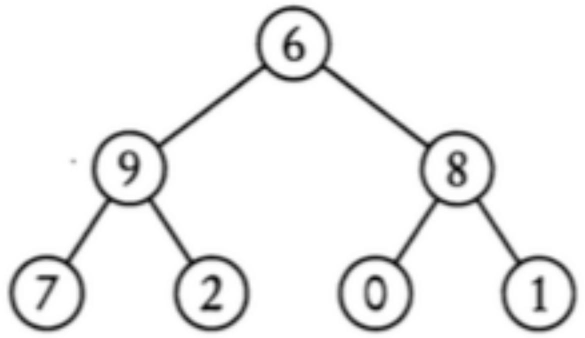
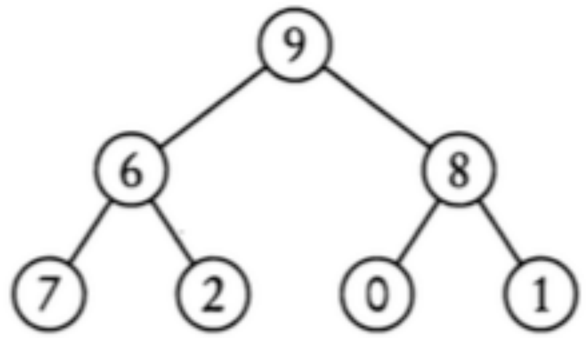
7. An array with no duplicate values is to be sorted into increasing order using heapsort. Step one is to insert the elements of the array sequentially into a max-heap. (Recall that a max-heap with no duplicates is a complete binary tree in which the value in each node is larger than the values in its children's nodes.)

If the given array is 6 7 1 9 2 0 8, what will the contents of the max-heap be after all the elements are inserted?

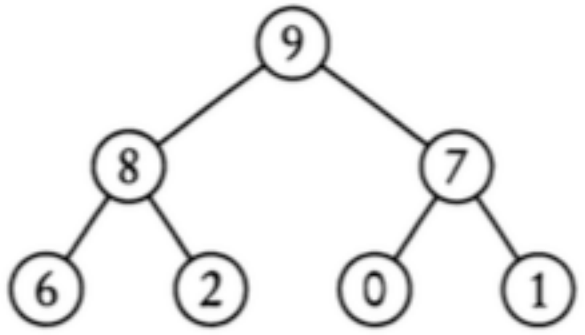
(A)



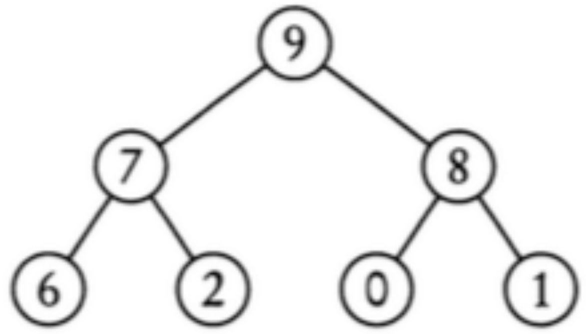
(B)



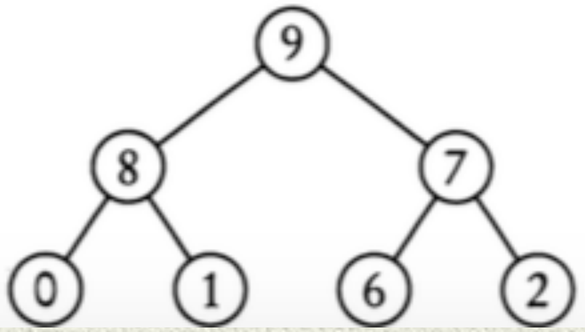
(C)

