

A Recursive Factorial Function

```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

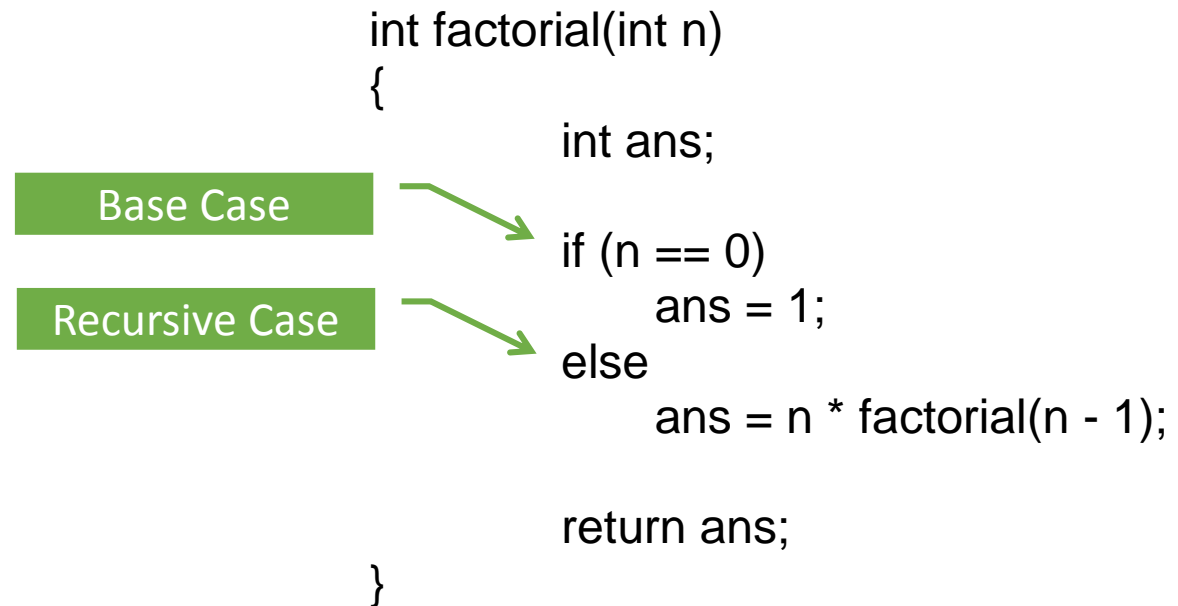
    return ans;
}
```

