Data Structures & Algorithms

Coursework 1

Buckinghamshire New University

Felix Saraiva

21713532

2020

Table of Content

Introduction

This study aims to create a useful, interactive and capable data-focused system.

Such study shall implement and explain how several data structures could interact in modelling such a data-driven application.

To apply the best data structures for the appropriate features, the system in the study must be scalable and reliable.

This study will majorly focus on the data structures employed, alternative approaches and performance considerations. Demonstrating such topics throughout detailed design and implementing such design in an Object-Oriented approach system.

Bank Data Management System

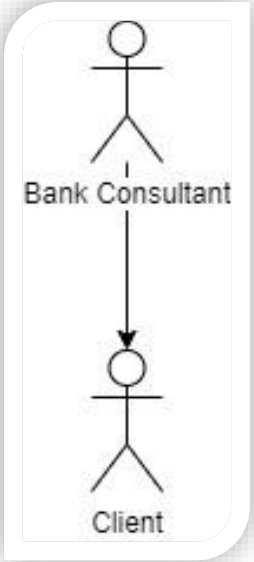
The aim of this study is the creation of a Bank Data Management System.

Such System is to be responsible for the processing of client's data, like Account Numbers, amount of money, and Staff data.

Although the System is directly related to the bank clients, it is not meant for the clients to use, but for the bank consultants. The reason is that such System holds the functionalities of verifying private client data, creating new accounts, removing accounts and analysing the overall bank database system.

Keeping in mind that the System provides a service to the client, a specialised consultant represents the bridge between the client and the service required by the same.

For the consultant to start any operation, he requires the client to provide any form of valid identification or the personal bank card, and if the operation is aimed for a staff member the bank consultant will need personal information about the same.



System Functional Requirements:

ID	Requirement	Specification
FR.1	Menu	The system shall have a Menu that displays the several options and operations.
FR.2	Insert Account	The system shall allow the consultant to insert new accounts.
FR.3	Delete Account	The system shall allow the consultant to delete accounts.
FR.4	Search Account	The system shall have a search feature that confirms the existence of an account and displays the account's information.
FR.5	Account Successor	The system shall be able to find the account that success a specific account in the database.
FR.6	Display Accounts	The system shall be able to organize accounts in order and display them.
FR.7	Database Height	The system shall allow the user to consult the database size.
FR.8	Unlock a blocked Account	The system shall allow the consultant to provide the clients with a temporary Pin that unlocks their account.
FR.9	Display Employees	The system shall display the list of bank Employees.
FR.10	Register New Employee	The system shall allow to Register a new employee.
FR.11	Remove an Employee	The system shall allow to remove an employee.
FR.	Quit	The system shall have a fast way to shut down.

System Non-Functional Requirements:

ID	Requirement	Specification
NFR.1	Scalability	This system shall be able to operate under a large amount of Data.
NFR.2	Reliability	This system's features shall have a very low probability of failure.
NFR.3	Capacity	The Capacity of this system shall not affect its performance.
NFR.4	Data Integrity	This systems data shall be accurate and consistent.
NFR.5	Usability	This system shall be easy and straightforward to its users.

System Implementation

After the analyse and study of the possible data structures that could be applied to this system Binary Search Trees appears to be the best to manage Clients Data, Hash tables are the choice to build an Employees Database, and Queues are the most suitable choice for dealing with high security requests.

1. Vectors:

This system uses vectors instead of arrays because although they consume more memory, they also provide storage management and dynamic growth in a more efficient way. (C Plus Plus, 2000-2020)

2. Binary Search Trees:

The decision of using trees instead of linked lists is because binary trees share the benefits of both sorted arrays and linked lists, as search is as quick as in a sorted array and insertion or deletion are fast as in linked lists. (Tutorials Point, 2020)

This system applies Binary Search trees to the creation, search and deletation of the Client's Accounts. Using this Data structure the Clients Accounts data is sorted using Big O of Log(n), this allows the data structure to use Binary Search which makes it much faster than a normal Tree Structure.

3. Hash Table:

The reason why this system uses hash tables and not the tree structure to build the employees database is because hash tables have a very fast access of data when the index of the desired data is known.

Different from the Clients Database, the bank possesses the information about its own workers, so this makes it possible to use the search performance of hash tables, and its algorithm, O(1) . (Tutorials Point, 2020)

This system uses an array of staff to store the data and an hash function to create the respetive keys, this particular table is built to accept any type of data.

4. Priority Queues:

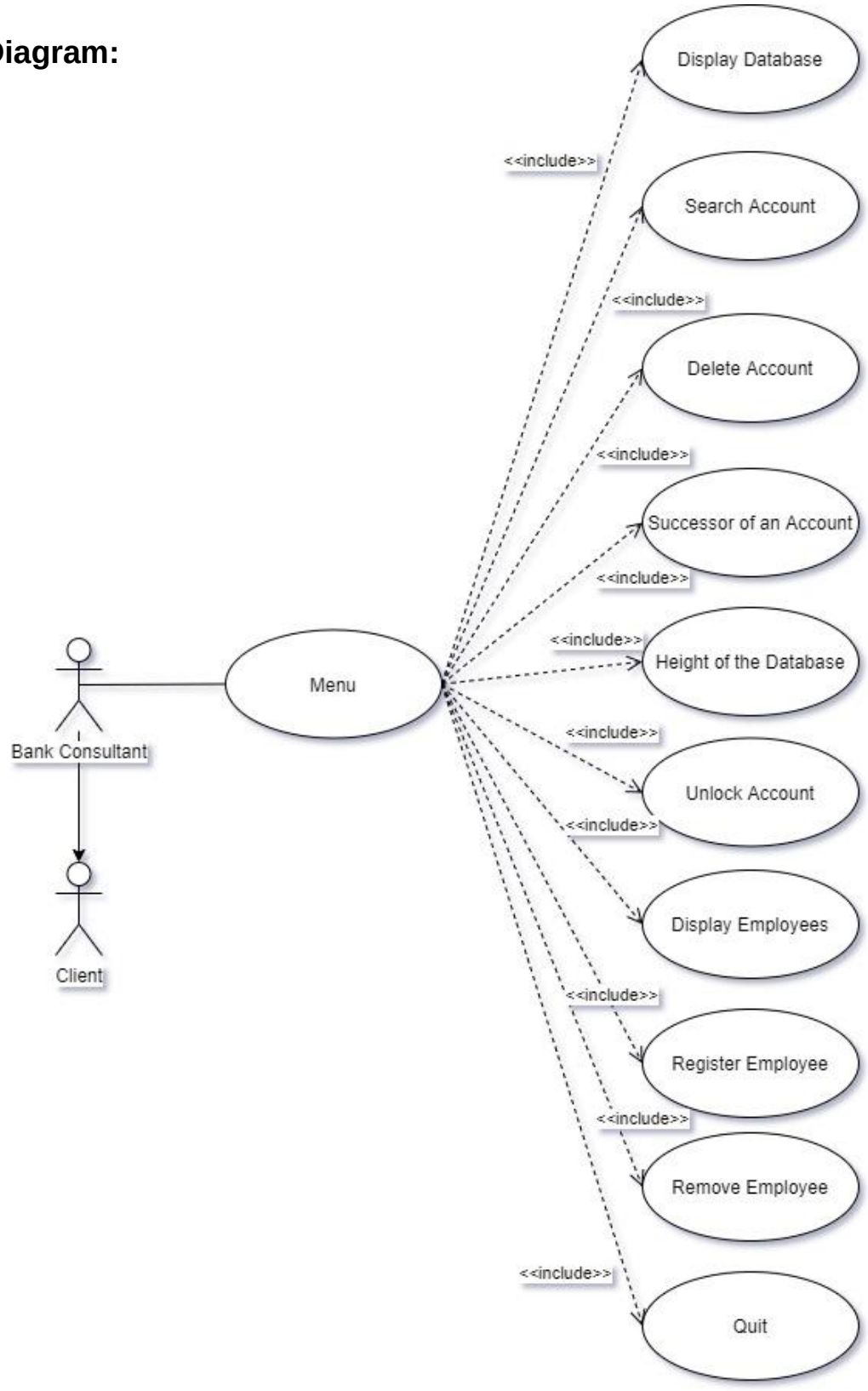
The reason why Queues are applied in this system, is because of its fast performance on removing items. Queues usually have an Insert performance of O(n) and a Remove performance of O(1), the fastest algorithm on Big-O notation. (Geeks for Geeks, 2020)

