E1.8
E2.7A
E2.18

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2007

ISE PART I/EEE PART II

**SOFTWARE ENGINEERING: INTRODUCTION, ALGORITHMS AND DATA STRUCTURES**

Friday, 8 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) : G.A. Constantinides

**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++ function that computes the $N^{th}$ term of the following arithmetic progression: $a_1 = 1$, $a_{k+1} = a_k + b$, with $k \geq 1$ where $b$ is an integer. For example for $b = 2$ and $N = 3$, the function should return a value of 5. Identify five errors in the C++ code.

```
void calculateNterm (N, int b) {
    int i;
    int ak = 1;
    for (i=0; i < N; i=i+2);
        ak = ak + b;
    return ak;
}
```

Figure 1.1 calculateNterm function.

[ 6 ]

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

[ 6 ]

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

- {10, 23, 5, 80, 100, 9, 15}
- {10, 23, 50, 80, 90, 100}

[ 4 ]

ii) An alternative data structure to an ordered binary tree is an ordered link list. Repeat the same process as in part (i) for an ordered list (ascending order) and draw the list for both sets.

[ 3 ]

iii) Compare the two data structures in terms of the average number of operations required to find an item, and in terms of memory efficiency, for both sets of numbers.

[ 3 ]

d) Construct a parse tree for the following expressions:

i) $(4*3+12)*6$ [ 2 ]

ii) $3*(5+4)/7*8$ [ 2 ]

[continued on the following page]

e) Consider the C++ code segment in Figure 1.2. With justification, state the values of variables *x* and *y* after the code segment is executed. The code for the function *swap ()* is shown in Figure 1.3.

```
int x=10;
int y=20;
int *px = &x;
int *py;
py = new int;
*py = y;
swap (x, y);
*px = *px + 1;
*py = *py + 10;
```

Figure 1.2 Code segment.

```
void swap (int& x, int& y) {
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Figure 1.3 swap () function.

[ 7 ]

f) Figure 1.4 shows the type declaration for a dynamic linked list of integers in C++. Write a C++ recursive function that takes as one of its inputs the hdList pointer and returns the sum of the data values in the list.

```
class Node {
    public:
        int data;
        Node *next;
};

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

Figure 1.4 Linked list declaration.

[ 7 ]

2. Figure 2.1 illustrates a state machine. The state machine takes one bit as input and its output is one bit. Every state has a unique *state number*.

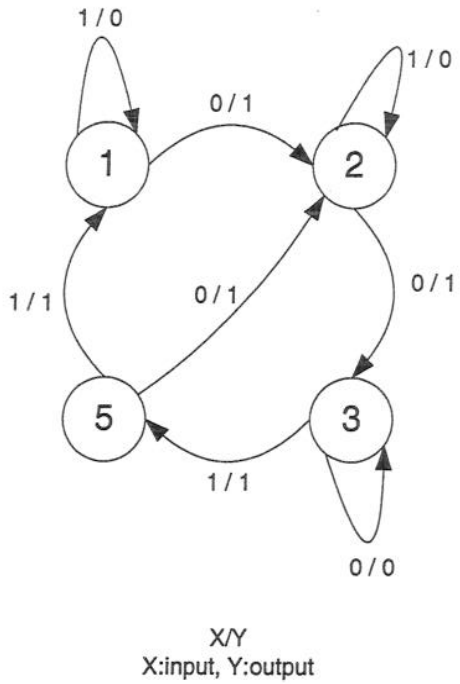


Figure 2.1 State machine.

a) Define a structure *State* capable of representing a state of the state machine. (Hint: You should be able to represent the next state and output for the two different values of the input).

[ 15 ]

b) Write a recursive function that takes as inputs a pointer to an initial state, and a string *S*. The string *S* represents the value of the input at each clock cycle. The function should return the pointer to the final state. The following two functions are available:

```
bool notEmpty(const string& expression);
//returns true if expression is not the empty string

char getNextChar(string& expression);
//returns next character in expression, character is consumed
```

[ 15 ]

[continued on the following page]

c) Modify your function in the previous question to also return the corresponding values of the output $Y$ in a string. The first character in the output string should correspond to the output signal of the starting state. The following function is available:

```
void addChar(string& expression, char c);
//adds the character c to the end of the string
```

[ 15 ]

d) Write a function that takes as input a pointer to an initial state, and returns the number of the states in the state machine that can be reached from that initial state. Add, if it is necessary, an extra field in your structure *State*.

