`

CHANDIGARH UNIVERSITY

3D MODELING AND ANIMANTION

SEMESTER PROJECT

Student Name:	**VISHAY KUMAR**
Branch:	**BCA (AR&VR)**
Subject Name:	**DATA STRUCTURES**
UID:	**23BCA20032**
Section/Group:	**BCV-1/B**
Submitted to:	**Ms. Mehak Bhatia**
Subject Code:	**23CAT-201**

CONTACT REGISTER PROJECT USING DATA STRUCTURES IN C++

1. INTRODUCTION

In today's world, efficient storage and management of contact information are essential. The Contact Register Program is a C++ project that leverages data structures to store, retrieve, and manage contacts effectively. This project utilizes structures like linked lists, binary search trees (BST), and hash tables to provide fast access and efficient contact management. The system supports functionalities like adding, deleting, searching, and updating contacts, making it a comprehensive solution for managing personal or organizational contact information.

2. BACKGROUND

2.1 Purpose and Motivation

A contact register is an essential tool for storing and retrieving information on individuals efficiently. The motivation behind this project is to provide a robust program for handling contacts with various data structures, each chosen based on their specific advantages for storing, searching, and organizing contact information.

2.2 Role of Data Structures

Data structures play a critical role in optimizing the performance of any application that requires storing and retrieving data frequently. By using a binary search tree, the program ensures contacts are stored in a sorted order, allowing quick searches. A hash table provides constant-time complexity for searching contacts by a unique identifier, like a phone number or email. A linked list offers sequential access, which is useful for features like displaying the entire contact list.

2.3 Problem Statement

The program aims to solve common challenges in contact management, such as:

- Quick and efficient storage of contacts
- Fast retrieval of contacts by multiple criteria (e.g., name, phone number)
- Easy updating and deletion of contact information

3. SYSTEM DESIGN

3.1 System Architecture

The system is designed with modular classes to represent contacts and manage them using specific data structures. The program's architecture includes:

- Contact: A class representing individual contact details (e.g., name, phone, email).
- ContactManager: A main class for managing contacts using various data structures.
    - Linked List for sequentially stored contacts.
    - Binary Search Tree for sorted contact storage and quick lookups.
    - Hash Table for quick access by unique identifiers like email.

3.2 Data Flow and Workflow

1. User Interface: A menu-driven console-based UI lets users interact with the program to add, view, update, delete, or search for contacts.
2. Data Storage: Depending on the contact's storage needs, the system utilizes different data structures:
    - Linked List for sequential display.
    - Binary Search Tree for sorted storage.
    - Hash Table for unique identifier search.
3. Operations: Each function (add, delete, search) is implemented to optimize performance based on the data structure used.

3.3 Key Components

1. Contact Class:
    - Fields: name, phone, email, address.
    - Methods: Getters and Setters for each field, Input Validation.
2. Data Structure Modules:
    - Linked List Module: Manages contacts sequentially.
    - Binary Search Tree Module: Ensures sorted storage for efficient searching.
    - Hash Table Module: Optimized for quick search by email or phone.

4. IMPLEMENTATION

4.1 Development Environment

The program is developed using C++ in a console environment, tested on compilers such as GCC and Microsoft Visual Studio.

4.2 Code Structure

1. Contact Class Implementation:
    - The Contact class contains fields and methods for handling each contact's information.
2. ContactManager Class Implementation:
    - Add Contact: Accepts user input, creates a contact, and stores it in the appropriate data structure.
    - Search Contact: Searches contacts by name, phone, or email, using the BST or Hash Table based on criteria.

   - Update Contact: Finds and updates an existing contact’s information.
   - Delete Contact: Removes a contact from all data structures.
3. User Interface:
   - Provides a console-based interface for user interaction with options for each operation (Add, Search, Update, Delete).

Note: code input and output are on the next page. The output is of all the working possibilities out there.

