```
1. #include <stdio.h>
#include <string.h>

// Function to check if two strings are anagrams
int areAnagrams(char *s, char *t) {
    int sLen = strlen(s);
    int tLen = strlen(t);

    // If lengths are different, they can't be anagrams
    if (sLen != tLen) {
        return 0; // false
    }

    // Initialize an array to store the count of each character
    int count[26] = {0};

    // Increment count for characters in string s
    for (int i = 0; i < sLen; i++) {
        count[s[i] - 'a']++;
    }

    // Decrement count for characters in string t
    for (int i = 0; i < tLen; i++) {
        count[t[i] - 'a']--;
    }

    // If all counts are zero, the strings are anagrams
    for (int i = 0; i < 26; i++) {
        if (count[i] != 0) {
            return 0; // false
        }
    }

    return 1; // true
}

int main() {
    char s1[] = "anagram";
    char t1[] = "nagaram";
    printf("Example 1: %s\n", areAnagrams(s1, t1) ? "true" : "false");

    char s2[] = "rat";
    char t2[] = "car";
    printf("Example 2: %s\n", areAnagrams(s2, t2) ? "true" : "false");

    return 0;
}
```

2. #include <stdio.h>

```
#include <string.h>

// Function to find the longest common prefix
char* longestCommonPrefix(char **strs, int strsSize) {
    // If the array is empty, return an empty string
    if (strsSize == 0) {
        char* result = "";
        return result;
    }

    // Initialize the result string with the first string in the array
    char* result = strs[0];

    // Iterate through the array of strings
    for (int i = 1; i < strsSize; i++) {
        int j = 0;

        // Compare characters of the current string with the corresponding characters in the result string
        while (result[j] != '\0' && strs[i][j] != '\0' && result[j] == strs[i][j]) {
            j++;
        }

        // Null-terminate the result string at the common prefix
        result[j] = '\0';
    }

    return result;
}

int main() {
    char *strs1[] = {"flower", "flow", "flight"};
    printf("Example 1: %s\n", longestCommonPrefix(strs1, 3));

    char *strs2[] = {"dog", "racecar", "car"};
    printf("Example 2: %s\n", longestCommonPrefix(strs2, 3));

    return 0;
}
3. #include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to generate letter combinations
void generateCombinations(char *digits, char **letterMap, int index, char *current, char **result, int *count) {
    // If all digits are processed, add the current combination to the result
    if (digits[index] == '\0') {
        result[*count] = strdup(current);
        (*count)++;
        return;
    }

    // Get the current digit
    char digit = digits[index] - '0';

    // Iterate over the letters corresponding to the current digit
    for (int i = 0; i < strlen(letterMap[digit]); i++) {
        // Append the current letter to the current combination
        current[index] = letterMap[digit][i];

        // Recursively generate combinations for the next digit
        generateCombinations(digits, letterMap, index + 1, current, result, count);
    }
}

// Function to initialize and call the recursive function
char** letterCombinations(char *digits, int *returnSize) {
    // Mapping of digits to letters
    char *letterMap[] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

    // Calculate the maximum possible combinations
    int maxCombinations = 1;
    int length = strlen(digits);
    for (int i = 0; i < length; i++) {
        int digit = digits[i] - '0';
        maxCombinations *= strlen(letterMap[digit]);
    }

    // Allocate memory for the result array
    char **result = (char **)malloc(sizeof(char *) * maxCombinations);
    for (int i = 0; i < maxCombinations; i++) {
        result[i] = (char *)malloc(sizeof(char) * (length + 1));
    }

    // Initialize count to keep track of the number of combinations
    int count = 0;

    // Call the recursive function to generate combinations
    generateCombinations(digits, letterMap, 0, result[0], result, &count);

    // Update the return size
    *returnSize = count;

    return result;
}

// Function to free the memory allocated for the result array
void freeResult(char **result, int size) {
    for (int i = 0; i < size; i++) {
        free(result[i]);
    }
    free(result);
}

int main() {
    char digits1[] = "23";
    int size1;
```

```
char **result1 = letterCombinations(digits1, &size1);

printf("Example 1: [");
for (int i = 0; i < size1; i++) {
    printf("\"%s\"", result1[i]);
    if (i < size1 - 1) {
        printf(", ");
    }
}
printf("]\n");

// Free the memory allocated for the result array
freeResult(result1, size1);

char digits2[] = "";
int size2;
char **result2 = letterCombinations(digits2, &size2);

printf("Example 2: [");
for (int i = 0; i < size2; i++) {
    printf("\"%s\"", result2[i]);
    if (i < size2 - 1) {
        printf(", ");
    }
}
printf("]\n");

// Free the memory allocated for the result array
freeResult(result2, size2);

char digits3[] = "2";
```

```
    int size3;
    char **result3 = letterCombinations(digits3, &size3);

    printf("Example 3: [");
    for (int i = 0; i < size3; i++) {
        printf("\"%s\"", result3[i]);
        if (i < size3 - 1) {
            printf(", ");
        }
    }
    printf("]\n");

    // Free the memory allocated for the result array
    freeResult(result3, size3);

    return 0;
}
```