







## 函式清單及說明：

	lib	存放library，內有libBST.a	2022/11/30 下午 06:19	檔案資料夾	
	obj	存放object file，內有BST.o	2022/11/30 下午 06:05	檔案資料夾	
	BST	C原始檔碼	2022/11/30 上午 12:19	C 檔案	5 KB
	BST	head 檔	2022/11/29 下午 11:21	H 檔案	1 KB
	main	功能測試檔	2022/11/30 下午 05:57	C 檔案	4 KB
	test	執行檔案	2022/11/30 下午 05:57	應用程式	133 KB

## library BST 說明：

在這個作業裡我自己設計出的 library，用於實作 HW2 的 BST，裡面共有 8 個 function，以下是對於設計出的 library 內 function 的介紹：

### 第一個 function 是 [insertNode](#)：

```

btreeNode_t *insertNode(void *element, btreeNode_t *root, int (*compare)(void *elementA, void *elementB))
{
    if(root == NULL)
        root = (btreeNode_t *)element;
    else
    {
        if(compare(element, root) == -1) // element < root, left
            root->left = insertNode(element, root->left, compare);
        else if(compare(element, root) == 1) // element > root, right
            root->right = insertNode(element, root->right, compare);
        else if(compare(element, root) == 0) {} // element == root, do nothing
    }
    return root;
}

```

他用於插入一個節點至樹當中，並回傳插入節點後的樹給使用者。

使用者呼叫這個 function 的方式為：

**insertNode(要插入的節點，樹，使用者自用 function)。**

使用者要使用這個 function 時還需另外設計自己的 function 以完成他想做的工作。

### 設計這個 function 的想法：

當傳入一棵樹，首先要先檢查這個樹是否為空，如果他為空的話，就直接將樹根指向傳入的節點，最後回傳這個樹；如果他不為空的話，需要比較節點與樹根的大小，以決定要在樹的右邊還是左邊，如果節點跟樹根的大小一樣，就不需做任何動作。

使用範例：

```
// 宣告資料節點l
student_t *node1 = (student_t *)malloc(sizeof(student_t));
strcpy(node1->ID, "A4");
node1->math = rand() % 100;
node1->eng = rand() % 100;
node1->treeNode.left = NULL;
node1->treeNode.right = NULL;
treeRoot = insertNode(node1, treeRoot, compareID); // 插入進去樹裡
```

## 第二個 function 是 deleteNode：

```
btreeNode_t *deleteNode(void *element, btreeNode_t *root, int (*compare)(void *elementA, void *elementB))
{
    btreeNode_t *temp;

    if(root == NULL) {} // 樹是空的
    else
    {
        if(compare(element, root) == -1) // 找左子
            root->left = deleteNode(element, root->left, compare);
        else if(compare(element, root) == 1) // 找右子
            root->right = deleteNode(element, root->right, compare);
        else if(compare(element, root) == 0) // 找到了
        {
            if(root->left && root->right) // 要刪的點同時有左子跟右子
            {
                temp = findMinNode(root->right); // 找右子中最小的值
                root = temp; // 使其取代他
                root->right = deleteNode(root, root->right, compare); // 刪掉要刪掉的值
            }
            else if(root->left == NULL) // 要刪的點只有右子
            {
                temp = root;
                root = root->right;
                free(temp);
            }
            else if(root->right == NULL) // 要刪的點只有左子
            {
                temp = root;
                root = root->left;
                free(temp);
            }
        }
    }
    return root;
}
```

他用於將一個節點從樹當中刪除，並回傳刪除節點後的樹給使用者。

使用者呼叫這個 function 的方式為：

**deleteNode(要刪除的節點，樹，使用者自用 function)。**

使用者要使用這個 function 時還需另外設計自己的 function 以完成他想做的工作。

### 設計這個 function 的想法：

當傳入一棵樹，首先要先檢查樹是否為空，是的話就不需做任何動作(因為在這裡要做的是刪除樹的節點)；不是的話就開始比較樹根跟要刪除的節點，找到要刪除的節點後，還需要確認這個節點是否有左子跟右子，如果只有左子或右子其中一個，那就先將 temp 指向當前 root(此時他為要刪除的節點)，再將當前的 root 指向其右子/左子，最後再釋放 temp 之記憶體空間；如果要刪除的

節點同時有右子跟左子，為了保持 binary tree 的特性，需從右子中找尋最小的節點(因為右子中最小的節點一定大於左子中最大的節點)，找到之後再將其設為新的 root，如此就可以完成刪除節點這個動作。

