E2.7A & E2.7B

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

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES

Tuesday 25th May 2004 2:00pm

There are THREE questions on this paper.

Answer TWO questions.

Corrected Copy

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 types, `TArray` and `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;

type TArray = array[1..N] of integer;
```

(a) Write a function with the following header that takes two arrays and returns a linked list of all integers that occur in both arrays.

```
function Matches(A1 : TArray; A2 : TArray): TList
```

[12]

Ensure that the list returned does not contain duplicates.

(b) In general, how many integer comparisons will the procedure perform in the best case? When does the best case occur? Explain your answers. [8]

2. (a) An amoeba reproduces asexually, so each individual has only one parent. Define a Pascal data type `TFamily` that can represent the family tree of an amoeba. Each node in the tree should contain the name of a parent, and have potentially any number of sub-nodes for children. [6]

   (b) Write a function with the following header that takes the family tree of an amoeba and two names and returns `True` if they are *siblings* (ie: have the same parent) and `False` otherwise. You may assume that every name in the tree is unique.

```
function Siblings(Family : TFamily;
  Name1 : string; Name2 : string): boolean
```
[10]

   (c) Describe in words how you modify your data structure to allow for two parents. [4]

3. Figure 3.1 depicts a binary tree of characters. The tree is not ordered.

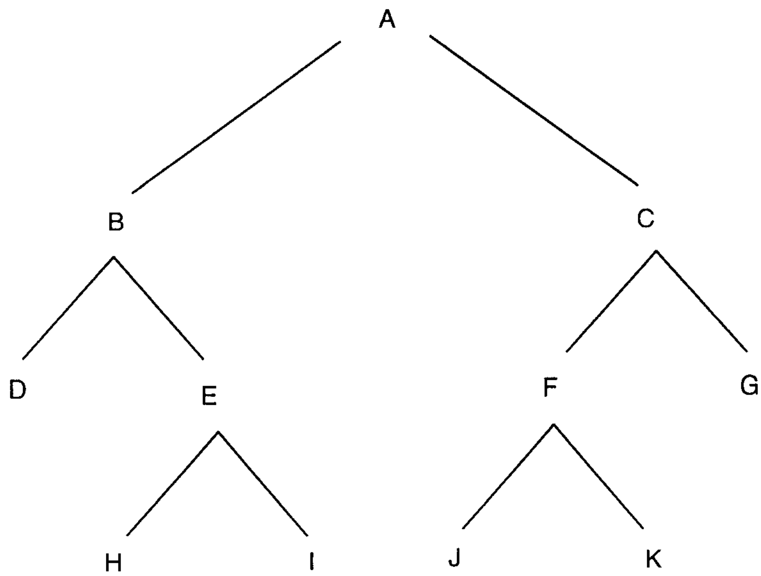


Figure 3.1

(a) Write out the sequence of nodes that would be visited by a procedure that traversed the tree in left-root-right order. [5]

(b) Draw an *ordered* binary tree with the same contents as the tree in Figure 3.1. [5]

(c) If a pointer takes up two bytes in memory, what is the storage requirement for the tree in Figure 3.1, assuming it is represented as a dynamic data structure? Explain your answer. [5]

(d) Draw a sketch showing how the left-half of the tree in Figure 3.1 might be represented in an array rather than using pointers. Explain your answer. [5]

Paper Number(s): E2.7B

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING

Wednesday 9th June 2004 2:00pm

There are THREE questions on this paper.

Answer TWO questions.

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): Constantinides, G.A
Second Marker(s): Demiris,Y.K.

Corrected Copy

Special Information for Invigilators:
This section of the examination is open book. Candidates may bring any *written* or *printed* material to the examination.

Information for Candidates
Throughout this section of the paper, the notation "0x" before a number means that the number is expressed using hexadecimal representation.

The Questions

1.

A subroutine `scramble` is shown below.

