**★★☆☆☆**

**題組：基礎48題**

**題號：Q12347 - Binary Search Tree**

**整理者：陳紫淇**

**學號：ADT105142**

**使用語言:C++**

**解題日期：2018年8月29日**



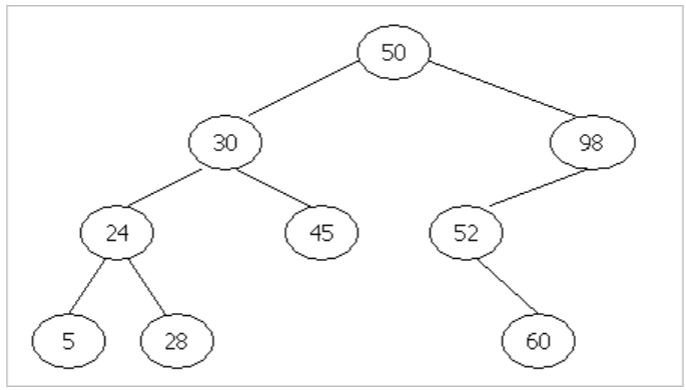
**題目:**

A binary search tree is a binary tree that satisfies the following properties:

• The left subtree of a node contains only nodes with keys less than the node’s key.

• The right subtree of a node contains only nodes with keys greater than the node’s key.

• Both the left and right subtrees must also be binary search trees.



Pre-order traversal (Root-Left-Right) prints out the nodes key by visiting the root node then traversing the left subtree and then traversing the right subtree. Post-order traversal (Left Right-Root) prints out the left subtree first and then right subtree and finally the root node. For example, the results of pre-order traversal and post-order traversal of the binary tree shown in Figure 1 are as follows:

**Pre-order:** 50 30 24 5 28 45 98 52 60

**Post-order:** 5 28 24 45 30 60 52 98 50

Given the pre-order traversal of a binary search tree, you task is to find the post-order traversal of this tree.

**Input**

The keys of all nodes of the input binary search tree are given according to pre-order traversal. Each node has a key value which is a positive integer less than 106 . All values are given in separate lines (one integer per line). You can assume that a binary search tree does not contain more than 10,000 nodes and there are no duplicate nodes.

**Output**

The output contains the result of post-order traversal of the input binary tree. Print out one key per line.

**Sample Input**

50

30

24

5

28

45

98

52

60**Sample Output**

5

28

24

45

30

60

52

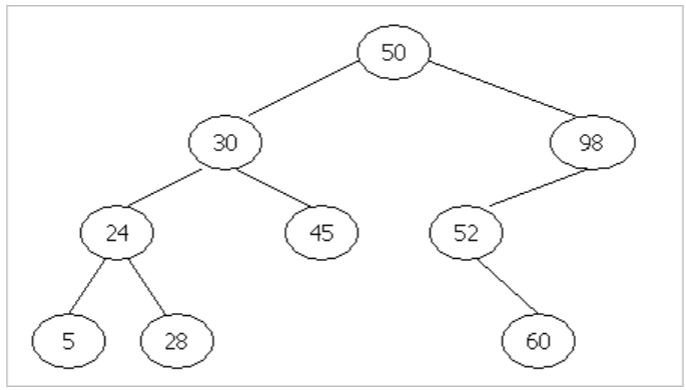
98

50

**問題描述：**

二元搜尋樹是一種二元樹且滿足以下性質：

* 某節點左子樹的節點的值都小於該節點的值。
* 某節點右子樹的節點的值都大於該節點的值。
* 左子樹和右子樹也必須為二元搜尋樹。

右圖為二元搜尋樹的範例：

要拜訪一棵二元樹所有的節點有幾種不同的方式。前序追蹤（Pre-order traversal）會列印根節點（root）的值，然後拜訪並列印左子樹，最後拜訪並列印右子樹。而後序追總（Post-order traversal）則先拜訪並列印左子樹，再拜訪並列印右子樹，最後才列印根節點的值。例如上圖兩種方式列印節點值的順序如下：

**Pre-order:** 50 30 24 5 28 45 98 52 60

**Post-order:**     5 28 24 45 30 60 52 98 50

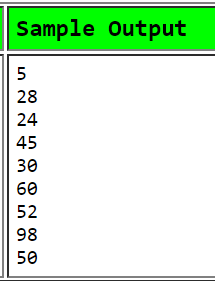
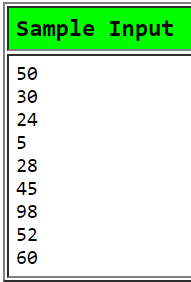
給你一棵二元樹前序追蹤的結果，請你輸出其後序追蹤的結果。

**Input**

輸入為一棵二元樹前序追蹤的結果，每個節點一列。

**Output**

輸出此二元樹後序追蹤的結果，每個節點一列。



**解法:**

**真的建一棵樹。**

**解法範例：**

1. 建立一結構(Tree)，裡面有val(自己的值)，左子樹指標和右子樹指標
2. 建立insert函式，傳入一個根節點和要insert的值(new\_val)

新增一個Tree，設定其val為new\_val

//若此Tree為第一個節點，直接回傳新的Tree

依照new\_val大小來尋找一個適合他的空葉子(依照二元搜尋樹的規則)

找到之後將此新節點加入，然後回傳根節點

1. 樹建好之後就用Post\_Order將其印出

討論：

1. 或許insert時不用回傳根節點，作法是將根節點設定成全域變數，在函式裡將其更新

|  |
| --- |
| #include <cstdio>  #include <iostream>  using namespace std;  struct Tree  {  int val = 0;  Tree \*left = nullptr;  Tree \*right = nullptr;  };  Tree \*Insert(Tree \*old\_tree, int new\_val)  {  Tree \*New\_node = new Tree;  New\_node->val = new\_val; //New\_node是新的節點  if (old\_tree == nullptr) //如果是第一個節點(根)，回傳自己  return New\_node;  Tree \*parent = old\_tree; //parent用來記住父母  Tree \*next = old\_tree; //next記住空的葉子  while (next != nullptr) //尋找空的葉子  {  parent = next;  if (new\_val > next->val)  next = next->right;  else  next = next->left;  }  //找到空的葉子之後將新節點加入  if (new\_val > parent->val)  parent->right = New\_node;  else  parent->left = New\_node;  return old\_tree;  }  void Post\_Order(Tree \*node)  {  if (node == nullptr) return;  Post\_Order(node->left);  Post\_Order(node->right);  printf("%d\n", node->val);  }  int main()  {  Tree \*myTree = nullptr;  int input;  while (cin>>input)  {  myTree = Insert(myTree,input);  }  Post\_Order(myTree);  } |