Assignment 5

**Abstract**

Complete the EmployeeDB class to store Employee Objects. The EmployeeDB class will store Employee objects using a binary search tree. Create a driver to provide a text based menu for the EmployeeDB class.

**Employee Class**

The Employee Class is provided in the Employee.h file. This class is complete and requires no changes. Each Employee object represents a single employee record to be stored.

**EmployeeDB Class**

The EmployeeDB class will store employee records using the EmpBinaryTree class.

Each EmployeeDB instance should create a single EmpBinaryTree instance to store many Employee objects.

The EmployeeDB specification is provided. The EmployeeDB.h file is complete and should not be changed. Complete this class in the separate EmployeeDB.cpp file.

The EmployeeDB will have the following methods:

* insertEmployee: This method takes a single employee as an argument and inserts the employee into the BST.
* deleteEmployee: This method takes a (temp) employee as an argument then deletes the Employee from the BST with a matching name.
* searchEmployee: This method accepts a string as an argument, then searches the BST for an Employee with a name matching this string argument.
* displayRecords: This method prints the name of every Employee using an InOrder BST traversal.

**EmpBinaryTree**

See the provided EmpBinaryTree specification in EmpBinaryTree.h. Complete this class in the separate EmpBinaryTree.cpp file.

The EmpBinaryTree class represents a Binary Search Tree of Employee Objects.

**EmployeeDB Driver**

Provide a looping menu with the following options:

- Insert New employee. This creates a new employee, prompts for employee info, updates the employee, inserts this employee into the database.

- Delete Employee: This prompts the user for the name of the employee to search for then deletes the Employee with this name.

- Search for an employee: Prompt the user for a name to search for. If the Employee exists, print out all of the information about this Employee. Otherwise: Print error message.

- Display all of the employee records: Display all of the records to the screen using the displayInOrder() method from the BST.

**Class Source Code**

The provided Employee.h, EmpBinaryTree.h, EmployeeDB.h files are complete and should not be modified.

Complete the EmpBinaryTree class in EmpBinaryTree.cpp, then complete the EmployeeDB class in EmployeeDB.cpp, then create a useful driver application in main.cpp.

The classes must be defined in a header file with no implementation. Additionally, create an implementation file with all of the implementation code. See the provided Die class example for an example of this format.

**Deliverables**

Provide the source files for your project.

Provide screenshots of your driver output. (10% of score)

Submit only your own original work before the posted due date on Blackboard.

Late or unoriginal work will not be accepted.

All needed files are provided below

EmpBinaryTree.h

#ifndef EMPBINARYTREE\_H

#define EMPBINARYTREE\_H

#include "Employee.h"

#include <string>

class EmpBinaryTree

{

private:

struct TreeNode

{

Employee e;

TreeNode \*left;

TreeNode \*right;

};

TreeNode \*root;

void insert(TreeNode \*&, TreeNode \*&);

void destroySubTree(TreeNode \*);

void deleteNode(Employee, TreeNode \*&);

void makeDeletion(TreeNode \*&);

void displayInOrder(TreeNode \*) const;

void displayPreOrder(TreeNode \*) const;

void displayPostOrder(TreeNode \*) const;

public:

// Constructor

EmpBinaryTree()

{ root = nullptr; }

// Destructor

~EmpBinaryTree()

{ destroySubTree(root); }

// Binary tree operations

void insertEmployee(Employee);

bool searchEmployee(Employee);

Employee getEmployee(string);

void remove(Employee);

void displayInOrder() const

{ displayInOrder(root); }

void displayPreOrder() const

{ displayPreOrder(root); }

void displayPostOrder() const

{ displayPostOrder(root); }

};

#endif

Employee.h

// Specification file for the Employee class

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

#include <string>

using namespace std;

// Exception class

class InvalidEmployeeNumber {};

class Employee

{

private:

string name; // Employee name

string number; // Employee number

string hireDate; // Hire date

public:

// Default constructor

Employee()

{ name = ""; number = ""; hireDate = ""; }

// Constructor

Employee(string aName, string aNumber, string aDate)

{ // Test aNumber...

int num = atoi(aNumber.data());

if (num >= 0 && num <= 9999)

{

// aNumber is valid.

name = aName;

number = aNumber;

hireDate = aDate;

}

else

throw InvalidEmployeeNumber();

}

// Mutators

void setName(string n)

{ name = n; }

void setNumber(string aNumber)

{ int num = atoi(aNumber.data());

if (num >= 0 && num <= 9999)

number = num;

else

throw InvalidEmployeeNumber();

}

void setHireDate(string date)

{ hireDate = date; }

// Accessors

string getName() const

{ return name; }

string getNumber() const

{ return number; }

string getHireDate() const

{ return hireDate; }

};

#endif /\* EMPLOYEE\_H\_ \*/

**eMPLOYEEbd.H**

#ifndef EMPLOYEEDB\_H\_

#define EMPLOYEEDB\_H\_

#include "EmpBinaryTree.h"

#include <string>

class EmployeeDB{

private:

EmpBinaryTree db;

public:

EmployeeDB();

void insertEmployee(Employee);

bool deleteEmployee(Employee);

Employee searchEmployee(string);

void displayRecords();

};

#endif /\* EMPLOYEEDB\_H\_ \*/

Main.cpp

#include <iostream>

#include <string>

#include "EmployeeDB.h"

using namespace std;

int main() {

EmployeeDB db;

for (int i=0;i<100;i++){

Employee e;

std::string name = "Employee" + to\_string(i);

e.setName(name);

std::cout << "Inserting: " << e.getName() << std::endl;

db.insertEmployee(e);

}

std::cout << "\nDisplay Records" << std::endl;

db.displayRecords();

Employee temp = db.searchEmployee("Employee99");

std::cout << "Deleting: Employee99" << std::endl;

db.deleteEmployee(temp);

std::cout << "\nDisplay Records" << std::endl;

db.displayRecords();

return 0;

}