Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
- PRACTICE-11_FUN-PTR
    - Circular Queue
    - Primes are here again
    - The Clones of the Clones
- LAB-PRAC-11_FUN-PTR
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC

The Clones of the Clones

PRACTICE-11_FUN-PTR

We have seen that C functions can call each other. However, C functions can call themselves as well. The process of a function calling itself is called recursion and it can be used to write very elegant solutions to complex-looking problems.

The template code given with this problem gives you the recursive code to print the locations of all occurrences of a given character in a string. Modify this code to print all occurrences of a given substring in a string. Both the string, as well as the substring will be at most 99 characters long and will be given in a single line. We will first give the substring then the string in two separate lines.

WARNING: programs that use recursion excessively can be slow due to the cloning process being a bit expensive. All C programs that are written using recursion, can also be written in a way that does not use any recursion. This result is a landmark and fundamental result in the theory of computation (CSE students will study this topic in the course CS340), guaranteed by the Church-Turing thesis. However, the non-recursive code can sometimes look very messy to humans whereas the recursive code looked very elegant, even though the messy code is faster than the elegant code.

The template code is reproduced below in case you erase it while coding.

```
------------ TEMPLATE CODE ------------
#include <stdio.h>
#include <string.h>

// whereIsChar recursively calls itself to find out
// 1. the locations of all occurrences of c
// 2. total number of occurrences of c
// offset tells us how many chars of the string have we already searched
int whereIsChar(char* string, char c, int offset){
    char* ptr = strchr(string, c); // Find next occurrence of c
    if(ptr != NULL){ // Did we find c ?
        int foundIndex = offset + (int)(ptr - string);
        printf("Found at index %d\n", foundIndex);
        // Continue searching for c in the remaining string
        return 1 + whereIsChar(ptr + 1, c, foundIndex + 1);
    }else
        return 0; // Nothing more to do - no more occurences of c
}

int main(){
    char c, str[100];
    scanf("%c\n", &c);
    gets(str);

    // Offset is 0 since none of the string has been searched so far
    printf("Found %d occurrence(s) of char %c", whereIsChar(str,c,0), c);
    return 0;
}
```

Start Solving! (/editor/practice/6214)