E1.8
E2.18

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2010

ISE PART I/EEE PART II

SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES

Tuesday, 25 May 2:00 pm

Time allowed: 1:30 hours

There are THREE questions on this paper.

Answer Question ONE and ONE other question.

This exam is OPEN BOOK.

Question One carries 40% of the marks; Questions Two and Three each carry 60%.

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

Examiners responsible	First Marker(s) :	C. Bouganis
	Second Marker(s) :	L.G. Madden

Special information for invigilators:
Students may bring any written or printed aids into the examination.

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

The Questions

1. **[Compulsory]**

a) Figure 1.1 shows a C++ procedure that calculates the value of the function described in equation (1.1), for a value of n where n is a non-negative integer.

$$f(n) = \begin{cases} 1 & n = 0 \\ (n+1) * f(n-1) & n > 0 \end{cases} \tag{1.1}$$

Identify four errors in the C++ code shown in Figure 1.1.

```
void calculateF (bool N, int result) {
    result=1;
    int i;
    for (i=1; i <= n+1; j=j+1)
        result = result * i;
}
```

Figure 1.1 calculateF() procedure.

[6]

b) Write a C++ recursive function that performs the calculation described in part (a).

[6]

[continued on the following page]

c) i) A set of numbers is inserted in an ordered binary tree (ascending ordered tree). Draw a tree for the following set assuming that the elements in the set are inserted in the order shown.

$\{10, 15, 20, 18, 12, 9\}$

[4]

ii) Delete element 15 from the ordered tree in part (i) maintaining the ordering. Draw the resulting tree.

[2]

iii) An alternative data structure is a hash table. Assume a chained hash table with three entries and a hash function $H(x) = x \bmod 2$, where the hash key x is the value of the inserted number. Draw a hash table, without any ordering imposed, for the set of numbers of part (i) assuming that the elements in the set are inserted in the order shown.

[2]

iv) Comment on the utilisation of the hash table structure of part (iii), and propose a new hash function $F(x)$ in order to utilise all available entries of the hash structure. Draw the resulting hash table, without any ordering imposed, that utilises your proposed function for the set of numbers of part (i) assuming that the elements in the set are inserted in the order shown.

[4]

d) Construct a parse tree for the following expressions, assuming the normal priorities of the operators:

i) $3 + 2 * 10$

[2]

ii) $3 - (3 * 4)/5$

[2]

[continued on the following page]

e) Consider the C++ code segment in Figure 1.2. With justification, state the values of variables x, y and z at points A and B of the code. With justification, state whether this code segment has a memory leak or not. The functionA() function is given in Figure 1.3.

```
    int x=3;
    int y=4;
    int z=0;
    int *p1 = &x;
    int *p2 = &y;
    int *p3 = new int;
    *p1 = 10;
    *p2 = 20;
    *p3 = x*y;
A
    functionA(x,y);
    z = *p3;
B
```

Figure 1.2 Code segment.

```
void functionA(int x, int y) {
    x = y;
}
```

Figure 1.3 functionA() function.

[6]

[continued on the following page]

f) Figure 1.4 shows the type declaration for a dynamic linked list of integers in C++.

```
struct Node {
    int data;
    Node * next;
};

typedef Node * NodePtr;
NodePtr hdList = NULL;
```

Figure 1.4 Linked list declaration.

i) Write a C++ recursive function that takes as input the *hdList* pointer, and returns the pointer to the last node of the list. If the list is empty, the function should return NULL.

[4]

ii) Write a C++ function that takes as inputs the *hdList* pointer, and performs the same operation as in part (i) using an iteration.

[2]

2. Consider a binary tree structure where each node stores an integer. Figure 2.1 illustrates an example of such a tree.

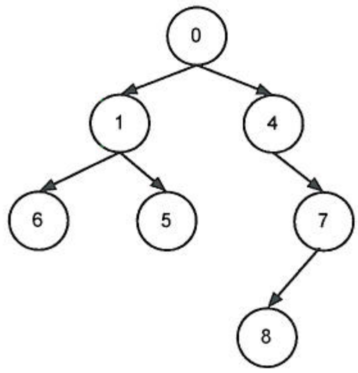


Figure 2.1 Binary Tree.

a) Define a structure *Node* capable of representing a node of the tree.

