Exercise 07 - Elementary Data Structures

The elementary data structures—stacks, queues, linked lists, sets, and binary (or rooted) trees—can be efficiently implemented using either an array with at most two index references (and an array length) or linked nodes with at most three links.

The header file '`Containers.h`' defines the generic classes *Array* and *Node*. Each class provides methods for accessing its references—first() and second() for *Array*, and first(), second(), and third() for *Node*—as well as methods for accessing their stored data—operator[] for *Array* and data() for *Node*. Additionally, *Array* contains a capacity() method that retrieves its capacity.

Create a C++ file named '`exercises07.cpp`' that includes '`Containers.h`' and defines various methods for each elementary data structure using the required implementation container.

1. Define the `Push()` (insertion) operation for a stack implemented using an *Array* container using a single index reference [first()]. The method must accept the container reference and the input element as parameters, and must correctly handle the case where the container is full due to its fixed capacity.

2. Define the `Dequeue()` (strictly removal) operation for a queue implemented with an *Array* container using two index references [first() and second()]. The method must take the container reference as a parameter and use a circular array strategy to ensure efficient space utilization.

3. Define the `RemoveFirst()` operation for a doubly linked list implemented with a single *Node* container using two link references (next and previous) [first() and second(), respectively]. The method shall accept a pointer reference to the container as a parameter and remove the last node using a null-terminated approach.

4. Define the `Contains()` operation for a set implemented as a singly linked list using a *Node* container with a single next link [first()]. The method shall accept the container pointer and the target element as parameters and return a boolean indicating whether the target element is present in the list.

5. Define the `Degree()` operation for a rooted tree represented as a *Node* container with three link references–parent, left child, and right sibling [implemented as a singly linked list] [first(), second(), and third(), respectively]. The method must accept the container pointer as a parameter and return the number of children of the node or -1 if the node is empty.