2021

DATA STRUCTURES MTE-REPORT

SUBMITTED TO-
DR. RITU AGARWAL

Innovative Project Report

On

Phonebook Management System using BST

Submitted in complete fulfillment of the requirements for the mid-term examination evaluation in

[Data Structures]

Submitted by

Hardik Arora

2K20/IT/52

and

Lakshay Chandra

2K20/IT/79

Under the Supervision

of

Dr. Ritu Agarwal

Department of Information Technology

Delhi Technological University

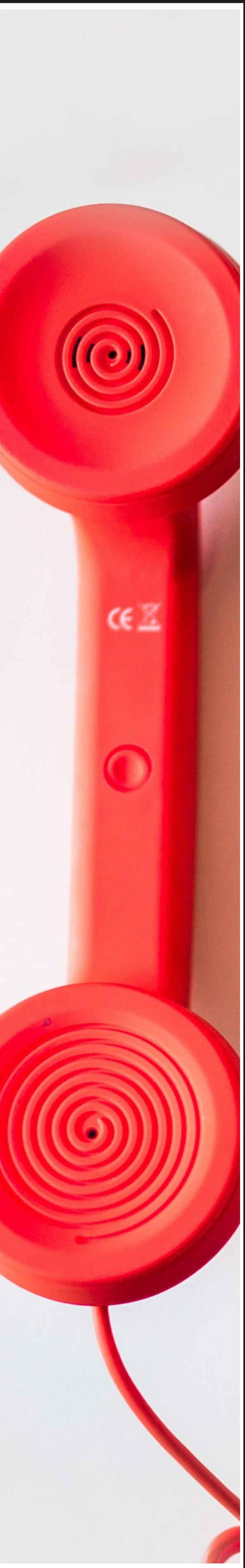
Bawana Road, New Delhi- 110042

INTRODUCTION TO PROBLEM

A telephone directory (phone book) is a listing of telephone subscribers in a geographical area provided by the organization that publishes the directory. It consists of the name as well as the telephone number of people added as a contact. Name and telephone number are displayed in alphabetical order.

In today’s technological era, customers need more convenient ways to obtain their required results in the proper time. When it comes to finding any particular record, the old-fashioned manual book is pretty inconvenient. Even the publishers have to print the new records in their new volume that are published each year. The searching process of a particular record is a time-consuming process for the customers as they are required to navigate through the index page to the details in alphabetical order following various pages to get their desired records.

The objective of this project is to design an Electronic Telephone Directory console-based application in C++ using a Binary Search tree as an internal data structure. The application is capable of saving the name and contact number of desired persons and performing specific operations.



In order to keep the phone book directory updated, the admin will have the authority to delete as well as modify the existing records. The users of the directory will only have the authority to add contacts, search for any particular record, and list details of all available records. Admin will have the authority to perform various operations such as add customer records, search any particular record, delete records, etc. To provide the search result within a short interval of time optimized search algorithm code has been used which will be able to provide the results in minimum time.

The console application contains the Menu describing the following functions:

- Adding a contact
- Loading a text file
- Searching for a contact
- Filtering a contact (Displaying all contacts before a specific contact in sorted order)
- Deleting a contact
- Updating a contact

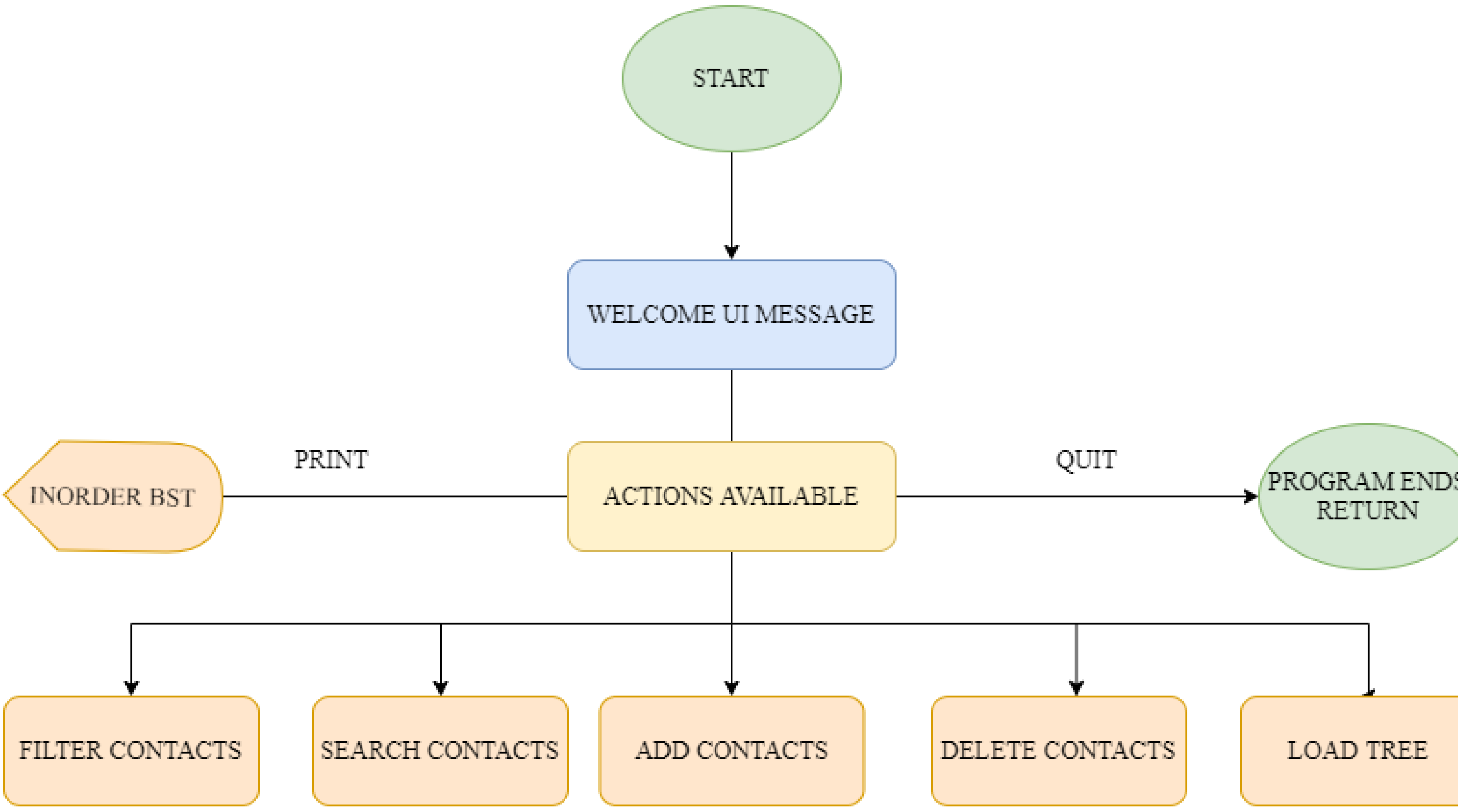
The concept of file management, data structures, and object orientation are used extensively throughout the application.

When the admin adds a contact to the phone book, he will be asked for personal information: Full Name, and phone number. Every full name has a first name and a last name separated by a space character. The fields are separated by the newline character “\n”. The entries will be validated before performing any further operations.

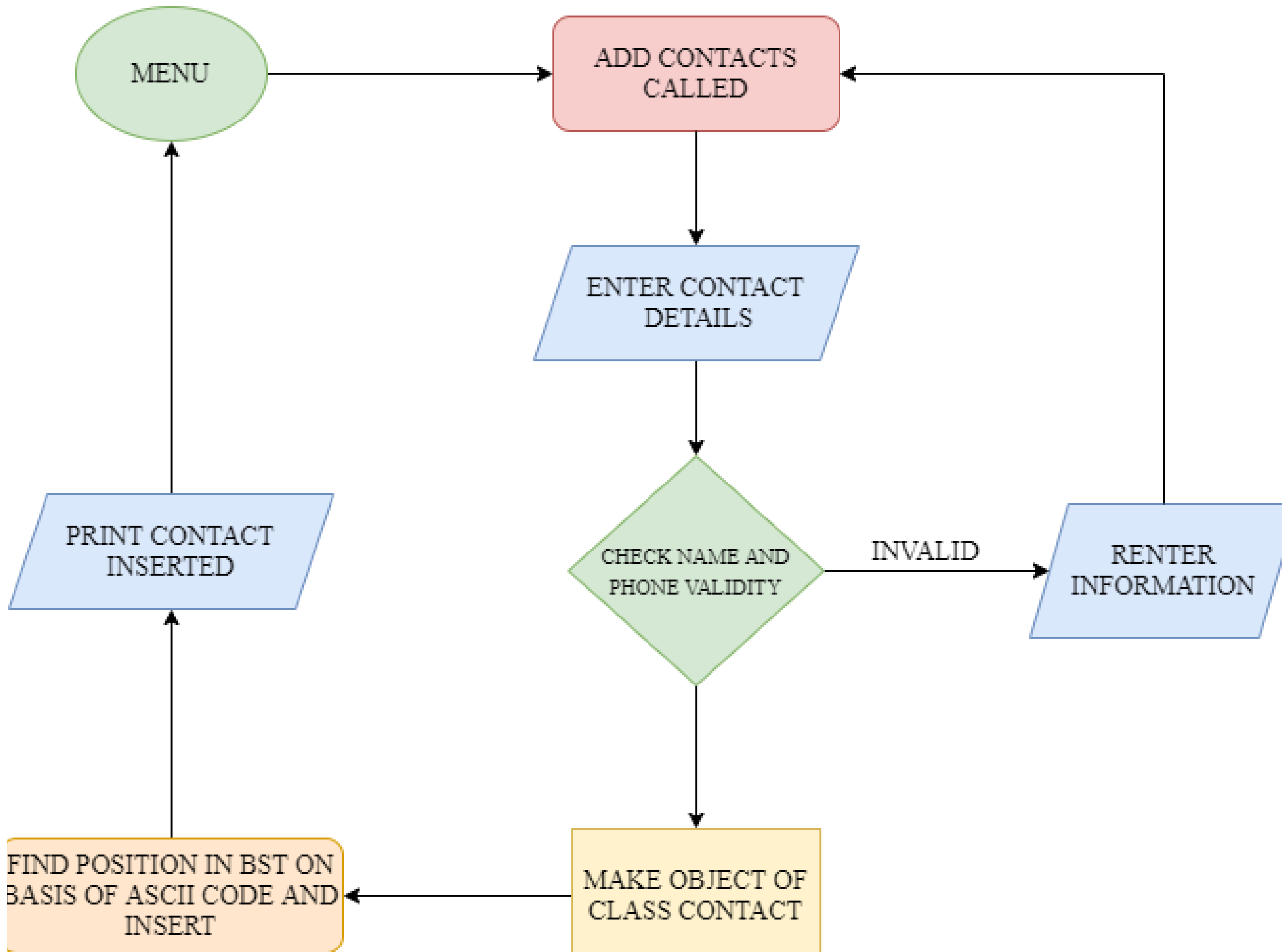
If the data is loaded from a file, the entries have the same format: The two fields are Full Name and Contact number. Every entry in the file is present in a new line and the entries may be unsorted.

