IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2011

ISE PART I: MEng, BEng and ACGI

Corrected Copy

SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES

Friday, 17 June 2:00 pm

Time allowed: 1:30 hours

Q 1 e) at 2.45pm

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) :	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. 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. $f(1) = 0.5$).

$$f(n) = \begin{cases} 0 & n = 0 \\ f(n-1) + \frac{n}{2} & n > 0 \end{cases} \tag{1.1}$$

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

```
int calculateF (n) {
    result=3.0;
    for (int i=1; i <= N; i++)
        result = result + i/2.0;
}
```

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.

$\{5, 10, 20, 4\}$

[2]

ii) Comment whether or not the tree of part (i) needs balancing. If the tree needs to be balanced, draw the resulting tree.

[2]

iii) Insert the number 21 in the resulting tree from part (ii) and draw the final tree.

[2]

iv) Comment whether or not the tree of part (iii) needs balancing. If the tree needs to be balanced, draw the resulting tree.

[2]

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

$\{20, 10, 30, 40, 35, 5\}$

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) $4 * 5 * (6 + 7)$

[2]

ii) $(4 + 5) * 6 + 3/2$

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

2.45pm

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.

```
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 an *id* value, and checks whether a node with such *id* exists in the list.

[3]

ii) Write a C++ function/procedure that takes as input the *hdList* pointer, and returns the pointer to the node with the maximum *data* value in the list. If the list is empty, the function/procedure should return NULL.

[4]

2. Consider the computer network of a company. Assume that the network can be represented using a binary tree structure. Figure 2.1 illustrates an example of such a tree. There are two types of nodes. The internal nodes model the routers in the network, where the leaf nodes model the computers. Each node has a unique *id* in the whole network. Each edge on the tree represent a link between two routers or a router and a computer. The number on the edges represent the maximum bandwidth that can be achieved over the link, and takes integer values. Assume that the structure can be accessed only by the main router, which is the root of the tree (i.e. node with $id = 0$). The circles represent the routers, where the squares represent the computers.

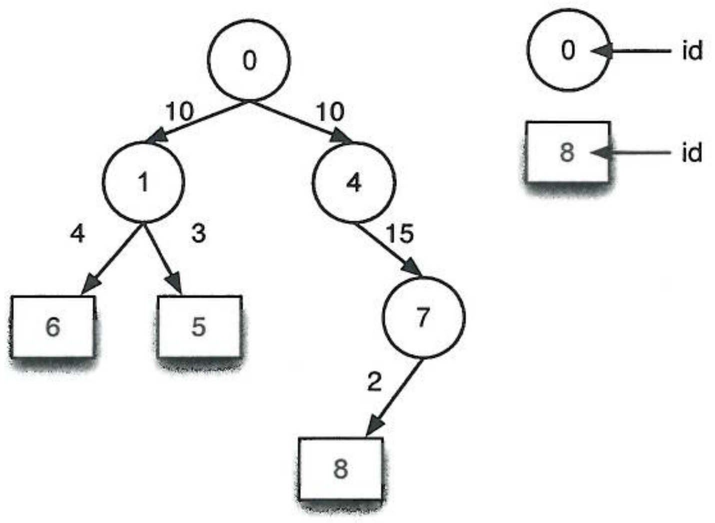


Figure 2.1 Network.

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

[10]

b) Write a recursive function/procedure that takes as input a pointer to the root of the tree and returns the number of routers in the network. In the case where the tree is empty, the function/procedure should return the value 0. Show how your recursive function/procedure will be invoked. You can always pass more input arguments in your function/procedure.

[10]

c) Write a recursive function/procedure that takes as input the pointer to the root of the tree and returns the number of routers that are fully utilised (i.e. all connections are used). For this example, there are two fully utilised routers in the network (nodes 0 and 1). Show how your recursive function/procedure will be invoked. You can always pass more input arguments in your function/procedure.

[10]

[continued on the following page]

d) Write a recursive function/procedure that takes as input a pointer to the root of the tree, and an *id* value that belongs to a computer, and returns the number of links that the computer is away from the main router (i.e. root of the tree). For example, for node with *id*=8, the function/procedure should return 3. Show how your recursive function/procedure will be invoked. You can always pass more input arguments in your function/procedure.

[15]

e) Write a recursive function/procedure that takes as input a pointer to the root of the tree, and an *id* value that belongs to a computer, and returns the maximum achievable bandwidth between that computer and the main router (i.e. root of the tree). For example, for node with *id*=8, the function/procedure should return 2. Show how your recursive function/procedure will be invoked. You can always pass more input arguments in your function/procedure.

[15]

1) a)

The correct code is:

```
①               ②
float calculateF(int n) {
    ④             ⑤
    float result = 0;
                          ③
    for (int i = 1; i <= n; i++)
        result = result + i / 2.0;
    return result; ⑥
}
```

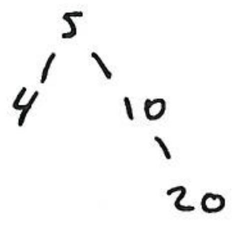
b)

[6]

```
float recCalculateF(int n) {
    if (n==0)
        return 0;
    else
        return recCalculateF(n-1) + n / 2.0;
}
```

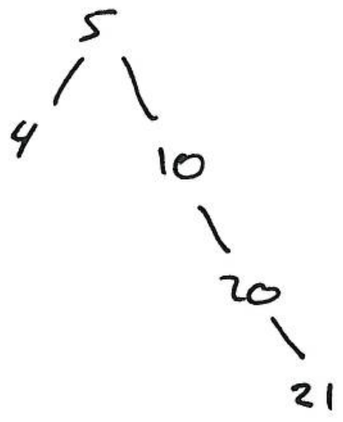
[6]

c) i)



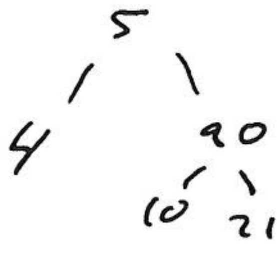
ii) The above tree is already balanced. [2]

iii) [2]



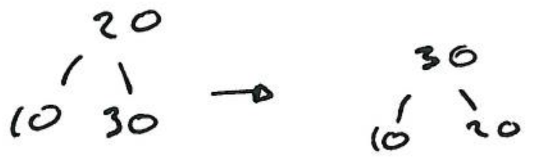
iv) [2]

The tree in part (iii) is not balanced in node 10. Perform a left rotation, resulting in a balanced tree.

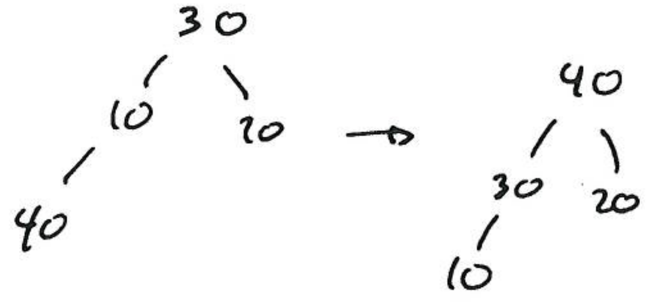


[2]

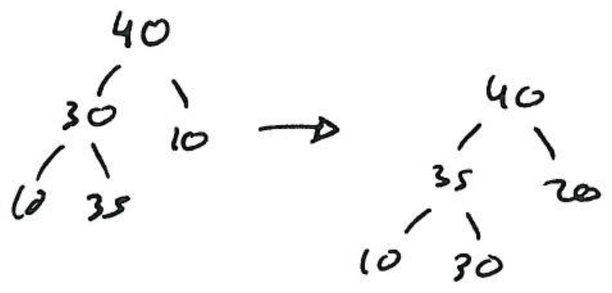
v)



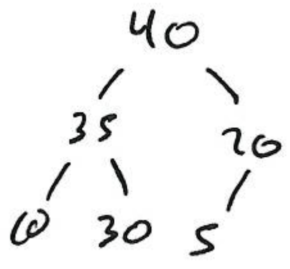
(+30)



(+40)



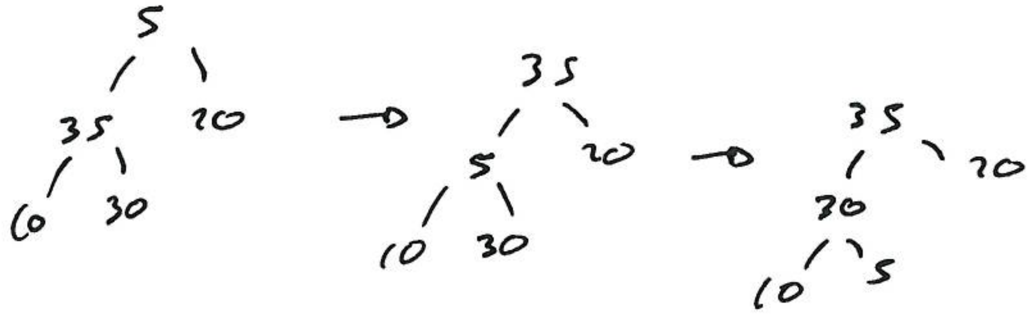
(+35)



(+5)

[2]

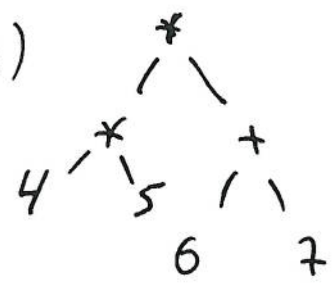
vi)



[2]

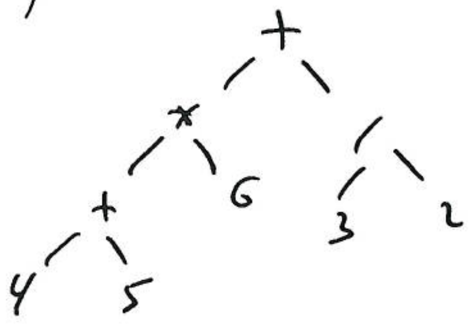
d)

i)



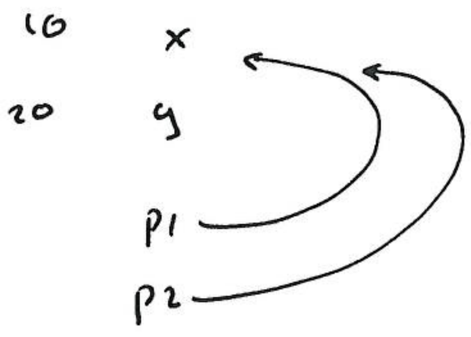
[2]

ii)



[2]

e)



$*p1 = *p1 + *p2 \Longrightarrow x = 20$, A: $x=20$, $y=20$

$p1 \rightarrow$ somewhere

$y = y + *p2 \Longrightarrow y = y + 20$, B: $x=20$, $y=40$

$\Longrightarrow y = 40$

[5]

7) i)

