Doubly Linked Lists

Academic Honesty

The assignment is individual work. Students are allowed to consult books and online resources but must acknowledge the resources used in the report.

We use MOSS to detect software plagiarism.

Sign the Academic Honesty Pledge in the report.

1. Specification

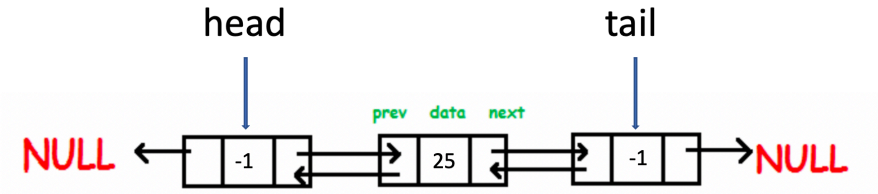
A doubly linked list allows for a variety of quick update operations, including insertions and removals at both ends and in the middle of the list. A node in a doubly linked list stores two pointers: a "next" link, which points to the next node in the list, and a "prev" link, which points to the previous node in the list.

To simplify programming, it is convenient to add special nodes at both ends of a doubly linked list: a "head" node just before the front element of the list, and a "tail" node just after the rear of the list. These "dummy" nodes do not store any useful data. As a result, if a user inserts 6 data elements, the resulting linked list will have 8 nodes in total (6 data elements plus the 2 dummy nodes).

The "head" has a valid "next" reference, but a null "prev" reference. The "tail" has a valid "prev" reference, but a null "next" reference.

In this assignment, the doubly linked list stores non-negative integers. The two dummy nodes store -1 (although their data are not used for anything). Variables "head" and "tail" are global variables, with "head" always pointing to the dummy node at the front of the list, and "tail" always pointing to the dummy node at the rear of the list.

Write a C program to implement the search, insertion and removal operations as described below.



2. Implementation

- The program to be submitted is named `dlist.c`. Use the given template `dlist.c` and fill in your code. **Submit file `dlist.c`.**
- You are also given a file named `dmain.c` to test your code. Do not submit file `dmain.c`.
- Implement the following functions: `insertFirst()`, `insertLast()`, `removeFirst()`, `removeLast()`, and `search()`. See file `dlist.c` for their specifications.
- Function `insertFirst(d)`: The new element is to be inserted at the front of the list, right after the dummy node `head`. If a new node cannot be created (e.g., insufficient memory), the function calls function `prtError()` to display an error message and returns NULL. Otherwise, it returns the pointer to the newly created node.
- Function `insertLast(d)`: The new element is to be inserted at the end of the list, right in front of the dummy node `tail`. If a new node cannot be created (e.g., insufficient memory), the function calls function `prtError()` to display an error message and returns NULL. Otherwise, it returns the pointer to the newly created node.
- Function `removeFirst()`: If the list is empty (i.e., no element other than the dummy nodes), the function calls function `prtError()` to display an error message and returns -1. Otherwise, it removes the first element with useful data (i.e., the node right behind the dummy node `head`) and returns the data (integer) of the removed node.
- Function `removeLast()`: If the list is empty (i.e., no element other than the dummy nodes), the function calls function `prtError()` to display an error message and returns -1. Otherwise, it removes the last element with useful data (i.e., the node right in front of the dummy node `tail`) and returns the data (integer) of the removed node.
- Function `search(k)`: If there is an element containing non-negative integer `k` then return the pointer to that element. Otherwise, return NULL. If there is more than one element containing `k`, return the pointer to the first encountered element.
- Assume that all inputs are non-negative integers and there can be duplicates.
- You may define your own variables inside the above functions.
- In file `dlist.c` you are given these utility functions: `init()`, `getFirst()`, `getLast()`, `prtError()` and `prtList()`. DO NOT modify these functions.
- Do not modify the function or structure definitions in file `dlist.c` or `dlist.h`.
- To compile the C files, use the following commands, then run the executable file `dmain`.

```
gcc -Wall -c dlist.c

gcc -Wall -c dmain.c

gcc dlist.o dmain.o -o dmain
```

- See file `dmain_out.txt` for the output from running programs `dlist.c` and `dmain.c`.
- You should create your own `main()` programs following the template `dmain.c` to test each function separately (unit testing) from the simplest case to more complex cases. Do not add a `main()` function to the submitted file `dlist.c`, or it won't compile with our grading programs.

Important Notes

- Do not use any C library function except malloc(), calloc(), and free(). Do not add any other header file to your C file(s).
- The functions (algorithms) must be the most efficient in terms of running time and memory usage.
- **Submit file `dlist.c`.** Complete the header of the C file with your student and contact information. Include sufficient comments in your code to facilitate code inspection.
- **Submit a report in PDF** with following information: references (sources); error conditions and actions taken; algorithm; running time of the function (algorithm) and an explanation. See the template a6report.docx for an example.
- Your code will be graded automatically by a script, so make sure to strictly follow the specifications.
- Do not use any output statements (for example, printf) in your code, or they will mess up the grading scripts.
- C compilers are machine-dependent, and programs may behave differently on different systems. C assignments will be graded on an EECS server from the command line by a script. To make sure your code passes these tests, it should be tested on an EECS server (for example, red.eecs.yorku.ca) before the final submission.
- We will use more test cases for grading.