# ECE264 Spring 2018
# Exam 3, 630-730PM, April 10, 2018

## Name:

In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action.

**Signature:**

*You must sign here. Otherwise you will receive a* **1-point** *penalty.*

**Read the questions carefully.**
**Some questions have conditions and restrictions.**

This is an *open-book, open-note* exam. You may use any book, notes, or program printouts. No electronic device is allowed. You may **not** borrow books from other students.

This exam tests two learning objectives:
Recursion (Questions 1-4)
Dynamic Structure (Questions 2-4)
You must obtain 50% or more points in each of the corresponding question to pass the learning objective.

**Remove the top sheet.**
**Write your answers on page 2.**
**Return only the top sheet.**

# Contents



| | Q1 | Q2 | Q3 |
|---|---|---|---|
| A | | | |
| B | | | |
| C | | | |
| D | | | |

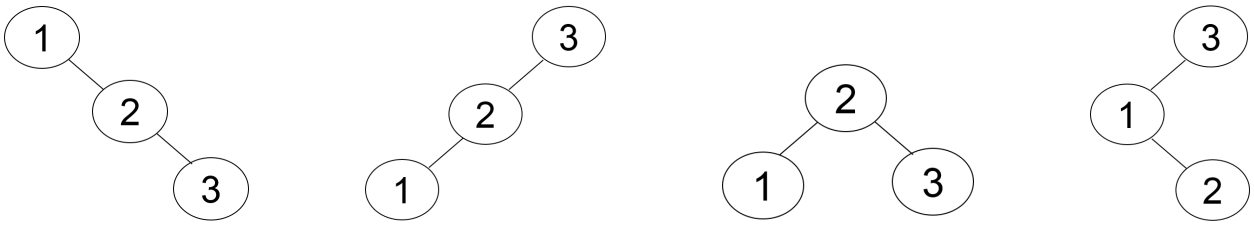
| | Q4 | Q5 |
|---|---|---|
| A | | |
| B | | |
| C | | |
| D | | |

This page is blank.

# 1 Shape of Binary Search Trees (20 points)

The shape of a binary search tree depends on the order of data.

- If 1, 2, 3 are inserted in the order 1, 2, 3, the binary search tree is shown in Figure 1 (a).
- If the order is 3, 2, 1, the binary search tree is shown in Figure 1 (b).
- If the order is 2, 1, 3, the binary search tree is shown in Figure 1 (c). Please notice that if the order is 2, 3, 1, the tree is the same.
- If the order is 3, 1, 2, the binary search tree is shown in Figure 1 (d).




(a) (b) (c) (d)

Figure 1: Different shapes of binary search trees.

A: There are 24 (4!) different orders of 1, 2, 3, 4. How many different binary search trees will these different orders create?

B: There are 120 (5!) different orders of 1, 2, 3, 4, 5. How many different binary search trees will these different orders create?

C: There are 720 (6!) different orders of 1, 2, 3, 4, 5, 6. How many different binary search trees will these different orders create?

D: There are 5040 (7!) different orders of 1, 2, 3, 4, 5, 6, 7. How many different binary search trees will these different orders create?

**Answer:**
14
42
132
429

# 2 Binary Search Tree (20 points)

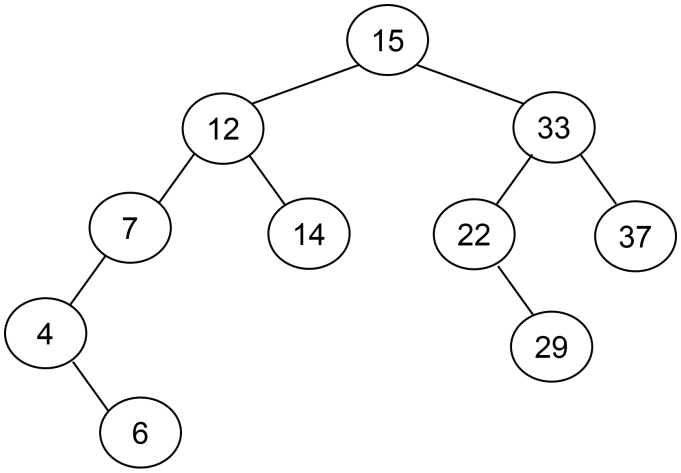



Figure 2: Binary Search Tree of 10 Nodes.

A: Using **pre-order** traversal, what is the sixth number of the ouput?

B: Using **post-order** traversal, what is the sixth number of the ouput?

C: Assume the insertion function is correct. If 9, 20, 25, and 1 are inserted (in this order) into the binary search tree shown in Figure 2, what is the **tenth** number of the output when using **pre-order** traversal?

D: Assume the insertion function is correct. If 9, 20, 25, and 1 are inserted (in this order) into the binary search tree shown in Figure 2, what is the **tenth** number of the output when using **post-order** traversal?

