NAME: Matthew Irvine

ID: 1001401200

CSE 2320 Homework 3 written part

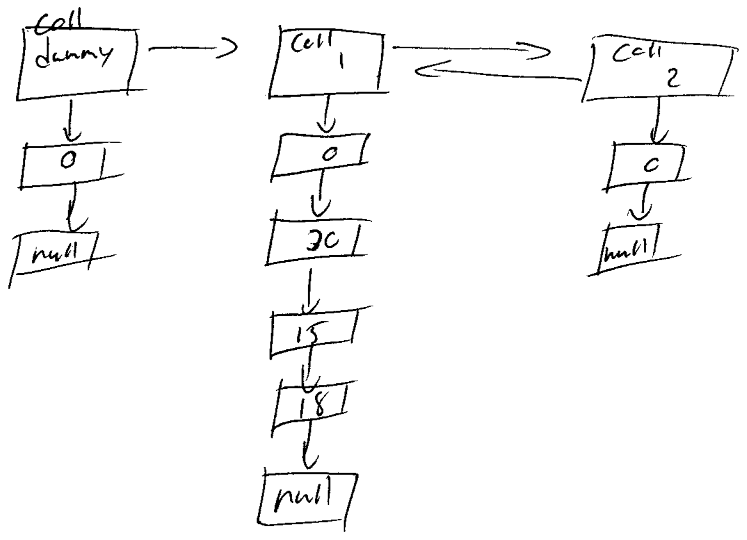
Task 1 (12 points)

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, **<u>show all the data as done in class</u> (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.**

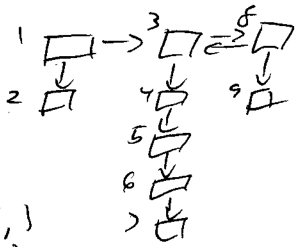
a) (8 points) Draw two nodes (of type `struct coll_node`) that point to each other. For one of them Ld should be empty (but not NULL) and for the other one Ld should contain 3 nodes with useful DATA: 30->15->18 (in addition to the dummy node). Use the representation with a DUMMY node for any list, Ld,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

//assume not including nulls

There are 9 [illegible] boxes
3 boxes are coll_nodes and have 2ptrs each $(3\times2\times8=48)$
6 boxes are normal nodes with 1ptr and 1int $(1\times8\times6=48,$
$1\times4\times6=24)$



Thus 48 bytes
48 bytes
+ 24 bytes

$\boxed{120 \text{ bytes}}$

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)

Didn't precount # of occurences to end early, thus only need to loop once to check every position in the loop. # of occurences doesn't matter since every position is checked.

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(N,M) = $\Theta(N*M)$

independent, multiply M and N

Task 3

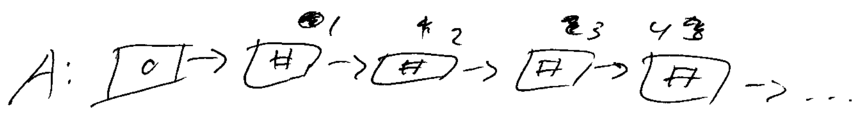
Test cases to be implemented in student_test_sublist(). Add new test cases if needed.

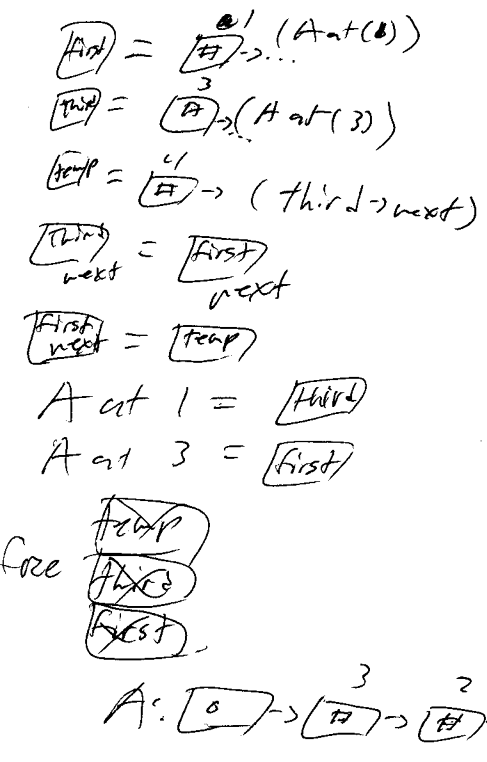
Test case	Data/code And expected result	Does my code handle it? Here: handle= does NOT crash	
Index out of bounds	A: 10 ->10 ->40 ->20 pos_list: (-7) -> 3 or pos_list: 3 -> **80000** -> 3 result: fct returns NULL	does not crash r: null	
A is NULL	struct node * A = NULL; result: fct returns NULL	does not crash	returns null
A is empty	struct node * A = new_list(); result: fct returns NULL	~~done~~ DNC	returns null
pos_list is empty	struct node * pos_list = NULL; result: fct returns NULL	DNC	returns null
pos_list is NULL	link pos_list = newList(); result: fct returns NULL	DNC	r: null
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	DNC A not affected	
Test case from hw write-up	A: 15 -> 100 -> 7 -> 5 -> 100 -> 7 -> 30 pos_list: 3 -> 0 -> 6 -> 4	5, 15, 30, 100 DNC	
Repeated position	A: 5 pos_list: 0 -> 0 -> 0 result: returns: 5-> 5-> 5	DNC	

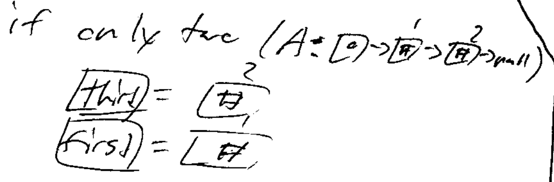
For your convenience below are test cases for delete_occurrences(). You do NOT have to write code for them as part of the homework requirements, but your code will be tested against them.

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.)







```
A->next = third
third->next = first
first->next = null.
```

```
void Swap-first-third( Struct node*A)
{ int size = compute_length(A);
  // check if A is empty or
            Null.
Struct node* third = Null;
Struct node* first = A->next;
                  // skip dummy
if (size==2)
{ third =(A->next)->next;
  A->next = third;
  third->next= first;
} first->next = null;
else {
 Third=((A->next)->next))->next;
 Struct node* temp= third->next;
 Third ->next= first->next;
 first-> next= temp;
  A->next = third;
((A->next)->next)->next = first;
  free (temp)
}
free(third);
free (first);
return;
}
```