Duranne B. Duran

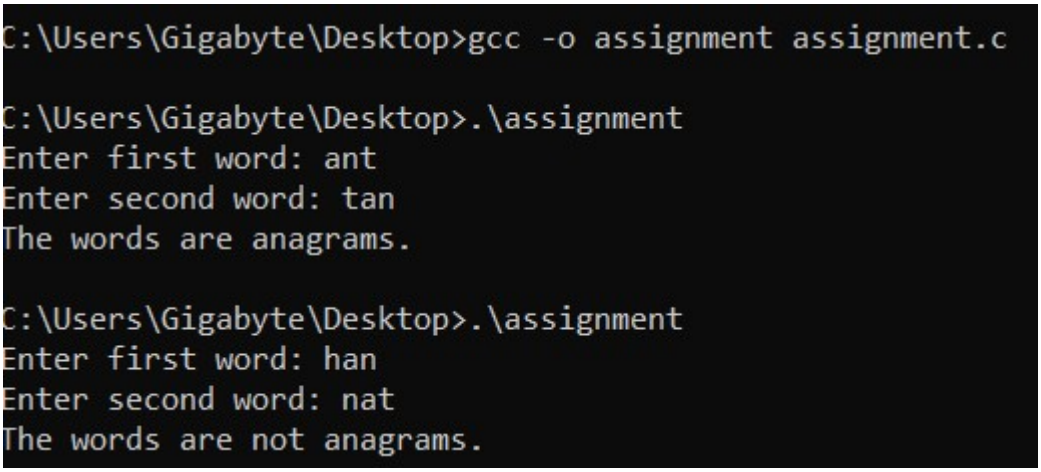
202250030

CMSC 21 Lec11 – Pointers and Multidimensional Arrays

1. Code:

```c
//Duranne B. Duran
//Assignment number 1

#include <stdio.h>
#include <ctype.h> /*toupper, isalpha*/
#include <stdbool.h>

void scan_word(int occurences[26]){
char c;
    while ((c = getchar()) != '\n'){ //checks if the character stored in c is not the new line character
        if (isalpha(c)){
            occurences[toupper(c) - 'A']++; //maps the index to occurence array and increments it
            }
        }
    }

bool is_anagram(int occurrences1[26], int occurrences2[26]) {
    //checks if the values of both arrays are equal
    //iterates through occurences array
    for (int i = 0; i < 26; i++) {
        if((occurrences1[i] != occurrences2[i])){
            return false; //returns false if not equal
        }
    }
    return true;
}

int main(void){
    int letters1[26] = {0};
    int letters2[26] = {0};

    printf("Enter first word: ");
    scan_word(letters1); //call function to scan first word

    printf("Enter second word: ");
    scan_word(letters2); //call function to scan second word

    bool same = is_anagram(letters1, letters2); //'same' will hold the return value of is_anagram

    if (same){ //if true
        printf("The words are anagrams.\n");
        return 0;
    }// if false
    printf("The words are not anagrams.\n");
    return 0;
}
```

Sample output:



Explanation:

The only things I added were the *scan_word()* and *is_anagram()* functions. The first function scans and evaluates each character of the word and updates the occurrences array. The while loop inside this function gets the characters that are inputted by the user and runs when it is not the *"\n"* character; and if the character is an alphabetical character, then it increments its corresponding index in the occurrence array.

The second function, *is_anagram(),* is a Boolean function that compares the updated occurrences arrays. It iterates through the 26 indices and compares if the values of the responding index from both arrays are equal. If the values are equal, then it returns true and therefore both words are anagrams.

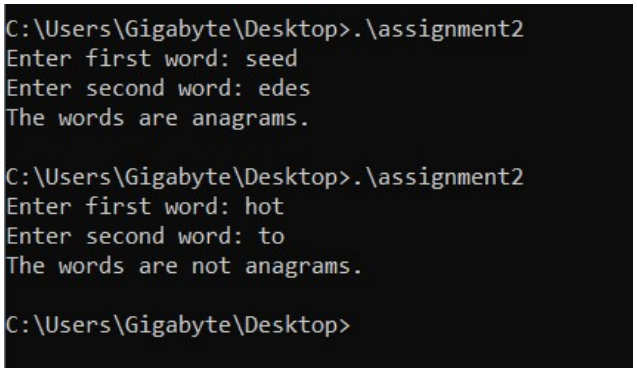
In the main function, I called *scan_word()* twice to scan the first word and second word. I also declared and initialize a Boolean variable *same* and let it hold the resulting value of the *is_anagram()* function. If *same* has a *true* value, then both words are anagrams; otherwise, they are not.

2. Code:

Users > Gigabyte > Desktop > C assignment2.c > ...

```c
//Duranne B. Duran
//Assignment number 2

#include <stdio.h>
#include <ctype.h> /*toupper, isalpha*/
#include <stdbool.h>

void scan_word(int *occurences){
char c;
    while ((c = getchar()) != '\n'){ //checks if the character stored in c is not the new line character
        if (isalpha(c)){
           (*(occurences + toupper(c) - 'A'))++; //maps the index to occurence array and increments it
            }
        }
    }

bool is_anagram(int *occurrences1, int *occurrences2) {
    //checks if the values of both arrays are equal
    //iterates through occurences array
    for (int i = 0; i < 26; i++) {
        if (*(occurrences1 + i) != *(occurrences2 + i)) {
            return false; //returns false if not equal
        }
    }
    return true;
}

int main(void){
    int letters1[26] = {0};
    int letters2[26] = {0};

    printf("Enter first word: ");
    scan_word(letters1); //call function to scan first word

    printf("Enter second word: ");
    scan_word(letters2); //call function to scan second word

    bool same = is_anagram(letters1, letters2); //'same' will hold the return value of is_anagram

    if (same){ //if true
        printf("The words are anagrams.\n");
        return 0;
    }// if false
    printf("The words are not anagrams.\n");
    return 0;
}
```

Sample output:



Explanation:

There are not that many changes to the code. The only things I changed were the parameters of the function and the way of navigating through the arrays. Firstly, I added the pointer operation before each parameter so that the function takes pointers as arguments. And secondly, since we are using pointers, we must use pointer arithmetic to access and navigate through the elements of the array. So, for example, occurrences[i] must be changed to *(occurrences + i); their values are the same and they return the i-th index of the array.