CS19001: Programming and Data Structures Laboratory

Assignment No. 5A (Functions and Recursions)

Date: 07-September-2019

Problem Statement:

A group of White-Rabbits are chasing the Mock-Turtle in the planned city of Wonderland where the streets follow the inverted-tree structure as shown in Figure 1. In every turn, the Mock-Turtle moves and then all the White-Rabbits are moving. Each move is along an edge (street) of the tree. If any of the White-Rabbit is on the same node as the Mock-Turtle, before or after Mock-Turtle's move, then the White-Rabbits are successful in catching the Mock-Turtle. The streets in the Wonderland city are planned so well that these always form a nice inverted-tree structure. The nodes/junctions are numbered from 1 to $2^H - 1$ (for height H), as shown in Figure 1. An example layout upto height 3 is given in Figure 1. Now, any White-Rabbit as well as the

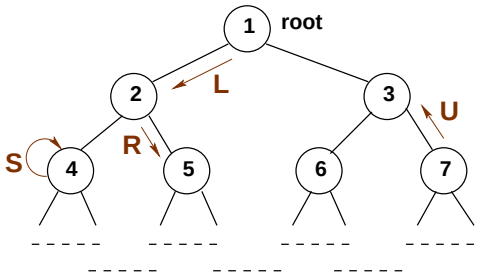

Figure 1: Layout of Wonderland City (Junctions + Streets) and Indication of Moves

Mock-Turtle can move one step in the following way (each move is characterized by one letter):

- **U** – moving up from node numbered i to node numbered $\lfloor \frac{i}{2} \rfloor$ (e.g., from node 7 to node 3 in Figure 1)
- **L** – moving to left child from node numbered i to node numbered $2i$ (e.g., from node 1 to node 2 in Figure 1)
- **R** – moving to right child from node numbered i to node numbered $2i+1$ (e.g., from node 2 to node 5 in Figure 1)
- **S** – stay at same place/node (e.g., from node 4 to node 4 in the example Figure 1)

The kingdom of White-Rabbits is always situated at the root (topmost) node in the tree. So, all the White-Rabbits start moving from the root, while the Mock-Turtle can start from any node. Note that, any White-Rabbit catches the Mock-Turtle if it is on the same spot before or after the move; or if they cross each other on their way.

You have to write a C-program that takes as input the moves of the Mock-Turtle and the moves of all the White-Rabbits and decides when the Mock-Turtle is caught or whether the Mock-Turtle has escaped.

- The **input line** contains three numbers (space separated) and M strings of letters (each string is space separated): $N \quad M \quad L \quad STR_1 \quad STR_2 \quad \ldots \quad STR_M$, where N = number of White-Rabbits; M = total number of moves/steps that are allowed to make; and L = Mock-Turtle's initial location ($1 \leq N \leq 10^3$ and $1 \leq M \leq 10^4$). These numbers are followed by M groups/strings of $N+1$ letters each which define the moves of Mock-Turtle (the first letter) and each one of the N White-Rabbits (the next N letters).
- The **output line** contains the move number after which the Mock-Turtle is caught, and the index of the White-Rabbit (White-Rabbit indexes start from 1) who caught the Mock-Turtle (if several has, give the minimal one), or 0 0 if the Mock-Turtle survives the search (not get caught).

Example Inputs/Outputs:

Sample-1:

```
Inputs: 4 2 3 LLLRR SLRLR
Outputs: 2 3
```

Sample-2:

```
Inputs: 2 6 8 LLR ULS SRS USR USU SUU
Outputs: 0 0
```

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

CS19001: Programming and Data Structures Laboratory

Assignment No. 5B (Functions and Recursions)

Date: 07-September-2019

Problem Statement:

The Queen-of-Hearts in Wonderland city wants to partition all its Gryphons into various groups. For n (positive integer) Gryphons, the partitioning is a way of dividing n Gryphons into groups whose sum becomes n. For example, all the partitions of 5 Gryphons are: $(1+1+1+1+1)$, $(1+1+1+2)$, $(1+1+3)$, $(1+2+2)$, $(1+4)$, $(2+3)$ and (5). Permuting the summands in a partition does not give a new partition. For example, the partition $(1+2+2)$ is treated the same as $(2+1+2)$ and also as $(2+2+1)$.

Part-I: In this part, you are asked to compute the number of partitions of n Gryphons. Write a recursive function to this effect. In order to avoid repetitions, we choose the summands in a non-decreasing order. That is, if 1 and 2 are already chosen as the previous summands, the next summand cannot be less than 2. Pass two arguments to your recursive function. The first stands for the amount left to be balanced by summands and the second stands for the largest summand chosen so far. Recursion stops when the first argument becomes 0. Here is a possible prototype for the function.

```
unsigned long countP ( unsigned long balance , unsigned long minchoice );
```

Part-II: Repeat *Part-I* with the added constraint that no summand is repeated in the partition. For example, only three out of the seven partitions of 5 correspond to no repetitions: $(1+4)$, $(2+3)$ and (5). Modify the recursive function of *Part-I* slightly so as to generate the count of partitions without repetitions. A prototype for this function would be:

```
unsigned long countPNoRep ( unsigned long balance , unsigned long minchoice );
```

Finally, here is a recommended set of activities for the main() function of your C-program:

- Take as input from user the number of Gryphons (non-negative long integer).
- Print the number/count of all partitions, calling the `countP` function.
- Print the number/count of all partitions without repetitions, calling the `countPNoRep` function.

Test the output of your program on the following values of n: 6, 10, 25, 50, 100.

Example Inputs/Outputs:

Sample-1:

```
Enter Number of Gryphons (n): 5
Count of All Partitions                 = 7
Count of Partitions without Repetitions = 3
```

Sample-2:

```
Enter Number of Gryphons (n): 40
Count of All Partitions                 = 37338
Count of Partitions without Repetitions = 1113
```

Sample-3:

```
Enter Number of Gryphons (n): 80
Count of All Partitions                 = 15796476
Count of Partitions without Repetitions = 77312
```

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