Due Date: Mon, Oct 21, 2024, 11:59 PM

# CSC 2720: Data Structures
# Homework 1

**Problem 1:** An array of elements is given. The task is to find the next greater for each element in the array. If there is no greater element to the right of the element, then return -1. While a brute-force solution using nested loops has a time complexity of O(n^2), the required solution should have a time complexity of O(n), which can be achieved using a stack.

|  |  |
|---|---|
| **Sample Input**: | 2 1 4 3 |
| **Sample Output:** | 4  4 -1 -1 |

**Problem 2:** Standard web browsers contain features to move backward and forward among the pages recently visited. One way to implement these features is to use two stacks to keep track of the pages that can be reached by moving backward and forward. You are asked to simulate this feature as a Python console application. The application should accept the following commands:

1. BACK: If the backward stack is empty, the command is ignored. Otherwise, push the current page on the top of the forward stack. Pop the page from the top of the backward stack, making it the new current page.
2. FORWARD: If the forward stack is empty, the command is ignored. Otherwise, push the current page on the top of the backward stack. Pop the page from the top of the forward stack, making it the new current page.
3. VISIT <url>: Push the current page on the top of the backward stack, and make the URL specify the new current page. The forward stack is emptied.
4. QUIT: Quit the application.

Consider handling any exceptional scenarios (e.g. invalid attempts to go BACK or FORWARD, missing ‘url’ for the visit command, misspelled/invalid commands etc.).

## Input

Input contains a series of commands. The keywords BACK, FORWARD, VISIT, and QUIT are all in uppercase. URLs have no whitespace and have at most 50 characters. The end of input is indicated by the QUIT command and the application should terminate when this command is received.

## Output

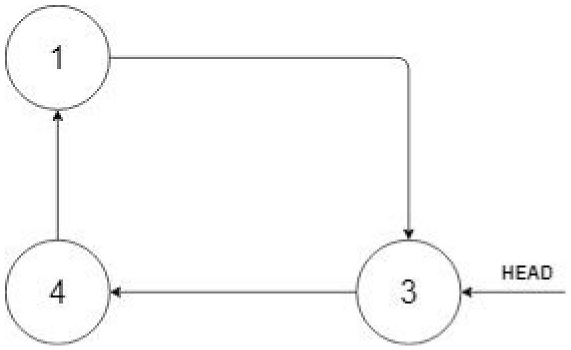
For each command, print the URL of the current page (in a line) after the command is executed if the command is not ignored. Otherwise, print `Ignored`

| Sample input: | Sample output |
|---|---|
| VISIT http://www.gmail.com/<br>VISIT http://www.facebook.com/<br>BACK<br>BACK<br>BACK<br>FORWARD<br>VISIT http://acm.sgu.ru/<br>BACK<br>BACK<br>FORWARD<br>FORWARD<br>FORWARD<br>QUIT | http://www.gmail.com/<br>http://www.facebook.com/<br>http://www.gmail.com/<br>http://www.google.com/<br>Ignored<br>http://www.gmail.com/<br>http://acm.sgu.ru/<br>http://www.gmail.com/<br>http://www.google.com/<br>http://www.gmail.com/<br>http://acm.sgu.ru/<br>Ignored |

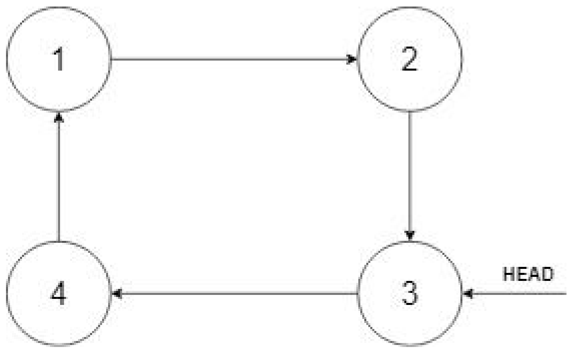
**Problem 3:** Given a Circular Linked List node, which is sorted in non-descending order, write a function to insert a value insertVal into the list such that it remains a sorted circular list. The given node can be a reference to any single node in the list and may not necessarily be the smallest value in the circular list.

If there are multiple suitable places for insertion, you may choose any place to insert the new value. After the insertion, the circular list should remain sorted.

