# listnode – a simple linked list of nodes

## Table of Contents

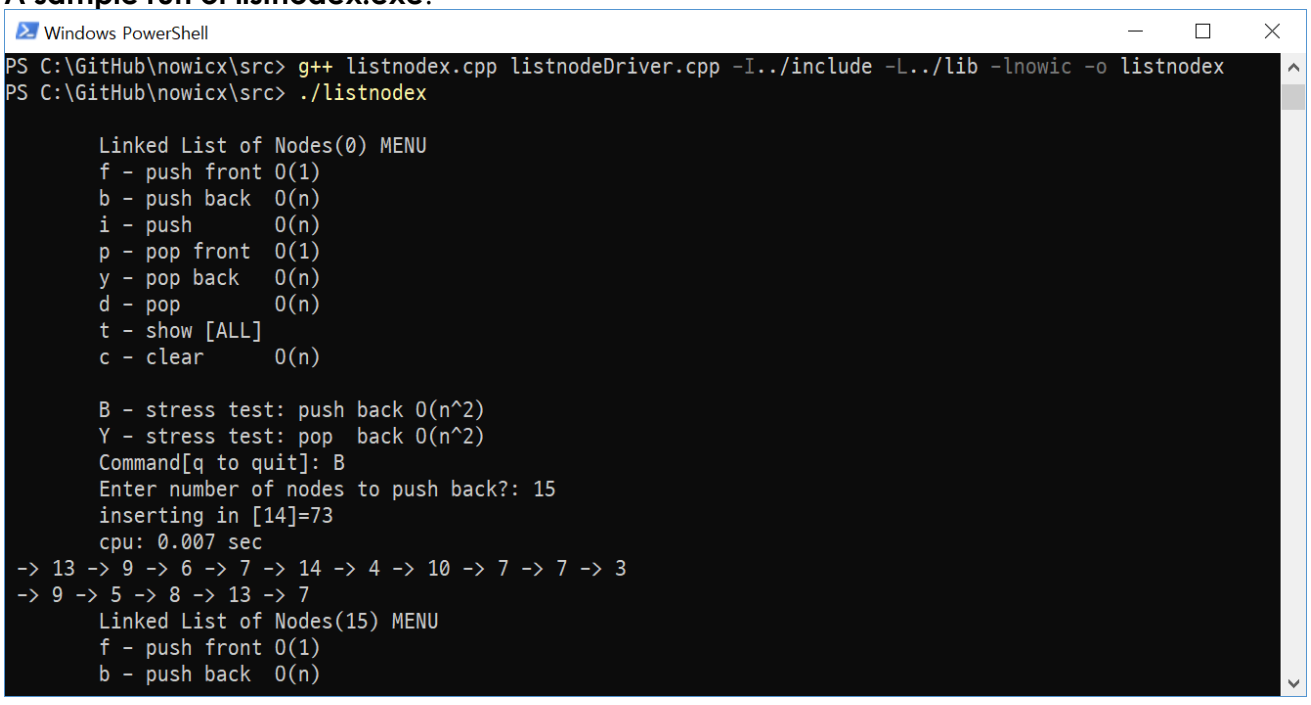


## Getting Started

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 – a sample solution

**A sample run of listnodex.exe**.



## 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.

**Hint: 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 nullpr**. Therefor 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.

## Step 3: show() and clear()

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 [all]" option is already implemented, your job is to implement the "show [head/tail]" option. This option displays the pmax = 10 items from the begging and the end of the list. The pmax means the maximum number of items displayed per line. If the size of the list is equal to pmax * 2 (or 20) or less, then displays them all.

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 4: Stress test – "B" and "Y"

Don't start this step unless you finish Step 1 through Step 3 completely.

We want to implement a piece of code that tests your implementation of functions you have done so far.

- There are two options:
  "B – stress test: push back, O(n)"
  "Y – stress test: pop back, O(n)"
- "B" option asks the user to specify the number of nodes to be added and performs the task. This option is already implemented for you.
- "Y" option that you must implement asks the user to specify the number of nodes to be removed and performs the task. If the user specifies a number out of the range, it simply removes all the nodes.

## 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 **hw7** folder in Piazza.

## Due and Grade

- Due: 11:55 pm, April **24**, 2019
- Grade: 1 point per step.