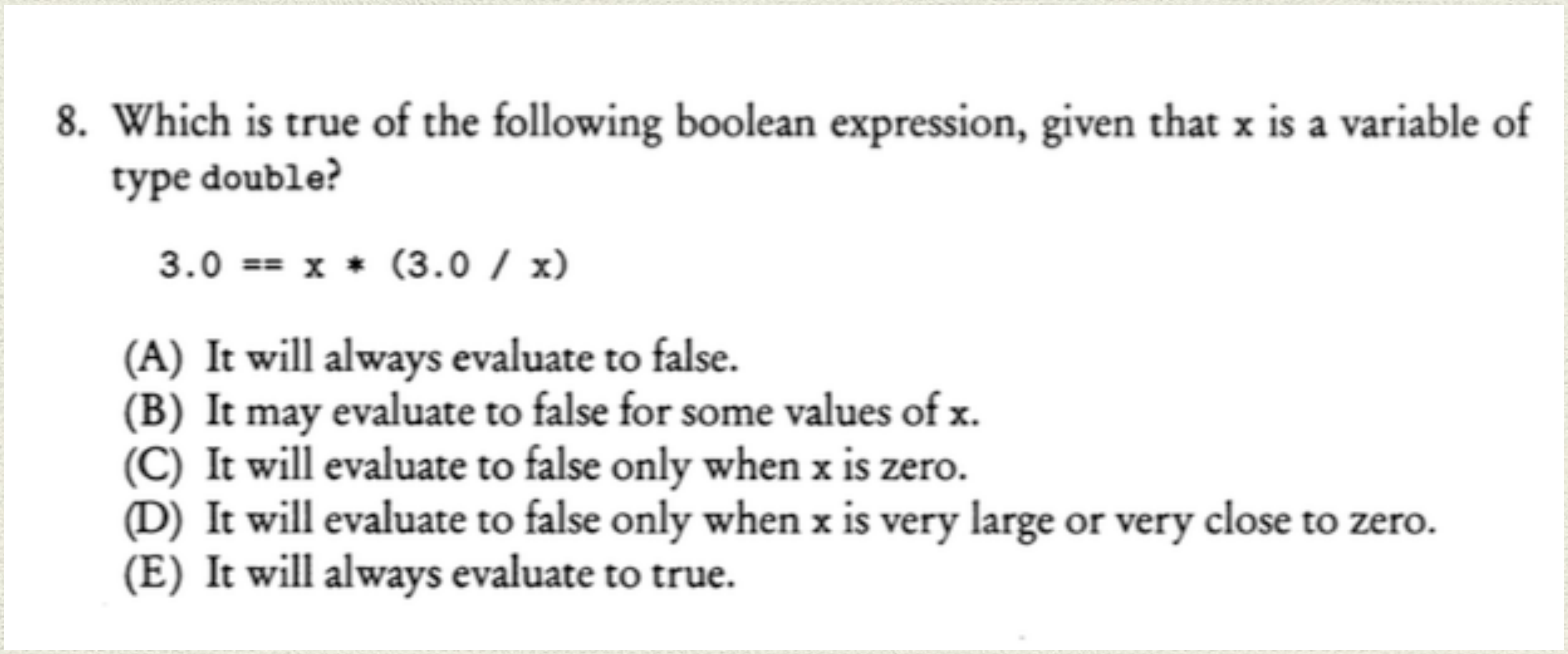


(D)



(E)

