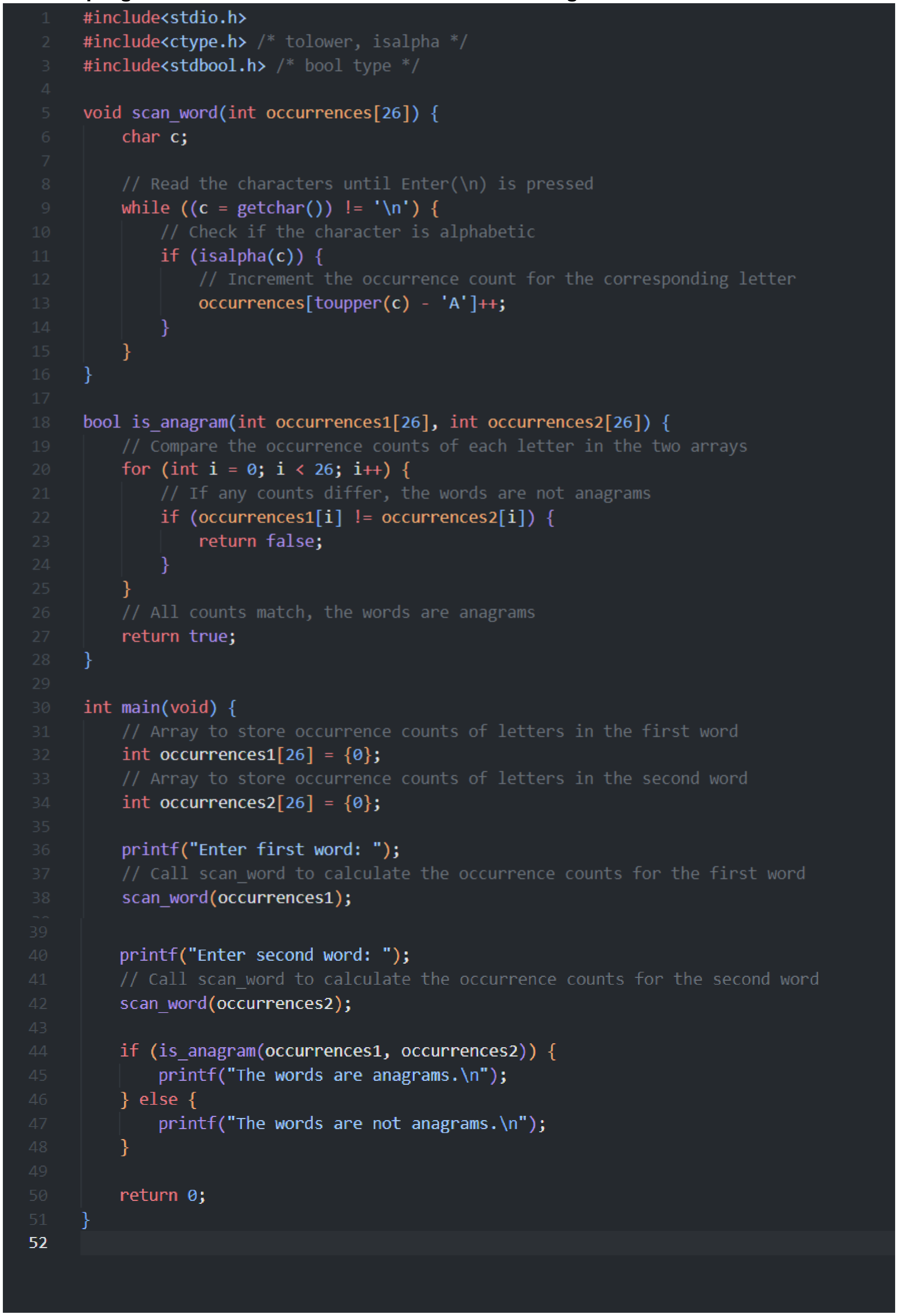
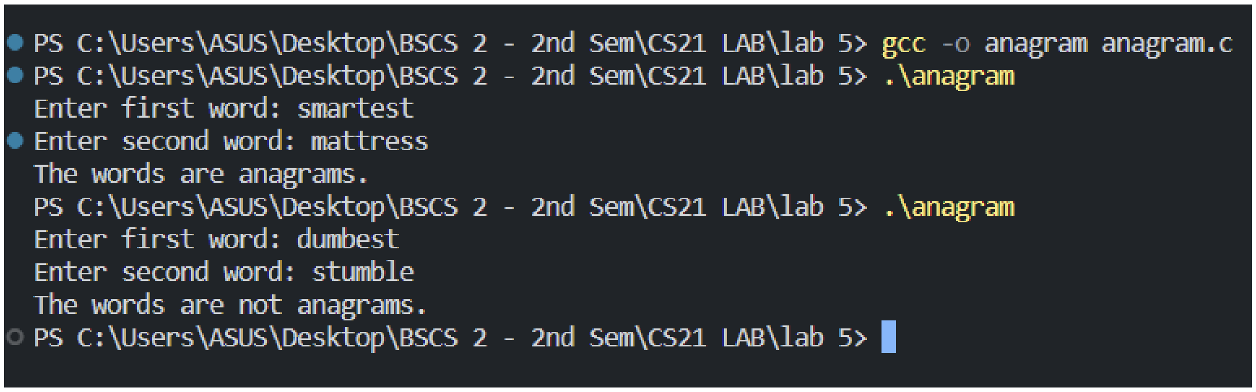
1. **The program below tests whether two words are anagrams:**



```c
#include<stdio.h>
#include<ctype.h> /* tolower, isalpha */
#include<stdbool.h> /* bool type */

void scan_word(int occurrences[26]) {
    char c;

    // Read the characters until Enter(\n) is pressed
    while ((c = getchar()) != '\n') {
        // Check if the character is alphabetic
        if (isalpha(c)) {
            // Increment the occurrence count for the corresponding letter
            occurrences[toupper(c) - 'A']++;
        }
    }
}

bool is_anagram(int occurrences1[26], int occurrences2[26]) {
    // Compare the occurrence counts of each letter in the two arrays
    for (int i = 0; i < 26; i++) {
        // If any counts differ, the words are not anagrams
        if (occurrences1[i] != occurrences2[i]) {
            return false;
        }
    }
    // All counts match, the words are anagrams
    return true;
}

int main(void) {
    // Array to store occurrence counts of letters in the first word
    int occurrences1[26] = {0};
    // Array to store occurrence counts of letters in the second word
    int occurrences2[26] = {0};

    printf("Enter first word: ");
    // Call scan_word to calculate the occurrence counts for the first word
    scan_word(occurrences1);

    printf("Enter second word: ");
    // Call scan_word to calculate the occurrence counts for the second word
    scan_word(occurrences2);

    if (is_anagram(occurrences1, occurrences2)) {
        printf("The words are anagrams.\n");
    } else {
        printf("The words are not anagrams.\n");
    }

    return 0;
}
```