A Recursive Factorial Function

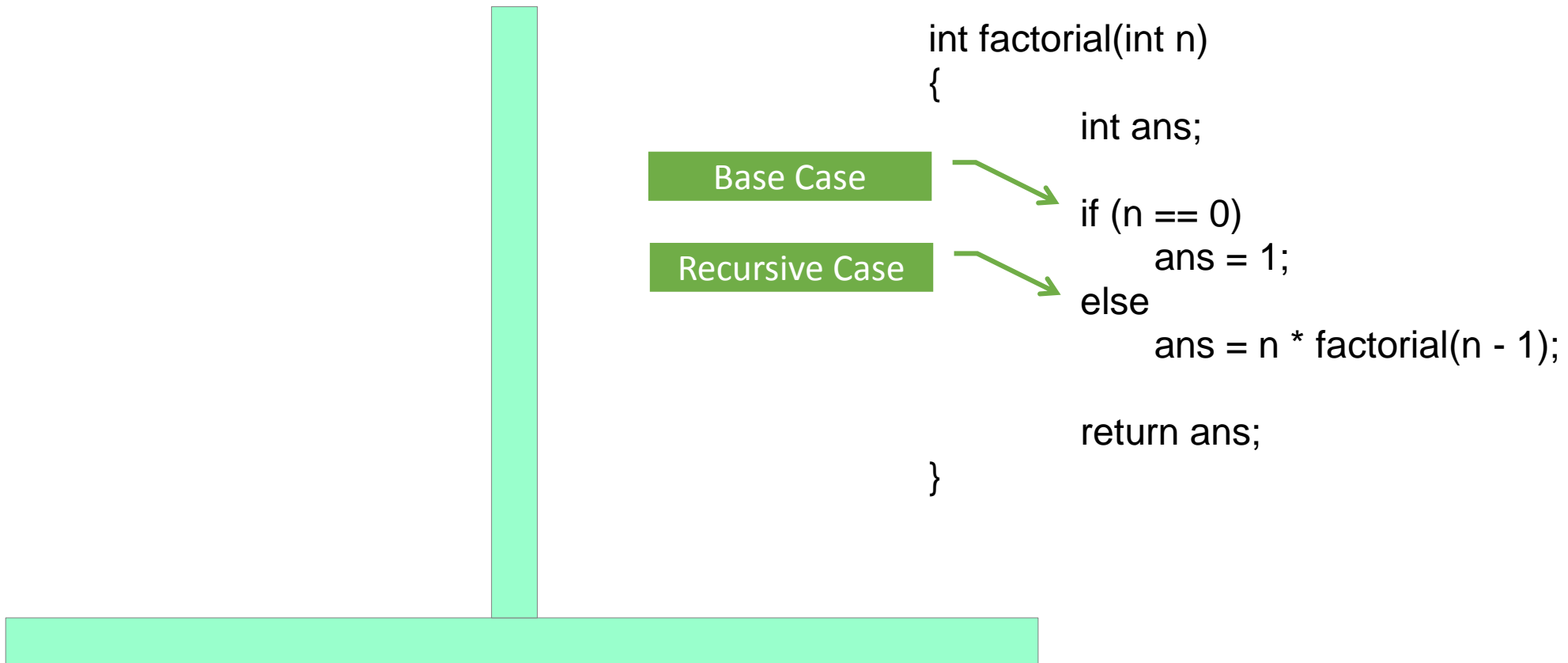
```
int factorial(int n)
{
    int ans;

    Base Case → if (n == 0)
                  ans = 1;
    Recursive Case → else
                      ans = n * factorial(n - 1);

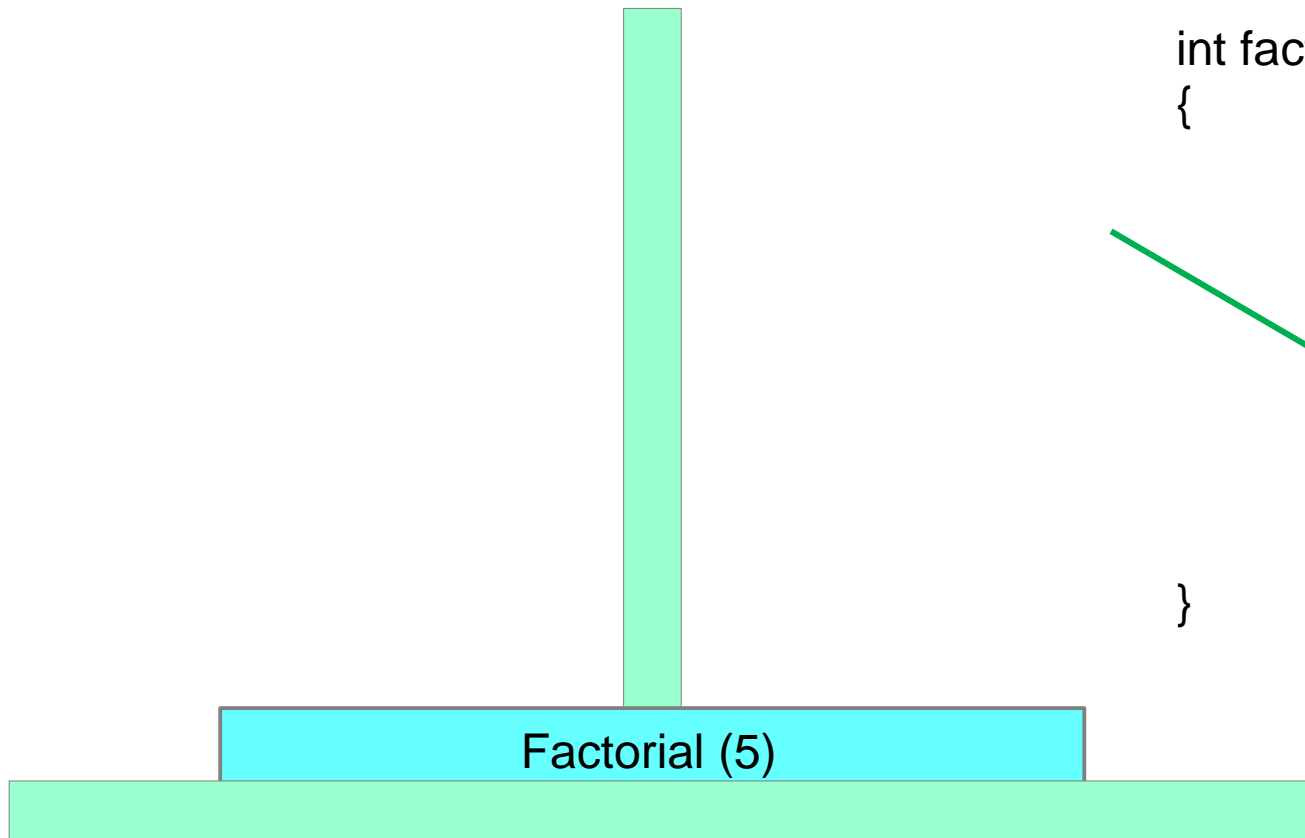
    return ans;
}
```



Call Stack Factorial (5)



Call Stack Factorial (5)



