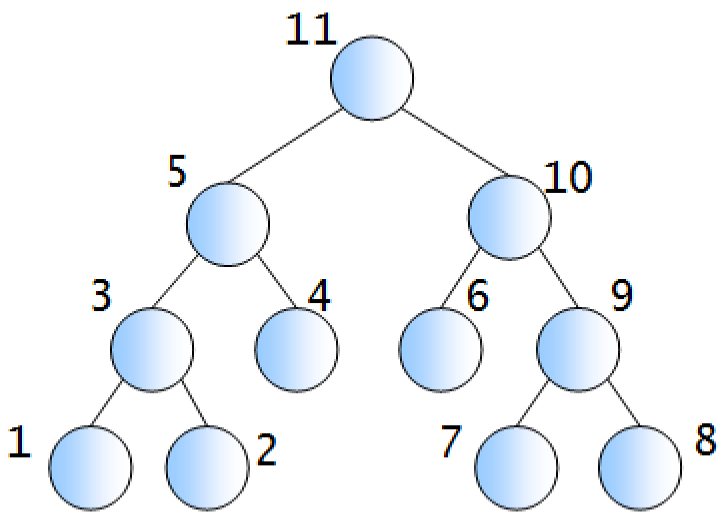
1. (10 points: Hand-written homework)
   Please write up the call order of postorder traversal in the following binary tree

```
postorder(x)
if x≠NULL
           postorder(LEFT(x));
           postorder(RIGHT(x));
           print DATA(x);
```



?

2. (25 points: Programming)
Implement the function that finds a successor of node in the inorder traversal.

- Refer to the following pseudo code.
- Use the given data structure and simple binary tree example in p3.

```
Tree_successor(x)
{
          if x->right != NULL //x’s right subtree is not null
                    return the leftmost node of right subtree

          //x’s right subtree is null
          y = x->parent
          while (y != NULL and x == y->right) {
                    x = y;
                    y = y->parent;
          }
          return y;
}
```

```c
typedef struct TreeNode {
            int data;
            struct TreeNode *left, *right, *parent;
} TreeNode;

//          G
//    C        F
//  A   B    D   E
TreeNode n1 = { 'A', NULL, NULL, NULL };
TreeNode n2 = { 'B', NULL, NULL, NULL };
TreeNode n3 = { 'C', &n1, &n2, NULL };
TreeNode n4 = { 'D', NULL, NULL, NULL };
TreeNode n5 = { 'E', NULL, NULL, NULL };
TreeNode n6 = { 'F', &n4, &n5, NULL };
TreeNode n7 = { 'G', &n3, &n6, NULL };
TreeNode *exp = &n7;

TreeNode *tree_successor(TreeNode *p) {
            .....
}
void main()
{
            TreeNode *q = exp;
            n1.parent = &n3;
            n2.parent = &n3;
            n3.parent = &n7;
            n4.parent = &n6;
            n5.parent = &n6;
            n6.parent = &n7;
            n7.parent = NULL;

            while (q->left) q = q->left;         // Go to the leftmost node
            do
            {
                        printf("%c\n", q->data); // Output data
                         // Call the successor
                        q = tree_successor(q);
            } while (q);    // If not null
}
```

```
Output:
A
C
B
G
D
F
E
```

3. (25 points: Programming)
   Implement the function that finds a predecessor of node in the inorder traversal.
   - Write up the pseudo code by referring to the pseudo code of ‘`Tree_successor(x)`’.
   - Use the given data structure and simple binary tree example in p5.

```c
typedef struct TreeNode {
            int data;
            struct TreeNode *left, *right, *parent;
} TreeNode;

//          G
//     C        F
//   A   B    D   E
TreeNode n1 = { 'A', NULL, NULL, NULL };
TreeNode n2 = { 'B', NULL, NULL, NULL };
TreeNode n3 = { 'C', &n1, &n2, NULL };
TreeNode n4 = { 'D', NULL, NULL, NULL };
TreeNode n5 = { 'E', NULL, NULL, NULL };
TreeNode n6 = { 'F', &n4, &n5, NULL };
TreeNode n7 = { 'G', &n3, &n6, NULL };
TreeNode *exp = &n7;

TreeNode *tree_predecessor(TreeNode *p) {
            .....
}
void main()
{
            TreeNode *q = exp;
            n1.parent = &n3;
            n2.parent = &n3;
            n3.parent = &n7;
            n4.parent = &n6;
            n5.parent = &n6;
            n6.parent = &n7;
            n7.parent = NULL;
            // Go to the rightmost node
            ...
            do
            {
                        printf("%c\n", q->data); // Output data
                        // Call the predecessor
                        q = tree_predecessor(q);
            } while (q);    // If not null
}
```

```
Output:
E
F
D
G
B
C
A
```

4. (40 points: Programming)

