NAME: Gabriel de Sa ID: 1001676832

CSE 2320 Homework 3 written part

Task 1 (12 points)

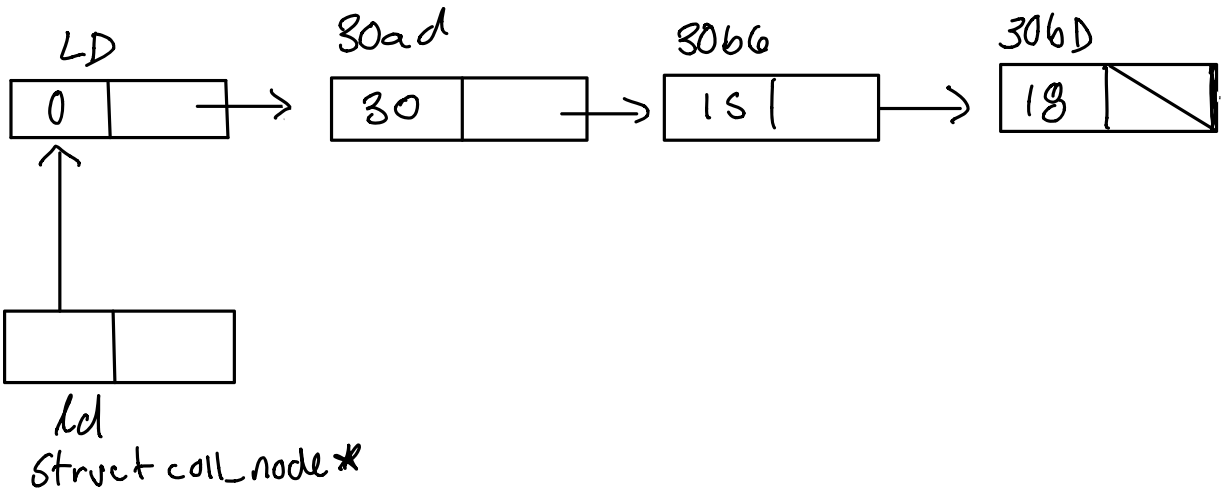
A new node structure (intended to be used to create a list of lists) is defined in the table below (using `struct node`):

```
struct node  {
 int item;
 struct node * next;
};
```

```
// new node structure
struct coll_node  {
  struct node * Ld; // Ld must be represented with a dummy starting node
  struct coll_node * next;
};
```

In your drawings, **show all the data as done in class** (including the list nodes, of type `struct node`). Use boxes for all member variables and write their value inside the box and their name outside the box.

a) (8 points) Draw two nodes (of type `struct coll_node`**)** that point to each other. For one of them Ld should be empty (but not NULL) and for the other one Ld should contain 3 nodes with useful DATA: 30->15->18 (in addition to the dummy node). Use the representation with a DUMMY node for any list, Ld, part of nodes of type `struct coll_node`.



b) (4 points) Assume that an `int` is stored in 4 Bytes and a memory address is 8 Bytes. How much space will the above two nodes (and the data that they reference) occupy? That is, give the total space needed to store in memory what you drew above. **SHOW YOUR WORK**.

$5 \cdot 4\text{bytes} = 20\text{bytes}$

$5_{\text{Pointer}} \cdot 8\text{bytes} = 40\text{bytes}$

$= 60$ bytes of memory.

Task 2 (10 points)

For your answers bellow, assume list `A` has N nodes and list `pos` has M nodes. (Use N and M as needed and do not worry about +/- 1 for the dummy node. Since it is a constant, it can be excluded from the analysis.)

a) (2 pts) swap_first_third(struct node *A): T(N) = $\Theta(1)$

b) (4 pts) delete_occurrences(struct node * A, int V)

assume *only one* occurrence of v in A. Give the worst case time complexity for that:

T(N) = $\Theta(N)$

assume there are *t occurrences* of v in A. Give the worst case time complexity for that:

