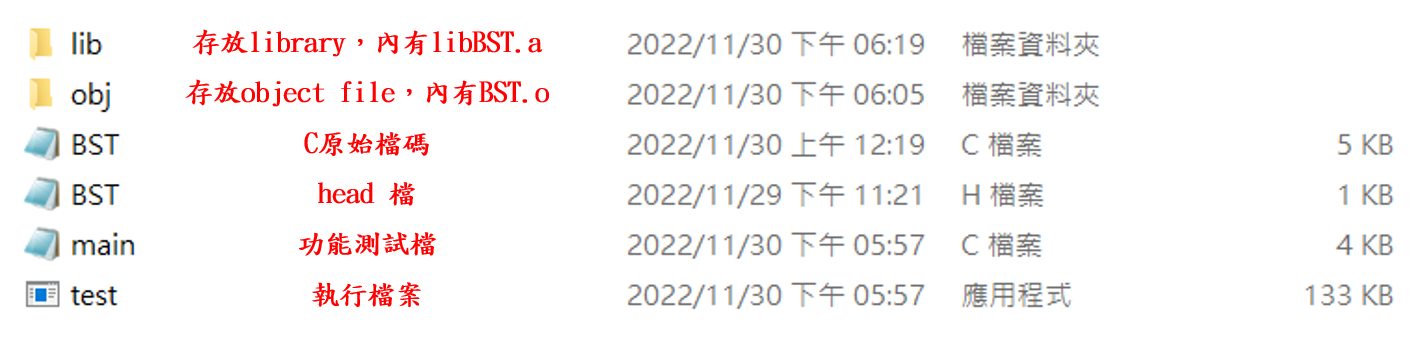
**410921202 資工三 林芷萱 DS HW2**

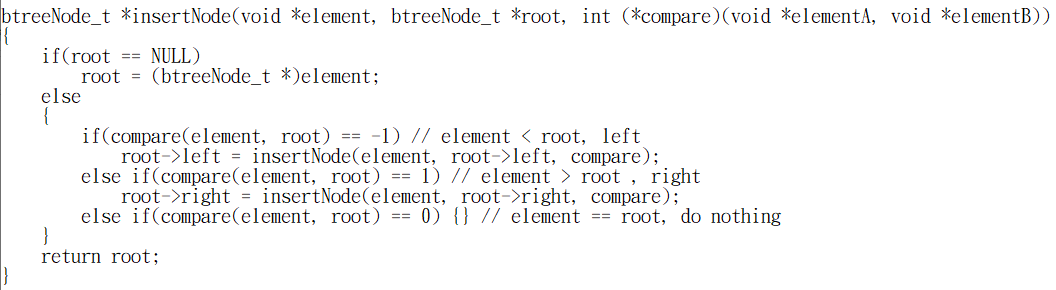
**函式清單及說明：**



**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**：



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：

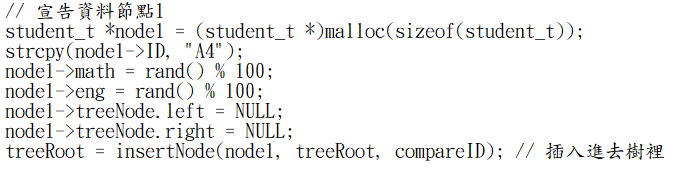
**insertNode(要插入的節點，樹，使用者自用function)**。

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。

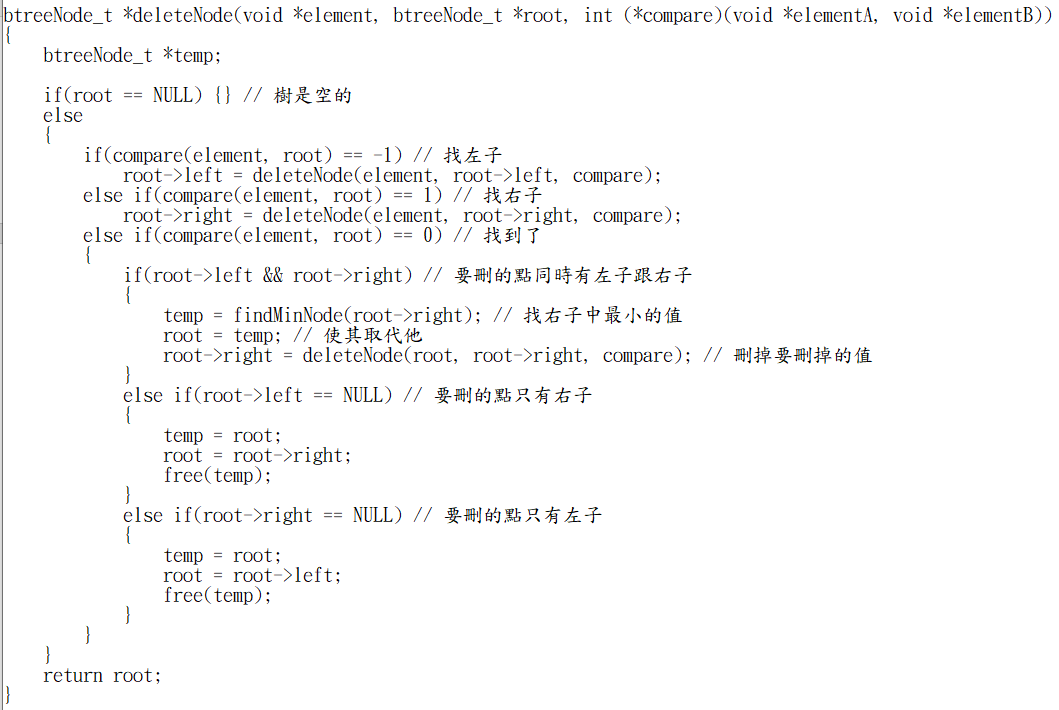
**設計這個function的想法：**

　　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：**



第二個function是**deleteNode**：



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。

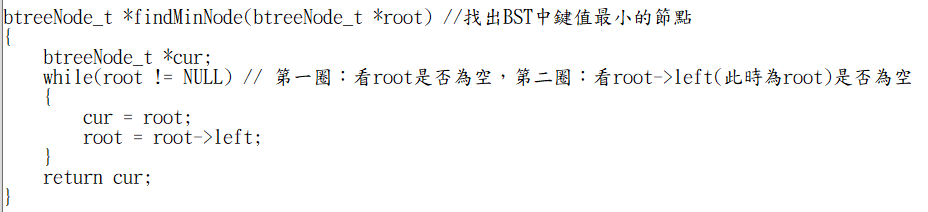
**設計這個function的想法：**

　　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：**



