The following materials have been collected from the numerous sources including my own and my students over the years of teaching and experiences of programming. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Please send any comments or criticisms to idebtor@gmail.com. Your assistances and comments will be appreciated.

PSet listdbl: a doubly-linked list

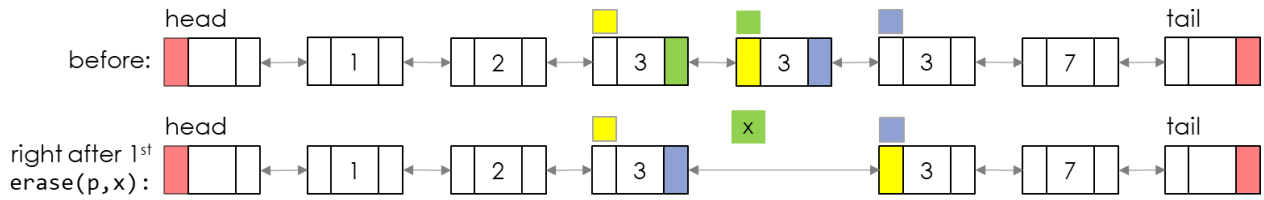
Table of Contents

- This Pset is an extension of Lab8.
  It covers the Advanced Operations of Doubly Linked Lists.
- You continue on coding by using listdbl.cpp that you coded in Lab8.
  - listdbl.h – Don't change this file.
  - driver.cpp – Don't change this file.
  - listdbl.cpp – Begin with the file you coded in Lab8 file.
  - listdblx.exe, listdblx – Provided for your references only. It may have bugs.

Step 1: unique()*

This function removes extra nodes from the list that has duplicate values It removes all but the **first** node from every **consecutive group** of equal nodes in the list. Notice that a node is only removed from the list if it compares equal to the node immediately **preceding** it. Thus, this function especially works for sorted lists in either ascending or descending. It should be done in O(n).

For example, the second and third occurrence of 3 must be removed in the following list.



A skeleton code is provided with some bugs

```
void unique(pList p) {
  if (size(p) <= 1) return;
```

```
  for (pNode x = begin(p); x != end(p); x = x->next)
    if (x->data == x->prev->data) erase(p, x);
} // version.1 buggy - it may not work in some machines or a large list.
```

To debug the skeleton code, answer the following question by yourself:
"Can x point the next node 3 **right after the first erase(p, x)**?"

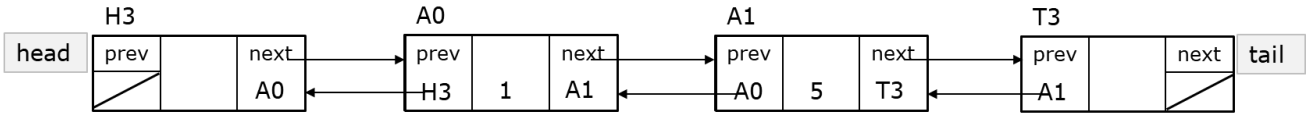
Step 2: reverse()

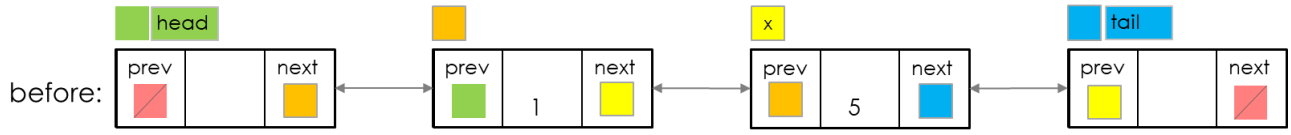
This function reverses the order of the nodes in the list. The entire operation does not involve the construction, destruction or copy of any node. Nodes are not moved, but pointers are moved within the list. It must perform in O(n),

```
// reverses the order of the nodes in the list. Its complexity is O(n).
void reverse(pList p) {
  if (size(p) <= 1) return;
// your code here
}
```

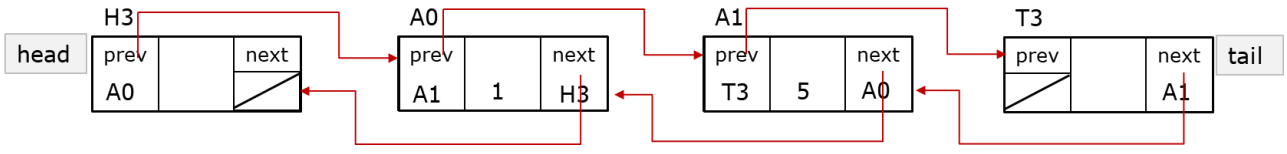
It may be the most difficult part of this pset. The following diagram shows two step procedures that may help you a bit.