On loading the file, the entries are stored as a Binary search tree data structure with the respective key-value pairs. The key used is the full name of the contact. To perform operations on this data, the application prompts the admin to enter the key. It then implements binary search on the key entered and returns to the menu if the key is found. If no such record will be available, a proper error message will be displayed as per the input provided to the system before returning to the menu.

FLOWCHART

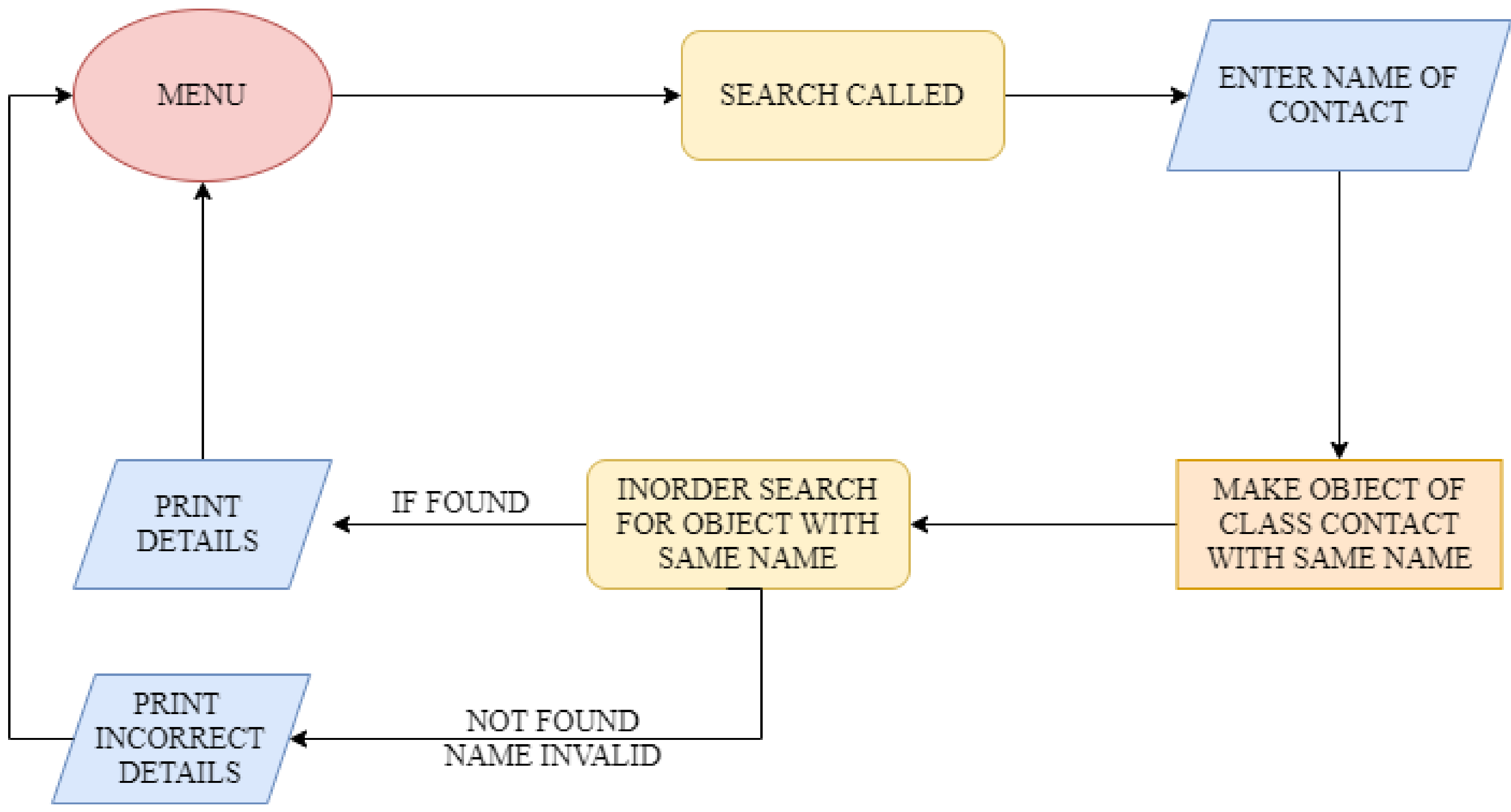


ADD CONTACTS

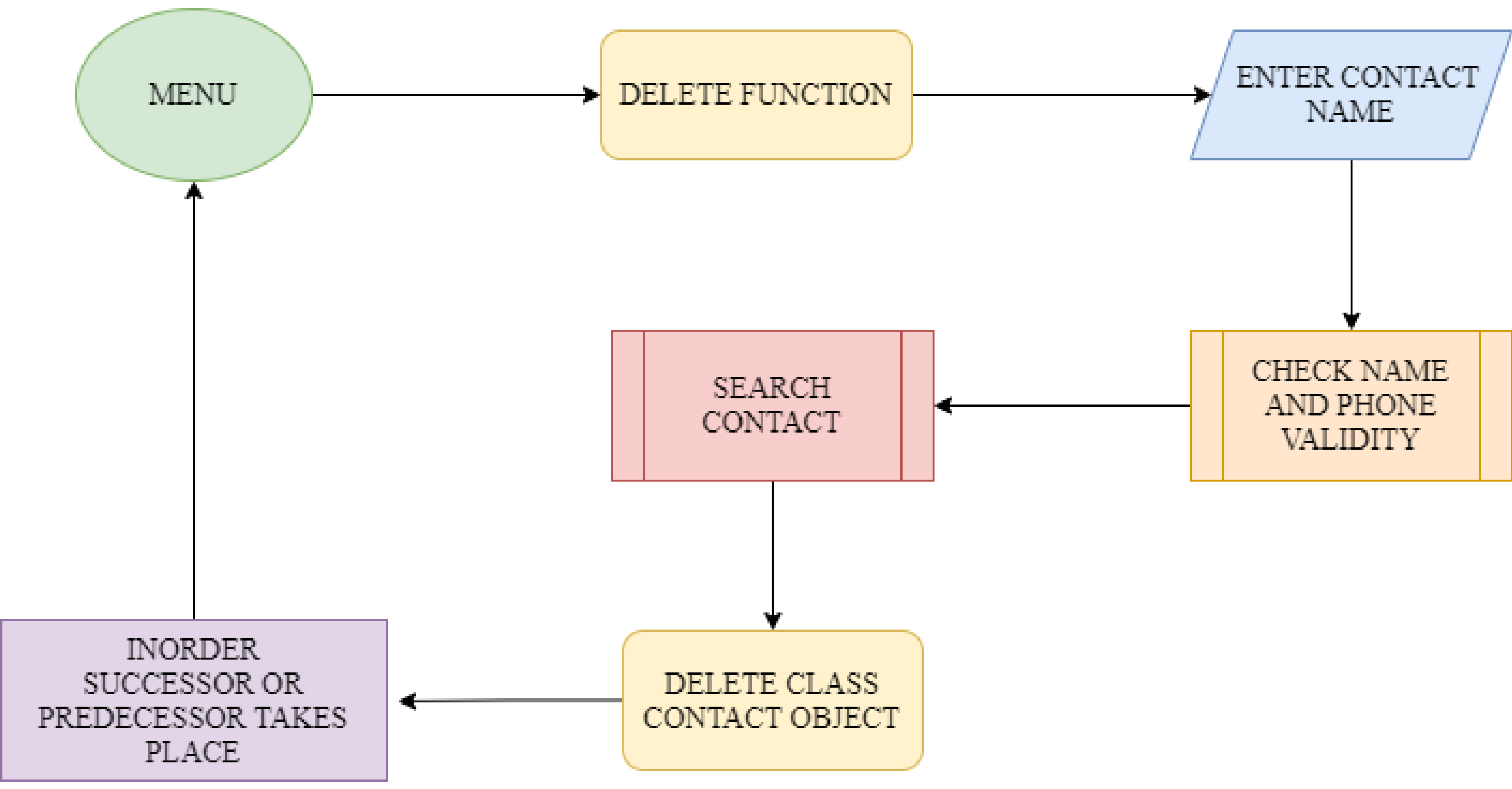


MENU
ADD CONTACTS CALLED
ENTER CONTACT DETAILS
CHECK NAME AND PHONE VALIDITY
INVALID
RENTER INFORMATION
PRINT CONTACT INSERTED
MAKE OBJECT OF CLASS CONTACT
FIND POSITION IN BST ON BASIS OF ASCII CODE AND INSERT