使用範例：

```
treeRoot = deleteNode(node1, treeRoot, compareID);
```

第三個 function 是 findMinNode：

```
btreeNode_t *findMinNode(btreeNode_t *root) //找出BST中鍵值最小的節點
{
    btreeNode_t *cur;
    while(root != NULL) // 第一圈：看root是否為空，第二圈：看root->left(此時為root)是否為空
    {
        cur = root;
        root = root->left;
    }
    return cur;
}
```

他用於尋找一棵樹當中的最小值，並回傳找到的最小值給使用者。

使用者呼叫這個 function 的方式為：findMinNode(樹)。

設計這個 function 的想法：

因為這個 function 是要找樹中最小的節點，根據 binary tree 的特性：樹根的左子<樹根<樹根的右子，那麼我只需要在樹的左子裡找尋就可以，因此做法為先宣告一個指標型態的變數 cur，當 root 存在時，cur 會在每次的 loop 都指向 root，而 root 會指向其的左子，這樣做的話第一圈的用意是在檢查 root 是否為空，而第二圈之後的用意則是檢查 root 的左子是否為空，在跳出迴圈後最後再將 cur 回傳給使用者。

使用範例：

```
printf("\nmin:");
print(findMinNode(treeRoot));
```

測試結果：

```
min:ID[A1]: math=78, eng=58
```

#### 第四個 function 是 findMaxNode :

```
btreeNode_t *findMaxNode(btreeNode_t *root) //找出BST中鍵值最大的節點
{
    btreeNode_t *cur = root;
    while(root != NULL) // 邏輯同findMinNode
    {
        cur = root;
        root = root->right;
    }
    return cur;
}
```

他用於尋找一棵樹當中的最小值，並回傳找到的最小值給使用者。

使用者呼叫這個 function 的方式為：**findMinNode(樹)**。

~~~~~  
設計這個 function 的想法：

與上一個 function **findMinNode** 的想法一樣，只是在這個 function 是從樹的右子中開始找，實際做法也一樣，在最後會將找到的節點 **cur** 回傳給使用者。

~~~~~  
使用範例：

```
printf("\nmax:");
print(findMaxNode(treeRoot));
```

測試結果：

```
max:ID[A7]: math=81, eng=27
```

~~~~~  

#### 第五個 function 是 findNode :

```
btreeNode_t *findNode(void *element, btreeNode_t *root, int (*compare)(void *elementA, void *elementB))
{
    if(root == NULL) {}
    else
    {
        if(compare(element, root) == -1) // 要找的比root小 => 在root左子
            return findNode(element, root->left, compare);
        else if(compare(element, root) == 1) // 要找的比root大 => 在root右子
            return findNode(element, root->right, compare);
        else if(compare(element, root) == 0) {} // 要找的就是root => do nothing
    }
    return root;
}
```

他用於找出使用者想在樹中找尋的節點，並回傳此節點給使用者。

使用者呼叫這個 function 的方式為：

**findNode(要找尋的節點，樹，使用者自用 function)**。

使用者要使用這個 function 時還需另外設計自己的 function 以完成他想做的工作。

~~~~~

設計這個 function 的想法：

當傳入一棵樹，首先要先檢查樹是否為空，是的話就不需做任何動作(因為如果樹是空的根本就找不到此節點)；不是的話就開始比較樹根跟要找尋的節點大小，如果小的話就去往左子搜尋，大的話則去右子搜尋，找到要搜尋的節點時則不需做任何動作，因為當前 root 就是要找的那個節點，最後再將找尋到的節點回傳給使用者。

使用範例：

```
printf("\nmax:");  
print(findMaxNode(treeRoot));
```

測試結果：

```
node6:ID[A5]: math=5, eng=45
```

.....

第六個 function 是 inOrder：

```
void inOrder(btreeNode_t *root, void (*print)(void *element)) //列印出BST根據中序追蹤法每個節點內容  
{  
    if(root)  
    {  
        inOrder(root->left, print);  
        print(root);  
        inOrder(root->right, print);  
    }  
}
```

他用於列印出使用者傳入的樹之中序遍歷順序。

使用者呼叫這個 function 的方式為：inOrder(樹，使用者自用 function)。

使用者要使用這個 function 時還需另外設計自己的 function 以完成他想做的工作。

.....

設計這個 function 的想法：

根據中序遍歷的規則，他會先拜訪左子節點，再拜訪父節點，最後拜訪右子節點。在這個 function 會使用到使用者自行設計的 function 來印出節點。

.....

