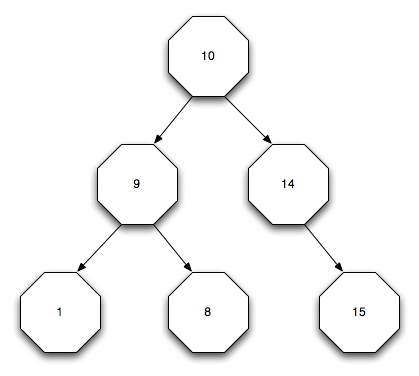
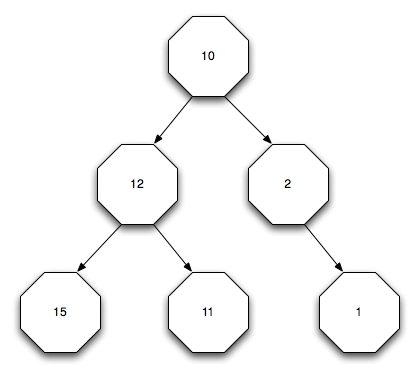
1. True or False: The following is a binary search tree



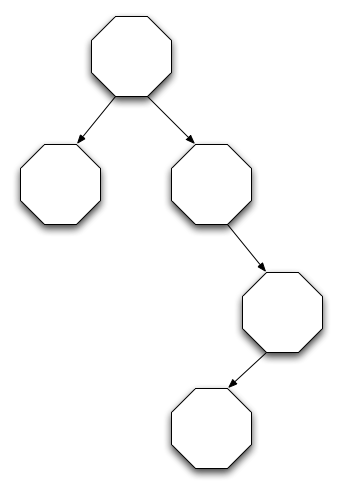
Answer: False. The 8 and 9 are out of order.

2. True or False: The following is a binary search tree



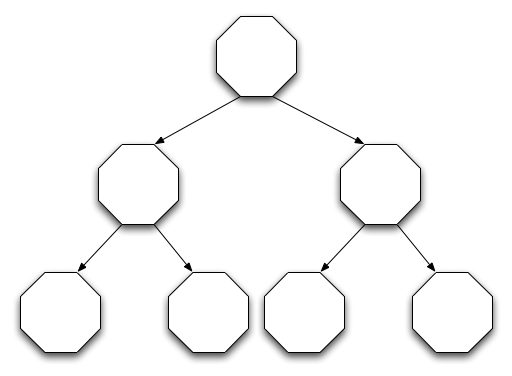
Answer: True. Using the definition in my slides, it is. It’s a descending binary search tree.

3. Is the following tree balanced? If so, explain what makes it balanced. If not, explain what needs to be done to make it balanced.



Answer: No. The root’s right child needs a left child for the tree to be balanced.

4. Is the following tree balanced? If so, explain what makes it balanced. If not, explain what needs to be done to make it balanced.



Answer: Yes, it is balanced. The depth of all leaves differ by ≤ 1.

5. Consider the following code:

SUBROUTINE INSERT(T, V)

TYPE(TREE), INTENT(INOUT) :: T

INTEGER, INTENT(IN) :: V

TYPE(TREE), POINTER :: N

ALLOCATE(N)

IF (.NOT. ASSOCIATED(T)) THEN

T => N

ELSE IF (V < T%VALUE) THEN

CALL INSERT(T%LEFT, V)

ELSE

CALL INSERT(T%RIGHT, V)

END IF

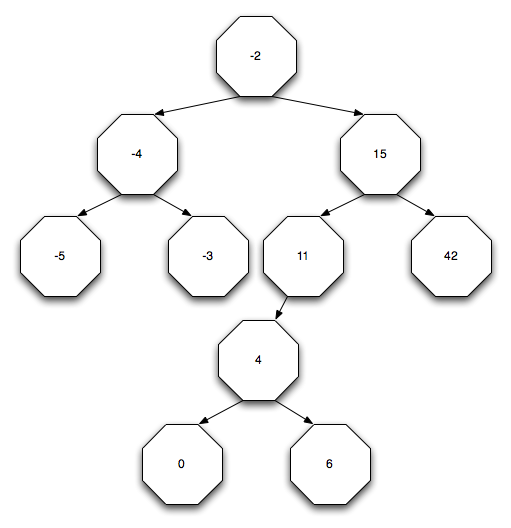
END SUBROUTINE INSERT

Assume that the TREE type has been defined elsewhere with pointers LEFT and RIGHT for the left and right children and an integer field named VALUE. What bug(s) exist? How could they be fixed?

Answer: There are two bugs.

1. The subroutine calls itself but it is not marked as RECURSIVE
2. A new node is allocated but the value is never saved to it

6. What are the pre-order, in-order, and post-order traversals of the following tree?



Answer:

1. -2 -4 -5 -3 15 11 4 0 6 42
2. -5 -4 -3 -2 0 4 6 11 15 42
3. 5 -3 -4 0 6 4 11 42 15 -2