**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.

**CODE:**

#include <stdio.h>

#include <string.h>

int main() {

char s[100], t[100];

printf("Enter the first string: ");

scanf("%s", s);

printf("Enter the second string: ");

scanf("%s", t);

int lenS = strlen(s);

int lenT = strlen(t);

if (lenS != lenT) {

printf("%s and %s are not anagrams.\n", s, t);

} else {

int countS[256] = {0};

int countT[256] = {0};

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

countS[s[i]]++;

}

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

countT[t[i]]++;

}

int areAnagrams = 1;

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

if (countS[i] != countT[i]) {

areAnagrams = 0;

break;

}

}

if (areAnagrams) {

printf("True");

} else {

printf("False");

}

}

return 0;

}

**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 "".**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

int n,i,j;

printf("Enter the number of strings: ");

scanf("%d", &n);

char\*\* strs = (char\*\*)malloc(n \* sizeof(char\*));

for (i = 0; i < n; i++) {

printf("Enter string %d: ", i + 1);

strs[i] = (char\*)malloc(100 \* sizeof(char));

scanf("%s", strs[i]);

}

if (n == 0) {

printf("Longest Common Prefix: \"\"\n");

} else {

int minLen = strlen(strs[0]);

for (i = 1; i < n; i++) {

int len = strlen(strs[i]);

if (len < minLen) {

minLen = len;

}

}

int commonLen = 0;

for (i = 0; i < minLen; i++) {

char currentChar = strs[0][i];

for (j = 1; j < n; j++) {

if (strs[j][i] != currentChar) {

commonLen = i;

break;

}

}

}

char\* result = malloc((commonLen + 1) \* sizeof(char));

strncpy(result, strs[0], commonLen);

result[commonLen] = '\0';

printf("Longest Common Prefix: %s\n", result);

free(result);

}

for (i = 0; i < n; i++) {

free(strs[i]);

}

free(strs);

return 0;

}

**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.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* mapping[] = {

"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"

};

int main() {

char digits[100];

printf("Enter digits (2 to 9): ");

scanf("%s", digits);

int n = strlen(digits);

int maxCombinations = 1;

int i;

for (i = 0; i < n; i++) {

int digit = digits[i] - '0';

maxCombinations \*= strlen(mapping[digit]);

}

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

for (i = 0; i < maxCombinations; i++) {

result[i] = (char\*)malloc((n + 1) \* sizeof(char));

result[i][n] = '\0';

}

int returnSize = 0;

char current[n + 1];

current[n] = '\0';

if (n > 0) {

int stack[n];

memset(stack, 0, sizeof(stack));

while (stack[0] < strlen(mapping[digits[0] - '0'])) {

for (i = 0; i < n; i++) {

int digit = digits[i] - '0';

current[i] = mapping[digit][stack[i]];

}

strcpy(result[returnSize], current);

returnSize++;

int j = n - 1;

while (j >= 0 && stack[j] == strlen(mapping[digits[j] - '0']) - 1) {

stack[j] = 0;

j--;

}

if (j >= 0) {

stack[j]++;

} else {

break;

}

}

}

printf("Possible letter combinations:\n");

for (i = 0; i < returnSize; i++) {

printf("%s\n", result[i]);

free(result[i]);

}

free(result);

return 0;

}