If the list is empty (i.e., the given node is null), you should create a new single circular list and return the reference to that single node. Otherwise, you should return the originally given node.



**Sample Input:** head = [3,4,1], insertVal = 2
**Sample Output:** [3,4,1,2]



**Problem 4:** You are given a list of processes where each process has a **Process ID (PID)** and a **Burst Time** (the amount of time the process requires on the CPU). The CPU uses a **Round Robin Scheduling** algorithm with a fixed **time quantum**. Your task is to implement this scheduling using a circular linked list where each process is represented as a node in the list.

## Requirements:

1. **Circular Linked List Implementation:**
    - Implement a circular linked list to manage the processes.
    - Each node should represent a process and contain:
        - `PID`: Process ID
        - `Burst Time`: The time the process requires on the CPU
        - Pointer to the next process in the list
2. **Round Robin Scheduling Algorithm:**
    - Each process is allowed to run for the duration of the **time quantum**.
    - If a process finishes (its burst time becomes zero or less), it is removed from the list.
    - If a process does not finish in the given time quantum, subtract the quantum from its burst time and move to the next process in the list.
    - Continue the process until all processes have been completed.
3. **Input/Output:**
    - Input:
        - A list of processes where each process is defined by its **Process ID** and **Burst Time**.
        - A **Time Quantum** defines how long each process gets to execute in each cycle.
    - Output:
        - Print the remaining burst time of each process after each time quantum.
        - Print the total time taken for all processes to complete.

## Functionality:

Implement the following functionality:

1. **Node Class:**
   - This class should represent a process in the system and store the Process ID, remaining burst time, and a reference to the next process.
2. **CircularLinkedList Class:**
   - Functions:
     - `append(pid, burst_time)`: Add a new process to the circular linked list.
     - `remove(node)`: Remove a process from the list.
     - `display()`: Display all processes and their remaining burst time.
     - `is_empty()`: Check if all processes are completed (i.e., list is empty).
3. **Round Robin Scheduler Function:**
   - Simulate the Round Robin Scheduling by running each process for a maximum of the given time quantum.
   - After each cycle, update the burst time of the process, and if the burst time is zero, remove the process from the list.
   - Continue until all processes are completed, and print the total elapsed time.

## Constraints:

- The number of processes can range from 1 to 50.
- The burst time of any process will be between 1 and 100 units.
- The time quantum will be between 1 and 10 units.

## Example:

**Input:**

processes = [(1, 10), (2, 5), (3, 8)]  # Process ID, Burst Time
quantum_time = 4

**Expected Output:**

Processes in the list:
Process 1: Remaining Burst Time = 10
Process 2: Remaining Burst Time = 5
Process 3: Remaining Burst Time = 8

```
Starting Round Robin Scheduling:

Timestamp: 0, Currently processing PID: 1
Process 1 now has 6 units remaining.

Timestamp: 4, Currently processing PID: 2
Process 2 completed.

Timestamp: 8, Currently processing PID: 3
Process 3 now has 4 units remaining.

Timestamp: 12, Currently processing PID: 1
Process 1 now has 2 units remaining.

Timestamp: 16, Currently processing PID: 3
Process 3 completed.

Timestamp: 20, Currently processing PID: 1
Process 1 completed.

All processes completed.
Total time elapsed: 22 units.
```

## Submission Instructions

(Please follow the instructions carefully and submit accordingly.)

· Name your source code file as “FULL_NAME_HW1.py”
· Submit this file in iCollege folder ‘**Homework1**’
· Due date: Mon, 10/21/2024 11:59 PM
· Late submission will be accepted until: Thu, 10/24/2024 11:59 PM

The late submission penalty will be determined based on the following formula:

PENALTY = 0.4 * NUMBER_OF_HOURS_LATE

Examples:

If your submission is 2 hours late, PENALTY = 0.8%
If your submission is 24 hours late, PENALTY = 9.6%
If your submission is 72 hours late, PENALTY = 28.8%

*Note:*
*-All submissions must be made through iCollege. No email submission will be accepted.*

**Grading Breakdown:**

| **Task** | **Points** |
|---|---|
| Task 1 | 25 |
| Task 2 | 25 |
| Task 3 | 25 |
| Task 4 | 25 |

*Note: Add a few comments in your python script to explain how your code works. Comments should be meaningful and concise.*