An account gets blocked if a large number of attempts of using a wrong pin takes place.

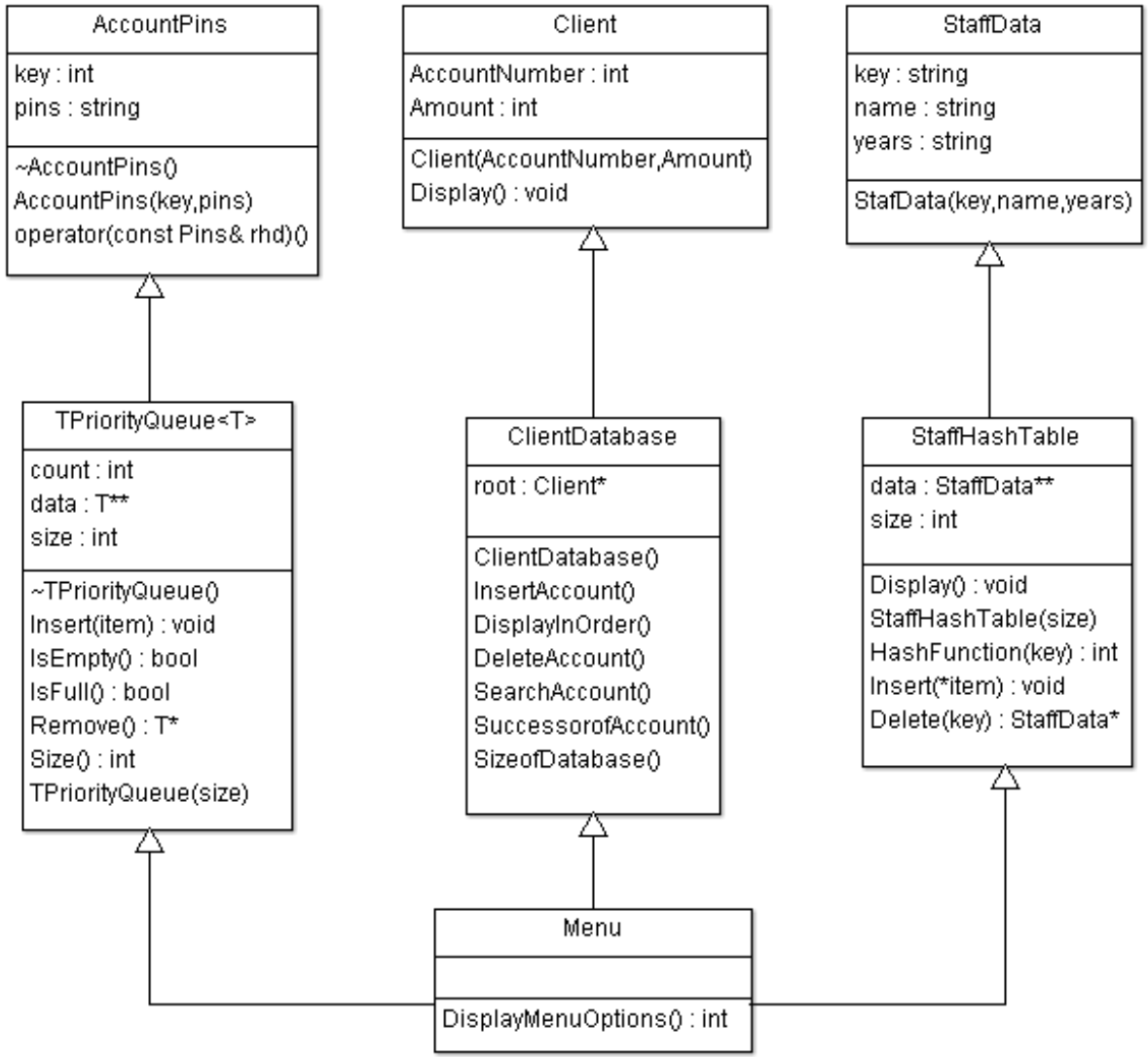
Since these Pins allow the Clients to unlock their accounts, after a pin is provided to a Client, the Pins must be deleted, the fastest possible. This makes Priority Queues the most suitable option.

