IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2012

EEE PART II: MEng, BEng and ACGI

ALGORITHMS AND DATA STRUCTURES

Monday, 11 June 2:00 pm

Time allowed: 1:30 hours

There are TWO questions on this paper.

Answer BOTH questions.

Question One carries 40% of the marks. Question Two carries 60%.
This exam is OPEN BOOK.

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

Examiners responsible First Marker(s) : C. Bouganis
Second Marker(s) : D.B. Thomas

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. 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 a non-negative integer (e.g. $f(0) = 0$).

$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ 2*f(n-1) + f(n-2) & n \geq 2 \end{cases} \quad (1.1)$$

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

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

Figure 1.1 calculateF() function.

[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, 8, 5, 20, 3}

[2]

ii) Comment whether or not the tree of part (i) is balanced. If the tree is unbalanced, balance it using single and/or double rotations and draw the resulting tree.

[2]

iii) Insert the number 6 in the resulting tree from part (ii) and draw the final tree.

[2]

iv) Comment whether or not the tree of part (iii) is balanced. If the tree is unbalanced, balance it using single and/or double rotations and draw the resulting tree.

[2]

v) The following set of numbers is stored in an array structure in the given order.

{1, 2, 3, 4, 5}

The structure is to be sorted using the heap sort algorithm. Draw the heap tree (Hint: the maximum number should be at the root).

[2]

vi) Draw the resulting tree, when the root of the heap tree is deleted.

[2]

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

i) $1+2+3+4$

[2]

ii) $(4+5)*(6+7)/3$

[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 at points A and B of the code. With justification, state whether this code segment has a memory leak or not.

```
    int x=10;
    int y=20;
    int *p1 = &x;
    int *p2 = &y;
    *p1 = *p1 + *p2;
    y= *p2;
A
    p1 = new int;
    *p2 = 10;
    p1 = p2;
    x = *p1 + x;
B
```

Figure 1.2 Code segment.

[5]

[continued on the following page]

f) Figure 1.3 shows the type declaration for a dynamic linked list, where each node stores an *id*, which is unique, and *data*. Both take positive integer values. An empty list is a valid instance of the data structure.

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

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

Figure 1.3 Linked list declaration.

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

[3]

ii) Write a C++ function/procedure that takes as input the *hdList* pointer and an *id* value, and increments by 1 the *data* field of the node with such *id*.

[4]

2. Consider the an ordered binary tree structure, where each node of the tree structure can store a unique *id* value, which takes non-negative values, and a pointer to a linked list. Each linked list, can store a set of positive integers. Also, it is legal for the list to be empty. The unique *id* value is used for the ordering of the binary tree structure. Note that no ordering has been imposed to the linked list structures. An instance of the above tree structure is shown below.

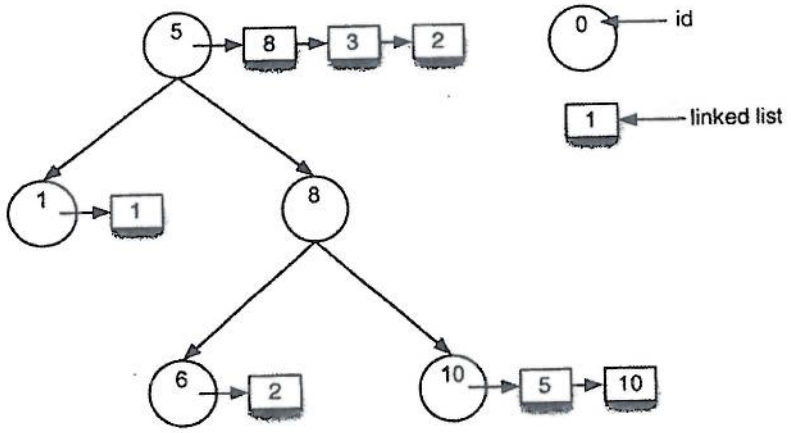


Figure 2.1 Binary tree structure.

a) Define a structure *treeNode* capable of representing a node of the tree. You are allowed to define auxiliary structures.

[10]

b) Write a recursive function/procedure that takes as input the pointer to the root of the tree and returns the *id* of the deepest node in the tree structure whose linked list structure has at least one element. Show how your recursive function/procedure will be invoked. You can always pass more input arguments in your function/procedure. You may use auxiliary functions/procedures.

