Assignment 4

COMP 2401

Date: October28, 2019

Due: on November 11, 2019 before 18:00

Submission: Electronic submission on cuLearn.

Objectives:

a. Memory management

b. Linked Lists

c. Unions

d. Function pointers

Assignment Total points: 100

Submission (10 points)

- Submission must be in cuLearn by the due date.
- Submit a single tar file, A4.tar, with all the c and h files.
- Submit a Readme.txt file explaining (can be included in the tar file)
    - 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
- All pointers are initialized to NULL
- All pointers when not used are set to NULL

Memory Leaks (10 points)

Memory leaks will be checked using valgrind.

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 employees and students at Carleton. The design team decided to use a linked list to accomplish the task. Your role is to code the required set of linked list functions. Your teammates will rely on your code to complete their task. Thus, you cannot deviate from the function definitions.

The design calls for a single struct (PersonInfo) which is used to hold an manipulate students and employees. The struct declarations are provided in the file person.h.

Provided Files

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

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

main.c – an example code which tests the linked list functions. Note that this is not a comprehensive set and it demonstrates how someone may use your code. As needed add test code to ensure that all coded functions work properly.

person.h – a file containing the structure of a person. The structure of a person has some fields that are common to students and employees and some specific fields for employee and some for a student. The specific fields are put together using a union. The file also contains function prototypes for printing an employee and a student.

person.c – a file containing the code of the person files. It contains that implementation of the two printing functions (printEmployee() and printStudent().

Compiling the test program:

In order to test your code you will need to compile and link your code with the test program (main.c). For example to create an a.out file use

gcc -g main.c linked_list.c person.c

Coding Instructions:

1. Add comments to code
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 may 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.
6. You are allowed to add additional functions (helper functions) to the file linked_list.c. These functions will not be tested.
7. Make sure that all pointers are initialized to NULL or set to NULL as needed.
8. All functions that insert a node into the list must copy the input data onto the newly created node.
9. All functions that search for a node or delete a node must copy the data from the node to the

output parameter if the output parameter is not NULL

10. Robust code – make sure that your code is robust. Examples are testing that memory was allocated; ensuring that the program does not crash if pointers that are past to a function have the value of NULL. Where applicable read assumptions that are associated with function.

Suggestions –

1. 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).
2. Try to reuse code that you already coded in other functions

Part I Tasks – Linked List functions (80 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 definitions. 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 (10 points)

1.1. (5 points) Insert to list as the first node

Purpose - create a new node and add it to the list as the first element

Function definition:

```
ListNode *insertToList(ListNode **head, PersonInfo *person);
```

1.2. (5 points) Insert to list after a given record

Purpose - create a new node and add it to the list after the first node that contain a person record that matches the given familyName

Function definition:

```
ListNode *insertAfter(ListNode *node, char *familyName, PersonInfo *person);
```

2. (15 points) Delete Functions

2.1. (5 points) Delete

Purpose - delete the first node record from the list

Function definition:

```
int deleteFromList(ListNode **head, PersonInfo *person);
```

2.2. (5 points) Search by name and delete the node

Purpose - delete the first node with the matching firstName

Function definition:
```
int deleteNodeByName(ListNode **head, char *familyName, PersonInfo *person);
```

2.3. (5 points) Delete the list and all the nodes
Purpose - deletes all nodes from the list

Function definition:
```
void deleteList(ListNode **head);
```

3. Search Functions (5 points)

3.1. (5 points) Search by name
Purpose - search for the first node with the matching familyName

Function definition:
```
ListNode *searchByName(ListNode *head, char *familyName, PersonInfo *person);
```

4. (15 points) Printing Functions

Where applicable use the provided functions printStudent() and printEmployee(). For example when printing the list using printList() function.

4.1. (0 points) Print a student

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

Function definition:
```
void printEmployee(PersonInfo *person)
```

4.2. (0 points) Print an employee

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

Function definition:
```
void printStudent(PersonInfo *person)
```

4.3. (5 points) Print the list
Purpose - prints all the records in the list

Function definition:
```
void printList(ListNode *head);
```

4.4. (5 points) Print the students in the list
Purpose - prints all the students in the list

Function definition:
```
void printStudents(ListNode *head);
```

4.5. (5 points) Print the students in the list
Purpose - prints all the employees in the list

Function definition:
```
void printEmployees(ListNode *head);
```

5. (5 points) Counting Functions

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

Purpose - counts the number of nodes in the list

Function definition:
```
unsigned int listSize(ListNode *head)
```

6. (30 points) Miscellaneous Functions

6.1. (10 points) Copy a list

Purpose – duplicate the input list

Create a duplicate list iteratively

Function definition:
```
int copyList(ListNode *head, ListNode **newListHead);
```

6.2. (10 points) Copy a list recursively

Purpose – duplicate the input list

Create a duplicate list using the recursively

Function definition:
```
int copyListRecursive(ListNode *head, ListNode **newListHead);
```

6.3. (10 points) Remove students

Purpose – removes all the nodes that contain a student from the list.

Function definition:
```
int removeStudents(ListNode **head);
```

Part II Tasks - Function Pointers (15 points)

In this part of the assignment you will demonstrate to your design the benefit of using functions pointers. You will do so by adding code that is based on function pointers.

7. (15 points) Traverse Function Iterative

Here you will create a general list traversal function that will accept two parameters: the head of list and a function pointer. The function will then traverse the linked list and for each visited node will invoke the function pointer that was given as input. The function pointer will accept one parameter which is a pointer to the struct PersonInfo.

7.1. (10 points) General traverse function

Purpose – create a general-purpose traverse function that uses function pointers.

Function definition:
void generalTravers(???, ???)
Hers you will have to determine the function definition.

For example, if the function is called with the function that you will code in 7.2 below, then the function will traverse all the nodes in the list and only print the students in the list.

7.2. (5 points) a function to print a student

Create a function that will accept a pointer to the structure pointer and would print the student information. This function will be passed as a function pointer to the generalTraverse() function that you created in 7.1

Function definition:
void printStudentPtr(???)
Hers you will have to determine the function definition.