1. Write the `expand_leaf` function. `Expand_leaf` takes three arguments: a binary tree, and two values. It returns a new tree that is identical to the original tree, except that every node that was a leaf in the original tree now has both a left and a right child, whose values are equal to the second and third arguments to `expand_leaf` respectively. `Expand_leaf` must not modify the original tree in any way.

Solution:

```
tree_type
expand_leaf(tree_type tree, tree_item_type leftval, tree_item_type rightval)
{
  if (empty_tree(tree))
    return(the_empty_tree);
  else if (empty_tree(left_child(tree)) && empty_tree(right_child(tree)))
    return(tree_cons(tree_data(tree),
                     tree_cons(leftval, the_empty_tree, the_empty_tree),
                     tree_cons(rightval, the_empty_tree, the_empty_tree)));
  else return(tree_cons(tree_data(tree),
                        expand_leaf(left_child(tree), leftval, rightval),
                        expand_leaf(right_child(tree), leftval, rightval)));
}
```

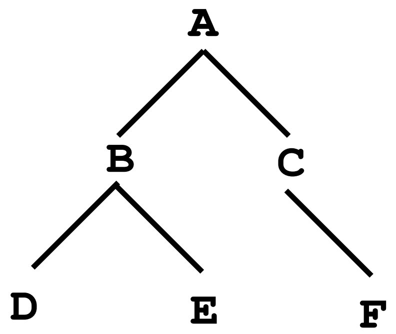
2. Write the `height` function. `Height` takes a binary tree as its argument, and returns the height of the tree. The height of a tree is the maximum level of any node in the tree. The level of a node in a tree is its distance from the root. Assume that the root of a tree has level 0, the children of the root have level 1, *etc.* `Height` should return `-1` if given the empty tree as argument.

Solution:

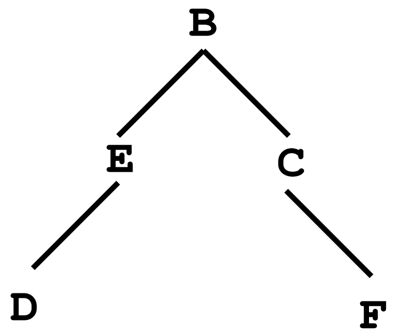
```
int
max(int num1, int num2)
{
  if (num1 > num2)
    return(num1);
  else return(num2);
}

int
height(tree_type tree)
{
  if (empty_tree(tree))
    return(-1);
  else return(1 + max(height(left_child(tree)), height(right_child(tree))));
}
```

3. Write the `delete_root` function. `Delete_root` takes a binary tree as its argument, and returns a new tree that contains every value stored in the original tree except for the value stored at its root. The root of the new tree should be one of the children of the root of the original tree (the other child of the original tree must remain where it was); each child of the root of the new tree should be either a child or a grandchild of the root of the old tree (with nodes not promoted to fill a blank remaining where they were); and so on. For example, if `delete_root` were called on the tree:



it might return the tree:



Note that other solutions are possible; your program may return any legal solution. If `delete_root` is called on the empty tree, or on a tree containing exactly one node, it should return the empty tree. `Delete_root` must not modify its argument in any way; it should produce a brand new tree.

Solution:

```
tree_type
delete_root(tree_type tree)
{
  if (empty_tree(tree))
    return(the_empty_tree);
  else if (empty_tree(left_child(tree)) && empty_tree(right_child(tree)))
    return(the_empty_tree);
  else if (!empty_tree(left_child(tree)))
    return(tree_cons(tree_data(left_child(tree)),
                     delete_root(left_child(tree)),
                     right_child(tree)));
  else return(tree_cons(tree_data(right_child(tree)),
                        left_child(tree),
                        delete_root(right_child(tree))));
}
```