# HW9

# 二元樹的節點刪除程式

班級:資訊三丙
姓名:楊燿寧
學號:D0745765

# 題目定義

## 1.資料結構

本次使用的資料結構為使用 struct 與結構指標所建成的二元樹資料結構。不過，跟以往不同的是，增加了「back」的結構指標，用途在於指向回自己的父節點。

## 2.演算法

此程式進行二元樹上節點的刪除，但刪除的方法卻會因為刪除節點的特性不同，而有不同的處理理方式。以下我們將分為三種不同的狀況下，而產生的刪除方式：

1. 刪除節點為末端，左右皆無小孩

   在此情況下，我們只要將父節點上連結欲刪除節點的指標清除即可。因此唯一的工作就是判斷是否為根節點 （root），若是根節點則直接將儲存根節點空間地址的變數（在此程式為 root）的值變為 NULL，並釋放空間。若根節點，則需判斷欲刪除節點為父節點的有小孩或是左小孩，我們使用 Key 值來進行判斷，再將父節點上儲存地址的變數設為 NULL，並釋放空間。

2. 健全節點，左右皆有小孩

   先將 temp 指向欲刪除位置(del)之左子節點

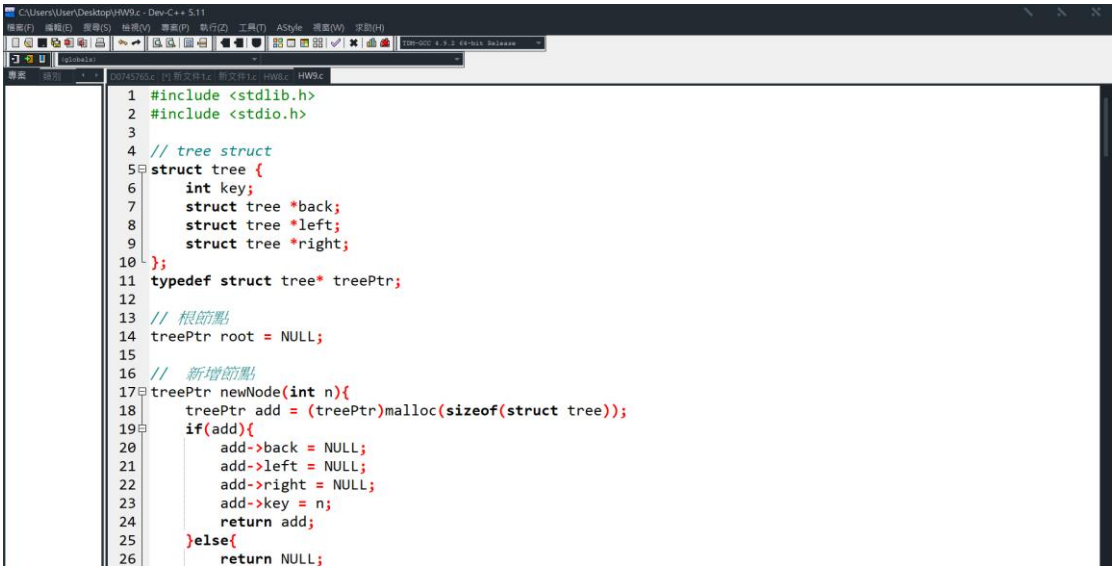
   此狀況又分為兩種狀況，其中一種為**若 temp 無右子節點**，我們將 temp 指標變數存入欲刪除節點的左小孩，之後，我們會將 temp 的 key 覆蓋給 del，並且將 temp 從樹中移除。

   另一種為**若 temp 有右子節點**，因此，我們必須將 temp 移動到最右邊的小孩（因為必須找到左邊最大的節來替換），找到後如同上一個狀況，將 temp 中的 Key 覆蓋給 del，並且將 temp 從樹中移除。

