#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_SIZE 26

struct TrieNode {

struct TrieNode \*children[ALPHABET\_SIZE];

int isEndOfWord;

};

struct TrieNode\* createNode() {

struct TrieNode \*node = (struct TrieNode \*)malloc(sizeof(struct TrieNode));

node->isEndOfWord = 0;

for (int i = 0; i < ALPHABET\_SIZE; i++) {

node->children[i] = NULL;

}

return node;

}

void insert(struct TrieNode \*root, const char \*word) {

struct TrieNode \*node = root;

for (int i = 0; word[i] != '\0'; i++) {

int index = tolower(word[i]) - 'a';

if (index < 0 || index >= ALPHABET\_SIZE) {

printf("Error: Only lowercase alphabetic characters are allowed.\n");

return;

}

if (!node->children[index]) {

node->children[index] = createNode();

}

node = node->children[index];

}

node->isEndOfWord = 1;

printf("Word '%s' inserted successfully.\n", word);

}

int hasChildren(struct TrieNode \*node) {

for (int i = 0; i < ALPHABET\_SIZE; i++) {

if (node->children[i]) return 1;

}

return 0;

}

int deleteHelper(struct TrieNode \*node, const char \*word, int depth) {

if (!node) return 0;

if (depth == strlen(word)) {

if (node->isEndOfWord) {

node->isEndOfWord = 0;

if (!hasChildren(node)) {

return 1;

}

return 0;

}

return 0;

}

int index = tolower(word[depth]) - 'a';

if (index < 0 || index >= ALPHABET\_SIZE) {

printf("Error: Only lowercase alphabetic characters are allowed.\n");

return 0;

}

if (deleteHelper(node->children[index], word, depth + 1)) {

free(node->children[index]);

node->children[index] = NULL;

return !node->isEndOfWord && !hasChildren(node);

}

return 0;

}

void deleteWord(struct TrieNode \*root, const char \*word) {

if (deleteHelper(root, word, 0)) {

printf("Word '%s' deleted successfully.\n", word);

} else {

printf("Word '%s' not found in the Trie.\n", word);

}

}

void displayHelper(struct TrieNode \*node, char \*buffer, int depth) {

if (node->isEndOfWord) {

buffer[depth] = '\0';

printf("%s\n", buffer);

}

for (int i = 0; i < ALPHABET\_SIZE; i++) {

if (node->children[i]) {

buffer[depth] = i + 'a';

displayHelper(node->children[i], buffer, depth + 1);

}

}

}

void display(struct TrieNode \*root) {

char buffer[100];

printf("Words in the Trie:\n");

displayHelper(root, buffer, 0);

}

void freeTrie(struct TrieNode \*node) {

for (int i = 0; i < ALPHABET\_SIZE; i++) {

if (node->children[i]) {

freeTrie(node->children[i]);

}

}

free(node);

}

int main() {

struct TrieNode \*root = createNode();

int choice;

char word[100];

while (1) {

printf("\nTrie Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter word to insert: ");

scanf("%s", word);

insert(root, word);

break;

case 2:

printf("Enter word to delete: ");

scanf("%s", word);

deleteWord(root, word);

break;

case 3:

display(root);

break;

case 4:

freeTrie(root);

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid choice! Please try again.\n");

}

}

return 0;

}

Trie Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter word to insert: a

Word 'a' inserted successfully.

Trie Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter word to insert: b

Word 'b' inserted successfully.

Trie Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter word to insert: c

Word 'c' inserted successfully.

Trie Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter word to insert: s

Word 's' inserted successfully.

Trie Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 2

Enter word to delete: s

Word 's' not found in the Trie.

Trie Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 3

Words in the Trie:

a

b

c

Trie Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 4

Exiting program.