Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Winter 2016

Lab 10 - More Linked List Exercises

Objective

To gain additional experience designing and implementing functions that operate on singly-linked lists.

Attendance/Demo

To receive credit for this lab, you must demonstrate your solutions to the exercises. **Also, you must submit your zipped project file to cuLearn**. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

General Requirements

You have been provided with three files:

- singly_linked_list.c contains incomplete implementations of the two functions you'll implement during this lab. It also contains function `print_linked_list`, which prints a linked list of integers using the format:

  ```
  value1 -> value2 -> value3 -> ...
  ```

  You can insert calls to this function in your solutions to the exercises, to help you debug your code.

- singly_linked_list.h contains declarations for a singly-linked list data structure and prototypes for functions that operate on this linked list;

- main.c contains a simple *test harness* that exercises the functions in singly_linked_list.c. Unlike the test harnesses provided in previous labs, this one does not use the sput framework. The harness doesn't compare the actual and expected results of each test and keep track of the number of tests that pass and fail. Instead, the expected and actual results are displayed on the console, and you have to review this output to determine if the functions are correct.

Finish each exercise (i.e., write the function and verify that it passes all of its tests) before you move on to the next one. Don't leave testing until after you've written all your functions.

None of the functions you write should perform console input; i.e., contain `scanf` statements.

Unless otherwise specified, none of your functions should produce console output; i.e., contain `printf` statements.

Instructions

1. Launch Pelles C and create a new Pelles C project named linked_list_lab_10 (all letters are lowercase, with underscores separating the words). The 64-bit version of Pelles C is installed on our lab computers, so the project type must be Win 64 Console program (EXE). (Although the 64-bit edition of Pelles C can build 32-bit programs, you may run into difficulties if you attempt to use the debugger to debug 32-bit programs.) If you're using your own computer, the project type should be Win 64 Console program (EXE) or Win32 Console program (EXE), depending on whether you installed the 64-bit or 32-bit edition of Pelles C. After creating the project, you should have a folder named linked_list_lab_10. Check this. If you do not have a project folder named linked_list_lab_10, close this project and repeat Step 1.

2. Download file main.c, singly_linked_list.c and singly_linked_list.h from cuLearn. Move these files into your linked_list_lab_10 folder.

3. You must add main.c and singly_linked_list.c to your project. From the menu bar, select Project > Add files to project... In the dialogue box, select main.c, then click Open. An icon labelled main.c will appear in the Pelles C project window. Repeat this for singly_linked_list.c.

   You don't need to add singly_linked_list.h to the project. Pelles C will do this after you've added main.c.

4. Build the project. It should build without any compilation or linking errors.

5. Execute the project. The test harness will show that the solutions to the exercises do not produce correct results (look at the output printed in the console window and, for each test case, compare the expected and actual results). This is what we'd expect, because these functions are incomplete.

The two exercises in this lab were taken from recent final exams. These questions clearly identified those students who understand how to design linked-list functions (in contrast to those who attempted to cobble together functions by replicating and rearranging code fragments from memorized lecture examples and lab solutions).

Please read each problem statement carefully, including the hints, before starting to develop your solutions.

Exercise 1

Write a function named `remove_duplicates` that is passed a pointer to a singly-linked list. The function prototype is:

```
void remove_duplicates(IntNode *head);
```

Memory for all the nodes was allocated from the heap.

The `remove_duplicates` function traverses the linked list. If any two adjacent nodes contain the same value, the second node is removed from the list. The removed nodes must be deallocated properly; in other words, your function must not cause memory leaks.

For example, suppose `my_list` points to this linked list:

```
3 -> 3 -> 1 -> 4 -> 4 -> 4 -> 3 -> 3 -> 5
```

and `remove_duplicates` is called this way:

```
remove_duplicates(my_list);
```

When the function returns, `my_list` points to this linked list: `3 -> 1 -> 4 -> 3 -> 5`.

The function terminates (via `assert`) if it is passed a pointer to an empty linked list.

Your function must be iterative. Do not write a recursive solution.

Hint: A complete solution requires no more than 15-20 lines of code. A solution is that is longer than this is likely more complex than it needs to be. This function only needs to make one traversal of the linked list; however, don't use a

```
for (p = head; p != NULL; p = p->next)
```

loop to do the traversal. (If you use this loop, your function will almost certainly be more complicated than necessary.) We recommend that you design the function by drawing some memory diagrams that clearly show each step in the solution, before you write any code.

Exercise 2

Write a function named `reverse` that is passed a pointer to a singly-linked list. The function prototype is:

```
IntNode *reverse(IntNode *head);
```

This function reverses the linked list by modifying the `next` members in all the nodes, and returns a pointer to the first node in the reversed linked list. For example, suppose `short_list` points to this linked list:

```
1 -> 2 -> 3 -> 4
```

and `reverse` is called this way:

```
short_list = reverse(short_list);
```

When the function returns, `short_list` points to the first node in this linked list:

```
4 -> 3 -> 2 -> 1
```

The function returns a `NULL` pointer if it is passed a pointer to an empty linked list.

Your function is not permitted to allocate new nodes or free nodes. Your function is not permitted to reverse the list by moving the integers stored in the `value` members of the nodes. Your function must be iterative (do not write a recursive function).

Hint: There are at least two different solutions, each requiring fewer than 15 lines of code. A solution is that is longer than this is likely more complex than it needs to be. This function only needs to make one traversal of the linked list; however, don't use a

```
for (p = head; p != NULL; p = p->next)
```

loop to do the traversal. (If you use this loop, your function will almost certainly be more complicated than necessary). We recommend that you design the function by drawing some memory diagrams that clearly show each step in the solution, before you write any code.

Wrap-up

1. Remember to have a TA review and grade your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

2. The next thing you'll do is package the project in a ZIP file (compressed folder) named linked_list_lab_10.zip. To do this:

    2.1. From the menu bar, select Project > ZIP Files... A Save As dialog box will appear. If you named your Pelles C project linked_list_lab_10, the zip file will have this name by default; otherwise, you'll have to edit the File name: field and rename the file to linked_list_lab_10 before you save it. **Do not use any other name for your zip file** (e.g., lab10.zip, my_project.zip, etc.).

    2.2. Click Save. Pelles C will create a compressed (zipped) folder, which will contain copies of the the source code and several other files associated with the project. (The original files will not be removed). The compressed folder will be stored in

your project folder (i.e., folder linked_list_lab_10).

3. Log in to cuLearn and submit linked_list_lab_10.zip. To do this:

   3.1. Click the Submit Lab 10 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag linked_list_lab_10.zip to the File submissions box. **Do not submit another type of file (e.g., a Pelles C .ppj file, a RAR file, a .txt file, etc.)**

   3.2. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status of your file is "Draft (not submitted)". If you're ready to finish submitting the file, jump to Step 3.4. If you aren't ready to do this; for example, you want to do some more work on the project and resubmit it later, you can leave the file with "draft" submission status. When you're ready to submit the final version, you can replace or delete your "draft" file submission by following the instructions in Step 3.3, then finish the submission process by following the instructions in Step 3.4.

   3.3. You can replace or delete the file by clicking the Edit my submission button. The page containing the File submissions box will appear.

      3.3.1. To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

      3.3.2. To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

   3.4. Once you're sure that you don't want to make any changes, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading".