Lab 4: Search Data Structure

1 Binary Tree - Binary Search Tree

Each Node of a Binary (Search) Tree is define as follow:

```
struct NODE{
    int key;
    NODE* left;
    NODE* right;
};
```

Students are required to implement the following functions:

1. Initialize a `NODE` from a given value:
   - `NODE* createNode(int data)`
2. Add a `NODE` with given value into a given Binary Search Tree:
   - `void Insert(NODE* &pRoot, int x)`
3. Pre-order Traversal:
   - `void NLR(NODE* pRoot)`
4. In-order Traversal:
   - `void LNR(NODE* pRoot)`
5. Post-order Traversal:
   - `void LRN(NODE* pRoot)`
6. Level-order Traversal:
   - `void LevelOrder(NODE* pRoot)`
7. Calculate the height of a given Binary Tree;
   - `int Height(NODE* pRoot)`
8. Count the number of `NODE` from a given Binary Tree:
   - `int countNode(NODE* pRoot)`
9. Calculate the total value of all `NODE`s from a given Binary Tree:
   - `int sumNode(NODE* pRoot)`
10. Find and return a `NODE` with given value from a given Binary Search Tree:
    - `NODE* Search(NODE* pRoot, int x)`

11. Remove a `NODE` with given value from a given Binary Search Tree:
    - `void Remove(NODE* &pRoot, int x)`
12. Initialize a Binary Search Tree from a given array:
    - `NODE* createTree(int a[], int n)`
13. Completely remove a given Binary Search Tree:
    - `void removeTree(Node* &pRoot)`
14. Calculate the height of a `NODE` with given value: *(return -1 if value not exist)*
    - `heightNode(NODE* pRoot, int value)`
15. * Calculate the level of a given `NODE`:
    - `int Level(NODE* pRoot, NODE* p)`
16. * Count the number leaves from a given Binary Tree:
    - `int countLeaf(NODE* pRoot)`
17. * Count the number of `NODE` from a given Binary Search Tree which key value is less than a given value:
    - `int countLess(NODE* pRoot, int x)`
18. * Count the number of `NODE` from a given Binary Search Tree which key value is greater than a given value:
    - `int countGreater(NODE* pRoot, int x)`
19. * Determine if a given Binary Tree is Binary Search Tree:
    - `bool isBST(NODE* pRoot)`
20. * Determine if a given Binary Tree is a Full Binary Search Tree:
    - `bool isFullBST(NODE* pRoot)`

2 AVL Tree

Each Node of an AVL Tree is define as follow:

```
struct NODE{
    int key;
    NODE* left;
    NODE* right;
    int height;
};
```

Students are required to implement the following functions:

1. Initialize aNODEfrom a given value:
   - `NODE* createNode(int data)`
2. Add a `NODE` with given value into a given AVL tree (Notify if the given value existed):
   - `void Insert(NODE* &pRoot, int x)`
3. Remove a `NODE` with given value from a given AVL Tree(Notify if the given value not existed):
   - `void Remove(NODE* &pRoot, int x)`
4. * Determine if a given Binary Tree is an AVL Tree:
   - `bool isAVL(NODE* pRoot)`