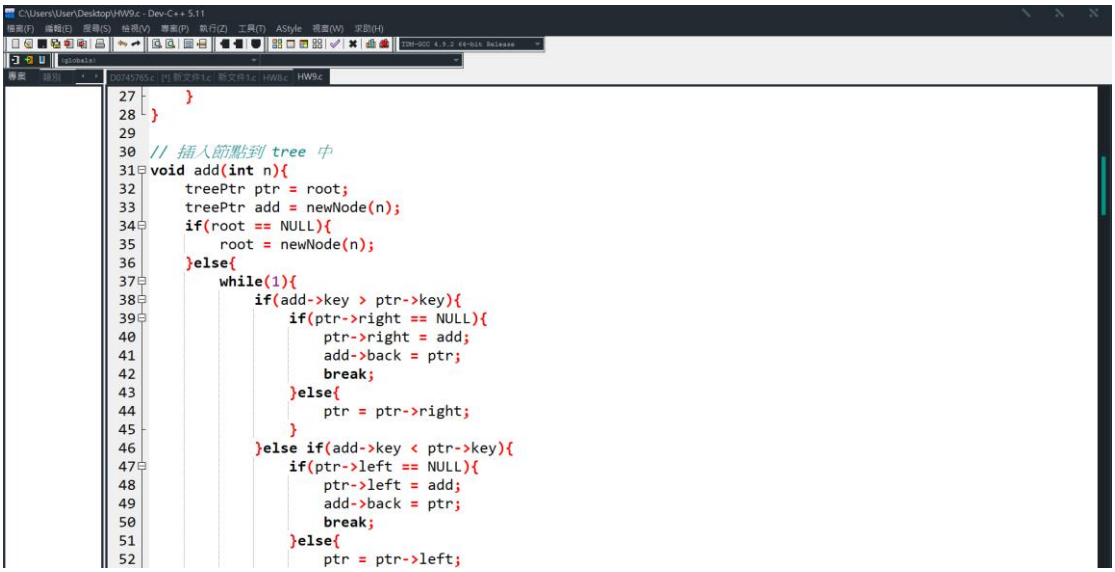
3. 刪除節點只有一邊小孩，另外一邊為空

   此時我們必須判斷是否為 root，若為 root 則直接將 root 指標變數往右小孩（在此假設只擁有右子樹）。若非 root，則我們必須先判斷欲刪除節點為父節點的左或右小孩。並將其右小孩的節點給往父節點，並將欲刪除節點空間清除釋放。