PROGRAM CODE

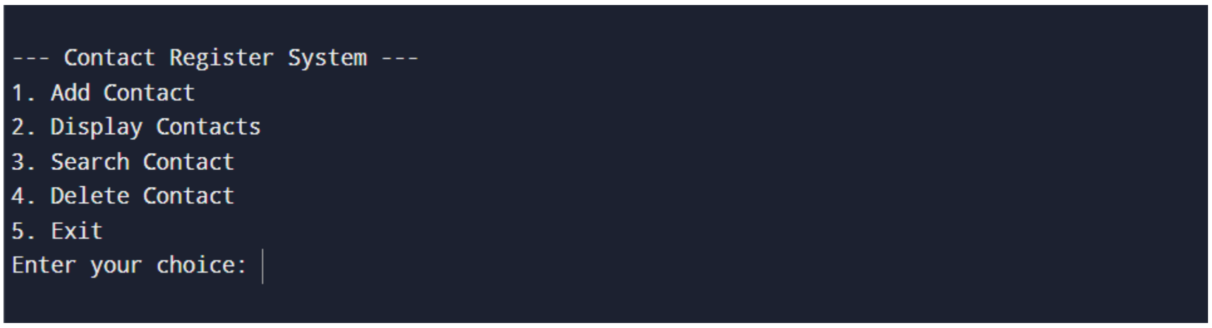
```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

struct Contact {
    string name;
    string phone;
    string email;
};


void addContact(vector<Contact>& contacts) {
    Contact newContact;
    cout << "Enter name: ";
    cin.ignore();
    getline(cin, newContact.name);
    cout << "Enter phone: ";
    getline(cin, newContact.phone);
    cout << "Enter email: ";
    getline(cin, newContact.email);

    contacts.push_back(newContact);
    cout << "Contact added successfully!\n";
}


void displayContacts(const vector<Contact>& contacts) {
    if (contacts.empty()) {
        cout << "No contacts to display.\n";
        return;
    }
    cout << "Contacts List:\n";
    for (const auto& contact : contacts) {
        cout << "Name: " << contact.name << ", Phone: " <<
contact.phone << ", Email: " << contact.email << endl;
    }
}
```

```
void searchContact(const vector<Contact>& contacts) {
    cout << "Enter name to search: ";
    string searchName;
    cin.ignore();
    getline(cin, searchName);

    bool found = false;
    for (const auto& contact : contacts) {
        if (contact.name == searchName) {
            cout << "Contact found: Name: " << contact.name
<< ", Phone: " << contact.phone << ", Email: " <<
contact.email << endl;
            found = true;
            break;
        }
    }
    if (!found) {
        cout << "Contact not found.\n";
    }
}


void deleteContact(vector<Contact>& contacts) {
    cout << "Enter name to delete: ";
    string deleteName;
    cin.ignore();
    getline(cin, deleteName);

    auto it = remove_if(contacts.begin(), contacts.end(),
                        [&deleteName](const Contact& contact) {
return contact.name == deleteName; });
```

```
    if (it != contacts.end()) {
        contacts.erase(it, contacts.end());
        cout << "Contact deleted successfully.\n";
    } else {
        cout << "Contact not found.\n";
    }
}

int main() {
    vector<Contact> contacts;
    int choice;

    while (true) {
        cout << "\n--- Contact Register System ---\n";
        cout << "1. Add Contact\n";
        cout << "2. Display Contacts\n";
        cout << "3. Search Contact\n";
        cout << "4. Delete Contact\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                addContact(contacts);
                break;
            case 2:
                displayContacts(contacts);
                break;
            case 3:
                searchContact(contacts);
                break;
            case 4:
                deleteContact(contacts);
                break;
            case 5:
                cout << "Exiting the program.\n";
                return 0;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }
}
```

