E1.8
~~E2.7A~~
E2.18

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2009

ISE PART I/EEE PART II

SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES

Friday, 5 June 2:00 pm

Time allowed: 1:30 hours

Corrected Copy

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. (Closed book for JGT)

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++ function that calculates the value of the function described in equation (1.1), for a value of n where n is integer.

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ f(n-1)+1 & \text{if } n \text{ is odd} \\ 2 * f(n-1) & \text{if } n \text{ is even} \end{cases} \quad (1.1)$$

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

```
void calculateF (int N) {
    result=0;
    int i;
    for (i=1; i <= n; i=i+2){
        if ((i % 2) == 0)
            result = result * 3;
        else
            result = result + 1;
    }
    return result;
}
```

Figure 1.1 calculateF() function.

[3]

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

[3]

[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.

$\{20, 5, 3, 1, 32, 40\}$

[2]

ii) An alternative data structure to a tree is a list. Draw an ordered list for the set of part (i) assuming that the elements in the set are inserted in the order shown.

[1]

iii) An alternative data structure is a Hash table. Assume a chained hash table with five entries and with a hash function $H(x) = x \bmod 5$, 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.

[1]

iv) Balance the tree of part (i) and draw the resulting tree. Comment on the type of rotation that you are performing and indicate which nodes are the root and pivot nodes for the required rotation(s).

[2]

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

i) $3/2+1+5$

[1]

ii) $1+5*6/7+8$

[1]

[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 addOne() function is given in Figure 1.3.

```
    int x=5;
    int y=10;
    int z=1;
    int *p1 = &x;
    int *p2;
    p2 = &y;
    int *p3 = new int;
    *p3 = 5;
    *p1 = 2*y;
A
    y = addOne(x);
    p3 = p1;
B
```

Figure 1.2 Code segment.

```
int addOne(int x) {
    x = x + 1;
    return x;
}
```

Figure 1.3 addOne() function.

[3]

[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 inputs the *hdList* pointer and an integer *N*, and returns the pointer to the first node of the list that has its data field value equal to *N*. If no such node exists, the function should return NULL.

[2]

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

[1]

2. An AVL tree is a self-balanced ordered binary tree. Consider an AVL tree structure where each node stores an integer and its balance factor BF. Figure 2.1 illustrates an example of such a tree.

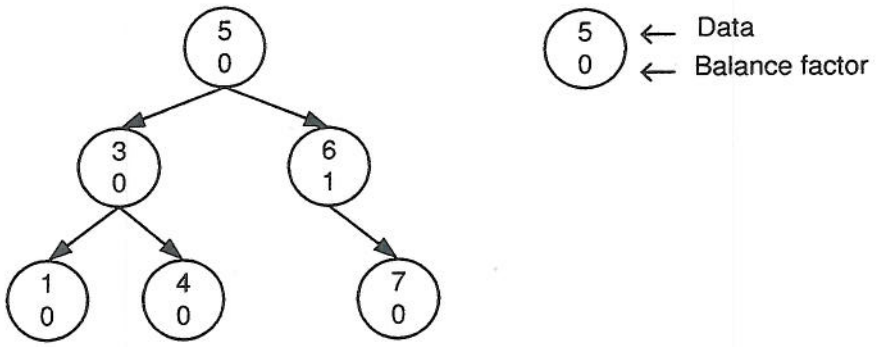

Figure 2.1 AVL Tree.

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

[3]

b) Write a recursive function/procedure that takes as input a pointer to a node of the tree and returns the height of the tree that has as root that node. In the case of an empty tree, the function should return the value -1.

[3]

c) Using the function/procedure of part (b), or otherwise, write a function/procedure that takes as input a pointer, and updates the balance factor BF of the node that the input pointer points to.

[3]

d) Using the function/procedure of part (c), or otherwise, write a recursive function/procedure that takes as input a pointer to a tree and updates the balance factor fields of all the nodes in the tree.

[5]

e) Write a function/procedure that takes as input a pointer to a node R, and performs single left rotation having node R as the root node. You can assume that the pivot node always exists. Note, that you do not need to update the BF factors.

[3]

f) Using the function/procedure of part (e), or otherwise, write a function/procedure that takes as input a pointer to a node and performs a double right rotation, i.e. a single rotation to the left followed by a single rotation to the right. Note, that you do not need to update the BF factors, and you can assume that all the necessary pivot nodes exist.

[3]

3. In any biological organism, the expression level of a gene affects the expression levels of other genes. This is usually graphically represented by a directed acyclic graph. Each node represents a unique gene, where every arrow in the graph models the existence of a direct influence between two genes. Figure 3.1 presents an instance of such a graph. You can assume that every node has a unique number *id*, every node has up to two children, there are no cycles in the graph, and all nodes are reachable from the root node of the graph ($id = 0$).