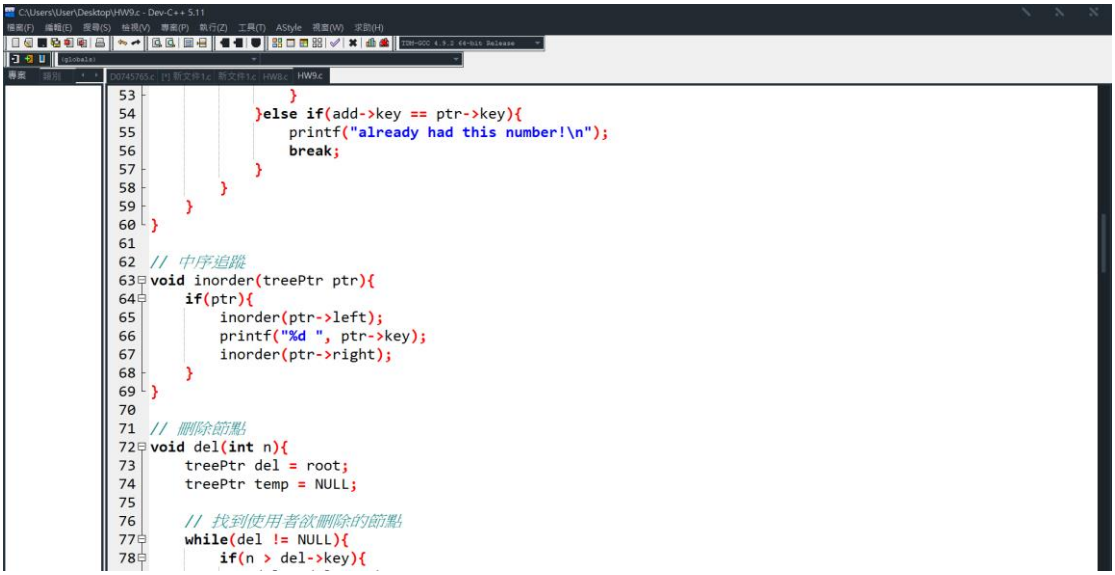
# 原始程式碼



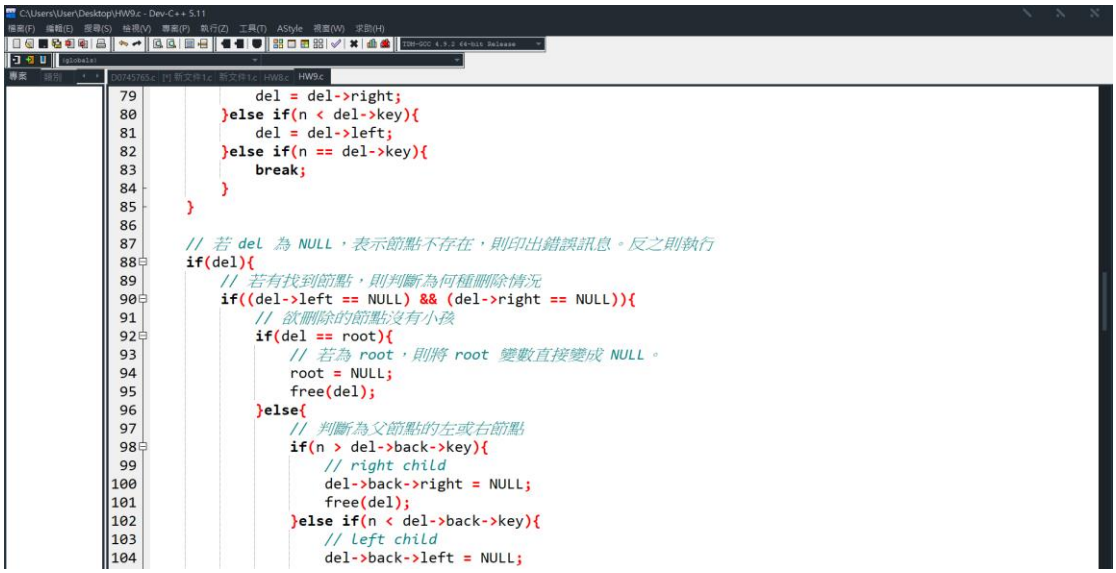
```c
#include <stdlib.h>
#include <stdio.h>

// tree struct
struct tree {
    int key;
    struct tree *back;
    struct tree *left;
    struct tree *right;
};
typedef struct tree* treePtr;

// 根節點
treePtr root = NULL;

//  新增節點
treePtr newNode(int n){
    treePtr add = (treePtr)malloc(sizeof(struct tree));
    if(add){
        add->back = NULL;
        add->left = NULL;
        add->right = NULL;
        add->key = n;
        return add;
    }else{
        return NULL;
```