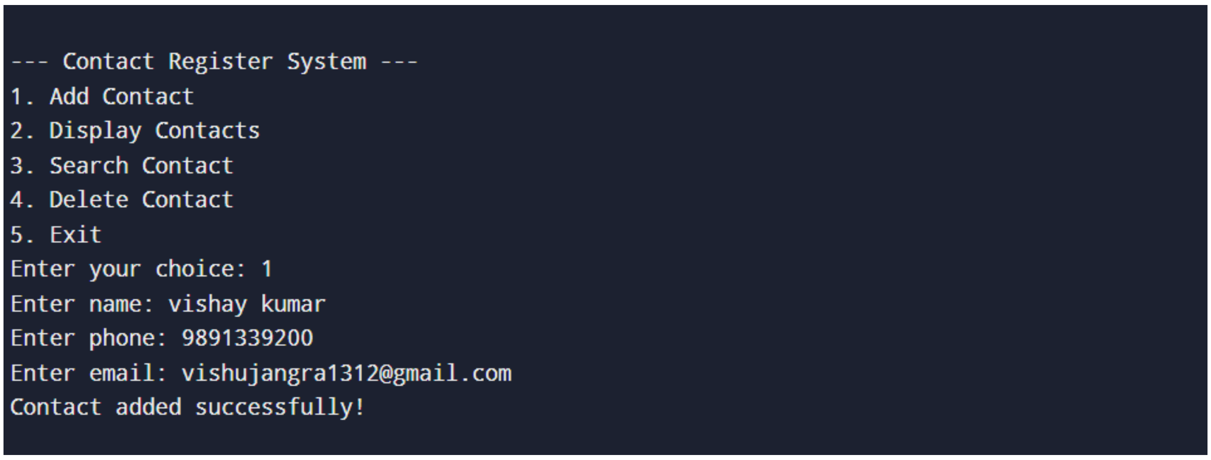
PROGRAM OUTPUT

1. The main output of the program:



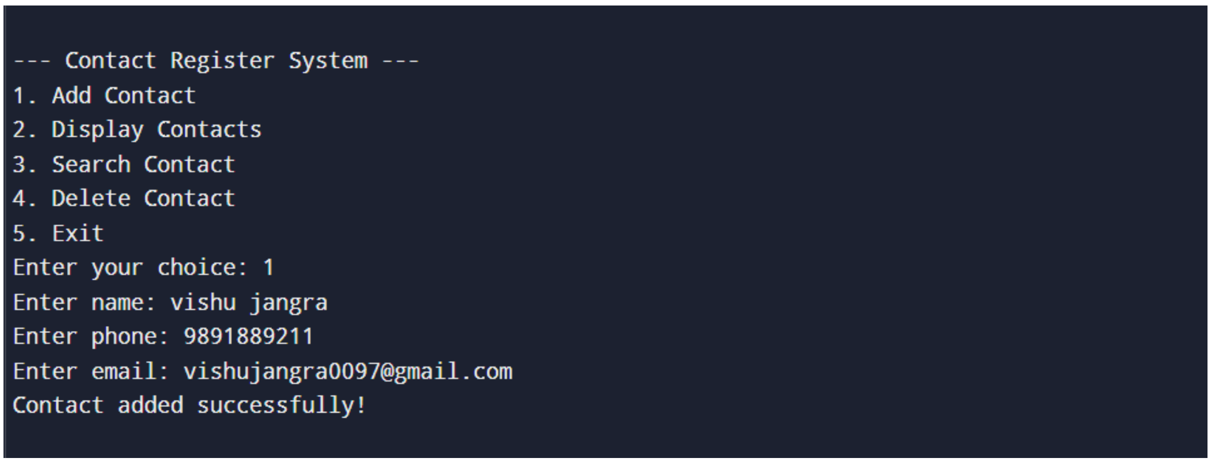
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice:
```

2. Saving the first contact in the register:



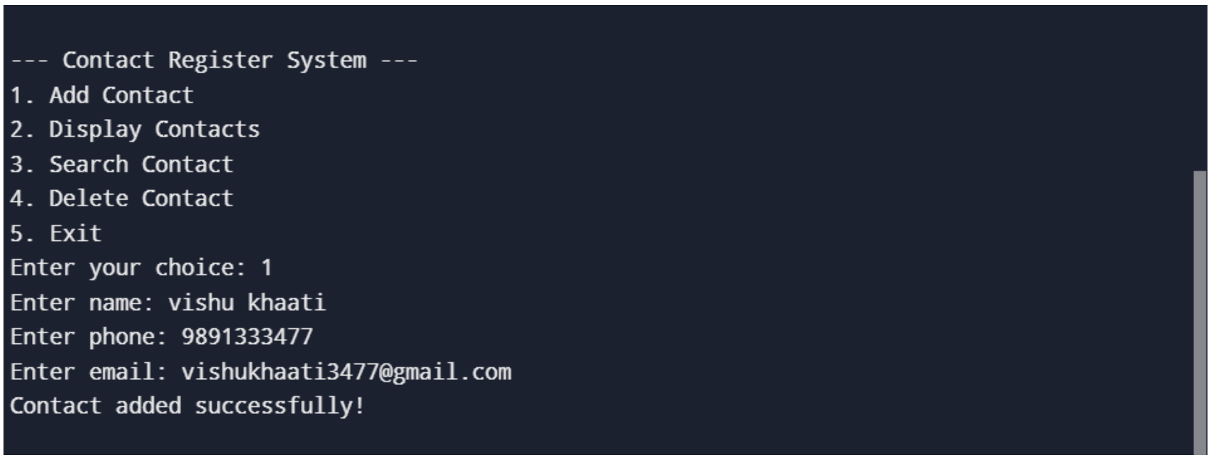
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 1
Enter name: vishay kumar
Enter phone: 9891339200
Enter email: vishujangra1312@gmail.com
Contact added successfully!
```

