Assignment 3

COMP 2401

Date: October 28, 2015

Due: on November 10 2015 before 23:00 (11:00 PM)

Submission: Electronic submission on cuLearn.

Objectives:

a. Memory allocation.

b. Linked Lists

Submission

- Submission must be in cuLearn by the due date.
- Submit a Makefile
- Submit your test program
- Submit a single tar file with all the .c files, .h files and Makefile.
- Submit a Readme.txt file explaining
    - Purpose of software
    - Who the developer is and the development date
    - How the software is organized (partitioned into files)
    - Instruction on how to compile the program (give an example)
    - Any issues/limitations problem that the user must be aware of
    - Instructions explaining how to use the software

Grading:

You will be graded with respect to what you have submitted. Namely, you earn points for working code and functionality.

The grading will look at how the functions are coded, whether functions meet the required functionality, memory allocation and de-allocation, setting pointers to NULL as needed, etc.

Background

You are tasked to manipulate person records using a linked list

The following information defines the fields:

- First Name

- Family Name
- Id

Coding Instructions:

1. Comments in Code – as provided in the slides given in class
2. No usage of global variables. All data must be passed or received via function parameters.
3. Each node in the linked list must be allocated independently of other nodes.
4. Test your program with valgrind to ensure that no memory leakage occurs.
5. This assignment will be tested using a test code. Namely, your file linked_list.c and linked_list.h will be linked together with the test program. Therefore, you cannot deviate from the function declaration. You are allowed however, to add additional functions to the file linked_list.c and linked_list.h.

   Your code will be tested automatically. The test will be done as follows:

   a. A test program (main.c) will use the code that you generated by invoking your functions to manipulate a linked list.

   b. The program output will be directed to a file and compared against a precomputed output.

   c. The test program output should match your code.

      Thus it is important that you use the provided printNode function.

      A sample test code is provided in file main.c. Two object files are also provided:

      - DN_linked_list.o which contains most of the functions that you need to code. This will allow you to compare your code output with the expected output. For example you can link the provided main.c with DN_linked_list.o to see the output.
      - DN_merge_sort.o – this file contains a merge sort routine which will be needed in order to test the removeDuplicates function. This assumes that you did not code the bonus question.

1) Suggestion –

   a) Test functions - all functions with the exception of the functions in Part II will be quite short 2-15 lines of code. As you code design and implement test functions to ensure that your code covers all cases (e.g., head == NULL).

   b) Create function for each menu option. The function should accept as input the array of person and the number of elements in the array.

Part I Tasks (60 points)

In this part of the assignment you will code several basic functions used to manipulate a linked list. The files linked_list.c and linked_list.h contain the functions and their declarations. You will have to complete the body of the function. A more detailed information about the input and output parameters and return information is given in the linked_list.c file

Code the following functions

1. **(5 points)** Insert into list as the first node (10-30 minutes)
   To do - create a new node and add it to the list as the first element
   Function declaration:
   PersonalInfo *insertToList(PersonalInfo **head, unsigned int id, char *firstName, char *familyName)

2. **(5 points)** Insert to list after a given node (10-15 minutes)
   To do - create a new node and add it to the list after the given record
   Function declaration:
   PersonalInfo *insertAfter(PersonalInfo *node, unsigned int id, char *firstName, char *familyName)

3. **(7 points)** Insert a node at the end of the list (10 minutes)
   To do - create a new node and add it at the end of the list
   Function declaration:
   PersonalInfo *insertLast(PersonalInfo **head, unsigned int id, char *firstName, char *familyName)

4. **(5 points)** Search by Name(10-15 minutes)
   To do - search for the first node with the matching firstName
   Function declaration:
   PersonalInfo *searchNodeByName(PersonalInfo *head, char *firstName)

5. **(5 points)** Search by Id (10-15 minutes)
   To do - search for the first node with the matching firstName
   Function declaration:
   PersonalInfo *searchNodeById(PersonalInfo *head, id)

6. **(10 points)** Delete (15-30 minutes)
   To do - delete the first node with the matching id

   Function declaration:
   int deleteNodeById(PersonalInfo **head, int id, char *firstName, , char *familyName)

7. **(5 points)** Print the list (15 minutes)
   To do - prints all the records in the list. **Note you must use the provided printNode function**

   Function declaration:
   void printList(PersonalInfo *head)

8. **(5 points)** Count the number of elements in the list (5-10 minutes)
   To do - counts the number of nodes in the list
   Function declaration:

   int listSize(PersonalInfo *head)

9. **(5 points)** Count specific records (10-15 minutes)
   To do - counts the number of nodes in the list with a matching firstName

Function declaration:

```
int countElements(PersonalInfo *head, char *firstName)
```

10. **(7 points)** Delete the list and all the nodes (15-30 minutes)
    To do - deletes all nodes from the list

    Function declaration:
    ```
    void deleteList(PersonalInfo **head)
    ```

Part II Tasks (40 points)

11. **(25 points)** Merge two lists into a single list (60-180 minutes)
    To do - merges two lists into a single list in a sorted order. The two input list are given in a sorted order. The lists are to be sorted with respect to the first name.

    Function declaration:
    ```
    void mergeList(PersonalInfo **headList1, PersonalInfo **headList2, PersonalInfo **outHead)
    ```

12. **(15 points)** Remove Duplicate Records Delete (45-90 minutes)
    To do - removes all duplicates records from the list. Duplicate records are determined by their first name. Note, that the linked list is in sorted order by the first name. This implies that if there are any duplicated records then they would appear in the linked list consecutively (one after the other). Note, I provide a sorting function (mergeSortDN()) that can be used to sort the list by first name. The object file is DN_merge_sort.o and you should linked it to your program.

    Function declaration:
    ```
    void removeDuplicates(PersonalInfo *head)
    ```

Part III Bonus (30 points)

13. **(30 points)** Merge two lists into a single list (45-90 minutes)
    Merge sort – here you will be given a linked list and you will have to sort it. Merge sort is quite simple once you have correctly implemented the merge step (see Part II).

    The merge sort routine is carried out by using recursion.

    Here is the pseudo code
    ```
    Void mergeSort(PersonalInfo **mainList)

    // check boundary condition
    If (list size <= 1) return the same list

    Else {
       Divide the list into two lists, l1, and l2 of roughly the same size

       Sort each of the lists by calling mergeSort on each of the two lists

       merge the two lists, l1 and l2, into a single list, the mainList
       use the merge lists function that you coded in Part II
     }
    ```