Sample Output:

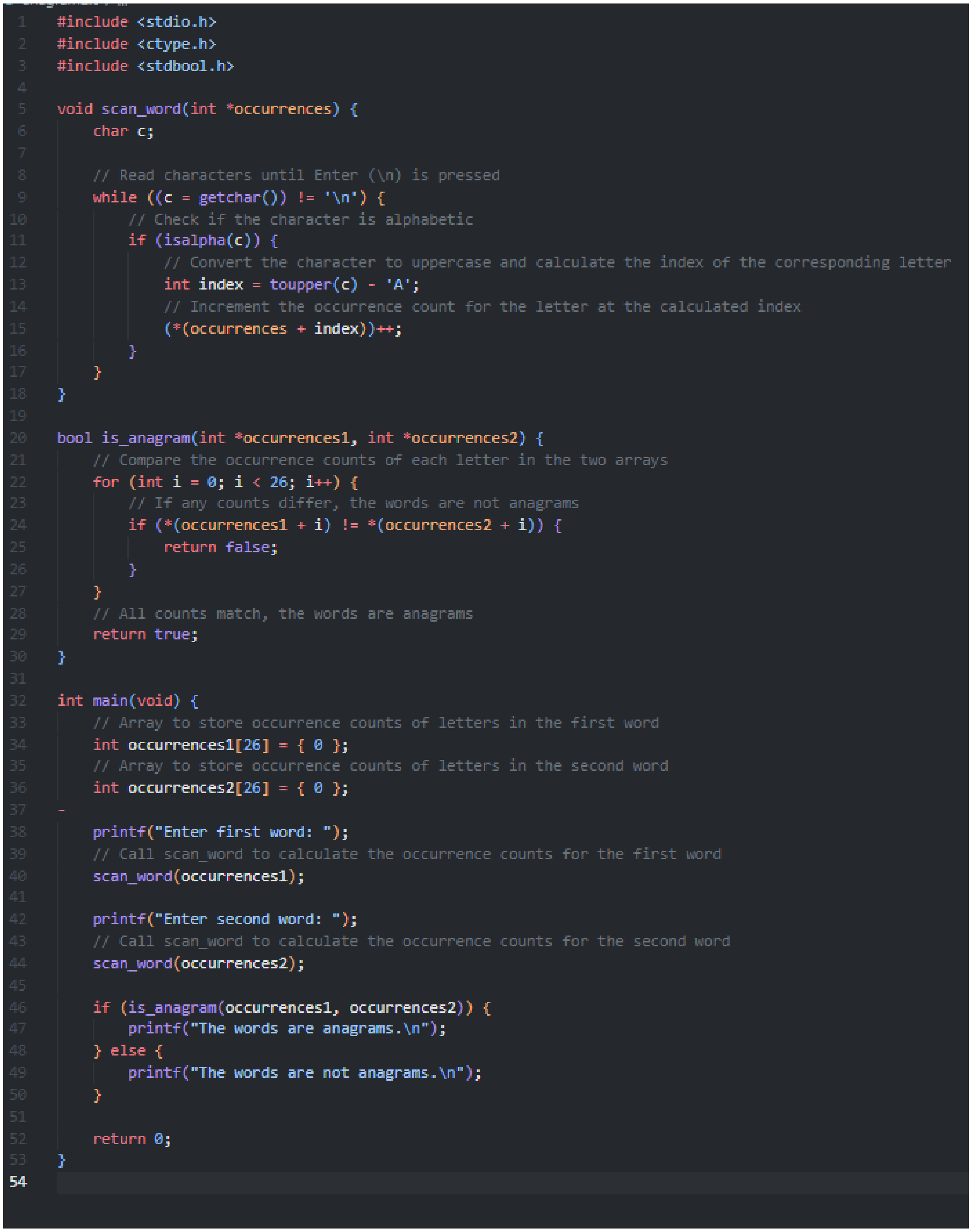


```
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5> gcc -o anagram anagram.c
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5> .\anagram
Enter first word: smartest
Enter second word: mattress
The words are anagrams.
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5> .\anagram
Enter first word: dumbest
Enter second word: stumble
The words are not anagrams.
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5>
```

Explanation of the code:

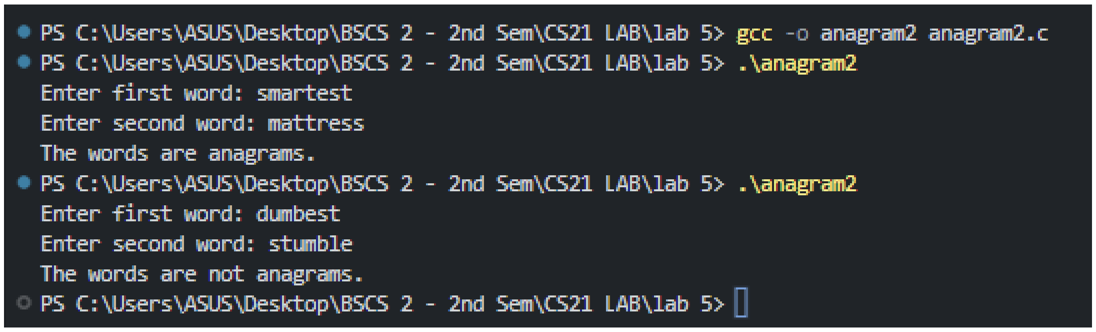
1. The scan_word function on line 5 takes the ‘occurrences’ array and reads the characters from the input until the Enter key is pressed. The while loop checks if the character ‘c’ is alphabetic using the ‘isalpha’ function. If it is true, the character is converted into an uppercase letter using the ‘toupper’ function and the ASCII value of ‘A’ is subtracted to get the corresponding index in the ‘occurrences’ array. For example the index of A would be 0 because the ASCII value of A is 65 and A itself was subtracted from it. If the letter was B, it would be B=B-A=1 because 66-65=1. Finally, the value in that index is incremented by one to keep track of the occurrence count of each letter.
2. The is_anagram function on line 18 takes ‘occurrences1’ and ‘occurrences2’ as the parameters. This compares the occurrence counts of each letter in the two arrays and if any of the counts differ, the 2 words are not anagrams and the function returns ‘false’ while the opposite would return ‘true’ as the counts match and are therefore anagrams.
3. In the main function, ‘occurrences1’ and ‘occurrences2’ store the occurrence counts of the letters in the two words which are initialized as 0 for a clear counting. After taking the first word, the scan_word function passing the ‘occurrences1’ array as an argument will calculate the occurrences counts of each letter in the word and store them in the ‘occurrences1’ array. Similarly, it prompts the user to enter the second word and calls the ‘scan_word’ function again taking the ‘occurrences2’ array as an argument.
4. Once the ‘is_anagram’ function is called with the arguments ‘occurrences1’ and ‘occurrences2’, the words will be checked whether they are anagrams or not. If the return value is true, they are anagrams, and if false, they are not.

