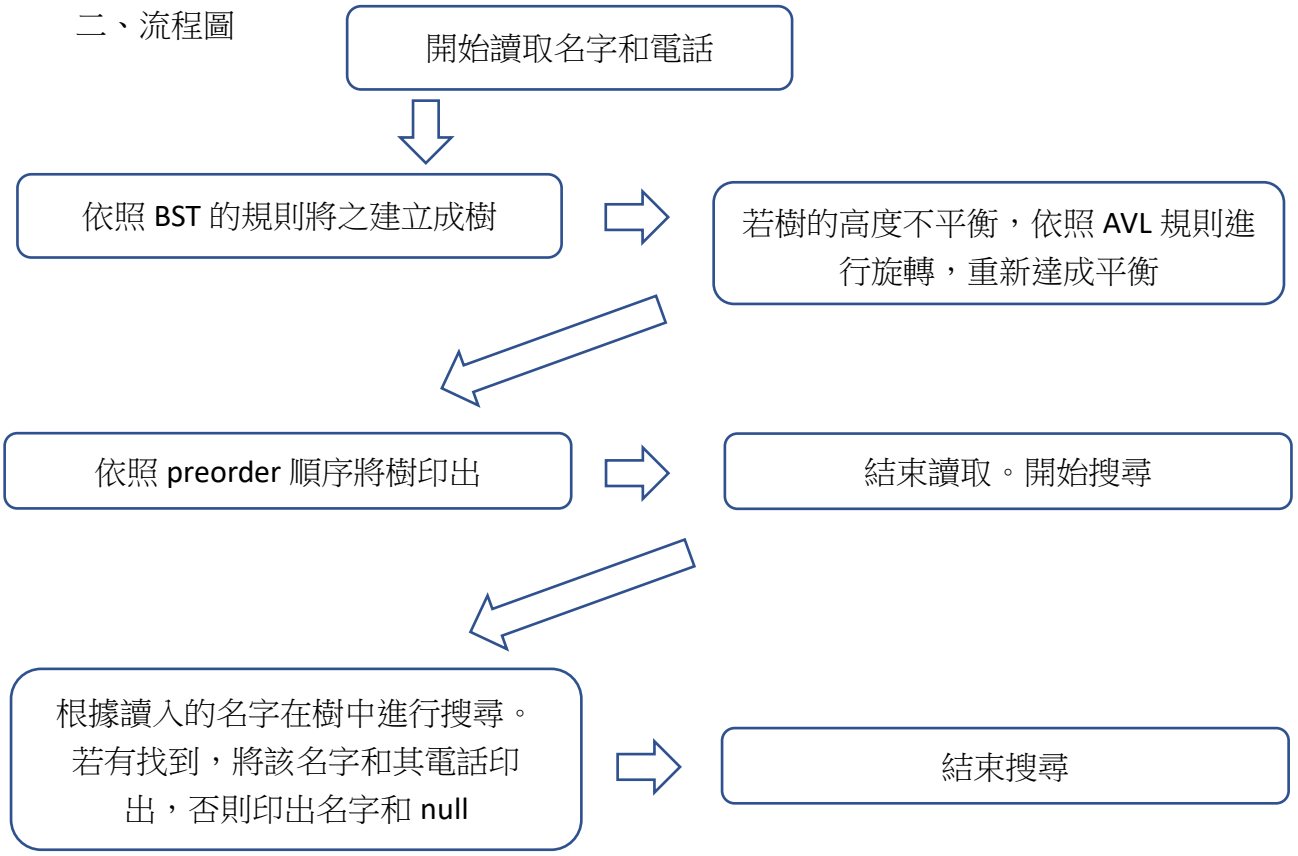
一、執行結果

```
User@LEO MINGW64 ~
$ cd /d/Users/User/Desktop/DS_hw6/code

User@LEO MINGW64 /d/Users/User/Desktop/DS_hw6/code
$ gcc -std=c11 -o hw6 hw6.c

User@LEO MINGW64 /d/Users/User/Desktop/DS_hw6/code
$ ./hw6.exe < input0_windows.txt > ans_output0_windows.txt

User@LEO MINGW64 /d/Users/User/Desktop/DS_hw6/code
$ diff ./output0_windows.txt ./ans_output0_windows.txt
```

```
David Bob Alice Charlie Paul Ruby
Paul 0900000002
Amy null
Ruby 0900000004
```

二、流程圖



三、函式說明

- struct node *insert(struct node *node, char *name, char *phone)
  一開始同一般 BST。不一樣的是每個 node 額外存了它的高度和平衡因子(BF)。當當前 node 的 BF 值=正負 2，就看它是屬於 LL, LR, RR, RL 四種型態的哪一種，並分別進行旋轉。
- int getHeight(struct node *node)
  回傳 node 的高度。若 node==null，回傳-1。
- int getBalanceFactor(struct node *node)
  回傳 node 的 left 高度減去 node 的 right 高度的結果。若 node==null，回傳 0。
- struct node *LeftRotation(struct node *node)
  先宣告 tmp 指向 node 的 right，tmp2 指向 node 的 right 的 left。再將 node 的 right 改成指向 tmp2，tmp 的 left 改指向 node。然後對 node 和 tmp 的高度和 BF 做更新。最後回傳 tmp。
- struct node *RightRotation(struct node *node)
  為 LeftRotation 的鏡像。
- void printPreorder(struct node *node)
  依照 NLR 的順序印出值。
- searchAndPrint(struct node *node, char *searchWho)
  如果 searchWho 和 node 存的名字符合，表示找到，印出該名字和其電話。否則就遞迴往深層去找。當到達 null 的節點時，表示都沒找到 searchWho，則印出該名字和 null。