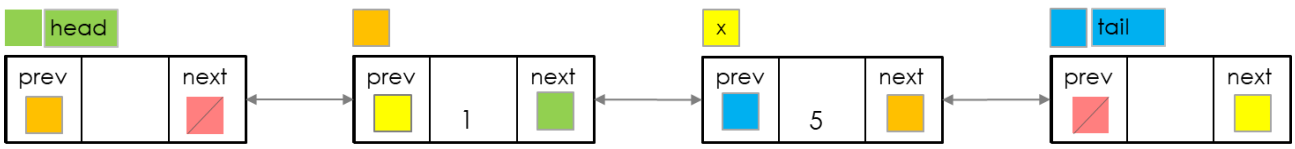
Original list given:





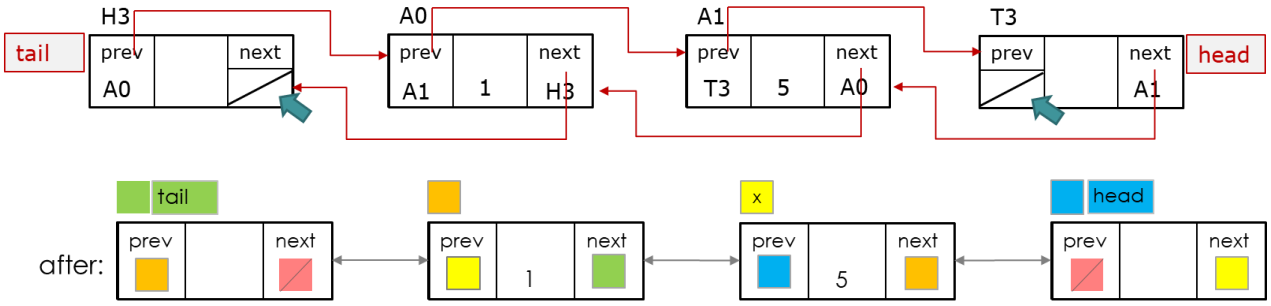
step 1: swap prev and next in every node.





Notice that prev and next in sentinel nodes are also swapped.

step 2: swap head and tail node.



Step 3: Randomize()

There are many ways to implement this function. The most well-known algorithm is called the Fisher-Yates shuffle. You may refer to Wikipedia or **random.cpp** for detail. **The naive method** we use here swaps each element with another element chosen randomly from all elements. Even though this method is not the best, however, it is still acceptable for our practice purpose now.

The following functions are already implemented and given in the skeleton code.

```
// a helper function
pNode find_by_index(pList p, int n_th) {
  pNode curr = begin(p);
  int n = 0;
  while (curr != end(p)) {
    if (n++ == n_th) return curr;
    curr = curr->next;
  }
  return curr;
}

void randomize(pList p) {
  int N = size(p);
  if (N <= 1) return;
  pNode curr = begin(p);
  srand((unsigned)time(nullptr));

  curr = begin(p);
  while (curr != end(p)) {
    int x = rand_extended() % N;
    pNode xnode = find_by_index(p, x);
    swap(curr->data, xnode->data);
    curr = curr->next;
  }
}
```

The time complexity of the randomize() function shown above is O(n^2).

Your job is to rewrite this code such that its time complexity becomes O(n). The main culprit of O(n^2) is to use find_by_index() O(n) inside while loop.

One way to implement it is to randomize the list and save them into an array first at the same time. You may use the same approach used in the Fisher-Yates shuffle "inside-out" algorithm defined in nowic/src/rand.cpp. Then, go through the list again to overwrite the values in list with aux[] which is already randomized.

Since it goes through the list twice, the time complexity becomes O(n) + O(n). One is to randomize the list and save them into an array aux[], and the other is to put them back in the list. Therefore the time complexity overall for the function is O(n).

Submitting your solution

- Include the following line at the top of your every file with your name signed. On my honour, I pledge that I have neither received nor provided improper assistance in the completion of this assignment. Signed: _____
- Make sure your code **compiles** and **runs** right before you submit it.
- If you only manage to work out the homework 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 may submit as often as you like. **Only the last version** you submit before the deadline will be graded.

Files to submit

- Submit the following files.
  - listdbl.cpp
- Use pset folder in piazza to upload your files.

Due and Grade points

- Due: 11:55 pm