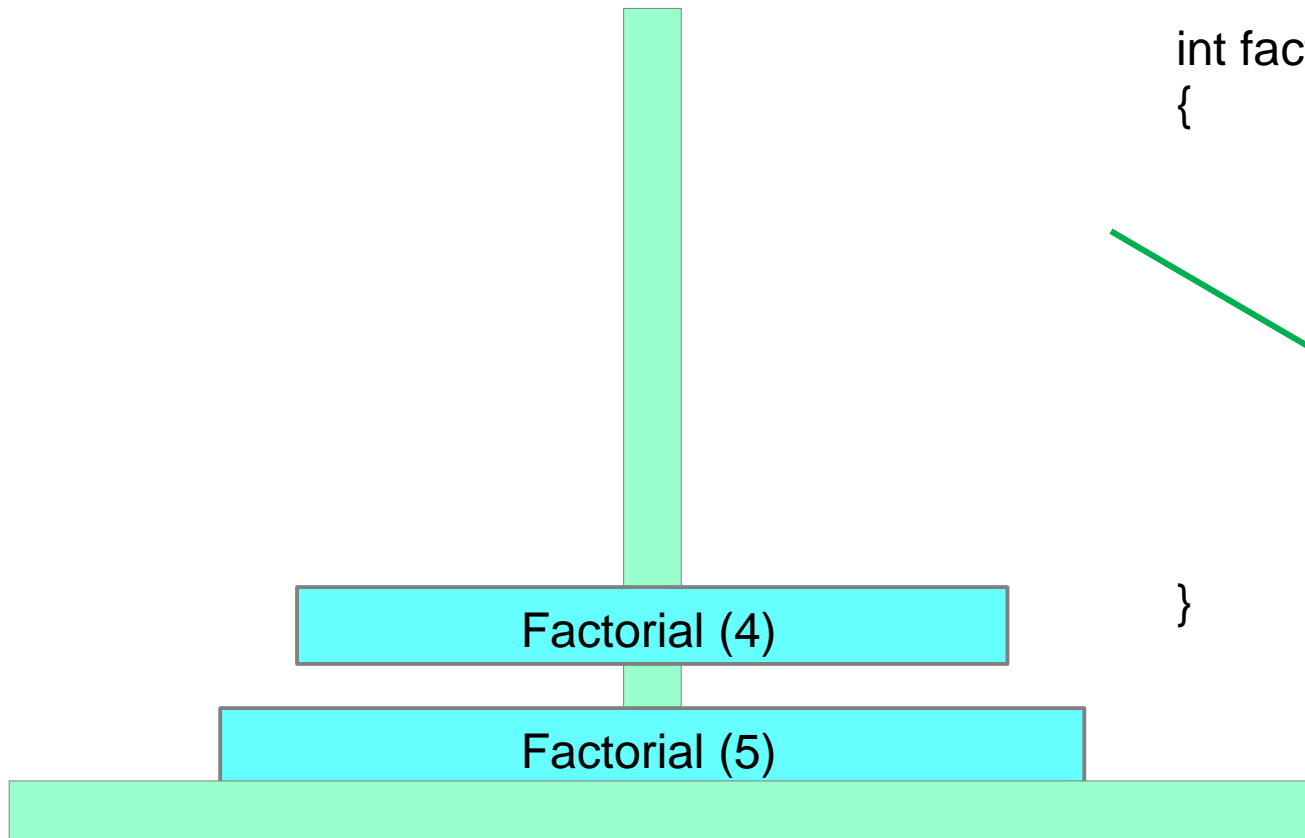
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);
    return ans;
}
```

A green arrow points from the `factorial(n - 1)` call in the code to the `Factorial (5)` frame in the call stack diagram above.

5 * factorial(4)

Call Stack Factorial (5)

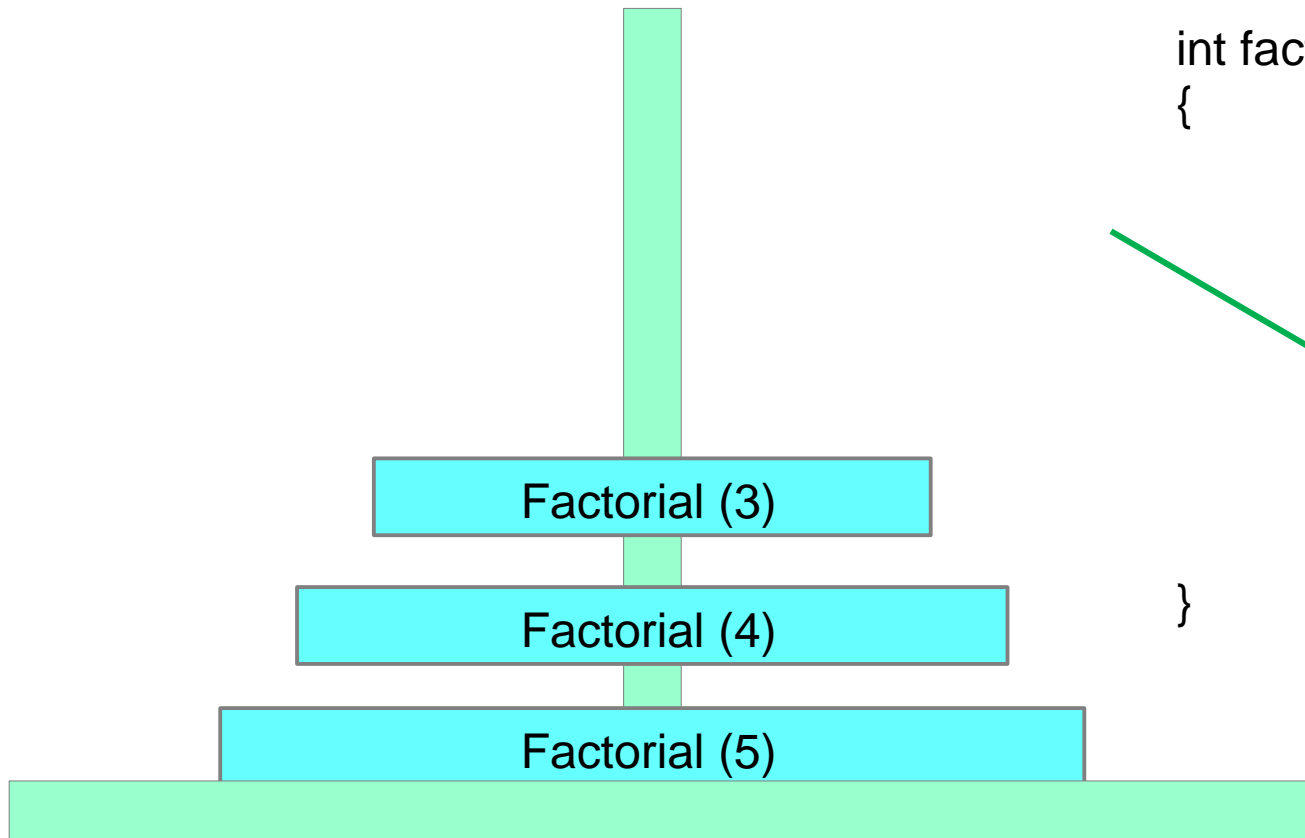


