```
/*-------------------------------------------------------------------------------------------------------------
 Practical No: 01 - A C++ Program to implement Hash Table Data Structure
                    and Handle Collision using Linear and Quadratic Probing
                    and Compare no. of comarisons required for Searching a set of Keys.
        Data Used: Key = Client Name and Value = Telephone No.
        Type of Hashing = Closed Hashing.
-------------------------------------------------------------------------------------------------------------*/


                //.....Header Files
#include <iostream>
#include <cstring>
using namespace std;



                //......Create Hash Table of Size 10.
int Tablesize = 10;

struct HashTable
{
  char Key[10];                         //......Key : Client Name

  int Val;                          //......Value: Telephone Number

} HT1[10], HT2[10];             //......HT1 = Handles Collision by Linear Probing.
                //......HT2 = Handles Collision by Quadratic Probing.



                //......Function to initialise Keys and Values of Hash Table

void init ()
{
  int i;

  for (i = 0; i < Tablesize; i++)
    {
      strcpy (HT1[i].Key, ".....");
      HT1[i].Val = 0;
    }

  for (i = 0; i < Tablesize; i++)
    {
      strcpy (HT2[i].Key, ".....");
      HT2[i].Val = 0;
    }
}
```

```
                    //.....Function for Hash Function

int HashFun (char key[])                //.....Division Method
{
  int i, index, sum = 0;

  for (i = 0; key[i] != '\0'; i++)
    {
      sum = sum + key[i];
    }
  index = sum % Tablesize;    //.....index of Hash Table.

  return index;
}



                //.....Function for Insertion in HT1

void insert_HT1 (char Cname[], int Tele)     //.....insert(Key,Value)
{
  int i, id, index;

  index = HashFun (Cname);

  if (strcmp (HT1[index].Key, ".....") == 0)   //....No Collision
    {
      strcpy (HT1[index].Key, Cname);
      HT1[index].Val = Tele;
    }
  else                                  //....if Collision, check linearly for empty location
    {
      while (strcmp (HT1[index].Key, ".....") != 0)
            {
              index = (index + 1) % Tablesize;
            }

      strcpy (HT1[index].Key, Cname);
      HT1[index].Val = Tele;
    }
}


                //.....Function for Insertion in HT2

void insert_HT2 (char Cname[], int Tele)     //.....insert(Key,Value)
{
  int i, index;

  index = HashFun (Cname);

  if (strcmp (HT2[index].Key, ".....") == 0)   //....No Collision
```

```
    {
     strcpy (HT2[index].Key, Cname);
     HT2[index].Val = Tele;
    }
  else                                  //....if Collision, check linearly for empty location
    {
     i = 1;
     int id = index;
     while (strcmp (HT2[id].Key, ".....") != 0)
          {
           id = (index + i * i) % Tablesize;
           i++;
          }

     strcpy (HT2[id].Key, Cname);
     HT2[id].Val = Tele;
    }
}
```

//.....Function to display Hash Tables

```
void display ()
{
  int i;
  cout << "\n\n------ Hash Table 01 ------ ";
  cout << "\n\n Bucket (Key , Value)";
  for (i = 0; i < Tablesize; i++)
    {
     cout << "\n " << i << " - (" << HT1[i].Key << " , " << HT1[i].
          Val << ")";
    }

  cout << "\n\n------ Hash Table 02 ------ ";
  cout << "\n\n Bucket (Key , Value)";
  for (i = 0; i < Tablesize; i++)
    {
     cout << "\n " << i << " - (" << HT2[i].Key << " , " << HT2[i].
          Val << ")";
    }

}
```

//.....Function to Search Keys In Hash Table-01

```
void search_HT1 (char Cname[])
{
  int index = HashFun (Cname);
  int cnt = 1;
  int id = index;

  if (strcmp (HT1[id].Key, Cname) == 0)
```

```
    {
     cout<<"\n\t"<<HT1[id].Key<<" : "<<HT1[id].Val<<" : "<<cnt;
    }
  else                              //....if Collision, linearly Search_HT1
    {
     int i = 1;
     while (strcmp (HT1[id].Key, Cname) != 0)
          {
           id = (index + i) % Tablesize;

           i++;                     //i= 1,2,3,.....

           cnt++;                   // To count comparisons
          }

    cout<<"\n\t"<< HT1[id].Key<<" : "<<HT1[id].Val<<" : "<<cnt;
    }

}
```