使用範例：

```
printf("inorder:\n");  
inOrder(treeRoot, print);
```

測試結果：

```
inorder:  
ID[A1]: math=78, eng=58  
ID[A2]: math=34, eng=0  
ID[A3]: math=62, eng=64  
ID[A4]: math=41, eng=67  
ID[A5]: math=5, eng=45  
ID[A6]: math=69, eng=24  
ID[A7]: math=81, eng=27
```

.....

## 第七個 function 是 treeCopy :

```
void treeCopy(btreeNode_t *A, btreeNode_t *B, void (*copy)(void *elementA, void *elementB)) //把A複製到B
{
    if(A)
    {
        treeCopy(A->left, B->left, copy); // 複製左節點
        treeCopy(A->right, B->right, copy); // 複製右節點
        copy(A, B); // 把A的內容複製到B
    }
}
```

他用於將使用者傳入的兩顆樹，A 樹複製到 B 樹。

使用者呼叫這個 function 的方式為：

**treeCopy(A 樹，B 樹，使用者自用 function)。**

使用者要使用這個 function 時還需另外設計自己的 function 以完成他想做的工作。

~~~~~  
設計這個 function 的想法：

如果使用者傳入的 A 樹存在的話，就先將 A 樹的左子節點複製到 B 樹，再將 A 樹的右子節點複製到 B 樹，最後再根據使用者自行設計的 function 將 A 樹的內容複製到 B 樹。

## 第八個 function 是 treeEqual :

```
int treeEqual(btreeNode_t *A, btreeNode_t *B, int (*compare)(void *elementA, void *elementB)) //比較二個BST是否相同
{
    int flag = 1;
    if(A && B)
    {
        if(compare(A, B) != 0) // 比樹根
            flag = 0;
        else // 樹根一樣比下面的孩子
        {
            if(treeEqual(A->left, B->left, compare) == 0)
                flag = 0;
            if(treeEqual(A->right, B->right, compare) == 0)
                flag = 0;
        }
    }
    else if(!A && !B) {} // 都沒有東西
    else
        flag = 0;
    return flag;
}
```

他用於比較使用者傳入的兩棵樹(A 樹及 B 樹)是否相等，並回傳結果給使用者，1 代表相等，0 代表不相等。

使用者呼叫這個 function 的方式為：

**treeEqual(A 樹，B 樹，使用者自用 function)。**

使用者要使用這個 function 時還需另外設計自己的 function 以完成他想做的工作。

~~~~~  
設計這個 function 的想法：

先宣告一個整數型態變數 flag，將其值設為 1，接著有四種可能性：第一種可能性是 A 樹及 B 樹都存在，那麼第一步是先檢查其樹根是否相同，否的話

則將 flag 設為 0，樹根相等的话就開始比較其左子及右子是否相等，只要發現有不相等的節點，就會將 flag 設為 0；第二種可能性為 A 樹跟 B 樹都不存在，那就不需要做任何事，因為 A 樹跟 B 樹在這裡是相等的；第三及第四種可能性為 A 樹跟 B 樹其中一顆樹不存在，若出現這種情況也不需要比較，直接將 flag 設為 0，因為他們一定不會是相同的樹。最後再將結果(即 flag)回傳給使用者，flag==1 為相等，flag==0 為不相等。

---

## main.c 檔案(測試檔)說明：

在 main.c 檔裡我總共設計了兩個自用 function：compareID、print。

我的自用資料節點型態為：

```
// 資料節點
typedef struct myBST {
    btreeNode_t treeNode;
    char ID[10];
    int math;
    int eng;
} student_t;
```

.....

而第一個自用 function，是用來比較節點的 compareID：

```
// 用來比較節點的function
int compareID(void *elementA, void *elementB) {
    int i;
    char *aid = ((student_t *)elementA)->ID;
    char *bid = ((student_t *)elementB)->ID;
    for (i=0;i<10;i++) {
        if(aid[i]>bid[i]) {
            return 1;
        }else if(aid[i]<bid[i]){
            return -1;
        }
    }
    return 0;
}
```

.....

第二個自用 function，是用來將節點印出的 print：

```
// 用來將結點印出的function
void print(void *element)
{
    student_t *cur = (student_t *)element;
    printf("ID[%s]: math=%d, eng=%d\n", cur->ID, cur->math, cur->eng);
}
```

.....