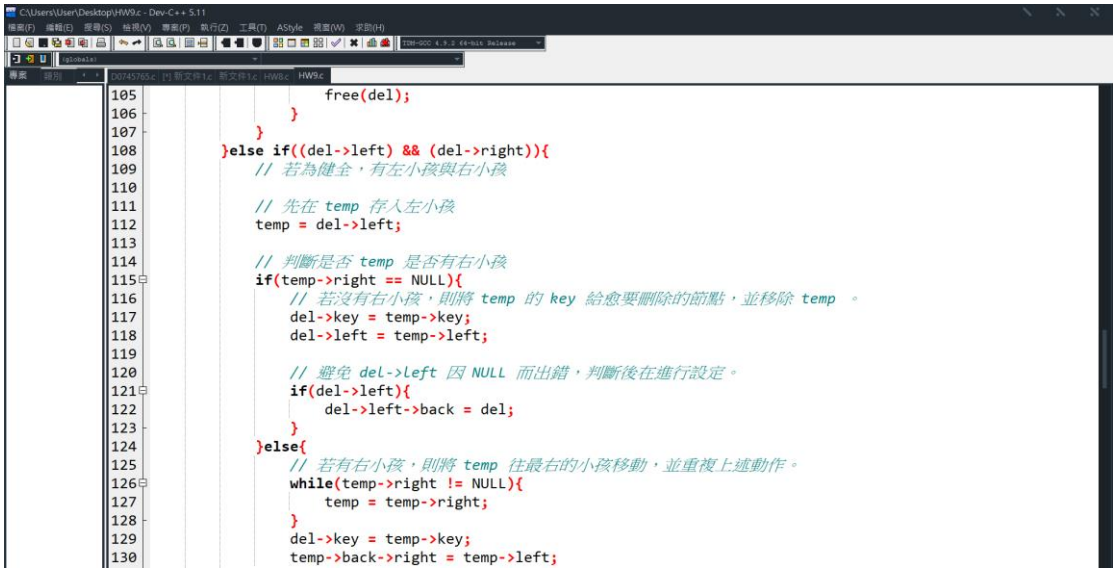
```c
    }
}

// 插入節點到 tree 中
void add(int n){
    treePtr ptr = root;
    treePtr add = newNode(n);
    if(root == NULL){
        root = newNode(n);
    }else{
        while(1){
            if(add->key > ptr->key){
                if(ptr->right == NULL){
                    ptr->right = add;
                    add->back = ptr;
                    break;
                }else{
                    ptr = ptr->right;
                }
            }else if(add->key < ptr->key){
                if(ptr->left == NULL){
                    ptr->left = add;
                    add->back = ptr;
                    break;
                }else{
                    ptr = ptr->left;
```



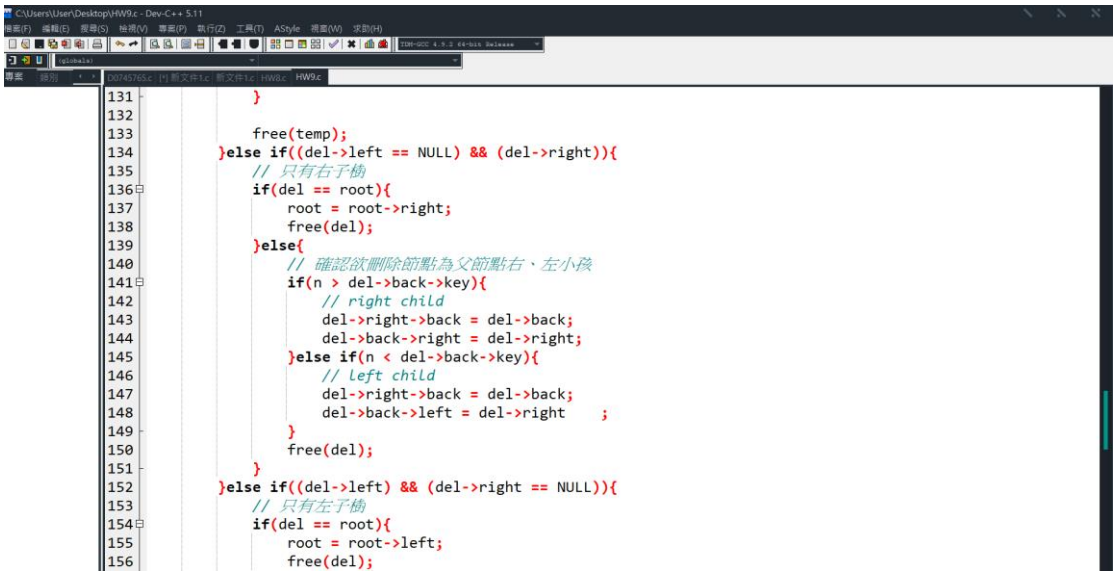
```c
                }
            }else if(add->key == ptr->key){
                printf("already had this number!\n");
                break;
            }
        }
    }
}

// 中序追蹤
void inorder(treePtr ptr){
    if(ptr){
        inorder(ptr->left);
        printf("%d ", ptr->key);
        inorder(ptr->right);
    }
}

// 刪除節點
void del(int n){
    treePtr del = root;
    treePtr temp = NULL;

    // 找到使用者欲刪除的節點
    while(del != NULL){
        if(n > del->key){
```