第三個function是**findMinNode**：



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。

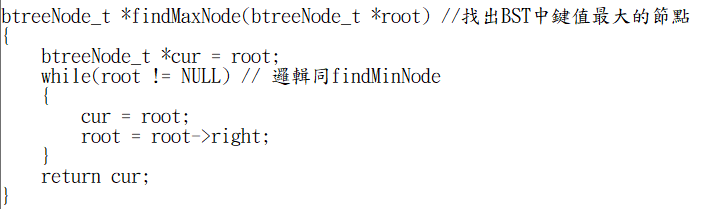
**使用範例：**



**測試結果:**



第四個function是**findMaxNode**：



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。

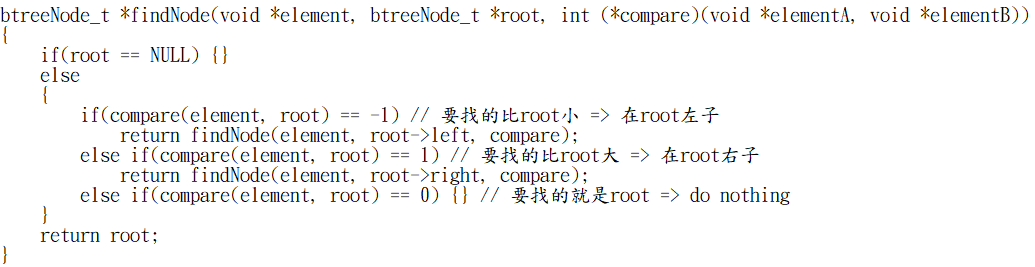
**使用範例：**



**測試結果：**



第五個function是**findNode**：



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。

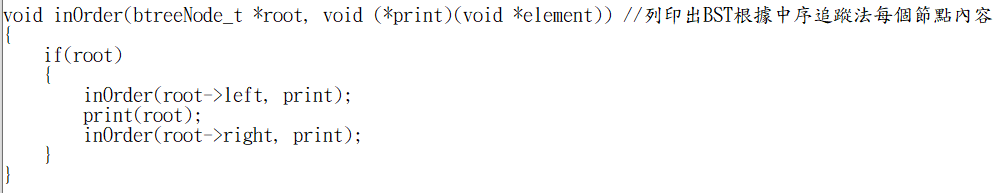
**使用範例：**

****

**測試結果：**



第六個function是**inOrder**：



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。

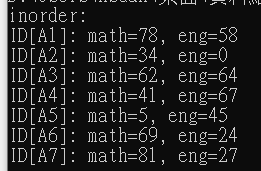
**設計這個function的想法：**

　　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。

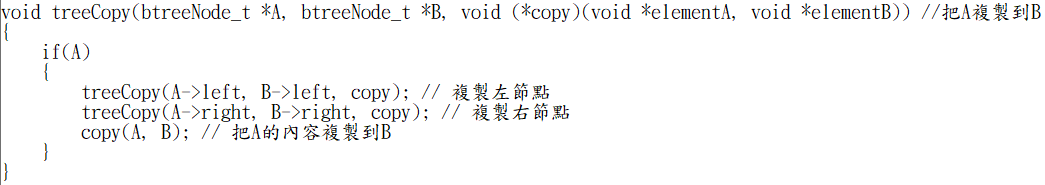