System Design

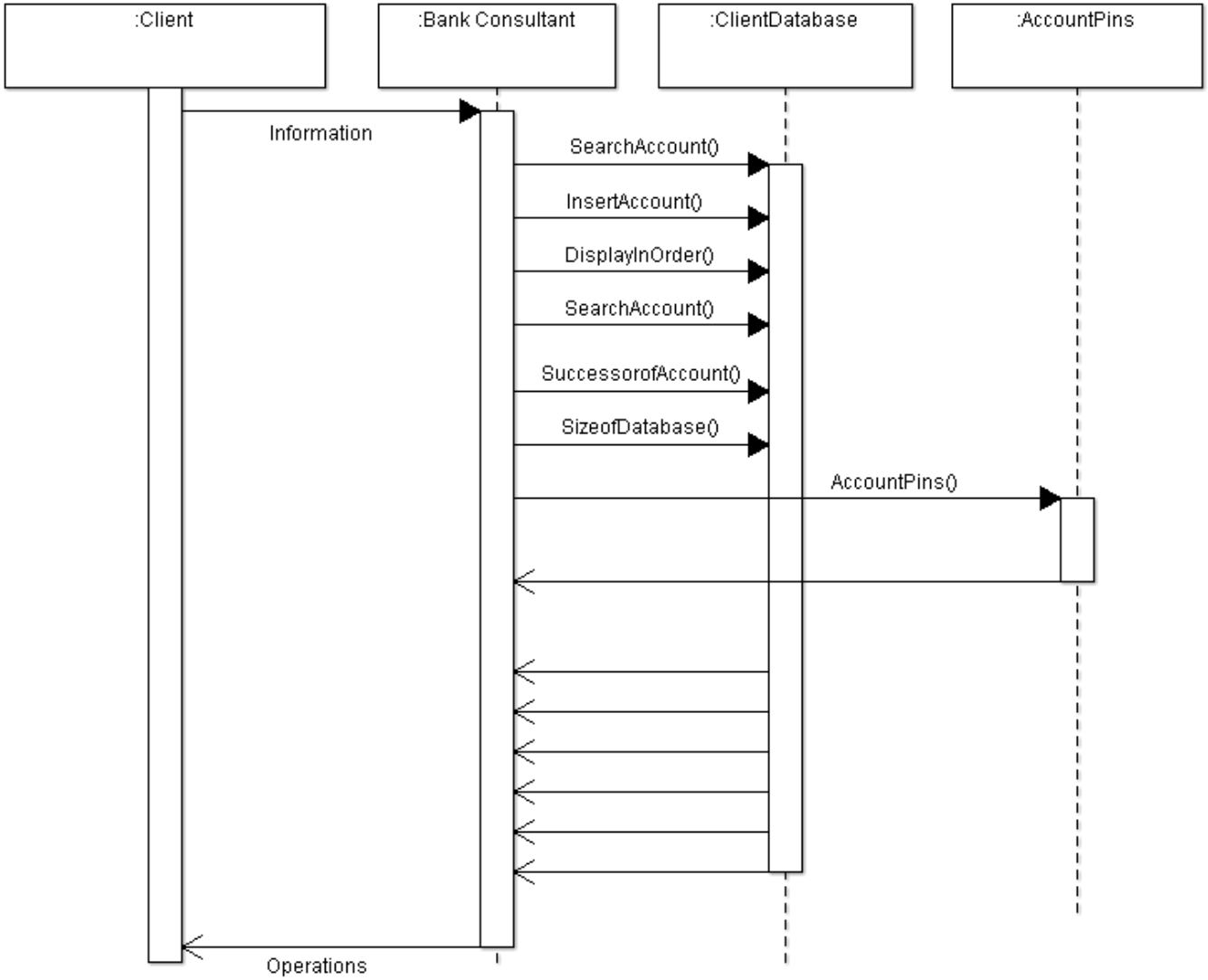
Use Case Diagram:

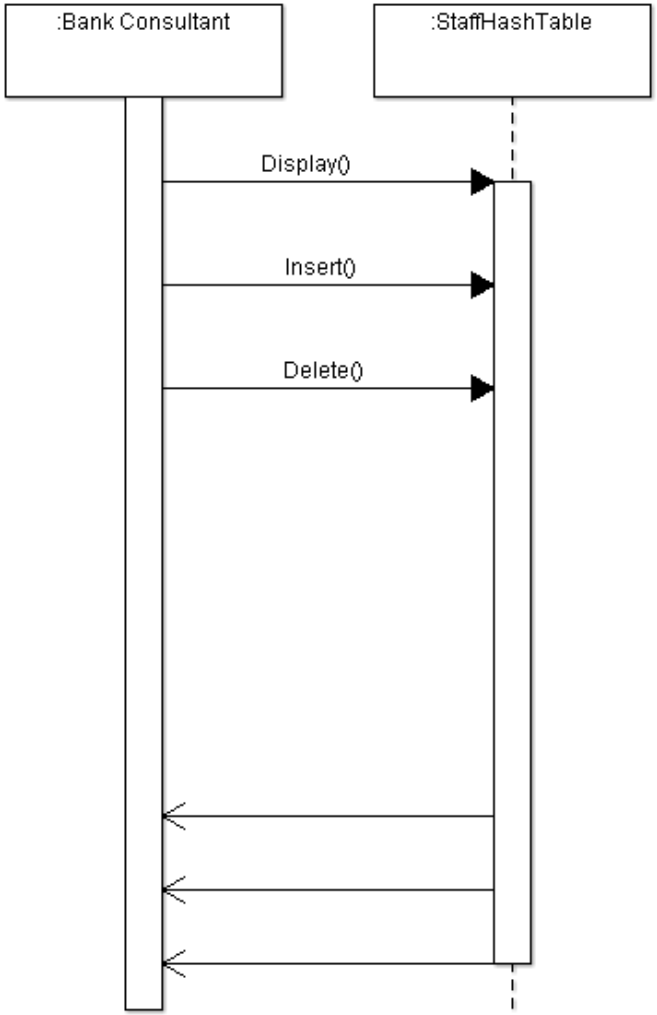


Class Diagram:



Sequence Diagram





Conclusive Discussion

The development of this system accomplished all the objectives defined in this study.

All the implemented databases work successfully and with a fast approach, completing the defined operations.

The Non-Functional Requirements were achieved by having in consideration the potential growth of such system and respecting the rules of Object-Oriented development.

Tests were made regarding the processing speed and complexity of the system.

In the Tree Data structure, more than 1000000 accounts were created automatically without losing significant performance. The other data structures suffered similar tests, ending in great success, where the Priority Queue with its Delete() functionality ended up being the fastest, confirming a favourable choice of using it in a security-focused feature.

Although this final discussion reflects a very positive outcome of both the development and design of this system, there is always space for improvement.

With a more critical point of view, regarding the choice of using Tree Structure for the Clients database and Hash Tables for the Employees, the conclusion is that the Data Structures should have been switched. In other words, during the test phase, the Hash Table performance of search and insertion did not change regarding on the size of the database, and since in any data-focused business the number of clients is always more significant than the number of employees, the specific performance of Hash tables would have been more useful on the creation of the Clients Database.

Concluding, the system’s performance is very positive, but if applied in a real-world environment it would require some adjustments.