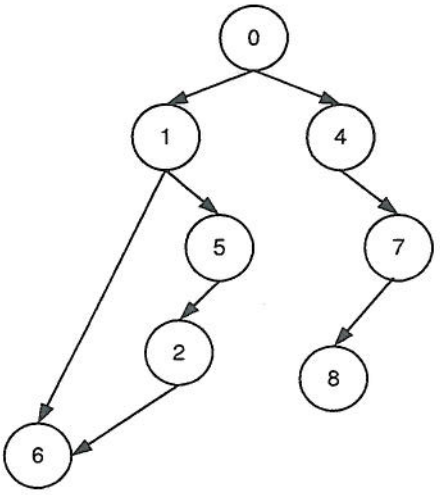

Figure 3.1 An instance of the graph showing the id of each node.

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

[3]

b) Write a recursive function/procedure that takes as inputs the pointer to root node ($id = 0$) and two input variables *id1* and *id2*, and indicates whether gene with its id field equal to *id1* directly affects the gene with its id field equal to *id2* (i.e. the node with *id1* has as a child the node with *id2*). You may use extra arguments.

[3]

c) Write a recursive function/procedure that takes as inputs the pointer to the root node ($id = 0$) and an input variable *id* and returns the pointer that points to the node with its id field equal to the input *id*. The function/procedure should return NULL value if such node does not exists.

[4]

[continued on the following page]

d) Write a function/procedure that takes as inputs the pointer to the root node ($id = 0$) and two input variables *id1* and *id2*, and indicates whether there is a directed path from node with its id field equal to *id1* to node with its id field equal to *id2*. For example, there is a directed path from node with $id = 1$ to node with $id = 2$, but there is not from node with $id = 2$ to node with $id = 1$.

[5]

e) Write a recursive function/procedure that takes as inputs the pointer to a node of the tree P and an input variable *id*, and returns the length of the shortest directed path between the node that the input pointer P points to, and the node with id equal to the input *id*. For example, if node P points to node with $id = 0$ and $id = 6$, the function/procedure should return the value 2.

[5]

Ē1.8
(Ē2.18)

SOLUTIONS 2009

1. **(This question covers most of the syllabus. New application of theory.)**

a) The correct code is shown in Figure 1.1.

```
int calculateF (int n) {
    int result=0;
    int i;
    for (i=1; i <= n; i++){
        if ((i % 2) == 0)
            result = result * 2;
        else
            result = result + 1;
    }
    return result;
}
```

Figure 1.1 calculateF() function.

[3]

b) The solution is shown in Figure 1.2.

```
int calculateFR(int n) {
    if (n==0)
        return 0;
    else
        if ((n % 2) == 0)
            return 2*calculateFR(n-1);
        else
            return calculateFR(n-1) + 1;
}
```

Figure 1.2 Solution 1b.

[3]

c) i) Solution in Figure 1.3.

[2]

ii) Solution in Figure 1.4.

[1]

iii) Solution in Figure 1.5.

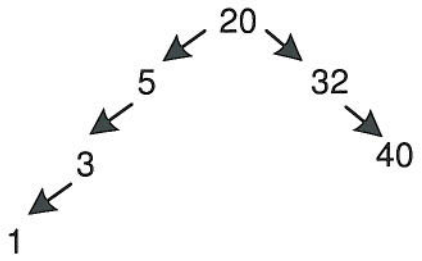

Figure 1.3 Solution 1ci.

Figure 1.4 Solution 1cii.

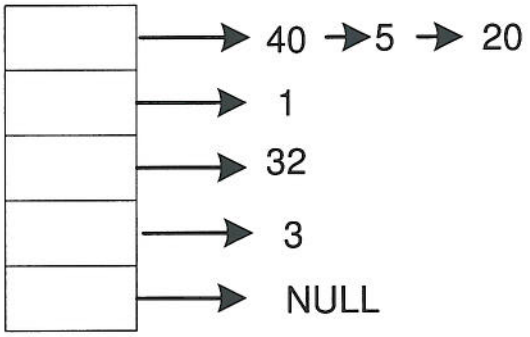

Figure 1.5 Solution 1ciii.

[1]

iv) Solution in Figure 1.6. A single right rotation is required having node 5 as root and node 3 as pivot.

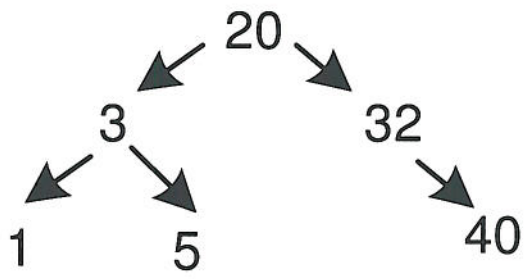

Figure 1.6 Solution 1civ.

[1]

d) i) Solution in Figure 1.7.

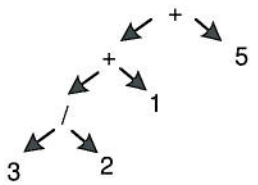

Figure 1.7 Solution 1di.

[1]

ii) Solution in Figure 1.8.

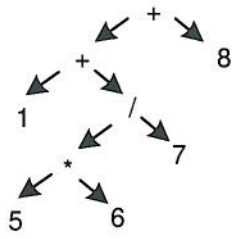

Figure 1.8 Solution 1dii.

[1]

e) At point A: $x = 20$, $y = 10$ and $z = 1$. At point B: $x = 20$, $y = 21$ and $z = 1$. There is also a memory leak because there is no pointer any more pointing to the dynamic allocated memory. The students are expected to provide comments along the lines of the code.