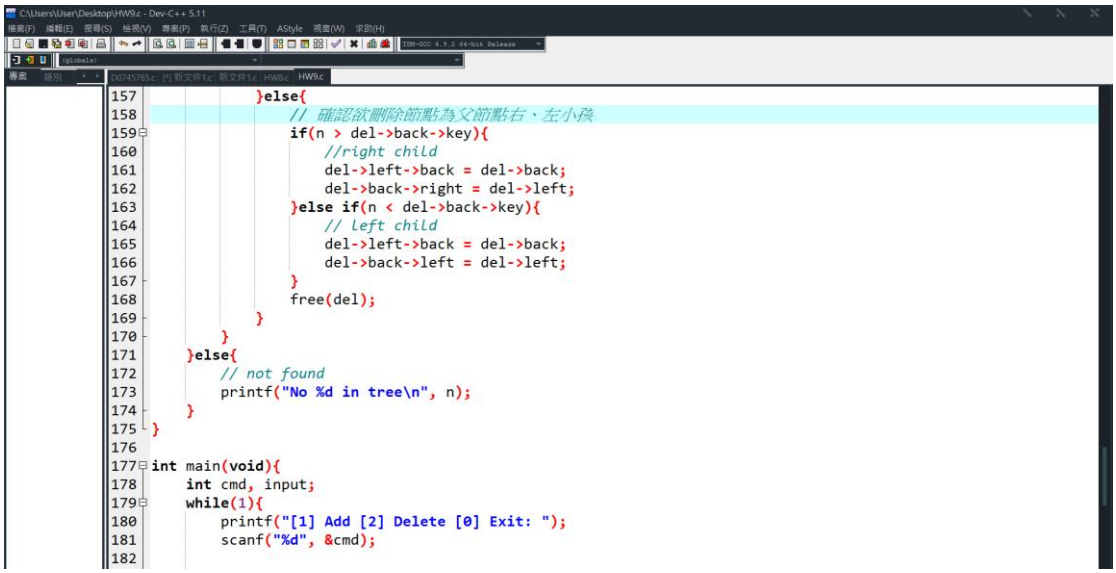
```
79              del = del->right;
80          }else if(n < del->key){
81              del = del->left;
82          }else if(n == del->key){
83              break;
84          }
85      }
86
87      // 若 del 為 NULL，表示節點不存在，則印出錯誤訊息。反之則執行
88      if(del){
89          // 若有找到節點，則判斷為何種刪除情況
90          if((del->left == NULL) && (del->right == NULL)){
91              // 欲刪除的節點沒有小孩
92              if(del == root){
93                  // 若為 root，則將 root 變數直接變成 NULL。
94                  root = NULL;
95                  free(del);
96              }else{
97                  // 判斷為父節點的左或右節點
98                  if(n > del->back->key){
99                      // right child
100                     del->back->right = NULL;
101                     free(del);
102                 }else if(n < del->back->key){
103                     // left child
104                     del->back->left = NULL;
```



