Pset listnode: a singly-linked list

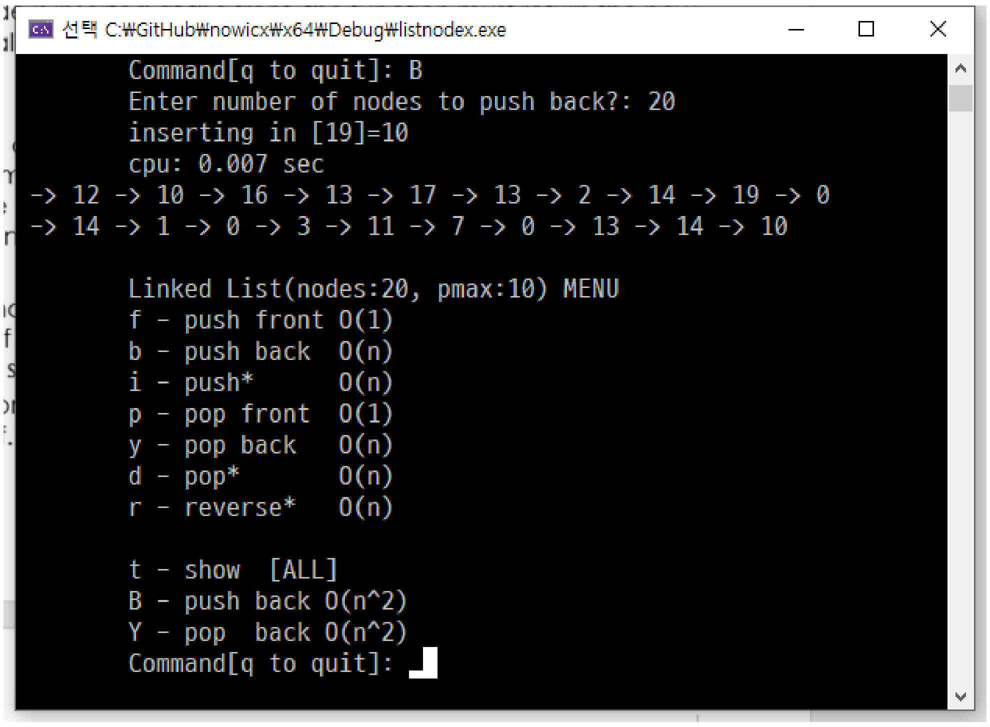
Table of Contents

Getting Started

This problem set consists of implementing a simple singly-linked list of nodes. Your job is to complete the given program, **listnode.cpp**, that implements a singly linked list that don't have a header structure nor sentinel nodes. It simply links node to node and the first node always becomes a head node.

- listnode.cpp – a skeleton code, most of **your implementation goes here**.
  listnode.h – an interface file, you are **not** supposed to modify this file.
- listnodeDriver.cpp – a driver code to test your implementation.
- listnodex.exe, listnodex – a sample solution for pc or mac

Sample Run:.



```
        Command[q to quit]: B
        Enter number of nodes to push back?: 20
        inserting in [19]=10
        cpu: 0.007 sec
-> 12 -> 10 -> 16 -> 13 -> 17 -> 13 -> 2 -> 14 -> 19 -> 0
-> 14 -> 1 -> 0 -> 3 -> 11 -> 7 -> 0 -> 13 -> 14 -> 10

        Linked List(nodes:20, pmax:10) MENU
        f - push front O(1)
        b - push back  O(n)
        i - push*      O(n)
        p - pop front  O(1)
        y - pop back   O(n)
        d - pop*       O(n)
        r - reverse*   O(n)

        t - show  [ALL]
        B - push back O(n^2)
        Y - pop  back O(n^2)
        Command[q to quit]: 
```

Step 1: push_front(), push_back(), push()*

The first function you must implement is **push_front()** which insert a node in front of the list. Since we don't have any extra header structure to maintain, the first node always acts like the head node. Since the new node is inserted at the front, the function must return the new pointer to the head node to the caller. The caller must reset the first node information. This is O(1) operation.

For **push_back()** function, we must add a new node to the end of the existing list of nodes. If no list exists, the new node becomes the head node. To add a new node at the end, you must go through the list to find the last node. Once you find the last node, then you may add the new node there. This is O(n) operation.

The function **push()** inserts a new node with **val** at the position of the node with **x**. The new node is actually inserted **in front of the node with x**. It returns the first node of the list. This effectively increases the container size by one.

- If the list is empty, the new node with **val** becomes the first node or head of the list. In this case, the function argument **x** is ignored.
- If a node with **x** is not found in the list, it returns the original head pointer.
- Pay attention when you have just one node and push a node in that position. In this case, you can find the positional node with x, but there is no **prev** node to link the new node with **val**.

Hint: To insert a new node at (in front of) the node with x, you must have both the previous node of x and the node with x itself. Since the exactly same thing is used during **clear()**, you may refer to the code example in **clear()** provided.

```
pNode push(pNode p, int val, int x);
```

Step 2: pop_front(), pop_back(), pop()

These functions delete a node from the linked list of nodes. The **pop_front()** removes the first node in the list and the second node becomes the first node or head node. This function return the pointer the newly first node which used to be a second node. This is O(1) operation.

