Test I

1. True
2. True
3. False
4. True
5. False
6. True
7. True
8. False
9. True
10. True
11. True

Test II

1. It is because C stores array in a row-major order. The first dimension is not necessary because it will determine the rows. The dimensions beyond it on the other hand, will determine the columns. When columns are not specified, the compiler will not know how to group them which will cause an error since it will not also know how to store it into memory.

2. A. bool isPalindrome(char *string);
   B. float computeAverage(float *arr[20]);
   C. void reverseSentence(void);
   D. float *squareRoot(int num);

3.

   A. Error 1: printf("%s", Inside function fun\n"); and

   printf("%s", Inside function bored\n");

   Fix: Add another quotation mark on the start of the string

   Correct Code: printf("%s", "Inside function fun\n"); and

printf(“%s”, “Inside function bored\n”);

Error 2: The whole bored() function

Fix: The whole function body must be outside the other function body

Error 3: Both functions does not bear anything.

Fix: Change int to void

Correct Code: void fun(void) and void bored(void)

B. Error: There is no return statement

Fix: Add a return statement that will return the variable result

Correct Code: return result;

C. Error 1: void fun (float a);

Fix: Remove the semicolon since we are creating a whole function, and not a function prototype.

Error 2: float a @ Line 3

Fix: Remove the whole line since the variable a has already been declared as the function’s argument.

D. Error 1: printf("%s", "Enter three integers: ")

Fix: Add a semicolon at the end

Correct Code: printf("%s", "Enter three integers: ");

Error 2: return total;

Fix: Since it is a void function, a return statement is not needed. It must be removed.

4. A. int numbers[5] = {1,2,3,4,5};
   B. int *ptr;
   C. ptr = &numbers[0];
   D. for(int i = 0; i < 5; i++){
         printf("%d ", *(ptr + i));

}

E. for(int i = 0; i < 5; i++){
printf("%d ", *(numbers + i));
}

F.

F.1) numbers[2];
F.2) *(numbers+ 2);
F.3) ptr[2];
F.4) *(ptr + 2)

G. Address = 2508, Value = 3

5. A. ++ is located at the left of the pointer. It should be the opposite.

B. There is no asterisk at the left of xp. It should be num = *xp;

C. There is an asterisk at the left of xp. It should be num = xp[1];

D. You cannot increment the array since there is no operation.

Test III

1.

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

int o1[26], o2[26];                                          //Declare global variable for occurence1 and occurence2

//Function Declarations
void scan_word(int occurences[26], int condition);
bool is_anagram(int occurences1[26], int occurences2[26]);

int main(void) {
    int letters[26] = {0};                                   //Set the values of the array to 0

    printf("Enter first word: ");
    scan_word(letters, 1);                                   //Calls scan_word(), 1 means that it is the first word
    printf("Enter second word: ");
    scan_word(letters, 2);                                   //Calls scan_word(), 2 means that it is the second word

```

```
    if (is_anagram(o1, o2)) {
        /*Checks if the two arrays have equal values,
        if they do, then the two words are anagrams*/
        printf("The words are anagrams.\n");
        return 0;
    }
    printf("The words are not anagrams.\n");
    return 0;
}

void scan_word(int occurences[26], int condition){
    /* This function will receive an empty array which will be
    the basis for the occurences of a letter in the word, and a condition
    if it is the first or second word.*/
    char c;
    while ((c = getchar()) != '\n') {
        if (isalpha(c)){
            if(condition == 1) o1[toupper(c) -
'A']++;                 //Check if it is the first word, then the final
occurence of the word will be saved to the array named o1
            else o2[toupper(c) -
'A']++;                             //Same logic with the line above,
the only difference is that it will be saved to the array named o2
        }
    }
}

bool is_anagram(int occurences1[26], int occurences2[26]){
    /*This function will check each value of the two arrays,
    if any index have different values, then it will return false.
    Otherwise, the loop will continue and if no difference is found,
    then it will return true.*/
    for(int i = 0; i < 26; i++){
        if(occurences1[i] != occurences2[i]){
            return false;
            break;
        }
    }
    return true;
}

```

2.

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

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

//Function Declarations
void scan_word(int occurences[26]);
bool is_anagram(int occurences1[26], int occurences2[26]);

int main(void) {

    int o1[26] = {0}, o2[26] = {0};                 //Declare the arrays that
will hold the letter occurence

    printf("Enter first word: ");
    scan_word(&o1);                                 //Calls scan_word(), pass
the address of o1
    printf("Enter second word: ");
    scan_word(&o2);                                 //Calls scan_word(), pass
the address of o2

    if (is_anagram(&o1, &o2)) {
        /*Checks if the two arrays have equal values,
        if they do, then the two words are anagrams.
        The arguments are the addresses of o1 and o2*/
        printf("The words are anagrams.\n");
        return 0;
    }
    printf("The words are not anagrams.\n");
    return 0;
}

void scan_word(int occurences[26]){
    /* This function will receive the address of an array which will be
    the basis for the occurences of a letter in the word. Since the array's
address was passed,
    changes made from here can be applied from the array.*/
    char c;
    while ((c = getchar()) != '\n') {
        if (isalpha(c)){
            *(occurences + (toupper(c) - 'A')) += 1;                //This line
is the same with occurences[toupper(c) - 'A']++; The value of the element will
add up everytime it exists on the word
        }
    }
}
```

```
bool is_anagram(int occurences1[26], int occurences2[26]){
    /*This function will check each value of the two arrays,
    if any index have different values, then it will return false.
    Otherwise, the loop will continue and if no difference is found,
    then it will return true.*/
    for(int i = 0; i < 26; i++){
        if(*(occurences1 + i) != *(occurences2 +
i)){               //*(occurences1 + i) == occurences[i]; same logic applies to
occurences2
            return false;
            break;
        }
    }
    return true;
}
```

Github Link:
https://github.com/dreeew05/CMSC21/tree/master/SecondLongExam