```
scramble
        STMED       r13!, {r0-r3}
        MOV         r3, #0
loop
        LDRB        r2, [r0], #1
        ADD         r2, r2, r3
        CMP         r2, #'Z'
        SUBGT       r2, r2, #('Z'-'A')
        ADD         r3, r3, #1
        CMP         r3, #('Z'-'A')
        MOVGT       r3, #0
        STRB        r2, [r0,#-1]
        SUBS        r1, r1, #1
        BNE         loop
        LDMED       r13!, {r0-r3}
        MOV         pc, lr
```

a) Consider the following instructions. For each one, state which registers, memory locations, and flags may be modified as a result of execution.

```
(i)    LDRB     r2, [r0], #1
(ii)   STRB     r2, [r0,#-1]
(iii)  SUBGT    r2, r2, #('Z'-'A')
(iv)   SUBS     r1, r1, #1
(v)    STMED    r13!, {r0-r3}
```

[7]

b) Describe the purpose of the link register.

[2]

c) Assuming that on entry `r0` points to a message consisting of upper-case characters, what is the function of subroutine `scramble`?

[2]

Prior to subroutine entry, `r0` has the value 0x8100 and `r1` has the value 0x3. A partial content listing of the memory is shown in Table 1, below.

Table 1

Address	Data
0x8100	ASCII encoding of 'A'
0x8101	ASCII encoding of 'B'
0x8102	ASCII encoding of 'C'

d) Provide a partial content listing of the memory after subroutine execution, listing all locations where that data has changed.

[3]

e) Modify the code so that any spaces in the original message are left unscrambled.

[6]

2.

Given a block of N memory words, a length-2 moving sum filter finds the arithmetic sum of the values of 2 consecutive memory words, as illustrated in Figure 1. This process is repeated a total of N-1 times, with the starting location of the window of 2 locations changing by one memory word per iteration.

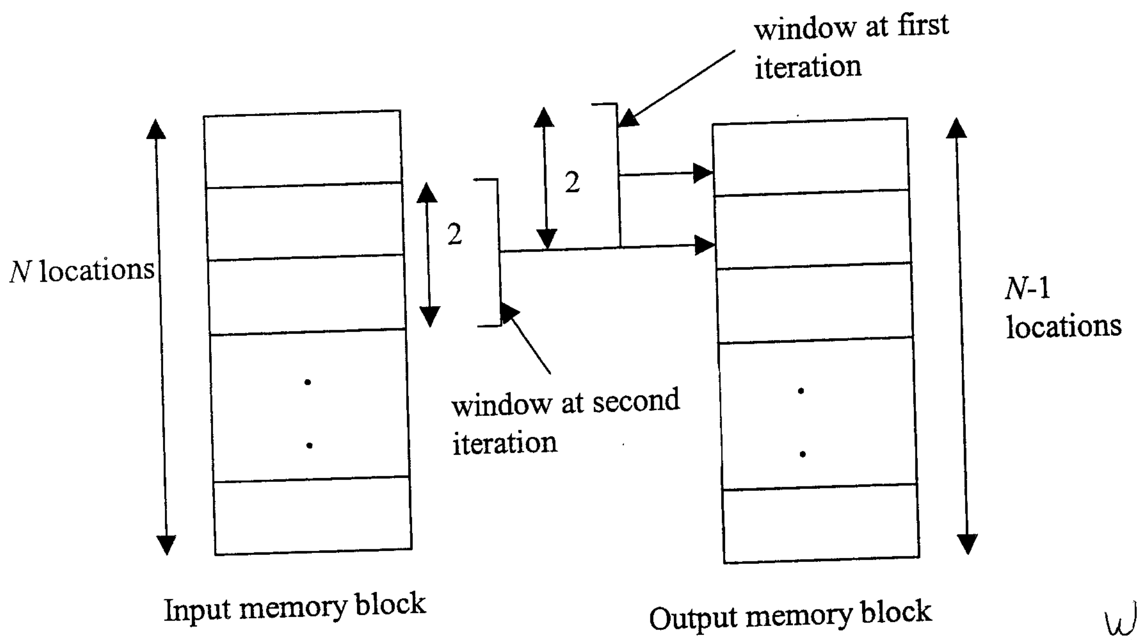


Figure 1

WRITTEN ON BOARD BEFORE EXAM STARTED

a) Write a subroutine called `moving`~~`avg`~~ sum to implement this moving ~~average~~ sum filter. The subroutine should take three arguments: `r0` should contain the starting address of the input block of memory, `r1` should contain the starting address of the output block, and `r2` should contain N.

[10]

b) In general, a length-K moving sum filter finds the arithmetic sum of the values of K consecutive memory words. This process is repeated a total of N-K+1 times, with the starting location of the window of K locations still changing by one memory word per iteration.

Extend your subroutine to this general case. The subroutine should now have four arguments: `r0` should contain the starting address of the input block of memory, `r1` should contain the starting address of the output block, `r2` should contain N, and `r3` should contain K.

[10]

3.

a) Assemble the following sequence of ARM instructions into (binary or hex) machine code.