[3]

f) i) Solution in Figure 1.9.

```
NodePtr FindNode(NodePtr hdlist, int N) {
    if (hdlist == NULL)
        return NULL;
    else
        if (hdlist->data == N)
            return hdlist;
        else
            return FindNode(hdlist->next, N);
}
```

Figure 1.9 Solution 1f (i).

[2]

ii) Solution in Figure 1.10.

```
NodePtr FindNodeI(NodePtr hdlist, int N) {
    while (hdlist != NULL) {
        if (hdlist->data == N)
            return hdlist;
        else
            hdlist = hdlist -> next;
    }
    return NULL;
}
```

Figure 1.10 Solution 1f (ii).

[1]

2. **(This question tests students' ability to construct abstract data types and work with AVL trees.)**

a) Solution in Figure 2.1.

```
struct Node {
    int data;
    int BF;
    Node * Left;
    Node * Right; };

(optional)
typedef Node* NodePtr;
```

Figure 2.1 Solution 2a.

[3]

b) Solution in Figure 2.2.

```
int CalcHeight(NodePtr Tree) {
    int HeightL = -1;
    int HeightR = -1;

    if (Tree == NULL)
        return (-1);

    if (Tree->Left != NULL)
        HeightL = CalcHeight(Tree->Left);

    if (Tree->Right != NULL)
        HeightR = CalcHeight(Tree->Right);

    if (HeightL > HeightR)
        return HeightL + 1;
    else
        return HeightR + 1;
}
```

Figure 2.2 Solution 2b.

[3]

c) Solution in Figure 2.3.

[3]

d) Solution in Figure 2.4.

[5]

```
void UpdateNodeBF(NodePtr Tree) {
    Tree->BF = CalcHeight(Tree->Right) - CalcHeight(Tree->Left);
}
```

Figure 2.3 Solution 2c.

```
void UpdateTreeBF(NodePtr Tree) {
    if (Tree != NULL) {
        UpdateTreeBF(Tree->Left);
        UpdateNodeBF(Tree);
        UpdateTreeBF(Tree->Right);
    }
}
```

Figure 2.4 Solution 2d.

e) Solution in Figure 2.5.

```
NodePtr SingleRotL(NodePtr Tree) {
    NodePtr Temp = Tree->Right;
    Tree->Right = Tree->Right->Left;
    Temp->Left = Tree;
    return Temp;
}
```

Figure 2.5 Solution 2d.

[3]

f) Solution in Figure 2.6. The students are expected to provide also a single right rotation function, which is similar to the previous question.

[3]

```
NodePtr DoubleRotationLR(NodePtr Tree) {
    Tree->Left = SingleRotL(Tree->Left);
    return SingleRotR(Tree);
}
```

Figure 2.6 Solution 2d.

3. **(This question tests students' ability to manipulate a general data structure.)**

a) Solution in Figure 3.1.

```
struct Node {
    int id;
    Node * Left;
    Node * Right;
};
(optional)
typedef Node* NodePtr;
```

Figure 3.1 Solution 3a.

[3]

b) Solution in Figure 3.2. Initializations of *found* is false.

[3]

c) Solution in Figure 3.3.

[4]

d) Solution in Figure 3.4.

[5]

e) Solution in Figure 3.5.

[5]

```
void CheckDirectImpact(NodePtr Tree, int id1, int id2, bool &found) {
    if (Tree == NULL)
        return;
    else
        if (Tree->id == id1){
            if (Tree->Left != NULL)
                if (Tree->Left->id == id2)
                    found = true;
            if (Tree->Right != NULL)
                if (Tree->Right->id == id2)
                    found = true;
        }
        else {
            CheckDirectImpact(Tree->Left,id1,id2,found);
            CheckDirectImpact(Tree->Right,id1,id2,found);
        }
}
```

Figure 3.2 Solution 3b.

```
NodePtr FindId(NodePtr Tree, int id) {
    if (Tree == NULL)
        return NULL;
    else
        if (Tree->id == id)
            return Tree;
        else {
            NodePtr Temp1 = FindId(Tree->Left,id);
            if (Temp1==NULL)
                Temp1 = FindId(Tree->Right,id);
            return Temp1;
        }
}
```

Figure 3.3 Solution 3c.

```
bool FindPath(NodePtr Tree, int id1, int id2) {
    NodePtr T1 = FindId(Tree, id1);
    if (T1 != NULL) {
        NodePtr T2 = FindId(T1, id2);
        if (T2 != NULL)
            return true;
        else
            return false;
    } else
        return false;
}
```

Figure 3.4 Solution 3d.

```
void FindShortestPath(NodePtr Tree, int id, int Cdist, int &dist){
    if (Tree == NULL)
        return;
    else {
        if (Tree->id == id) {
            if (Cdist <= dist)
                dist = Cdist;
        }
        FindShortestPath(Tree->Left, id, Cdist+1, dist);
        FindShortestPath(Tree->Right, id, Cdist+1, dist);
    }
}
```

Figure 3.5 Solution 3e.