3. Saving the second contact in the register



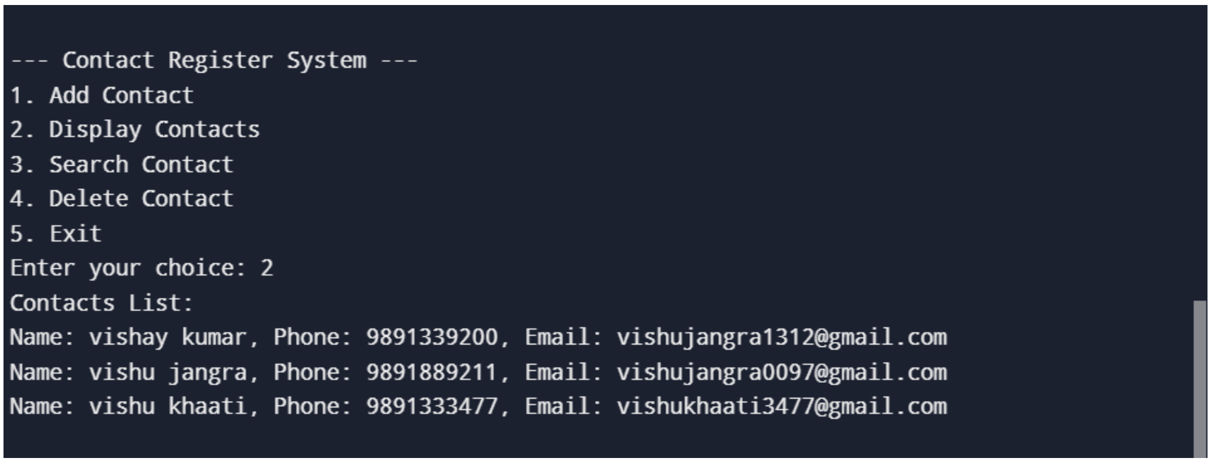
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 1
Enter name: vishu jangra
Enter phone: 9891889211
Enter email: vishujangra0097@gmail.com
Contact added successfully!
```

4. Saving the third contact in the register:



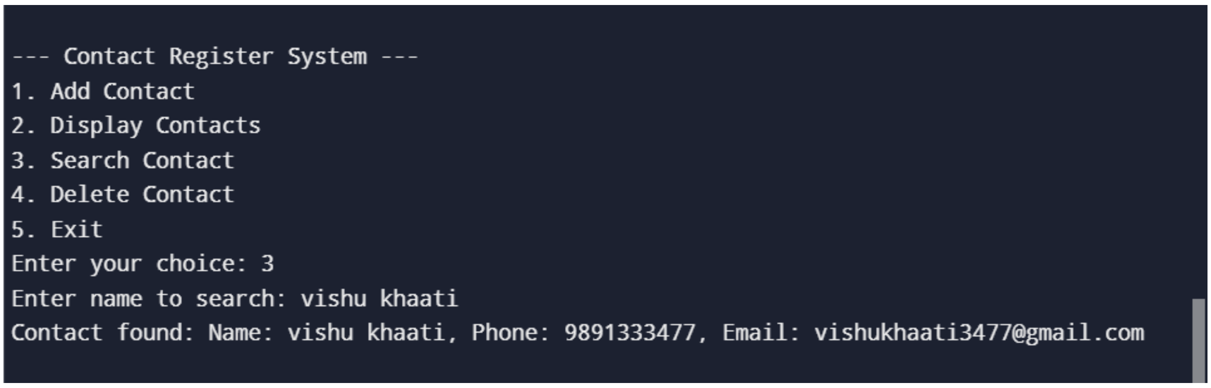
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 1
Enter name: vishu khaati
Enter phone: 9891333477
Enter email: vishukhaati3477@gmail.com
Contact added successfully!
```