**Answer:**
14 (15 12 7 4 6 14 33 22 29 37)
29 (6 4 7 14 12 29 22 37 33 15)
22 (15 12 7 4 1 8 9 14 33 22 20 25 29 37)
29 (1 9 8 4 7 14 12 25 20 29 22 37 33 15)

# 3 Insertion in Binary Search Tree (20 points)

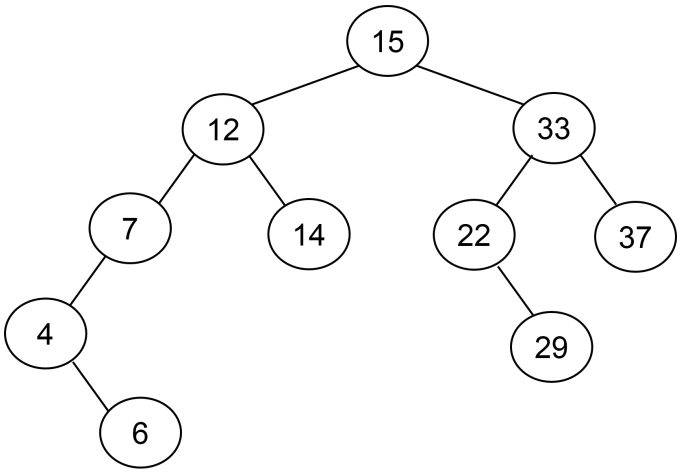



Figure 3: Binary Search Tree of 10 Nodes. This is the same as the figure in Q2.

Use the following incorrect implementation of the insert function to insert 7, 14, 22, and 33 (in this order) to the binary search tree shown in Figure 3.

```
1  // treeinsert.c
2  #include "tree.h"
3  #include <stdlib.h>
4  static TreeNode * TreeNode_construct(int val)
5  {
6    TreeNode * tn;
7    tn = malloc(sizeof(TreeNode));
8    tn -> left = NULL;
9    tn -> right = NULL;
10   tn -> value = val;
11   return tn;
12 }
13
14 TreeNode * Tree_insert(TreeNode * tn, int val)
15 {
16   if (tn == NULL)
17     {
18       // empty, create a node
19       return TreeNode_construct(val);
20     }
21   // not empty
22   if (val == (tn -> value))
23     {
```

```
24        // do not insert the same value
25        // <--- ERROR --->
26        // should return tn;
27        // but does not return anything
28      }
29    if (val < (tn -> value))
30      {
31        tn -> left = Tree_insert(tn -> left, val);
32      }
33    else
34      {
35        tn -> right = Tree_insert(tn -> right, val);
36      }
37    return tn;
38  }
```

A: Using **pre-order** traversal, what is the first number of the ouput?

B: Using **pre-order** traversal, what is the last number of the ouput?

C: Using **post-order** traversal, what is the first number of the ouput?

D: Using **post-order** traversal, what is the last number of the ouput?

**Answer:**
Pre-order: 15 12 7 4 8 7 14 14 33 22 29 22 37 33
Post-order: 6 4 7 7 14 14 12 22 29 22 33 37 33 15
A: 15
B: 33
C: 6
D: 15

# 4 Reverse Linked List (20 points)

This program asks you to write a function that reverses a linked list by reversing the individual links between the nodes. The function's input argument is the head of the linked list and returns the head of the reversed linked list. This function should not call `malloc`. Figure 4 shows an example list and its reversed form.