T(N) = $\Theta(N)$

c) (4 pts) sublist(struct node * A, struct node * pos_list) T(NM) = $\Theta(NM)$

Task 3

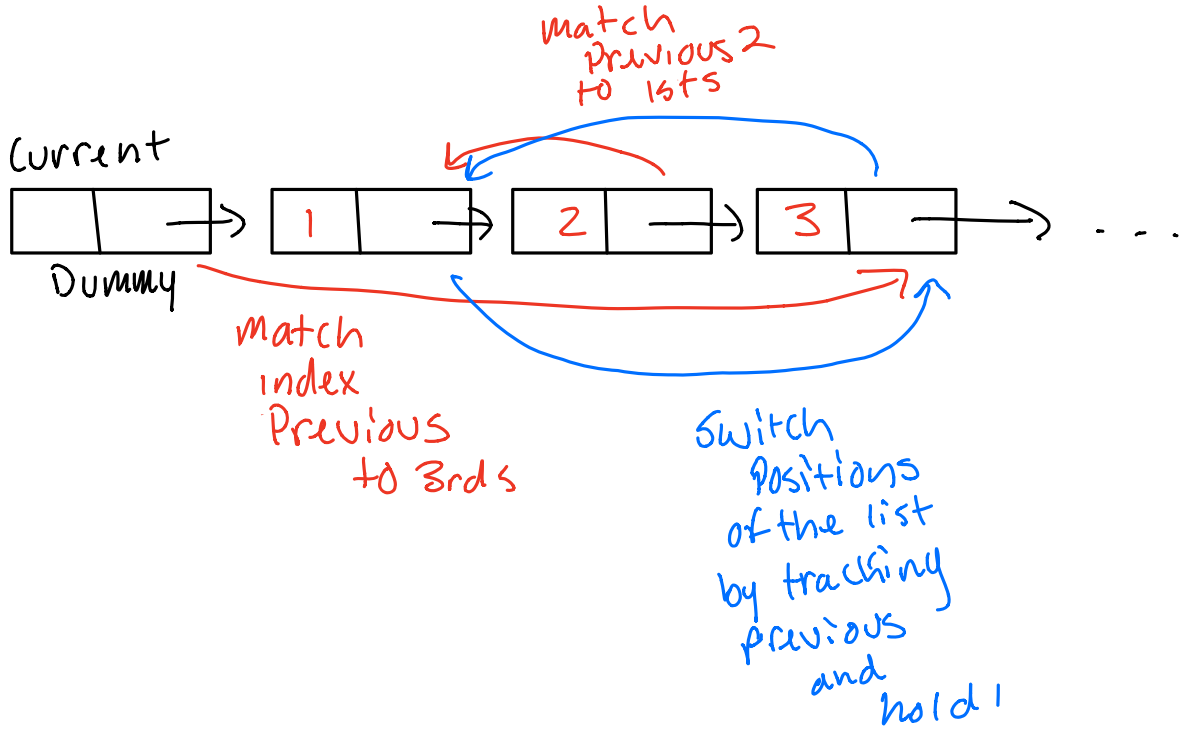
Test cases to be implemented in student_test_sublist(). Add new test cases if needed.

Test case	Data/code And expected result	Does my code handle it? Here: handle= does NOT crash
Index out of bounds	A: 10 ->10 ->40 ->20 pos_list: **(-7)** -> 3 or pos_list: 3 -> **80000** -> 3 result: fct returns NULL	Handle
A is NULL	struct node * A = NULL; result: fct returns NULL	Handle
A is empty	struct node * A = new_list(); result: fct returns NULL	Handle
pos_list is empty	struct node * pos_list = NULL; result: fct returns NULL	Handle
pos_list is NULL	link pos_list = newList(); result: fct returns NULL	Handle
A is not modified by sublist(…) ….	A: 15 -> 100 -> 7 -> 5 -> 100 pos_list: 3 -> 0 ->2 result: A will still be : 15 -> 100 -> 7 -> 5 -> 100	Handle
Test case from hw write-up	A: 15 -> 100 -> 7 -> 5 -> 100 -> 7 -> 30 pos_list: 3 -> 0 -> 6 -> 4	Handle
Repeated position	A: 5 pos_list: 0 -> 0 -> 0 result: returns: 5-> 5-> 5	Handle

For your convenience below are test cases for delete_occurrences(). You do NOT have to write code for them as part of the homework requirements, but your code will be tested against them.

Normal data, V is in A (as in hw write-up)	A: 15 -> 100 -> 7 -> 5 -> 100 -> 7 -> 30 V is 7, Result: A will become: 15-> 100-> 5 -> 100 -> 30
V does not occur in A	A: 15 -> 100 -> 7 -> 5 V is 9, Result: A does not change: 15-> 100-> 7-> 5
Repeated consecutive occurrences	A: 15 -> 7 -> 7 -> 5 V is 7, Result: A becomes: 15 -> 5
A has one item and that is V	A: 7 V is 7 Result: A becomes Empty
A has only items with value V in it	A: 7->7-> 7 V is 7 Result: A becomes empty
A is NULL	A = NULL Result: A is not changed
A is empty	A = new_list() Result: A is not changed

CODE & DRAWING for swap_first_third (struct node * A) (Use additional pages if needed.)



```
void swap_first_third(struct node * A)
{
  //swap the first and third node in a link by adjusting links not copying items.
  if(A == NULL || compute_length(A) == 1)
    return;
  struct node* hold1, *hold2, *prev1, *prev2, *current;
  current = A;
  if(compute_length(current) == 2)
  {
    prev1 = getNodePtr(A, -1);
    prev2 = getNodePtr(A, 0);
    hold1 = getNodePtr(A, 0);
    hold2 = getNodePtr(A, 1);
    if(prev1 != NULL)
      prev1->next = hold2;
    if(prev2 != NULL)
      prev2->next = hold1;

    current = hold1->next;
    hold1->next = hold2->next;
    hold2->next = current;
  }
  else
  {
    prev1 = getNodePtr(A, -1);
    prev2 = getNodePtr(A, 1);
    hold1 = getNodePtr(A, 0);
    hold2 = getNodePtr(A, 2);
    if(prev1 != NULL)
      prev1->next = hold2;
    if(prev2 != NULL)
      prev2->next = hold1;

    current = hold1->next;
    hold1->next = hold2->next;
    hold2->next = current;

  }
  return;
}
```