**Assignment no 1  
Title: Hash Table Implementation**

**Code:**

class hashtable:

def \_\_init\_\_(self):

self.m = int(input("enter the size of hash table:"))

self.hashTable = [None] \* self.m

self.elecount = 0

self.comparisons = 0

print(self.hashTable)

def hashFunction(self, key):

return key % self.m

def isfull(self):

if self.elecount == self.m:

return True

else:

return False

def linearprobr(self, key, data):

index = self.hashFunction(key)

compare = 0

while self.hashTable[index] is not None:

index = index + 1

compare = compare + 1

if index == self.m:

index = 0

if self.hashTable[index] is None:

self.hashTable[index] = [key, data]

self.elecount += 1

print("data inserted at", index)

print(self.hashTable)

print("no of comparisons=", compare)

return

def getlinear(self, key, data):

index = self.hashFunction(key)

while self.hashTable[index] is not None:

if self.hashTable[index] == [key, data]:

return index

index = (index + 1) % self.m

return None

def quadraticprobr(self, key, data):

index = self.hashFunction(key)

compare = 0

i = 1

while self.hashTable[index] is not None:

index = (index + i \* i) % self.m

compare += 1

i += 1

self.hashTable[index] = [key, data]

self.elecount += 1

print("data inserted at:", index)

print(self.hashTable)

print("no of comparisons=", compare)

def getQuadratic(self, key, data):

index = self.hashFunction(key)

i = 1

while self.hashTable[index] is not None:

if self.hashTable[index] == [key, data]:

return index

index = (index + i \* i) % self.m

i += 1

return None

def insertvialinear(self, key, data):

if self.isfull():

print("table is full.")

return False

index = self.hashFunction(key)

if self.hashTable[index] is None:

self.hashTable[index] = [key, data]

self.elecount += 1

print("data inserted at:", index)

print(self.hashTable)

else:

print("collision occured apply Linear method-")

self.linearprobr(key, data)

def insertviaQuadratic(self, key, data):

if self.isfull():

print("table is full.")

return False

index = self.hashFunction(key)

if self.hashTable[index] is None:

self.hashTable[index] = [key, data]

self.elecount += 1

print("data inserted at:", index)

print(self.hashTable)

else:

print("collision occured apply quadratic method-")

self.quadraticprobr(key, data)

def menu():

obj = hashtable()

ch = 0

while ch != 3:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("1.Linear Probe \*")

print("2.Quadratic Probe \*")

print("3.Exit")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

ch = int(input("Enter Choice:"))

if ch == 1:

ch2 = 0

while ch2 != 3:

print("\*\*1.Insert\*\*")

print("\*\*2.Search\*\*")

print("\*\*3.Exit\*\*")

ch2 = int(input("enter your choice:"))

if ch2 == 1:

a = int(input("enter the phone number:"))

b = str(input("enter name:"))

obj.insertvialinear(a, b)

elif ch2 == 2:

k = int(input("enter key to be searched:"))

b = str(input("enter name:"))

f = obj.getlinear(k, b)

if f is None:

print("key not found.")

else:

print("key found at:", f)

elif ch == 2:

ch2 = 0

obj1 = hashtable()

while ch2 != 3:

print("\*\*1.Insert\*\*")

print("\*\*2.Search\*\*")

print("\*\*3.Exit\*\*")

ch2 = int(input("enter your choice:"))

if ch2 == 1:

a = int(input("enter phone number:"))

b = str(input("enter name:"))

obj1.insertviaQuadratic(a, b)

elif ch2 == 2:

k = int(input("enter key to be searched:"))

b = str(input("enter name:"))

f = obj1.getQuadratic(k, b)

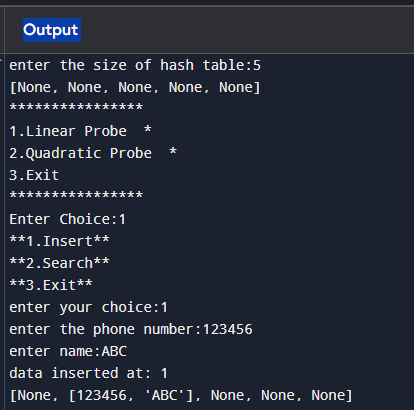
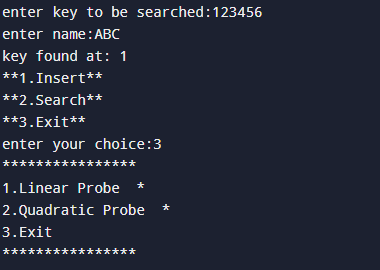
if f is None:

print("key not found.")

else:

print("key found at:", f)

menu()

**OUTPUT:  
** ****