**Code:**

class HashTable1:

    """linear Probing Without Replacement"""

    def \_\_init\_\_(self, size: int) -> None:

        self.record = []

        self.m = size

        for \_ in range(size):

            self.record.append([0, ""])

    def display\_table(self) -> None:

        print("Hash Table Using Linear Probing Without Replacement")

        for i in range(len(self.record)):

            print(i, self.record[i])

    def hash\_function(self, tel: int) -> int:

        key = (tel % self.m)

        return key

    def generate\_table(self, recs: list[list]) -> None:

        for rec in recs:

            self.insert\_rec(rec)

    def insert\_rec(self, rec: list) -> None:

        key = self.hash\_function(rec[0])

        if (self.record[key][0] == 0):

            self.record[key][0] = rec[0]

            self.record[key][1] = rec[1]

        else:

            while (self.record[key][0] != 0):

                key = ((key+1) % self.m)

            self.record[key][0] = rec[0]

            self.record[key][1] = rec[1]

class HashTable2:

    """linear Probing With Replacement"""

    def \_\_init\_\_(self, size: int) -> None:

        self.record = []

        self.m = size

        for \_ in range(size):

            self.record.append([0, "", -1])

    def display\_table(self) -> None:

        print("Hash Table Using Linear Probing With Replacement")

        for i in range(len(self.record)):

            print(i, self.record[i])

    def hash\_function(self, tel: int) -> int:

        key = (tel % self.m)

        return key

    def generate\_table(self, recs: list[list]) -> None:

        for rec in recs:

            self.insert\_rec(rec)

    def insert\_rec(self, rec: list) -> None:

        key = self.hash\_function(rec[0])

        if (self.record[key][0] == 0):

            self.record[key][0] = rec[0]

            self.record[key][1] = rec[1]

            self.record[key][2] = -1

        else:

            if (self.hash\_function(self.record[key][0]) == key):

                last\_elmt = key

                while (self.record[last\_elmt][2] != -1):

                    last\_elmt = self.record[last\_elmt][2]

                k = last\_elmt

                while (self.record[k][0] != 0):

                    k = ((k+1) % self.m)

                self.record[last\_elmt][2] = k

                self.record[k][0] = rec[0]

                self.record[k][1] = rec[1]

                self.record[k][2] = -1

            else:

                for i in range(self.m):

                    if (self.record[i][2] == key):

                        prev\_link\_key = i

                old\_rec\_tel = self.record[key][0]

                old\_rec\_name = self.record[key][1]

                old\_rec\_link = self.record[key][2]

                self.record[key][0] = rec[0]

                self.record[key][1] = rec[1]

                self.record[key][2] = -1

                k = key

                while (self.record[k][0] != 0):

                    k = ((k+1) % self.m)

                self.record[prev\_link\_key][2] = k

                self.record[k][0] = old\_rec\_tel

                self.record[k][1] = old\_rec\_name

                self.record[k][2] = old\_rec\_link

class HashTable3:

    """Double hashing"""

    def \_\_init\_\_(self, size: int) -> None:

        self.record = []

        self.m = size

        for \_ in range(size):

            self.record.append([0, ""])

        if (size <= 3):

            self.prime = size

        else:

            prime = [2, 3]

            for i in range(size):

                for j in prime:

                    if (i % j == 0):

                        p = False

                        break

                if (p):

                    prime.append(i)

            self.prime = prime[-1]

    def hash1(self, key: int) -> int:

        return (key % self.m)

    def hash2(self, key: int) -> int:

        return (self.prime - (key % self.prime))

    def display\_table(self) -> None:

        print("Hash Table Using Double Hashing")

        for i in range(len(self.record)):

            print(i, self.record[i])

    def generate\_table(self, recs: list[list]) -> None:

        for rec in recs:

            self.insert\_rec(rec)

    def insert\_rec(self, rec: list) -> None:

        i = 0

        key = self.hash1(rec[0])

        k2 = (key + i\*self.hash2(rec[0])) % self.m

        while (self.record[k2][0] != 0):

            k2 = (key + i\*self.hash2(rec[0])) % self.m

            i += 1

        self.record[k2][0] = rec[0]

        self.record[k2][1] = rec[1]

def input\_records(n: int) -> list[list]:

    records = []

    for i in range(n):

        name = input("Enter Name of the person:")

        tel = int(input("Enter Telephone Number:"))

        records.append([tel, name])

    return records

n = int(input("Enter the total number of records:"))

records = input\_records(n)

ch = 1

while(ch != 5):

    print("MENU")

    print("1. Input Records")

    print("2. Use linear Probing Without Replacement")

    print("3. Use linear Probing With Replacement")

    print("4. Use Double Hashing")

    print("5. Exit")

    ch = int(input("Enter your choice:"))

    match (ch):

        case 1:

            n = int(input("Enter the total number of records:"))

            records = input\_records(n)

        case 2:

            t1 = HashTable1(n)

            t1.generate\_table(records)

            t1.display\_table()

        case 3:

            t2 = HashTable2(n)

            t2.generate\_table(records)

            t2.display\_table()

        case 4:

            t3 = HashTable3(n)

            t3.generate\_table(records)

            t3.display\_table()

        case 5:

            print("Thank you !")

        case default:

            print("Invalid Choice")

**Output:**

\*\*\*\*\*\*\* Book Information \*\*\*\*\*\*\*

Enter the total number of records:5

Enter Name of the person:khushal

Enter Telephone Number:10

Enter Name of the person:ajay

Enter Telephone Number:5

Enter Name of the person:sumit

Enter Telephone Number:8

Enter Name of the person:yash

Enter Telephone Number:4

Enter Name of the person:vijay

Enter Telephone Number:9

MENU

1. Input Records

2. Use linear Probing Without Replacement

3. Use linear Probing With Replacement

4. Use Double Hashing

5. Exit

Enter your choice:2

Hash Table Using Linear Probing Without Replacement

0 [10, 'khushal']

1 [5, 'ajay']

2 [9, 'vijay']

3 [8, 'sumit']

4 [4, 'yash']

MENU

1. Input Records

2. Use linear Probing Without Replacement

3. Use linear Probing With Replacement

4. Use Double Hashing

5. Exit

Enter your choice:3

Hash Table Using Linear Probing With Replacement

0 [10, 'khushal', 1]

1 [5, 'ajay', -1]

2 [9, 'vijay', -1]

3 [8, 'sumit', -1]

4 [4, 'yash', 2]

MENU

1. Input Records

2. Use linear Probing Without Replacement

3. Use linear Probing With Replacement

4. Use Double Hashing

5. Exit

Enter your choice:4

Hash Table Using Double Hashing

0 [10, 'khushal']

1 [5, 'ajay']

2 [9, 'vijay']

3 [8, 'sumit']

4 [4, 'yash']