```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);
    return ans;
}
```

The code block shows the implementation of the factorial function. A green arrow points from the 'Factorial (4)' block in the call stack to the recursive call line: `ans = n * factorial(n - 1);`. The expression `4 * factorial(3)` is highlighted in yellow, indicating the current state of the calculation where `n` is 4.

Call Stack Factorial (5)

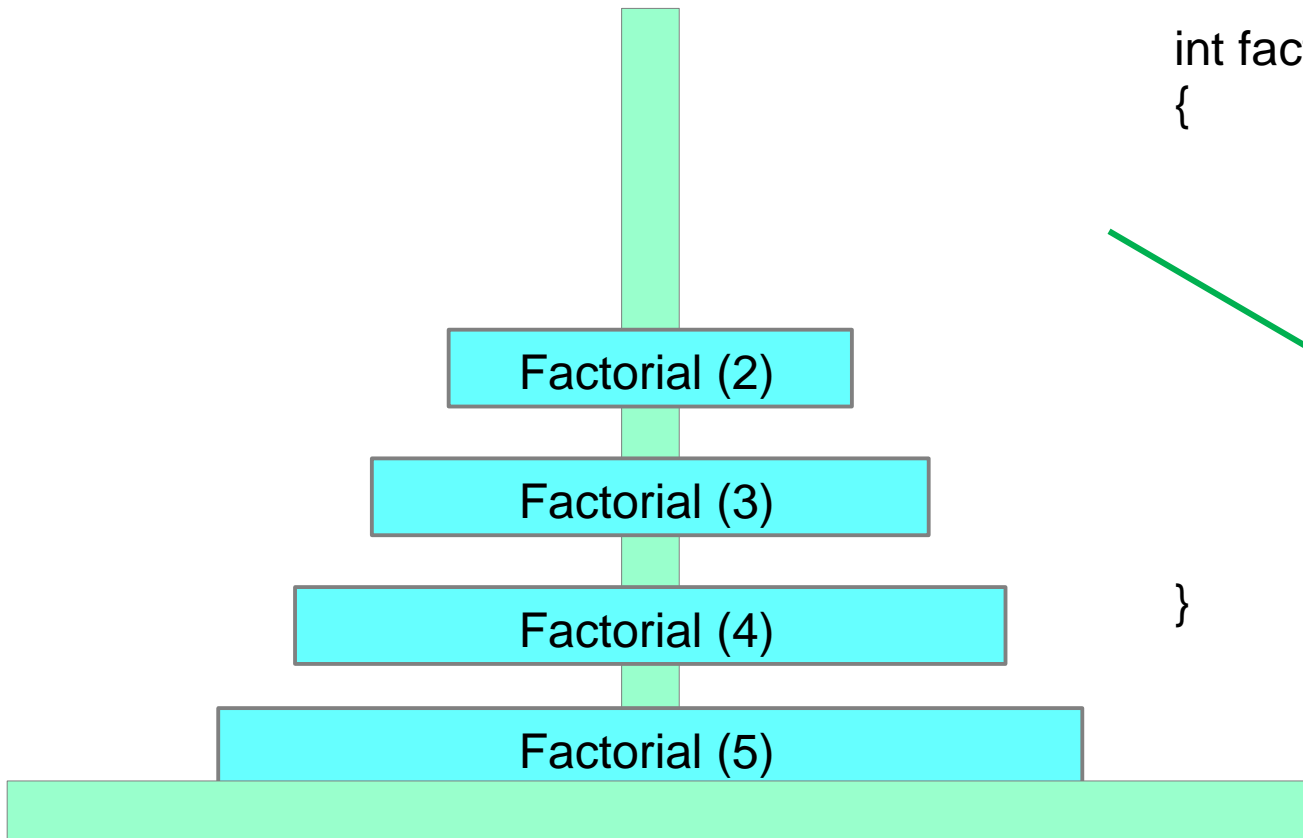


```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);
    return ans;
}
```

A green arrow points from the 'else' branch of the code to the 'Factorial (3)' call in the stack diagram. The expression `3 * factorial(2)` in the code is highlighted in yellow.

Call Stack Factorial (5)



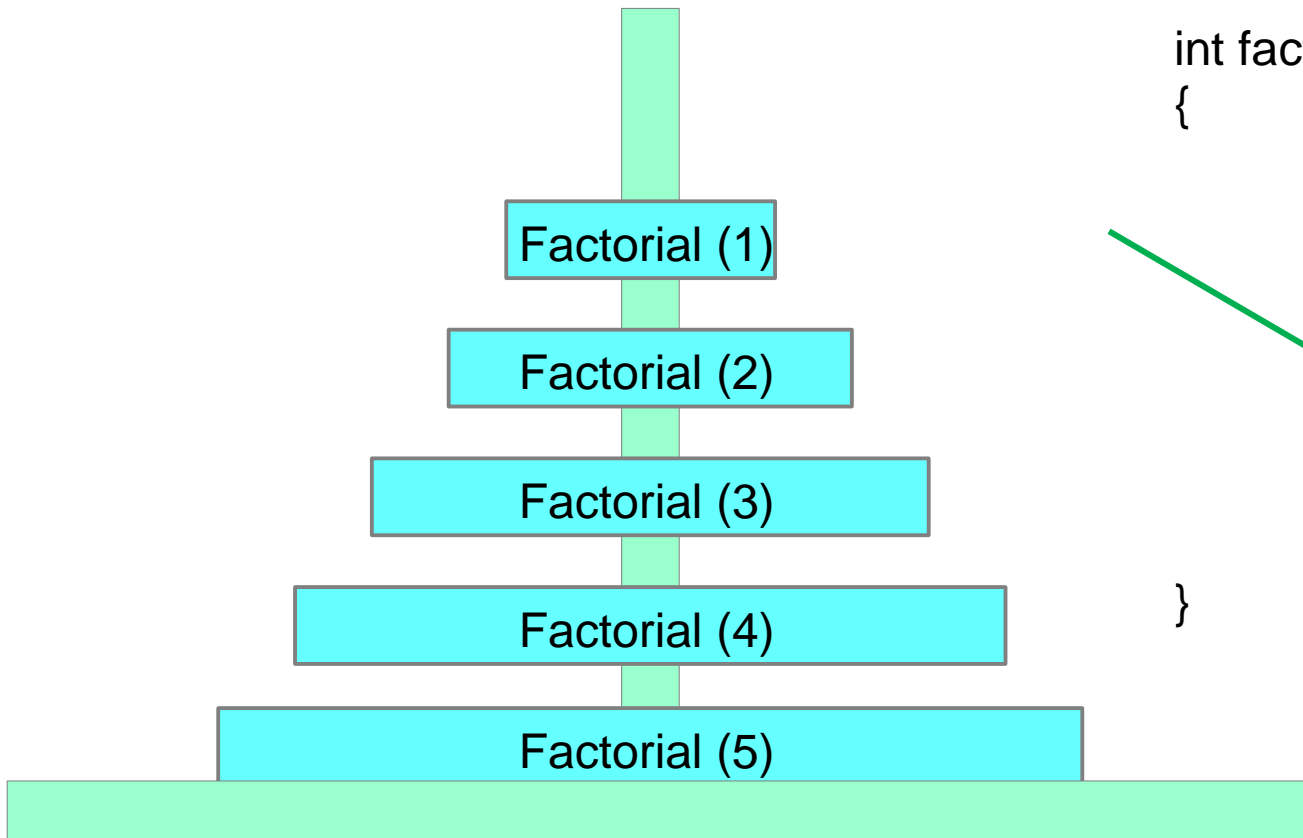
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);
    return ans;
}
```

