CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2021
Webpage: http://www.cs.clemson.edu/~bcdean/ — MWF 9:05-9:55
Handout 3: Lab #2 — Powers 208

1 Linked List Practice

The goal of this lab is to practice implementing simple manipulations of linked lists — a useful skill in any computer scientist's repertoire. In lecture, we randomly selected 5 out of 10 practice tasks to attempt. Each is shown below as a function prototype and a comment explaining what the function should do:

```
// Insert v into a linked list that is sorted, keeping it sorted
// Returns a pointer to the head node in the resulting list
Node *insert_keep_sorted(Node *head, int v);

// Insert a new node with value v at index i (e.g., i==0 to insert a
// new head node; the node formerly at index i would become index i+1).
// Return a pointer to the head node in the resulting list
// You may assume i is a valid index from 0 up to the list length
Node *insert_at_pos(Node *head, int i, int v);

// Delete the element at index i (e.g., if i==0, delete the head node)
// Return a pointer to the head node in the resulting list
// You may assume i is a valid index from 0 up to the list length minus one
Node *delete_at_pos(Node *head, int i);

// Given a linked list that is sorted except with one element "out of
// order" (too high or too low), restore the list to sorted order.
// Return a pointer to the head node of the resulting list
Node *restore_sorted (Node *head);

// Build a linked list containing a copy of the elements in
// positions i...j of a linked list and return it.  You may assume
// i <= j and that i and j are valid indices (between 0 and the
// list length minus one).
Node *splice(Node *head, int i, int j);
```

2 Coding Guidelines

Starter code with a `Node` structure definition is given in

`/group/course/cpsc212/f21/lab02/linkedlist.cpp`

The starter code also has functions to generate random lists and sorted lists of any given size, and also to print a list.

Here are some requirements and guidelines for your code:

- Submit all your code in one file, `lab2.cpp`.
- Use the exact same function headers from the previous page (that's how we'll be testing your code). Your functions can call "helper" functions though if they want.
- For full credit, all of your functions should run in $O(n)$ time.
- You can write your functions recursively if you want.
- Watch out for potential special cases — for example, inserting or deleting at the beginning or end of a list. We have a habit of checking those types of things when grading...
- Don't make any unnecessary assumptions, for example that a list contains only distinct values.
- Your code shouldn't leak memory or allocate / de-allocate memory beyond what is required (e.g., if you are inserting a new entry in a linked list, you should generally be allocating just one new node structure in the process). If you are deleting from a list, you should de-allocate the node being deleted. Be sure not to reference memory after you de-allocate it with `delete`.

3 Submission and Grading

Code should be submitted electronically on on `handin.cs.clemson.edu` by 11:59pm on the evening of Sunday, September 5. No late submissions will be accepted. Ten possible points are given for this lab, split evenly among the five tasks.

4 Food for Thought

Here are the five questions not selected for this lab, which you might want to consider optionally practicing on your own time:

- Interleave two equal-length linked lists.
- Delete all copies of x from a linked list.
- Given a linked list with distinct values, find the element 17 positions before the element of value x (NULL if x does not appear in the list, or if it appears too early).
- Make a copy of a linked list, leaving out all the even elements.
- Given two sorted linked lists, find the middle element between them both (i.e., the element that would be in the middle if they were merged into one sorted list).