ZEAL EDUCATION SOCIETY'S
ZEAL COLLEGE OF ENGINEERING AND RESEARCH
NARHE | PUNE -41 | INDIA

Department of Computer Engineering

Data Structures and Algorithms Laboratory

ASSIGNMENT NO: 1

NAME: SUJEET UDDHAV PAWAR

CLASS: SE-C

ROLL NO: S213009

Group A: Assignment No:01

Problem Statement:

Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client ‘s telephone Number. Make Use of two Collision Techniques and Compare them using number of Comparisons required to find a set of telephone numbers.

Program:

Filename – LinearProbing.py

```
# Program to implement Hashing with Linear Probing


from Record import Record


class hashTable:

    # initialize hash Table

    def __init__(self):

        self.size = int(input("Enter the Size of the hash table : "))

        # initialize table with all elements 0

        self.table = list(None for i in range(self.size))

        self.elementCount = 0
        self.comparisons = 0
    # method that checks if the hash table is full or not
    def isFull(self):
        if self.elementCount == self.size:
            return True
        else:
            return False
    # method that returns position for a given element
    def hashFunction(self, element):
        return element % self.size
    # method that inserts element into the hash table
    def insert(self, record):
        # checking if the table is full
        if self.isFull():
            print("Hash Table Full")
            return False
            isStored = False
            position = self.hashFunction(record.get_number())
            # checking if the position is empty
            if self.table[position] == None:
                self.table[position] = record
                print("Phone number of " + record.get_name() + " is at position " + str(position))
                isStored = True
                self.elementCount += 1

            # collision occured hence we do linear probing
            else:
                print("Collision has occured for " + record.get_name() + "'s phone number at position "
        + str(
                    position) + " finding new Position.")
                while self.table[position] != None:
                    position += 1
                    if position >= self.size:
                        position = 0

                self.table[position] = record

                print("Phone number of " + record.get_name() + " is at position " + str(position))

                isStored = True

                self.elementCount += 1

            return isStored

        # method that searches for an element in the table

        # returns position of element if found

        # else returns False

        def search(self, record):

            found = False

            position = self.hashFunction(record.get_number())

            self.comparisons += 1

            if (self.table[position] != None):

                if (self.table[position].get_name() == record.get_name() and self.table[
                    position].get_number() == record.get_number()):
```

```
                isFound = True

                print("Phone number found at position {} ".format(position) + " and total
comparisons are " + str(1))

                return position









            # if element is not found at position returned hash function

            else:

                position += 1

                if position >= self.size - 1:
                    position = 0

                while self.table[position] != None or self.comparisons <= self.size:

                    if (self.table[position].get_name() == record.get_name() and self.table[
                        position].get_number() == record.get_number()):
                        isFound = True

                        # i=0
                        i = self.comparisons + 1

                        print(
                            "Phone number found at position {} ".format(position) + " and total
comparisons are " + str(
                                i))

                        return position

                    position += 1

                    # print(position)

                    if position >= self.size - 1:
                        position = 0

                    # print(position)

                    self.comparisons += 1

                    # print(self.comparisons)

                if isFound == False:
                    print("Record not found")

                    return false

    # method to display the hash table

    def display(self):
        print("\n")

        for i in range(self.size):
            print("Hash Value: " + str(i) + "\t\t" + str(self.table[i]))

        print("The number of phonebook records in the Table are : " + str(self.elementCount))
```

Filename – Record.py

```
class Record:
    def __init__(self):
        self._name = None
        self._number = None

    def get_name(self):
        return self._name

    def get_number(self):
        return self._number

    def set_name(self,name):
        self._name = name

    def set_number(self,number):
        self._number = number

    def __str__(self):
        record = "Name: "+str(self.get_name())+"\t"+"\tNumber: "+str(self.get_number())
        return record
```