A green arrow points from the `factorial(n - 1)` call in the code to the `Factorial (4)` box in the call stack diagram.

2 * factorial(1)

Call Stack Factorial (5)

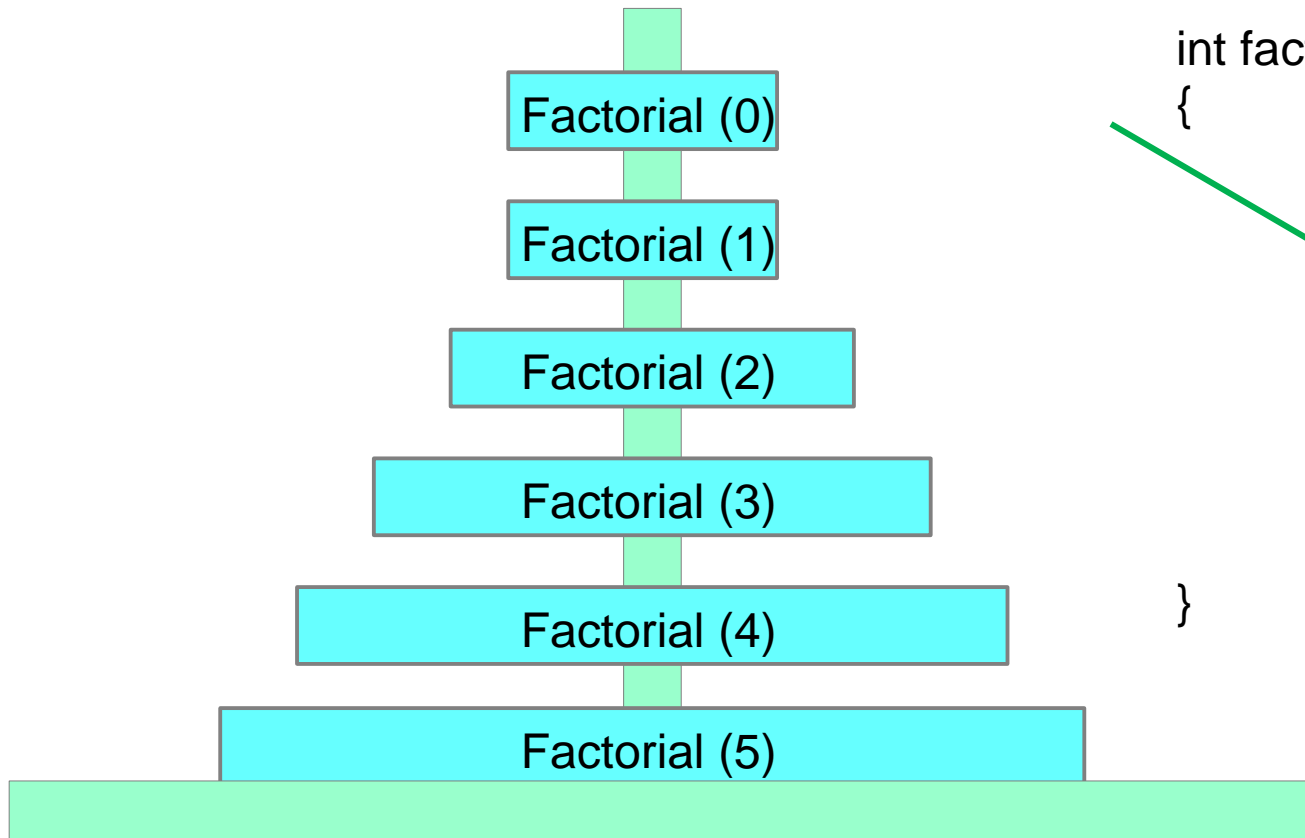


```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);
    return ans;
}
```

A green arrow points from the 'Factorial (4)' call in the stack diagram to the recursive call line in the code: `ans = n * factorial(n - 1);`. In this line, the expression `1 * factorial(0)` is highlighted in yellow, indicating the state of the program when the stack reaches the 'Factorial (1)' call.

