BST.h 檔

```
#ifndef _BST_H_
#define _BST_H_

typedef struct node {
  struct node * left;
  struct node * right;
}node_t;

node_t * insertNode(void * element, node_t * root, int(*compare)(void * elementA,
void* element));//需傳入 compare 來比較大小, root 要以 node_t 型別傳入
node_t * deleteNode(void * element, node_t * root, int(*compare)(void * elementA,
void* element),void(*Replace_data)(void* elementA,void* element));//需要傳入一替
代 data 的方程式，以及一個比較大小的 compare, root 要以 node_t 型別傳入

node_t * findMinNode(void *root);//回傳最小值所在的節點
node_t * findMaxNode(void *root); //回傳最小值所在的節點
node_t * findNode(char *element,void *root,int(* compare)(char * elementA,void *
element));//給主函數尋找資料用，要比較大小所以要用到 compare
void inOrder(void *root,void(* Print_data)(void *element)); //因為要印出 data，所以要
傳進一個叫做 print  的函式來印出
node_t *treeCopy(node_t *copy,void *root,void(*Replace_data)(void*
elementA,void* element),int size);//因為要複製 data，所以要傳入一替代 data 的方程式
int treeEqual(void *tree1,void *tree2,int(*compare)(void * elementA,void *
element));
void buildTree(void * root);//  自己加的，懶得打 node->left=NULL 和 node->right=NULL
#endif // _BST_H_
```

BST.C 檔，實作

```
#include <stdio.h>
#include <stdlib.h>
#include "Bst.h"
void buildTree(void *root)
{
        ((node_t *)root)->right=NULL;
        ((node_t *)root)->left=NULL;
}
node_t * insertNode(void * element, node_t * root, int(*compare)(void * elementA,
void* element))
{
        if(root==NULL)
        {
                root=(node_t *)element;
                return root;
        }
        if(compare(element,root)==1)
        {
                root->right=insertNode(element, root->right, compare);

        }
        else if(compare(element,root)==-1)
        {
                root->left=insertNode(element, root->left, compare);

        }
        else
        {
                printf("already have this data\n");
                return NULL;
        }
        return root;
}
node_t * deleteNode(void * element, node_t * root, int(*compare)(void * elementA,
void* element),void(*Replace_data)(void *elementA,void *element))
{
        node_t *temp;
        if(root == NULL)
                return NULL;
        if(compare(element,root)==1)
        {
                root->right=deleteNode(element,root->right,compare,Replace_data);
        }
        else if(compare(element,root)==-1)
        {
                root->left=deleteNode(element,root->left,compare,Replace_data);
        }
```

```
	else//find the data
	{
		if(root->left!=NULL&&root->right!=NULL)
		{
			temp=findMinNode(root->right);
			Replace_data(root,temp);
			root->right=deleteNode(temp,root->right,compare,Replace_data);
		}
		else if(root->left==NULL)
		{
			temp=root;
			root=root->right;
			free(temp);
		}
		else
		{
			temp=root;
			root=root->left;

		}
	}
	return root;
}
node_t * findMinNode(void *root)
{
	if((node_t *)root==NULL||((node_t *)root)->left==NULL)
		return (node_t *)root;
	return findMinNode(((node_t *)root)->left);
}
node_t * findMaxNode(void *root)
{
	if((node_t *)root==NULL||((node_t *)root)->right==NULL)
		return (node_t *)root;
	return findMinNode(((node_t *)root)->right);
}
node_t * findNode(char * element, void * root,int(*compare)(char * elementA,void
* element))
{
}
	if((node_t *)root==NULL)
	{
		puts("There is no according data");
		return NULL;
	}
	if(compare(element,root)==1)
		return findNode(element,((node_t *)root)->right,compare);
	else if(compare(element,root)==-1)
		return findNode(element,((node_t *)root)->left,compare);
	else
		return (node_t *)root;

}
void inOrder(void *root,void(* Print_data)(void *element))
{
	if(root!=NULL)
	{
		inOrder(((node_t *)root)->left, Print_data);
		Print_data(root);
		inOrder(((node_t *)root)->right, Print_data);
	}
}
node_t *treeCopy(node_t *copy,void *root,void(*Replace_data)(void*
elementA,void* element),int size)
{
	if((node_t *)root!=NULL)
	{
		node_t *L=(node_t *)malloc(sizeof(size));
		node_t *R=(node_t *)malloc(sizeof(size));
		copy->left=treeCopy(L,((node_t *)root)->left,Replace_data,size);
		copy->right=treeCopy(R,((node_t *)root)->right,Replace_data,size);
		Replace_data(copy,root);
		return copy;
	}
	return NULL;
}
int treeEqual(void *tree1,void *tree2,int(*compare)(void * elementA,void *
element))
```

```
{

if((tree1==NULL&&tree2==NULL)||((tree1!=NULL&&tree2!=NULL&&compare(tree1,t
ree2)==0)&&(treeEqual(((node_t *)tree1)->left,((node_t *)tree2)->left,compare))!=-
1&&(treeEqual(((node_t *)tree1)->right,((node_t *)tree2)->right,compare))!=-1))
        {
                return 1;
        }
        else
                return -1;
}
```