COSC 220 - Computer Science II
Lab 7

Dr. Joe Anderson

Due: 29 October 2019

1 Objectives

1. Practice linked lists, dynamic memory
2. Practice recursive algorithms

2 Preliminaries

In this lab, you will work with "bare" lists; that is, linked lists that are not contained in class functionality. Instead, we will use a **recursive** view of a linked list as: *any "node" that is either 1) null, or 2) pointing to a linked list.* You will implement several linked list methods, many of which will follow the general structure

```
Node* doSomethingRecursive(Node *head .... ) {
   // base case
   if (head == null) {
       do the ``obvious'' case
   }
   else {
      // recursive case
      do something at the head of the list, then
      call the method on the rest of the list:

      doSomethingRecursive(head->next .....)
   }
}
```

Keep in mind that the `head` variable in the above scenario isn't the real head, but only the head of the linked list in the current sub-problem.

As an example, if we wanted to implement a `search` function that returns a pointer to the first node that contains a given value:

```
Node* search(Node* head, int target){
  if( head == nullptr || head->value == target ){
    return head;
  } else {
    return search(head->next, target);
  }
}
```

Then one could call `Node* found = search(head, 10)` to get a reference to the node in the list.

3 Tasks

1. Begin this lab by defining a basic node structured data type:

```
struct Node {
  int value;
  Node* next;

  // Constructor, only in C++!
  Node(){
    value = 0;
    next = nullptr;
  }

  // Non-default ctor, okay in C++
  Node(int n){
    value = n;
    next = nullptr;
  }
};
```

2. You will add several functions to perform basic list operations, recursively! Do not use any loop constructs in your implementations. While considering solving the problems, it will often be helpful to try small examples on paper, tracing through the code thoroughly.

   (a) Write a function called `length` that computes and returns the length of a list. Example call: `int len = length(head)`.

   (b) Write a function called `print` to print the contents of the list, from the head to the tail.

   (c) Write a function called `reversePrint` to print the contents of the list in reverse.

   (d) Write a function to insert a value into a sorted linked list. Example call: `head = insertSorted(head, 10);`.

   (e) Write a function called `remove` to remove the first node with a passed value from a list. Do not assume the list is sorted or obeys any extra structure. The function should "unlink" the node from the list and return a pointer to it, so it can be freed in the calling function, if desired. Example call: `removedNode = remove(head, 3);`.

   (f) Write a function called `appendList` to append one list onto the end of another. Example call: `head1 = appendList(head1, head2);`.

   (g) Write a function called `reverse` to reverse all the pointers of the list, making the tail node the new head. You may follow this approach:

      i. The base case is when the `next` pointer of `head` is `nullptr`, in which case no operation is needed. Simply return the address of `head`.

      ii. The recursive case is that, the reverse of the list is given by the `head` of the current list appended to the end of reversing all the nodes past the head. This means that you should recursively call `reverse` first, storing the pointer returned, then flip the pointer of the current head node, and return the *same value* that the recursive call returned.

      iii. The return value should be a pointer to the *head* element of the reversed array segment. That is, the least recently reversed node.

3. Write a `main` function to thoroughly test and demonstrate the correctness of each of the above methods (so, having a print function finished early will be helpful).

4. Include a `Makefile` to compile your program
5. Run your program under `valgrind` to verify that there are no memory leaks. Include the output in your submission.

4 Submission

Upload your project files to the course canvas system in a single zipped folder: To zip on Linux:

1. To zip a single folder into a single archive called "myZip.zip":

   ```
   zip -r myZip.zip folderName
   ```

2. To zip multiple files into a zip file called "myZip.zip":

   ```
   zip myZip.zip file1.cpp file2.h file3 file4.cpp
   ```

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.