COL106: Data Structures **Spring 2019**

Assignment 1 : Stacks and Tower of Hanoi

1 [15 Points] Implementing a Stack

In this part, you will implement a stack using linked lists. Read about Java Generics and implement the stack class using generics. **Do not** use in-built list data structures.

Deliverables:

- [15 Points] Write a java class `MyStack<E>` in file `MyStack.java` that implements stack with following methods.
  - `public void push(E item)` : Pushes the `item` onto the top of the stack.
  - `public E pop()` : Removes the element at the top of the stack and returns it. Throw `EmptyStackException` if the stack is empty.
  - `public E peek()` : Returns the top element without removing from top of the stack. Throw `EmptyStackException` if the stack is empty.
  - `public bool empty()` : Returns `true` if the stack is empty and `false` otherwise.

  All other methods or variables in your implementation should be *private*.

2 [35 Points] Tower of Hanoi

In this part, you will solve the classic towers of hanoi problem.

Towers of hanoi puzzle consists of three rods and n disks of different sizes. Let the three rods be numbered 1, 2 and 3. Initially, n disks are stacked on rod 1 in ascending order of the sizes with the largest disk at the bottom and smallest disk at the top. Your goal is to move this entire stack from rod 1 to rod 3 using a sequence of moves. In a move, you are allowed to take a disk from the top of a stack on one of the rods and move to the top of the stack on other rods with a caveat that at *no point of time, a disk is on top of a smaller disk.*

Write a class `TowerOfHanoi` in file `TowerOfHanoi.java` with implementations of following methods.

Deliverables:

- [5 Points] Implement a method `toh_with_recursion` with following signature.

```
public static void toh_with_recursion(int num_disks, int start_pos, int end_pos)
```

  - Use *recursion* to solve the tower of hanoi problem.
  - `num_disks:` Number of disks.
  - `start_pos:` Integer which is either 1, 2 or 3 denoting the number of rod on which stack of disks is initially located.
  - `end_pos:` Integer which is either 1, 2 or 3 denoting the number of rod on which stack of disks is located at the end.
  - `Output Format:` Print a line for each move. Each line consists of two integers. In each line, print the number of rod from which the disk is taken from followed by a `space` followed by the number of rod onto which the disk is moved. A sample output is given below.

```
1  2
1  3
2  1
⋮  ⋮
```

- [30 Points] Implement a method `toh_without_recursion` with following signature.

  ```
  public static void toh_without_recursion(int num_disks, int start_pos, int end_pos)
  ```

  - All the variables mean same as in above part.
  - You *cannot* use recursion in this method.
  - You should use the *stack* that you have implemented to simulate recursion.
  - Output format is same as above.

3 [50 Points] Generalized Tower of Hanoi

In this part, you will solve a general version of the `Towers of Hanoi`.

Here we have $2n$ disks. Let the disks be numbered 0 to $2n-1$. A disk can be colored either red or black. Size and color of disk i are as follows

$$size(i) = i + 1$$

$$color(i) = \begin{cases} red, \text{ for } i \text{ odd} \\ black, \text{ for } i \text{ even} \end{cases}$$

Initially, disks are arranged in a stack on rod 1 in increasing order of their sizes i.e., disk $2n-1$ is at the bottom and disk 0 is at the top. Your goal is to stack all red disks on rod 1 and all black disks on rod 2 in the increasing order of their sizes. You can move a disk from top of a stack and place it on top of another stack and at no point of time, *a larger disk is on top of a smaller disk.*

Write a java class `GeneralizedTowerOfHanoi` in file `GeneralizedTowerOfHanoi.java` with implementations of following methods.

Deliverables:

- [10 Points] Implement a method `gtoh_with_recursion` with following signature.

  ```
  public static void gtoh_with_recursion(int num_disks, int start_pos, int[]
                                         final_positions)
  ```

  - Use *recursion* to solve this problem.
  - `int num_disks` : No. of disks. Sizes increase from disk number 0 to disk number `num_disks-1`.
  - `int start_pos` : This is an integer between 1 and 3 both inclusive. This denotes the rod on which the stack of disks is initially on. Initial stack has the largest disk i.e., disk number `num_disks-1` on bottom and disk number 0 on top.
  - `int[] final_positions` : This is an integer array of length `num_disks`. Each value in the array is either 1, 2 or 3. `final_positions[i]` denotes the final rod on which `i`th disk is to be placed eventually. *No larger disk should be on top of a smaller disk at any point of time.*
  - Output format is same as above.

- [40 Points] Implement a method `gtoh_without_recursion` with following signature.

  ```
  public static void gtoh_without_recursion(int num_disks, int start_pos, int[]
                                            final_positions)
  ```

- All variables mean same as in above part.
- You *cannot* use recursion in the implementation.
- You should use *stack* you have implemented to simulate recursion.
- Output format is same as above.