```
void checkId(NodePtr hdList, int id, bool &found
        if (hdList != NULL)
            if (hdList->id == id)
                found = true;
            else
                checkId(hdList->next, id, found);
}
```

[redacted]
[3]

ii)

```
void findMaxNode(NodePtr hdList, int &max, NodePtr &maxNode
     if (hdList != NULL) {
         if (hdList->data >= max) {
              max = hdList->data;
              maxNode = hdList;
         }
         findMaxNode(hdList->next, max, maxNode);
}
```

call the func with max = -1, maxNode = hdList.

[4]

2) a) (new app[illegible]

```
struct Node {

        int id;
        int type;        //0 for router, 1 for computer
       Node * left;
       Node * right;
       int  bandwidthLeft;
       int  bandwidth right;
}
```

optional:

```
typedef Node * NodePtr.
NodePtr hdTree = NULL;
```

[10]

b)

```
void    numRouters (NodePtr hdTree, int &num){
     if (hdTree != NULL) {
          if (hdTree -> type == 0)
               num ++;

          numRouters (hdTree->left, num);
          numRouters (hdTree -> right, num);
     }

}
```

[10]

Initialization of num = 0

c)

```
void    findFUrouters(NodePtr kdTree, int &num) {
        if (kdTree != NULL) {
            if (kdTree->type == 4)
                if (kdTree->left != NULL) && (kdTree->right !=
                    num++

            findFUrouters(kdTree->left, num);
            findFUrouters(kdTree->right, num);
        }
    }
```

[10]

d)

```
void    findLinks(NodePtr kdTree, int id, int currlinks, int &links) {
    if (kdTree != NULL) {
        if (kdTree->id == id)
            links = currlinks;
        findLinks(kdTree->left, id, currlinks+1, links);
        findLinks(kdTree->right, id, currlinks+1, links);
    }

}
```

[15]

e)

```
void   maxBavail (NodePtr hdTree, int id, int band, int &mb
    int  temp;
    if  (hdTree != NULL)  {

        if (id == hdTree->id)
                mband = band;
        // check left
        if  (band ≥ hdTree->bandwidthleft)
                temp = hdTree->bandwidthleft
        else
                temp = band;
        maxBavail ( hdTree->left, id, temp, mband);
        // check right
        if    (band ≥ hdTree->bandwidthright)
              temp = hdTree->bandwidthright
        else
              temp = band;
        maxBavail ( hdTree->right, id, temp, mband);
    }

}
```

[15]