CS 152
Exam 2
Winter 2018

1:30pm class

Structs / Unions / Tags	/15
ADT Functionality	/15
Linked List Imp	/30
Tree Implementation	/ 30
Total	/90

1. No one may leave for any reason and come back to work on his/her test. If you need to go to the restroom, go now.

2. You may have 1 sheet of notes, hand-written, double-sided

3. Your backpack must be zipped up.

4. You may not use any electronic devices of any kind. Make sure your cell phone is turned off so it does not ring during the test.

5. You may not wear hats, hoods, sunglasses, or do anything that would obscure your eyes.

6. We will not answer questions during the exam.

7. Do not hold your test up. It must stay on your desk.

8. If you seem to be looking around, you will be moved to the front to allow you to look around without having other students' test in your field of vision.

9. Do not write on the back of your exam.

1. Structs, etc.

A tagged union can be used to implement nodes in a linked list, allowing the tag to distinguish between the head, the tail, and a middle node. *I do not recommend this design – this is just for the purposes of this exam.*

```
enum node_tag { ND_HEAD, ND_MID, ND_TAIL };
typedef struct _lnode lnode;
typedef struct {
      void *item;
      lnode *next;
} head;

typedef struct {
      lnode *next;
      void *item;
} mid;

typedef struct {
void *item;
} tail

union _unode {
      head hd;
      mid md;
      tail tl;
} ;

typedef struct _node{
      enum node_tag tag;
      union _unode und;
} lnode;
```

a. (5) A single struct lnode has space for how many pointers and why?

It has space for 2 pointers since the only pointers come from allocating the _unode which allocates for the largest [illegible] possibility which holds 2 [illegible] *next and *item.

b. (5) Write a line of code that will extract the value from an lnode, *given that you already know it is a tail node*.

lnode *n;

.....

void *v =(n.und.tl)-> item // extract value from n

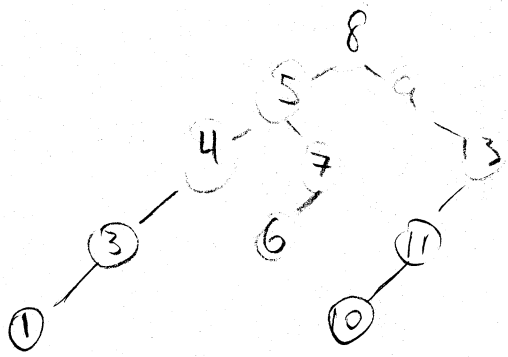
c. (5) Write a function that will return the next pointer from an lnode. If it is a tail node, then return null.

lnode *get)next(lnode *n)

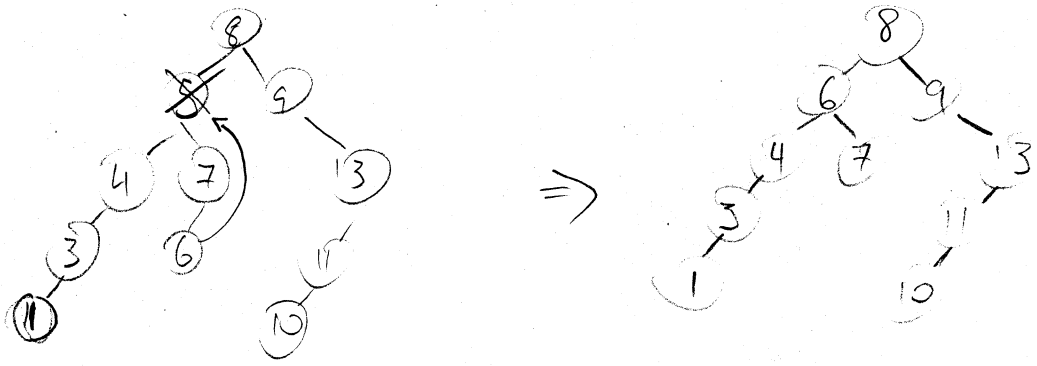
```
{
   if((*n).tag == ND_HEAD){
      return((*n).und.hd -> next)
   }else if ((*n).tag == ND_MID){
      return ((*n).und.md -> next)
   }else {
      return NULL;
   }
}
```

2. ADT Functionality

a. (5) Draw the binary search tree that results from this sequence of inserts:
8, 9, 13, 5, 7, 6, 4, 11, 10, 3, 1

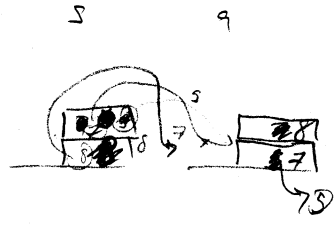


b. (5) Draw the BST that results from removing the number 5 from the tree in (a). Use the algorithm from class.



c. (5) The following operations are performed. What is printed out by each printf?

