COP 3502C – Exam 2 Review

1. Which would be the most efficient data structure to use to implement a queue from the options below?
   1. Singly linked list, storing just a head pointer
   2. Doubly linked list, storing both head and tail pointer
   3. Doubly linked list, storing just a head pointer
   4. Singly linked list, storing both head and tail pointer
2. Which would be the most efficient data structure to use to implement a stack from the options below?
   1. Singly linked list, storing just a head pointer
   2. Doubly linked list, storing both head and tail pointer
   3. Doubly linked list, storing just a head pointer
   4. Singly linked list, storing both head and tail pointer
3. What is the post-fix expression that represents the following in-fix expression?  
   ((7+5)/(7-5)\*1/6+3)
4. Find the value represented by the following post-fix notation:  
   4 5 4 + 3 / 3 / 1 / +
5. Write the pseudo code for a function that would do what we just did on question 4.  
   Assume you have access to all stack functions.  
     
   int findVal(char \* arr, int arrLen)  
   {  
     
   // create a new stack that we can use

Node \* head = NULL

For (int i = 0; i < arrLen; i++)

// check if it is an integer

if it is an integer we push to the stack

if it is not an integer

pop the top 2 things

do the operation that the operand tells us on those two integers

push the result of the operation unto the stack

return head->data   
}

1. How could we change the following function to make it a preorder traversal?  
   // put the print statement above the recursive calls

// change the second recursive step to root->right

void preOrder(BSTNode \* root)  
{  
 if (!root)  
 return;  
 preOrder(root->left);  
 preOrder(root->left);  
 printf(“%d”, root->data);  
}

1. Insert the following values into an initially empty AVL tree and print the pre-order traversal after each insertion as well as the rebalance case, if needed:  
   21, 26, 30, 9, 4, 15, 28, 18, 15
2. Draw a valid tree for the following post-order:  
   2, 5, 6, 4, 3, 1, 7
3. Perform the following stack operations. Print the stack after the last operation.
   1. Push(5)
   2. Push(6)
   3. Push(7)
   4. Pop()
   5. Top()
   6. Push(4)
   7. Push(5)
   8. Pop()
   9. Pop()  
        
      5

6 5

7 6 5

6 5

6 5

4 6 5

5 4 6 5

4 6 5

6 5

1. Write a function that takes in the root of a rooted tree and frees all the nodes. The children are stored as double pointer, with the size as numChildren and the capacity as cap. Use the below prototype and struct definition.  
     
   struct Node {  
    Node \*\* children;  
    Node \* parent;  
    int numChildren, cap;  
    int data;  
   }  
   // post order traversal  
   void freeTree(Node \* root)  
   {

// base case  
 if (root == NULL)

return;

// recursive call

for (int i = 0; i < root->numChildren; i++) {

freeTree(root->children[i]);

}

free(root->children);

free(root);

}