Name: Om Chandrakant Bhavsar

Class: SE-A

Roll No: COSA75

Practical No. 1

```
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
    def isFull(self):
        if self.elementCount == self.size:
            return True
        else:
            return False

    def hashFunction(self, element):
        return element % self.size
    def insert(self, record):
        if self.isFull():
            print("Hash Table Full")
```

```
            return False
        isStored = False

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

import Record from Record
```

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

        self.table = list(None for i in range(self.size))
        self.elementCount = 0
        self.comparisons = 0

    def isFull(self):
        if self.elementCount == self.size:
            return True
        else:
            return False

    def h1(self, element):
        return element % self.size

    def h2(self, element):
        return 5-(element % 5)

    def doubleHashing(self, record):
        posFound = False

        limit = self.size
        i = 1
        while i <= limit:
            # calculate new position by quadratic probing
            newPosition = (self.h1(record.get_number()) + i*self.h2(record.get_number())) %
self.size
            if self.table[newPosition] == None:
                posFound = True
                break
            else:
                # as the position is not empty increase i
                i += 1
        return posFound, newPosition

    def insert(self, record):
        if self.isFull():
            print("Hash Table Full")
            return False

        posFound = False

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

```
        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

        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

 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

            else:
                limit = self.size
                i = 1

                newPosition = 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(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

    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))



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

def input_record():
    record = Record()
    name = raw_input("Enter Name:")
    number = int(raw_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:

```
************************
1. Linear Probing      *
2. Double Hashing      *
3. Exit                *
************************
Enter Choice2
Enter the Size of the hash table : 5
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
Enter Choice1
Enter Name: B
Enter Number:0
Phone number of B is at position 0
************************
1. Insert              *
2. Search              *
3. Display             *
4. Back                *
************************
Enter Choice1
Enter Name: A
Enter Number:1
Phone number A of  is at position 1
************************
```

1. Insert *
2. Search *
3. Display *
4. Back *
****
Enter Choice1
Enter Name: S
Enter Number:2
Phone number of S is at position 2
****
1. Insert *
2. Search *
3. Display *
4. Back *
****
Enter Choice1
Enter Name: D
Enter Number:3
Phone number of D is at position 3
****
1. Insert *
2. Search *
3. Display *
4. Back *
****
Enter Choice1
Enter Name: F
Enter Number:4
Phone number of F is at position 4
****
1. Insert *
2. Search *
3. Display *
4. Back *
****
Enter Choice3

Hash Value: 0 Name: B Number: 0
Hash Value: 1 Name: A Number: 1
Hash Value: 2 Name: S Number: 2
Hash Value: 3 Name: D Number: 3
Hash Value: 4 Name: F Number: 4
The number of phonebook records in the Table are : 5
****
1. Insert *

2. Search *
3. Display *
4. Back *
****
Enter Choice2
Enter Name: B
Enter Number:0
Phone number found at position 0 and total comparisons are 1
****
1. Insert *
2. Search *
3. Display *
4. Back *
****
Enter Choice2
Enter Name: F
Enter Number:4
Phone number found at position 4 and total comparisons are 1
****
1. Insert *
2. Search *
3. Display *
4. Back *
****
Enter Choice