2) (10 pts) DSN (Linked Lists)

Consider the problem of determining the number of changes in direction of a sequence of integers stored in a linked list. We define a change in direction as any three consecutive values a, b, and c in the sequence where either a > b and b < c, or where a < b and b > c. For example, the sequence 3, 8, 2, 2, 5, 6, 4, 8, 3 has 4 changes in direction: (3, 8, 2), (5, 6, 4), (6, 4, 8) and (4, 8, 3). Notice how consecutive equal values in a sequence affect whether an actual change of direction is detected by this definition. Write an **iterative** function that takes in a pointer to the front/head of a linked list, ***front***, and returns the number of changes of direction, as defined above, in the sequence of integers in the linked list pointed to by ***front***. Note: lists of size 0, 1 or 2, by definition, have no changes of direction.

```
typedef struct node {
    int data;
    struct node* next;
} node;

int numDirChange(node *front) {

    // Grading: 3 pts for these initial cases.
    if (front == NULL || front->next == NULL || front->next->next == NULL)
        return 0;

    // Grading: 1 pt to set up some storage before loop.
    node* a = front;
    node* b = a->next;
    node* c = b->next;
    int res = 0;

    // Grading: Loop mechanics - 2 pts
    while (c != NULL) {

        // 2 pts for this check.
        if (a->data > b->data && b->data < c->data) res++;

        // 2 pts for this check.
        if (a->data < b->data && b->data > c->data) res++;

        // This is part of the loop mechanics.
        a = b;
        b = c;
        c = c->next;
    }

    // Grading - 0 pts, being nice here if someone forgets this, since
    // points are all allocated.
    return res;



}
```