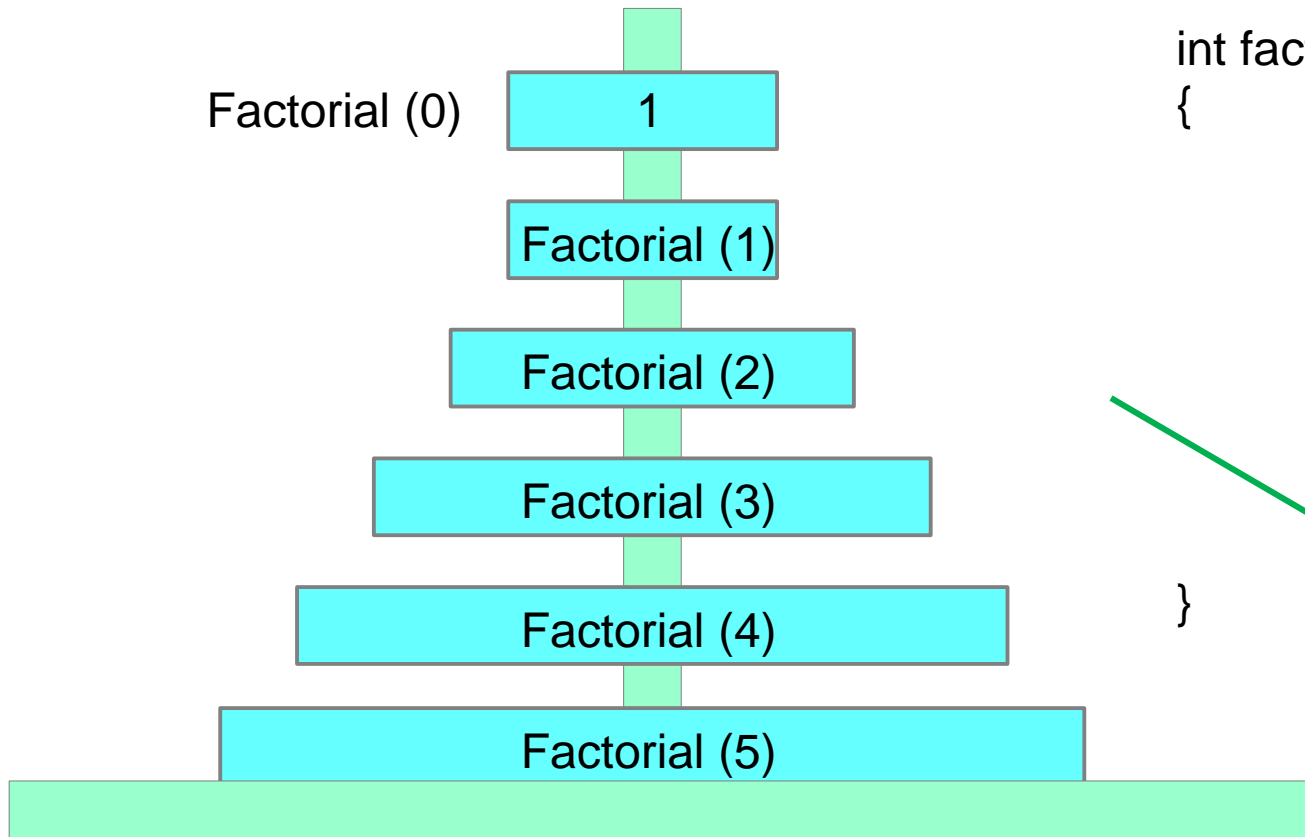
Call Stack Factorial (5)



```
int factorial(int n)
{
    int ans;
    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);
    return ans;
}
```

A green arrow points from the opening curly brace of the `factorial` function to the `if (n == 0)` condition.

Call Stack Factorial (5)



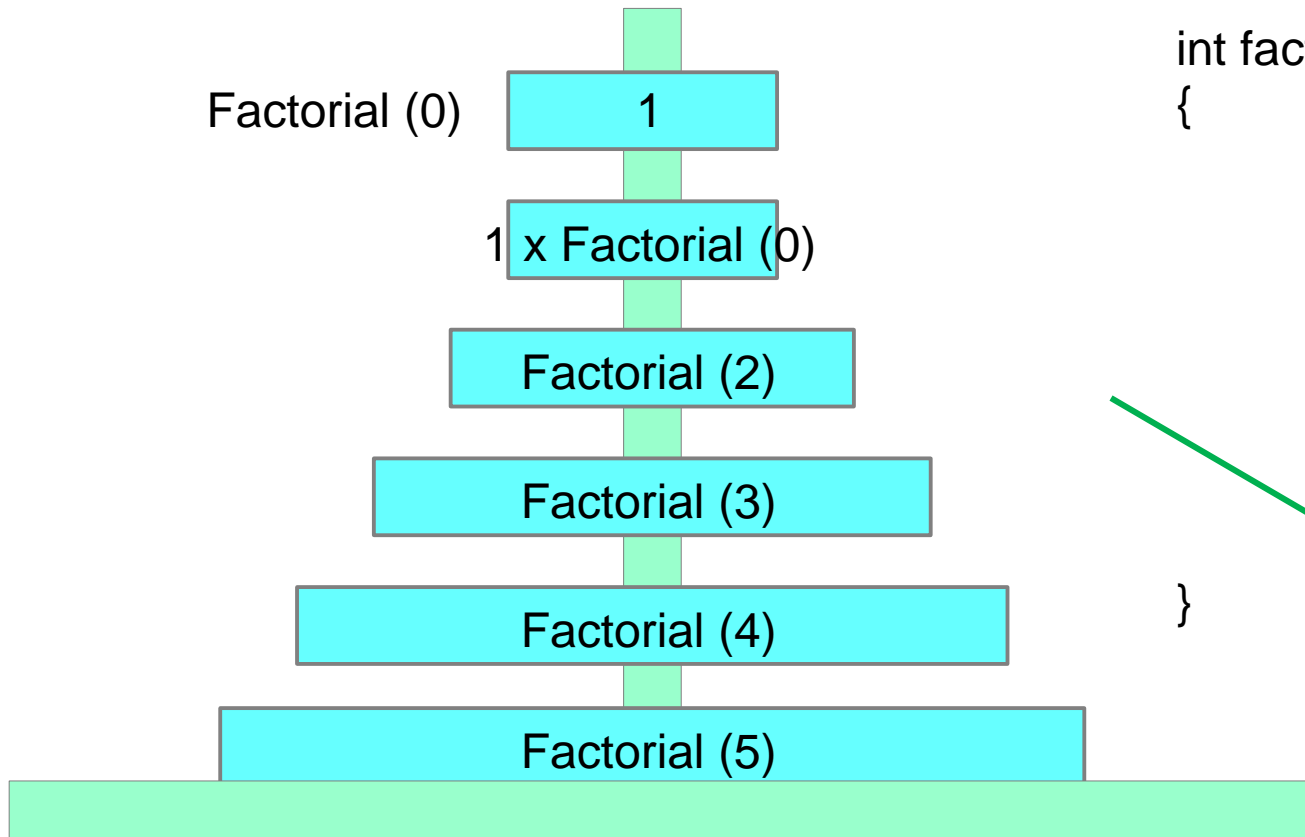
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

    return ans; 1
}
```

A green arrow points from the 'return ans;' line in the code to the 'Factorial (4)' box in the call stack diagram.

Call Stack Factorial (5)