References

C Plus Plus. (2000-2020). *Priority Queue*. Retrieved from C Plus Plus: http://www.cplusplus.com/reference/queue/priority_queue/

C Plus Plus. (2000-2020). *vector*. Retrieved from C Plus Plus: http://www.cplusplus.com/reference/vector/vector/

Cooervo. (2020). *Big-O Notation*. Retrieved from Data Structures & Algorithms : https://cooervo.github.io/Algorithms-DataStructures-BigONotation/index.html

Geeks for Geeks. (2020). *Applications of Priority Queue*. Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/applications-priority-queue/

McDonnell, M. (2019, Jan 30). *Integralist*. Retrieved from Data Types and Data Structures: https://www.integralist.co.uk/posts/data-types-and-data-structures/#data-types

MIT. (2020). *Big O Notation*. Retrieved from MIT Edu: https://web.mit.edu/16.070/www/lecture/big_o.pdf

mycodeschool. (2014, Jan 25). *Data structures: Binary Search Tree*. Retrieved from YouTube: https://www.youtube.com/watch?v=pYT9F8_LFTM&list=PL2_aWCzGMAwI3W_JlcBbtYTwiQSsOTa6P&index=27

mycodeschool. (2014, Jan 25). *Data structures: Binary Search Tree*. Retrieved from YouTube: https://www.youtube.com/watch?v=pYT9F8_LFTM&t=53s

Software Testing Help. (2020, April 16). *Priority Queue Data Structure In C++ With Illustration*. Retrieved from Software Testing Help: https://www.softwaretestinghelp.com/priority-queue-in-cpp/

Study Tonight. (2020). *Introduction to Data Structures and Algorithms*. Retrieved from Study Tonight: https://www.studytonight.com/data-structures/introduction-to-data-structures

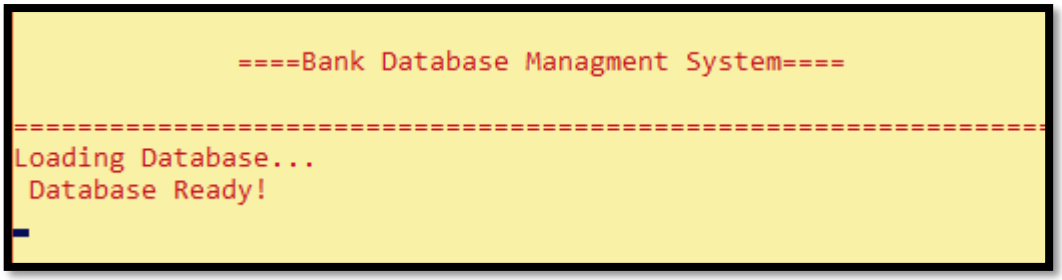
Tutorials Point. (2020). *Data Structure and Algorithms*. Retrieved from Tutorials Point: https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm

Tutorials Point. (2020). *Data Structure and Algorithms - Hash Table*. Retrieved from Tutorials Point: https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm

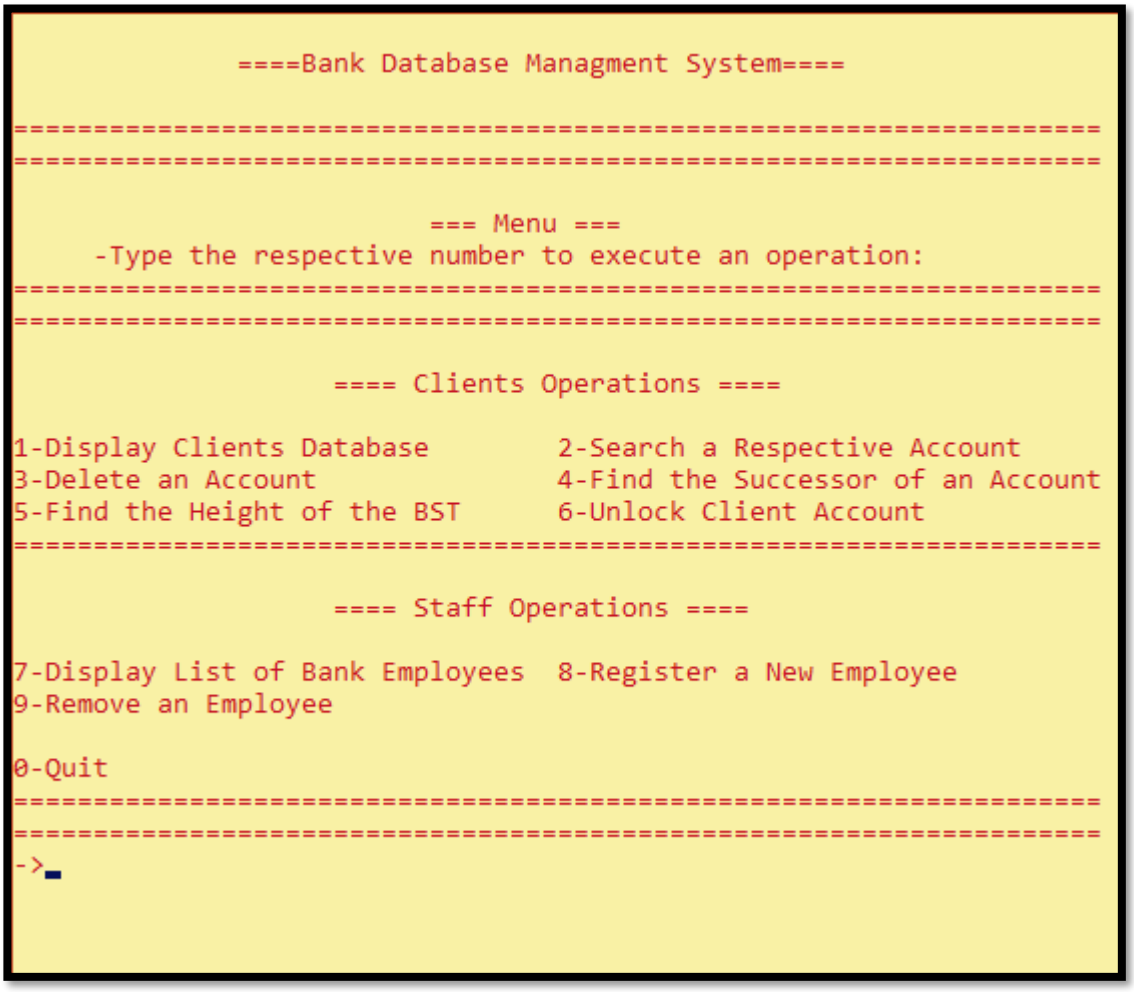
Tutorials Point. (2020). *Data Structure and Algorithms - Tree*. Retrieved from Tutorials Point: https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm

Functionality Screenshots

Loading Database:



Main Menu:



Display Clients Database:

```
========================================================================
========================================================================
->1

      ==== Client Database ====
-Account Number: 2530 -Amount: 21490 $
-Account Number: 4665 -Amount: 28561 $
-Account Number: 5998 -Amount: 31532 $
-Account Number: 6097 -Amount: 16948 $
-Account Number: 6830 -Amount: 21114 $
-Account Number: 8120 -Amount: 28061 $
-Account Number: 9139 -Amount: 26005 $
-Account Number: 9502 -Amount: 18305 $
-Account Number: 9772 -Amount: 6960 $
-Account Number: 9773 -Amount: 30861 $
-Account Number: 10846 -Amount: 12928 $
-Account Number: 13442 -Amount: 3360 $
-Account Number: 16970 -Amount: 30928 $
-Account Number: 18511 -Amount: 16465 $
-Account Number: 21195 -Amount: 4238 $
-Account Number: 25435 -Amount: 904 $
-Account Number: 27639 -Amount: 4856 $
-Account Number: 28738 -Amount: 13280 $
-Account Number: 29510 -Amount: 18155 $
-Account Number: 30893 -Amount: 26047 $


========================================================================
```

