

1.HASHING IN C

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#define TABLE_SIZE 10
```

```
// Node structure for linked list
```

```
typedef struct Node {
```

```
    char *key;
```

```
    struct Node *next;
```

```
} Node;
```

```
// Hash table structure
```

```
typedef struct HashTable {
```

```
    Node *table[TABLE_SIZE];
```

```
} HashTable;
```

```
// Hash function
```

```
unsigned int hash(const char *key) {
```

```
    unsigned long hash = 5381;
```

```
    int c;
```

```
    while ((c = *key++)) {
```

```
        hash = ((hash << 5) + hash) + c;
```

```
    }
```

```
    return hash % TABLE_SIZE;
```

```
}
```

```
// Create a new node
```

```
Node* create_node(const char *key) {  
    Node *new_node = (Node *)malloc(sizeof(Node));  
    new_node->key = strdup(key); // Duplicate the key  
    new_node->next = NULL;  
    return new_node;  
}
```

// Initialize the hash table

```
void init_table(HashTable *ht) {  
    for (int i = 0; i < TABLE_SIZE; i++) {  
        ht->table[i] = NULL;  
    }  
}
```

// Insert a key into the hash table

```
void insert(HashTable *ht, const char *key) {  
    unsigned int index = hash(key);  
    Node *new_node = create_node(key);  
  
    // Insert at the beginning of the linked list  
    new_node->next = ht->table[index];  
    ht->table[index] = new_node;  
}
```

// Search for a key in the hash table

```
int search(HashTable *ht, const char *key) {  
    unsigned int index = hash(key);  
    Node *current = ht->table[index];  
  
    while (current != NULL) {  
        if (strcmp(current->key, key) == 0) {
```

```

        return 1; // Key found
    }

    current = current->next;
}

return 0; // Key not found
}

// Delete a key from the hash table
void delete(HashTable *ht, const char *key) {
    unsigned int index = hash(key);
    Node *current = ht->table[index];
    Node *prev = NULL;

    while (current != NULL) {
        if (strcmp(current->key, key) == 0) {
            if (prev == NULL) {
                // Removing the first node in the list
                ht->table[index] = current->next;
            } else {
                prev->next = current->next;
            }

            free(current->key);
            free(current);
            return;
        }

        prev = current;
        current = current->next;
    }
}

```

```
}
```

```
// Print the hash table
```

```
void print_table(HashTable *ht) {  
    for (int i = 0; i < TABLE_SIZE; i++) {  
        Node *current = ht->table[i];  
        printf("Bucket %d: ", i);  
  
        while (current != NULL) {  
            printf("%s -> ", current->key);  
            current = current->next;  
        }  
        printf("NULL\n");  
    }  
}
```

```
// Free the hash table
```

```
void free_table(HashTable *ht) {  
    for (int i = 0; i < TABLE_SIZE; i++) {  
        Node *current = ht->table[i];  
        while (current != NULL) {  
            Node *temp = current;  
            current = current->next;  
            free(temp->key);  
            free(temp);  
        }  
    }  
}
```

```
int main() {  
    HashTable ht;
```

```
init_table(&ht);

insert(&ht, "apple");
insert(&ht, "banana");
insert(&ht, "grape");

printf("Hash table after inserts:\n");
print_table(&ht);

printf("Searching for 'banana': %s\n", search(&ht, "banana") ? "Found" : "Not Found");
printf("Searching for 'orange': %s\n", search(&ht, "orange") ? "Found" : "Not Found");

delete(&ht, "banana");
printf("Hash table after deleting 'banana':\n");
print_table(&ht);

free_table(&ht);
return 0;
}
```

Output:

Hash table after inserts:

Bucket 0: banana -> NULL

Bucket 1: NULL

Bucket 2: NULL

Bucket 3: NULL

Bucket 4: grape -> NULL

Bucket 5: NULL

Bucket 6: NULL

Bucket 7: apple -> NULL

Bucket 8: NULL

Bucket 9: NULL

Searching for 'banana': Found

Searching for 'orange': Not Found

Hash table after deleting 'banana':

Bucket 0: NULL

Bucket 1: NULL

Bucket 2: NULL

Bucket 3: NULL

Bucket 4: grape -> NULL

Bucket 5: NULL

Bucket 6: NULL

Bucket 7: apple -> NULL

Bucket 8: NULL

Bucket 9: NULL