CS19003 : Programming and Data Structures Laboratory

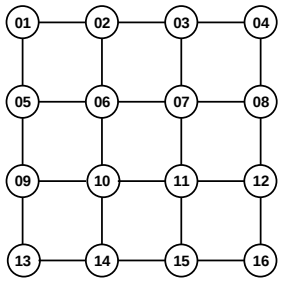
Assignment 5A : Functions in C

Date: 01-Jun-2021

Problem Statement – A:

Getting some assured clues of one of the murder suspects, Detective Byomkesh Bakshi involved Police to chase down that Murderer. Accordingly, a group of Police started chasing a Murderer inside the city where the streets follow a square-grid structure. In every turn, the Murderer moves and then all the Polices are moving. Each move is along an edge (either horizontal or vertical direction) of the square-grid. If any of the Police is on the same node as the Murderer, before or after Murderer's move, then the Polices are able to catch the Murderer.

The streets in the city are planned so well that these always form square-grid structure as shown in the figure (right-side). In particular, the position numbers are laid out in an incremental manner from left to right row-wise and from top to bottom. From the corner, boundary and intermediate island/positions, anyone can move to two, three or four alternative next positions/spots, respectively. As an example, check the square-grid of length 4 (length to be provided by user), given in the figure along with positions marked using numbers.



Now, any Police as well as the Murderer can move one step in the following way (each move is characterized by one letter):

- **U** – moving up in the grid (for example, from position [07] to position [03] in the figure above)
- **D** – moving down in the grid (for example, from position [01] to position [05] in the figure above)
- **L** – moving left in the grid (for example, from position [11] to position [10] in the figure above)
- **R** – moving right in the grid (for example, from position [15] to position [16] in the figure above)
- **S** – stay in the same position (for example, from position [04] to position [04] in the figure above)

The Police station is always situated at the top-left position in the grid (Position [01] according to the figure). So, all the Polices start moving from there always, while the Murderer can start from any position arbitrarily. Note that, any Police can catch the Murderer provided that both reach to the same position before or after the move made by anyone (murderer or police turn-wise).

Can you write a C-program that gets the moves of the Murderer and that of all the Polices and decides when the Murderer is caught or whether the Murderer has escaped? In particular, your program will do the following:

- *Take from user the inputs in following input format:*

  The input line contains `Four` numbers (space separated) and `M` strings of letters (each string is space separated): `K N M L` $\texttt{STR}_1$ $\texttt{STR}_2$ ... $\texttt{STR}_\texttt{M}$, where `K` = length of the square-grid; `N` = number of Polices; `M` = total number of moves/steps that are allowed to make; and `L` = Murderer's initial location. These numbers are followed by `M` groups/strings of $(\texttt{1} + \texttt{N})$ letters each which define the moves of Murderer (the first letter) and each one of the `N` Polices (the next `N` letters).

- *Detail out the movements through positions reached by murderer and the polices at each step* (refer to the sample execution for details).

- Finally, *print the following three numbers as output (space-separated):* `X Y Z` or `0 0 0`

  where, `X` = total number of moves after which Murderer is caught, `Y` = index of the Police (Police indexes start from 1) who caught the Murderer (if several has, give minimal one) and `Z` = grid-position of murderer where (s)he has been caught; or `0 0 0` if the Murderer survives the search (not get caught).

Note: You may assume that, $1 \leq \texttt{K} \leq 100$, $1 \leq \texttt{N} \leq 1000$, and $1 \leq \texttt{M} \leq 10000$.

Example Execution Details:

Sample-1:

```
Input: 4 4 2 1 RDRDR DLRRR

Movements:
    -- Murderer's Initial Position: 1
    -- Polices' Initial Position: 1 1 1 1
Output: 0 1 1
```

Sample-2:

```
Input: 4 4 6 11 LRRDD DDRRD RRRDD RSDRR UDDDR LDLRS

Movements:
    -- Murderer's Initial Position: 11
    -- Polices' Initial Position: 1 1 1 1
    -- Murderer moves to Position: 10
    -- Polices move to Position:: 2 2 5 5
    -- Murderer moves to Position: 14
    -- Polices move to Position:: 6 3 6 9
    -- Murderer moves to Position: 15
    -- Polices move to Position:: 7 4 10 13
    -- Murderer moves to Position: 16
    -- Polices move to Position:: 7 8 11 14
    -- Murderer moves to Position: 12
    -- Polices move to Position:: 11 12 15 15
Output: 5 2 12
```

