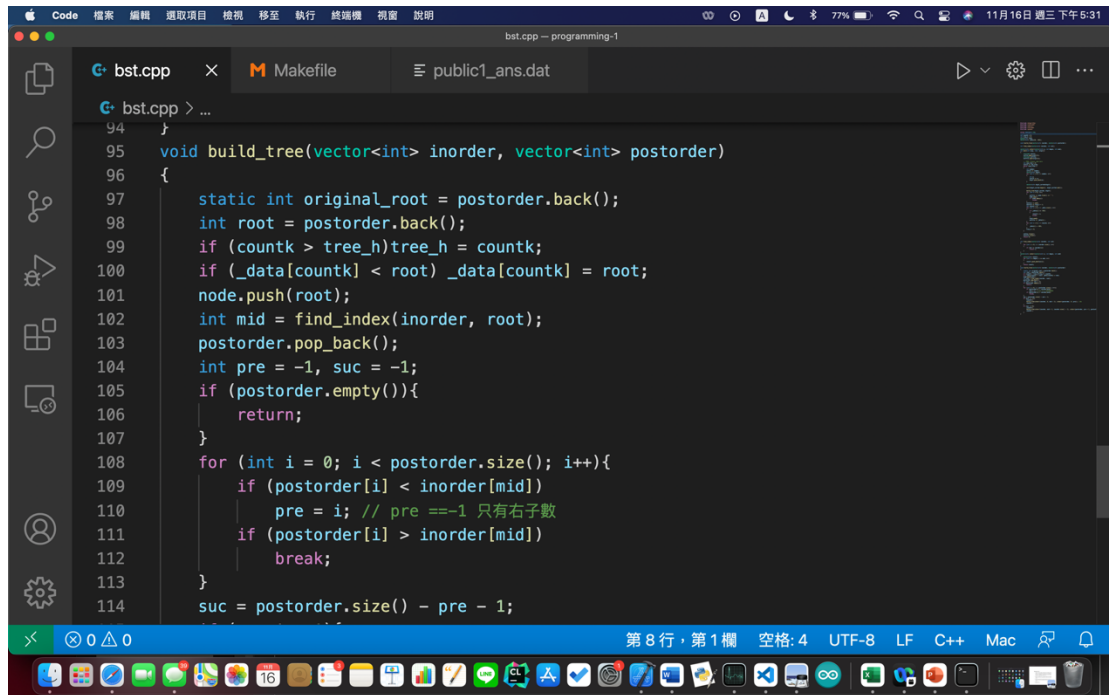


1.

```
build_tree(inorder,postorder);
```

利用 build 去做遞迴，參數是 inorder 跟 postorder

將 postorder 的最後一個數字，也就是樹的 root 存到 queue 裡面之後 pop 掉，然後找出在 postorder 裡 root 的左右子樹，及對應到的 inorder，分別切割，再傳進函式裡就可。



```
bst.cpp
94 }
95 void build_tree(vector<int> inorder, vector<int> postorder)
96 {
97     static int original_root = postorder.back();
98     int root = postorder.back();
99     if (countk > tree_h) tree_h = countk;
100     if (_data[countk] < root) _data[countk] = root;
101     node.push(root);
102     int mid = find_index(inorder, root);
103     postorder.pop_back();
104     int pre = -1, suc = -1;
105     if (postorder.empty()){
106         return;
107     }
108     for (int i = 0; i < postorder.size(); i++){
109         if (postorder[i] < inorder[mid]){
110             pre = i; // pre == -1 只有右子數
111             if (postorder[i] > inorder[mid])
112                 break;
113         }
114     }
115     suc = postorder.size() - pre - 1;
```

切割函式

```
vector<int> cutarr(vector<int> a, int begin, int end)
{
    vector<int> result;
    for (int i = begin; i <= end; i++)
    {
        result.push_back(a[i]);
    }
    return result;
}
```

2.preorder

把 queue 裡的資料一一印出來就是 preorder 的順序

```
for (int i = 0;; i++)
{
    outfile << node.front() << " ";
    node.pop();
    if (node.empty())
        break;
}
```

3.tree_height / maximum

Tree_h:

利用 countk 去計算現在是在第幾層，在遞迴的每次開始記錄 countk，若比前一個大就把 countk 更新成最大高度。

Maximum:

開一個一維陣列，index 值就是 countk 也就是層數-1，每次開始遞迴時，利用 countk 知道現在層數，也知道 node 值就可比較，同樣層數的另一個 node 值比前一個大就換，如此一來可得每一層的 max 值

```
if (pre != -1){ //判斷有沒有左子樹
    countk++;
    build_tree(cutarr(inorder, 0, mid - 1), cutarr(postorder, 0, pre));
    countk--;
}
if (suc != 0){ //判斷有沒有右子樹
    countk++;
    build_tree(cutarr(inorder, mid + 1, inorder.size() - 1), cutarr(postorder, pre + 1, suc));
    countk--;
}
```