Exercise 2

12 min

4p **a)** You are given the following structures which represent a Doubly Linked List *header* and *node*:

```c
typedef struct {
  int count;
  Node *first;
  Node *last;
} List;
```

```c
typedef struct node_type {
    int key;
    struct node_type *next;
    struct node_type *prev;
} Node;
```

The following function tries to insert a new node into a Doubly Linked List, at the front, but something is missing. The function `isEmpty()` returns 1 if the list has no node and 0 if it has at least one node. Add the missing code at line 7.

```c
int insertAtFront(List *listPtr, Node *p) {
    if (listPtr) {
        if (isEmpty(listPtr)) {
            listPtr->first = listPtr->last = p;
            p->prev = p->next = NULL;
        } else {
            // insert code here
        }
        listPtr->count++;
        return 1; // success
    }
    return 0; // failure
}
```

2p **b)** The structure of the node has changed and now has a new property called "parity" which is initially 0:

```c
typedef struct node_type {
    int key;
    int parity;
    struct node_type *next;
    struct node_type *prev;
} Node;
```

You are given the following function which tries to insert a node at the end of the list with a **twist**. Mark the newly added node's parity with 1 if it is on an even position and 0 if it is on an odd position in the Doubly Linked List. You can assume that the other nodes have the correct parity.

```
1  int insertAtRear(List *listPtr, Node *p) {
2      if (listPtr) {
3          if (isEmpty(listPtr)) {
4              listPtr->first = listPtr->last = p;
5              p->prev = p->next = NULL;
6          } else {
7              // write your code in here
8          }
9          listPtr->count++;
10         return 1; // success
11     }
12     return 0; // failure
13 }
```

Note: Parity does not depend on the value of the `key` property. It depends on the position of the node in the list.