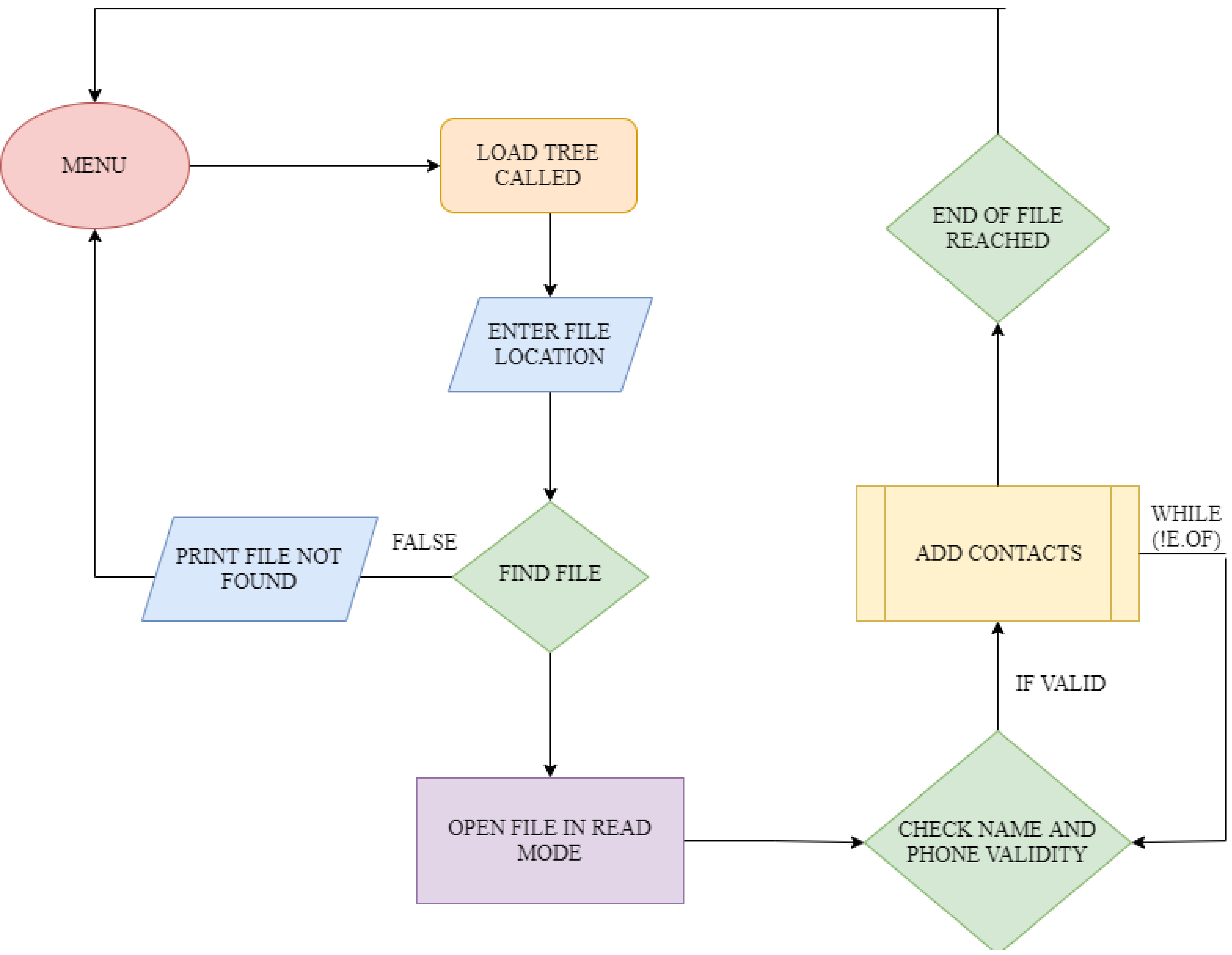
SEARCH CONTACTS



DELETE CONTACTS

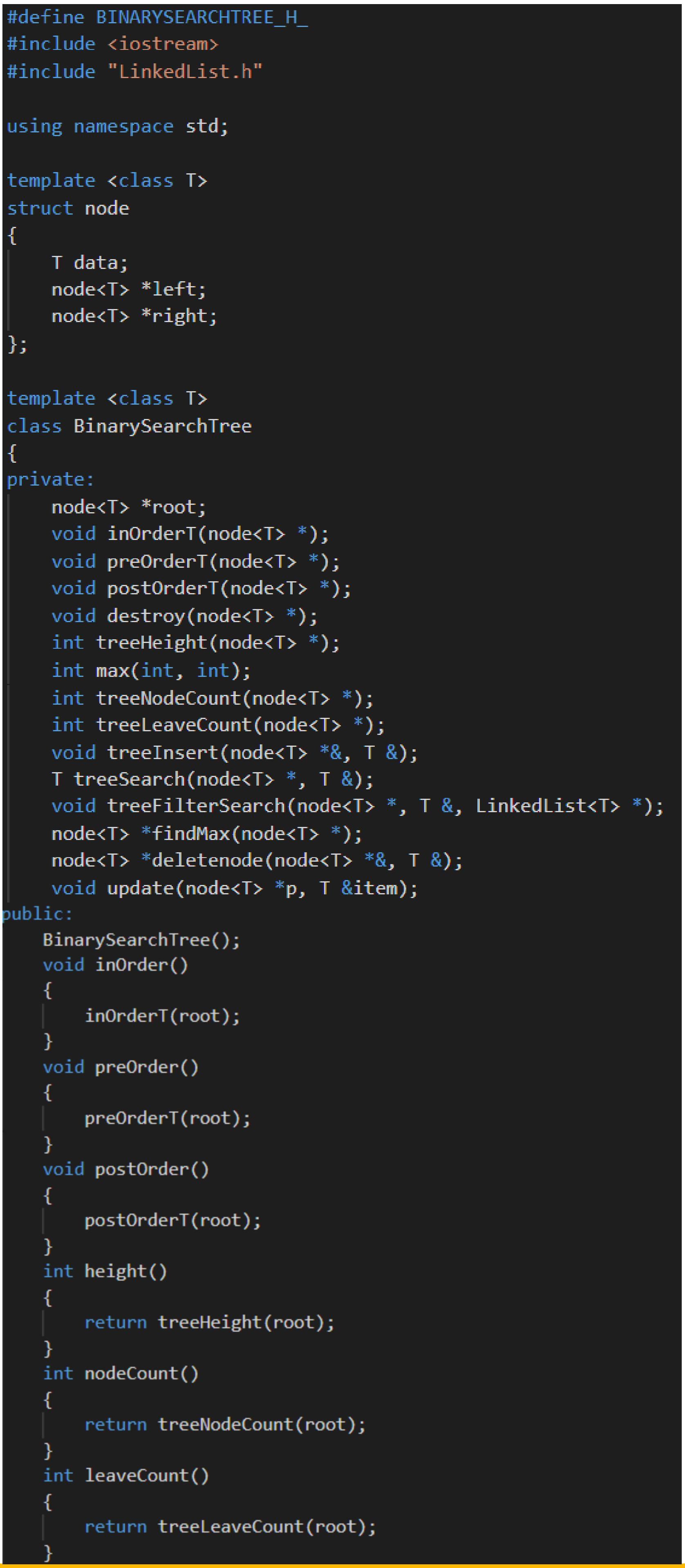

MENU
DELETE FUNCTION
ENTER CONTACT NAME
CHECK NAME AND PHONE VALIDITY
SEARCH CONTACT
DELETE CLASS CONTACT OBJECT
INORDER SUCCESSOR OR PREDECESSOR TAKES PLACE