Search an Account:

```
====================================

=Request:
-Search Account:

Enter the Account Number to be searched:2530
Valid Account!
====================================
Account Number: 2530
Amount: 21490$
========================================================================
```

Delete Account:

```
=====================================================================
->3
====================================

=Request:
-Delete Account:
Enter the Account Number to be deleted:2530
->Deleating Account: 2530
->Deleating Account: 2530
Account Deleted!


=====================================================================
```

Find the Successor:

```
====================================

=Request:
-Account Successor:
Enter the Account Number to find respective Sucessor:4665

=Request:
Getting the Successor of Account Number: 4665
...

The successor is the Account Number: 21195
=====================================================================
```

Find the Height of the BST:

```
=====================================================================
->5
====================================

=Request:
-Height of the Database Binary Tree
The Height of the Database Binary Tree root is: 5

=====================================================================
```

Unlock an Account:

```
=====================================================================
->6
====================================

=Request:
-Clients Account Re-Validation:

WARNING:
!! For security reasons, the next Operation will delete Pins Database!!

Client's Name: Arnold Gates     Client Temporary Pin: 245
Client's Name: Jon Smith        Client Temporary Pin: 613
Client's Name: Steve Roggers    Client Temporary Pin: 1309
Client's Name: Tony Stark       Client Temporary Pin: 5325
Client's Name: Ronda Jobs       Client Temporary Pin: 6761
Client's Name: Buggy Barns      Client Temporary Pin: 8757
=====================================================================
```

Display Staff Database:

```
=====================================================================
->7

                          ==== Staff Database ====

Staff Member ID: 45218392       Employee Name: Edward Teach     Years of Service: 15
Staff Member ID: 14587227       Employee Name: James Flint      Years of Service: 13
Staff Member ID: 12458755       Employee Name: Anne Bonny       Years of Service: 11
Staff Member ID: 15248756       Employee Name: William Kidd     Years of Service: 2
Staff Member ID: 12566388       Employee Name: Charles Vane     Years of Service: 20


=====================================================================
```

New Employee Registration:

```
=====================================================================
->8
====================================

=Request:

      ==== New Employee Registration ====

Enter the New Employeer Name:Peter Parker
Enter the New Employeer ID:21713535

New Employee Added!
=====================================================================
```

Firing an Employee:

```
=====================================================================
->9
====================================

=Request:

      ==== Firing an Employee ====

Enter Employeer ID:15248756
====================================
Employee Deleted!
=====================================================================
```

Result:

```
=====================================================================
->7

                          ==== Staff Database ====

Staff Member ID: 45218392       Employee Name: Edward Teach     Years of Service: 15
Staff Member ID: 14587227       Employee Name: James Flint      Years of Service: 13
Staff Member ID: 12458755       Employee Name: Anne Bonny       Years of Service: 11
Staff Member ID: 12566388       Employee Name: Charles Vane     Years of Service: 20
Staff Member ID: 1713535        Employee Name: Peter Parker     Years of Service: 0


=====================================================================
```

Quit:

```
=====================================================================
->0
Thank you for your Service!
Press any key to continue . . .
```

C++ Files

BankManagmentDataSystem.cpp

```
// BankManagmentDataSystem.cpp : This file contains the 'main' function. Program execution begins
and ends there.
//

#include"Menu.h"

using namespace std;

int main()
{
	system("color e4");

	Menu MenuClass;
	MenuClass.DisplayMenuOptions();
}
```

Menu.h

```
#pragma once
#include"Client.h"
#include"ClientTree.h"
#include <string>
#include <iostream>
#include <windows.h>
#include <ctime>
#include <cstdlib>
#include "Pins.h"
#include "TPriorityQueue.h"
#include"StaffInfoStr.h"
#include "EmployeesHashTable.h"

class Menu
{
public:
	int DisplayMenuOptions();
private:
};
```