在測試時我的作法是：首先先宣告一顆樹並將其初始化：

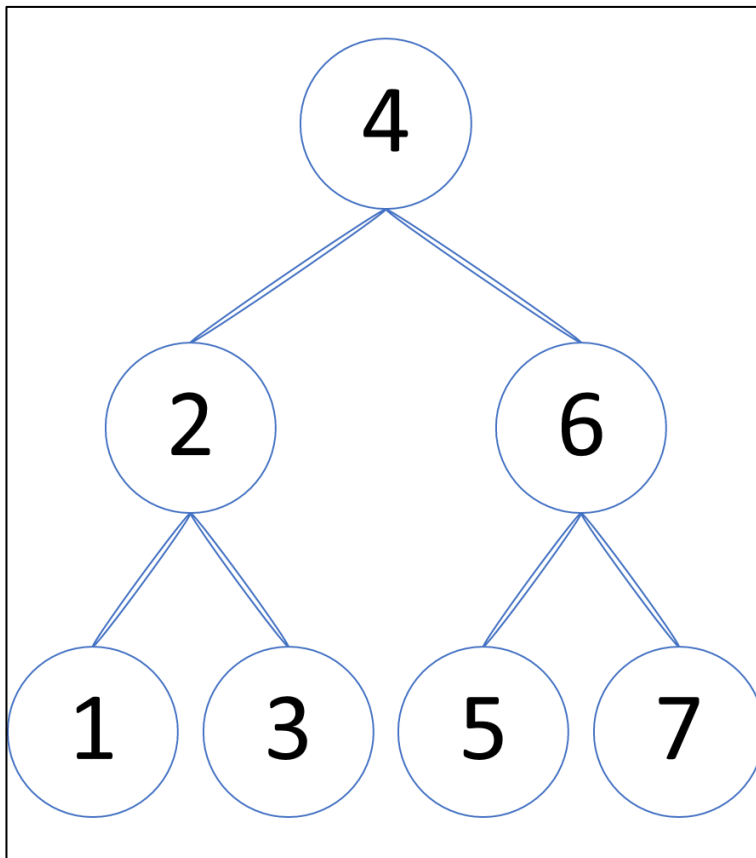
```
// 初始化BST(樹A)
bstreeNode_t *treeRoot = NULL;
```

接著宣告了七個資料節點，分別存取我要存取的資料，以第一個資料節點 node1 為例：

```
// 宣告資料節點1
student_t *node1 = (student_t *)malloc(sizeof(student_t));
strcpy(node1->ID, "A4");
node1->math = rand() % 100;
node1->eng = rand() % 100;
node1->treeNode.left = NULL;
node1->treeNode.right = NULL;
treeRoot = insertNode(node1, treeRoot, compareID); // 插入進去樹裡
```

在這裡我跟記憶體要了一塊空間用來放我的 node1，接著分別放入節點內的資料，最後再使用 library 內的 insertNode 這個 function 來將我的 node1 插入樹中。

在插入了七個節點後，這棵樹會長下面這個樣子：



(註：其中序遍歷應為：1→2→3→4→5→6→7)



接著測試 library 內的 function：

```
// 找節點
printf("\nmin:");
print(findMinNode(treeRoot));

// 找最大節點
printf("\nmax:");
print(findMaxNode(treeRoot));

// 找最小節點
printf("\nnode6:");
print(findNode(node6, treeRoot, compareID));

// 中序遍歷
printf("inorder:\n");
inOrder(treeRoot, print);
```

最後再將這七個資料節點的空間釋放回記憶體：

```
free(node1);
free(node2);
free(node3);
free(node4);
free(node5);
free(node6);
free(node7);
```

---

使用 GGC 製作靜態庫、測試檔執行結果：

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19045.2251]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。
D:\Users\hsuan\桌面\資料結構\HW2>gcc -c BST.c
D:\Users\hsuan\桌面\資料結構\HW2>ar -cr libBST.a BST.o
D:\Users\hsuan\桌面\資料結構\HW2>gcc main.c libBST.a -o test.exe
D:\Users\hsuan\桌面\資料結構\HW2>test.exe
min:ID[A1]: math=78, eng=58
max:ID[A7]: math=81, eng=27
node6:ID[A5]: math=5, eng=45
inorder:
ID[A1]: math=78, eng=58
ID[A2]: math=34, eng=0
ID[A3]: math=62, eng=64
ID[A4]: math=41, eng=67
ID[A5]: math=5, eng=45
ID[A6]: math=69, eng=24
ID[A7]: math=81, eng=27
```

由上圖可看到測試結果節點按照中序遍歷的順序印出，依序為：  
ID[A1]→ID[A2]→ID[A3]→ID[A4]→ID[A5]→ID[A6]→ID[A7]。