[9]

b) Write a recursive function/procedure that takes as input a pointer to the root of the tree and returns the number of nodes in the tree. In the case where the tree is empty, the function/procedure should return the value 0.

[9]

c) Write a recursive function/procedure that takes as inputs the pointer to the root of the tree and an integer number N, and returns the number of entries in the tree with data less than N.

[12]

d) Assume that direct access to each node of the tree is needed. This can be achieved by constructing a linked list structure that stores pointers that point to a node of the tree structure. Define a structure *ListNode* capable of holding this information.

[9]

e) Write a function/procedure that takes as inputs a pointer to the root of the binary tree, and a pointer to the head of the list structure described in part (d), and constructs the list described in part (d) that provides links to all nodes of the binary tree. You can assume that the pointer that points to the head of the list has been properly initialised to NULL. Your function/procedure should return the pointer that points to the head of the resulting linked list.

[12]

f) Modify the above function, or otherwise, in order for the resulting linked list to provide links only to the leaves of the binary tree structure.

[9]

3. The exam marks for the Software Engineering course are stored in a binary tree structure. Each node of the structure stores the *CID* of the student and his/her *mark*. The marks are within $[0, 100]$ range. Figure 3.1 presents an instance of such a structure.

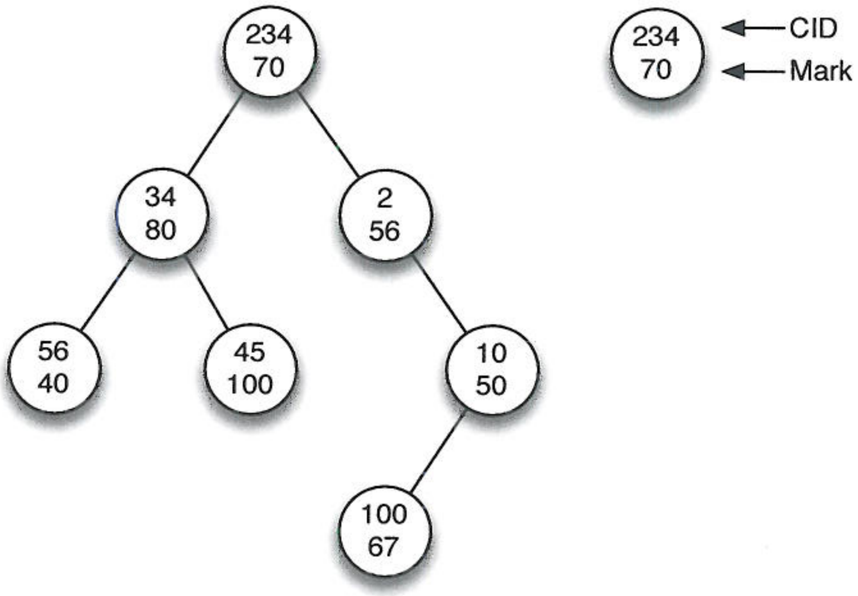


Figure 3.1 An instance of the tree structure, without any ordering imposed.

a) Define a structure *Node* capable of representing a node of the structure shown in Figure 3.1.

[9]

b) Write a recursive function/procedure that takes as inputs a pointer to a node and returns the minimum and maximum values of the marks in the tree that has that node as its root.

[12]

c) Using the function from part (b), or otherwise, write a recursive function/procedure that takes as input a pointer to the root of the tree and checks whether an ordering with respect to the marks has been imposed to the tree structure. The function should return TRUE is an ordering has been imposed, otherwise it should return FALSE.

[18]

d) Write a function/procedure that takes as inputs the pointer to the root node, and an integer variable *N*, and calculates the percentage of the students that have achieved a mark greater or equal to *N*. You can pass extra parameters to your function/procedure. (Hint: The most efficient solution requires one pass of the tree only).

[12]

[continued on the following page]

e) Assuming that an ordering according to the marks has been imposed to the tree structure, write a recursive function/procedure that returns the pointer to the node with the highest mark.