The **pop_back()** removes the last node in the list. To remove the last one, you must go through the list to locate the **last node** as well as the **previous node**. Since the previous node of the last node becomes the last node, we must set its **next field to nullptr**. Therefore, you must get the last node as well as the previous node of the last one.

The **pop()** deletes a node with a specific value that the user choose. It could be any node in the list.

Hint: To remove a node with x, you must have both the previous node of the node x and the node x itself. Since the exactly same thing is used during **clear()**, you may refer to the code example in **clear()** provided.

Grace Step: show() and clear() – same as the last pset

The **show()** function displays the linked list of nodes. The function has an optional argument called **bool all = true**. Through the menu option in the driver, the user can toggle between "show [all]" and "show [head/tail]". This functionality is useful when you debug your code.

The show() function has another optional argument i**nt pmax = 10**. If the size of the list is less than or equal to pmax *2, then it displays all the items in the list no matter which **option "all"** was chosen. If show[all] option is chosen, then display them all. If show[HEAD/TAIL] option is chosen, it displays the pmax items at most from the begging and the end of the list.

The **clear()** function you must implement deallocates all the nodes in the list. Make sure that you call "delete" N times where N is the number of nodes.

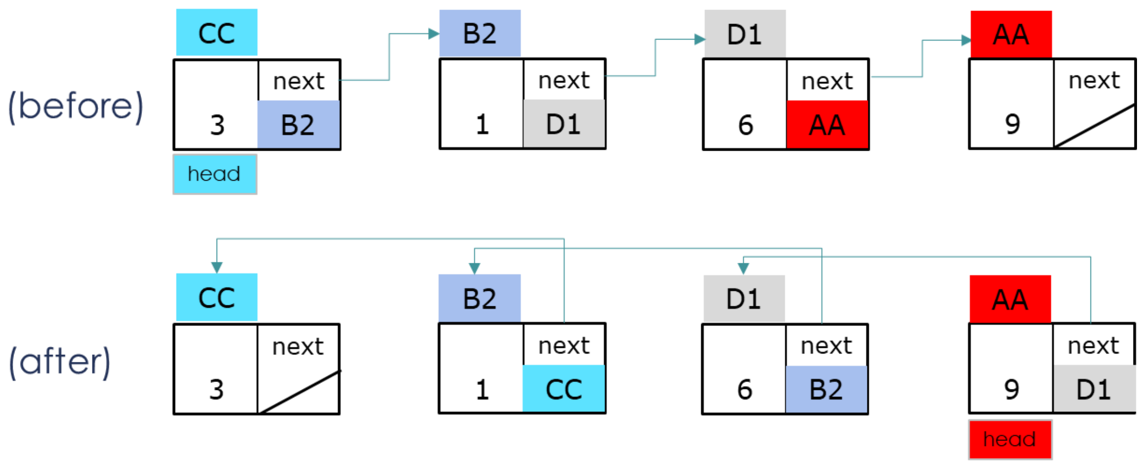
Step 3: Stress test – "B" and "Y"

Implement two test codes to test your implementation of a couple of functions.

- There are two options:
  "B – push back, O(n^2)"
  "Y – pop back, O(n^2)"
- Each option asks the user to specify the number of nodes to be added or removed and performs the task. In "Y' option, if the user specifies a number out of the range, it simply removes all the nodes.

Step 4: Reverse a singly-linked list

Reverse a singly-linked list in O(n). The function reverses a singly-linked list and returns the new head. The last node of the input list becomes the head node of the newly formed list. Since it goes through the list once, the time complexity of this function is O(n).



Tips and Hints:

While you go through the list and swap two pointers of two nodes, you must save a couple of pointers before you make assignments.

Before while() loop, set prev = nullptr, and curr = head. During while() loop,

(1) Before setting curr→next to a new pointer, store curr→next as a temporary node temp.

(2) Before going for the next node in while loop, make sure two things:

   A. set prev to curr (e.g. curr becomes prev)
   B. set curr to the next node you will process.

Submitting your solution

- Include the following line at the top of your every source file with your name signed.
  On my honor, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.
  Signed: _____________ Section: _______ Student Number: _________________

- Make sure your code **compiles** and **runs** right before you submit it. Every semester, we get dozens of submissions that don't even compile. Don't make "a tiny last-minute change" and assume your code still compiles. You will not get sympathy for code that "almost" works.
- If you only manage to work out the problem sets partially before the deadline, you still need to turn it in. However, don't turn it in if it does not compile and run.
- Place your source files in the folder you and I are sharing.
- After submitting, if you realize one of your programs is flawed, you may fix it and submit again as long as it is **before the deadline**. You will have to resubmit any related files together, even if you only change one. You may submit as often as you like. **Only the last version** you submit before the deadline will be graded.

Files to submit

Submit **one** file listed below.

- listnode.cpp
- Upload your file **pset9** folder in Piazza.

Due and Grade

- Due: 11:55 pm, Oct. **30**, 2019
- Grade: 1 point per step.