Day-3

Quiz-2

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

CODE:

```
#include <stdio.h>
#include <string.h>
int areAnagrams(const char *s, const char *t) {
    // Check if the lengths of the strings are different
    if (strlen(s) != strlen(t)) {
        return 0; // Not anagrams
    }
    int count[26] = {0};
    for (int i = 0; s[i] != '\0'; ++i) {
        count[s[i] - 'a']++;
    }
    for (int i = 0; t[i] != '\0'; ++i) {
        count[t[i] - 'a']--;
    }
    for (int i = 0; i < 26; ++i) {
        if (count[i] != 0) {
            return 0; // Not anagrams
        }
    }

    return 1;
}

int main() {
    char s[100], t[100];
    printf("Enter the first string: ");
    scanf("%s", s);
    printf("Enter the second string: ");
```

```
    scanf("%s", t);
    printf("Result: %s\n", areAnagrams(s, t) ? "true" : "false");

    return 0;
}
```

OUTPUT:

Example:1

```
Enter the first string: anagram
Enter the second string: nagaram
Result: true
```

Example:2

```
Enter the first string: rat
Enter the second string: car
Result: 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.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char* longestCommonPrefix(char** strs, int strsSize) {
    // Check if the array is empty
    if (strsSize == 0) {
        char* result = (char*)malloc(1);
        result[0] = '\0';
        return result;
    }
for (int i = 0; strs[0][i] != '\0'; ++i) {
        for (int j = 1; j < strsSize; ++j) {
            if (strs[j][i] != strs[0][i]) {
```

```
            char* result = (char*)malloc(i + 1);
            strncpy(result, strs[0], i);
            result[i] = '\0';
            return result;
        }
    }
  }
  return strdup(strs[0]);
}

int main() {
  int n;
  printf("Enter the number of strings: ");
  scanf("%d", &n);
  char** strs = (char**)malloc(n * sizeof(char*));
 printf("Enter the strings:\n");
  for (int i = 0; i < n; ++i) {
    strs[i] = (char*)malloc(100);
    scanf("%s", strs[i]);
  }
  char* result = longestCommonPrefix(strs, n);
  printf("Longest Common Prefix: \"%s\"\n", result);
  for (int i = 0; i < n; ++i) {
    free(strs[i]);
  }
  free(strs);
  free(result);

  return 0;
}
```

OUTPUT:

Example 1:

```
Enter the number of strings: 3
Enter the strings:
flower
flow
flight
Longest Common Prefix: "fl"
```

Example 2:

```
Enter the number of strings: 3
Enter the strings:
dog
racecar
car
Longest Common Prefix: ""
```

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"]

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char *digit_mapping[] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
void backtrack(char *digits, int index, char *path, char **result, int *resultSize) {
    if (digits[index] == '\0') {
        path[index] = '\0';
        result[*resultSize] = strdup(path);
        (*resultSize)++;
        return;
    }
    int current_digit = digits[index] - '0';
    char *letters = digit_mapping[current_digit];
    for (int i = 0; letters[i] != '\0'; i++) {

        path[index] = letters[i];
        backtrack(digits, index + 1, path, result, resultSize);
    }
}

char **letterCombinations(char *digits, int *returnSize) {
    int n = strlen(digits);
    if (n == 0) {
        *returnSize = 0;
        return NULL;
    }
    int maxCombinations = 1;
    for (int i = 0; i < n; i++) {
        int digit = digits[i] - '0';
        maxCombinations *= strlen(digit_mapping[digit]);
    }
    char **result = (char **)malloc(sizeof(char *) * maxCombinations);
    *returnSize = 0;
    char *path = (char *)malloc(sizeof(char) * (n + 1));
    backtrack(digits, 0, path, result, returnSize);
    free(path);
    return result;
}

int main() {
    char *digits1 = "23";
    int returnSize1;
    char **result1 = letterCombinations(digits1, &returnSize1);

    printf("Output 1:\n");
    for (int i = 0; i < returnSize1; i++) {
        printf("%s\n", result1[i]);
        free(result1[i]);
```

```
    }
    free(result1);

    char *digits2 = "";
    int returnSize2;
    char **result2 = letterCombinations(digits2, &returnSize2);

    printf("\nOutput 2:\n");
    for (int i = 0; i < returnSize2; i++) {
        printf("%s\n", result2[i]);
        free(result2[i]);
    }
    free(result2);

    char *digits3 = "2";
    int returnSize3;
    char **result3 = letterCombinations(digits3, &returnSize3);

    printf("\nOutput 3:\n");
    for (int i = 0; i < returnSize3; i++) {
        printf("%s\n", result3[i]);
        free(result3[i]);
    }
    free(result3);

    return 0;
}
```

OUTPUT: