Roll NO:27

NAME : Mali Anjali Prakash

DIV: FYMCA-A

Assignment no -12

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_SIZE 26

#define MAX\_WORD\_LENGTH 100

// Trie Node Definition

typedef struct TrieNode {

    struct TrieNode\* children[ALPHABET\_SIZE];

    int isEndOfWord;

} TrieNode;

// Function to create a new Trie node

TrieNode\* createNode() {

    TrieNode\* newNode = (TrieNode\*)malloc(sizeof(TrieNode));

    newNode->isEndOfWord = 0;

    // Initialize all children to NULL

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

        newNode->children[i] = NULL;

    }

    return newNode;

}

// Function to insert a word into the Trie

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

    TrieNode\* currentNode = root;

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

        int index = tolower(word[i]) - 'a'; // Convert character to lowercase and map it to an index

        if (currentNode->children[index] == NULL) {

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

        }

        currentNode = currentNode->children[index];

    }

    currentNode->isEndOfWord = 1;

}

// Function to check if a word matches the pattern

void searchPattern(TrieNode\* root, char\* pattern, int length, char\* word, int level) {

    if (level == length) {

        // If we have reached the length of the word, print it if it is a valid word

        word[level] = '\0'; // Null-terminate the string

        if (root->isEndOfWord) {

            printf("%s\n", word);

        }

        return;

    }

    if (pattern[level] == '?') {

        // Wildcard case: Try all possible characters

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

            if (root->children[i] != NULL) {

                word[level] = 'a' + i; // Set the character

                searchPattern(root->children[i], pattern, length, word, level + 1);

            }

        }

    } else {

        // Specific character case: Only go down the matching child

        int index = tolower(pattern[level]) - 'a';

        if (root->children[index] != NULL) {

            word[level] = pattern[level];

            searchPattern(root->children[index], pattern, length, word, level + 1);

        }

    }

}

// Main function to run the program

int main() {

    TrieNode\* root = createNode();

    char pattern[MAX\_WORD\_LENGTH];

    int wordLength, numKnownChars;

    // Predefined dictionary of words

    const char\* dictionary[] = {

        "frost", "foist", "first", "feast", "feist", "faust", "cast", "fast", "last", "mast"

    };

    int dictSize = sizeof(dictionary) / sizeof(dictionary[0]);

    // Insert all dictionary words into the Trie

    for (int i = 0; i < dictSize; i++) {

        insert(root, dictionary[i]);

    }

    // User input

    printf("Enter the word length: ");

    scanf("%d", &wordLength);

    printf("Enter the number of known characters: ");

    scanf("%d", &numKnownChars);

    printf("Enter the pattern (use '?' for unknown letters): ");

    scanf("%s", pattern);

    // Validate pattern length

    if (strlen(pattern) != wordLength) {

        printf("Pattern length doesn't match the specified word length.\n");

        return 1;

    }

    // Array to store current word

    char word[MAX\_WORD\_LENGTH];

    // Print all matching words

    printf("Suggested words:\n");

    searchPattern(root, pattern, wordLength, word, 0);

    return 0; }