Filename – DoubleHashing.py

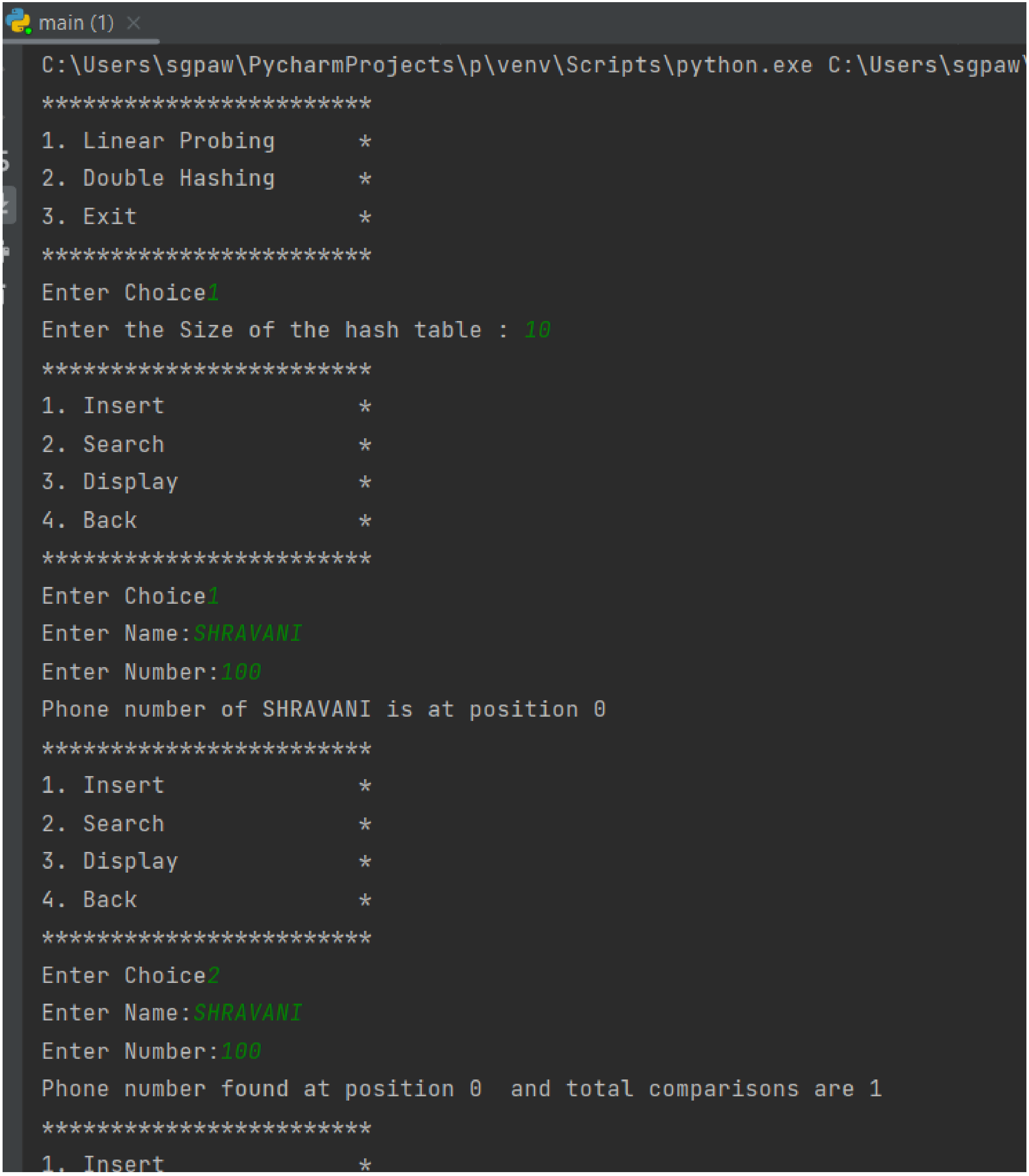
```
from Record import Record


class doubleHashTable:
    # initialize hash Table
    def __init__(self):
        self.size = int(input("Enter the Size of the hash table : "))

        # initialize table with all elements 0
        self.table = list(None for i in range(self.size))
        self.elementCount = 0
        self.comparisons = 0

    # method that checks if the hash table is full or not
    def isFull(self):
        if self.elementCount == self.size:
            return True
        else:
            return False

    # First hash function
    def h1(self, element):
        return element % self.size

    # Second hash function
    def h2(self, element):
        return 5 - (element % 5)

    # method to resolve collision by double hashing method
    def doubleHashing(self, record):
        posFound = False
        # limit variable is used to restrict the function from going into infinite loop
        # limit is useful when the table is 80% full
        limit = self.size
        i = 1
        # start a loop to find the position
        while i <= limit:
            # calculate new position by quadratic probing
            newPosition = (self.h1(record.get_number()) + i * self.h2(record.get_number())) %
self.size
            # if newPosition is empty then break out of loop and return new Position
            if self.table[newPosition] == None:
                posFound = True
                break
            else:
                # as the position is not empty increase i
                i += 1
        return posFound, newPosition

    # method that inserts element inside the hash table
    def insert(self, record):
        # checking if the table is full
        if self.isFull():
            print("Hash Table Full")
            return False

        posFound = False

        position = self.h1(record.get_number())

        # checking if the position is empty
        if self.table[position] == None:
            # empty position found , store the element and print the message
            self.table[position] = record
            print("Phone number of " + record.get_name() + " is at position " + str(position))
            isStored = True
            self.elementCount += 1

        # If collision occured
        else:
            print("Collision has occured for " + record.get_name() + "'s phone number at position "
+ str(
                position) + " finding new Position.")
