E1.8 - Jk 1
E2.18 - EE2

SOLUTIONS 2008 - JUNE

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=2; i <= n; i++)
        result = result + (i-1);
    return result;
}
```

Figure 1.1 Solution 1a.

[3]

b) The solution is shown in Figure 1.2.

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

Figure 1.2 Solution 1b.

[3]

c) i) Solution in Figure 1.3.

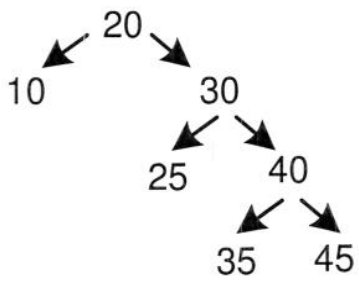

Figure 1.3 Solution 1ci.

[2]

ii) Solution in Figure 1.4.

Figure 1.4 Solution 1cii.

[1]

iii) Solution in Figure 1.5.

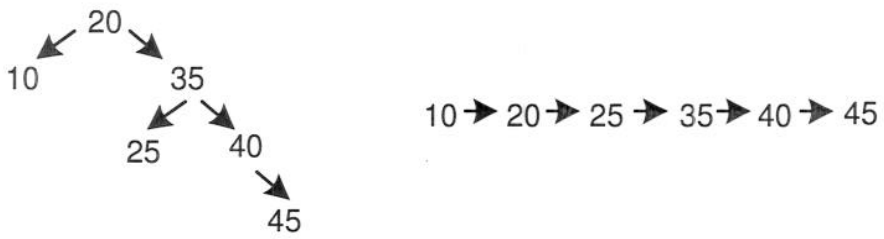

Figure 1.5 Solution 1ciii.

[1]

iv) In the case of an ordered list we need to search half the list on average to delete an item. In the case of a binary tree, the search for the item is faster, but the deletion operation may need many operations depending on the configuration of the tree.

[2]

d) i) Solution in Figure 1.6.

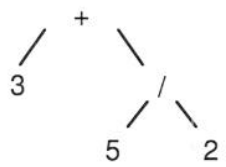

Figure 1.6 Solution 1di.

[1]

ii) Solution in Figure 1.7.

[1]

e) The *px* pointer points to a new location and never to *x*. *py* pointer points to *y*. Thus, $x = 1$ and $y = 10$.

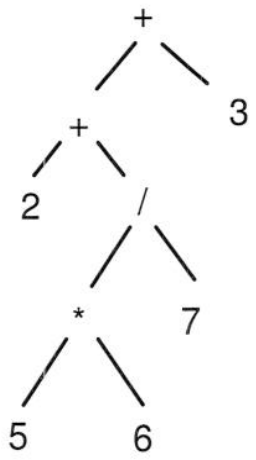

Figure 1.7 Solution 1dii.

[3]

f) i) Solution in Figure 1.8.

```
int CountZeroItems (NodePtr hdList){
    if (hdList != NULL)
        if (hdList->data == 0)
            return CountZeroItems (hdList->next) + 1;
        else
            return CountZeroItems (hdList->next);
    else
        return 0;
}
```

Figure 1.8 Solution 1f (i).

[2]

ii) Solution in Figure 1.9.

[1]

```
int CountZeroItemsIter (NodePtr hdList){
    int counter = 0;
    while (hdList != NULL){
        if (hdList->data == 0)
            counter = counter + 1;
        hdlist = hdList->next;
    }
    return counter;
}
```

Figure 1.9 Solution 1f (ii).

2. **(This question tests students' ability to construct abstract data types.)**

a) Solution in Figure 2.1.

```
class RNAnode {
    public:
        char data;
        RNAnode *next;
};

(optional)
typedef RNAnode* RNAnodePtr;
```

Figure 2.1 Solution 2a.

[5]

b) Solution in Figure 2.2.

```
int NumApp (RNAnodePtr hdlist, char b) {
    int count=0;
    while (hdlist != NULL) {
        if (hdlist->data == b)
            count++;
        hdlist = hdlist->next;
        }
    return count;
}
```

Figure 2.2 Solution 2b.

[5]

c) Solution in Figure 2.3.

[5]

```
bool ChekSeq(RNAnodePtr hdlist){
    if (hdlist==NULL)
        return false;
    else if (hdlist->next == NULL)
        return false;
    else {
        RNAnodePtr searchPtr = hdlist->next;
        RNAnodePtr oldPtr = hdlist;
        while (searchPtr != NULL) {
            if ((searchPtr->data == 'U') && (oldPtr->data == 'T'))
                return true;
            else {
                oldPtr = searchPtr;
                searchPtr = searchPtr->next;
            }
        }
    }
}
```

Figure 2.3 Solution 2c.

d) Solution in Figure 2.4.

[5]

```
int NumAppSeq(RNAnodePtr hdlist){
    int count = 0;
    if (hdlist==NULL)
        return count;
    else if (hdlist->next == NULL)
        return count;
    else {
        RNAnodePtr searchPtr = hdlist->next;
        RNAnodePtr oldPtr = hdlist;
        while (searchPtr != NULL) {
            if ((searchPtr->data == 'U') && (oldPtr->data == 'T'))
                count++;
            oldPtr = searchPtr;
            searchPtr = searchPtr->next;
        }
    }
    return count;
}
```

Figure 2.4 Solution 2d.

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

a) Solution in Figure 3.1.

```
class Node {
    public:
        int id;
        Node * left;
        Node * right;
};

(optional)
typedef Node* NodePtr;
```

Figure 3.1 Solution 3a.

[5]

b) Solution in Figure 3.2. Initializations of *counter* is zero, and *enable* is false.

[5]

c) Solution in Figure 3.3. Initializations of *counter* is zero, and *M* is zero.

[5]

d) Solution in Figure 3.4. Initialization of *counter* is zero.

[5]

```
void EvolutionId(NodePtr hdlist, int id, bool enable, int & counter) {
    if (enable)
        counter++;
    if (hdlist->id == id)
        enable = true;
    if (hdlist->left != NULL)
        EvolutionId(hdlist->left, id, enable, counter);
    if (hdlist->right != NULL)
        EvolutionId(hdlist->right, id, enable, counter);
}
```

Figure 3.2 Solution 3b.

```
void EvolutionN(NodePtr hdlist, int M, int N, int & counter) {
    if (M==N)
        counter++;
    if (hdlist->left != NULL)
        EvolutionN(hdlist->left, M+1, N, counter);
    if (hdlist->right != NULL)
        EvolutionN(hdlist->right, M+1, N, counter);
}
```

Figure 3.3 Solution 3c.

```
void NumOneInst(NodePtr hdlist, int & counter) {
    if ( ((hdlist->left == NULL) && (hdlist->right != NULL)) ||
        ((hdlist->left != NULL) && (hdlist->right == NULL)) )
        counter++;
    if (hdlist->left != NULL)
        NumOneInst(hdlist->left, counter);
    if (hdlist->right != NULL)
        NumOneInst(hdlist->right, counter);
}
```

Figure 3.4 Solution 3d.