### **Rubric for Hash Function (5 Marks)**

| **Criteria** | **Description** | **Marks** |
| --- | --- | --- |
| **Correct ASCII Calculation** | Properly computes the ASCII value of each character in the key. | 2 |
| **Weighted Sum** | Multiplies / Sum each ASCII value by its position and calculates the weighted sum correctly. | 2 |
| **Modulo Operation** | Applies the modulus operation with the table size to calculate the index correctly. | 1 |

### **Rubric for Insert Method (10 Marks)**

| **Criteria** | **Description** | **Marks** |
| --- | --- | --- |
| **Key-Value Pair Handling** | Inserts key-value pairs into the correct index in the table. | 2 |
| **Collision Handling** | Implements forward chaining correctly to handle collisions. | 3 |
| **Duplicate Key Handling** | Updates the value for an existing key instead of creating a duplicate entry. | 2 |
| **Use of Linked List for Chaining** | Properly links new nodes to handle collisions using linked lists. | 3 |

**Python (SET A) -**

class ListNode:

def \_\_init\_\_(self, key, value):

self.key = key

self.value = value

self.next = None

class HashTable:

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

self.size = size

self.table = [None] \* size

def hash\_function(self, key):

total = 0

position = 1

for char in key:

ascii\_value = 0

for bit in char:

ascii\_value += ord(bit)

total += position \* ascii\_value

position += 1

return total % self.size

def insert(self, key, value):

index = self.hash\_function(key)

new\_node = ListNode(key, value)

if not self.table[index]:

self.table[index] = new\_node

else:

current = self.table[index]

while current:

if current.key == key:

current.value = value

return

if current.next is None:

break

current = current.next

current.next = new\_node

def display(self):

for i in range(self.size):

print(f"Index {i}: ", end="")

if self.table[i]:

current = self.table[i]

while current:

print(f"({current.key}: {current.value})", end=" -> ")

current = current.next

print("None")

else:

print("None")

# Test case for handling collisions and rehashing

ht = HashTable(4)

ht.insert("AB", "HR")

ht.insert("BA", "Finance")

ht.insert("XY", "Engineering")

ht.insert("YX", "Marketing")

print("\nHash table after insertions with collisions:")

ht.display()

# Updating a key

ht.insert("AB", "Admin")

print("\nHash table after update:")

ht.display()

**Python Solve (SET B)**

def hash\_function(self, key):

total = 0

position = 1

for char in key:

ascii\_value = 0

for bit in char:

ascii\_value += ord(bit)

total += position + ascii\_value

position += 1

return total % self.size

**JAVA Set - A**

class ListNode {

String key;

String value;

ListNode next;

// Constructor for ListNode

public ListNode(String key, String value) {

this.key = key;

this.value = value;

this.next = null;

}

}

class EmployeeHashTable {

ListNode[] hashTable;

// Constructor that initializes the HashTable array

// DO NOT change this Constructor

public EmployeeHashEmployeeHashTableTable(int size) {

this.hashTable = new ListNode[size];

}

// This method is used to insert key-value pairs

// You need to COMPLETE this method

public void insertKeyValue(Object[] keyValuePair) {

String key = (String) keyValuePair[0];

String value = (String) keyValuePair[1];

int index = hashFunction(key);

ListNode newNode = new ListNode(key, value);

if (hashTable[index] == null) {

hashTable[index] = newNode;

} else {

ListNode current = hashTable[index];

while (current != null) {

if (current.key.equals(key)) {

current.value = value; // Update value if key already exists

return;

}

if (current.next == null) {

break;

}

current = current.next;

}

current.next = newNode; // Add to the end of the chain

}

}

// This method basically prints the HashTable

// DO NOT change this method// hashTable[] :: is the HashTable array that stores the ListNode objects

public void printHashTable() {

for (int i = 0; i < hashTable.length; i++) {

System.out.print("Index " + i + ": ");

ListNode current = hashTable[i];

if (current == null) {

System.out.println("null");

} else {

while (current != null) {

System.out.print("(" + current.key + ": " + current.value + ") -> ");

current = current.next;

}

System.out.println("null");

}

}

}

// You need to COMPLETE this method

// Write this method before insertKeyValue method since you'll need it there

private int hashFunction(String key) {

int total = 0;

for (int i = 0; i < key.length(); i++) {

total += (i + 1) \* key.charAt(i);

}

return total % hashTable.length;

}

}

// DO NOT TOUCH THIS TESTER CLASS

public class EmployeeHashTableTester {

// DO NOT TOUCH THIS TESTER MAIN METHOD

public static void main(String[] args) {

// DO NOT TOUCH ANY CODE BELOW

Object[][] keyValuePair = {

{"E123", "HR"},

{"BA", "Finance"},

{"XY", "Engineering"},

{"YX", "Marketing"},

{"E123", "Admin"} // Update existing key

};

int totalEntries = 4;

EmployeeHashTable ht = new EmployeeHashTable(totalEntries);

for (Object[] entry : keyValuePair) {

ht.insertKeyValue(entry);

}

System.out.println("\n:::HASH TABLE OUTPUT:::");

ht.printHashTable();

}

}

**JAVA Set - B**

**Code same as previous one, just change in formula in hashFunction**

private int hashFunction(String key) {

int total = 0;

for (int i = 0; i < key.length(); i++) {

total += (i + 1) + key.charAt(i);

}

return total % hashTable.length;

}