5. Displaying all the contact in the register:



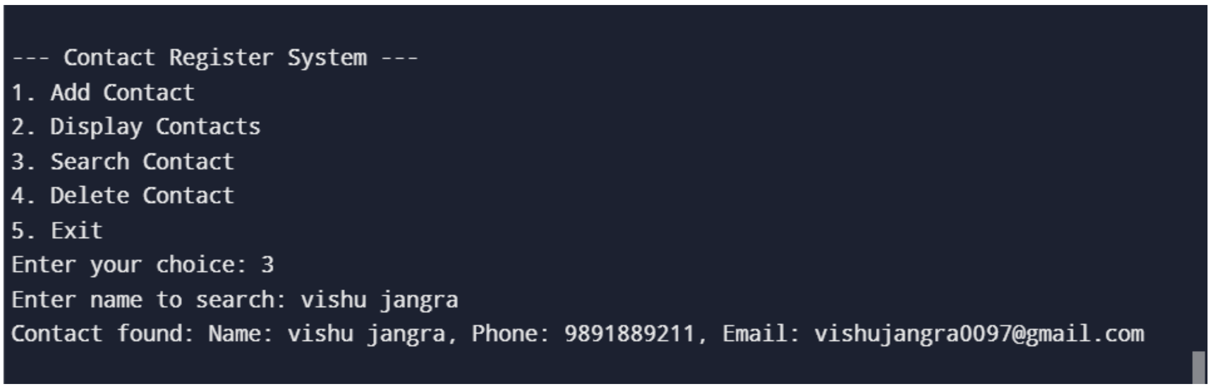
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 2
Contacts List:
Name: vishay kumar, Phone: 9891339200, Email: vishujangra1312@gmail.com
Name: vishu jangra, Phone: 9891889211, Email: vishujangra0097@gmail.com
Name: vishu khaati, Phone: 9891333477, Email: vishukhaati3477@gmail.com
```

6. Searching the first contact in the register:



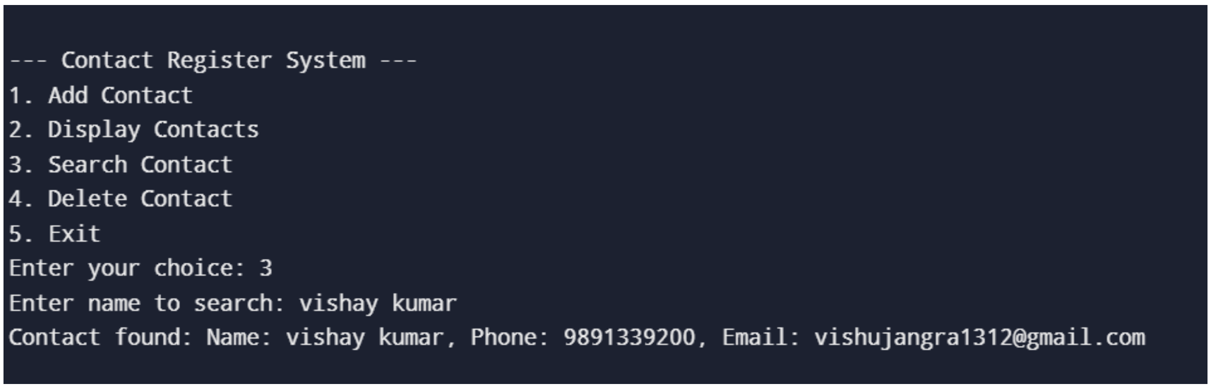
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 3
Enter name to search: vishu khaati
Contact found: Name: vishu khaati, Phone: 9891333477, Email: vishukhaati3477@gmail.com
```