**使用範例：**

****

**測試結果：**



第七個function是**treeCopy**：



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：

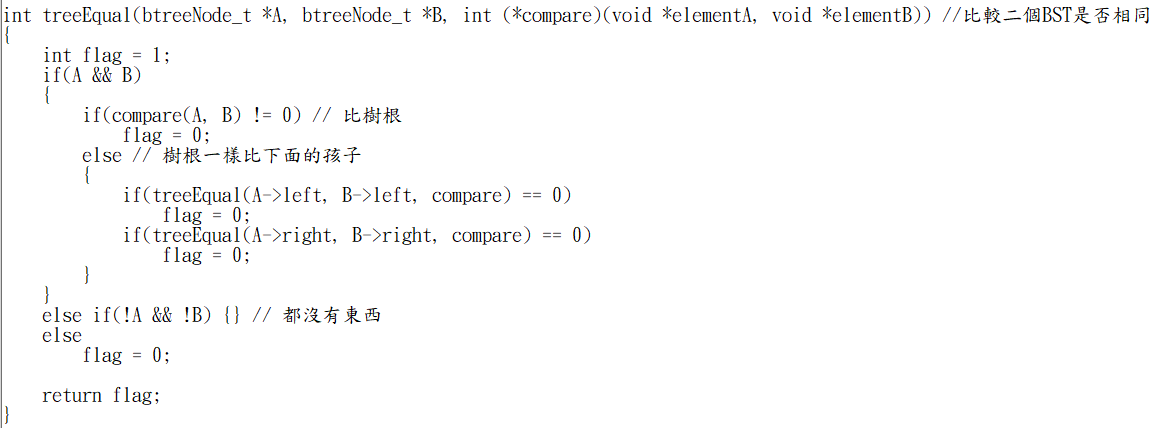
**treeCopy(A樹，B樹，使用者自用function)**。

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**：



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。

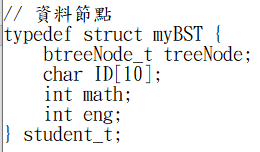
**設計這個function的想法：**

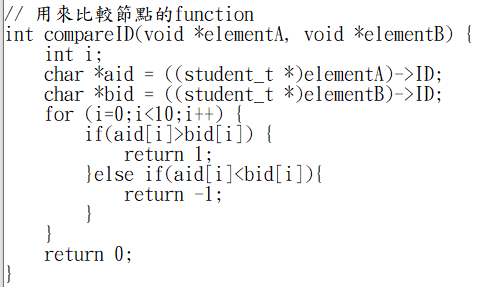
　　先宣告一個整數型態變數flag，將其值設為1，接著有四種可能性：第一種可能性是A樹及B樹都存在，那麼第一步是先檢查其樹根是否相同，否的話則將flag設為0，樹根相等的話就開始比較其左子及右子是否相等，只要發現有不相等的節點，就會將flag設為0；第二種可能性為A樹跟B樹都不存在，那就不需要做任何事，因為A樹跟B樹在這裡是相等的；第三及第四種可能性為A樹跟B樹其中一顆樹不存在，若出現這種情況也不需要比較，直接將flag設為0，因為他們一定不會是相同的樹。最後再將結果(即flag)回傳給使用者，flag==1為相等，flag==0為不相等。