```
int factorial(int n)
{
```

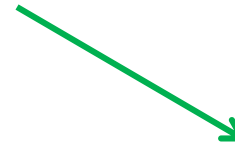
```
    int ans;
```

```
    if (n == 0)
        ans = 1;
```

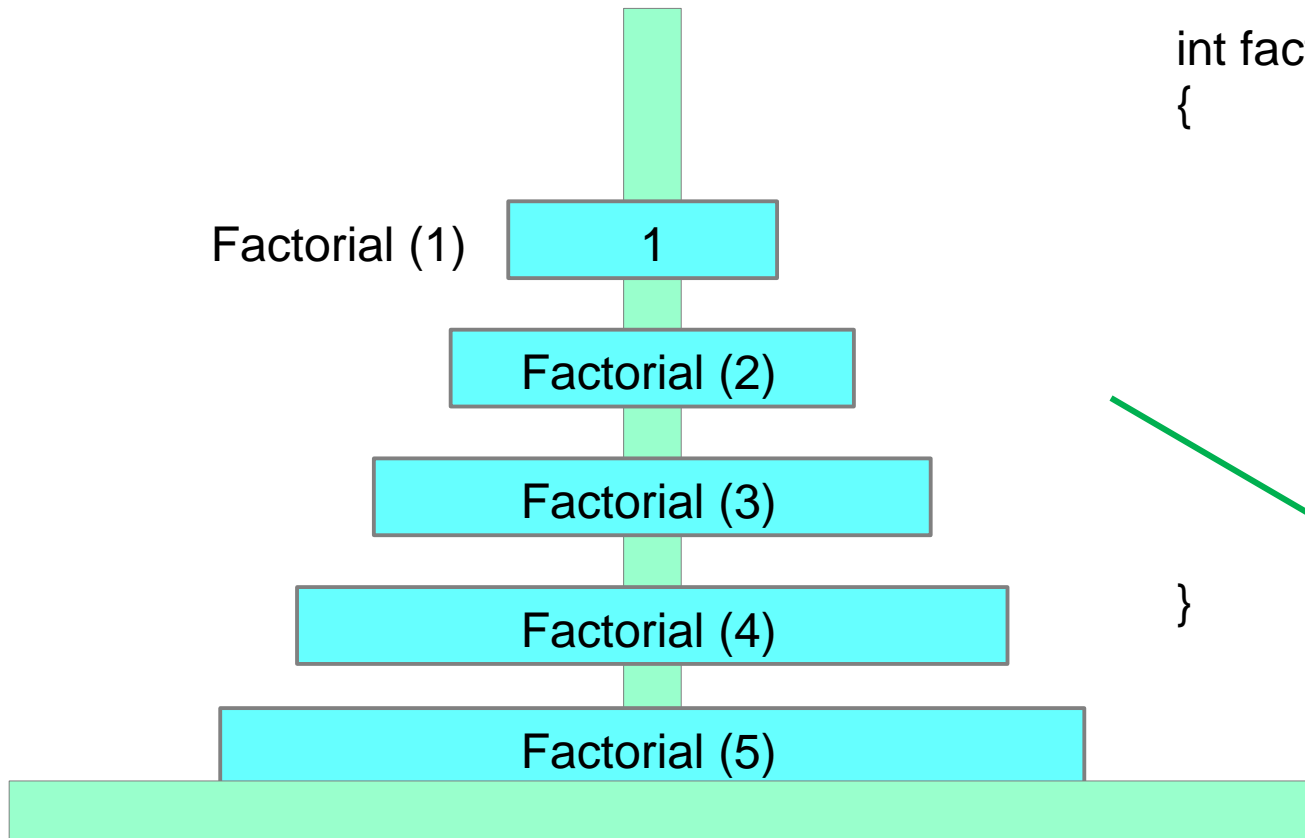
```
    else
        ans = n * factorial(n - 1);
```

