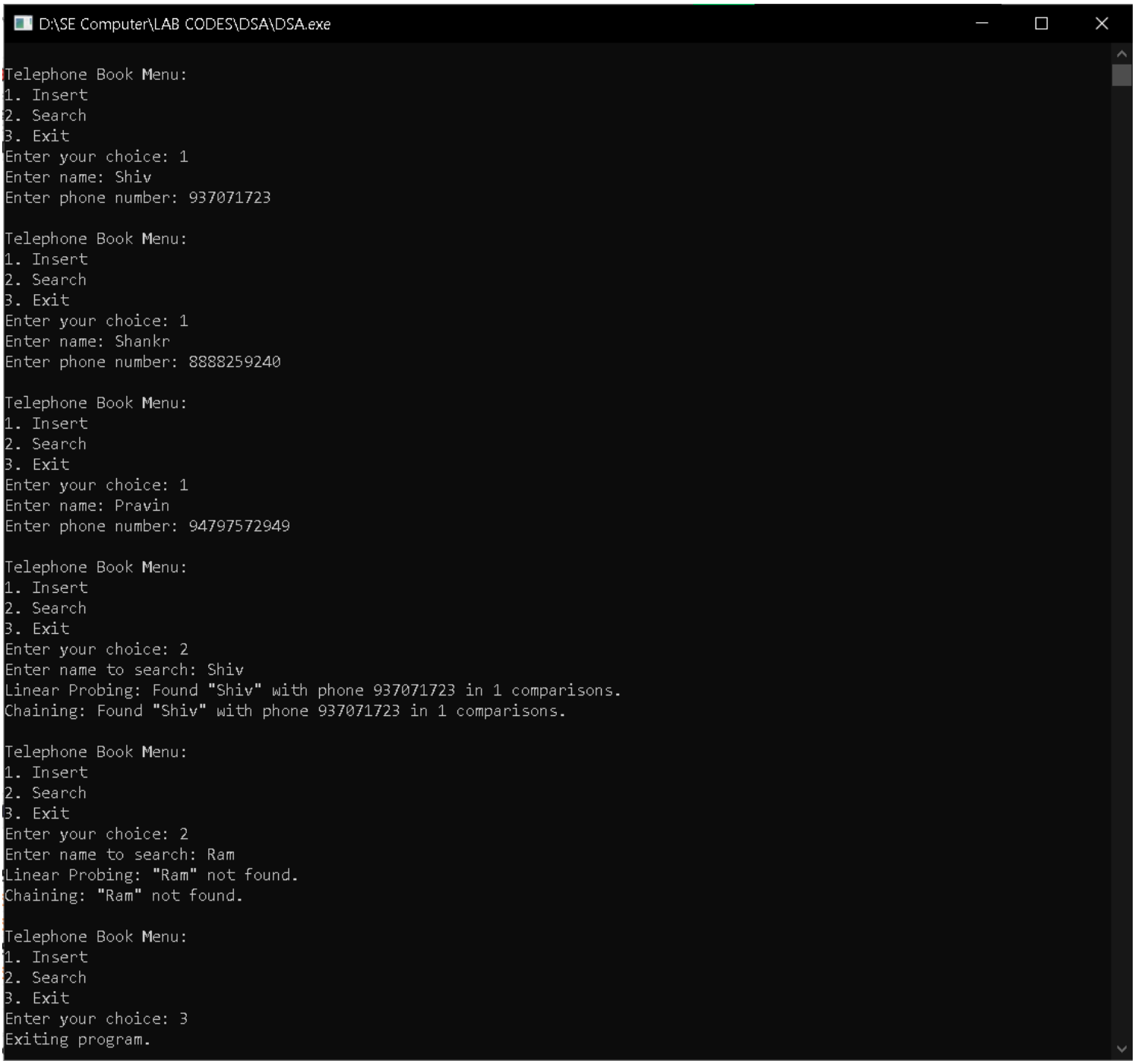
```cpp
#include <iostream>
#include <vector>
#include <list>
using namespace std;

const int TABLE_SIZE = 10;

// Entry structure
struct Entry
{
    string name;
    string phone;
};

// Linear Probing Hash Table
class LinearProbingHashTable
{
    vector<Entry> table;
    vector<bool> occupied;

public:
    LinearProbingHashTable() : table(TABLE_SIZE), occupied(TABLE_SIZE, false) {}

    int hashFunction(const string &key)
    {
        int hash = 0;
        for (char ch : key)
        {
            hash = (hash + ch) % TABLE_SIZE;
        }
        return hash;
    }

    void insert(const string &name, const string &phone)
    {
        int index = hashFunction(name);
        int originalIndex = index;
        while (occupied[index])
        {
            index = (index + 1) % TABLE_SIZE;
            if (index == originalIndex)
            {
                cout << "Hash table is full!\n";
                return;
            }
        }
        table[index] = {name, phone};
        occupied[index] = true;
    }
```

```
50
51      bool search(const string &name, string &phone, int &comparisons)
52      {
53          int index = hashFunction(name);
54          int originalIndex = index;
55          comparisons = 0;
56          while (occupied[index])
57          {
58              comparisons++;
59              if (table[index].name == name)
60              {
61                  phone = table[index].phone;
62                  return true;
63              }
64              index = (index + 1) % TABLE_SIZE;
65              if (index == originalIndex)
66                  break;
67          }
68          return false;
69      }
70  };
71
72  // Chaining Hash Table
73  class ChainingHashTable
74  {
75      vector<list<Entry>> table;
76
77  public:
78      ChainingHashTable() : table(TABLE_SIZE) {}
79
80      int hashFunction(const string &key)
81      {
82          int hash = 0;
83          for (char ch : key)
84          {
85              hash = (hash + ch) % TABLE_SIZE;
86          }
87          return hash;
88      }
89
90      void insert(const string &name, const string &phone)
91      {
92          int index = hashFunction(name);
93          table[index].push_back({name, phone});
94      }
95
96      bool search(const string &name, string &phone, int &comparisons)
97      {
98          int index = hashFunction(name);
99          comparisons = 0;
100         for (const auto &entry : table[index])
```

```
        {
            comparisons++;
            if (entry.name == name)
            {