7. Searching the second contact in the register:



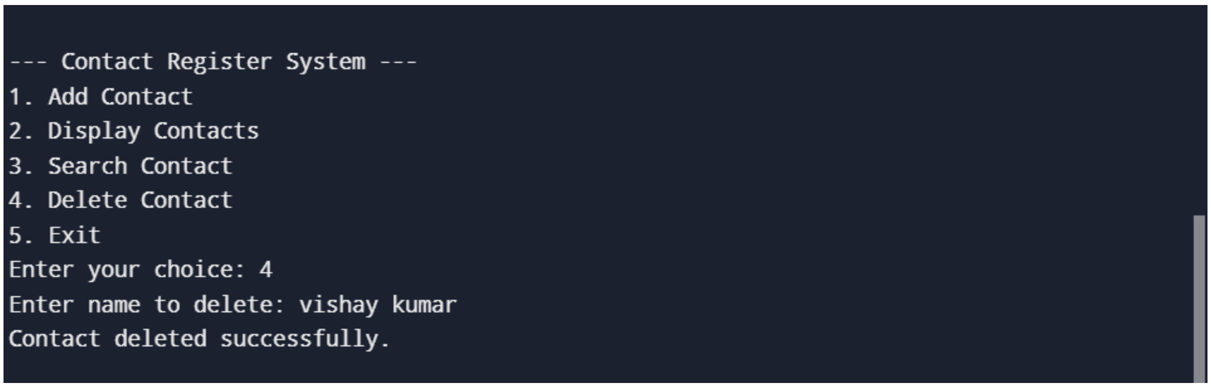
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 3
Enter name to search: vishu jangra
Contact found: Name: vishu jangra, Phone: 9891889211, Email: vishujangra0097@gmail.com
```

8. Searching the third contact in the register:



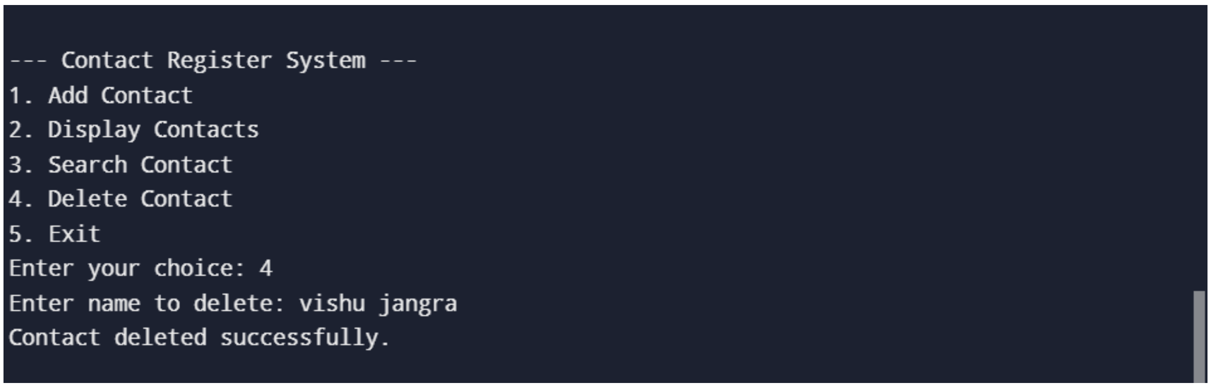
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 3
Enter name to search: vishay kumar
Contact found: Name: vishay kumar, Phone: 9891339200, Email: vishujangra1312@gmail.com
```

