Data Structure Assignment #3

程式相關報告

工資系 113 H54094015 張柏駿

```cpp
// C++ code
#include <iostream>
#include <string>
#include <queue>
#include <fstream>
#include <iomanip>
using namespace std;

class BST;
class TreeNode{
private:
    TreeNode *leftchild;
    TreeNode *rightchild;
    TreeNode *parent;
    int key;
    string element;
public:
    TreeNode():leftchild(0),rightchild(0),parent(0),key(0),element(""){};
    TreeNode(int a, string b):leftchild(0),rightchild(0),parent(0),key(a),element(b){};

    int GetKey(){return key;}              // 為了在main()要能夠檢視node是否正確
    string GetElement(){return element;}   // 才需要這兩個member function讀取private data

    // 其餘情況, 因為class BST是class TreeNode的friend class
    // 在class BST的member function中, 可以直接存取class TreeNode的private data

    friend class BST;   // 放在 private 或 public 都可以
};

```

這邊完全使用助教課提供的 BST 程式做模板，在 class Treenode 中沒有做更動，僅在標頭檔新增了後續會使用到的部分，如讀檔的<fstream>。

```cpp
class BST{
private:
    TreeNode *root;
    TreeNode* Leftmost(TreeNode *current);
    TreeNode* Successor(TreeNode *current);

public:
    BST():root(0){};
    TreeNode* Search(int key);
    void InsertBST(int key, string element);
    void InorderPrint();
    void PrintTree(TreeNode *current, int level);   //新增的function 以列印出樹的形狀
};
```

新增一個函數 PrintTree，有兩個引數，以印出樹的形狀。

```cpp
void BST::InorderPrint(){
    TreeNode *current = new TreeNode;
    current = Leftmost(root);
    while(current){
        cout << "Branch " << current->element << "'s profit: " << current->key << endl;  //按照格式輸出
        current = Successor(current);
    }
}
```

將 inorder 調整成好看的型式，使用助教課的 function，沒有做更動。

```
void BST::PrintTree(TreeNode *current, int level){
    if (current != NULL) {                                  // if current != NULL 用preorder
        for (int i = 1; i < level; i++) {
        cout << "\t";                                       //透過level每做一次就+1
                                                            //控制不同層要印幾個"\t"(8個空白)
        //cout << setw(10*i);
        }
        cout << current->element << " " << current->key << endl; //按照格式輸出
        PrintTree(current->leftchild, level+1);       // L遞迴
        PrintTree(current->rightchild, level+1);      // R遞迴

    }
}
```

設定一個函數可以印出樹的形狀，用 for 迴圈設定要印出的空白的數量，原本用 setw 設定，但會吃到第一個 root node 的資料，發現好像用”\t” ，比較不會跑掉，用老師教過的 preorder 作遞迴，裡面有兩個引數，一個為 current 指標，代表現在要印的節點資料，一個為第幾層，同一層要有相同的空白數量，每進到左小孩或右小孩節點，層數(level)也會進到下一層。

```
int main(){
    BST T;                                                  //二元搜尋樹物件
    ifstream in;
    ofstream out;                                           //建立檔案相關物件
    in.open("profit.txt");                                  //開啟讀取檔案
    string a;
    int b, c;
    cout << "\nReading data(profit.txt)....." << endl;
    while(!in.eof()){                                       //只要還沒讀到完，條件成立就繼續一直讀
        in >> a >> b >> c;                                  //傳入檔案內的資料並依照讀取順序輸出
        cout << "Branch " << a << "'s profit: " << c-b <<endl;
        T.InsertBST(c-b, a);                                //將檔案內資料插入進二元搜尋樹
    }
out.close();                                                //關檔
```

```
    cout << "\nThe shape of the tree:\n";
    TreeNode *node = T.Search(29);                          //利用class 內的search函數找到第一個root node
    T.PrintTree(node,1);                                    //輸出樹的形狀(從level 1開始)
    cout << "\nAfter Inorder Traversal:\n";
    T.InorderPrint();                                       //inorder 結果(由小到大排列)


    return 0;
};
```

主程式部分，先建立 BST 物件，輸出輸入物件，並且作讀檔，用 eof()設定 while 迴圈，讀取到沒有資料為主，傳入資料為 string(分店)/int(支出)/int(收

入)，依照檔案讀取順序輸出分店對應到的盈餘，並且將盈餘及分店按照助教課提供的 class 中的 insertBST 將資料建置進二元搜尋樹後關檔。透過 search 的函式來找到 current，並將層數先設定為 1，傳入 PrintTree 中 traversal 依照 preorder 順序來列印出作業要求的樹的形狀，最後再利用 inorder 將盈餘及對應的分店由小到大輸出。

範例輸出結果

```
PS D:\C++> cd "d:\C++\資料結構\" ; if ($?) { g++ hw_3_H54094015.cpp -o hw_3_H54094015 } ;
15 }

Reading data(profit.txt).....
Branch N_1's profit: 29
Branch N_2's profit: 13
Branch N_3's profit: 24
Branch N_4's profit: 18
Branch N_5's profit: 18
Branch M_1's profit: 20
Branch M_2's profit: 23
Branch M_3's profit: 12
Branch M_4's profit: 16
Branch M_5's profit: 16
Branch S_1's profit: 14
Branch S_2's profit: 25
Branch S_3's profit: 21
Branch S_4's profit: 24
Branch S_5's profit: 29

The shape of the tree:
N_1 29
        N_2 13
                M_3 12
                N_3 24
                        N_4 18
                                M_4 16
                                        S_1 14
                                        M_5 16
                                N_5 18
                                        M_1 20
                                                M_2 23
                                                        S_3 21
                        S_2 25
                                S_4 24
        S_5 29
```

```
After Inorder Traversal:
Branch M_3's profit: 12
Branch N_2's profit: 13
Branch S_1's profit: 14
Branch M_4's profit: 16
Branch M_5's profit: 16
Branch N_4's profit: 18
Branch N_5's profit: 18
Branch M_1's profit: 20
Branch S_3's profit: 21
Branch M_2's profit: 23
Branch N_3's profit: 24
Branch S_4's profit: 24
Branch S_2's profit: 25
Branch N_1's profit: 29
Branch S_5's profit: 29
```