            while not posFound:
                posFound, position = self.doubleHashing(record)
                if posFound:
                    self.table[position] = record
                    # print(self.table[position])
                    self.elementCount += 1
                    # print(position)
                    # print(posFound)
                    print("Phone number of " + record.get_name() + " is at position " + str(position))

        return posFound

    # searches for an element in the table and returns position of element if found else returns
False
    def search(self, record):
        found = False
        position = self.h1(record.get_number())
        self.comparisons += 1
    if (self.table[position] != None):
      if (self.table[position].get_name() == record.get_name()):
        print("Phone number found at position {}".format(position) + " and total
comparisons are " + str(1))
        return position

      # if element is not found at position returned hash function
      # then we search element using double hashing
      else:
        limit = self.size
        i = 1

        newPosition = position
        # start a loop to find the position
        while i <= limit:
          # calculate new position by double Hashing
          position = (self.h1(record.get_number()) + i * self.h2(record.get_number())) %
self.size
          self.comparisons += 1
          # if element at newPosition is equal to the required element

          if (self.table[position] != None):
            if self.table[position].get_name() == record.get_name():

              found = True
              break

            elif self.table[position].get_name() == None:
              found = False
              break
```

```
                        else:
                            # as the position is not empty increase i
                            i += 1

            if found:
                print("Phone number found at position {}".format(position) + " and total comparisons are " + str(i + 1))
            # return position
            else:
                print("Record not Found")
                return found

                # method to display the hash table

    def display(self):
        print("\n")
        for i in range(self.size):
            print("Hash Value: " + str(i) + "\t\t" + str(self.table[i]))
        print("The number of phonebook records in the Table are : " + str(self.elementCount))
```