2. Converted source code:



```c
#include <stdio.h>
#include <ctype.h>
#include <stdbool.h>

void scan_word(int *occurrences) {
    char c;

    // Read characters until Enter (\n) is pressed
    while ((c = getchar()) != '\n') {
        // Check if the character is alphabetic
        if (isalpha(c)) {
            // Convert the character to uppercase and calculate the index of the corresponding letter
            int index = toupper(c) - 'A';
            // Increment the occurrence count for the letter at the calculated index
            (*(occurrences + index))++;
        }
    }
}

bool is_anagram(int *occurrences1, int *occurrences2) {
    // Compare the occurrence counts of each letter in the two arrays
    for (int i = 0; i < 26; i++) {
        // If any counts differ, the words are not anagrams
        if (*(occurrences1 + i) != *(occurrences2 + i)) {
            return false;
        }
    }
    // All counts match, the words are anagrams
    return true;
}

int main(void) {
    // Array to store occurrence counts of letters in the first word
    int occurrences1[26] = { 0 };
    // Array to store occurrence counts of letters in the second word
    int occurrences2[26] = { 0 };

    printf("Enter first word: ");
    // Call scan_word to calculate the occurrence counts for the first word
    scan_word(occurrences1);

    printf("Enter second word: ");
    // Call scan_word to calculate the occurrence counts for the second word
    scan_word(occurrences2);

    if (is_anagram(occurrences1, occurrences2)) {
        printf("The words are anagrams.\n");
    } else {
        printf("The words are not anagrams.\n");
    }

    return 0;
}
```

Sample Output:



```
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5> gcc -o anagram2 anagram2.c
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5> .\anagram2
Enter first word: smartest
Enter second word: mattress
The words are anagrams.
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5> .\anagram2
Enter first word: dumbest
Enter second word: stumble
The words are not anagrams.
PS C:\Users\ASUS\Desktop\BSCS 2 - 2nd Sem\CS21 LAB\lab 5>
```

Explanation of the code:

The use of pointers enables direct access and modification of arrays by eliminating the need for unnecessary copying of arrays or returning modified arrays from the functions. This makes it efficient by allowing direct manipulation of array elements using pointer arithmetic.

1. The task of the scan_word function remains the same as the one in the source code but is now taking the pointer 'occurrences' as a parameter. This pointer is used to access the occurrence counts array in the main function. Using the pointer arithmetic '(*(occurrences + index))++', the function increments the occurrence count for a specific letter at the calculated index.
2. The is_anagram function also functions similarly to the previous code but is now taking the pointers 'occurrences1' and 'occurrences2' as the parameters. These pointers allow the function to access the occurrence counts arrays for both words in the calling function.
3. In the main function, The declared arrays are passed to the 'scan_word' function and the function, which act as pointers to the first elements of the arrays.

LINKS:
https://github.com/Andrea-Laserna/CMSC21/blob/main/Assignments/Assignment_LEC11_Laserna_1.c

https://github.com/Andrea-Laserna/CMSC21/blob/main/Assignments/Assignment_LEC11_Laserna_2.c