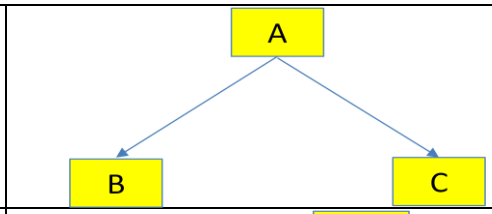
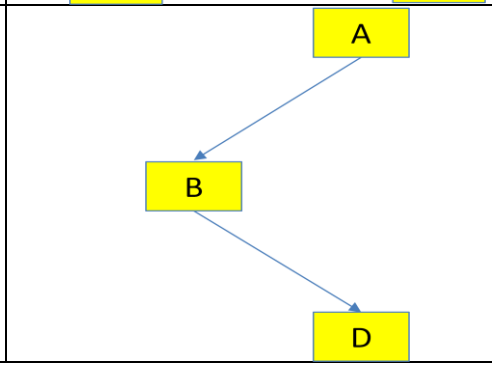
CS 2110 In-Class Quiz #2 -- Solutions

1. Suppose that we are given a binary tree that contains a dictionary of N words. Each node contains a word and has a left and a right child, sorted into alphabetic order.

T	**F**	For any such tree, finding a word will be possible in log(N) "node examination" operations.	
T	F	If the root node of the tree contains the word "mendacity", then any word alphabetically larger than "mendacity" would be found in the right-hand child of that node.	
T	F	It is important to insert the words in a random order; otherwise, the tree could turn out to be of depth N	
T	F	If a binary tree is *balanced*, the maximum node depth would be log(N)	
T	F	An N-word dictionary can actually be represented by any of N! different trees, representing permutations of the words, but many would not be balanced and many would not be useful for fast search.	
T	F	For the tree shown to the right, an inorder traversal would print B A C	
T	**F**	For the tree shown to the right, a preorder traversal would print B A D	
T	**F**	An advantage of a tree over a list (even a double-linked list) is that trees always store data in order but it is impossible to keep a list in order	
T	**F**	A static method, like Main, can call a dynamic method of a class by qualifying the method name with the class name.	
T	F	If a class xyz extends class abc, the constructor of xyz should call the constructor of abc (typically by invoking super ()) because otherwise, the constructor associated with abc won't be executed.	

2. Write a delete method for the double-linked list class shown below. The method should find the node with the given key (if any) and delete it, fixing pointers and the head pointer as required.

```
public class List {
        public  List    head;  // If non-null, points to first element
        public class ListNode {
                Object key;             // Key ("name") for this node
                Object value;           // Data associated with this list element
                List    next, last;     //  Pointers to next and last element
                public ListNode(Object k, Object v) { this.key = k; this.value = v; }
```

```
	}
	public void add(Object k, Object va) {
		ListNode ln = new ListNode(k, v);
		If(head == null) {
			// List was empty
			head = ln.next = ln.last = ln;
			return;
		}
		// Make the new ListNode the last on the list
		ln.next = head;
		head.last.next = ln;
		ln.last = head.last;
		head.last = ln;

	}
	public Object find(Object k) {
		If(head == null) return null;
		ListNode ln = head;
		do {
			if(ln.key.equals(k)) return ln.value;
			ln = ln.next;
		}
		while(ln != head);
		return null;
	}
	public void delete(Object k) {
		// First find the node
		If(head == null) return;
		ListNode ln = head;
		do {
			if(ln.key.equals(k)) {
				// Unlink it from the list
				ln.last.next = ln.next;
				ln.next.last = ln.last;
				// Empty list?  Set head to null.
				if(head.next == head)
					head = null;
				return;
			}
			ln = ln.next;
		}
		while(ln != head);
	}
}
```