

Solutions ~~Summer~~ 2013

1.a)

```
void calculateF(N) {  
    int result = 0;  
    for (i = 0; i <= n; i++) {  
        result = result + n;  
    }  
    return result;  
}
```

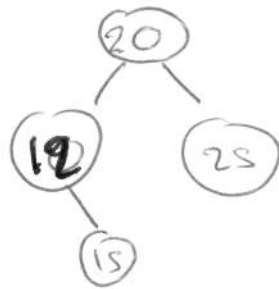
Annotations: (1) points to `int` before `calculateF`; (2) points to `n` in `i <= n`; (3) points to `int` before `result`; (4) points to the closing brace of the `for` loop; (5) points to `i` in `i++`; (6) points to `int` before `N`.

[6]

b)

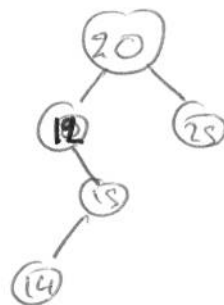
```
int calculateFR (int n) {  
    if (n == 0)  
        return 0;  
    else  
        return calculateFR(n-1) + n;  
}
```

c) i)



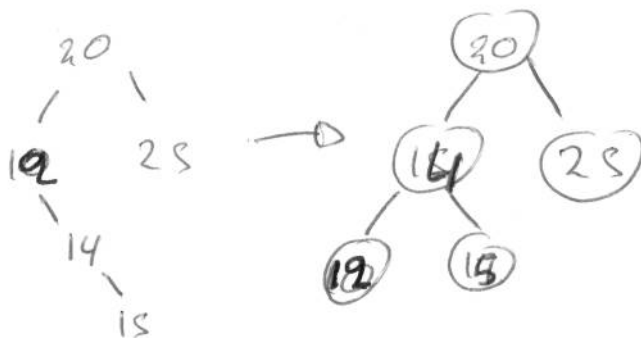
ii) The tree in part (i) is balanced as all the BFs are within the range $[-1, 1]$.

iii)



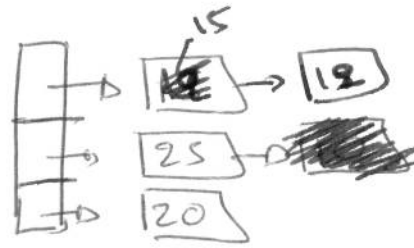
iv) The tree is not balanced. Resulting tree.

(double rotation).



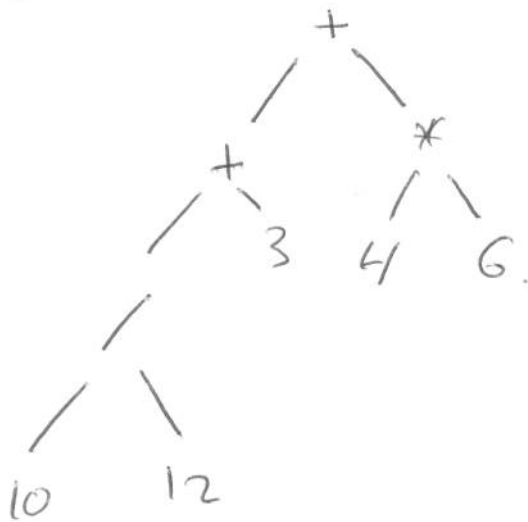
v) 2

vi)

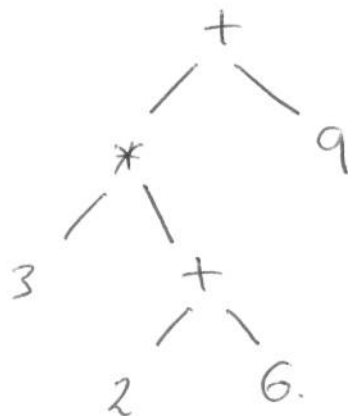


$$\begin{aligned} 20 \bmod 3 &= 2 \\ 12 \bmod 3 &= 0 \\ 25 \bmod 3 &= 1 \\ 15 \bmod 3 &= 0 \end{aligned}$$

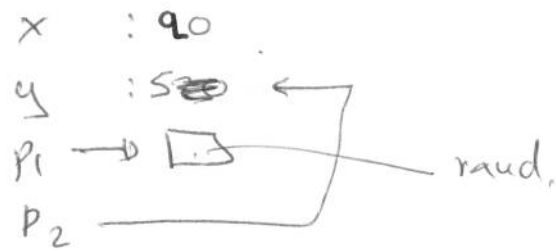
d) i)



ii)



e)



$y = *p_1$: assigns a random number.

A: y : random num
 $x = 20$

$p_1 \rightarrow x$

$p_2 \rightarrow x$

$x = 2 \cdot x$

$y = *p_2 = 2 \cdot x$

B: $x = 40$
 $y = 40$

memory leak at $p_1 = 2x$.

f) i)

```
NodePtr returnPointer (NodePtr hdlst) {  
    while (hdlst != NULL) {  
        if (hdlst->data >= 10)  
            return hdlst;  
        hdlst = hdlst->next;  
    }  
    return hdlst;  
}
```

ii)

```
NodePtr void incOne (NodePtr hdlst) {  
    while (hdlst != NULL) {  
        hdlst->data = hdlst->data + 1;  
        hdlst = hdlst->next;  
    }  
}
```

2)

a)

```
struct treeNode {
    int id;
    int age;
    int salary;
    treeNode * left;
    treeNode * right;
}
```

```
typedef treeNode * treeNodePtr; (optional)
```

b)

```
void numEupl (treeNodePtr hdTree, int &count) {
    if (hdTree != NULL) {
        count++;
        numEupl (hdTree->left, count);
        numEupl (hdTree->right, count);
    }
}
```

c)

```
void numEupl numMax (treeNodePtr hdTree, int &count) {
    if (hdTree != NULL) {
        if ((hdTree->left == NULL) && (hdTree->right != NULL))
            || (hdTree->left != NULL) && (hdTree->right == NULL)
            count++;
        numMax (hdTree->left, count);
        numMax (hdTree->right, count);
    }
}
```

d)

```

void averSalary (treeNode* htree, int count, int salary) {
    if (htree != NULL) {
        count++;
        salary = (htree->salary + salary * (count-1)) / count;
        averSalary (htree->left, count, salary);
        averSalary (htree->right, count, salary);
    }
}

```

e)

```

void peopleUnderID (treeNode* htree, int count, int flag,
                    root id id) {
    if (htree != NULL) {
        if (flag) count++;
        if (htree->id == id)
            flag = true;
        if (htree->left != NULL)
        peopleUnderID (htree->left, count, flag, id);
        " " " "
        peopleUnderID (htree->right, count, flag, id);
    }
}

```