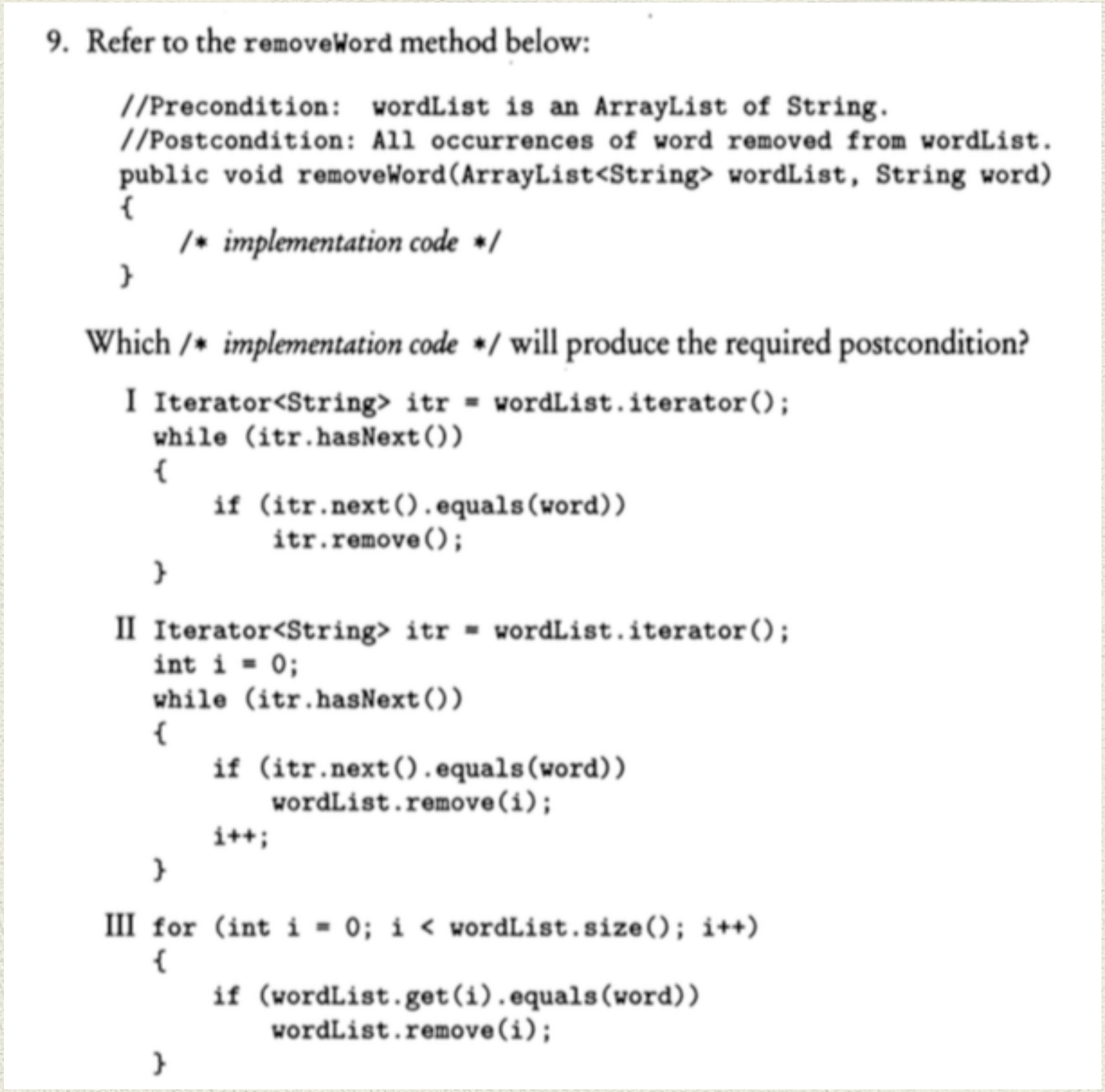




8. Which is true of the following boolean expression, given that `x` is a variable of type `double`?

```
3.0 == x * (3.0 / x)
```

(A) It will always evaluate to false.
(B) It may evaluate to false for some values of `x`.
(C) It will evaluate to false only when x is zero.
(D) It will evaluate to false only when x is very large or very close to zero.
(E) It will always evaluate to true.