[ 15 ]

3. Figure 3.1 illustrates how the clock signal is distributed in a device. Every arrow in the figure corresponds to a wire, and every node represents a connection between wires. You can assume that every node has a unique number (*id*). The number on each arrow represents the delay of the signal along the corresponding wire. The delay on each wire is always an integer. Assume that each node connects an incoming wire to two outgoing wires. The node 0 represents the start of the clock distribution.

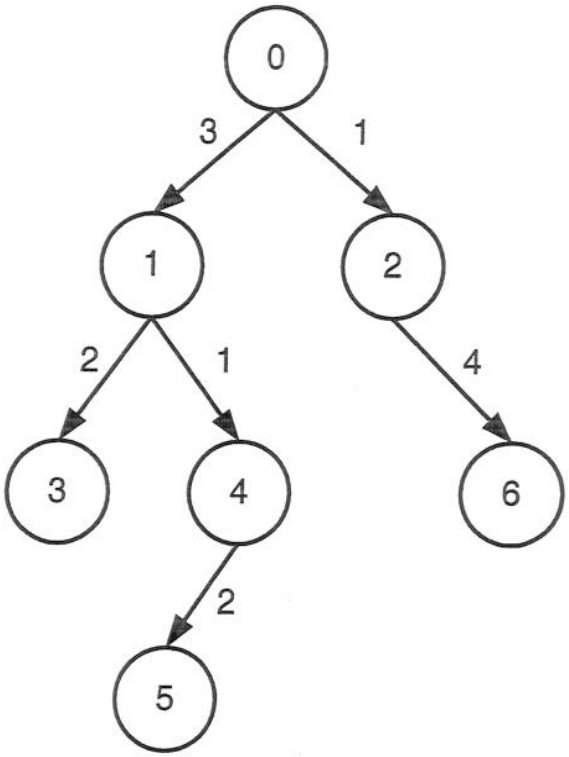


Figure 3.1 Clock distribution.

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

[ 15 ]

b) Write a function *CalculateDelay* that takes as input a node number and the pointer to node 0, and calculates the total delay from the start of the clock (node 0) to that node. For example, for the node with $id = 5$, the total delay is $3+1+2=6$.

[ 20 ]

c) Write a function that finds the node with the maximum delay from node 0. The function should also return the *id* of the node that corresponds to that maximum delay. The function should take as input the pointer to node 0.

[ 25 ]

E1&/E27A/
E2.1&

# SOLUTIONS 2007

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

a) The correct code is shown in Figure 1.1.

```
int calculateNterm (int N, int b) {
    int i;
    int ak = 1;
    for (i=1; i < N; i=i+1)
        ak = ak + b;
    return ak;
}
```

Figure 1.1 Solution 1a.

[6]

b) The solution is shown in Figure 1.2.

```
int calculateNtermRec (int N, int b) {
    if (N==1)
        return 1;
    else
        return calculateNtermRec (N-1, b) + b;
}
```

Figure 1.2 Solution 1b.

[6]

c) i) Solution in Figure 1.3.

[4]

ii) Solution in Figure 1.4.

[3]

iii) First set of numbers. The tree structure is more efficient in finding an item on average and it uses more memory than the list structure.

Second set of numbers. The tree structure requires the same number of operations on average to find an item as the list structure. It also uses more memory than the list structure.

[3]

d) i) Solution in Figure 1.5.

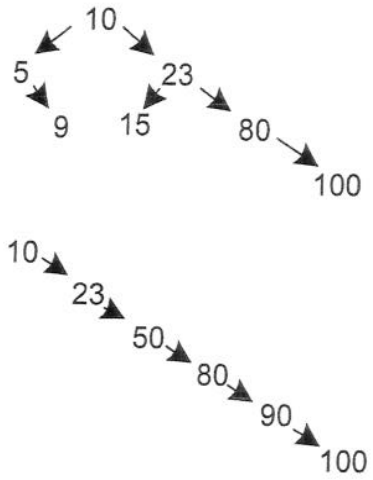


Figure 1.3 Solution 1ci.

5 → 9 → 10 → 15 → 23 → 80 → 100

10 → 23 → 50 → 80 → 90 → 100

Figure 1.4 Solution 1cii.

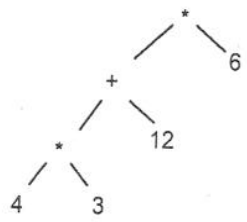


Figure 1.5 Solution 1di.

