ASSIGNMENT 6

**Name: Soham Makarand Limaye**

**Roll No: 222038**

**PRN: 22111058**

**Batch: B2**

**Aim:**The aim of this program is to implement a hash table data structure in Java using linear probing collision resolution method.

**Objective**: To create a hash table that can store key-value pairs and perform operations like put, get, and remove efficiently. Linear probing collision resolution method will be used to handle collisions that occur when multiple keys hash to the same index.

**Theory :**

A hash table is a data structure that uses a hash function to map keys to indices in an array. Each index in the array can store a key-value pair.Collisions can occur when multiple keys have the same hash value and are mapped to the same index. If a collision occurs at index i, the next available index (i+1) is checked, and if it is empty, the key-value pair is stored at that index. If index (i+1) is also occupied, the next available index (i+2) is checked, and so on until an empty index is found.

**Code :**

import java.util.\*; class

LinearProbingHashTable

{

private int currentSize, maxSize; private

String[] keys; private String[] vals; public

LinearProbingHashTable(int capacity)

{

currentSize = 0; maxSize = capacity; keys = new

String[maxSize]; vals = new

String[maxSize];

}

public void makeEmpty()

{

currentSize = 0; keys = new String[maxSize]; vals = new

String[maxSize];

}

public int getSize()

{

return currentSize;

}

public boolean isFull()

{

return currentSize == maxSize;

}

public boolean isEmpty()

{

return getSize() == 0;

}

public boolean contains(String key)

{

return get(key) != null;

}

private int hash(String key)

{

return key.hashCode() % maxSize;

}

public void insert(String key, String val) { int tmp = hash(key); int i = tmp; do

{ if (keys[i] == null)

{

keys[i] = key; vals[i] = val; currentSize++; return;

}

if (keys[i].equals(key))

{ vals[i] = val; return;

}

i = (i + 1) % maxSize;

} while (i != tmp);

}

public String get(String key) { int i = hash(key); while (keys[i] != null)

{

if (keys[i].equals(key))

return vals[i]; i = (i + 1) % maxSize;

} return null;

}

public void remove(String key)

{

if (!contains(key)) return; int i = hash(key); while

(!key.equals(keys[i])) i = (i + 1) % maxSize; keys[i] = vals[i] = null; for (i = (i + 1) % maxSize; keys[i] != null; i = (i + 1) % maxSize)

{

String tmp1 = keys[i], tmp2 = vals[i];

keys[i] = vals[i] = null; currentSize--; insert(tmp1, tmp2);

}

currentSize--;

}

public void printHashTable()

{

System.out.println("\nHash Table: "); for (int i = 0; i < maxSize; i++) if (keys[i] != null)

System.out.println(keys[i] +" "+ vals[i]);

System.out.println();

}

}

public class Main

{

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in); System.out.println("Hash Table Test\n\n");

System.out.println("Enter size");

LinearProbingHashTable lpht = new

LinearProbingHashTable(scan.nextInt() ); char ch; do

{

System.out.println("\nHash Table Operations\n");

System.out.println("1. insert ");

System.out.println("2. remove");

System.out.println("3. get");

System.out.println("4. clear"); System.out.println("5. size"); int choice = scan.nextInt(); switch (choice)

{

case 1 :

System.out.println("Enter key and value"); lpht.insert(scan.next(), scan.next() ); break; case 2 :

System.out.println("Enter key"); lpht.remove( scan.next() ); break; case 3 :

System.out.println("Enter key");

System.out.println("Value = "+ lpht.get( scan.next() )); break; case 4 :

lpht.makeEmpty();

System.out.println("Hash Table Cleared\n"); break; case 5 :

System.out.println("Size = "+ lpht.getSize() ); break; default :

System.out.println("Wrong Entry \n "); break;

}

lpht.printHashTable();

System.out.println("\nDo you want to continue (Type y or n)

\n");