LOAD CONTACTS

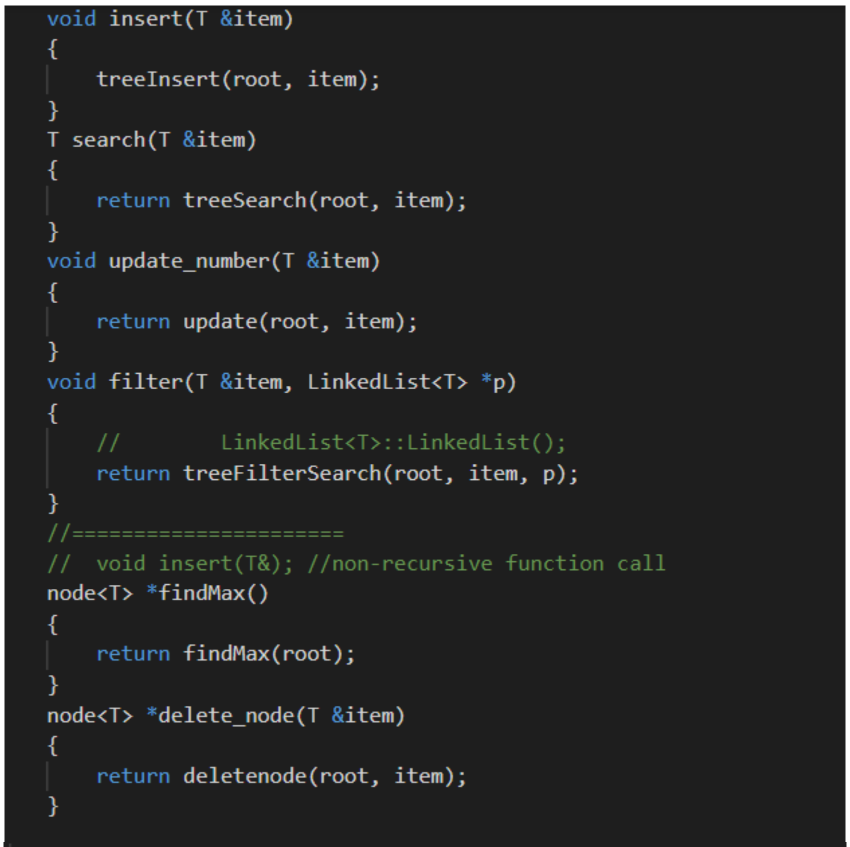


CODING IMPLEMENTATION

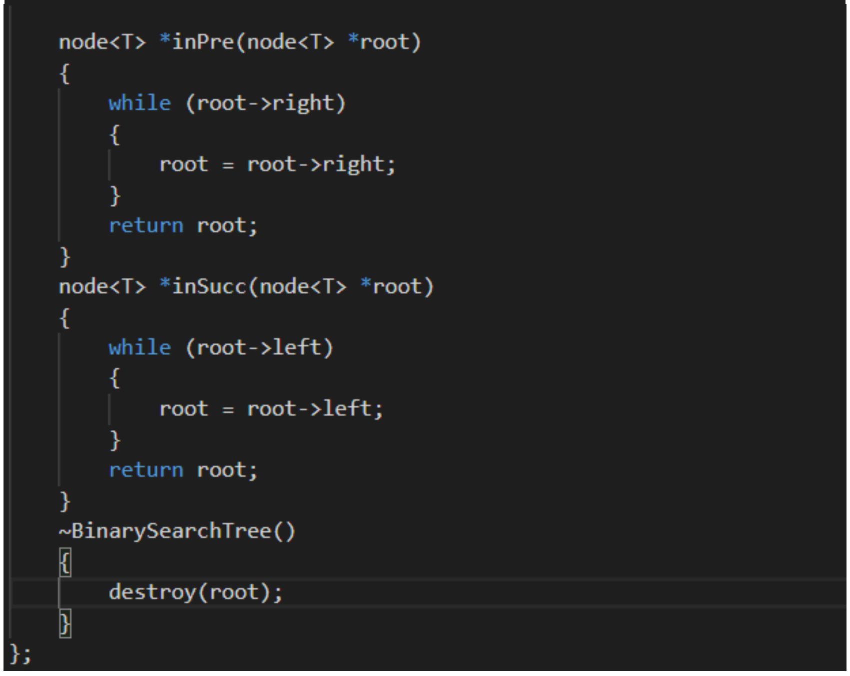
1. BinarySearchTree.h



```cpp
#define BINARYSEARCHTREE_H_
#include <iostream>
#include "LinkedList.h"

using namespace std;

template <class T>
struct node
{
    T data;
    node<T> *left;
    node<T> *right;
};

template <class T>
class BinarySearchTree
{
private:
    node<T> *root;
    void inOrderT(node<T> *);
    void preOrderT(node<T> *);
    void postOrderT(node<T> *);
    void destroy(node<T> *);
    int treeHeight(node<T> *);
    int max(int, int);
    int treeNodeCount(node<T> *);
    int treeLeaveCount(node<T> *);
    void treeInsert(node<T> *&, T &);
    T treeSearch(node<T> *, T &);
    void treeFilterSearch(node<T> *, T &, LinkedList<T> *);
    node<T> *findMax(node<T> *);
    node<T> *deletenode(node<T> *&, T &);
    void update(node<T> *p, T &item);
public:
   BinarySearchTree();
   void inOrder()
   {
       inOrderT(root);
   }
   void preOrder()
   {
       preOrderT(root);
   }
   void postOrder()
   {
       postOrderT(root);
   }
   int height()
   {
       return treeHeight(root);
   }
   int nodeCount()
   {
       return treeNodeCount(root);
   }
   int leaveCount()
   {
       return treeLeaveCount(root);
   }
```



```cpp
void insert(T &item)
{
    treeInsert(root, item);
}
T search(T &item)
{
    return treeSearch(root, item);
}
void update_number(T &item)
{
    return update(root, item);
}
void filter(T &item, LinkedList<T> *p)
{
    //        LinkedList<T>::LinkedList();
    return treeFilterSearch(root, item, p);
}
//======================
//  void insert(T&); //non-recursive function call
node<T> *findMax()
{
    return findMax(root);
}
node<T> *delete_node(T &item)
{
    return deletenode(root, item);
}
```

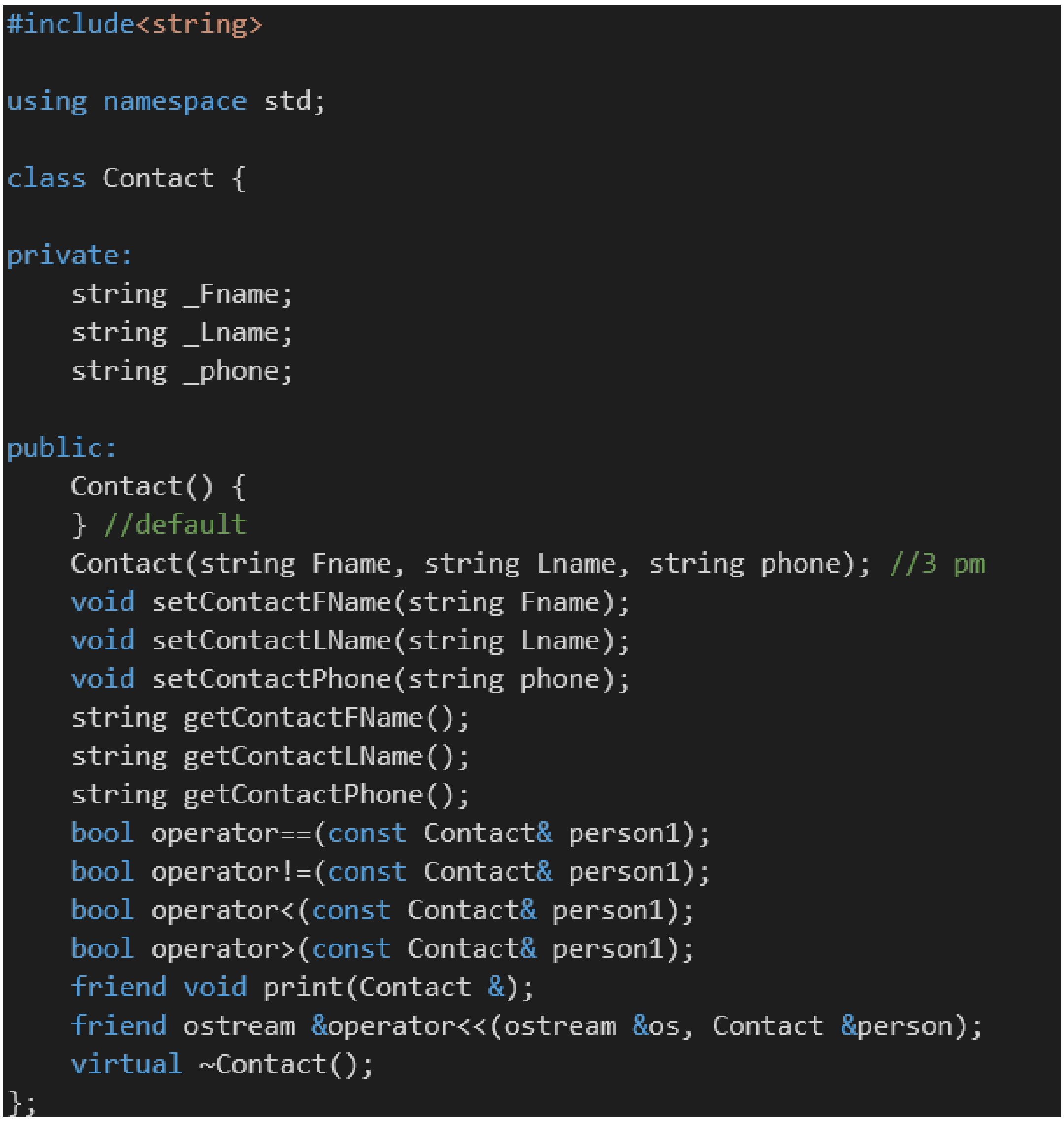


```cpp
 node<T> *inPre(node<T> *root)
 {
     while (root->right)
     {
         root = root->right;
     }
     return root;
 }
 node<T> *inSucc(node<T> *root)
 {
     while (root->left)
     {
         root = root->left;
     }
     return root;
 }
 ~BinarySearchTree()
 {
     destroy(root);
 }
};
```

- The BST has been implemented using a doubly-linked list.