Sample-3:

```
Input: 4 4 5 11 SRRDD LRDRD DRRDD RDSRR RDDDR

Movements:
    -- Murderer's Initial Position: 11
    -- Polices' Initial Position: 1 1 1 1
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 2 2 5 5
    -- Murderer moves to Position: 10
    -- Polices move to Position:: 3 6 6 9
    -- Murderer moves to Position: 14
    -- Polices move to Position:: 4 7 10 13
    -- Murderer moves to Position: 15
    -- Polices move to Position:: 8 7 11 14
    -- Murderer moves to Position: 16
    -- Polices move to Position:: 12 11 15 15
Output: 0 0 0
```

Sample-4:

```
Input: 4 2 12 11 SRD SRD SRD SDR SDR SDR SLU SLU SUL SUL SRD SDR

Movements:
    -- Murderer's Initial Position: 11
    -- Polices' Initial Position: 1 1
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 2 5
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 3 9
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 4 13
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 8 14
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 12 15
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 16 16
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 15 12
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 14 8
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 10 7
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 6 6
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 7 10
    -- Murderer moves to Position: 11
    -- Polices move to Position:: 11 11
Output: 12 1 11
```

Submit a single C source file. Do not use global/static variables.

CS19003 : Programming and Data Structures Laboratory

Assignment 5B : Recursions in C

Date: 01-Jun-2021

Problem Statement – B:

Due to the chase down of the murderers/suspects for the twin murder case by police over the past few weeks, Detective Byomkesh Bakshi and Ajit can finally cut down the number of suspects to a few potential list of murderers and ranked them into their order/index of suspicion (higher rank indicating highly suspicious). Now, they would like to further interrogate these potential murderers one by one. However, Mr. Bakshi do not want the suspected people to know their rank/order (suspicion index) and hence he will not interrogate them in the same order as their suspicion rank. More precisely, the i-th ranked person will be interrogated in any possible place except his/her name appearing in i-th position in the interrogation list. So, Ajit thought of forming all the possible misplacements of the ranked list and show it to Mr. Bakshi. As an example, you may think of 3 suspects ranked as $1, 2, 3$ can be interrogated in two possible ways, either in $3, 1, 2$ order, or in $2, 3, 1$ order.

It may be noted that, there are several possibilities, say $D(n)$, of such shuffling (**derangements**) for n suspects. To be mathematically precise, the total count can be expressed recursively using the following formula:

$$\texttt{[REC]} \quad D(n) = (n-1)\times\big[D(n-1)+D(n-2)\big] \text{ with } D(1)=0,\ D(2)=1 \qquad \text{which solves to,} \quad D(n) = n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}$$

Now, looking meticulously into the closed form (by our so called computational/iterative way) from *left-to-right* (L-R) and from *right-to-left* (R-L) provides two different nested/iterative series structures as illustrated below.

$$\texttt{[L-R]} \quad D(n) = \Big(n!\Big) - \Big(\frac{n!}{1}\Big) + \Big(\frac{n!}{1\times 2}\Big) - \Big(\frac{n!}{1\times 2\times 3}\Big) + \cdots + \Big(\frac{(-1)^{n-2}.n!}{1\times\cdots\times(n-2)}\Big) + \Big(\frac{(-1)^{n-1}.n!}{1\times\cdots\times(n-1)}\Big) + \Big(\frac{(-1)^{n}.n!}{1\times\cdots\times n}\Big)$$

$$\texttt{[R-L]} \quad D(n) = \Big((-1)^n\Big) + \Big((-1)^{n-1}.n\Big) + \Big((-1)^{n-2}.n.(n-1)\Big) + \cdots + \Big(n.(n-1)...4.3\Big) - \Big(n.(n-1)...3.2\Big) + \Big(n.(n-1)...2.1\Big)$$

Note: If you follow the parenthesized expressions (one after another from left-to-right in the above sum of terms), you'll be able to find both variants of the iterative process.