9. Refer to the `removeWord` method below:

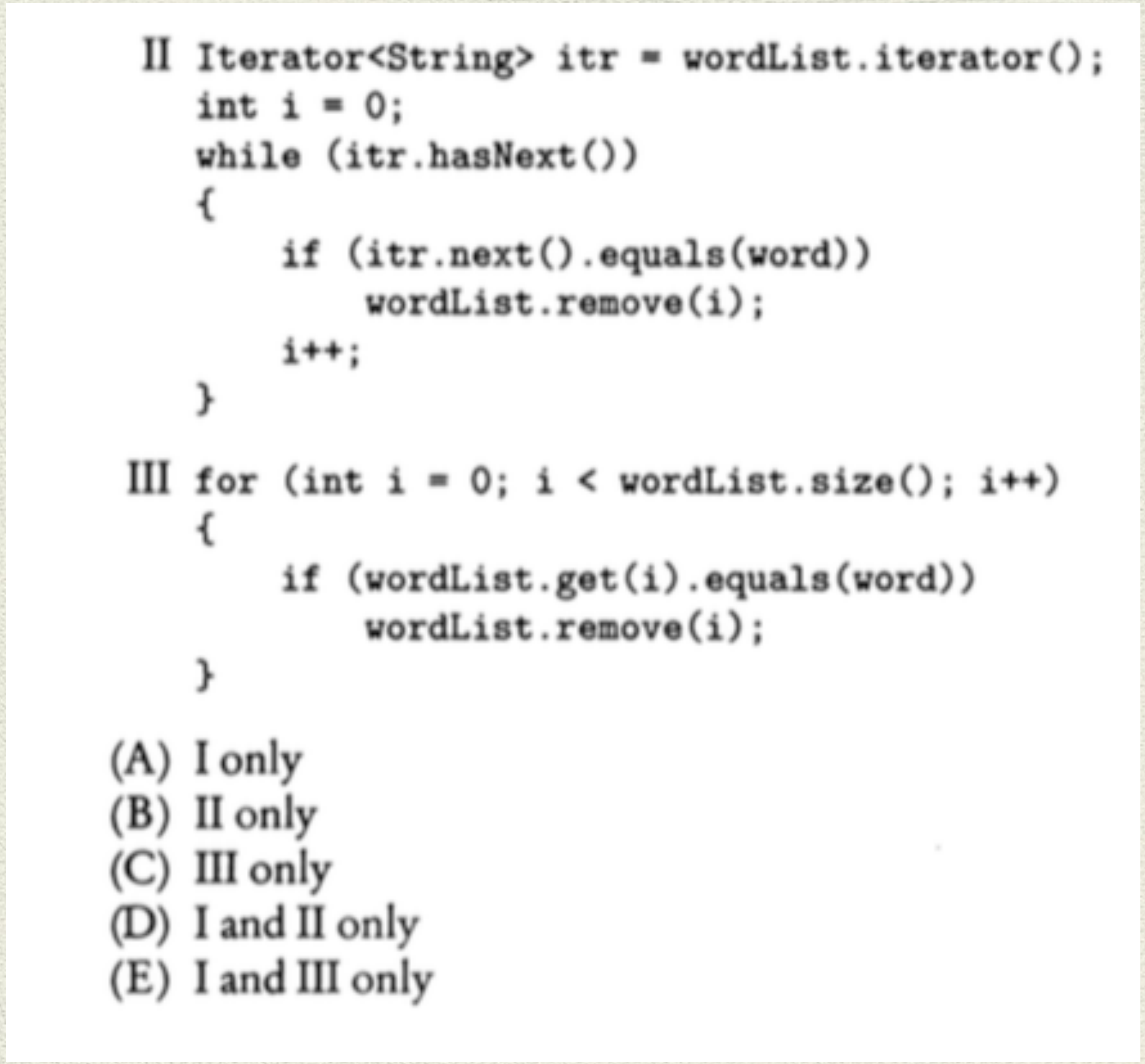
```
//Precondition:  wordList is an ArrayList of String.
//Postcondition: All occurrences of word removed from wordList.
public void removeWord(ArrayList<String> wordList, String word)
{
    /* implementation code */
}
```

Which /* *implementation code* */ will produce the required postcondition?

I
```
Iterator<String> itr = wordList.iterator();
while (itr.hasNext())
{
    if (itr.next().equals(word))
        itr.remove();
}
```

II
```
Iterator<String> itr = wordList.iterator();
int i = 0;
while (itr.hasNext())
{
    if (itr.next().equals(word))
        wordList.remove(i);
    i++;
}
```