- Various functions related to tree traversals, Node count, the height of the tree, inserting a Node, searching for a key, deleting a Node, etc. have been implemented inside the BinarySearchTree generic class.

2) Contact.h



```cpp
#include<string>

using namespace std;

class Contact {

private:
    string _Fname;
    string _Lname;
    string _phone;

public:
    Contact() {
    } //default
    Contact(string Fname, string Lname, string phone); //3 pm
    void setContactFName(string Fname);
    void setContactLName(string Lname);
    void setContactPhone(string phone);
    string getContactFName();
    string getContactLName();
    string getContactPhone();
    bool operator==(const Contact& person1);
    bool operator!=(const Contact& person1);
    bool operator<(const Contact& person1);
    bool operator>(const Contact& person1);
    friend void print(Contact &);
    friend ostream &operator<<(ostream &os, Contact &person);
    virtual ~Contact();
};
```

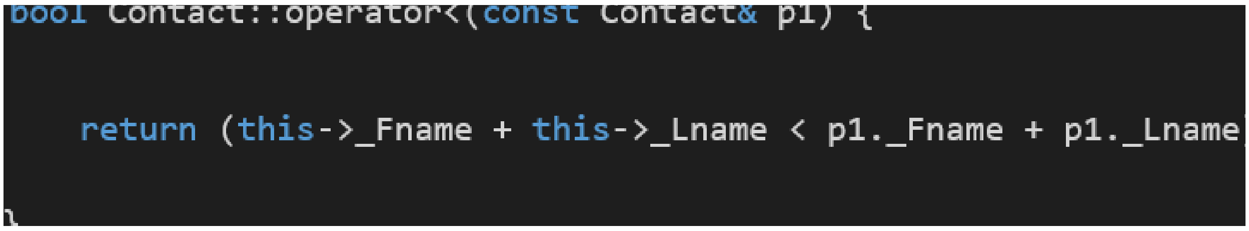
- This header file contains the class Contact which has the following attributes (string data type variables declared in private):

    1. First Name
    2. Last Name
    3. Phone Number

- Getter and setter functions are defined for each of the attributes along with the constructor and a destructor.

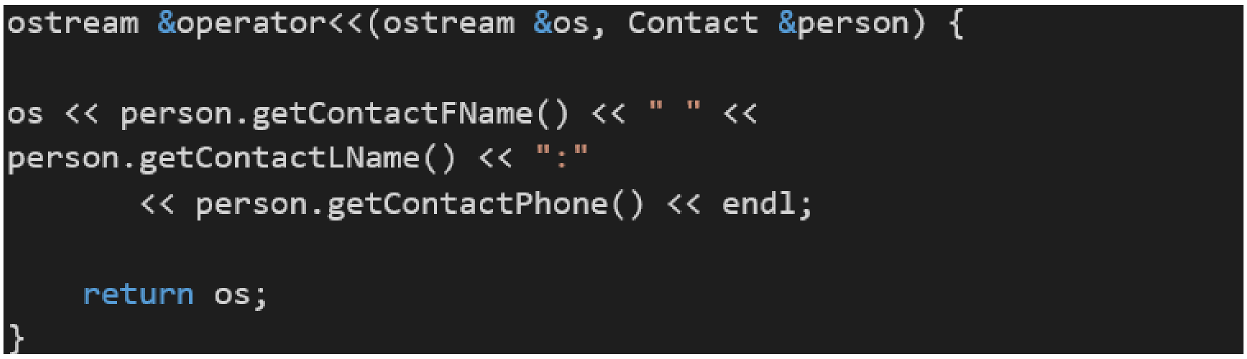
- The ==, !=, > and < operators are overloaded to make comparison of strings

- **Code for < operator**



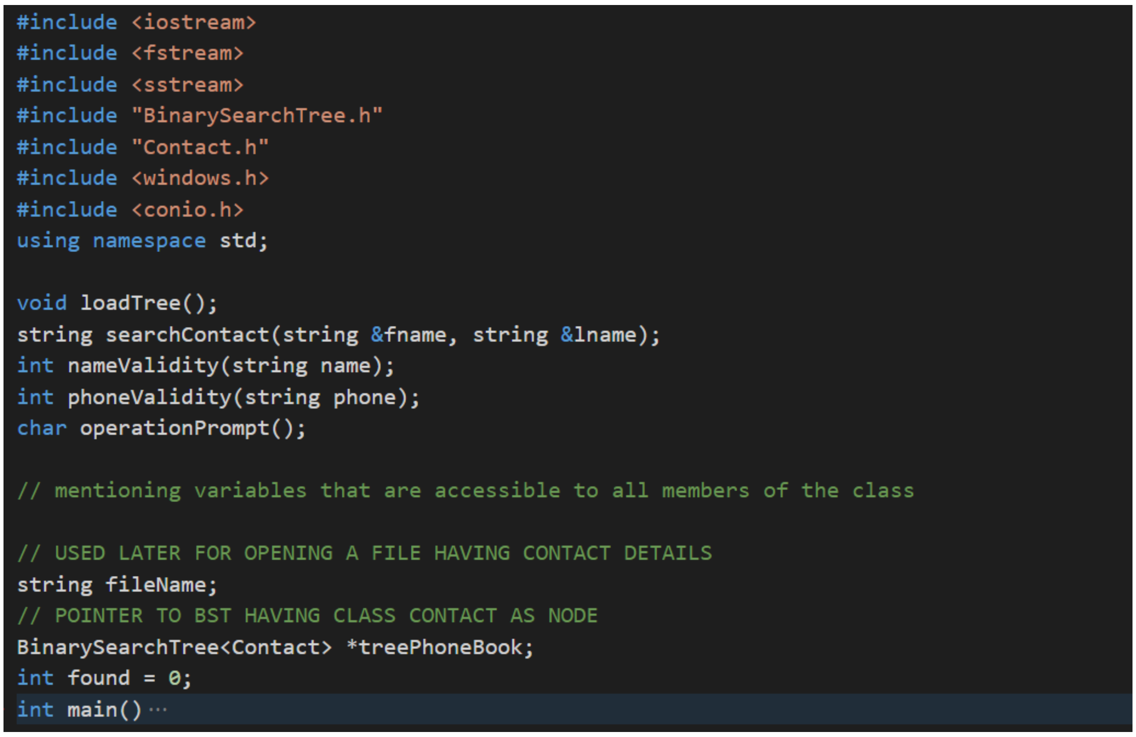
```
bool Contact::operator<(const Contact& p1) {

    return (this->_Fname + this->_Lname < p1._Fname + p1._Lname
}
```

- A friend function is used to overload the insertion “<<” operator to display the class attributes (First Name, Last Name, Phone Number) in the desired manner.



```
ostream &operator<<(ostream &os, Contact &person) {

os << person.getContactFName() << " " <<
person.getContactLName() << ":"
       << person.getContactPhone() << endl;

    return os;
}
```

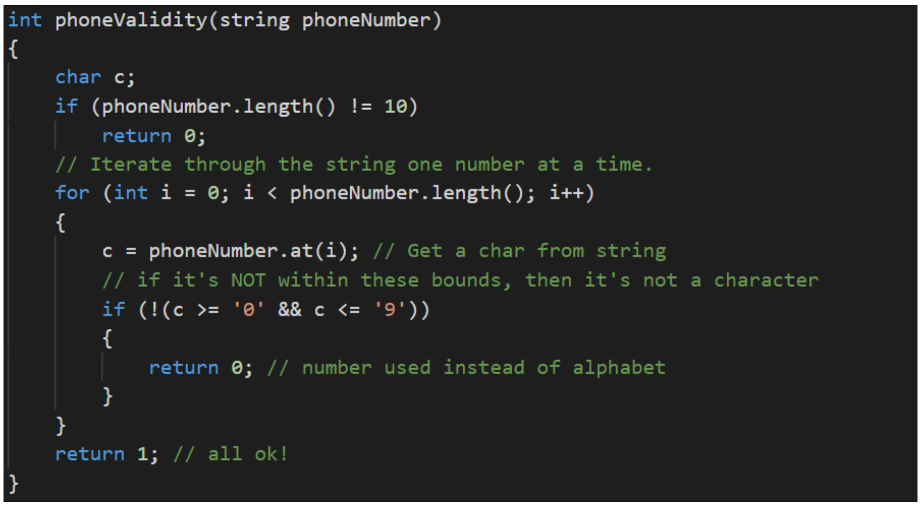
3) phonebook.cpp



```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include "BinarySearchTree.h"
#include "Contact.h"
#include <windows.h>
#include <conio.h>
using namespace std;

void loadTree();
string searchContact(string &fname, string &lname);
int nameValidity(string name);
int phoneValidity(string phone);
char operationPrompt();

// mentioning variables that are accessible to all members of the class

// USED LATER FOR OPENING A FILE HAVING CONTACT DETAILS
string fileName;
// POINTER TO BST HAVING CLASS CONTACT AS NODE
BinarySearchTree<Contact> *treePhoneBook;
int found = 0;
int main()⋯
```

- This is the main console application. To perform file processing in C++, standard header files **<iostream>** and **<fstream>** have been included. **<sstream>** has been included to provide templates and types that enable interoperation between stream buffers and string objects.
  "BinarySerachTree.h" and **"Contact.h"** have also been included.