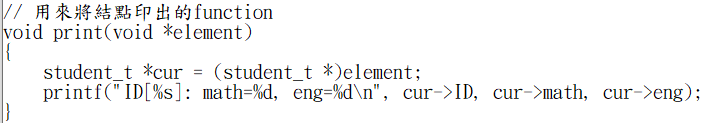
**main.c檔案(測試檔)說明：**

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

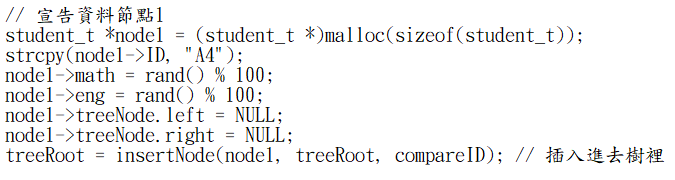
我的自用資料節點型態為：



而第一個自用function，是用來比較節點的compareID：

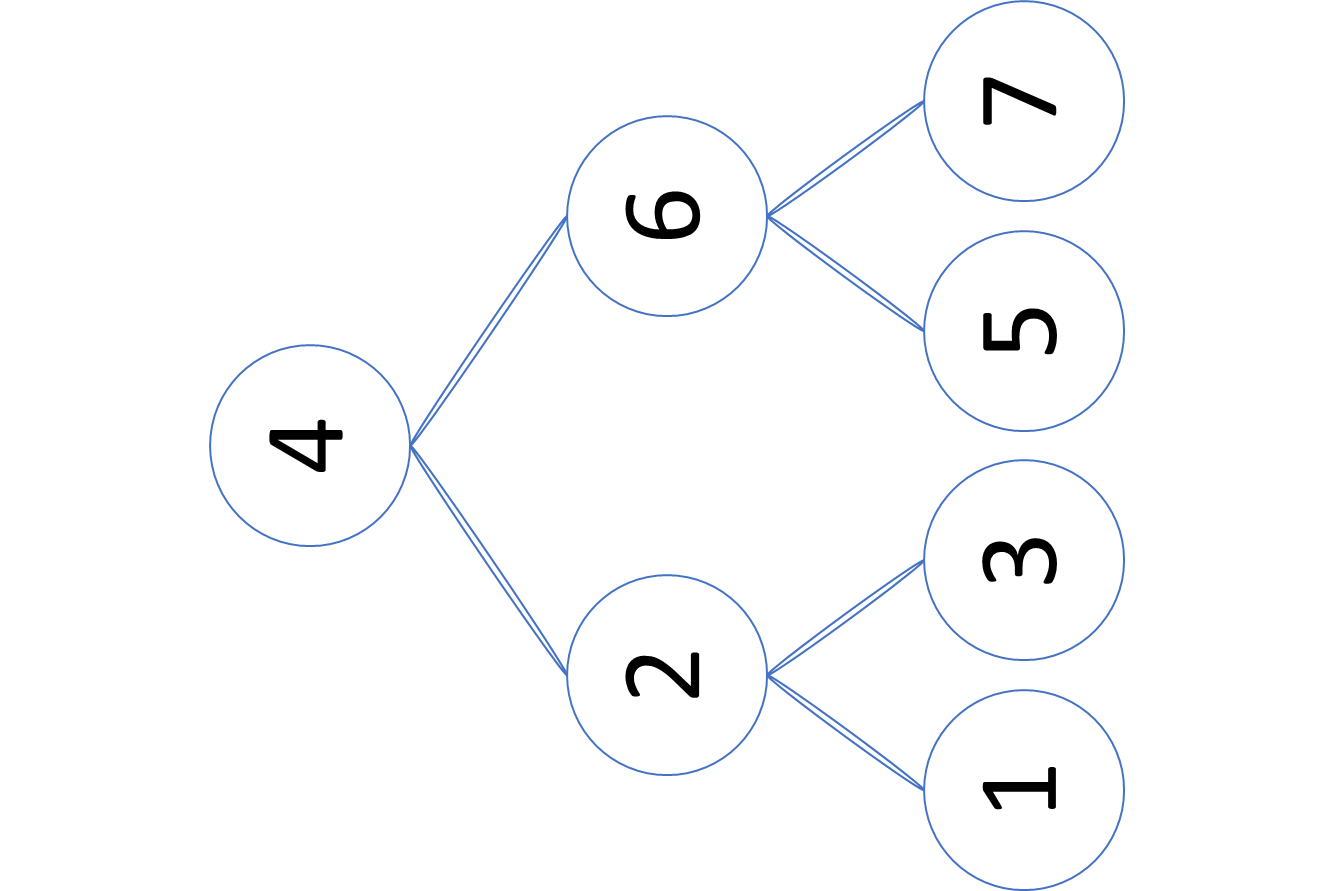
第二個自用function，是用來將節點印出的print：

在測試時我的作法是：首先先宣告一顆樹並將其初始化：

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：

