Computer Programming

Strings in C

a string is an array of characters terminated by a null character (`\0`)

Input strings

scanf("%s", str); // This will not read spaces.

Calloc, Malloc ,Realloc

Malloc- Allocates a specified number of bytes of memory but does not initialize the memory.

```
void* malloc(size_t size);
```

```
int* ptr = (int*) malloc(5 * sizeof(int)); // Allocates memory for 5 integers
```

Calloc-Allocates memory for an array of elements and initializes all the bytes to zero.

```
void* calloc(size_t num, size_t size);
```

```
int* ptr = (int*) calloc(5 ,sizeof(int)); // Allocates memory for 5 integers
```

Realloc- Resizes a previously allocated memory block, preserving its content as much as possible.

```
void* realloc(void* ptr, size_t new_size);
```

If `new_size` is larger than the current size, `realloc()` expands the memory block and retains the existing data. However, the newly allocated space is uninitialized.

If `new_size` is smaller, the memory block is shrunk, and any extra data may be lost.

```
ptr = (int*) realloc(ptr, 10 * sizeof(int)); // Resize the memory block for 10 integers
```

Pointers

Pointers are variables that store the memory address of another variable.

```
int num = 10;

int *ptr = &num; // 'ptr' stores the address of 'num'
```

```
int arr[3] = {10, 20, 30};

int *ptr = arr; printf("%d\n", *ptr); // 10

ptr++; // Move to the next element

printf("%d\n", *ptr); // 20
```

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *ptr;
    ptr = (int *)malloc(sizeof(int));  // Dynamically allocate memory for an int
    if (ptr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }
    *ptr = 50;
    printf("Value: %d\n", *ptr); // 50
    free(ptr); // Free the allocated memory
    return 0;
}
```

```
int *ptr = NULL;

*ptr = 10; // Error! Dereferencing a NULL pointer
```

```
#include <stdio.h>

int main() {

int arr[] = {10, 20, 30, 40, 50};

int *ptr = arr; // Initialize pointer to the first element of the array

printf("Initial value: %d\n", *ptr); // 10

printf("*(ptr++): %d\n", *(ptr++)); // Dereference then increment pointer (prints 10, ptr moves to arr[1])

printf("*ptr++: %d\n", *ptr++); // Dereference then increment pointer (prints 20, ptr moves to arr[2])

printf("*(ptr++): %d\n", *(ptr++)); // Dereference then increment pointer (prints 30, ptr moves to arr[3])

printf("++*ptr: %d\n", ++*ptr); // Increment value at ptr (40 becomes 41, ptr still at arr[3])

printf("*(++ptr): %d\n", *(++ptr)); // Increment pointer, then dereference (prints 50, ptr moves to arr[4])

return 0; }
```

Problem 1

Given a string s consisting of words and spaces, return *the length of the **last** word in the string.*

Example 1

Input: s = "Hello World"

Output: 5

Explanation: The last word is "World" with length 5.

Example 2

Input: s = "   fly me   to   the moon "

Output: 4

Explanation: The last word is "moon" with length 4.

Code

```
int lengthOfLastWord(char* s) {
    int b=0;
    for(int i=strlen(s)-1;i>=0;i--){
        if(s[i]!=' '){
            b=i;
            break;
        }
    }
    int a=-1;
    for(int i=0;i<b+1;i++){
        if(s[i]==' '){
            a=i;
        }
    }

    return b-a;
}
```

Problem 2

Harry Potter is in charge of Gryffindor Quidditch Team at Hogwarts. Each house member has a level ranging between 1 and 1000.

Harry needs to sort the students of house by their level so that he knows the preparation of his team. Your task is to help Harry by writing a C program that:

1. Reads an integer n (the number of students).
2. For each student, reads their level.
3. Sorts the students in house by their level using a approach which is of complexity O(n+k) where k is max(values entered) and space complexity O(n)
4. Prints the sorted levels for the house

Input Format:

- The first line contains an integer n, the number of students.
- The next n lines each contain an integer representing the student's level (an integer between 1 and 1000).

Output Format:

- For each house, print the sorted list of levels of the students in the house.

Code

```
int count[maxLevel + 1];

for (int i = 0; i <= maxLevel; i++)

{

count[i] = 0;

}

 for (int i = 0; i < n; i++) {

count[levels[i]]++;

}

for (int i = 1; i <= maxLevel; i++)

 {

while (count[i] > 0) {

printf("%d\n", i);

count[i]--; } }
```

DOUBTS

THANK YOU