[10]

c) Write a recursive function/procedure that takes as input a pointer to the root of the tree, an *id* value and a positive integer. The function/procedure should insert the positive integer to the linked list of the node with the input *id*. In the case where such node does not exist, your function/procedure should add a new such node in the binary tree structure without destroying the ordering. Show how your recursive function/procedure will be invoked. You can always pass more input arguments in your function/procedure. You may use auxiliary functions/procedures.

[20]

[continued on the following page]

d) Write a recursive function/procedure that takes as inputs a pointer to the root of the tree, and an *id* value, and returns the sum of the elements that have been stored in the linked list of the node with the input *id*. In the case where such node does not exist, your function/procedure should return the value -1, and if the node exists but its linked list is empty, your function/procedure should return the value 0. For example, for the given instance of the tree, for node with *id*=10, the function/procedure should return 15. Show how your recursive function/procedure will be invoked. You can always pass more input arguments in your function/procedure. You may use auxiliary functions/procedures.

[20]

ALGORITHMS and DATA STRUCTURES — SOLUTIONS 2012 — EE1-4/EE2-1+ — 1/9

1) a). [New Application] from

① the function should return int.

② the passing argument should be of type int

③ i in the for-loop is not declared

④ It should be i++ and not just i

⑤ in the for loop, it should be 1, and not N

⑥ if (n==0)
result = 0; ← and not 2.

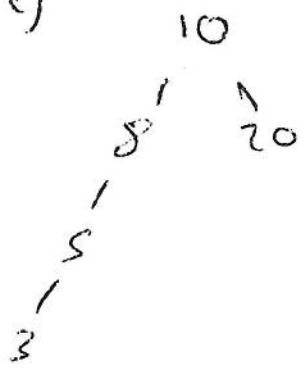
6) [6]

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

[6]

1.c) [Backwards, New examples]

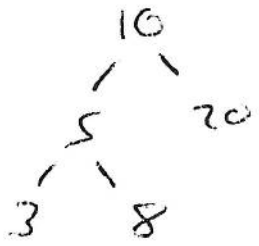
i)



[2]

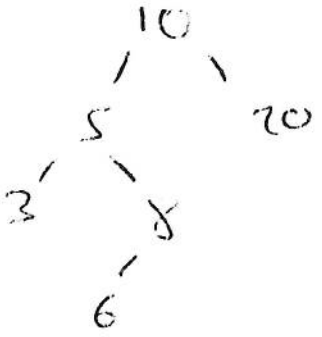
ii)

Yes, at node 8. (deepest imbalanced node)



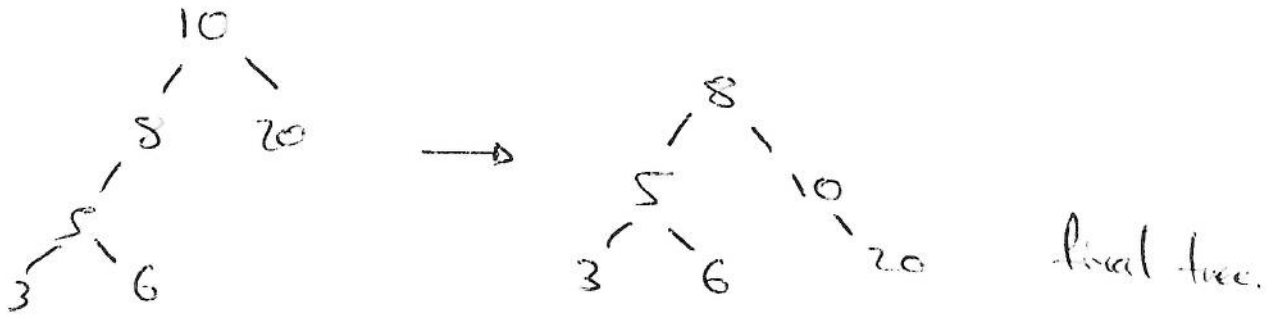
[2]

iii)



[2]

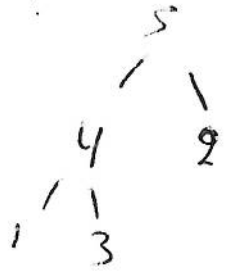
iv) Yes Needs balancing. at node 10.
Double rotation.