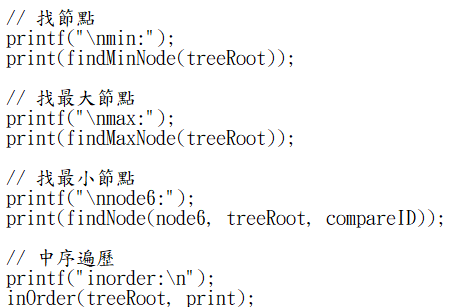
在這裡我跟記憶體要了一塊空間用來放我的node1，接著分別放入節點內的資料，最後再使用library內的inserNode這個function來將我的node1插入樹中。

在插入了七個節點後，這棵樹會長下面這個樣子：

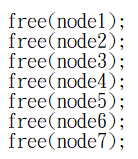


**(註：其中序遍歷應為：1→2→3→4→5→6→7)**

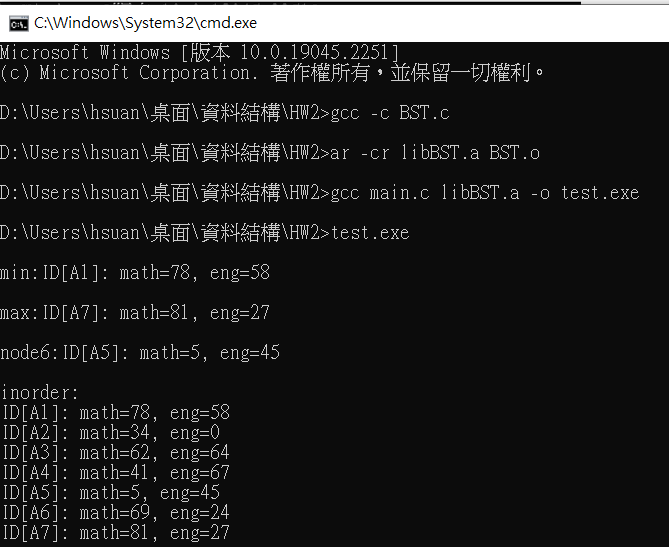
接著測試library內的function：



最後再將這七個資料節點的空間釋放回記憶體：



使用GGC製作靜態庫、測試檔執行結果：



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]。