Assignment 1

Georgiy Krylov

September 11, 2024

ASSIGNMENT IS TO BE COMPLETED INDIVIDUALLY BY ALL STUDENTS!

1 Description

This assignment is to start the class working on the data structures that later can be used throughout the course, as well as to freshen up C programming skills and practice memory allocation using malloc(...) and deallocation, using free(...), as well as memcpy(...) and sizeof(...) routines. Having a good memory management skills is important for the practical part of this course.

The assignment is due by 11:59 p.m. on Friday, 20th of September 2024.

1.1 Task

Your task is as follows:

Implement ONE data structure of your choice from slide 1.58 (linked list, doubly linked list, circular linked list) and use it to support the commands described below.

An example of the data structure that can be used for implementing the linked list:

```
typedef struct _node{
    struct _node* next_elem;
    char* data; // For the future assignments you might
       need to use more elements / elements of different
        type.
} Node;
```

For this assignment, the data will consist of short arbitrary-sized sequences of characters, no longer than a hundred elements each. (This clarification can be used to simplify the reading process, not for storing the data in the node. In other words, you are not allowed to use char[100] in the Node declaration, but you are allowed to use such variable as a buffer in the scanf function). Using this data structure (or a slightly altered version for doubly-linked list), please implement the following functionality:

- `void add(Node**, char*)` - adds an element specified by the second parameter `(char*)` to the end of the list pointed to by the first parameter `Node**`. This function should allow for creating duplicates.

- `void delete(Node**, char*)` - removes the first occurrence of the element specified by the second parameter `(char*)` from the data structure. In the case there's no element with such data in your data structure, the data structure should remain unchanged.

- `int countItems(Node*, char*)` - a query that prints the number of exact copies of the string specified by the second parameter `(char*)` stored in your data structure.

- `int removeDuplicates(Node*)` - removes all nodes that contain duplicate words from your data structure.

- `int hasItem(Node*, char*)` - a query that returns 0 or 1, depending on whether the element specified by the second parameter `(char*)`is present in your data structure or not. The value of 0 should be returned if element is not found, otherwise the function should return 1.

- `void findAndReplace(Node*, char*, char*)` - looks up for the first occurrence of an element specified by the second parameter `char*` and replaces it with the value specified by the third parameter `char*` in the list pointed to by the first parameter `Node*`. In the case the character sequence specified by the second parameter is not present in the data structure, no changes should happen.

- `void print(Node*)` - a function that iterates through all elements in the data structure pointed to by the first parameter `Node*`, printing each string to the console.

- `int stop(Node**)` - a function that frees all resources from your data structure pointed to by the first parameter `Node**`, and stops reading the input. You may use the return value to terminate the program.

For this assignment, your program should create one empty data structure at the very beginning and later use it to perform all operations on it. In other words:

```
Node* head = NULL;
```

After that, your program should read the actions that it should take from console. The actions are encoded by the first letters of the corresponding function name e.g., **a** for `add`, **f** for `findAndReplace`, etc.

Your program should properly free all the memory allocated for the data structure elements before termination. I.e., no **memory leaks** are allowed. To refresh your knowledge, a memory leak happens when you have allocated a memory to a pointer, then changed the address the pointer is pointing to, or allocated new memory chunk using `malloc(...)`, but did not release the resources used by previous allocation, i.e., did not `free(...)` the memory.

The tests are guaranteed to run to completion in a working program.

2 Input/Output format

If your program is presented with the following input:

```
a hello
a hi
a hi
p
s
```

The expected output is:

```
hello
hi
hi
```

Second example:

```
a hello
a hi
a hey
p
f hey yes
p
d hi
p
f hi maybe
p
s
```

Should produce the following output

```
hello
hi
hey
hello
hi
yes
hello
yes
hello
yes
```

3 Submission instructions

Please submit your C file(s) (and .h file(s) if any) to D2L Assignment box. Make sure your code compiles and runs. Make sure your code follows the specified input/output format. You must use C programming language to solve this assignment. Important to note in this course: if your program produces correct output, it doesn't necessarily mean you have properly managed the resources and penalties are possible. This assignment focuses on memory management, and therefore the TA will be asked to make sure the memory is properly deallocated. Those who are interested in automated memory verification tool may look up **valgrind** tool, that is part of UNB FCS Linux lab machines installation.

NOTE: THE INPUT AND OUTPUT OF YOUR PROGRAM IS SPECIFIED SUCH THAT THE MARKER CAN AUTO TEST YOUR SUBMISSIONS. PLEASE FOLLOW THE SPECIFICATIONS! INPUT WILL BE READ VIA stdin (E.G., KEYBOARD). OUTPUT SHOULD BE PRINTED TO stdout (E.G., MONITOR).