[ 2 ]

ii) Solution in Figure 1.6.

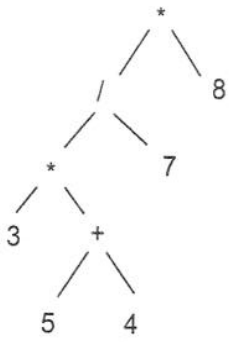


Figure 1.6 Solution 1dii.

[ 2 ]

e) The *px* pointer points to *x*, where the *py* pointer points to a copy of *y*. Thus, $x = 21$ and $y = 10$

[ 7 ]

f) Solution in Figure 1.7.

```
int SumRec (NodePtr hdList){
    if (hdList == NULL)
        return 0;
    else
        return SumRec(hdList->next) + hdList->data;

}
```

Figure 1.7 Solution 1f.

[ 7 ]

2. **(This question tests students' ability to construct abstract data types. The first three parts can be answered without any knowledge about the existence of cycles. The fourth part is more challenging, since the existence of cycles should be handled. Directed cyclic graphs have not been covered as an example during the course.)**

a) Solution in Figure 2.1.

```
class State {
    public:
        int stateID;
        State *nextStateHigh;
        State *nextStateLow;
        char outputHight;
        char outputLow;
};

(optional)
typedef State* StatePtr;
```

Figure 2.1 Solution 2a.

[ 15 ]

b) Solution in Figure 2.2.

```
StatePtr processInput (StatePtr ST, string& S) {
    char c;
    if (notEmpty(S)) {
        c = getNextChar(S);
        if (c=='1')
            ST = ST->nextStateHigh;
        else
            ST = ST->nextStateLow;
        return processInput(ST, S);
    }
    else
        return ST;
}
```

Figure 2.2 Solution 2b.

[ 15 ]

c) Solution in Figure 2.3.

[ 15 ]

d) Solution in Figure 2.4. Add the following field in the structure: **int** visited;, and initialize it to zero. If this variable has value 1, it means that the state has been visited, otherwise has not.

```
StatePtr processInputOutput (StatePtr ST, string& S, string& output) {
    char c;
    char out;
    if notEmpty(S) {
        c = getNextChar(S);
        if (c=='1') {
            out = ST->outputHigh;
            ST = ST->nextStateHigh;
        }
        else {
            out = ST->outputLow;
            ST = ST->nextStateLow;
        }
        addChar(output, out);
        return processInputOutput(ST, S, output);
    }
    else
        return ST;
}
```

Figure 2.3 Solution 2c.

```
void stateNumber (StatePtr ST, int& N) {
    if (ST->visited == 0) {
        ST->visited =1;
        N = N + 1;
        stateNumber (ST->nextStateHigh, N);
        stateNumber (ST->nextStateLow, N);
    }
}
```

Figure 2.4 Solution 2d.

3. **(This question tests students' ability to manipulate a binary tree data structure. The challenging part of the question is that the students should handle the information on the edges too.)**

a) Solution in Figure 3.1.

```
class CT {
    public:
        int id;
        CT *left;
        CT *right;
        int leftWeight;
        int rightWeight;
};

(optional)
typedef CT* CTPtr;
```

Figure 3.1 Solution 3a.

[ 15 ]

b) Solution in Figure 3.2.

```
void calculateDelay (CTPtr node, int id, int N, int& Delay, bool& found) {
    found = false;
    if (node->id == id) {
        Delay = N;
        found = true;
    }
    else {
        if ( (!found) && (node->left != NULL) )
            calculateDelay (node->left, id, N + node->leftWeight, Delay, found);
        if ( (!found) && (node->right != NULL) )
            calculateDelay (node->right, id, N + node->rightWeight, Delay, found);
    }
}
```

Figure 3.2 Solution 3b.

[ 20 ]

c) Solution in Figure 3.3.

[ 25 ]

```
void maximumDelay (CTPtr node, int currentDelay, int& id, int& maxDelay) {
    if (maxDelay < currentDelay) {
        maxDelay = currentDelay;
        id = node->id;
    }
    if (node->left != NULL)
        maximumDelay (node->left, currentDelay + node->leftWeight, id, maxDelay);
    if (node->right != NULL)
        maximumDelay (node->right, currentDelay + node->rightWeight, id, maxDelay);
}
```

Figure 3.3 Solution 3c.