```
loop    LDR      r2, [r0], #4
        ADD      r2, r2, #1
        CMP      r2, #0
        STRGT    r2, [r0,#-4]
        BGT      loop
        SWI      0x11
```

[10]

The address of the first instruction is 0x8000, and `r0`=0x1000 immediately before entering this code fragment. A partial content listing of the memory is shown in Table 2, below.

Table 2

Address	Data
0x1000	0x00000100
0x1004	0x00000200
0x1008	0x00000000

b) Write a time-ordered list of instruction fetch accesses for this code. For each memory access, state the address of the word accessed, whether the access is a read or write, and the data read or written (in hex).

[NB: You may assume that the processor is not pipelined]

[2]

c) Write a time-ordered list of memory data accesses performed by this code. For each memory access, state the address of the word accessed, whether the access is a read or write, and the data read or written (in hex).

[2]

d) It is proposed to use both an instruction cache and a separate data cache to speed up the execution of this code fragment. There are to be 4 lines in each cache, each of one word.

Assuming the caches are initially empty, draw a diagram illustrating the cache contents after the access sequence above has completed. For each cache line, include the tag, the valid bit, and the data.

[4]

e) For each cache, state the number of hits and misses caused by this execution.

[2]

Model Answers

1. (a) [New theoretical application]

```
function Matches(A1 : TArray; A2 : TArray): TList;
var X, Y : integer;
begin
  Ans := EmptyList;
  for X := 1 to N do
    for Y := 1 to N do
      if A1[X] = A2[Y]
      then Ans := AddND(A1[X],Ans);
  return Ans;
end;
```

If the student gets above right but omits to check for duplicates, they should get half the total marks.

```
function AddND(Z : integer; List : TList): TList;
{ Add with check for duplicates }
var Ptr : TList;
begin
  Ptr := List;
  while (Ptr <> EmptyList) and (First(Ptr) <> Z) do
    Ptr := Rest(Ptr);
  if Ptr = EmptyList
  then return Add(Z,List)
  else return List;
end;
```

(b) [New theoretical application]

The best case is when the two arrays have no elements in common. Then the calls to `AddND` will not require any integer comparisons and the total number is N^2 – once for each call to `AddND`. There will be N^2 calls because the invocation is embedded in two nested for loops, each of which carries out N iterations.

(If the student's answer allows for early exit of the inner for loop, then this will be reduced to N comparisons)

The worst case is where the two arrays are identical. Then the total number of comparisons is . We have the same N^2 comparisons as before, plus the comparisons carried out by the calls to `AddND`. The outer loop executes N times, and the i[th] iteration of the inner loop requires $i-1$ comparisons, because the list of common elements will have length $i-1$.

2. (a) [New theoretical application]