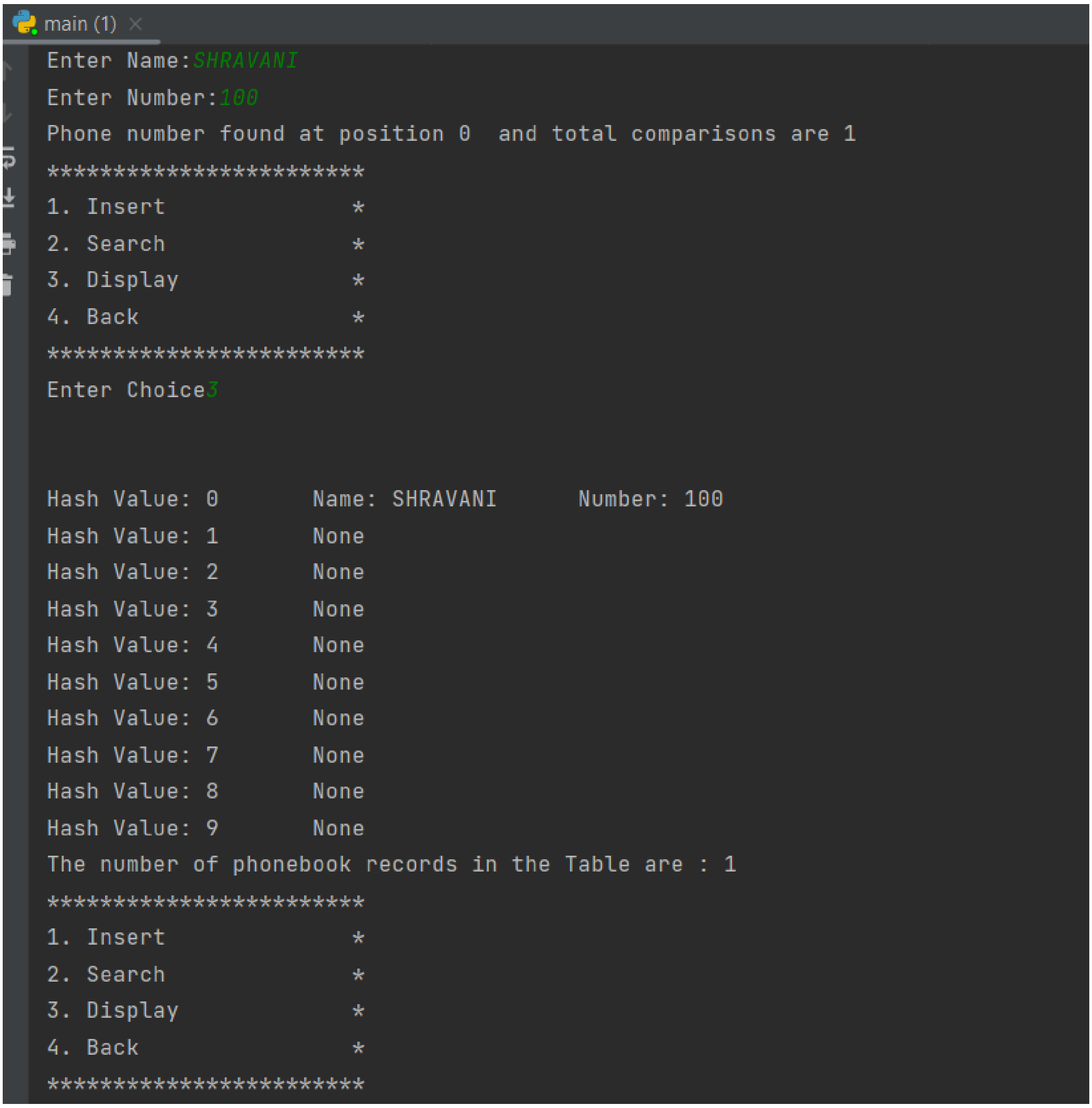
Filename – main.py

```
from LinearProbing import hashTable
from Record import Record
from DoubleHashing import doubleHashTable



def input_record():
    record = Record()
    name = input("Enter Name:")
    number = int(input("Enter Number:"))
    record.set_name(name)
    record.set_number(number)
    return record



choice1 = 0
while (choice1 != 3):
    print("************************")
    print("1. Linear Probing      *")
    print("2. Double Hashing      *")
    print("3. Exit                *")
    print("************************")

    choice1 = int(input("Enter Choice"))
    if choice1 > 3:
        print("Please Enter Valid Choice")

    if choice1 == 1:
        h1 = hashTable()
        choice2 = 0
        while (choice2 != 4):
            print("************************")
            print("1. Insert              *")
            print("2. Search              *")
            print("3. Display             *")
            print("4. Back                *")
            print("************************")

            choice2 = int(input("Enter Choice"))
            if choice2 > 4:
                print("Please Enter Valid Choice")
            if (choice2 == 1):
                record = input_record()
                h1.insert(record)

            elif (choice2 == 2):
                record = input_record()
                position = h1.search(record)

            elif (choice2 == 3):
                h1.display()




    elif choice1 == 2:
        h2 = doubleHashTable()
        choice2 = 0
        while (choice2 != 4):
            print("************************")
            print("1. Insert                *")
            print("2. Search                *")
            print("3. Display               *")
            print("4. Back                  *")
            print("************************")

            choice2 = int(input("Enter Choice"))
            if choice2 > 4:
                print("Please Enter Valid Choice")

            if (choice2 == 1):
                record = input_record()
        h2.insert(record)

    elif (choice2 == 2):
        record = input_record()
        position = h2.search(record)

    elif (choice2 == 3):
        h2.display()
```

