National University
of Computer & Emerging Sciences
Peshawar Campus

Problem Set:	Assignment: A03	**Semester**:	Spring 2019
Points:	*10*		
Date Set:	*See SLATE*	**Due Date**:	*See SLATE*
Course:	CS217 - OOP	**Instructor**:	Dr. Nauman

1 A Class for a List

In this assignment, we are going to work with some existing code and finish it to achieve the following goal.

We want to create a special type of list which allows us to create something similar to a stack. Recall that a stack is a LIFO structure – which means that the value we put in *last* comes back out *first*.

We are going to create a new class called `List`. This class is going to hold a list that we created during lectures. For instance, we can use this class to hold a list which has the following elements:

`[ 1 4 30 20 ]`

The basic three operations are the following:

1. `push`: This inserts a new element at the *end* of the list. For the above list, if we `push` 7 to it, the list will then look like the following: `[ 1 4 30 20 7 ]`
2. pop: Removes the *last* element from the list and returns it. For instance, after the `push` above, if we pop from the list, the list will look like `[ 1 4 30 20]` again and the function will return 7 to us.
3. `print_list`: Simply outputs the list on the console in the above format.

Now take a look at the starter file. It already has the code for the `push` function defined. However, quite a few pieces are missing outside it which it needs. Let's do that first.

1. Define the node structure.
2. Create two node pointers `head` and `last` as class members (making sure you initialize them). The `head` is obvious. The `last` pointer is used to keep track of the last node of the list – this will be useful later.
3. Define the `print_list` function, which outputs the whole list as above.

After the above, you should be able to execute Part - 1 of the starter file. Make sure it runs properly and outputs the list as desired. Also make sure you understand the code of the `push` function. It has two cases: one where the list is empty and the other when there's already an element in the list. We've done the second case during lectures but you will need to think just a little bit about how the first case works.

2 Pop

The second function, which is your main task, is to code the pop function. Let's do this in two steps.

2.1 Case: More Than One Nodes

After Part - 1 executes, your list should look like the following:

`[ 5 15 30 ]`

Our pop function should return 30 but also remove it from the list. Here's how you can do it:

1. We already did a `delete_after_node` function during lectures. First, create a new `delete_after_node` method in class. You can copy the body for the method exactly from what we did during lectures.
2. Then, you need to first save the value of the *last* node in some variable for returning it later.

3. Finally, you need to find the node that is *just before the last* and send that to the `delete_after_node` method. That way, the last node will be deleted from the list and then you can return the value you saved earlier.

After doing this, you should be able to call `pop` two times from the main function on the above list. Try that and make sure you get the following list as a result.

```
[ 5 ]
```

Issue: This logic breaks down when you have just one node in the list because we can't find the *previous* node if there is only one node. Let's handle that in the next section.

2.2 Case: A Single Node

We need to modify the `pop` function you wrote above for this. First, we need to check if there is only one node in the list. This is very straight-forward. If the `next` pointer of the `head` is `NULL`, we have just one node in the list. So, now, in the `pop` function, we are going to decide: if we have one node, we do the following logic (and if we have more than one nodes, we have already done that above). You might want to take a look at `push` to get an idea of how two cases are handled.

For this case, the logic is again quite simple. Just save the value and delete `head` node (making sure the `head` and `last` pointer are not dangling).

Once this is done, you should be able to `pop` once more from the list and get an empty list.

3 Submission

After you're done making your code work, write down (**read: assignment must be hand-written**) the full code you've created and submit that on or before the deadline. Late assignments (or soft copies) will not be accepted.