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

Lab 11 - More Linked List Functions

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.

These exercises present a greater challenge than the functions you developed for Lab 10. The algorithms are more difficult, and there are more test cases. Don't be surprised if you are unable to complete all the exercises during the 2-hour lab period.

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.

For this lab, you'll need your linked_list project from last week's lab. If you don't have a copy, you should be able to download the zip file you submitted to cuLearn and extract the project.

Instructions

1. Launch Pelles C and open the linked_list project you worked on during Lab 10. (If you don't have this project, follow the instructions in Lab 10 to create it. You don't need to do the

exercises from Lab 10 before starting this lab; none of the functions you'll write this week require the functions from that lab.)

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

Exercise 1 (Difficulty Level: Moderate)

In singly_linked_list.c, define a function that is passed a pointer to the first node in a linked list, a (non-negative) integer index, and an integer `x`. This function will insert a new node containing `x` at the specified index (position). The function prototype is:

```
IntNode *insert(IntNode *head, int index, int x);
```

Parameter `head` points to the first node in the linked list, or is `NULL` if the linked list is empty. The function **does not** terminate the program (via `assert`) if the linked list is empty.

This function uses the numbering convention that the first node is at index 0, the second node is at index 1, and so on. Parameter `index` must be in the range 0..*length* (where *length* is the number of nodes in the linked list). If `index` is invalid, the function should terminate via `assert`.

Note that this function does not replace the contents of the node (if any) that is currently at position `index`. Instead, that node will be at position `index + 1` after the new node is inserted ahead of it.

The function returns a pointer to the first node in the modified linked list.

There are several cases you should consider when designing this function.:

- The linked list is empty. There are two subcases:
    - `index` is 0. The function will return a pointer to a list containing one node.
    - `index` is invalid.
- The function is passed a pointer to a linked list containing one or more nodes. There are four subcases:
    - `index` is 0. The function will insert the node at the front of the linked list.
    - `index` is greater than 0 and less than the number of nodes in the linked list. The function will insert a new node at the specified position.
    - `index` equals the number of nodes in the linked list. The function will append a new node after the last node.
    - `index` is invalid.

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 `insert`. 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 (Difficulty Level: Challenging)

In singly_linked_list.c, define a function that is passed a pointer to the first node in a linked list and a (non-negative) integer index. This function will remove the node at the specified index (position). The function prototype is:

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

Parameter `head` points to the first node in the linked list, or is `NULL` if the linked list is empty. The function should terminate (via `assert`) if the linked list is empty.

This function uses the numbering convention that the first node is at index 0, the second node is at index 1, and so on. Parameter `index` must be in the range 0..*length*-1 (where *length* is the number of nodes in the linked list). If `index` is invalid, the function should terminate via `assert`.

The function returns a pointer to the first node in the modified linked list (this pointer will be `NULL` if the function deletes the only node in a linked list containing one node, resulting in an empty linked list).

There are several cases you should consider when designing this function.:

- The linked list is empty. For this case the function will terminate via `assert`.
- The function is passed a pointer to a linked list containing exactly one node. There are two subcases:
    - `index` is 0 (the index of the first (and only) node). The function will return `NULL`, indicating that the linked list is now empty.
    - `index` is invalid.
- The function is passed a pointer to a linked list containing two or more nodes. There are three subcases:
    - `index` is 0 (the index of the first (and only) node).
    - `index` is greater than 0 and less than the number of nodes in the linked list.

- `index` is invalid.

Remember to return the deleted node to the heap.

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 `delete`. 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 (Difficulty Level: Challenging)

In singly_linked_list.c, define a function that is passed a pointer to the first node in a linked list. The function removes the first node in a singly-linked list that contains an integer equal to `target`. The function prototype is:

```
IntNode *delete_target(IntNode *head, int target,
                       _Bool *removed);
```

Parameter `head` points to the first node in the linked list, or is `NULL` if the linked list is empty. The function **does not** terminate the program (via `assert`) if the linked list is empty.

If `target` is found, the variable pointed to by parameter `removed` should be set to `true`; otherwise it should be set to `false`.

The function returns a pointer to the first node in the list (this pointer will be `NULL` if the function deletes the only node in a linked list containing one node, resulting in an empty linked list).

There are several cases you should consider when designing this function.:

- The linked list is empty. For this case the function will return `NULL` (the target value cannot be in an empty linked list).
- The function is passed a pointer to a linked list containing exactly one node. There are two subcases:
    - `target` is in the first (and only) node. The function will return `NULL`, indicating that the linked list is now empty.
    - `target` is not in the first (and only) node.
- The function is passed a pointer to a linked list containing two or more nodes. There are three

subcases:

- `target` is in the first node.
- `target` is in the linked list, in any node other than the first node.
- `target` is not in the linked list.

Remember to return the deleted node to the heap.

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 `delete_target`. 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

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 11 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".