III
```
for (int i = 0; i < wordList.size(); i++)
{
    if (wordList.get(i).equals(word))
        wordList.remove(i);
}
```



```
II  Iterator<String> itr = wordList.iterator();
    int i = 0;
    while (itr.hasNext())
    {
        if (itr.next().equals(word))
            wordList.remove(i);
        i++;
    }

III for (int i = 0; i < wordList.size(); i++)
    {
        if (wordList.get(i).equals(word))
            wordList.remove(i);
    }
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only

Assume that linked lists are implemented with the `ListNode` class provided.

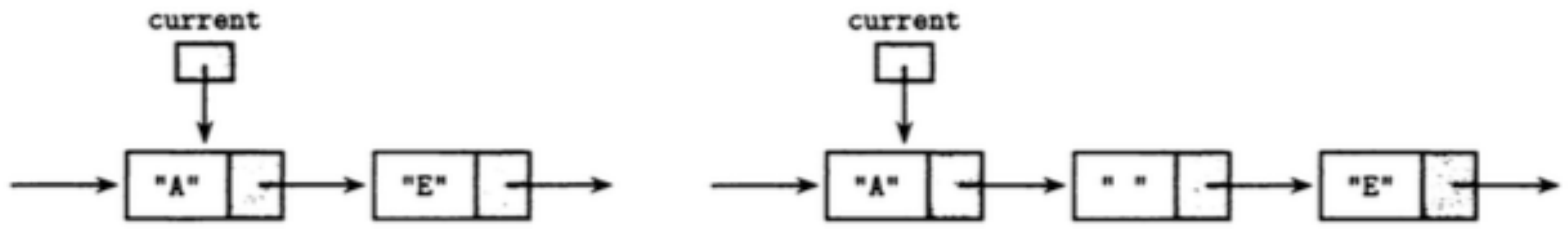
Refer to method `insertBlank` for Questions 10 and 11.

```
//Precondition:  current refers to a node in a linear linked
//               list of character strings. current is not null.
