二元搜尋樹刪除

資訊四丙

D0683497

柯利韋

目錄

[一、 資料結構 2](#_Toc59604391)

[1. 樹 2](#_Toc59604392)

[二、 演算法說明 2](#_Toc59604393)

[1. 輸入要操作的選項 2](#_Toc59604394)

[2. 如果是插入的話，若樹為空直接建立新節點，若樹不為空則接著判斷插入數字的大小，若一樣則印出此數字已經存在返回，若是比當前節點小則往右找，若是比當前節點大則往左找，直到當前節點為空時，建立心結點返回 2](#_Toc59604395)

[3. 如果是刪除的話，若樹為空則印出樹為空，若樹不為空則接著找要刪除的節點，若是比當前節點小則往右找，若是比當前節點大則往左找，直到找到當前節點為空表示無此節點，印出找不到節點 2](#_Toc59604396)

[三、 程式原始碼 3](#_Toc59604397)

[四、 執行結果 7](#_Toc59604398)

[五、 心得 8](#_Toc59604399)

1. 資料結構
   1. 樹

typedef struct Tree

{

int data;

Tree \*left;

Tree \*right;

} Tree;

1. 演算法說明
   1. 輸入要操作的選項
   2. 如果是插入的話，若樹為空直接建立新節點，若樹不為空則接著判斷插入數字的大小，若一樣則印出此數字已經存在返回，若是比當前節點小則往右找，若是比當前節點大則往左找，直到當前節點為空時，建立心結點返回
   3. 如果是刪除的話，若樹為空則印出樹為空，若樹不為空則接著找要刪除的節點，若是比當前節點小則往右找，若是比當前節點大則往左找，直到找到當前節點為空表示無此節點，印出找不到節點
2. 程式原始碼

#include <stdio.h>

#include <stdlib.h>

typedef struct Tree

{

int data;

Tree \*left;

Tree \*right;

} Tree;

// 新增節點

Tree\* NewNode(int input)

{

Tree\* temp = (Tree\*)malloc(sizeof(Tree));

temp->data = input;

temp->left = temp->right = NULL;

return temp;

}

// 插入

Tree\* Insert(Tree\* tree, int input)

{

if (tree == NULL)

{

return NewNode(input);

}

if (input == tree->data)

{

printf("%d already exist.\n", input);

}

else if (input < tree->data)

{

tree->left = Insert(tree->left, input);

}

else

{

tree->right = Insert(tree->right, input);

}

return tree;

}

// 輸出

void Preorder(Tree\* root)

{

if (root != NULL)

{

printf("%d ",root->data);

Preorder(root->left);

Preorder(root->right);

}

}

Tree\* MinValueNode(Tree\* node)

{

Tree\* current = node;

while (current && current->left != NULL)

{

current = current->left;

}

return current;

}

// 刪除

Tree\* Delete(Tree\* root, int input)

{

if (root == NULL)

{

printf("%d does not exist in the tree.\n", input);

return root;

}

if (input < root->data)

{

root->left = Delete(root->left, input);

}

else if (input > root->data)

{

root->right = Delete(root->right, input);

}

else if (root->data == input)

{

if (root->left == NULL)

{

Tree\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

Tree\* temp = root->left;

free(root);

return temp;

}

Tree\* temp = MinValueNode(root->left);

root->data = temp->data;

root->left = Delete(root->left, temp->data);

}

return root;

}

int main()

{

Tree\* root = NULL;

int option, input;

while(1)

{

printf("[1]Insert [2]Delete [3]Print [0]Exit: ");

scanf("%d", &option);

if (option == 0)

{

break;

}

else

{

switch (option)

{

case 1: // 插入

printf("Insert a number: ");

scanf("%d", &input);

root = Insert(root, input);

break;

case 2: // 刪除

if (root == NULL)

{

printf("Tree is Empty.\n");

}

else

{

printf("Delete a number: ");

scanf("%d", &input);

root = Delete(root, input);

}

break;

case 3: // 輸出

printf("Preorder:");

Preorder(root);

printf("\n");

break;

default:

break;

}

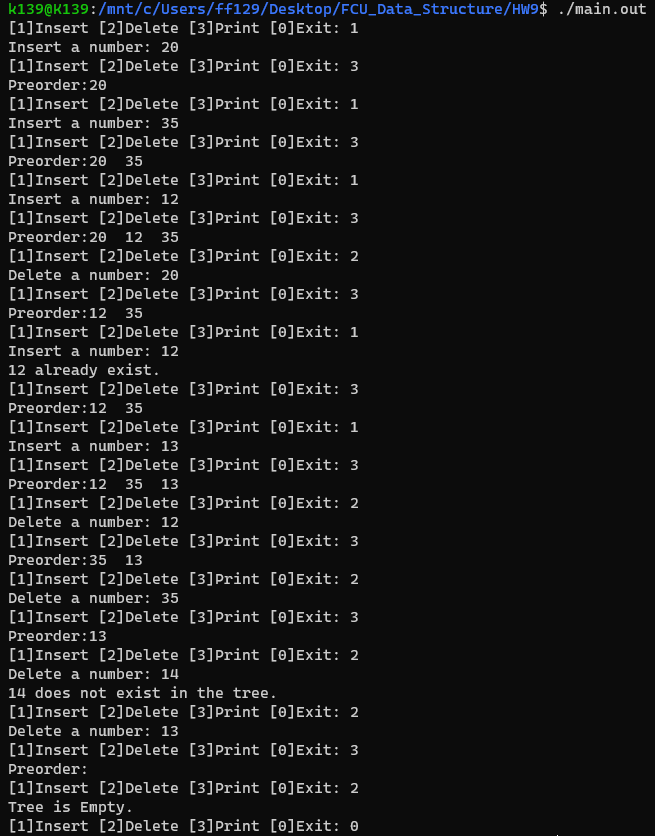
}

}

return 0;

}

1. 執行結果



1. 心得

實習課已經練習過了，所以基本上沒有什麼問題。