Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Fall 2013

Lab 10 - Linked Lists

Objective

To develop some functions that operate on linked lists.

Attendance/Demo

To receive credit for this lab, you must make the effort to finish a reasonable number of exercises and demonstrate the code you complete.

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. Finish any exercises that you don't complete by the end of the lab on your own time. Also, you must submit your lab work to cuLearn by the end of the lab period. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

General Requirements

In Exercises 1 through 3, you are going to define functions that operate on lists of integers that are implemented using a singly-linked list data structure. You will also write the tests for these functions.

Finish each exercise (i.e., write the function that performs the specified list operation, 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 define in singly_linked_list.c 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.

You have been provided with three files:

- singly_linked_list.c contains several functions that operate on singly-linked lists, many of which were presented in lectures. This module contains a `print_linked_list` function, which prints a linked list of integers using the format:

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

  In addition, this file has functions that:

  - search a linked list to determine if it contains a specified value;
  - insert an integer at the front of a linked list;
  - append an integer to the rear of a linked list;
  - remove the first node in a linked list;

    - remove the last node in a linked list.

- singly_linked_list.h contains declarations for the linked list data structure and the function prototypes;

- test_singly_linked_list.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 compare the actual and expected results of each test and keep track of the number of tests that pass and fail. Instead, results are displayed on the console, and you have to review this output to determine if the functions are correct.

Instructions

1. Launch Pelles C and create a new Pelles C project named linked_list. The project type must be Win32 Console program (EXE). You should now have a folder named linked_list.

2. Download file test_singly_linked_list.c, singly_linked_list.c and singly_linked_list.h from cuLearn. Move these files into your linked_list folder. **You must also add test_singly_linked_list.c and singly_linked_list.c to your project:** from the menu bar, select Project > Add files to project... In the dialogue box, select test_singly_linked_list.c, then click Open. An icon labelled test_singly_linked_list.c will appear in the Pelles C project window. Repeat this for singly_linked_list.c. Pelles C will automatically add singly_linked_list.h to the project.

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

4. Execute the project. The test harness will run. Read the console output carefully. Read the main function, and determine what the tests do.

Exercise 1

In singly_linked_list.c, define a function that counts the number of nodes that contain an integer equal to `target`, and returns that number. The function prototype is:

```
int count(IntNode *head, int target);
```

Parameter `head` points to the first node in the linked list.

This function should return 0 if the list is empty (parameter `head` is `NULL`).

Remember to add the function prototype to singly_linked_list.h.

In test_singly_linked_list.c, write some code at the end of `main` that tests `count`. At a minimum, test the following cases:

- the linked list is empty;
- the linked list is not empty, but the target value is not in the linked list;

- there is one occurrence of the target value, in the first node in the linked list;
- there is one occurrence of the target value, in the last node in the linked list;
- there are multiple occurrences of the target value in the linked list.

Your tests should output the following information:

- the name of the function that is being called, and the value of its arguments;
- the expected result;
- the actual result

Exercise 2
In singly_linked_list.c, define a function that returns the index (position) of the first node in a linked list that contains an integer equal to `target`. The function prototype is:

```
int index(IntNode *head, int target);
```

Parameter `head` points to the first node in the linked list. This function uses the numbering convention that the first node is at index 0, the second node is at index 1, and so on.

This function should return -1 if the list is empty (parameter `head` is `NULL`).

If `target` is not in the list, the function should return -1.

Remember to add the function prototype to singly_linked_list.h.

In test_singly_linked_list.c, write some code at the end of `main` that tests `index`. At a minimum, test the following cases:

- the linked list is empty;
- the linked list is not empty, but the target value is not in the linked list;
- the first occurrence of the target value is in the first node in the linked list;
- the only occurrence of the target value is in the last node in the linked list;
- there are multiple occurrences of the target value in the linked list.

Your tests should output the following information:

- the name of the function that is being called, and the value of its arguments;
- the expected result;
- the actual result

Exercise 3
In singly_linked_list.c, define a function that is passed a pointer to the first node in a linked list and an integer index. This function will return the integer stored in the node at the specified index (position). The function prototype is:

```
int fetch(IntNode *head, int index);
```

Parameter `head` points to the first node in the linked list.

This function must handle these three cases:

- If the list is empty, the function should terminate via `assert`.
- If parameter `index` is valid (0 <= `index` < the number of nodes in the linked list), the function should return value in the corresponding node. (The index of the first node is 0, the index of the second node is 1, etc.)
- If parameter `index` is negative or >= the number of nodes in the linked list, the function should terminate via `assert`.

Remember to add the function prototype to singly_linked_list.h.

In test_singly_linked_list.c, write some code at the end of `main` that tests `fetch`.

Wrap-up

1. Remember to have a TA review and grade your solutions to the exercises before you leave the lab.
2. The next thing you'll do is package the project in a ZIP file (compressed folder). From the menu bar, select Project > ZIP Files... A Save As dialog box will appear. Click Save. Pelles C will create a compressed (zipped) folder named linked_list.zip, 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).
3. Log in to cuLearn, click the Submit Lab 10 link and submit linked_list.zip. After you click the Add submission button, drag the file to the File submissions box. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status for your file is "Draft (not submitted)". You can resubmit the file by clicking the Edit my submission button. After you've finished uploading your file, remember to click the Submit assignment button. This will change the submission status to "Submitted for grading".