//Postcondition: The node following the node that current
//               refers to contains a blank.
public static void insertBlank(ListNode current)
```

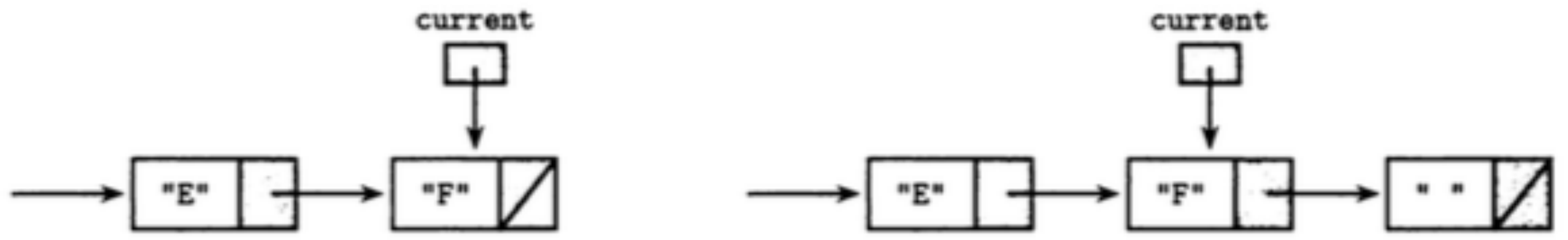
Examples:

Before calling `insertBlank` After calling `insertBlank`

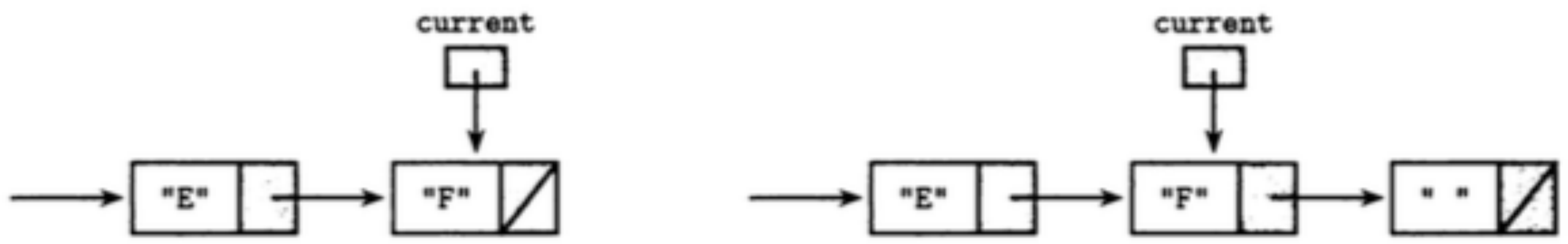
Example 1



Example 2



Example 2



10. Which of the following could be used as the body of `insertBlank` such that its postcondition is satisfied?

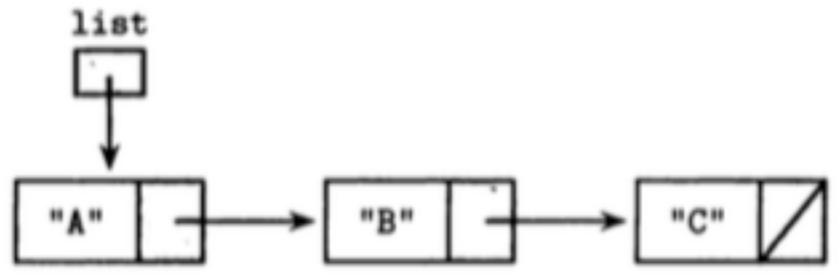
I
```
current.setNext(new ListNode(" ", current.getNext()));
```

II
```
ListNode p = new ListNode(" ", current.getNext());
current = p;
```

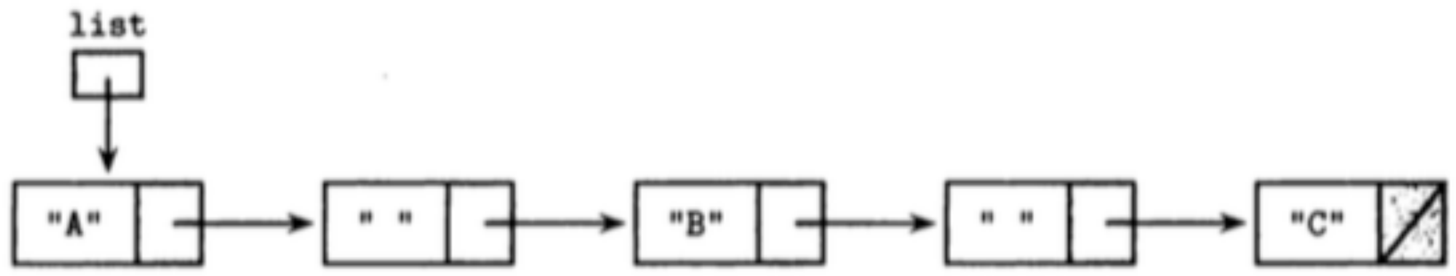
III
```
ListNode p = new ListNode(null, null);
p.setNext(current.getNext());
p.setValue(" ");
current.setNext(p);
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only