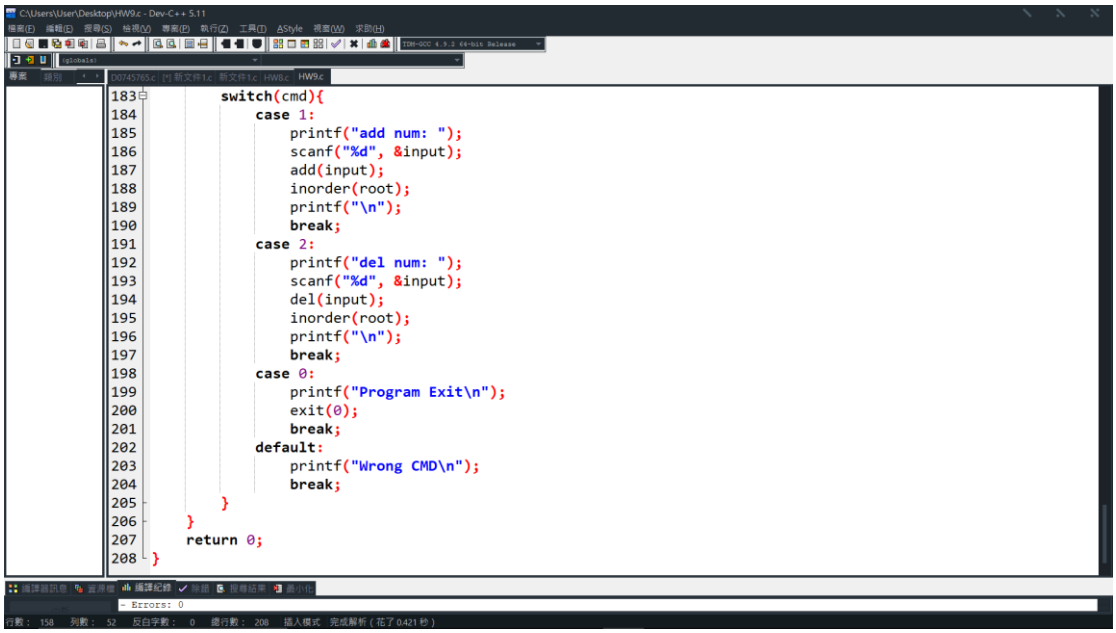
```
105                 free(del);
106             }
107         }
108     }else if((del->left) && (del->right)){
109         // 若為健全，有左小孩與右小孩
110
111         // 先在 temp 存入左小孩
112         temp = del->left;
113
114         // 判斷是否 temp 是否有右小孩
115         if(temp->right == NULL){
116             // 若沒有右小孩，則將 temp 的 key 給愈要刪除的節點，並移除 temp 。
117             del->key = temp->key;
118             del->left = temp->left;
119
120             // 避免 del->left 因 NULL 而出錯，判斷後在進行設定。
121             if(del->left){
122                 del->left->back = del;
123             }
124         }else{
125             // 若有右小孩，則將 temp 往最右的小孩移動，並重複上述動作。
126             while(temp->right != NULL){
127                 temp = temp->right;
128             }
129             del->key = temp->key;
130             temp->back->right = temp->left;
```



```
131         }
132
133         free(temp);
134     }else if((del->left == NULL) && (del->right)){
135         // 只有右子樹
136         if(del == root){
137             root = root->right;
138             free(del);
139         }else{
140             // 確認欲刪除節點為父節點右、左小孩
141             if(n > del->back->key){
142                 // right child
143                 del->right->back = del->back;
144                 del->back->right = del->right;
145             }else if(n < del->back->key){
146                 // left child
147                 del->right->back = del->back;
148                 del->back->left = del->right    ;
149             }
150             free(del);
151         }
152     }else if((del->left) && (del->right == NULL)){
153         // 只有左子樹
154         if(del == root){
155             root = root->left;
156             free(del);
```

