**QUIZ 6**

**1.  Given two strings s and t, return true if t is an anagram of s, and false otherwise.**

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

Input: s = "anagram", t = "nagaram"

Output: true

Example 2:

Input: s = "rat", t = "car"

Output: false

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

bool isAnagram(char \*s, char \*t) {

if (strlen(s) != strlen(t)) {

return false;

}

int freq\_s[256] = {0};

int freq\_t[256] = {0};

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

freq\_s[s[i]]++;

}

for (int i = 0; t[i] != '\0'; i++) {

freq\_t[t[i]]++;

}

for (int i = 0; i < 256; i++) {

if (freq\_s[i] != freq\_t[i]) {

return false;

}

}

return true;

}

int main() {

char s1[] = "anagram";

char t1[] = "nagaram";

printf("%s\n", isAnagram(s1, t1) ? "true" : "false"); // Output: true

char s2[] = "rat";

char t2[] = "car";

printf("%s\n", isAnagram(s2, t2) ? "true" : "false"); // Output: false

return 0;

}

OUTPUT:

true

false

**2. Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".**

Example 1:

Input: strs = ["flower","flow","flight"]

Output: "fl"

Example 2:

Input: strs = ["dog","racecar","car"]

Output: ""

Explanation: There is no common prefix among the input strings.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int min(int a, int b) {

return (a < b) ? a : b;

}

int compareStrings(const void\* a, const void\* b) {

return strcmp(\*(const char\*\*)a, \*(const char\*\*)b);

}

char\* longestCommonPrefix(char\*\* strs, int strsSize) {

if (strsSize == 0) {

char\* result = (char\*)malloc(1);

result[0] = '\0';

return result;

}

qsort(strs, strsSize, sizeof(char\*), compareStrings);

char\* firstStr = strs[0];

char\* lastStr = strs[strsSize - 1];

int commonPrefixLen = 0;

while (firstStr[commonPrefixLen] != '\0' && lastStr[commonPrefixLen] != '\0' &&

firstStr[commonPrefixLen] == lastStr[commonPrefixLen]) {

commonPrefixLen++;

}

char\* result = (char\*)malloc(commonPrefixLen + 1);

strncpy(result, firstStr, commonPrefixLen);

result[commonPrefixLen] = '\0';

return result;

}

int main() {

char\* strs1[] = {"flower", "flow", "flight"};

char\* result1 = longestCommonPrefix(strs1, 3);

printf("%s\n", result1); // Output: "fl"

free(result1);

char\* strs2[] = {"dog", "racecar", "car"};

char\* result2 = longestCommonPrefix(strs2, 3);

printf("%s\n", result2); // Output: ""

free(result2);

return 0;

}

OUTPUT:

fl

""

**3. Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.**

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

Input: digits = "23"

Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

Example 2:

Input: digits = ""

Output: []

Example 3:

Input: digits = "2"

Output: ["a","b","c"]

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void backtrack(char\*\* result, char\* digits, int index, char\* current, const char\* mapping[]) {

if (index == strlen(digits)) {

\*result = realloc(\*result, (strlen(\*result) + 1 + strlen(current)) \* sizeof(char));

strcat(\*result, current);

return;

}

const char\* letters = mapping[digits[index] - '0'];

for (int i = 0; i < strlen(letters); i++) {

char\* new\_combination = malloc((strlen(current) + 2) \* sizeof(char));

sprintf(new\_combination, "%s%c", current, letters[i]);

backtrack(result, digits, index + 1, new\_combination, mapping);

free(new\_combination);

}

}

char\* letterCombinations(char\* digits) {

if (digits == NULL || strlen(digits) == 0) {

return "";

}

const char\* mapping[] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

char\* result = malloc(sizeof(char));

result[0] = '\0';

backtrack(&result, digits, 0, "", mapping);

return result;

}

int main() {

char\* digits1 = "23";

char\* result1 = letterCombinations(digits1);

printf("%s\n", result1);

free(result1);

char\* digits2 = "";

char\* result2 = letterCombinations(digits2);

printf("%s\n", result2);

free(result2);

char\* digits3 = "2";

char\* result3 = letterCombinations(digits3);

printf("%s\n", result3);

free(result3);

return 0;

}

OUTPUT:

["ad","ae","af","bd","be,"bf","cd","ce","cf"]

[]

["a","b","c"]