Menu.cpp

```
#include "Menu.h"

using namespace std;

int Menu::DisplayMenuOptions()
{
	cout << "\n              ====Bank Database Managment System====\n" << endl;
	cout << "=====================================================================" << endl;
	int x = 1;
	srand(time(0));
	cout << "Loading Database..." << endl;

	////////////////////////////////////////////////////////////////////////////////
	//Creation of the Bank Databases
	ClientTree* BankTree = new ClientTree();//Tree Clients Accounts
	TPriorityQueue<Pins>* templateIDs = new TPriorityQueue<Pins>(20);//Queue Pins
	EmployeesHashTable* StaffTable = new EmployeesHashTable(16);//Hash Table Staff

	//Employees Database
	StaffTable->Insert(new StaffInfoStr("15248756", "William Kidd","2"));
	StaffTable->Insert(new StaffInfoStr("45218392", "Edward Teach","15"));
	StaffTable->Insert(new StaffInfoStr("12566388", "Charles Vane","20"));
	StaffTable->Insert(new StaffInfoStr("12458755", "Anne Bonny","11"));
	StaffTable->Insert(new StaffInfoStr("14587227", "James Flint","13"));

	//Creation of Accounts Database
	for (int i = 0; i < 20; i++)
	{
		BankTree->InsertAccount(rand(), rand());
	}

	//Pins Database
	templateIDs->Insert(new Pins(rand() % 10000, "Ronda Jobs"));
	templateIDs->Insert(new Pins(rand() % 10000, "Jon Smith"));
	templateIDs->Insert(new Pins(rand() % 10000, "Arnold Gates"));
	templateIDs->Insert(new Pins(rand() % 10000, "Tony Stark"));
	templateIDs->Insert(new Pins(rand() % 10000, "Steve Roggers"));
	templateIDs->Insert(new Pins(rand() % 10000, "Buggy Barns"));

	cout << " Database Ready!" << endl;
	Sleep(2000);
	system("cls");
	////////////////////////////////////////////////////////////////////////////////
	cout << "\n              ====Bank Database Managment System====\n" << endl;
	cout << "=====================================================================" << endl;
	while (x > 0)
	{
		//Menu Interface
		cout << "=====================================================================\n" <<
endl;
		cout << "                          === Menu ===" << endl;
		cout << "     -Type the respective number to execute an operation:" << endl;
		cout << "=====================================================================" <<
endl;
		cout << "=====================================================================\n" <<
endl;
		cout << "                    ==== Clients Operations ====\n" << endl;
		cout << "1-Display Clients Database        2-Search a Respective Account " << endl;
		cout << "3-Delete an Account               4-Find the Successor of an Account" <<
endl;
```

```
		cout << "5-Find the Height of the BST	6-Unlock Client Account"<< endl;
		cout << "=====================================================================\n" <<
endl;
		cout << "		==== Staff Operations ====\n" << endl;
		cout << "7-Display List of Bank Employees  8-Register a New Employee" << endl;
		cout << "9-Remove an Employee		" << endl;
		cout << "\n0-Quit" << endl;
		cout << "=====================================================================" <<
endl;
		cout << "=====================================================================" <<
endl;
		cout << "->";
		cin >> x;

	////////////////////////////////////////////////////////////////////////////////////////
///////
		switch (x)
		{
		//Quit
		case 0:
			cout << "Thank you for your Service!" << endl;
		break;

	////////////////////////////////////////////////////////////////////////////////////////
///////
		//Display the Accounts Database (Tree DS)
		case 1:
		{
			BankTree->IsBst(BankTree->root, INT_MIN, INT_MAX);//Cheks if BST Structure is
being respected

			cout << "\n	==== Client Database ====" << endl;
			BankTree->DisplayInOrder(BankTree->root);
			cout << "\n" << endl;
		}break;

	////////////////////////////////////////////////////////////////////////////////////////
///////
		//Search an Account (Tree DS)
		case 2:
		{
			int number;
			cout << "====================================" << endl;
			cout << "\n=Request:" << endl;
			cout << "-Search Account:\n" << endl;

			cout << "Enter the Account Number to be searched:";
			cin >> number;
			//If the Account is found, print "EXISTS", the account number and the account
amound
			if (BankTree->Search(BankTree->root, number) == true) {
				Client* found = BankTree->Find(number);
				cout << "Valid Account!\n";
				cout << "====================================" << endl;
				cout<< "Account Number: " << found->AccountNumber << endl;
				cout << "Amount: " << found->Amount << "$"<< endl;
			}
			else cout << "Not Found\n";
		}break;

	////////////////////////////////////////////////////////////////////////////////////////
///////
```

```
		//Delete an Account (Tree DS)
		case 3:
		{
			int number;
			cout << "====================================" << endl;
			cout << "\n=Request:" << endl;
			cout << "-Delete Account:" << endl;

			cout << "Enter the Account Number to be deleted:";
			cin >> number;
			BankTree->root = BankTree->DeleteAccount(BankTree->root, number);
			Sleep(3000);
			cout << "Account Deleted!\n";
			cout << "\n" << endl;
		}break;

	////////////////////////////////////////////////////////////////////////////////////////////////
///////
		//Find the Successor of an account in the tree Structure (Tree DS)
		case 4:
		{
			cout << "====================================" << endl;
			cout << "\n=Request:" << endl;
			cout << "-Account Successor:" << endl;

			int number;
			cout << "Enter the Account Number to find respective Sucessor:";
			cin >> number;
			cout << "\n=Request:" << endl;
			struct Client* successor = BankTree->Getsuccessor(BankTree->root, number);
			if (successor == NULL) cout << "No successor Found\n";
			else
				cout << "The successor is the Account Number: " << successor-
>AccountNumber << "\n";
		}break;

	////////////////////////////////////////////////////////////////////////////////////////////////
///////
		//Find the Height of the Root of this Binary tree database Structure (Tree DS)
		case 5:
		{
			cout << "====================================" << endl;
			cout << "\n=Request:" << endl;
			cout << "-Height of the Database Binary Tree" << endl;

			cout << "The Height of the Database Binary Tree root is: ";
			cout << BankTree->FindHeight(BankTree->root);
			cout << "\n\n";
		}break;

	////////////////////////////////////////////////////////////////////////////////////////////////
///////
		//Unlocking an Account (Queue DS)
		case 6:
		{
			cout << "====================================" << endl;
			cout << "\n=Request:" << endl;
			cout << "-Clients Account Re-Validation:\n" << endl;

			cout << "WARNING: \n!! For security reasons, the next Operation will delete
Pins Database!!\n" << endl;
```

```
                    while (!templateIDs->IsEmpty()) {
                           Pins* obj = templateIDs->Remove();
                           cout << "Client's Name: " << obj->name << "\tClient Temporary Pin: " <<
obj->key  << endl;
                    }
             }break;

      ////////////////////////////////////////////////////////////////////////////////////////////
///////
             //Displat the Staff Database (Hash Table DS)
             case 7:
             {
                    cout << "\n                              ==== Staff Database ====\n" << endl;
                    StaffTable->Display();
                    cout << "\n" << endl;

             }break;

      ////////////////////////////////////////////////////////////////////////////////////////////
///////
             //Registration of a New Employee (Hash Table DS)
             case 8:
             {
                    string IDE;
                    string newE;
                    string Eyear = "0";

                    cout << "====================================" << endl;
                    cout << "\n=Request:" << endl;
                           cout << "\n      ==== New Employee Registration ====\n" << endl;

                           if (Eyear=="0")
                           {
                                  cout << "Enter the New Employeer Name:";
                                  cin.get();
                                  getline(cin, newE);

                                  cout << "Enter the New Employeer ID:";
                                  cin.get();
                                  getline(cin, IDE);
                           }

                           StaffTable->Insert(new StaffInfoStr(IDE, newE, Eyear));
                           cout << "\nNew Employee Added!\n";
             }break;

      ////////////////////////////////////////////////////////////////////////////////////////////
///////
             //Delete of a Staff Member (Hash Table DS)
             case 9:
             {
                    string ID;

                    cout << "====================================" << endl;
                    cout << "\n=Request:" << endl;
                    cout << "\n      ==== Firing an Employee ====\n" << endl;

                    cout << "Enter Employeer ID:";
                    cin >> ID;
                    StaffTable->Delete(ID);
                    cout << "====================================" << endl;
                    cout << "Employee Deleted!\n";
```

```
			}break;

		////////////////////////////////////////////////////////////////////////////////////////////////
///////
			default:
				cout << "Incorrect Command!!" << endl;
			}
		}
		system("pause");
		return 0;
}
```

Client.h

```
#pragma once
#include <iostream>

using namespace std;

class Client
{
public:
	int AccountNumber;
	int Amount;
	Client* leftChild;
	Client* rightChild;

	Client(int AccountNumber, int Amount);
	void Display();
	bool LessThan(Client* c1, Client* c2);
private:
};
```

Client.cpp

```
#include "Client.h"

Client::Client(int AccountNumber, int Amount)
{
	this->AccountNumber = AccountNumber;
	this->Amount = Amount;

	leftChild = 0;
	rightChild = 0;
}

void Client::Display()
{
	cout << "-Account Number: " << this->AccountNumber << " -Amount: " << this->Amount <<" $"<<
endl;
}

bool Client::LessThan(Client* c1, Client* c2)
{
	if (c1->AccountNumber< c2->AccountNumber)
		return true;
	else
		return false;
}
```