```
type
  TFamily = ^TNode;
  TNode =
  record
    Parent : string;
    Kids : TList;
  end;

  TList = ^TLink;
  TLink =
  record
    First : TFamily;
    Rest : TList;
  end;
```

(b) [New theoretical application]

```
function Siblings(Family : TFamily;
  Name1 : string; Name2 : string): boolean;
var Kids : TList; Found : boolean;
begin
  if Family = nil
  then return False
  else begin
    Kids := Family^.Kids;
    if Member(Name1,Kids) and Member(Name2,Kids)
    then return True
    else begin
      Found := False;
      while Kids <> nil and not Found do
      begin
        if Siblings(Kids^.First,Name1,Name2)
        then Found := True
        else Kids := Kids^.Rest;
      end;
      return Found;
    end;
  end;
```

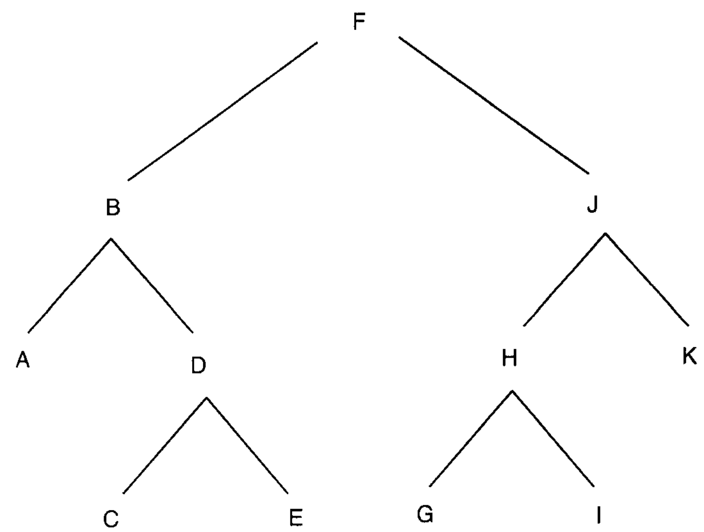
(c) [New theoretical application]

The `TNode` type definition would have to include a field for each parent. But it would not be possible to encode every hereditary relationship in a single tree. To do this, we would require multiple interconnected trees with different root nodes (a forest).

3. (a) [New theoretical application]

   D B H E I A J F K C G

   (b) [New theoretical application]



   (c) [New theoretical application]

   Each node *including the leaves* requires two bytes of storage per pointer plus one for the character. So the total is 55 bytes.

   (d) [New theoretical application]

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

   Each node takes up three elements of the array. The first element is the character, the second is the index of the left sub-node, and the third is the index of the third sub-node. If a node has no sub-node, then a 0 goes in the appropriate location (analogous to the nil pointer).

E1.9 (a)
E2.7 B

PRINCIPLES OF COMPUTERS — MODEL ANSWERS
2004

1. a)

INSTR	REGS	MEM	FLAGS
(i)	r0, r2	[r0]	NONE
(ii)	r2	[r0 - 1]	NONE
(iii)	r2	NONE	NONE
(iv)	r1	NONE	ALL
(v)	r13	[r13] to [r13 - 15]	NONE

b) TO STORE THE RETURN ADDRESS FROM A SUBROUTINE CALL (BL INSTRUCTION)

c) IT ENCODES THE MESSAGE. IF p_i DENOTES PLAINTEXT CHARACTER i, c_i DENOTES CIPHERTEXT CHARACTER i, AND COUNTING OF CHARACTERS STARTS FROM ZERO, THEN

$$c_i = i + p_i$$

WITH WRAP-AROUND FROM $Z \rightarrow A$.

d)

ADDRESS	DATA
0x8101	ASCII ENCODING OF 'C'
0x8102	ASCII ENCODING OF 'E'

e)