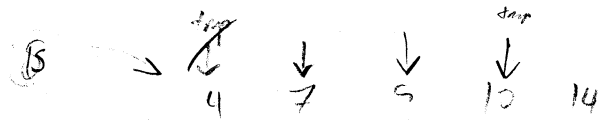
```
stack *s = create_and_init_stack();
queue *q = create_and_init_queue();
push(s, 8);
push(s, 5);
enqueue(q,pop(s));
push(s, 7);
enqueue(q,pop(s));
printed("Dequeue: %d\n",dequeue(q));
enqueue(q,pop(s));
printed("Dequeue: %d\n",dequeue(q));
printed("Dequeue: %d\n",dequeue(q));
```



5

7

8



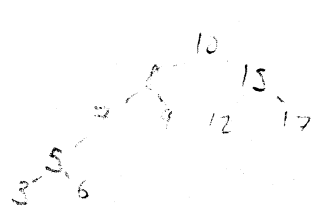
3. Linked List Implementation

30. Write an ***efficient*** function (fewest calls to comp) that removes (and frees the corresponding nodes of) any items greater than the passed-in item in a ***sorted*** linked list. Return the number of items removed. comp returns -1 for less than, 0 for equal, 1 for greater than. Do NOT use the unions from above. Instead, use the following:

```
typedef struct _node node;
struct _node{
    void *item;
    node *next;
} node;
int remove_greater_than(node *list, node *ref, int (*comp)(void*, void*))
{
```

Assuming Sort low → high ?

```
node *tmp, *indexToRemove, *counter, *tmp2;
int count = 0;
tmp = list;
if (comp(ref->item, tmp->item) == -1){
    while (tmp != NULL){
        count++;
        tmp = tmp->next;
    }
    return count;
}
while (comp(ref->item, tmp->item) >= 0){
    if(tmp->next->next != NULL)
        tmp = tmp->next;
}
indexToRemove = tmp->next;
counter = tmp->next;
tmp->next = NULL;
while(counter != NULL){
    count++;
    tmp = counter;
    counter = counter->next;
    free(tmp);
}
}
```



4. Tree Implementation

30. Write an ***efficient*** function (fewest calls to comp) that finds the number of items in a **BST** that are ***greater than*** the second parameter and ***less than*** the third parameter. comp returns -1 for less than, 0 for equal, 1 for greater than. You may continue your code onto the next page – do not write on the back of the exam. Assume the following definitions:

```
typedef struct _bnode bnode;
typedef struct _bnode {
        void  *item;
        bnode *lsub;
        bnode *rsub;
};

typedef struct {
        int (*compare)(void*, void*);
        btreenode *root;
} bst;

int count_in_between(bst *tree,  void *low, void *high)
{       // if no tree or no numbers possible between low & high, return 0
        if ((tree == NULL) || (tree->compare(low, high) >= 0))
                return 0;
        else
                return count_in_between(tree->root, low, high, tree->compare);
}
int count_in_between(bnode *root,  void *low, void *high, int (*comp)(void *, void *))
{
```

```
if ((comp(root->item, low) <= 0) && (comp(root->item, high) <= 0)){
if(root == NULL){   return count_in_between(root->lsub, low, high, comp)
return 0;
} else if ((comp(root->item, low) > 0) && (comp(root->item, high) >= 0)){
} else {
        bnode *tmp = root;
        int lnum = 0;
        while (comp(min, tmp->value) == -1){
                tmp = tmp->lsub
        } bnode tmp2 = tmp;
        while (tmp != NULL) {
                lnum ++;
                tmp = tmp->lsub;
        } tmp = tmp2;
        while (tmp != NULL) {
                lnum ++
                tmp = tmp->rsub;
        }
```

int rnum = 0

// same as for loop to find rnum

return lnum + rnum