3 Hash Table

3.1 Content

The tax code information in this lab is collected from 1250 Vietnam company. You can find it in *"MST.txt"*, which has the content as follow:

```
Ten cong ty|MST|Dia chi
CONG TY TNHH BEE VIET NAM|0108927262|So 8 - K8, Khu nha o lien ke trung tam 75, Tong cuc II, Bo Quoc Phong, thon Lai Xa, Xa Kim Chung, Huyen Hoai Duc, Thanh pho Ha Noi
CONG TY CO PHAN THUONG MAI CHAU DUC PHAT|3502406778|So 266 Ap Phuoc Trung, Xa Tam Phuoc, Huyen Long Dien, Tinh Ba Ria - Vung Tau
CONG TY CO PHAN XAY DUNG DAU TU PHAT TRIEN DI SAN SAO VIET|0315938079|30/18 Truong Sa, Phuong 17, Quan Binh Thanh, Thanh pho Ho Chi Minh
CONG TY TNHH MTV THAN TAN HOANG LONG|0315938103|2/47 Duong Thanh Loc 31, Khu Pho 3C, Phuong Thanh Loc, Quan 12, Thanh pho Ho Chi Minh
CONG TY TNHH NONG NGHIEP CONG NGHE CAO MIEN DONG VIET|3401194911|Thon 5, Xa Tan Phuc, Huyen Ham Tan, Tinh Binh Thuan
CONG TY TNHH XAY DUNG VINH GIA PHAT|3603671412|So 171, Xom 4, Khu 2, Ap Bau Ca, Xa Trung Hoa, Huyen Trang Bom, Tinh Dong Nai
CONG TY TNHH THUONG MAI DICH VU PHU LONG RIVERSIDE|3401194929|243 Huynh Thuc Khang, KP1, Phuong Mui Ne, Thanh pho Phan Thiet, Tinh Binh Thuan
CONG TY TNHH HANH TRANG PHAT|3603671155|To 5, Ap Thanh Binh, Xa Loc An, Huyen Long Thanh, Tinh Dong Nai
CONG TY TNHH THUONG MAI DICH VU THIET BI M.K.K|0315932380|154/1/34 Cong Lo, Phuong 15, Quan Tan Binh, Thanh pho Ho Chi Minh
CONG TY TNHH TM - DV - NHA HANG HAI SAN KY QUANG|0315933352|So 526 Duong Pham Van Dong, Phuong 13, Quan Binh Thanh, Thanh pho Ho Chi Minh
CONG TY TNHH MTV KIM LONG|1201613551|So 170 Nguyen Minh Duong, Ap 1, Xa Dao Thanh, Thanh pho My Tho, Tinh Tien Giang
CONG TY TNHH SONA AGENCY VIET NAM|0108926660|So 333 Bach Mai, Phuong Bach Mai, Quan Hai Ba Trung, Thanh pho Ha Noi
```

in which:

- The first line provides the included information fields.
- For the next lines, each one is the information of 1 company, separated by a straight dash (|).

For this lab, students are required to read the info of Companies from the *"MST.txt"* file into the `Company` data structure, and store as a hash table.

3.2 Programming

The `Company` data structure is defined as follow:

```
struct Company
{
    string name;
    string profit_tax;
    string address;
};
```

Fulfill the following requirements:

1. Read the companies information from a given file:
   - `vector<Company> ReadCompanyList(string file_name)`
   - Input: `file_name` direction to the input file (*"MST.txt"* for this lab).
   - Output: Companies list extracted from the file, which has the data type `vector<Company>`.
2. Hash a string (company name) function:
   - `long long HashString(string company_name)`
   - Input: `company_name` is the string (company name), that need to be hashed.
   - Output: `long long` positive integer, result of the hash formula given below.

- Hash formula:

$$hash(s) = (\sum_{i=0}^{n-1}(s[i] \times p^i))\ mod\ m$$

  in which:
  - `s` Last 20 characters of the `company_name`. The whole string is required if its size doesn't exceed 20.
  - `s[i]` ASCII code of the character at position `i` from `s`.
  - $p = 31$
  - $m = 10^9 + 9$

3. The function to create a hash table of size 2000, generated from the Companies list:
   - `Company* CreateHashTable(vector<Company> list_company)`
   - Input: `list_company` Companies list extracted from file.
   - Output: Generated hash table.
   - Note: Linear probing is required.
4. Add the info of 1 company into an existed hash table:
   - `void Insert(Company* hash_table, Company company)`
   - Input: - `hash_table`: Given hash table.
     - `company` the string name of the company, which need to be hashed.
5. Search for company information by its name:
   - `Company* Search(Company* hash_table, string company_name)`
   - Input: - `hash_table` Given hash table.
     - `company_name` the string name of the company, which data is needed.
   - Output: Information of the required company, store as `Company` data structure. Return NULL if the company cannot be found.