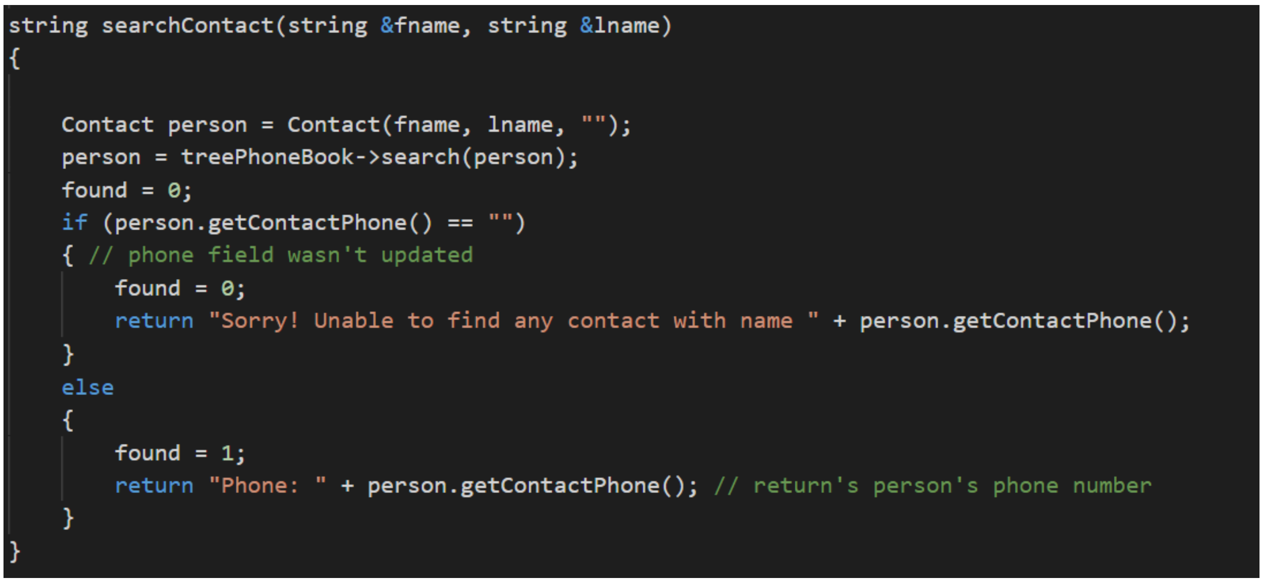
- The **loadTree()** method has been incorporated to load an external file (containing several contacts) during the execution time of the program.
- The validity methods: nameValidity() and phoneValidity() validate the information entered by the user, discarding any incorrect information.
- phoneValidity() has been described as follows:



```cpp
int phoneValidity(string phoneNumber)
{
    char c;
    if (phoneNumber.length() != 10)
        return 0;
    // Iterate through the string one number at a time.
    for (int i = 0; i < phoneNumber.length(); i++)
    {
        c = phoneNumber.at(i); // Get a char from string
        // if it's NOT within these bounds, then it's not a character
        if (!(c >= '0' && c <= '9'))
        {
            return 0; // number used instead of alphabet
        }
    }
    return 1; // all ok!
}
```

1. The method checks whether the number of digits entered **is equal to 10**. If not, the execution of the function ends and the program returns to the Menu.
2. If the first test is passed, the function checks whether the values entered are between **0 to 9** by iterating through the length of the phone number via a loop.
3. If the second test is also passed the function returns true.

- The **searchContact(string&, string&)** method searches for a specific contact within the BST.

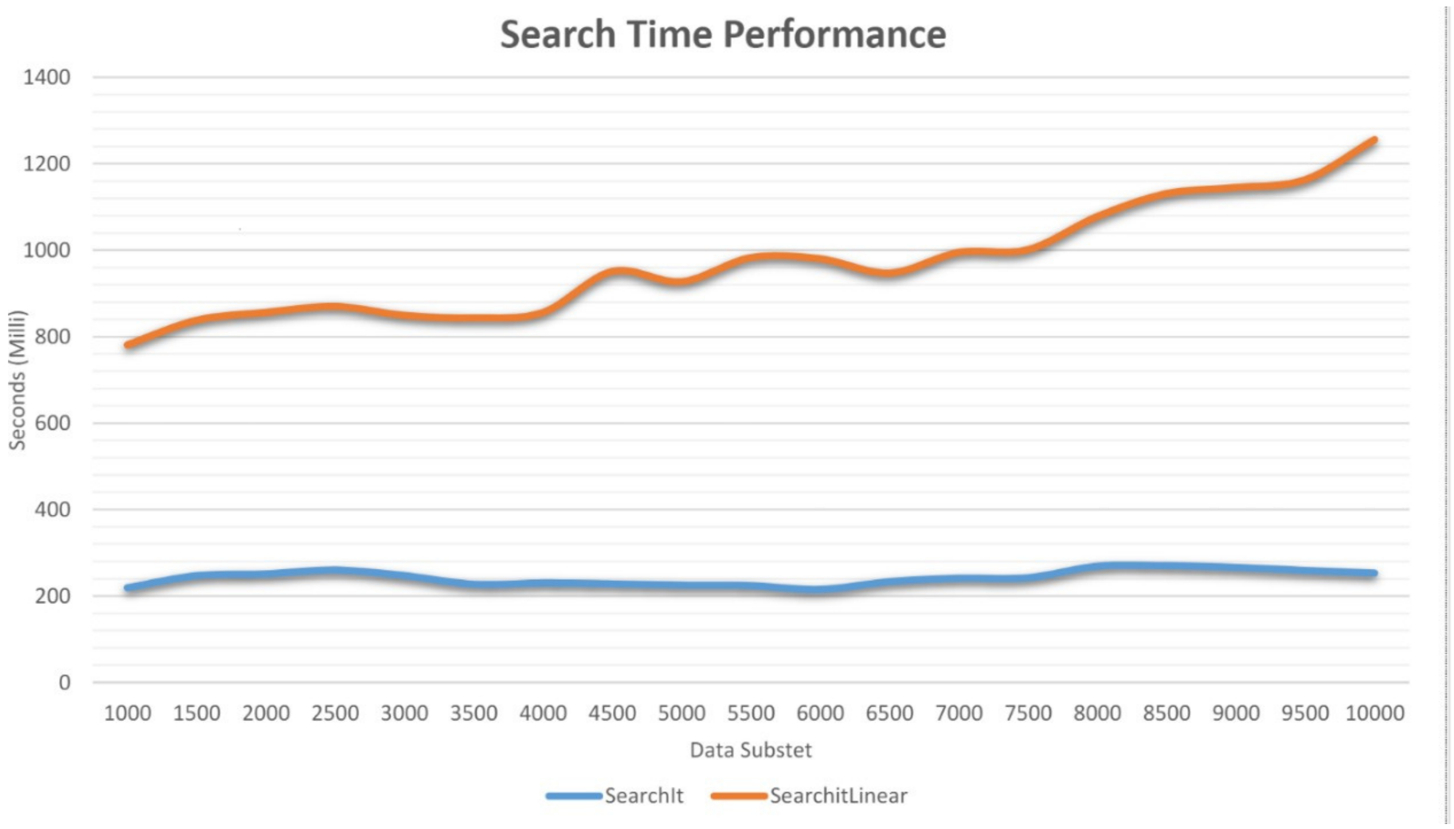


```
string searchContact(string &fname, string &lname)
{

    Contact person = Contact(fname, lname, "");
    person = treePhoneBook->search(person);
    found = 0;
    if (person.getContactPhone() == "")
    { // phone field wasn't updated
        found = 0;
        return "Sorry! Unable to find any contact with name " + person.getContactPhone();
    }
    else
    {
        found = 1;
        return "Phone: " + person.getContactPhone(); // return's person's phone number
    }
}
```

- The search result is stored in the variable: **found** (declared globally). If 1 the search is successful and the phone number of the contact is returned; otherwise the method terminates with a warning.

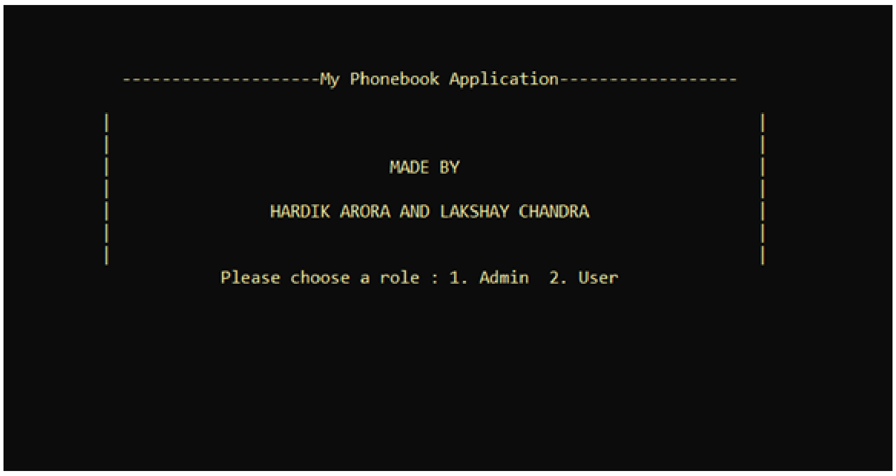
WHY BINARY SEARCH TREE?

