Assignment 3

COMP 2401

Date: March 2, 2017

Due: on March 21 2017 before 22:00 (10:00 PM)

Submission: Electronic submission on cuLearn.

Objectives:

a. Memory allocation.

b. Linked Lists

Assignment Total points: 100

Submission (10 points)

- Submission must be in cuLearn by the due date.
- Submit a single tar file with all the c and h files.
- 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, no memory leaks, etc.

Background

Your team was tasked to manipulate records of all the employees at Carleton. The design team decided to use a linked list in order to accomplish the task. Your role is to code the required set of linked list functions. One of your teammates was tasked to code a test program for testing the code set of functions. Thus, you cannot deviate from the function definitions.

The following information is kept for each employee:

- First Name
- Family Name
- Id

The design calls for the following structure

```
#define NAME_LENGTH 32

typedef struct personalInfo {
      struct personalInfo *next;
      unsigned int id;
      char firstName[NAME_LENGTH];
      char familyName[NAME_LENGTH];
} PersonalInfo;
```

Provided Files (in a3.tar file)

linked_list.h – this file contains the definition of the linked link functions

linked_list.ch – this file contains the code of the linked list functions

main.c – an example code for testing the linked list functions. Note that this is not a comprehensive set. A more comprehensive test program may be provided later.

Bubblesort.o – a sorting file that can be used to test the removeDuplicates() function.

Bublesort.h – a header file for bubblesort function

Makefile – a makefile for generating the code

Compiling the test program:

In order to try the test program you will need to compile and link the test program with your code and the bubblesort function. For example use

```
gcc -g main.c linked_list.c bubble_sort.o
```

Coding Instructions:

1. Add comments to 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 most likely be tested automatically using test program(s). Namely, your

file linked_list.c and linked_list.h will be linked together with test programs. Therefore, you cannot deviate from the function definitions. You are allowed however, to add additional functions (helper functions) to the file linked_list.c. These functions will not be tested.

6. Make sure that all pointers are initialized to NULL or set to NULL as needed.

Suggestions –

Test functions - all functions with the exception of the functions in Part II will be quite short 2-10 lines of code. As you code design and implement test functions to ensure that your code covers all cases (e.g., head == NULL).

Part I Tasks (90 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. The description of each function in the linked_link.c file provides detailed information about the functionality of each function including input and output parameters and return value.

Code the following functions

1. Insertion Functions (18 points)

1.1. (6 points) Insert to list as the first node
Purpose - 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)
```

1.2. (6 points) Insert to list after a given record
Purpose - 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)
```

1.3. (6 points) Insert a node at the end of the list
Purpose - 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)
```

2. Search Functions (10 points)

2.1. (5 points) Search by name
Purpose - search for the first node with the matching firstName
Function declaration:
```
PersonalInfo *searchByName(PersonalInfo *head, char *firstName)
```

2.2. (7 points) Search by id (5 points)
Purpose - search for the first node with the matching id
Function declaration:

PersonalInfo *searchById(PersonalInfo *head, unsigned int id)

3. (35 points) Delete Functions

3.1. (6 points) Delete
Purpose - delete the first node/record from the list

Function declaration:
int deleteFromList(PersonalInfo **head, unsigned int *id, char *firstName, char *FamilyName);

3.2. (10 points) Search by name and delete the node
Purpose - delete the first node with the matching firstName

Function declaration:
int deleteNodeByName(PersonalInfo **head, char *firstName, char *FamilyName, unsigned int *id)

3.3. (7 points) Delete the last node in the list
Purpose - delete the last node in the list

Function declaration:
int deleteLast(PersonalInfo **head, unsigned int *id, char *firstName, char *FamilyName)

3.4. (6 points) Delete After
Purpose - delete the record after the given record

Function declaration:
int deleteAfter(PersonalInfo *node, unsigned int *id, char *firstName, char *FamilyName)

3.5. (6 points) Delete the list and all the nodes
Purpose - deletes all nodes from the list

Function declaration:
void deleteList(PersonalInfo **head)

4. (5 points) Printing Functions

4.1. (0 points) Print a node

Purpose – This function is already coded. Do not change it.

Function declaration:
void printNode(PersonalInfo *node)

4.2. (5 points) Print the list
Purpose - prints all the records in the list

Function declaration:
void printList(PersonalInfo *head)

5. (10 points) Counting Functions

5.1. (5 points) Count the number of records in the list

Purpose - counts the number of records in the list
Function declaration:

```
int listSize(PersonalInfo *head)
```

5.2. (5 points) Count specific records
Purpose - counts the number of nodes in the list where the first name matches the provided key firstName

Function declaration:

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

6. (12 points) Miscellaneous Functions

6.1. (12 points) Remove Duplicate Records Delete
Purpose - removes all duplicates records from the list.

Duplicate records are determined by their first name. Note that the linked list is given in sorted order by the first name. This implies that if two employees have the same first name then the corresponding records would appear one after the other in the linked list.

Function declaration:

```
void removeDuplicates(PersonalInfo *head)
```