Meeting Planner

1 Overview

This planner is mainly depending on the binary search trees, every binary search tree has a root, child, and leaf.

The root is the top node of the tree and is the key of entering the tree.

The child is every node that does not a root, and have at least one child.

The leaf is the node that does not have any children.

This planner supports five main operations: Add a new plan, modify an existing plan, find an existing plan, delete a plan, and print all the existing plans.

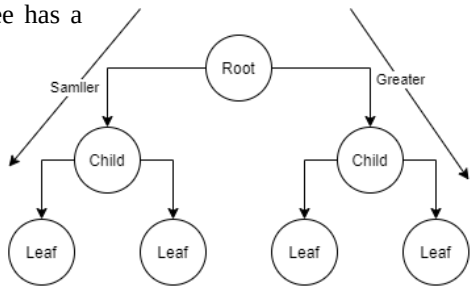

Figure 1 – Binary Search Tree

2 Main Flows

This code is divided into three files main.cpp, BST.cpp, and BST.h

As depicted in figure 2, is the BST.h file that has the main declaration of the class of the Binary Search tree and the functions used to implement the five operations of the planner.

The tree data is divided into 4 main categories, title is containing the title of the plan, key is the data stored in the tree, and this key contain the information of the day and the hour at the same time to reduce the complexity of the implementation I multiplied the day with 100 and added the value of the hour. (as example if the user entered 1 1 therefore the key value will be 101 and so on.), a pointer that points to the left of the tree, and one to the right.

Every function is having two version a public version that allows the user to interface with it, and a private function that has the main definition as an abstraction layer.

All what the public function do is to call the private function, and the private function will make what is desired.

The following figure 3 and 4 illustrate the main idea.

```
class BST
{
private:
    struct node{
        int key;
        string title;
        node* left;
        node* right;
    };
    node* root;
    void AddLeafPrivate(int d, int h, const string t, node* Ptr);
    void PrintInOrderPrivate(node* Ptr);
    node* ReturnNodePrivate(int d,int h, node* Ptr);
    node* MODPrivate(int d, int h, const string t, node* Ptr);
    int FindSmallestPrivate(node* Ptr);
    string ReturnTitlePrivate(int key, node* Ptr);
    void RemoveNodePrivate(int key, node* parent);
    void RemoveRootMatch();
    void RemoveMatch(node* parent, node* match, bool left);
public:
    BST();
    node* CreateLeaf(int key, const string t);
    void AddLeaf(int d, int h, const string t);
    void PrintInOrder();
    node* ReturnNode(int d, int h);
    node* MOD(int d, int h, const string t);
    int ReturnRootKey();
    void PrintChildren(int key);
    int FindSmallest();
    string ReturnTitle(int key);
    void RemoveNode(int key);
};
```

Figure 2 - The declaration of the BST

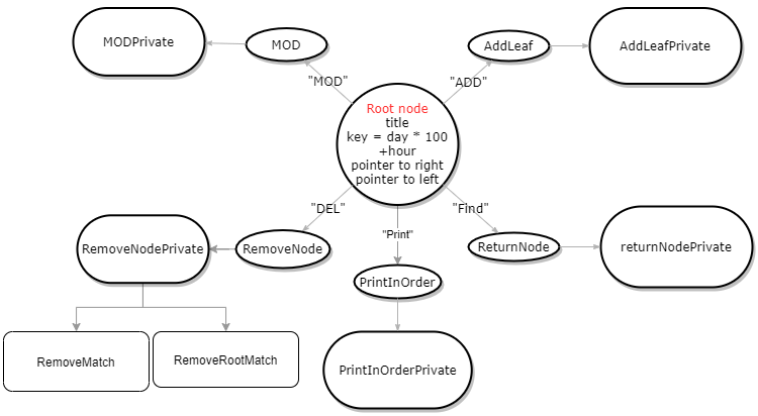

Figure 3 - The main operations of the planner

Key
101
102
103
...
...
...
36521
36522
36523

Figure 4 - The key value

The following flowchart illustrates the main procedure of the code:

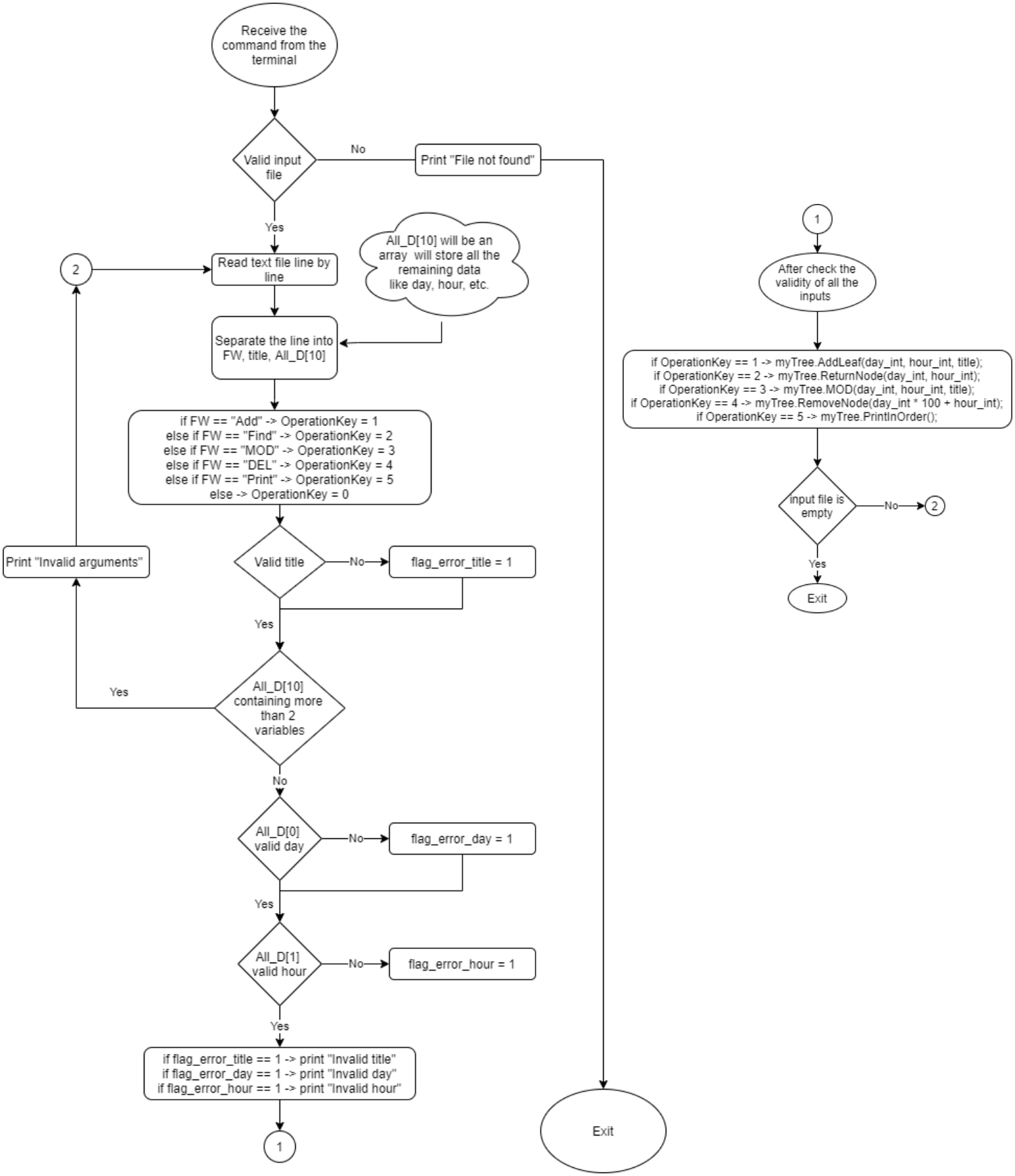

Figure 5 - The main procedure