ClientTree.h

```
#pragma once
#include "Client.h"
#include <iostream>

using namespace std;

class ClientTree
{
public:
	Client* root;

	ClientTree();
	Client* Find(Client* key);
	Client* Find(int key);
	Client* FindMin(Client* root);
	struct Client* DeleteAccount(struct Client* root, int AccountNumber);
	struct Client* Getsuccessor(Client* root, int AccountNumber);
	void DisplayInOrder(Client* localRoot);
	void InsertAccount(int AccountNumber, int Amount);
	bool Search(Client* root, int AccountNumber);
	int FindHeight(struct Client* root);
	bool IsBst(Client* root, int minValue, int maxValue);
private:
};
```

ClientTree.cpp

```
#include "ClientTree.h"
#include <windows.h>

ClientTree::ClientTree()
{
	root = 0;
}

Client* ClientTree::Find(Client* key)
{
	Client* current = root;
	while (current != key) {
		if (key < current)
			current = current->leftChild;
		else
			current = current->rightChild;
		if (current == 0)
			return 0;
	}
	return current;
}

//Function Overloading- In order to display Account details
Client* ClientTree::Find(int key)
{
	Client* current = root;
	while (current->AccountNumber != key) {
		if (key < current->AccountNumber)
			current = current->leftChild;
		else
			current = current->rightChild;
		if (current == 0)
			return 0;
	}
	return current;
}

void ClientTree::DisplayInOrder(Client* localRoot)
{
	if (localRoot != 0) {
		DisplayInOrder(localRoot->leftChild);
		localRoot->Display();
		DisplayInOrder(localRoot->rightChild);
	}
}

void ClientTree::InsertAccount(int AccountNumber, int Amount)
{
	Client* newNode = new Client(AccountNumber, Amount);
	if (root == 0)
		root = newNode;
	else {
		Client* current = root;
		Client* parent;
		while (true) {
			parent = current;
			if (newNode->LessThan(newNode, current)) {
				current = current->leftChild;
				if (current == 0) {
					parent->leftChild = newNode;
```

```
                                        return;
                                }
                        }
                        else {
                                current = current->rightChild;
                                if (current == 0) {
                                        parent->rightChild = newNode;
                                        return;
                                }
                        }
                }
        }
}

Client* ClientTree::FindMin(Client* root)
{
        while (root->leftChild != NULL) root = root->leftChild;
        return root;
}

Client* ClientTree::DeleteAccount(Client* root, int AccountNumber)
{
        cout << "->Deleating Account: " << AccountNumber << endl;
        if (root == NULL) return root;
        else if (AccountNumber < root->AccountNumber) root->leftChild = DeleteAccount(root-
>leftChild, AccountNumber);
        else if (AccountNumber > root->AccountNumber) root->rightChild = DeleteAccount(root-
>rightChild, AccountNumber);
        // Account Found-Ready to be deleted

        else {

                // 1st Case:  No child
                if (root->leftChild == NULL && root->rightChild == NULL) {
                        delete root;
                        root = NULL;
                }

                // 2nd Case: One child
                else if (root->leftChild == NULL) {
                        struct Client* temp = root;
                        root = root->rightChild;
                        delete temp;
                }
                else if (root->rightChild == NULL) {
                        struct Client* temp = root;
                        root = root->leftChild;
                        delete temp;
                }

                // 3rd Case: 2 children
                else {
                        struct Client* temp = FindMin(root->rightChild);
                        root->AccountNumber = temp->AccountNumber;
                        root->rightChild = DeleteAccount(root->rightChild, temp->AccountNumber);
                }
        }
        return root;
}

Client* ClientTree::Getsuccessor(Client* root, int AccountNumber)
{
```

```
	// Search the Node - O(n)

	struct Client* current = Find(root);
	if (current == NULL) return NULL;
	cout << "Getting the Successor of Account Number: " << AccountNumber << endl;
	cout << "...\n" << endl;
	Sleep(3000);
	if (current->rightChild != NULL) {  //Case 1: Node has right subtree
		return FindMin(current->rightChild); // O(n)
	}
	else {   //Case 2: No right subtree  - O(n)
		struct Client* successor = NULL;
		struct Client* ancestor = root;
		while (ancestor != current) {
			if (current->AccountNumber < ancestor->AccountNumber) {
				successor = ancestor; // so far this is the deepest node for which
current node is in left
				ancestor = ancestor->leftChild;
			}
			else
				ancestor = ancestor->rightChild;
		}
		return successor;
	}
}

bool ClientTree::Search(Client* root, int AccountNumber)
{
	if (root == NULL) {
		return false;
	}
	else if (root->AccountNumber == AccountNumber) {
		return true;

	}
	else if (AccountNumber <= root->AccountNumber) {
		return Search(root->leftChild, AccountNumber);
	}
	else {
		return Search(root->rightChild, AccountNumber);
	}

}

int ClientTree::FindHeight(Client* root)
{
	if (root == NULL) {
		return-1;
	}
	return max(FindHeight(root->leftChild), FindHeight(root->rightChild)) + 1;
}

bool ClientTree::IsBst(Client* root, int minValue , int maxValue)
{
	if (root == NULL)
	{
		return true;
	}
	if (root->AccountNumber > minValue && root->AccountNumber < maxValue && IsBst(root-
>leftChild, minValue ,root->AccountNumber) && IsBst(root->rightChild,root->AccountNumber ,maxValue))
	{
		return true;
```

```
        }
        else {
                cout << "Binary Search Tree Broken!";
                return false; }
}
```

Pins.h

```
#pragma once

#include <iostream>

using namespace std;

class Pins
{
public:
	int key;
	string name;
	Pins(int key, string name) :key(key), name(name) {}
	~Pins() {}

	bool operator>(const Pins& rhd) {
		return this->key > rhd.key;

	}

private:
};
```

TPriorityQueue.h

```
#pragma once
template <class T>
class TPriorityQueue {
	int size;
	T** data;
	int count;
public:
	TPriorityQueue(int size);
	~TPriorityQueue();
	void Insert(T* item);

	T* Remove();
	bool IsEmpty();
	bool IsFull();
	int Size();
};


template <class T>
TPriorityQueue<T>::TPriorityQueue(int size) :size(size) {
	data = new T * [size];
	count = 0;
}

template <class T>
TPriorityQueue<T>::~TPriorityQueue() {
}


template <class T>
void TPriorityQueue<T>::Insert(T* item) {
	// Queue is empty
	if (count == 0) {
		data[count++] = item;
	}
	else {
		int j;
		// Start at the end work backwards
		for (j = (count - 1); j >= 0; j--) {
			// If the new item is larger than current
			if (*item > * data[j]) {
				// Shift the current up
				data[j + 1] = data[j];
			}
			else {
				// Finished shifting
				break;
			}
		}

		data[j + 1] = item;
		count++;
	}
}
```

```
template <class T>
T* TPriorityQueue<T>::Remove() {
	return data[--count];
}

template <class T>
bool TPriorityQueue<T>::IsEmpty() {
	return count == 0;
}

template <class T>
bool TPriorityQueue<T>::IsFull() {
	return count == size;
}

template <class T>
int TPriorityQueue<T>::Size() {
	return size;
}
```