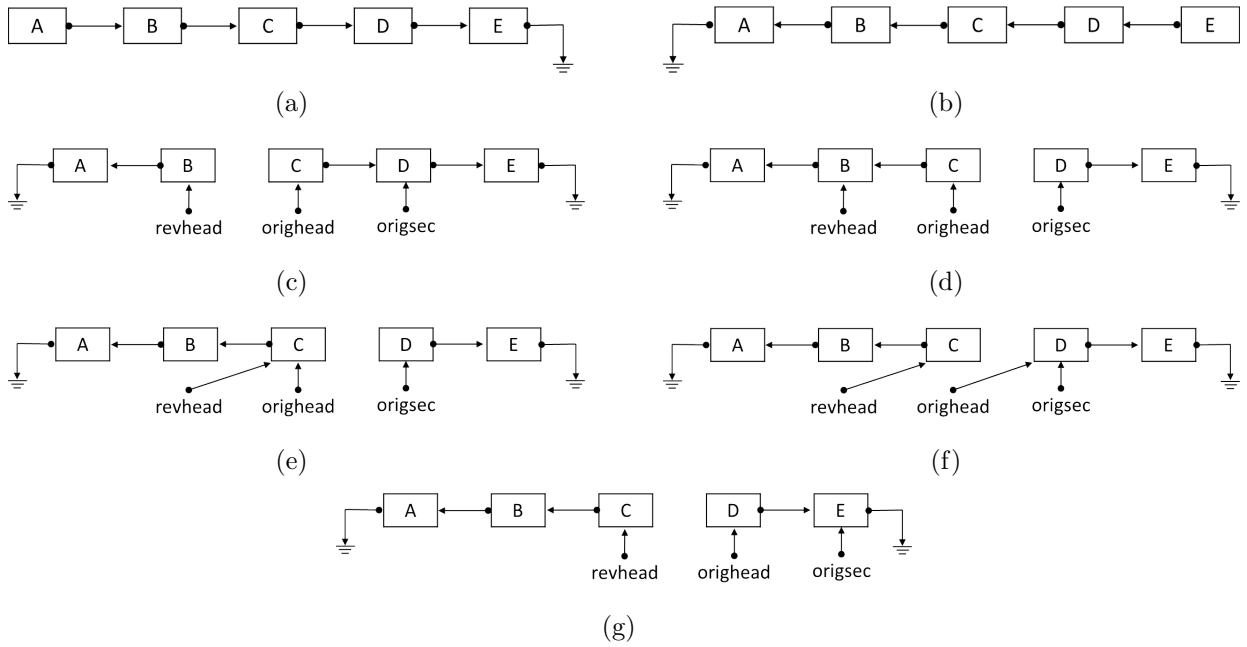



Figure 4: (a) The original linked list. The list's head points to `A`. (b) The reversed linked list. The list's head points to `E`. (c)-(g) Process for reversing a linked list.

Please fill the code.

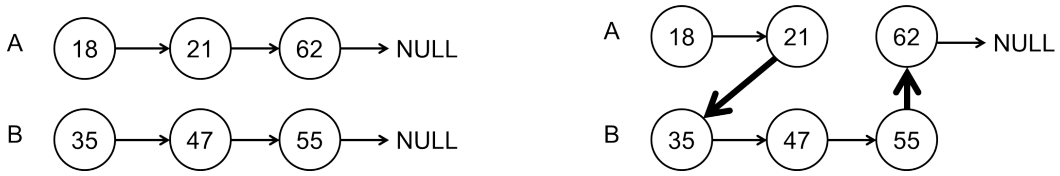
```
1  Node * List_reverse(Node * head)
2  {
3    if (head == NULL)
4      {
5        // empty list, nothing to do
6        return NULL;
7      }
8    Node * orighead = head;
9    Node * revhead = NULL; // must initialize to NULL
10   Node * origsec; // will be assigned before using
11   while (orighead != NULL)
12     {
13       // ---> FIX ME <--- A
14       // assign orighead's next to origsec
15
16
17       // ---> FIX ME <--- B
18       // assign revhead to orighead's next
19
20
21       // ---> FIX ME <--- C
22       // assign orighead to revhead
23
24       // ---> FIX ME <--- D
25       // assign origsec to orighead
26
27
28     }
29   return revhead;
30 }
```

**Answer:**

```
origsec = orighead -> next;
orighead -> next = revhead;
revhead = orighead;
orighead = origsec;
```

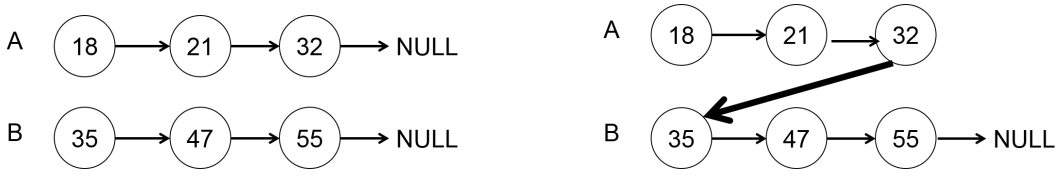
# 5 Merge Two Linked Lists (20 points)

This question asks you to merge two sorted linked lists into a single sorted linked list. An example of two sorted lists are shown in the figure below.
Merging the two sorted list changes **some** links. Figure (a) shows an example of two sorted linked lists. The modified links, as shown in Figure (b), are marked by thicker arrows. In this example, **two** links are changed.



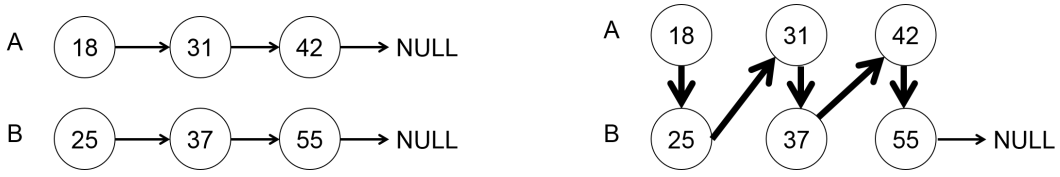

(a) (b)

This is another example of two sorted linked lists. Only one link needs to be modified.




(c) (d)

This is yet another example of two sorted linked lists. When they merge, five links need to be modified.




(e) (f)

Consider two sorted linked lists storing **distinct** values (i.e., every value is unique).

A: If each list has 2 nodes, what is the **minumum** number of changed links?

B: If each list has 2 nodes, what is the **maximum** number of changed links?

C: If each list has 4 nodes, what is the **minimum** number of changed links?

D: If each list has 4 nodes, what is the **maximum** number of changed links?

**Answer:**
A: 1
B: 3
C: 1
D: 7