Write a Recursive Function to Flatten a Tree

Let's write a function to **flatten such a tree**
- **into an array of integers**.
- **For `NULL`** subtrees, we **use** the symbolic constant **`ABSENT`**.

```
int32_t pack_tree (int32_t ar[],
    int32_t len, int32_t pos,
    node_t* root);
```

`pos` is the current writing position (starts at 0)

The function returns the final length written or -1 on failure (array too short to fit the tree).

Stopping Condition: Reached an Empty Subtree

We'll write the function recursively.

First, we check for **`NULL`**:

```
if (NULL == root) {
    if (len <= pos) {
        return -1;
    }
    ar[pos] = ABSENT;
    return (pos + 1);
}
```

Enough space to write **`ABSENT`**?

Add **`ABSENT`** to end of array.

Indicate that another space has been used.

Pack the Three Subtrees Recursively

Next, we write the three subtrees recursively.

On failure, we also fail.

```
if (-1 == (pos = pack_tree
    -1 =
    -1 =
    ret
}
```

This code is a little tricky.

First, the leap of faith:
`pack_tree` writes a tree into an array.
It works.
We haven't finished writing it yet.
But we have to assume that it works.
If it fails, it returns -1.

Pack the Three Subtrees Recursively

Next, we write the three subtrees recursively.

On failure, we also fail.

```
if (-1 == (pos = pack_tree
        (ar, len, pos, root->left)) ||
```

Return value gives the new array position for writing.

Pass current array position for writing.

Check for failure.

On failure, logical OR stops evaluating!