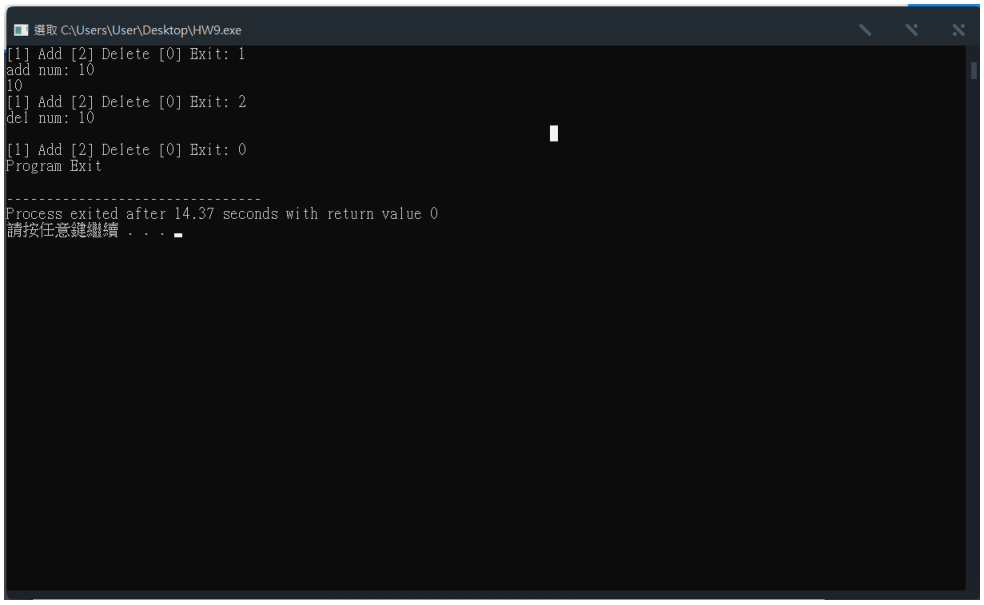


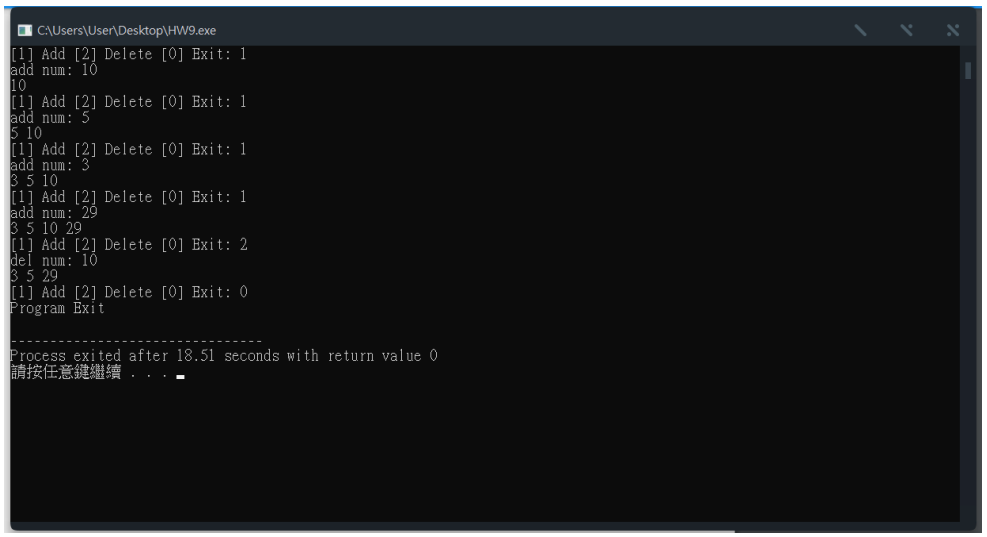
```
157                 }else{
158                     // 確認欲刪除節點為父節點右、左小孩
159                     if(n > del->back->key){
160                         //right child
161                         del->left->back = del->back;
162                         del->back->right = del->left;
163                     }else if(n < del->back->key){
164                         // left child
165                         del->left->back = del->back;
166                         del->back->left = del->left;
167                     }
168                     free(del);
169                 }
170             }
171         }else{
172             // not found
173             printf("No %d in tree\n", n);
174         }
175     }
176
177 int main(void){
178     int cmd, input;
179     while(1){
180         printf("[1] Add [2] Delete [0] Exit: ");
181         scanf("%d", &cmd);
182
```