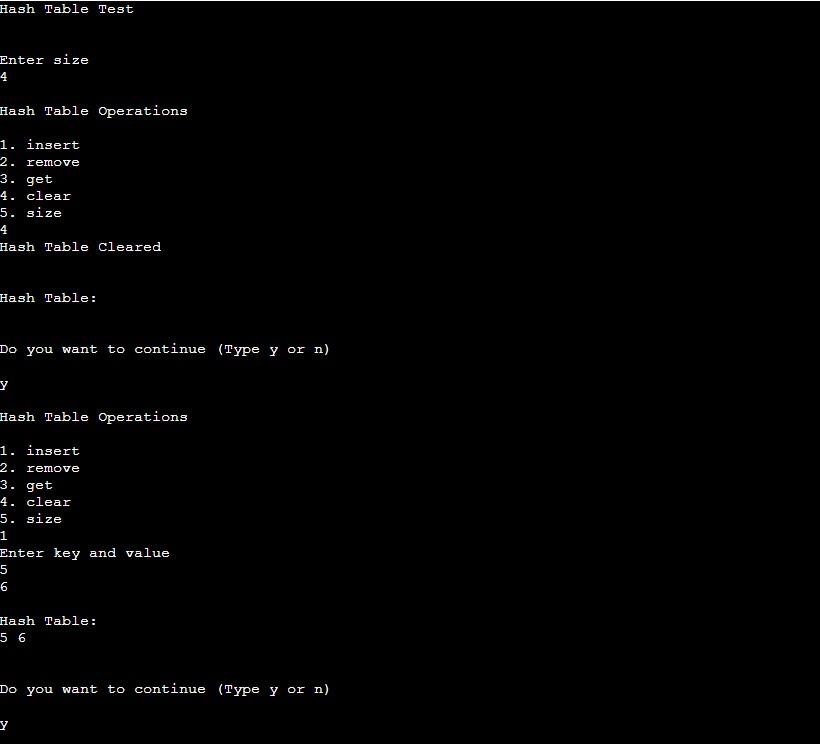
ch = scan.next().charAt(0);

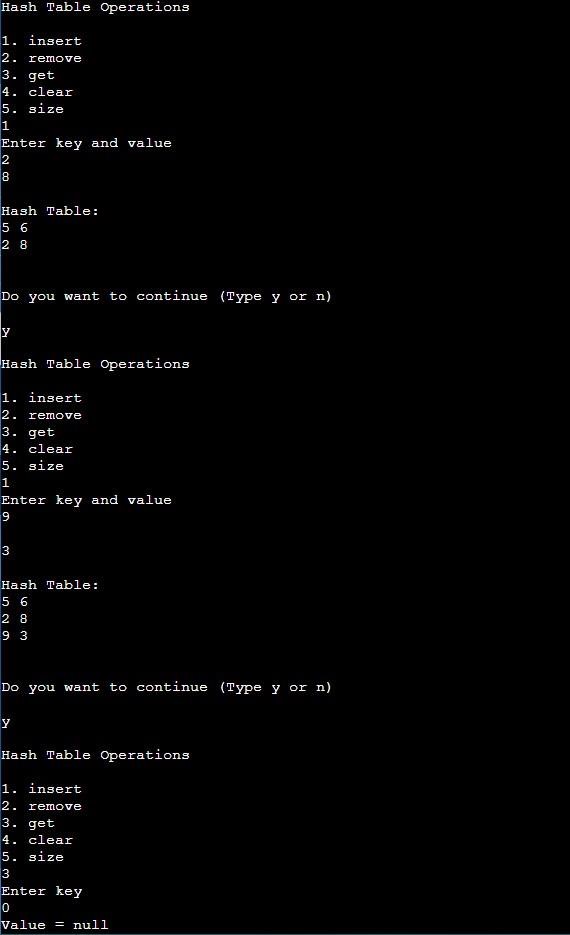
} while (ch == 'Y'|| ch == 'y');

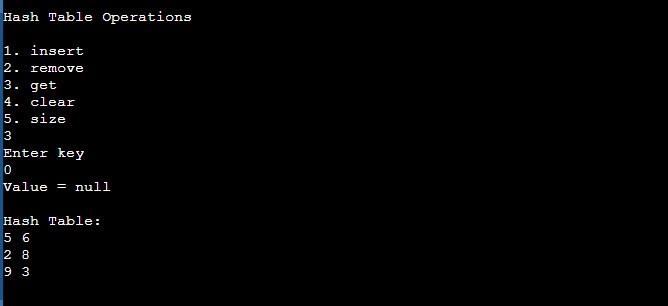
}

}

**Output :**







**Conclusion:**

The hash table implemented using the linear probing collision resolution method can efficiently store and retrieve key-value pairs. The linear probing method is a simple and effective way to handle collisions in hash tables. It may lead to clustering, where groups of keys are stored together, and the search time may increase.