//.....Function to Search Keys In Hash Table-02

```
void search_HT2 (char Cname[])
{
  int index = HashFun (Cname);
  int cnt = 1;
  int id = index;

  if (strcmp (HT2[id].Key, Cname) == 0)
    {
     cout<<"\n\t"<< HT2[id].Key<<" : "<<HT2[id].Val<<" : "<<cnt;
    }
  else                              //....if Collision, Search Quadraticly
    {
     int i = 1;
     while (strcmp (HT2[id].Key, Cname) != 0)
          {
           id = (index + i * i) % Tablesize;

           i++;                     //i= 1,2,3,.....

           cnt++;                   // To count comparisons
          }

     cout<<"\n\t"<< HT2[id].Key<<" : "<<HT2[id].Val<<" : "<<cnt;
    }

}
```

```
                                    //......Main Function
int main ()
{
 cout << "\n ----A C++ Program to implement Hash Table Data Structure----";

 init();                                    //......Intialise Hash Table-01,02.

                                            //......INSERT In Hash Table - 01
 insert_HT1 ("Amol", 915033);
 insert_HT1 ("Amit", 925033);
 insert_HT1 ("Ajay", 935033);
 insert_HT1 ("Sanjay", 945033);
 insert_HT1 ("Sanika", 955033);
 insert_HT1 ("Seeta", 965033);
 insert_HT1 ("Gita", 975033);
 insert_HT1 ("Babita", 985033);

                                    //......INSERT In Hash Table - 02
 insert_HT2 ("Amol", 915033);
 insert_HT2 ("Amit", 925033);
 insert_HT2 ("Ajay", 935033);
 insert_HT2 ("Sanjay", 945033);
 insert_HT2 ("Sanika", 955033);
 insert_HT2 ("Seeta", 965033);
 insert_HT2 ("Gita", 975033);
 insert_HT2 ("Babita", 985033);

 display ();
                                            //......SEARCH In Hash Table – 01


 cout<<"\n\n ----- Search in Hash Table-01 -----";

 cout<<"\n\t Client : Telephone : comparisons";
 search_HT1 ("Seeta");
 search_HT1 ("Sanika");
 search_HT1 ("Babita");

                                    //......SEARCH In Hash Table – 02

 cout<<"\n\n ----- Search in Hash Table-02 -----";

 cout<<"\n\t Client : Telephone : comparisons";
 search_HT2 ("Seeta");
 search_HT2 ("Sanika");
 search_HT2 ("Babita");

 return 0;

}
```

/*----------OUTPUT----------

----A C++ Program to implement Hash Table Data Structure----

------ Hash Table 01 ------

Bucket (Key , Value)
0 - (Sanika , 955033)
1 - (Gita , 975033)
2 - (Babita , 985033)
3 - (Amol , 915033)
4 - (Sanjay , 945033)
5 - (Amit , 925033)
6 - (..... , 0)
7 - (Gita , 975033)
8 - (Seeta , 965033)
9 - (Ajay , 935033)

------ Hash Table 02 ------

Bucket (Key , Value)
0 - (Sanika , 955033)
1 - (..... , 0)
2 - (Babita , 985033)
3 - (Amol , 915033)
4 - (Sanjay , 945033)
5 - (Amit , 925033)
6 - (..... , 0)
7 - (..... , 0)
8 - (Seeta , 965033)
9 - (Ajay , 935033)

----- Search in Hash Table-01 -----
Client : Telephone : comparisons
Seeta : 965033 : 1
Sanika : 955033 : 2
Babita : 985033 : 4

----- Search in Hash Table-02 -----
Client : Telephone : comparisons
Seeta : 965033 : 1
Sanika : 955033 : 2
Babita : 985033 : 46344

...Program finished with exit code 0
Press ENTER to exit console.
*/