StaffInfoStr.h

```
#pragma once
#include <iostream>

using namespace std;

class StaffInfoStr
{
public:
	string key;
	string name;
	string years;

	StaffInfoStr(string key, string name,string years);
};
```

StaffInfoStr.cpp

```
#include "StaffInfoStr.h"

StaffInfoStr::StaffInfoStr(string key, string name, string years)
{
	this->key = key;
	this->name = name;
	this->years = years;
}
```

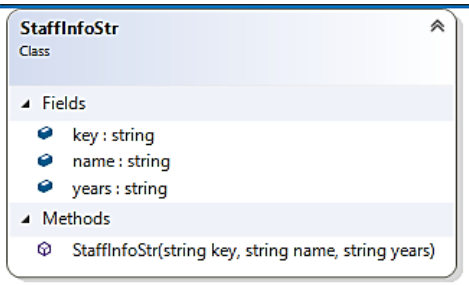
EmployeesHashTable.h

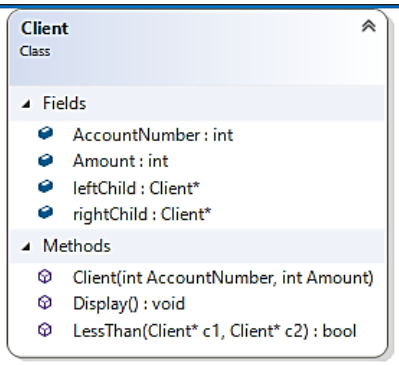
```
#pragma once
#include <iostream>
#include <string>
#include "StaffInfoStr.h"

using namespace std;

class EmployeesHashTable
{
public:

	int size;
	StaffInfoStr** data;

	EmployeesHashTable(int size);
	void Display();
	int HashFunction(string key);
	void Insert(StaffInfoStr* item);
	StaffInfoStr* Delete(string key);
private:
};
```

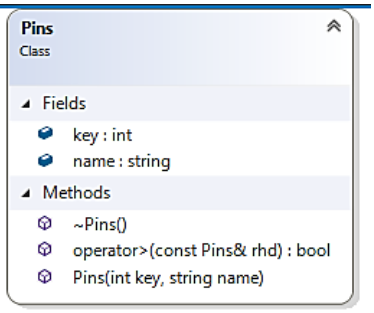
EmployeesHashTable.cpp

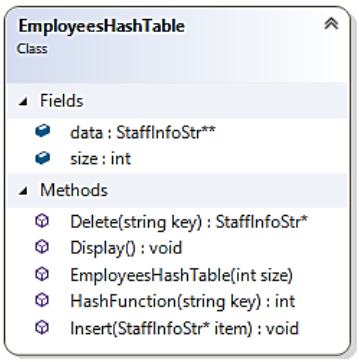
```
#include "EmployeesHashTable.h"

EmployeesHashTable::EmployeesHashTable(int size)
{
	this->size = size;
	data = new StaffInfoStr * [size]; //Array of Staff Info pointers
	for (int i = 0; i < size; i++)
		data[i] = NULL;
}

void EmployeesHashTable::Display()
{
	for (int i = 0; i < size; i++)
		if (data[i] != NULL)
			cout << "Staff Member ID: " << data[i]->key << "\tEmployee Name: " << data[i]-
>name <<"\tYears of Service: "<<data[i]->years<< endl;
}

int EmployeesHashTable::HashFunction(string key)
{
	int ASCIItot = 0;
	for (int i = 0; i < key.length(); i++) {
		ASCIItot += (int)key[i];
	}
	return ASCIItot % size;
}

void EmployeesHashTable::Insert(StaffInfoStr* item)
{
	string key = item->key;
	int hash = HashFunction(key);
	while (data[hash] != NULL) {
		++hash;
		hash %= size;
	}
	data[hash] = item;
}

StaffInfoStr* EmployeesHashTable::Delete(string key)
{
	int hash = HashFunction(key);
	while (data[hash] != NULL) {
		if (data[hash]->key == key) {
			StaffInfoStr* itemToDelete = data[hash];
			data[hash] = NULL;
			return itemToDelete;
		}
		hash++;
		hash = hash % size;
	}
	return NULL;
}
```

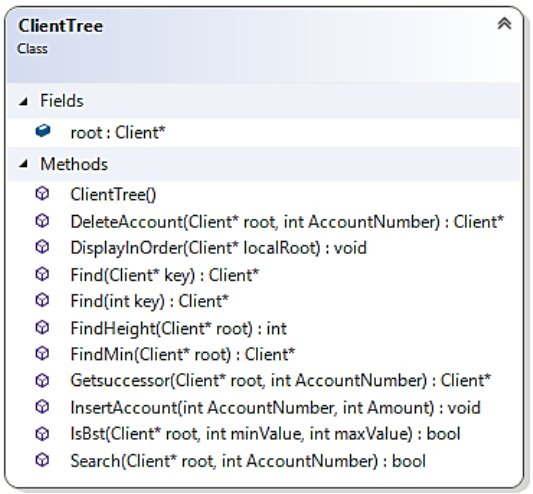
Visual Studio Class Diagram:

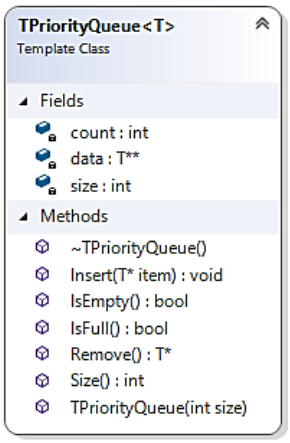


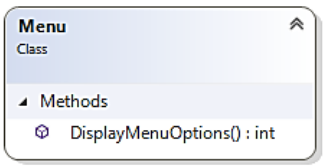












Software Artefact:

GitHub Link : https://github.com/Felix-Saraiva/BankManagmentDataSystem

School of Business, Law and Computing
Academic Year 2019 to 2020

Module Code and Title:	**CO658** Data Structures & Algorithms	**Module Leader:**	**Guy Walker**
Assignment No. and Type:	CW 1	**Assessment Weighting:**	100%
Submission Date:	29/05/2020	**Target Feedback Date:**	**+3 Weeks**
Student Name:	Felix A. Monteiro Saraiva	**Student ID:**	**21713532**