WEEK 16 - Implementation of Collision Resolution Techniques

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define TABLE_SIZE 10
typedef struct Node {
int data;
struct Node* next;
} Node;
Node* createNode(int data) {
Node* newNode = (Node*)malloc(sizeof(Node));
if (newNode == NULL) {
printf("Memory allocation failed!\n");
exit(1);
}
newNode->data = data;
newNode->next = NULL;
return newNode;
}
int hashFunction(int key) {
return key % TABLE_SIZE;
}
Node* insertOpenAddressing(Node* table[], int key) {
int index = hashFunction(key);
while (table[index] != NULL) {
index = (index + 1) % TABLE_SIZE;
}
table[index] = createNode(key);
return table[index];
```

```
}
void displayHashTable(Node* table[]) {
printf("Hash Table:\n");
for (int i = 0; i < TABLE_SIZE; i++) {
printf("%d: ", i);
Node* current = table[i];
while (current != NULL) {
printf("%d ", current->data);
current = current->next;
}
printf("\n");
}
}
Node* insertClosedAddressing(Node* table[], int key) {
int index = hashFunction(key);
if (table[index] == NULL) {
table[index] = createNode(key);
} else {
Node* newNode = createNode(key);
newNode->next = table[index];
table[index] = newNode;
}
return table[index];
}
int rehashFunction(int key, int attempt) {
// Double Hashing Technique
return (hashFunction(key) + attempt * (7 - (key % 7))) % TABLE_SIZE;
}
Node* insertRehashing(Node* table[], int key) {
int index = hashFunction(key);
int attempt = 0;
```

```
while (table[index] != NULL) {
attempt++;
index = rehashFunction(key, attempt);
}
table[index] = createNode(key);
return table[index];
}
int main() {
Node* openAddressingTable[TABLE_SIZE] = {NULL};
Node* closedAddressingTable[TABLE_SIZE] = {NULL};
Node* rehashingTable[TABLE_SIZE] = {NULL};
// Insert elements into hash tables
insertOpenAddressing(openAddressingTable, 10);
insertOpenAddressing(openAddressingTable, 20);
insertOpenAddressing(openAddressingTable, 5);
insertClosedAddressing(closedAddressingTable, 10);
insertClosedAddressing(closedAddressingTable, 20);
insertClosedAddressing(closedAddressingTable, 5);
insertRehashing(rehashingTable, 10);
insertRehashing(rehashingTable, 20);
insertRehashing(rehashingTable, 5);
// Display hash tables
displayHashTable(openAddressingTable);
displayHashTable(closedAddressingTable);
displayHashTable(rehashingTable);
return 0;
}
```

OUTPUT

```
Hash Table:
0: 10
1: 20
```

```
2:
3:
4:
5: 5
6:
7:
8:
9:
Hash Table:
0: 20 10
1:
2:
3:
4:
5: 5
6:
7:
8:
9:
Hash Table:
0: 10
1: 20
2:
3:
4:
5: 5
6:
7:
8:
9:
```