CMSC 21 2nd Long Exam

May 22-23, 2022

I. True or False

1. When calling a function, if the parenthesis is missing, the function won't get called.
2. The following version of sum_array is illegal: int sum_array(int a[n], int n){ … }
3. C allows functions to be nested.
4. The function prototype double average(); is illegal.
5. Suppose a variable fun is declared as a global variable with a value of 10 and redeclared as a local variable with a value of 5. If fun is printed inside the main function, the output is equal to 10.
6. If a variable *point* was declared as a pointer and a *var* was declared as an integer variable, **point = &var* is valid.
7. The name of an array always points to the value of the first element of an array.

Suppose that a is one-dimensional int array and p is a pointer to int variable. Assuming that the assignment p = a has just been performed, which of the following expressions are illegal because of mismatched types? State whether the expression from 8 to 11 is true or not.

8. p == a[0]
9. p == &a[0]
10. *p == a[0]
11. p[0] == a[0]

II. Provide the answers to the following:

1. Why is it that the first dimension in an array parameter be left unspecified, but not the other dimensions?
2. Write the function prototype given the following:
    a. Function isPalindrome that takes character type pointer argument string and returns a bool value.
    b. Function computeAverage that takes a floating-point array argument arr (with size of 20) and returns a float value.
    c. Function reverseSentence that does not take any argument and returns nothing.
    d. Function squareRoot that takes an integer number num and returns a floating-point result.
3. Find the error in each of the following code snippets and explain how the error may be corrected
    a.

```
int fun(void){
        printf("%s", Inside function fun\n");

        int bored(void){
                printf("%s", Inside function bored\n");
        }
```

```
}
```

b.
```
int product (int a, int b){
        int result = a * b;
}
```

c.
```
void fun (float a);
{
        float a;
        printf("%f", a);
}
```

d.
```
void sum(void){
        printf("%s", "Enter three integers: ")
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        int total = a + b + c;
        printf("Result is %d", total);
        return total;
}
```

4. Provide the answers to each of the following. Assumption: integer numbers are stored in 4 bytes, and the first element of the array is at location 2500 in memory.
    a. Define an integer array numbers with size = 5. Initialize the elements to values 1, 2, 3, 4, 5. Assume a constant SIZE is defined to 5.
    b. Define an integer pointer, ptr.
    c. Assign the address of the first element of array numbers to the pointer variable ptr.
    d. Print the elements of array numbers using pointer / offset notation with the pointer ptr
    e. Print the elements of array numbers using pointer/offset notation using the array name as the pointer
    f. Refer to element 2 of numbers using a pointer/offset notation using (f.1) array index notation, (f.2) pointer notation with array name as the pointer, (f.3) pointer index notation with ptr, (f.4) pointer notation with ptr.
    g. Assuming that ptr points to the address of the first element, what address is referenced by ptr+2? What value is stored at that address?
5. Find the error in the codes in a-d given initial code.

```
int *xp;  //references array x
void *vp = NULL;
int num;
int x[5] = {1, 2, 3, 4, 5};
vp = arr;
```

a. ++xp;
b. num = xp; //use pointer to access first element (assume xp is initialized)
c. num = *xp[1]; //assign element 1 (value 2) to num
d. ++x;

III. Application

1. The program below tests whether two words are anagrams (permutations of the same letters):

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

int main(void) {

    int i,
        same = 1,
        letters[26] = {0};
    char c;

    printf("Enter first word: ");
    while ((c = getchar()) != '\n') {
        if (isalpha(c)){
            letters[toupper(c) - 'A']++;
        }
    }
    printf("Enter second word: ");
    while ((c = getchar()) != '\n') {
        if (isalpha(c)){
            letters[toupper(c) - 'A']--;
        }
    }

    for (i = 0; i < 26; i++) {
        if (letters[i] != 0) {
            same = 0;
            break;
        }
    }
    if (same) {
        printf("The words are anagrams.\n");
        return 0;
    }
    printf("The words are not anagrams.\n");
    return 0;
}
```

Enter first word: smartest
Enter second word: mattress
The words are anagrams.

Enter the first word: dumbest
Enter the second word: stumble
The words are not anagrams.

The loop in lines 12-16 reads the first word, character by character, using an array of 26 integers to keep track of how many times each letter has been seen. (For example, after the word smartest has been read, the array should contain the values 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 2 2 0 0 0 0 0 0, reflecting the fact that smartest contains one a, one e, one m, one r, twos's and two t's.

The loop on lines 17-22 reads the second word, except this time decrementing the corresponding array element as each letter is read.

Both loops should ignore any characters that aren't letters and the function islpha is used. Both should treat upper-case letters in the same way as lower-case letters. One way to do this is by using toupper() to convert all letters to uppercase.

Header <ctype.h> allows the use of functions isalpha, tolower, or toupper.

After the second word has been read, use a third loop to check whether all the elements in the array are zero. If so the words are anagrams. Hint: You may wish to use.

The issue with the given code:

Duplications. Lines 11-16 and Lines 17-22 are basically doing the same thing. You could write a function that can perform the same task on different words.

Your task:

- Modify the anagram code above such that following functions are added:
    - void scan_word(int occurrences[26]);
    - bool is_anagram(int occurrences1[26], int occurrences2[26]);

main will **scan_word** twice, once for each of the two words entered by the user. As each character/letter of the word is being scanned, **scan_word** will use the characters in the word to update the occurrences array.

An array for each word will be declared.

int occurrences1[26] – keep track how many times each letter occurs in word 1

int occurrences2[26] – keep track how many times each letter occurs in word 2

main will then call **is_anagram**, passing it the two arrays (occurrences1 and occurrences2), **is_anagram** will return true if the elements in the two words are identical (including that the words are anagrams) and false otherwise.

2. Convert your source code in Application Item #1 such that you operate on the arrays using pointers.