[9]

E1.8

E1.8 | E2.18

SOFTWARE ENGINEERING — $\frac{1}{10}$

Introduction, Algorithms & Data Structures

2010

Solutions

1) a) bool → int

N → n

j=j+1 → i=i+1

int result → int & result

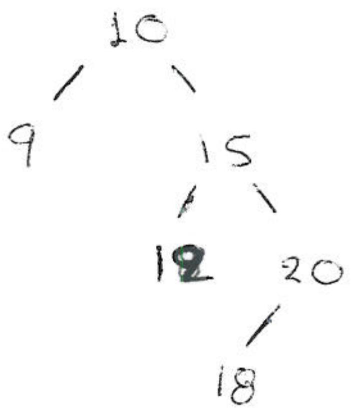
[6]

b)

```
int calculatefR(int n) {
    if (n==0)
        return 1;
    else
        return (n+1) * calculatefR(n-1);
}
```

[6]

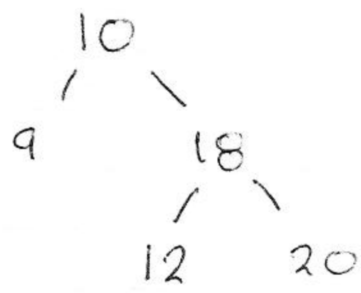
c) i)



[4]

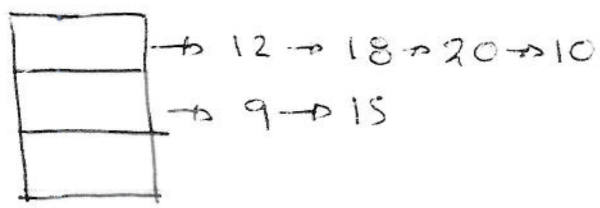
$\frac{2}{10}$

ii)



[2]

iii)



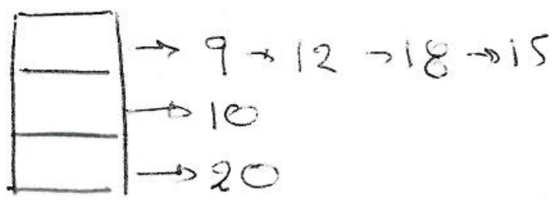
10 mod 2 = $\emptyset$
15 mod 2 = 1
20 mod 2 = $\emptyset$
18 mod 2 = $\emptyset$
12 mod 2 = $\emptyset$
9 mod 2 = 1

[2]

iv) One entry of the table is not used at all. This is due to the hash function.

F(x) = ~~x~~ mod 3 (other functions can be suggested).

10 mod 3 = 1
15 mod 3 = $\emptyset$
20 mod 3 = 2
18 mod 3 = $\emptyset$
12 mod 3 = $\emptyset$
9 mod 3 = $\emptyset$

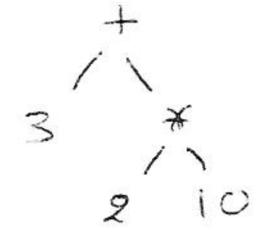


[2]

$\frac{3}{10}$

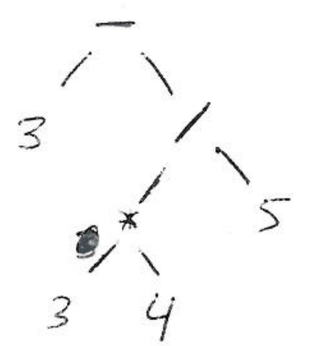
d)

i)



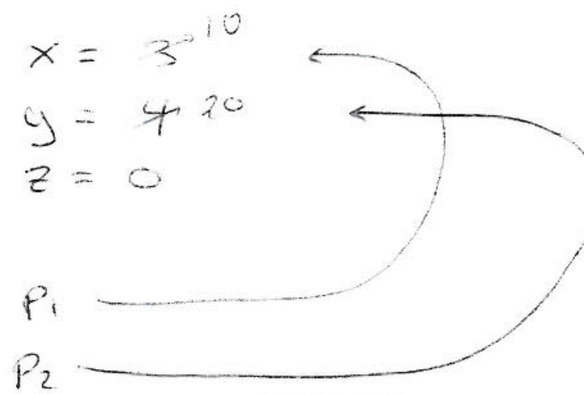
[2]

ii)



[2]

c)



A: $x = 10$
$y = 20$
$z = \emptyset$

B: $x = 10$
$y = 20$
$z = 200$

functionA() does not change the values of x, y.
Z = 200
There is not a memory leak.