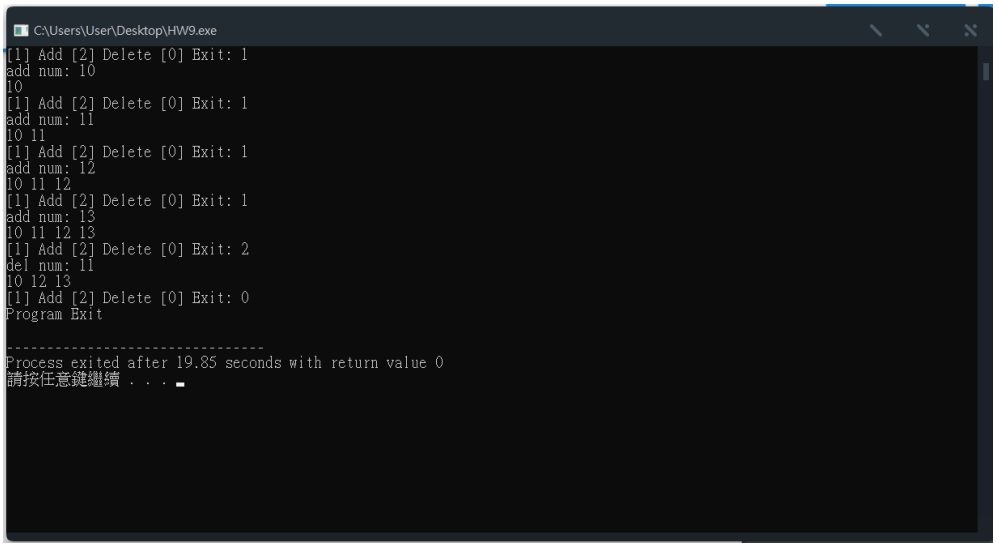


```
183         switch(cmd){
184             case 1:
185                 printf("add num: ");
186                 scanf("%d", &input);
187                 add(input);
188                 inorder(root);
189                 printf("\n");
190                 break;
191             case 2:
192                 printf("del num: ");
193                 scanf("%d", &input);
194                 del(input);
195                 inorder(root);
196                 printf("\n");
197                 break;
198             case 0:
199                 printf("Program Exit\n");
200                 exit(0);
201                 break;
202             default:
203                 printf("Wrong CMD\n");
204                 break;
205         }
206     }
207     return 0;
208 }
```

# 執行結果


選取 C:\Users\User\Desktop\HW9.exe
[1] Add [2] Delete [0] Exit: 1
add num: 10
10
[1] Add [2] Delete [0] Exit: 2
del num: 10
[1] Add [2] Delete [0] Exit: 0
Program Exit
Process exited after 14.37 seconds with return value 0
請按任意鍵繼續 . . .


C:\Users\User\Desktop\HW9.exe
[1] Add [2] Delete [0] Exit: 1
add num: 10
10
[1] Add [2] Delete [0] Exit: 1
add num: 5
5 10
[1] Add [2] Delete [0] Exit: 1
add num: 3
3 5 10
[1] Add [2] Delete [0] Exit: 1
add num: 29
3 5 10 29
[1] Add [2] Delete [0] Exit: 2
del num: 10
3 5 29
[1] Add [2] Delete [0] Exit: 0
Program Exit
Process exited after 18.51 seconds with return value 0
請按任意鍵繼續 . . .


C:\Users\User\Desktop\HW9.exe
[1] Add [2] Delete [0] Exit: 1
add num: 10
10
[1] Add [2] Delete [0] Exit: 1
add num: 11
10 11
[1] Add [2] Delete [0] Exit: 1
add num: 12
10 11 12
[1] Add [2] Delete [0] Exit: 1
add num: 13
10 11 12 13
[1] Add [2] Delete [0] Exit: 2
del num: 11
10 12 13
[1] Add [2] Delete [0] Exit: 0
Program Exit
Process exited after 19.85 seconds with return value 0
請按任意鍵繼續 . . .

# 時間複雜度

假設 n 為樹的節點個數 在不同的情況底下進行刪除，可能會有不同的時間複雜度，但可以知道，若欲刪除節點的全部子結點都在左樹，或者右樹，將會有最大的時間複雜度。因此，我們可以推論，del 副程式的時間複雜度為 O(n)。

因此，此程式的時間複雜度為： O(n)

# 心得

此次作業讓我更了解二元樹在不同的情況下，所表現出來的樣貌，並且因為不同的樣貌，而會讓一些變更刪除的動作變得有區別性。透過這次的作業，讓我可以更深入的瞭解了樹，也更了解在二元樹上操作的一些規則。