In the code provided in the page 60 and 61, the case 3 was implemented using the successor at the right subtree. Revise this part by using the predecessor at the left subtree. Verify your code by running the following three examples. Namely, the revised code should produces the same results of the original code. Refer to the attached original code ('bst_insertion_deletion.cpp').

```
typedef struct TreeNode {
int key;
struct TreeNode *left, *right;
} TreeNode;

//                  35
//          18                  68
//   7             26                    99
//3     12      22          30
//    10            24

TreeNode n12 = { 24, NULL, NULL };
TreeNode n11 = { 10, NULL, NULL };
TreeNode n10 = { 30, NULL, NULL };
TreeNode n9 = { 22, NULL, &n12 };
TreeNode n8 = { 12, &n11, NULL };
TreeNode n7 = { 3,  NULL, NULL };
TreeNode n6 = { 99, NULL, NULL };
TreeNode n5 = { 26, &n9, &n10 };
TreeNode n4 = { 7,  &n7, &n8 };
TreeNode n3 = { 68, NULL, &n6 };
TreeNode n2 = { 18, &n4, &n5 };
TreeNode n1 = { 35, &n2, &n3 };
```

<u>Three examples</u>

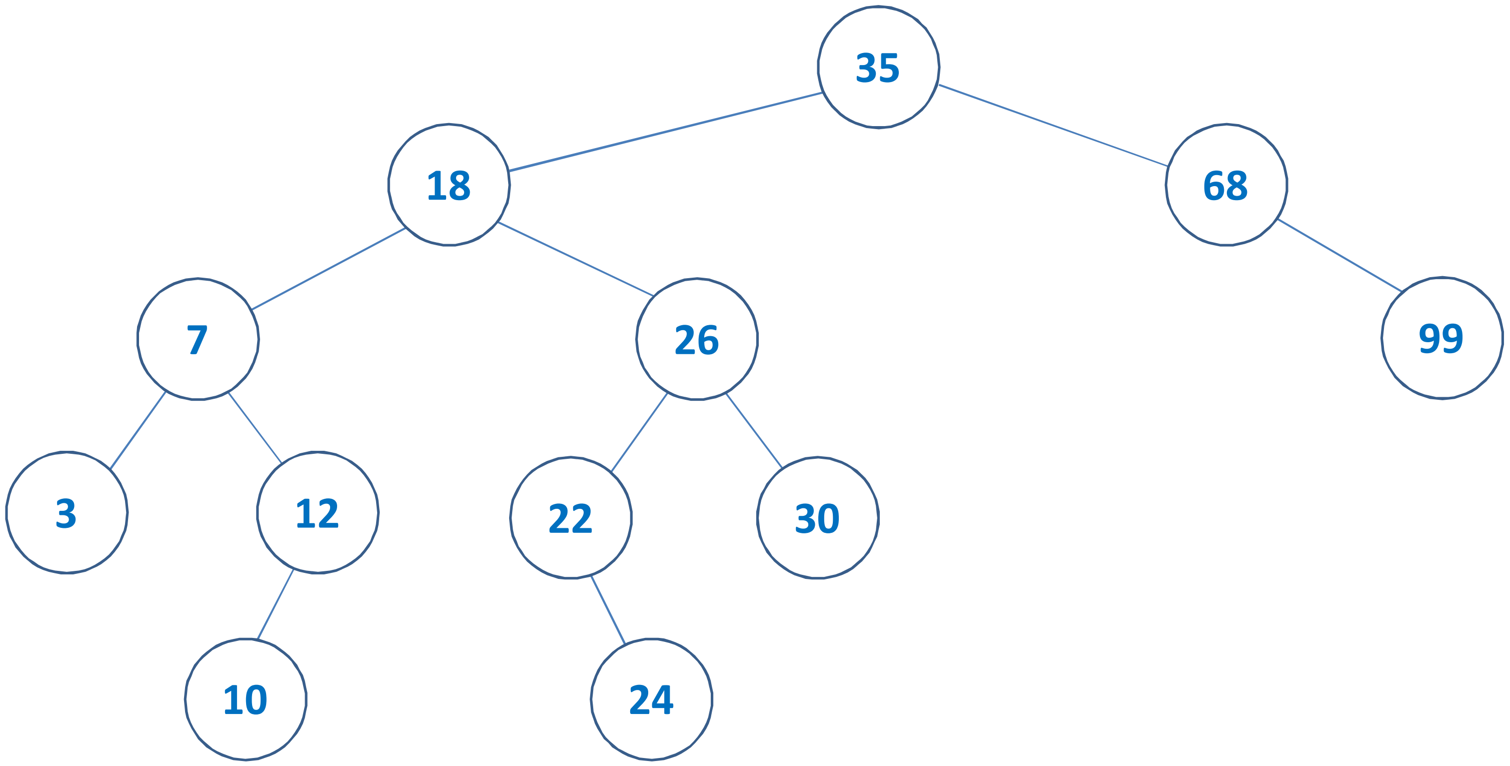
1) key = 18
2) key = 35
3) key = 7

```
void main()
{
            TreeNode *root = &n1;

            printf("Binary tree\n");
            inorder(root);
            printf("\n\n");

            delete_node(&root, key);

            printf("Binary tree\n");
            inorder(root);
            printf("\n\n");
}
```


35
18
68
7
26
99
3
12
22
30
10
24