[2]

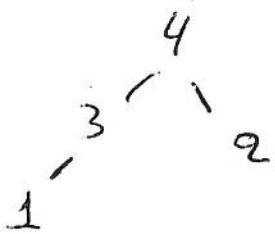
v)

final tree:



[2]

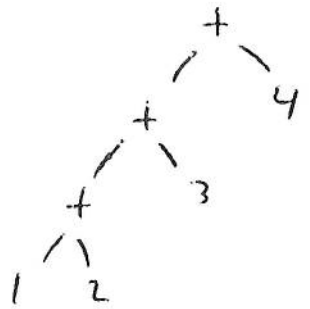
vi) Final tree



[2]

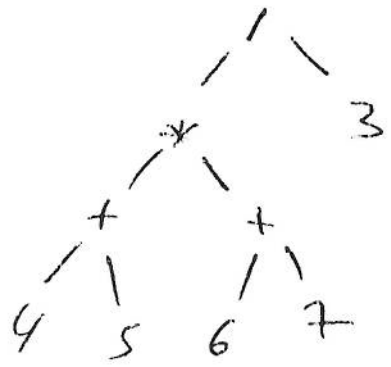
1.d).

i).



[2]

ii)



[2]

e).

A: $x = 30$, $y = 20$

B: $x = 40$, $y = 10$

There is memory leak: $p_1 = p_2$.

[5].

1. f).

i).

```
NodePtr FindLastElement (NodePtr hdlist) {

    NodePtr LastElemPointer = NULL;
    while (hdlist != NULL) {
        if (hdlist->next != NULL)
            LastElemPointer = hdlist;
        hdlist = hdlist->next;
    }
    return LastElemPointer;
}
```

[3]

ii)

```
void incNodeId (NodePtr hdlist, int id) {

    while (hdlist != NULL) {
        if (hdlist->id == id)
            hdlist->data = hdlist->data + 1;
        hdlist = hdlist->next
    }

}
```

[4]

* You can add a 'return' in the if statement to improve performance

2). a)

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

typedef listNode * listNodePtr;

struct treeNode {
        int id;
        listNodePtr idList;
        treeNode * left;
        treeNode * right;
};

typedef treeNode * treeNodePtr;
```

[10]

b) (-1)

```
void findDeepestNode(treeNodePtr hdtree, int &id, int currentDepth,
                     int &maxDepth)
{
    if (hdtree != NULL) {
        if (currentDepth >= maxDepth)
            if (hdtree->hdList != NULL) {
                id = hdtree->id;
                maxDepth = currentDepth;
            }
        findDeepestNode(hdtree->left, id, currentDepth+1, maxDepth);
        findDeepestNode(hdtree->right, id, currentDepth+1, maxDepth);
    }
}
```

[10]

```
int id = -1;
```

invoke by: `findDeepestNode(hdtree, id, ϕ, ϕ);`

c)

```
void addNodeList(listNodePtr &hdList, int data) {
    listNodePtr temp;
    temp = new ListNode
    temp->data = data;
    temp->next = hdList;
    hdList = temp;
}
```

```
void addNodeTree(treeNodePtr &hdTree, int id, int data){
    if (hdTree == NULL) {
        treeNodePtr temp = new treeNode;
        temp->id = id;
        addNodeList(temp->hdList, data);
        temp->left = NULL;
        temp->right = NULL;
    }
    else if (id == hdTree->id)
        addNodeList(hdTree->hdList, data);
    else if (id > hdTree->id)
        addNodeTree(hdTree->right, id, data);
    else
        addNodeTree(hdTree->left, id, data);
}
```

d)


```
void calcSumList(listNodePtr hdList, int &sum) {
    while (hdList != NULL) {
        sum = sum + hdList->data;
        hdList = hdList->next;
    }
}
```

```
void calcSumNode(treeNodePtr hdTree, int id, int &sum) {
    if (hdTree != NULL) {
        if (id == hdTree->id) {
            sum = 0;
            calcSumList(hdTree->hdList, sum);
        }
        else if (id > hdTree->id)
            calcSumNode(hdTree->right, id, sum);
        else
            calcSumNode(hdTree->left, id, sum);
    }
}
```

Call the function with sum = 0

[20]