Assignment 3 : Doubly Linked List

Deadline: 24/11/2020 11:55 PM

Instruction:

1. Write your code in the c file named “doubly_linkedlist.c”.
2. Write an explanatory comment for every important block of code(a loop or a condition). Your marks will be reduced to half if you don’t write the comments.
3. Avoid plagiarism. If you are found to adopt any unfair means you will get a straight 0.
4. Upload your code (only the c file) in elms.
5. Deadline is 24/11/2020 11:55 PM.
6. You can take reference from my lectures if you find any difficulty.

Task:

In this assignment you have to implement the doubly linkedlist. In the zip file you will find a c file named “doubly_linkedlist.c”. There you will see some function prototypes. Some of the functions are already implemented for your convenience. Your task is to complete the rest of these functions. Description and sample input output are listed below.

1. void insert_at(int item, int pos) function inserts an entry specified by “item” at the position delineated by “pos” by traversing the list starting from “head” of the linkedlist to the entry just before the entry delineated by “pos” and making necessary linking. Be careful about corner or boundary cases. For your convenience the boundary cases are listed below.
   - “pos” can not be negative.
   - “pos” can not be greater than “length” of the linked list.
   - If “pos” is 0, “item” is to be inserted as the first entry. In this specific scenario you can offload the task to void insert_first(int item).
   - If “pos” is equal to “length”, “item” is to be inserted as the last entry. In this specific scenario you can offload the task to void insert_last(int item).

   All the other cases fall under the general case. For example,

Sample input	Sample output	Explanation
Linkedlist: 10 15 25 30 insert_at(20, 2);	Linkedlist: 10 15 20 25 30	20 was inserted at 2nd position, now it is a linkedlist with 5 entries.
Linkedlist: 10 15 25 30 insert_at(20, -1);	Linkedlist: 10 15 25 30	“pos” can not be negative.
Linkedlist: 10 15 25 30 insert_at(20, 5);	Linkedlist: 10 15 25 30	“pos” can not be greater than “length” of the linked list.
Linkedlist: 10 15 25 30 insert_at(20, 0);	Linkedlist: 20 10 15 25 30	20 was inserted at 0th position, now it is a linkedlist with 5 entries.

Linkedlist: 10 15 25 30 insert_at(20, 4);	Linkedlist: 10 15 25 30 20	20 was inserted at the last position, now it is a linkedlist with 5 entries.

2. int search(int key) function searches for the entry "key" in the linkedlist starting from "head" of the linkedlist to the end of the linkedlist in a linear fashion. It should return the relative position (starting from "head" as 0) of the Node containing "key" if it is found, and -1 for an unsuccessful search.

3. void delete_first() function deletes the first entry of the linkedlist by redirecting the "head" pointer to the second entry of the list as the new "head" and making necessary linking. Take precaution if there is no second entry(i.e. the linked list has only one entry in it); although "head" will eventually be redirected to NULL by falling under the general case, you will need to redirect "tail" to NULL because upon deletion the list will become empty. Make sure to free the memory of the Node you just deleted. Be careful about corner or boundary cases. For your convenience the boundary cases are listed below.
   - When the linked list is empty, there is nothing to be deleted.

   All the other cases fall under the general case. For example,

Sample input	**Sample output**	**Explanation**
Linkedlist: 10 15 20 25 30 delete_first();	Linkedlist: 15 20 25 30	10 was deleted, now it is a linkedlist with 4 entries.
Linkedlist: 30 delete_first();	Linkedlist:	30 was deleted, now it is an empty linkedlist. "head" and "tail" are both redirected to NULL.
Linkedlist: delete_first();	Linkedlist:	It is an empty linkedlist. There is nothing to delete.

4. void delete_last() function deletes the last entry of the linkedlist by redirecting the "tail" pointer to the second-last entry of the list as the new "tail" and making necessary linking. Take precaution if there is no second-last entry(i.e. the linked list has only one entry in it); although "tail" will eventually be redirected to NULL by falling under the general case, you will need to redirect "head" to NULL because upon deletion the list will become empty. Make sure to free the memory of the Node you just deleted. As you know doubly linked lists have symmetric property you can achieve this function by re-writing the code of void delete_first() with occasional replacement of "head" with "tail" and vice versa, and "next" with "prev" and vice versa. See the class lecture of 18/11/2020 for explanation. Be careful about corner or boundary cases. For your convenience the boundary cases are listed below.
   - When the linked list is empty, there is nothing to be deleted.

   All the other cases fall under the general case. For example,

Sample input	Sample output	Explanation
Linkedlist: 10 15 20 25 30 delete_last();	Linkedlist: 10 15 20 25	30 was deleted, now it is a linkedlist with 4 entries.
Linkedlist: 10 delete_last();	Linkedlist:	10 was deleted, now it is an empty linkedlist. "head" and "tail" are both redirected to NULL.
Linkedlist: delete_last();	Linkedlist:	It is an empty linkedlist. There is nothing to delete.

5. void delete_at(int pos) function deletes the entry delineated by "pos" by traversing the list starting from "head" of the linkedlist to the entry just before the entry delineated by "pos" and making necessary linking. Make sure to free the memory of the Node you just deleted. Be careful about corner or boundary cases. For your convenience the boundary cases are listed below.
   - "pos" can not be negative.
   - "pos" can not be greater than or equal to "length" of the linked list.
   - If "pos" is 0, the first entry is to be deleted. In this specific scenario you can offload the task to void delete_first().
   - If "pos" is equal to "length"-1, the last entry is to be deleted. In this specific scenario you can offload the task to void delete_last().

   All the other cases fall under the general case. For example,

Sample input	Sample output	Explanation
Linkedlist: 10 15 20 25 30 delete_at(2);	Linkedlist: 10 15 25 30	20 was deleted, now it is a linkedlist with 4 entries.
Linkedlist: 10 15 25 30 delete_at(-1);	Linkedlist: 10 15 25 30	"pos" can not be negative.
Linkedlist: 10 15 25 30 delete_at(4);	Linkedlist: 10 15 25 30	"pos" can not be greater than or equal to "length" of the linked list.
Linkedlist: 10 15 25 30 delete_at(0);	Linkedlist: 15 25 30	First entry was deleted.
Linkedlist: 15 25 30 delete_at(2);	Linkedlist: 15 25	Last entry was deleted.

6. void delete_item(int item) function deletes the first occurrence of the entry specified by "item". You can facilitate this task by making use of the functions you have written previously. Just use int search(int key) to locate the position of the entry specified by "item" and then use void delete_at(int pos) to delete the entry. You don't have to delete anything if there is no entry specified by "item".

For example,

Sample input	Sample output	Explanation
Linkedlist: 10 15 20 25 20 delete_item(25);	Linkedlist: 10 15 20 20	First occurrence of 25 was deleted.
Linkedlist: 10 15 20 20 delete_item(20);	Linkedlist: 10 15 20	First occurrence of 20 was deleted.
Linkedlist: 10 15 20 delete_item(30);	Linkedlist: 10 15 20	30 was not found.

Mark distribution:

Ques	1	2	3	4	5	6	**Total**
Mark	4	2	2	1.5	4	1.5	**15**