                phone = entry.phone;
                return true;
            }
        }
        return false;
    }
};

int main()
{
    LinearProbingHashTable linearTable;
    ChainingHashTable chainingTable;

    int choice;
    string name, phone;

    do
    {
        cout << "\nTelephone Book Menu:\n";
        cout << "1. Insert\n2. Search\n3. Exit\nEnter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter name: ";
            cin >> name;
            cout << "Enter phone number: ";
            cin >> phone;
            linearTable.insert(name, phone);
            chainingTable.insert(name, phone);
            break;

        case 2:
            cout << "Enter name to search: ";
            cin >> name;
            int linearComparisons, chainingComparisons;
            if (linearTable.search(name, phone, linearComparisons))
            {
                cout << "Linear Probing: Found \"" << name << "\" with phone " <<
phone << " in " << linearComparisons << " comparisons.\n";
            }
            else
            {
                cout << "Linear Probing: \"" << name << "\" not found.\n";
            }
            if (chainingTable.search(name, phone, chainingComparisons))
```

```
            {
                cout << "Chaining: Found \"" << name << "\" with phone " << phone << " in " << chainingComparisons << " comparisons.\n";
            }
            else
            {
                cout << "Chaining: \"" << name << "\" not found.\n";
            }
            break;

        case 3:
            cout << "Exiting program.\n";
            break;

        default:
            cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 3);

    return 0;
}

```

Output :


D:\SE Computer\LAB CODES\DSA\DSA.exe
Telephone Book Menu:
1. Insert
2. Search
3. Exit
Enter your choice: 1
Enter name: Shiv
Enter phone number: 937071723
Telephone Book Menu:
1. Insert
2. Search
3. Exit
Enter your choice: 1
Enter name: Shankr
Enter phone number: 8888259240
Telephone Book Menu:
1. Insert
2. Search
3. Exit
Enter your choice: 1
Enter name: Pravin
Enter phone number: 94797572949
Telephone Book Menu:
1. Insert
2. Search
3. Exit
Enter your choice: 2
Enter name to search: Shiv
Linear Probing: Found "Shiv" with phone 937071723 in 1 comparisons.
Chaining: Found "Shiv" with phone 937071723 in 1 comparisons.
Telephone Book Menu:
1. Insert
2. Search
3. Exit
Enter your choice: 2
Enter name to search: Ram
Linear Probing: "Ram" not found.
Chaining: "Ram" not found.
Telephone Book Menu:
1. Insert
2. Search
3. Exit
Enter your choice: 3
Exiting program.