9. Deleting the first contact in the register:



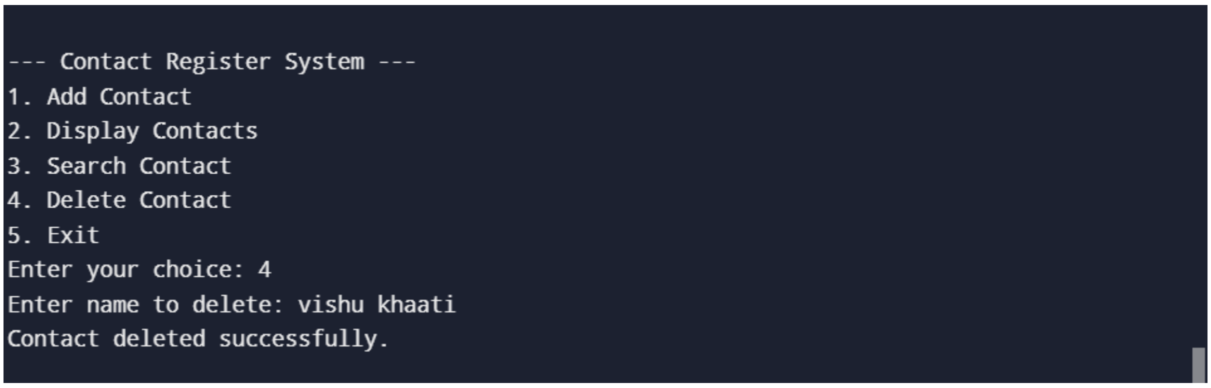
```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 4
Enter name to delete: vishay kumar
Contact deleted successfully.
```

10. Deleting the second contact in the register:

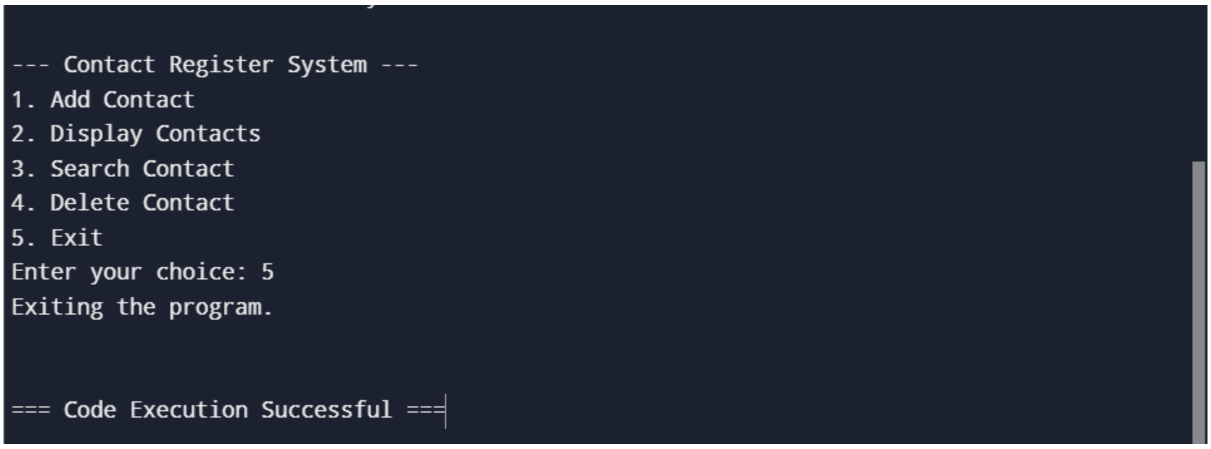


```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 4
Enter name to delete: vishu jangra
Contact deleted successfully.
```

11. Deleting the third contact in the register:


```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 4
Enter name to delete: vishu khaati
Contact deleted successfully.
```

12. Exiting the contact register:


```
--- Contact Register System ---
1. Add Contact
2. Display Contacts
3. Search Contact
4. Delete Contact
5. Exit
Enter your choice: 5
Exiting the program.


=== Code Execution Successful ===
```

5. CONCLUSION

The Contact Register Program effectively demonstrates how data structures can improve performance and efficiency in a contact management system. The use of multiple data structures, including linked lists, binary search trees, and hash tables, allows for flexible and optimized contact handling.

5.1 Key Takeaways

1. Efficiency: Data structures significantly impact performance, especially in search-heavy applications.
2. Modularity: The program's design promotes modularity, making it easy to extend or replace data structures if needed.
3. Scalability: The program can be extended to handle large numbers of contacts efficiently.

5.2 Future Work

To further enhance the project, additional features like a GUI, file-based storage for persistence, and sorting options can be implemented.

THANK YOU

By VISHAY KUMAR

23BCA20032 23BCV-1