**Lab Tasks**

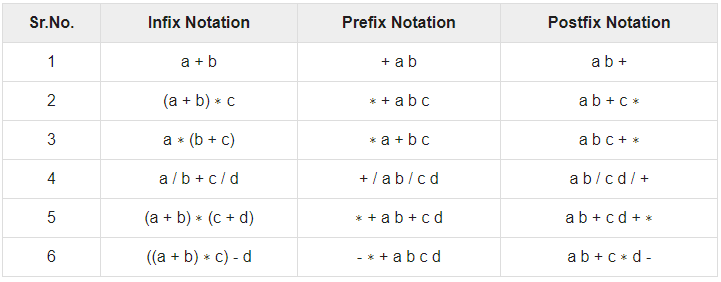
**Q1.** Design a Stack data structure using linked list, with a **top** pointer, and the following functions:

* Push (inserts an element into the stack **(at top)**)
* Pop (removes & returns the **top** element from the stack)
* Peek (displays the **top** element in the stack)
* isEmpty (returns whether stack is empty)

**Q2.** Design a Queue data structure using linked list, with **front** and **rear** pointers, and the following functions:

* Enqueue (inserts an element into the queue **(at rear)**)
* Dequeue (removes & returns the **front** element from the queue)
* Peek (displays the **front** element in the queue)
* isEmpty (returns whether queue is empty)

**Q3.** Create a function which takes a **prefix** expression as string as parameter, and returns the **postfix** expression. Note that your operands can be greater than 1 digit, deal with it accordingly by converting them into integer forms **by parsing through the string.**



**Q4.** Implement a priority queue by assigning “priority” values to every element in the queue. When a new element is enqueued, your queue must arrange itself accordingly (according to priority). Assume that a **smaller** priority value represents **higher** priority. Meaning a priority value of 1 will be dequeued **before** a priority value of 3. Do not confuse the **priority value** with the **actual value** of the element. They are 2 different things.

**Q5.** CPU Process Scheduling is a challenging task, which can be solved effectively with the right data structure. Some processes require resources urgently, while others may have been waiting for a long time despite not being that important. At any particular instance, CPU resources are limited. **Assume the following constraints:**

* Your CPU has 2 GB excess RAM to work with. If 2GB RAM has already been allocated and those processes are running, other processes must wait (unless higher priority).
* A process will either be in a waiting or running state.
* At any particular time, a process has a 15% chance of spawning. This process will have a certain priority level [1-20], require some RAM [1-500 MBs], take a certain amount of seconds to complete [1-10]. These will be randomly generated values.

**You must implement a CPU process scheduler, which handles these constraints and ensures that all the processes are allowed access to required resources, even if they must wait until it is freed by other processes.**

* Assume that 1 iteration of your main loop will represent the passing of 1 second. After 1 second has passed, you must check all the processes and update their remaining running times.
* In case a process spawns but RAM is not available, it must go into the waiting state.
* If a process spawned with higher priority but RAM is not available, and a lower priority process is using RAM, then the lower priority process must go into waiting state until the higher priority process has completed.
* If a process is waiting for RAM which is now available, it will go into running state.
* If a process has completed its running time, you must kill the process and free any RAM it was using.
* **After every second passes, you must print on the screen the current status of all processes, as well as the system resources in use (RAM). Run this simulation for 100 seconds.**

**Hint:** Read the question **carefully** and implement all the process **attributes**, as well as the functionalities required (system resources, process running times, process states).