CS 120 (Fall 21): Introduction to Computer Programming II

Short Project #6

due at 5pm, Thu 30 Sep 2021

1 Background: Doubly-Linked Lists

At this point, you've had some practice with singly-linked lists. We've mentioned that there is such a thing as a "doubly-linked list," and this Short Project will give you a tiny bit of experience with one.

As the name suggests, each node in a doubly linked list has two pointers: one to the next node (similar to singly-linked lists), and one to the previous node. Having this additional pointer simplifies some operations (such as removing a node from the middle of the list) but it also means that you have to do more work to make sure that the prev pointers are always correct.

1.1 What I'll Provide

I will provide a file, `dlist_node.py`, which defines a class, `DListNode`. It has three public fields: `val,next,prev` .

1.2 Special Limitation

Doubly-linked lists are special, because it's easy to modify the list, without having to iterate through it - because you can directly find your "prev" and "next" nodes. In this Project, you will be given a node, and told to "add before" or "add after" - and so it won't really be necessary to search through the list to find the places where you are making changes.

To force everybody to think this way, I'm placing a strange requirement: **no loops,** but also **no recursion!** So you cannot scan through the list - you will have to make small, local changes!

1.3 `utils.py`

You will notice a special file in the zipfile, named `utils.py`. This contains a set of utility functions that are used by the testcases. Please include it in the same directory as the testcase, any time that you want to run one.

(spec continues on next page)

1.4 Hints

Reference diagrams will help you! Changing the links in a doubly-linked list can sometimes be tricky, and it's easy to get it wrong. (That's why I don't have you do it at first.) But if you draw a reference diagram, this will help you figure out exactly what needs to change.

Temporary variables are useful! When you are re-arranging the links between multiple objects, it's sometimes useful to create a bunch of small temporaries. That can make it easier to keep track of all of the objects, and easier to set all of the fields to the correct other objects:

```
a = ... some object ...
b = ... another ...
c = ... another ...
a.next = ... something ...
a.prev = ... something ...
...
```

2 Inserting into a Doubly-Linked List

In a file `dl_insert.py`, write two functions: `dl_insert_before()` and `dl_insert_after()`. In both cases, they will take three parameters:

- A pointer to the head of a doubly-linked list. **You may assume that this is not** `None`.
- A pointer to a node in that same list. This may be the head, or any other node; you may assume that it is not `None`.
- A newly-created node (one where the `next,prev` pointers are both `None`.

In each case, the function should return a pointer to the head of the resulting list, after the insertion has happened.

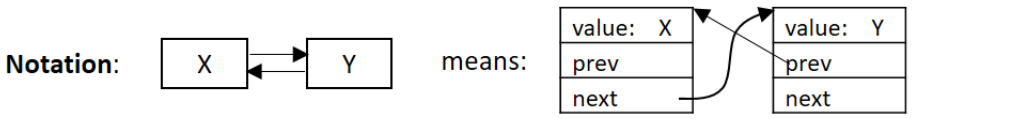
Note that neither function is required to create any new nodes; you will be passed the node to insert into the list.

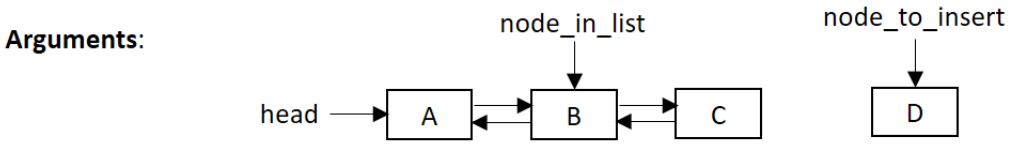
(spec continues on next page)

2.1 Requirements

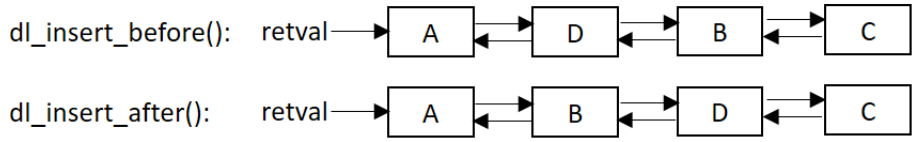
The function `dl_insert_before()` must insert the new node (that is, the third parameter) **ahead** of the pointed-to node in the list (that is, the second parameter). The function `dl_insert_after()` must insert the new node **after** the pointed-to node.

EXAMPLES





Return value:



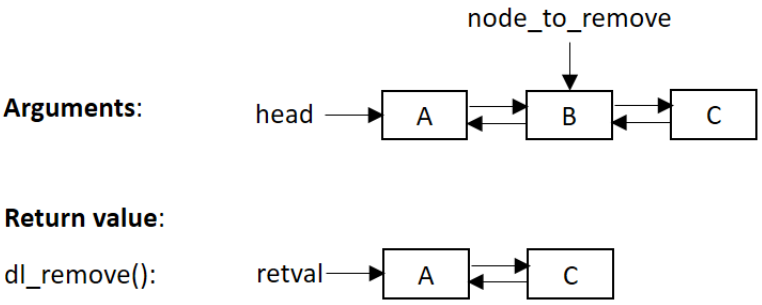
(spec continues on next page)

3 Removing from a Doubly-Linked List

In a file `dl_remove.py`, write a function `dl_remove()`, that removes a node from a doubly-linked list. As with the insert functions above, the first parameter is a pointer to the head of the list (which will not be `None`), and the second is a pointer to a node inside that list.

This function will remove the node from the list, and return the head of the list after the update. (Note that it's possible that the list might be empty, if the last node gets deleted.)

EXAMPLES



4 Turning in Your Solution

You must turn in your code using GradeScope. Turn in the following files:

- `dl_insert.py`
- `dl_remove.py`

5 Acknowledgements

Thanks to Saumya Debray and Janalee O'Bagy for many resources that I used and adapted for this class.