```c
#include <stdio.h>
typedef struct Entry {
    char key[256];
    int value;
    struct Entry* next;
} Entry;
typedef struct HashTable {
    Entry* table[TABLE_SIZE];
} HashTable;
unsigned int hash(const char* key) {
    unsigned long int hash_val = 0;
    int i = 0;
    while (key[i] != '\0') {
        hash_val = (hash_val << 5) + key[i];
        i++;
    }
    return hash_val % TABLE_SIZE;
}
void initTable(HashTable* ht) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        ht->table[i] = NULL;
    }
}
void insert(HashTable* ht, const char* key, int value) {
    unsigned int index = hash(key);
    Entry* new_entry = malloc(sizeof(Entry));
```

```
    Entry* new_entry = malloc(sizeof(Entry));
    if (new_entry == NULL) {
        perror("Failed to allocate memory");
        exit(EXIT_FAILURE);
    }
    strcpy(new_entry->key, key);
    new_entry->value = value;
    new_entry->next = ht->table[index];
    ht->table[index] = new_entry;

nt search(HashTable* ht, const char* key) {
    unsigned int index = hash(key);
    Entry* entry = ht->table[index];
    while (entry != NULL) {
        if (strcmp(entry->key, key) == 0) {
            return entry->value;
        }
        entry = entry->next;
    }
    return -1;

oid delete(HashTable* ht, const char* key) {
    unsigned int index = hash(key);
    Entry* entry = ht->table[index];
    Entry* prev = NULL;
    while (entry != NULL) {
```

```
        if (strcmp(entry->key, key) == 0) {
            if (prev == NULL) {
                ht->table[index] = entry->next;
            } else {
                prev->next = entry->next;
            }
            free(entry);
            return;
        }
        prev = entry;
        entry = entry->next;
    }
}
void freeTable(HashTable* ht) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        Entry* entry = ht->table[i];
        while (entry != NULL) {
            Entry* prev = entry;
            entry = entry->next;
            free(prev);
        }
    }
}
```

```c
int main() {
    HashTable ht;
    initTable(&ht);
    insert(&ht, "key1", 10);
    insert(&ht, "key2", 20);
    insert(&ht, "key3", 30);
    printf("Value for 'key1': %d\n", search(&ht, "key1"));
    printf("Value for 'key2': %d\n", search(&ht, "key2"));
    printf("Value for 'key3': %d\n", search(&ht, "key3"));
    delete(&ht, "key2");
    printf("Value for 'key2' after deletion: %d\n", search(&ht, "key2"));
    freeTable(&ht);
    return 0;
}
```

Close