Now, *how can you get all possible derangement sequences?* For that, you may frame the following recursive process: Start with the properly sequenced rank-array. Put each of the ranks ($1 \le i \le n$) into all other/different positions j ($1 \le j \ne i \le n$) and recursively call for the derangements for the remaining $(n-1)$ rank array. So, you may think of developing a recursive function having the prototype as follows:

```
unsigned long long    derangements ( int d[ ] , int length , int depth , int show )
```

Note the arguments to this function as follows. It contains the sequence/array that will contain derangements (`d[ ]`) along with its length (`length`) and the depth of the recursion (`depth`) indicating the position of i which you are currently recursing through. Additionally, since this function extracts all possible deranged sequences, so it can also give you a total count of these. So, you may take additional argument (`show`) to only show the count (no sequences) if `show` $= 0$, or to show all possible deranged sequences (no count) if `show` $= 1$. When you show the derangements, also calculate the **distance** of each derangements from proper positions. The distance is to be measured by the sum of the number of positions every place differs from its original/proper ranks. As an example, the interrogation of 3 suspects can be done in 2 ways, $3, 1, 2$ (which has, `distance` $= |3-1| + |1-2| + |2-3| = 4$) and $2, 3, 1$ (which has, `distance` $= |2-1| + |3-2| + |1-3| = 4$).

Can you write a C-program that performs this derangements of suspect list and also counts the total possibilities in multiple ways described? In particular, your program will do the following:

- Take from user as input the number of suspects (n). Assume that, $1 \le n \le 1024$.
- Print the rank list, that is the proper sequence, $1, 2, \ldots, n$.
- Compute total number of possible derangements using the recursive formula ([`REC`]) and print the count.
- Compute the total number of derangements possible using the iterative formula and print the count. Do it separately for left-to-right ([`L-R`]) as well as right-to-left ([`R-L`]) iterations. For the [`L-R`] type of counting process, please notice that you may need to write a *(recursive)* factorial function as well.

- Complete the definition/body of the function, `derangements`(). Then, call it with `show` = `0` to print the count (refer to the execution details where "# `COUNT` #" is shown) as well as call with `show` = `1` in the second time. While printing every deranged sequence, mention the corresponding distances as output (refer to the execution details where "[`Distance` = `VALUE`]" is shown).

Example Execution Details:

Sample-1:

```
++ Enter n [1-1024]: 1

** The Proper Sequence: 1
** Recursive Count of Derangements: 0
** Iterative Count of Derangements: 0 [Left-to-Right]
** Iterative Count of Derangements: 0 [Right-to-Left]
** The # 0 # Derangements are:
```

Sample-2:

```
++ Enter n [1-1024]: 2

** The Proper Sequence: 1 2
** Recursive Count of Derangements: 1
** Iterative Count of Derangements: 1 [Left-to-Right]
** Iterative Count of Derangements: 1 [Right-to-Left]
** The # 1 # Derangements are:
 2 1    [Distance = 2]
```

Sample-3:

```
++ Enter n [1-1024]: 3

** The Proper Sequence: 1 2 3
** Recursive Count of Derangements: 2
** Iterative Count of Derangements: 2 [Left-to-Right]
** Iterative Count of Derangements: 2 [Right-to-Left]
** The # 2 # Derangements are:
 3 1 2    [Distance = 4]
 2 3 1    [Distance = 4]
```

Sample-4:

```
++ Enter n [1-1024]: 4

** The Proper Sequence: 1 2 3 4
** Recursive Count of Derangements: 9
** Iterative Count of Derangements: 9 [Left-to-Right]
** Iterative Count of Derangements: 9 [Right-to-Left]
** The # 9 # Derangements are:
 4 1 2 3    [Distance = 6]
 4 3 1 2    [Distance = 8]
 4 3 2 1    [Distance = 8]
 3 4 2 1    [Distance = 8]
 3 4 1 2    [Distance = 8]
 3 1 4 2    [Distance = 6]
 2 4 1 3    [Distance = 6]
 2 3 4 1    [Distance = 6]
 2 1 4 3    [Distance = 4]
```

Submit a single C source file. Do not use global/static variables.