COP 3502C – Exam 2 Review Solutions (any answer that is not here is on the video 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  
      \* queue needs access to front and back  
      \* enqueue = add to the tail  
      \* dequeue = looks at head and removes it  
      \* peek = looks at the head  
      (FIFO) = First in first out like a normal line
2. Which would be the most efficient data structure to use to implement a stack from the options below?  
   (FILO) because whatever gets added first gets removed last
   1. Singly linked list, storing just a head pointer  
      we only ever need access to the head on a stack
   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 / +  
     
     
     
     
     
     
     
     
     
     
     
     
     
     
   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 onto the stack  
    return head->data or head->value   
     
     
     
     
     
     
     
     
     
   }
6. How could we change the following function to make it a preorder traversal?  
   \*\* we put the print statement above the recursive calls  
   \*\* we were not going to the right, so we changed the second recursive call  
   void preOrder(BSTNode \* root)  
   {  
    if (!root)  
    return;  
    printf(“%d”, root->data);  
    preOrder(root->left);  
    preOrder(root->right);  
     
   }
7. 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
8. Draw a valid tree for the following post-order:  
   2, 5, 6, 4, 3, 1, 7
9. 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  
   // first do recursion, then access the current root(free it)  
   void freeTree(Node \* root)  
   {  
    // base case  
    if (root == NULL)  
    return;  
    // recursion  
    for(int I = 0; I < root->numChildren; i++)  
    {  
    freeTree(root->children[i]);  
    }  
    // we are Not freeing every child, but rather we are freeing the pointer array

Free(root->children);  
 free(root);  
// we don’t free the parent, because that is just another node that gets freed in its own recursive call  
// if we were to free the parent, we would end up double freeing because that node would get freed anyways  
  
best case finding a value in any linked list: O(1)  
worst case and average case for fiding value = O(N)  
  
  
  
  
  
  
  
  
  
  
  
}