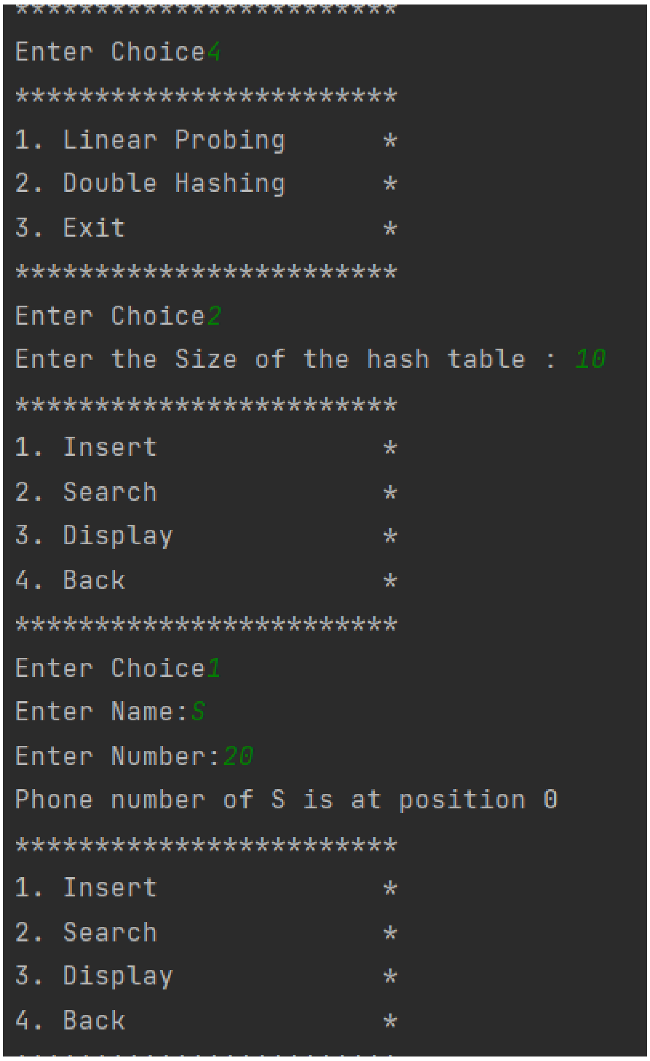
Output –



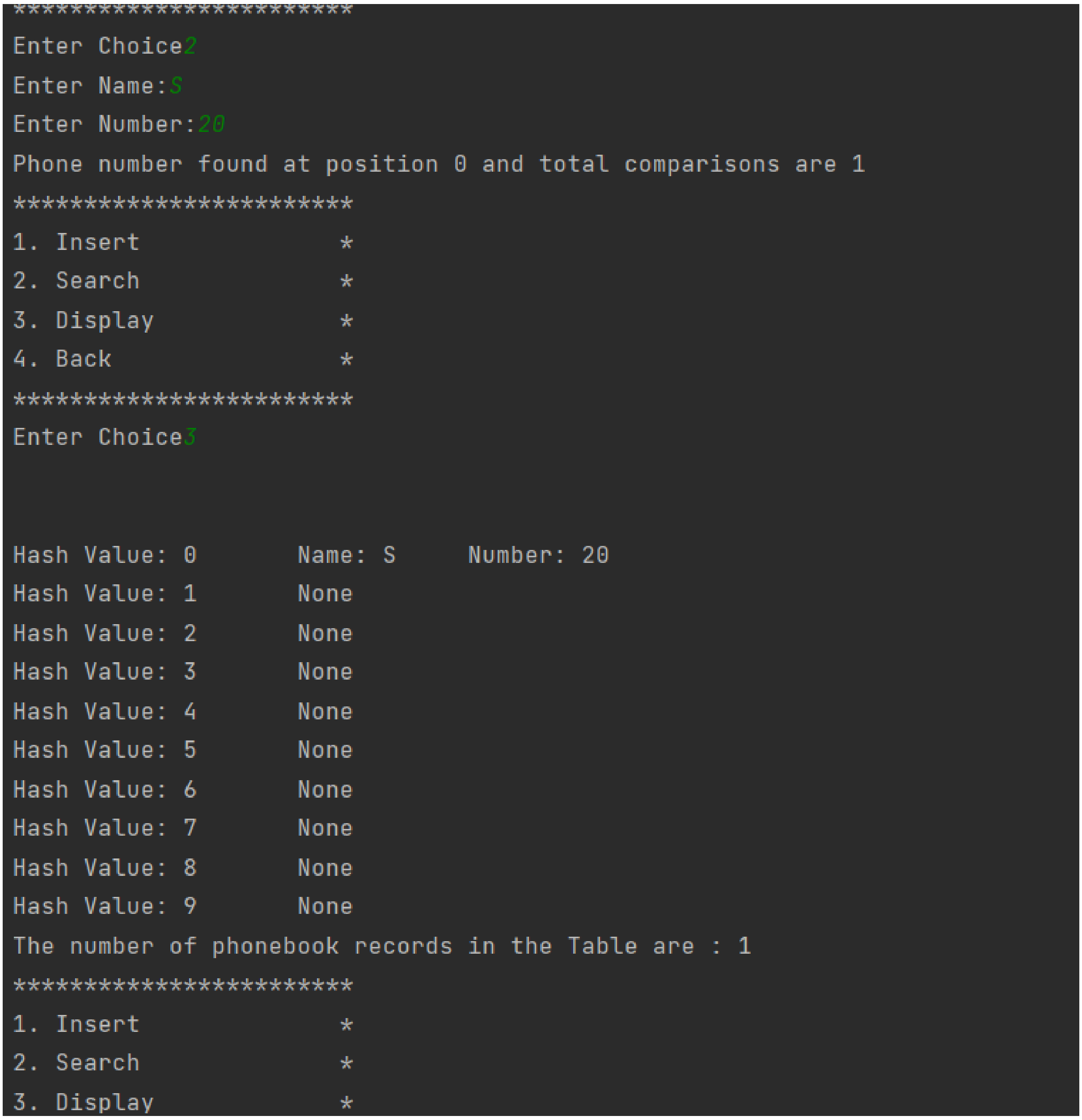
```
C:\Users\sgpaw\PycharmProjects\p\venv\Scripts\python.exe C:\Users\sgpaw\
************************
1. Linear Probing      *
2. Double Hashing      *
3. Exit                *
************************
Enter Choice1
Enter the Size of the hash table : 10
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
Enter Choice1
Enter Name:SHRAVANI
Enter Number:100
Phone number of SHRAVANI is at position 0
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
Enter Choice2
Enter Name:SHRAVANI
Enter Number:100
Phone number found at position 0  and total comparisons are 1
************************
1. Insert              *
```



```
Enter Name:SHRAVANI
Enter Number:100
Phone number found at position 0  and total comparisons are 1
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
Enter Choice3


Hash Value: 0       Name: SHRAVANI      Number: 100
Hash Value: 1       None
Hash Value: 2       None
Hash Value: 3       None
Hash Value: 4       None
Hash Value: 5       None
Hash Value: 6       None
Hash Value: 7       None
Hash Value: 8       None
Hash Value: 9       None
The number of phonebook records in the Table are : 1
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
```



```
************************
Enter Choice4
************************
1. Linear Probing      *
2. Double Hashing      *
3. Exit                *
************************
Enter Choice2
Enter the Size of the hash table : 10
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
Enter Choice1
Enter Name:S
Enter Number:20
Phone number of S is at position 0
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
```



```
************************
Enter Choice2
Enter Name:S
Enter Number:20
Phone number found at position 0 and total comparisons are 1
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
Enter Choice3


Hash Value: 0       Name: S     Number: 20
Hash Value: 1       None
Hash Value: 2       None
Hash Value: 3       None
Hash Value: 4       None
Hash Value: 5       None
Hash Value: 6       None
Hash Value: 7       None
Hash Value: 8       None
Hash Value: 9       None
The number of phonebook records in the Table are : 1
************************
1. Insert              *
2. Search              *
3. Display             *
```