//Assignment no 1

#include <iostream>

using namespace std;

class HashTable{

public:

int key;

int index;

};

class Hashing

{

private:

HashTable H[10];

public:

Hashing(){

for(int i=0;i<10;i++)

{

H[i].key=-1;

H[i].index=i;

}

}

void display();

void insert(int p\_ch);

int LinearP(int position);

int QuadraticP(int position);

};

void Hashing::display()

{

cout<<"Hash Table is"<<endl;

for(int i=0;i<10;i++)

{

cout<<H[i].key<<"\t\t\t\t\t"<<H[i].index<<endl;

}

}

void Hashing::insert(int p\_ch)

{

int pos;

int key1;

cout<<"Enter the Telephone number\n";

cin>>key1;

pos=key1%10;

if(H[pos].key==-1)

{

H[pos].key=key1;

}

else if(p\_ch==1){

int temp=LinearP(pos);

H[temp].key=key1;

}

else if(p\_ch==2){

int temp=QuadraticP(pos);

H[temp].key=key1;

}

}

int Hashing::LinearP(int position) {

for (int i = 0; i < 10; i++) {

int newPos = (position + i) % 10;

if (H[newPos].key == -1) {

return newPos;

}

}

return -1;

}

int Hashing::QuadraticP(int position) {

for (int i = 0; i < 10; i++) {

int newPos = (position + (i \* i)) % 10;

if (H[newPos].key == -1) {

return newPos;

}

}

return -1;

}

int main(){

Hashing obj;

int ch;

int p\_choice;

do{

cout<<"----MENU----\n";

cout<<"1. Insert\n";

cout<<"2. Display\n";

cout<<"3. Exit\n";

cout<<"Enter your choice\n";

cin>>ch;

if(ch==1){

cout<<"Enter the probing choice\n1. Linear Probing\n2. Quadratic Probing\n";

cin>>p\_choice;

}

else{

p\_choice=0;

}

switch(ch){

case 1:

{

obj.insert(p\_choice);

break;

}

case 2:

{

obj.display();

break;

}

case 3:

{

cout<<"End of program";

break;

}

default:

{

cout<<"Enter valid choice :\n";

}

}

}while(ch!=3);

}

output:

gescoe@gescoe-OptiPlex-3010:~/Desktop/SE-A-55$ g++ Hashing.cpp

gescoe@gescoe-OptiPlex-3010:~/Desktop/SE-A-55$ ./a.out

----MENU----

1. Insert
2. Display
3. Exit

Enter your choice

1

Enter the probing choice

1. Linear Probing
2. Quadratic Probing

1

Enter the Telephone number

951878884

----MENU----

1. Insert
2. Display
3. Exit

Enter your choice

1

Enter the probing choice

1. Linear Probing
2. Quadratic Probing

2

Enter the Telephone number

976700110

----MENU----

1. Insert
2. Display
3. Exit

Enter your choice

2

Hash Table is

976700110

* 1
* 1
* 1

951878884

* 1
* 1
* 1
* 1
* 1

----MENU----

1. Insert
2. Display
3. Exit

Enter your choice

3

End of [programgescoe@gescoe-OptiPlex-3010](mailto:programgescoe@gescoe-OptiPlex-3010):~/Desktop/SE-A-55$

### Theory for Hashing Code

This program demonstrates a **Hash Table** using different collision resolution techniques—**Linear Probing** and **Quadratic Probing**.

A **Hash Table** is a data structure that maps keys to values, using a **hash function** to compute the index of the array where the data should be stored. However, when two keys hash to the same index (a collision), we need a technique to resolve this collision. This code implements two common techniques for collision resolution:

1. **Linear Probing**: When a collision occurs, the algorithm checks the next slot in the hash table (i.e., moves one slot forward). If that slot is also occupied, it checks the next slot, and so on until an empty slot is found.
2. **Quadratic Probing**: In this case, the algorithm checks slots by increasing the distance in a quadratic manner. For example, if the first slot is occupied, it checks the 1st, 4th, 9th, etc., slots.

### Code Walkthrough

#### Classes

* **HashTable**: Represents an individual element in the hash table with a key (the telephone number) and an index (the position in the table).
* **Hashing**: The main class for managing the hash table. It holds an array H of HashTable objects and implements the logic for inserting, displaying, and resolving collisions using probing techniques.

#### Functions

1. **Constructor (**Hashing()**)**:
   * Initializes the hash table. Each element in the table is set to -1 for the key (indicating an empty slot), and the index is set from 0 to 9.
2. **Insert Function (**insert(int p\_ch)**)**:
   * Prompts the user for a telephone number and applies the hash function key % 10 to determine the position.
   * If the position is empty (i.e., key == -1), the key is inserted.
   * If the position is occupied, the program applies either **Linear Probing** or **Quadratic Probing** based on user input and inserts the key in the first available position.
3. **Linear Probing (**LinearP(int position)**)**:
   * This method resolves collisions by checking consecutive slots in the hash table starting from the given position until an empty slot is found.
4. **Quadratic Probing (**QuadraticP(int position)**)**:
   * Similar to linear probing, but instead of checking consecutive slots, it checks slots at increasing distances (i.e., position+12position + 1^2, position+22position + 2^2, etc.).
5. **Display Function (**display()**)**:
   * Displays the current contents of the hash table, showing both the key and index.

### Algorithm

1. **Insert**:
   * Take the key (telephone number).
   * Calculate the hash position: pos = key % 10.
   * If the slot at pos is empty, insert the key.
   * If the slot is full, apply **Linear Probing** or **Quadratic Probing** to find the next available slot.
2. **Linear Probing**:
   * For a given position pos, check the next available position by adding 1 to the index. Keep checking until an empty slot is found.
3. **Quadratic Probing**:
   * For a given position pos, check positions based on quadratic increments. For example, check positions: pos + 1^2, pos + 2^2, etc., until an empty slot is found.
4. **Display**:
   * Loop through the hash table and print the current state of each slot.

### Example Execution:

----MENU----

1. Insert

2. Display

3. Exit

Enter your choice

1

Enter the probing choice

1. Linear Probing

2. Quadratic Probing

1

Enter the Telephone number

951878884

----MENU----

1. Insert

2. Display

3. Exit

Enter your choice

1

Enter the probing choice

1. Linear Probing

2. Quadratic Probing

2

Enter the Telephone number

976700110

----MENU----

1. Insert

2. Display

3. Exit

Enter your choice

2

Hash Table is

976700110

• 1

• 1

• 1

951878884

• 1

• 1

• 1

• 1

• 1

----MENU----

1. Insert

2. Display

3. Exit

Enter your choice

3

End of program

### Key Observations from Output:

* The hash table is displayed after two insertions.
* The Linear Probing and Quadratic Probing choices determine how the collisions are resolved.
* The table prints the key values and their respective positions in the table, with collisions handled as per the chosen method.

### Code Summary

This program demonstrates the use of **hash tables** with collision resolution methods like **linear probing** and **quadratic probing**. It uses a simple hash function to determine the index and allows for dynamic collision handling by choosing the probing method at runtime. The program provides a good way to understand the basics of **hash table** operations and how collisions are managed in a table.