```
scramble
        STMED   r13!, {r0-r3}
        MOV     r3, #0
Loop
        LDRB    r2, [r0], #1
        CMP     r2, #' '
        BEQ     Skip
        ADD     r2, r2, r3
        CMP     r2, #'Z'
        SUBGT   r2, r2, #('Z' - 'A')
        ADD     r3, r3, #1
        CMP     r3, #('Z' - 'A')
        MOVGT   r3, #0
        STRB    r2, [r0, #-1]
Skip    SUBS    r1, r1, #1
        BNE     Loop
        LDMED   r13!, {r0-r3}
        MOV     pc, lr
```

2. a)

```
movingavg   STMED   r13!, {r0-r4}
            SUB     r2, r2, #1          ; N-1 iterations
Loop        LDR     r3, [r0], #4
            LDR     r4, [r0]
            ADD     r3, r3, r4          ; form sum
            STR     r3, [r1], #4
            SUBS    r2, r2, #1
            BNE     Loop
            LDMED   r13!, {r0-r4}
            MOV     pc, lr
```

b)

```
movingavg   STMED   r13!, {r0-r6}
            SUB     r3, r3, #1          ; more convenient to keep K-1
            SUB     r2, r2, r3          ; r2 holds N-K+1 now
Loop1       MOV     r6, #0              ; r6 holds running total
            MOV     r5, r3, LSL #2      ; offset for sum term (in bytes)
Loop2       LDR     r4, [r0, r5]
            ADD     r6, r6, r4          ; running total
            SUBS    r5, r5, #4          ; next term
            BPL     Loop2               ; positive or zero offset => more terms
            STR     r6, [r1], #4        ; final result
            ADD     r0, r0, #4          ; start of window
            SUBS    r2, r2, #1          ; sample update
            BNE     Loop1
            LDMED   r13!, {r0-r6}
            MOV     pc, lr
```

3. a)

LDR	r2, [r0], #4	→	0x E4902001	← (1)
ADD	r2, r2, #1	→	0x E2822001	← (2)
CMP	r2, #0	→	0x E3520000	← (3)
STRGT	r2, [r0, #-4]	→	0x C5002001	← (4)
BGT	Loop	→	0x CAFFFFFA	← (5)
SWI	0x11	→	0x EF000011	← (6)

b)

ADDRESS	READ/WRITE	DATA	MISS/HIT [FOR PART e]
0x8000	R	(1) ABOVE	M
0x8004	R	(2) "	M
0x8008	R	(3) "	M
0x800C	R	(4) "	M
0x8010	R	(5) "	M
0x8000	R	(1) "	M
0x8004	R	(2) "	H
0x8008	R	(3) "	H
0x800C	R	(4) "	H
0x8010	R	(5) "	M
0x8000	R	(1) "	M
0x8004	R	(2) "	H
0x8008	R	(3) "	H
0x80DC	R	(4) "	H
0x8010	R	(5) "	M
0x8014	R	(6) "	M

c)

ADDRESS	READ/WRITE	DATA	MISS/HIT [FOR PART e]
0x1000	R	0x100	M
0x1000	W	0x101	H
0x1004	R	0x200	M
0x1004	W	0x201	H
0x1008	R	0x000	M

d) INSTRUCTION CACHE

CACHE LINE #	TAG	VALID	DATA
0	0x801	✓(Y)	0x CAFFFFFA
1	0x801	✓(Y)	0x EF000011
2	0x800	✓(Y)	0x E3520000
3	0x800	✓(Y)	0x C5002001

DATA CACHE

CACHE LINE #	TAG	VALID	DATA
0	0x100	✓(Y)	0x101
1	0x100	✓(Y)	0x201
2	0x100	✓(Y)	0x000
3	?	X (N)	?

3. e) INSTRUCTION CACHE : 10 MISSES / 6 HITS

DATA CACHE : 3 MISSES / 2 HITS