- Binary Search **O(logn)** is more efficient than the linear search **O(n)**; We are able to implement Binary Search by storing the data (element, key) in a Binary Search Tree. Hence most of the operations in our program get implemented relatively fast.



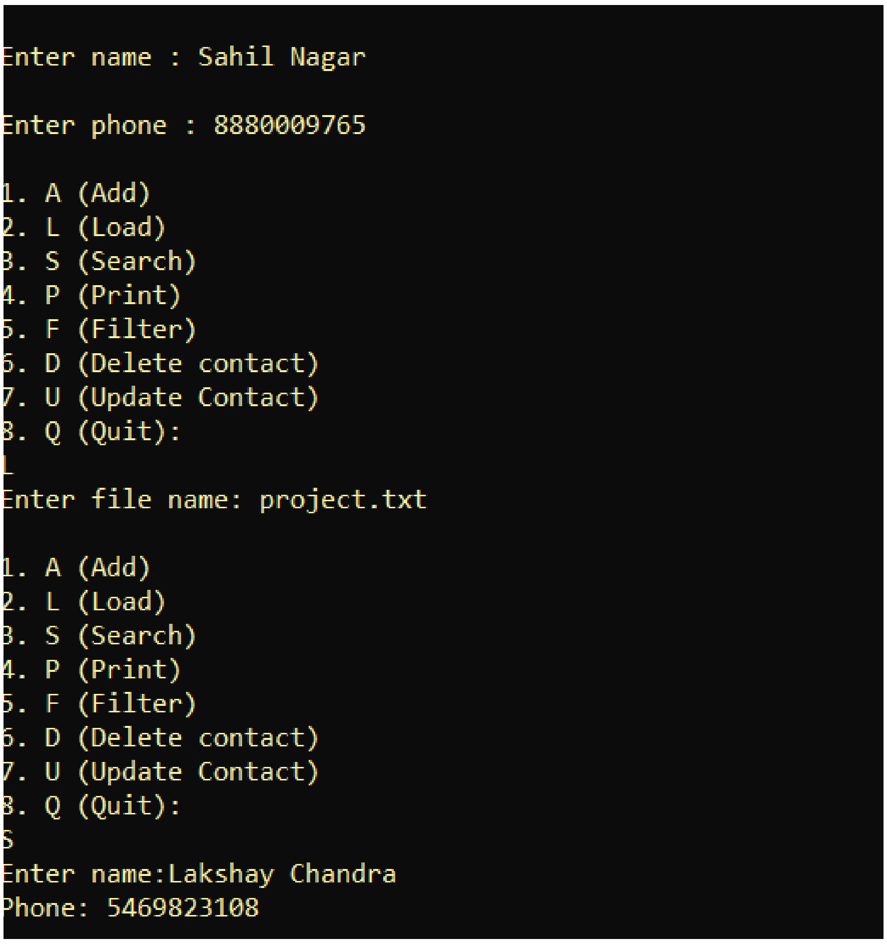
RESULTS

1) User Interface



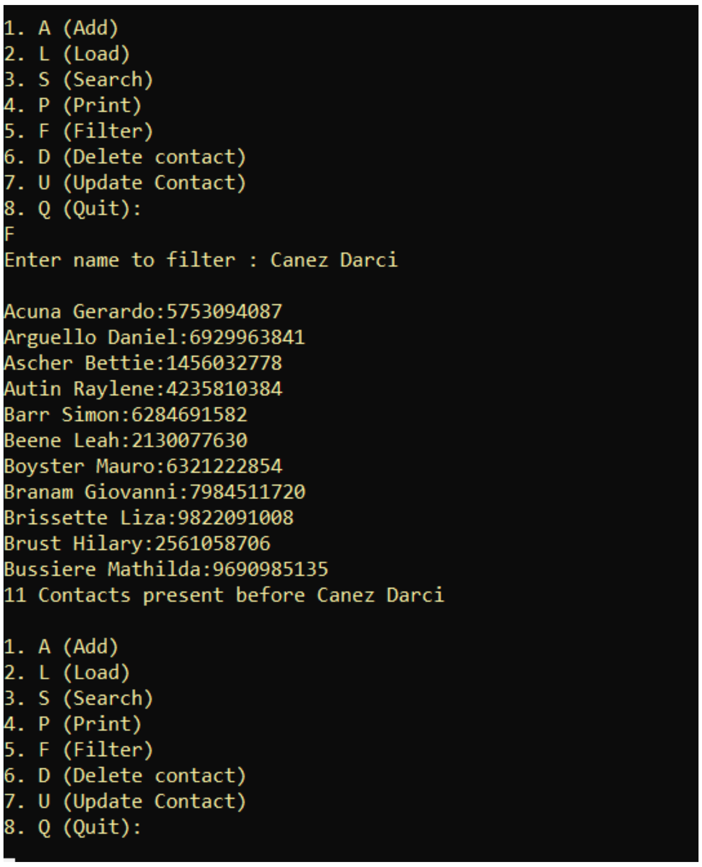
```
 -------------------My Phonebook Application-----------------

|                                                              |
|                                                              |
|                          MADE BY                             |
|                                                              |
|              HARDIK ARORA AND LAKSHAY CHANDRA                |
|                                                              |
|                                                              |
          Please choose a role : 1. Admin  2. User
```

2)Adding a Contact, loading a text file, and performing Search



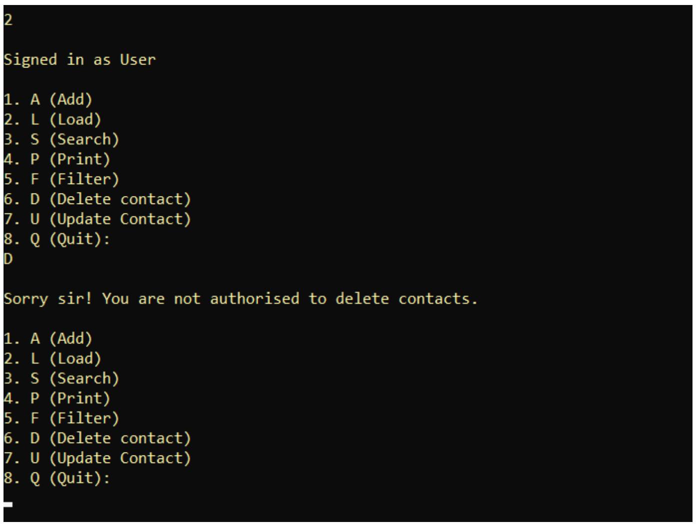
```
Enter name : Sahil Nagar

Enter phone : 8880009765

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
L
Enter file name: project.txt

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
S
Enter name:Lakshay Chandra
Phone: 5469823108
```

3) Filtering a Contact



```
1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
F
Enter name to filter : Canez Darci

Acuna Gerardo:5753094087
Arguello Daniel:6929963841
Ascher Bettie:1456032778
Autin Raylene:4235810384
Barr Simon:6284691582
Beene Leah:2130077630
Boyster Mauro:6321222854
Branam Giovanni:7984511720
Brissette Liza:9822091008
Brust Hilary:2561058706
Bussiere Mathilda:9690985135
11 Contacts present before Canez Darci

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
```

