2 Programming Component (50 pt)

A *Doubly-linked List* (DLL) is typically a linked list consisting of nodes that point to the next node in the list as well as the previous node in the list. Doubly-linked lists are difficult to implement in Rust, due to ownership. In this exam, you will be implementing a doubly-linked list for integers in Rust by centralizing the node ownership.

A doubly linked list is defined as follows:

```
struct DLL {
    elems: Vec<Node>,
    first_last: Option<(usize,usize)>,
    len: usize
}
```

where a `Node` is defined as

```
struct Node {
    next: Option<usize>,
    data: i32,
    prev: Option<usize>
}
```

Essentially, all nodes of a DLL `dll` are owned by the centralized vector of nodes `dll.elems`. Each node contains the data, as well as a the *indices* of the next node, and the previous node. Furthermore, when the DLL is nonempty, the indices of the first and last element are stored.

Question 8 (15 pt): Write the function `get_elem_index_of` that will retrieve the index (in the vector) of the element at the provided index (in the doubly-linked list).

For example, say we have the doubly linked list where `elems = vec![n0,n1,n2]` and `first_last = Some((1,0))`, where `n0` has no next, and a previous of 2. `n1` has a next of 2 and has no previous. `n2` has a next of 0 and a previous of 1. In this example, the linked list beings at index 1 with n1 (as represented by the first element of the tuple in `first_last` being 0, and having no previous index). It then proceeds to the next element, which is at index 2, so `n2`. n2 is the middle element, pointing previously to index 1, and next to index 0. At index 0 is n0, the last node. So, calling `get_elem_index_of(dll,0)` would return `Some 1`. `get_elem_index_of(dll,1)` would return `Some 2`. `get_elem_index_of(dll,2)` would return `Some 0`. Calling with any other index would return None. More examples are provided in the tests.

Question 9 (20 pt): Write the function `insert_at` that will update the DLL to insert a given piece of data at a provided index. The length of the list should be increased, and all the pointers should be adjusted to reflect this insertion. The `push_and_get_index` function will be helpful here. If an invalid index is provided (in other words, the index is not between 0 and dll.length inclusive) you can do any behavior. We will always provide valid indexes while testing.

Question 10 (15 pt): Write the function `insert_all_at_front`. This function should take in a reference to a vector of values as the first input, and a mutable reference to a DLL. It should then insert each of those values into the front of the DLL (potentially via the `insert_at_front` function). Then, a reference to the node at index 0 should be returned. You must write the function signature for this question. The tests have been commented out to enable compilation.