[6]

4/10

f) i)

```
NodePtr LastElem(NodePtr hdList){

    if (hdList ==NULL)
        return NULL;        //empty list
    else
        if (hdList->next ==NULL)
            return hdList;
        else
            return LastElem(hdList->next);
}
```

[4]

ii)

```
NodePtr LastElem(NodePtr hdList) {
    NodePtr temp = NULL;
    while (hdList != NULL) {
        temp =hdList;
        hdList = hdList->next;
    }
    return temp;
}
```

[2]

5/10

2 a)

```
struct Node {
        int data;
        Node * left;
        Node * right;
};

typedef Node * NodePtr;        (not necessary)
```

[9]

b)

```
int   countNodes (NodePtr hdTree) {
      if (hdTree == NULL)
         return 0;
      else
         return countNodes (hdTree->left) +
                   countNodes (hdTree->right) + 1
}
```

[9]

c)

```
void   countElemN ( NodePtr hdTree, int N, int &counter)

       if (hdTree != NULL) {
          if (hdTree->data < N)
               counter = counter + 1;
          countElemN (hdTree->left, N, counter);
          countElemN (hdTree->right, N, counter);
       }
}
```

[12]

6/10

d)

```
struct ListNode {
        Node * data;
        ListNode * next;
}

typedef ListNode * ListNodePtr;
```

9
[8]

e)

```
void constructList (NodePtr hdTree, ListNodePtr & hdList){

    ListNodePtr temp = NULL;

    if (hdTree != NULL) {

        //add element in the list
        temp = new ListNode;
        temp -> data = hdTree;
        temp -> next = hdList;
        hdList = temp;

        // cont. the search
        constructList (hdTree ->left, hdList);
        constructList (hdTree -> right, hdList);
    }
}
```

[8]
12

7/10

f)

```
void constructListLeaves (NodePtr hdTree, ListNodePtr & hdList){
    ListNodePtr  temp = NULL;

    if (hdTree != NULL) {

        if (hdTree->left == NULL) && (hdTree->right == NULL) {
            // This is a leaf
            temp = new ListNode;
            temp->data = hdTree;
            temp->next = hdList;
            hdList = temp;
        } else {
            // cont searching
            constructListLeaves (hdTree->left, hdList);
            constructListLeaves (hdTree->right, hdList);
        }
    }
}
```

[9]

$\frac{8}{10}$

3) a)

```
struct Node {
        int mark;
        int cid;
        Node * left;
        Node * right;
}

typedef Node * NodePtr; (optional)
```

[4]

b)

```
void findMinMax (NodePtr hdTree, int &min, int &max){

    if (hdTree != NULL) {
        if (min > hdTree -> mark)
            min = hdTree -> mark;
        if (max < hdTree -> mark)
            max = hdTree -> mark;

        findMinMax (hdTree -> left, min, max);
        findMinMax (hdTree -> right, min, max);
    }
}
```

min should be initialized to +101
max " " " " -1

[9]
12

9/10

c)

```
void checkOrder(NodePtr hdTree, bool &order) {

    if (hdTree != NULL) {

        int min = 101;
        int max = -1;
        findMinMax(hdTree->left, min, max);
        if ( max > hdTree->mark)
            order = false;

        min = 101;
        max = -1;
        findMinMax(hdTree->right, min, max);
        if (min < hdTree->mark)
            order = false;

        checkOrder(hdTree->left, order);
        checkOrder(hdTree->right, order);

    }
}
```

Call the above function with order = true.

[8]

$\frac{10}{10}$

d)

```
void calcPercN (NodePtr hdTree, int N, int &counter1,
                                       float &counter2,
                                       float &per ) {

    if (hdTree != NULL) {

        if (hdTree -> mark >= N)
            counter1 ++;
        counter2 ++ ;                          // number of students
        per = counter1 / counter2 ;            // It can be div by 100.

        calcPerN (hdTree -> left, N, counter1, counter2, per);
        calcPerN (hdTree -> right, N, counter1, counter2, per);

    }
}
```

[12] [13]

e)

```
NodePtr highestMarkNode (NodePtr hdTree) {

    if (hdTree == NULL)
        return NULL ;
    else
        if (hdTree -> right != NULL)
            return highestMarkNode (hdTree -> right);
        else
            return hdTree ;
}
```

[9]