11. A method `padList`, whose code is given below, is to insert a blank between each pair of existing nodes in its parameter, `list`, a linked list of character strings. For example, if the list is initially

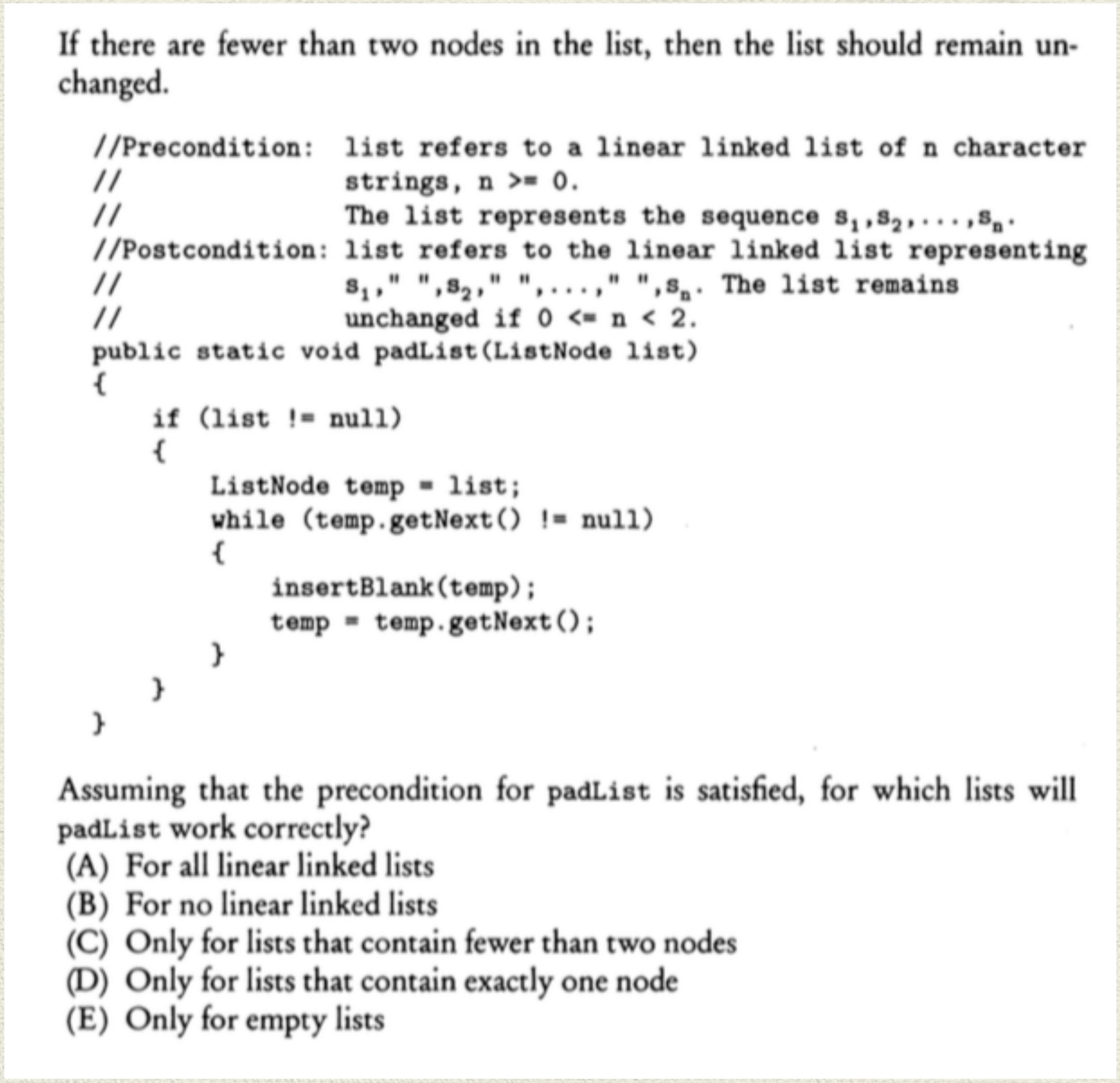


`padList(list)` should result in



If there are fewer than two nodes in the list, then the list should remain unchanged.

```
//Precondition:  list refers to a linear linked list of n character
//               strings, n >= 0.
//               The list represents the sequence s1,s2,...,sn.
//Postcondition: list refers to the linear linked list representing
//               s1," ",s2," ",...," ",sn. The list remains
//               unchanged if 0 <= n < 2.
```



If there are fewer than two nodes in the list, then the list should remain unchanged.

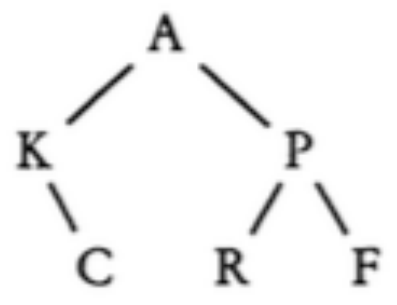
```
//Precondition:  list refers to a linear linked list of n character
//               strings, n >= 0.
//               The list represents the sequence s_1,s_2,...,s_n.
//Postcondition: list refers to the linear linked list representing
//               s_1," ",s_2," ",...," ",s_n. The list remains
//               unchanged if 0 <= n < 2.
public static void padList(ListNode list)
{
    if (list != null)
    {
        ListNode temp = list;
        while (temp.getNext() != null)
        {
            insertBlank(temp);
            temp = temp.getNext();
        }
    }
}
```

Assuming that the precondition for `padList` is satisfied, for which lists will `padList` work correctly?

(A) For all linear linked lists
(B) For no linear linked lists
(C) Only for lists that contain fewer than two nodes
(D) Only for lists that contain exactly one node
(E) Only for empty lists

12. The binary tree shown is traversed preorder. During the traversal, each element, when accessed, is pushed onto an initially empty stack s of `String`. What output is produced when the following code is executed?

```
while (!s.isEmpty())
    System.out.print(s.pop());
```



(A) `AKCPRF`
(B) `CKRFPA`
(C) `FPRACK`
(D) `APFRKC`
(E) `FRPCKA`