (a) Write a procedure that takes a two-dimensional array A1 of N by M integers and produces another N by M array A2 of integers in which each element [x,y] has been replaced by the average of the neighbourhood of five elements comprising [x,y] itself and the four elements above, below, and to either side of [x,y]. Copy the edges and corners of A2 straight from A1. [7]

(b) Assuming the compiler does not carry out any optimisation, how many add instructions *in total* will be executed by the procedure when it runs? Explain your workings. [7]

(c) Assume the existence of a data type `TTree` for a binary tree with the standard set of access procedures `EmptyTree`, `Left`, `Right`, and `Root`. Write a function `SameShape` that takes two binary trees and returns True if they have the same shape, ignoring the content of their nodes, and False otherwise. For example, the following two trees have the same shape,



while the following two trees do not.



[6]

3. (a) Write a non-recursive function `Match` that takes a string and returns False if the string contains non-matching round parentheses, but True otherwise. For example, if the string is "((P+Q)–R)+1" then the function should return True, but if the string is "Happy(Fred))" or "Happy(Fred" then the function should return False. You can assume the existence of a function `Length(S)` that returns the length of the string `S`.

   Hint: an easy way to do this is to work along the string, maintaining a count of the level of nesting. [6]

   (b) The following code defines a recursive function with the same purpose as the function in part (a).

```
function Match(S : string;
  I, J : integer): boolean;
begin
  if I > Length(S)
  then begin
    if J = 0
    then return True
    else return False;
  end
  else if J < 0
  then return False
  else begin
    if S[I] = '('
    then return Match(S,I+1,J+1)
    else if S[I] = ')'
    then return Match(S,I+1,J-1)
    else return Match(S,I+1,J);
  end;
end;
```

   Explain how the function works. Your explanation should say what the roles of the parameters `I` and `J` are, and what their values should be when the function is first called. [6]

   (c) Write a recursive procedure that computes the maximum level of nesting of brackets in a string. For example, the maximum level of nesting in the string "Happy(Fred)" is 1, while the maximum level of nesting in the string "Loves(Mother(x),x)" is 2. You may assume the string has matching parentheses. [8]

Model Answers

1. (a) [New theoretical application]

Iteration 1: List1 = [Chris, Ellen], List2 = [Billy, Darren], List3 = [Amber]

Iteration 2: List1 = [Chris, Ellen], List2 = [Darren], List3 = [Billy, Amber]

Iteration 3: List1 = [Ellen], List2 = [Darren], List3 = [Chris, Billy, Amber]

Iteration 4: List1 = [Ellen], List2 = [], List3 = [Darren, Chris, Billy, Amber]

Iteration 5: List1 = [], List2 = [], List3 = [Ellen, Darren, Chris, Billy, Amber]

(b) [New theoretical application]

The procedure takes two ordered lists List1, and List2, and produces a third list List3, in reverse order, which is the result of merging List1 and List2.

It works by repeatedly taking the head off one of the lists and adding it to the third (which is initialised to be empty). The head of the list chosen is always less-than-or-equal to the head of the other list, thus ensuring the reverse orderedness of the final list.

List1 and List2 must themselves be ordered for the procedure to work correctly.

(c) [New theoretical application]

If the arguments were declared as call-by-reference parameters, then the values of the variables assigned to List1 and List2 in the calling procedure would be modified. This would mean they would be overwritten by empty lists when the function terminated. The function would otherwise still work, however.

2. (a) [New theoretical application]

```
procedure Blur(A1 : TArray, var A2 : TArray);
var X, Y : integer;
begin
  for X := 2 to N-1
    for Y := 2 to M-1
    begin
      A2[X,Y] := A1[X,Y]+A1[X+1,Y]+A1[X,Y+1]+
                     A1[X-1,Y]+A1[X,Y-1];
      A2[X,Y] := A2[X,Y]/5;
    end;
    // Edges
    for X := 1 to N
    begin
      A2[X,1] := A1[X,1];
      A2[X,M] := A1[X,M];
    end;
    for Y := 1 to M
    begin
      A2[1,Y] := A1[1,Y];
      A2[N,Y] := A1[N,Y];
    end;
end;
```

(b) [New theoretical application]

For the main pair of nested for loops, the procedure will execute (N–2)*(M–2)*4 add instructions for the body of the loop, plus (N–2)*(M–2) adds for the inner loop variable, plus (N–2) adds for the outer loop variable. The two "edges" for loops will execute a further N+M add instructions. So the total is (N–2)*(M–2)*5+(N–2)+N+M.

(c) [New theoretical application]

```
function SameShape(T1, T2 : TTree): boolean;
begin
  if T1 = EmptyTree
  then return (T2 = EmptyTree)
  else begin
    F := SameShape(Left(T1),Left(T2));
    F := F and SameShape(Right(T1),Right(T2));
    return F;
  end;
end;
```

3. (a) [New theoretical application]

```
function Match(S : string): boolean;
var I, N : integer;
  Flag : boolean;
begin
  Flag := True;
  N := 0;
  for I := 1 to Length(S)
  begin
    if S[I] = '('
    then N := N+1
    else if S[I] = ')'
    then N := N-1;
    if N < 0 then Flag := False;
  end;
  if N <> 0 then Flag := False;
  return Flag;
end;
```

(b) [New theoretical application]

The function must be called with I = 1 and J = 0. I is an index into the string S, and J is the level of nesting of brackets in S within which the Ith character falls. Every time an open bracket is encountered, J is incremented, and the function is called recursively with the index advanced by one. Similarly, every time a close bracket is encountered, J is decremented. There are three base cases. First, if the end of the string is reached and the level of nesting (J) is zero, then the parentheses match, and the result ir True. Second, if the end of the string is reached and the level of nesting is not zero, then the brackets don't match and the result is False. Third, if the level of nesting goes negative, then a close bracket has been encountered without a matching open bracket before it, and the result is False.

(c) [New theoretical application]

```
procedure Nesting(S : string; I, J : integer;
var Max : integer);
begin
  if J > Max
  then Max := J;
  if I > Length(S)
  then return Max;
  else begin
    if S[I] = '('
    then Nesting(S,I+1,J+1,Max)
    else if S[I] = ')'
    then Nesting(S,I+1,J-1,Max)
    else Nesting(S,I+1,J,Max);
  end;
end;
```