CS 305 Lab 6: Linked Lists
Fall 2019

The purpose of this lab is to give you experience with linked lists and writing recursive functions on linked lists. Recall that recursive functions are written such that recursive calls of the function lead toward base case(s). For each of the recursive functions in this lab, the base case of lists being NULL is done for you. You are to complete the recursive cases in the functions.

This lab has a total of 100 possible points (30 points from pre-lab and 70 points from lab).

Sit with your assigned partner.

Objectives

Upon completion of the laboratory exercise, you will be able to do the following:

- Identify the base case(s) and recursive case(s) for the recursive functions.
- Traverse linked lists.
- Create linked lists.
- Merge two linked lists.

Part 1: Logging in (choose one person)

1. Follow the procedure from prior labs to log in and open Mobaxterm:

2. Go into your cs305 directory. Make a new lab6 directory:

```
cd /drives/p/
cd cs305
mkdir lab6
```

3. Get the lab6 file. Download `link_list_lab.c` from moodle to your `lab6` folder. Hopefully, you can see the file. Open it in emacs. You can focus on the first 130 (or so) lines of the file. This has the same implementation of linked lists as we saw in lecture, so the code should be familiar. The four new function prototypes are listed above the main function.

Part 2: lengthRec – Fix the recursive case

1. Examine the `lengthRec` function definition. This function is supposed to return the number of nodes in the linked list. An iterative version of length is defined later in the file. This one, however, is supposed to be recursive. Update the "else" case so that it recursively calls `lengthRec` on `list->next` and returns this function call plus 1. (one line of code)

2. In the main function, add to `lst1` so that it has 5 nodes where the values of the nodes are in <u>ascending</u> order.

3. In the main function, add to `lst2` so that it has 8 nodes where the values of the nodes are in ascending order. Have some of these values be the same as values in `lst1` and have some be different.

4. Compile and run your code.

```
gcc –o link_list link_list_lab.c
./link_list
```

If you want to use the debugger, include the –g flag when compiling.

Part 3: freeList – Fix the recursive case (switch driver)

1. Examine the `freeList` function. The function should be recursive. It should call `free` on `list->next` and then call `free` on `list`. (Two lines of code)

2. The function call to freeList on `lst1` and `lst2` are already in `main`. Compile and run your code.

3. Why must `list->next` be free’ed before `list`?

Checkpoint 1 [30 points]: Show your lab instructor/assistant your program running and show your function definitions. Show your answer to the question above.

Part 4: copyList – Fix the recursive case (switch driver)

1. Examine the `copyList` function. Update the code so that the function returns a call to `makeNode` with the first parameter as `list->num` and second parameter is a recursive call to `copyList` on the rest of the list.

2. In main, create a new list called `lst3` and copy `lst1` to `lst3`.

```
Node * lst3 = copyList(lst1);
print(lst3);
```

3. At the bottom of main, free `lst3`.

4. Do any other modification and testing to be sure `copyList` is working correctly.

5. What is the running time for `copyList` (suppose the linked list has N items)?
O(______)

Part 5: mergeList – Fix the recursive cases (switch driver)

1. Examine the `mergeList` function. The base cases are done for you. You should complete the recursive cases so that a new node is made with the appropriate value and its `next` field is the recursive call on the two appropriate lists. (each recursive case should be one line of code)

For example, if `lst1` has `2 4 5 8 10 12`
and `lst2` has `1 2 5 8`

then, the function should return `makeNode` value 1 (since `lst2->num` is greater than `lst1->num`) and recursively merge `lst1` and `lst2->next`.

Each case should be one line of code in `mergeList`.

2. Why do the recursive calls for `mergeList` eventually reach a base case?

____

3. Suppose list1 has 200 items and list2 has 200 items. How many calls does `mergeList` make to merge the two lists (in worst case)? ____________________

Checkpoint 2 [40 points]: Show your lab instructor/assistant your program running and your code. Show your answer to the question above.

If you finish early and want more practice, try writing the code for one or more of the following functions:

- `insertInOrder` – takes an `int n` and a `pointer to a list`; inserts a new node with num n into the list, keeping the list sorted in ascending order
- `reverse` – takes a `list` and returns a new list with the nodes from `list` in reverse order
- `numInRange` – takes a lower bound min and upper bound max and a list and returns the number of nodes with values >= min and <= max

Close Mobaxterm and any other applications. Log off the computer.

If any checkpoints are not finished, they are due in one week. You may submit screenshots, code files, and answers to questions electronically (on moodle).