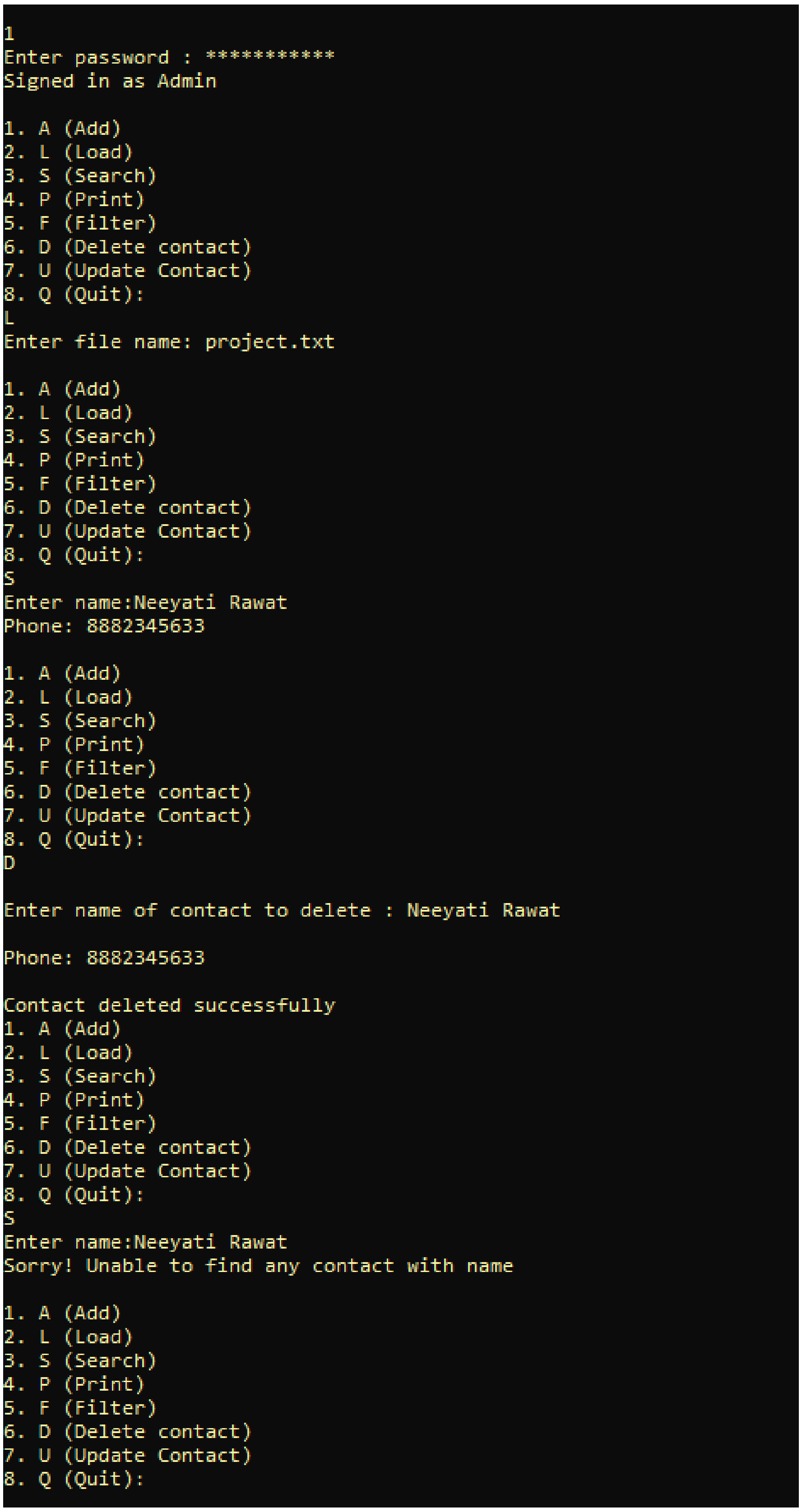
4) Deleting a Contact (User: Error)



```
2

Signed in as User

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
D

Sorry sir! You are not authorised to delete contacts.

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
```

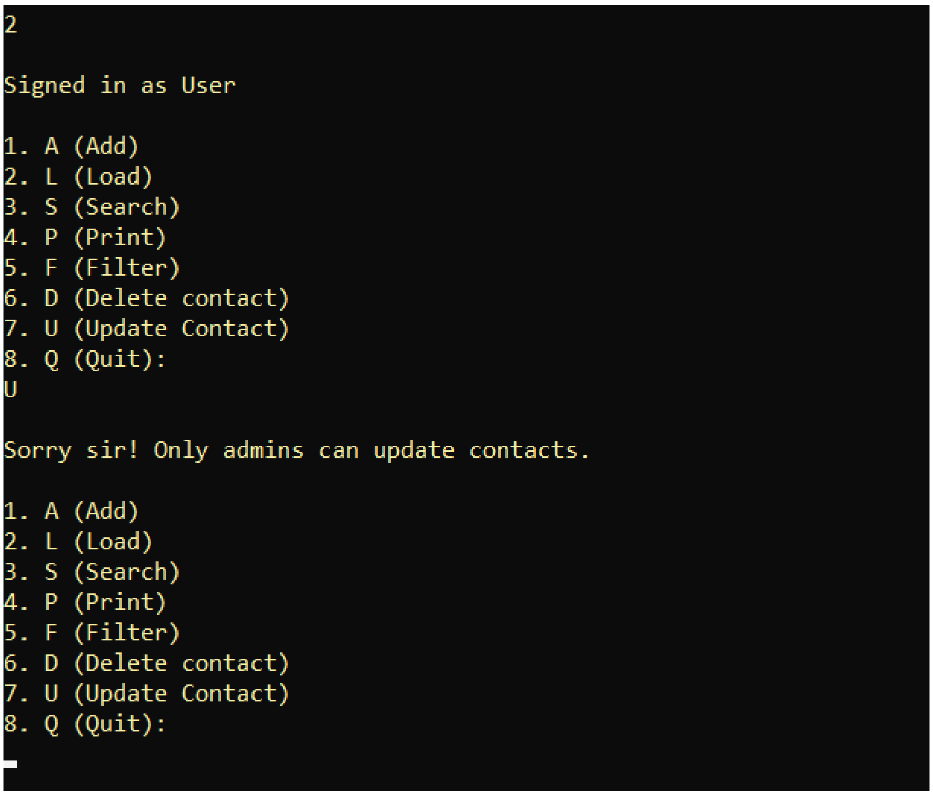
The removal of contacts can only be done by the admin; hence the corresponding function is password-protected. In order to prevent unknown users from deleting the contacts, the authority is vested in the administrator. **delete_node( )** is implemented by removing the node from BST and assigning the inorder successor/predecessor in its place by comparing the heights of the subtrees. This is done in order to make the binary tree more balanced and reduce searching time.

5) Deleting a contact (Admin)



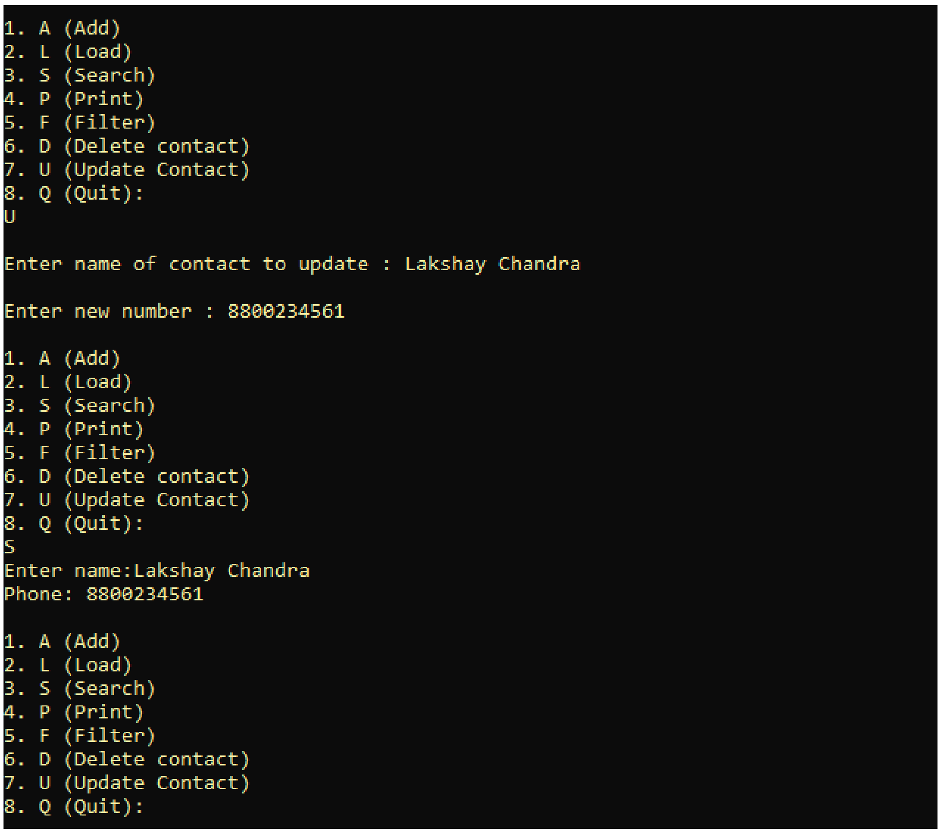
```
1
Enter password : ***********
Signed in as Admin

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
L
Enter file name: project.txt

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
S
Enter name:Neeyati Rawat
Phone: 8882345633

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
D

Enter name of contact to delete : Neeyati Rawat

Phone: 8882345633

Contact deleted successfully
1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
S
Enter name:Neeyati Rawat
Sorry! Unable to find any contact with name

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
```

6) Updating a Contact (User: Error)



```
2

Signed in as User

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
U

Sorry sir! Only admins can update contacts.

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
```

7) Updating a Contact (Admin)



```
1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
U

Enter name of contact to update : Lakshay Chandra

Enter new number : 8800234561

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
S
Enter name:Lakshay Chandra
Phone: 8800234561

1. A (Add)
2. L (Load)
3. S (Search)
4. P (Print)
5. F (Filter)
6. D (Delete contact)
7. U (Update Contact)
8. Q (Quit):
```

CONCLUSION

The proposed Phonebook management system overcomes the restrictions imposed by manual telephone directories. All the functionality has been added to meet the customer requirements in a matter of a few seconds. The system has been designed using a binary search tree data structure that optimizes the time complexity of search results. An object-oriented approach has been used to design the system in a pragmatic manner.

It is protected against unwanted modifications that could be made by the user to avoid vandalism of data by exercising admin rights. The users will not be able to delete/update any contact from the dataset.

As the new customers' details will be updated by the admin side, so users will be able to get updated information each time. To eliminate data redundancy and perform the validation process, background codes by the class will be responsible to do this task. If the admin will make any mistake while making an entry, he will be warned with an error message.

The simple UI of the system will also guide the customers to use this system in a convenient manner.

REFERENCES

1) http://www.cplusplus.com/reference/

2) https://www.geeksforgeeks.org/binary-search-tree-data-structure/

3) https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

4) https://en.cppreference.com/w/cpp/language/operators

5) Data Structures and Algorithm Analysis in C++ - Mark-Allen-Weise

6) Introduction to Algorithms - CLRS