The assignment for week 4 of CXX01 is to implement a doubly linked list as a reusable component. This assignment should be completed in groups of at most two students.

1 Doubly Linked List

Your group will write a reusable component which implements a doubly linked list data structure. Your library will have to adhere to the requirements listed below.

- The library implements a doubly linked list.
- The data within a list node consists of a `void` pointer and thus the list becomes generic. A generic data structure can contain data of any type.
- Nodes can be added to the list:
  - at the beginning;
  - at the end;
  - after a given node;
  - before a given node.
- A given node can be deleted from the list.
- The list can be searched for the *first* node which contains data that meets a specific condition. A function which checks such a condition is called a predicate. You can use a function pointer to specify the predicate. See: `genericFind.c`.
- The list can be searched for the *last* node which contains data that meets a specific condition.
- The list can be searched for the first node which contains specific data *after* a given node.
- The list can be searched for the first node which contains specific data *before* a given node.
- The number of nodes can be retrieved.

- The list itself can be disposed of properly (without any memory leaks!)

Of course you are totally free to implement extra functionalities.

Design the API (Application Programmers Interface) for your doubly linked list carefully. Discuss it with a teacher to verify your design.

2 Testing

All functions should have proper unit tests and you must use a test framework (e.g. µnit, see: https://nemequ.github.io/munit/.) to manage your testing. Write the unit tests for a specific function *before* you write the implementation of the function. This is called a test driven approach.

3 Implementation ideas

You should be able to use more than one doubly linked list in your program, therefore all functions should have a parameter which identifies the list upon which the operation must be performed. It is a good idea to use a pointer to a specific `struct` e.g. `DoublyLinkedList` to identify a list. A minimal example is given in Listing 1 but feel free to add more data members.

```
typedef struct DllNodeTag
{
    void *data;
    struct DllNodeTag *next;
    struct DllNodeTag *previous;
} DllNode;

typedef struct
{
    DllNode *head ;
    DllNode *tail;
} DoublyLinkedList;
```

Listing 1: A possible setup for your doubly linked list data structure.