```
    return ans;
```

```
}
```



Call Stack Factorial (5)



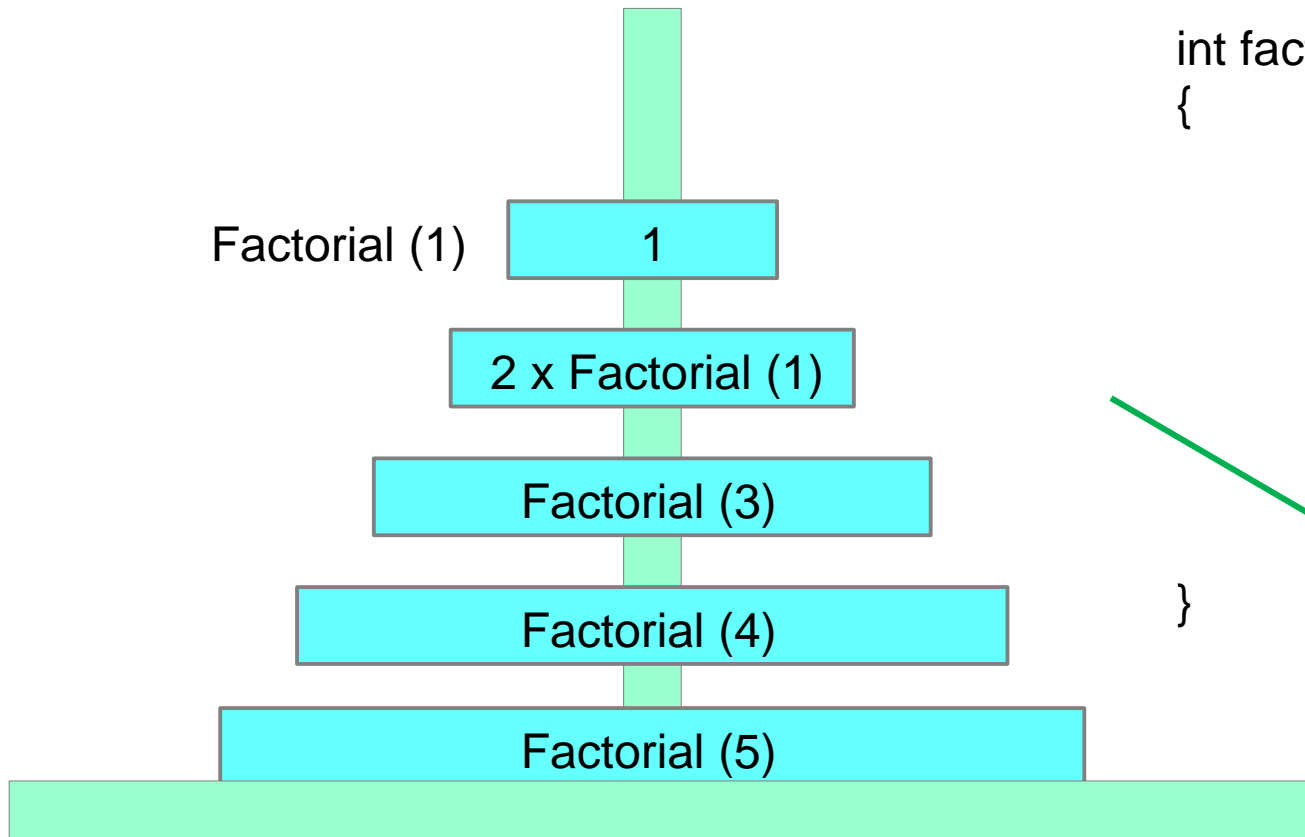
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

    return ans; 1
}
```

A green arrow points from the 'return ans;' line in the code to the 'Factorial (5)' call in the stack diagram.

Call Stack Factorial (5)



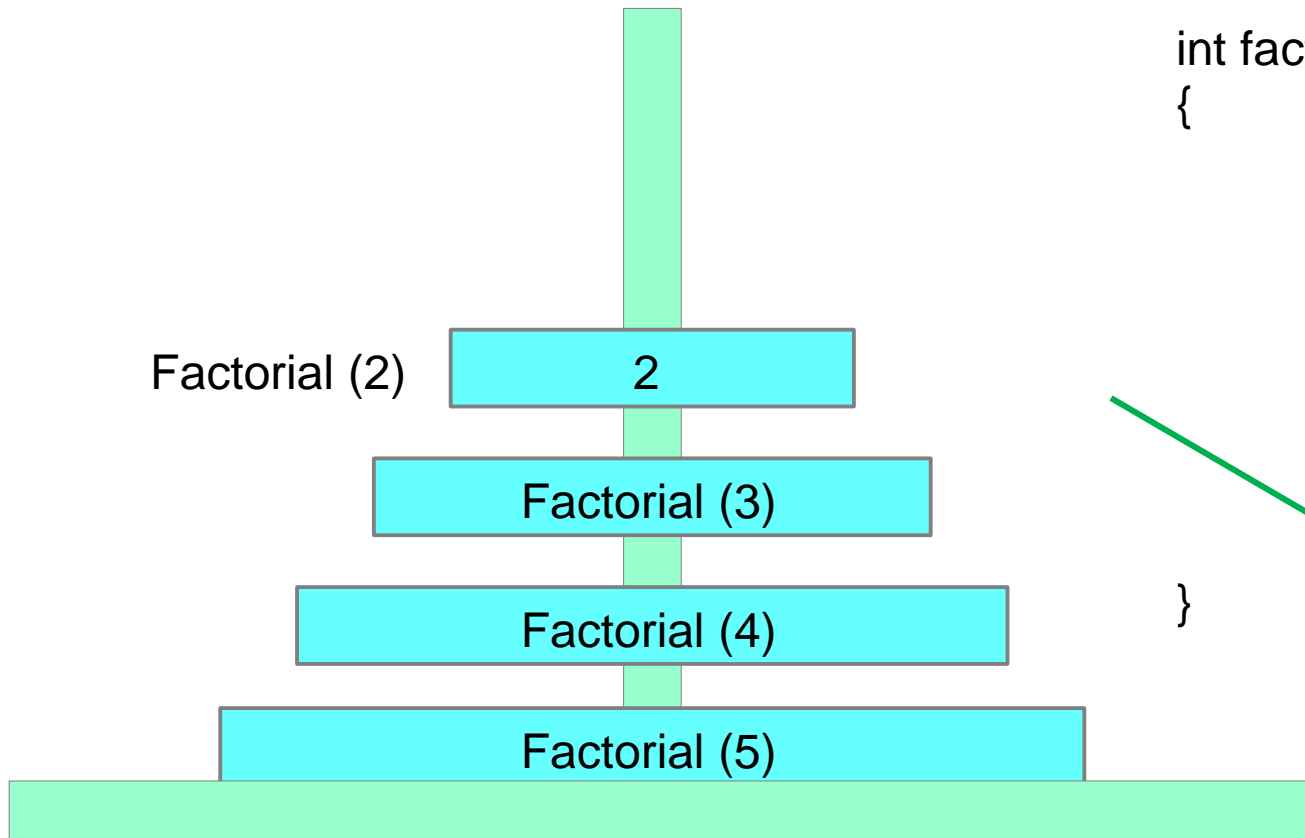
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

    return ans;
}
```

A green arrow points from the recursive call `factorial(n - 1)` in the code to the 'Factorial (4)' frame in the call stack diagram.

Call Stack Factorial (5)



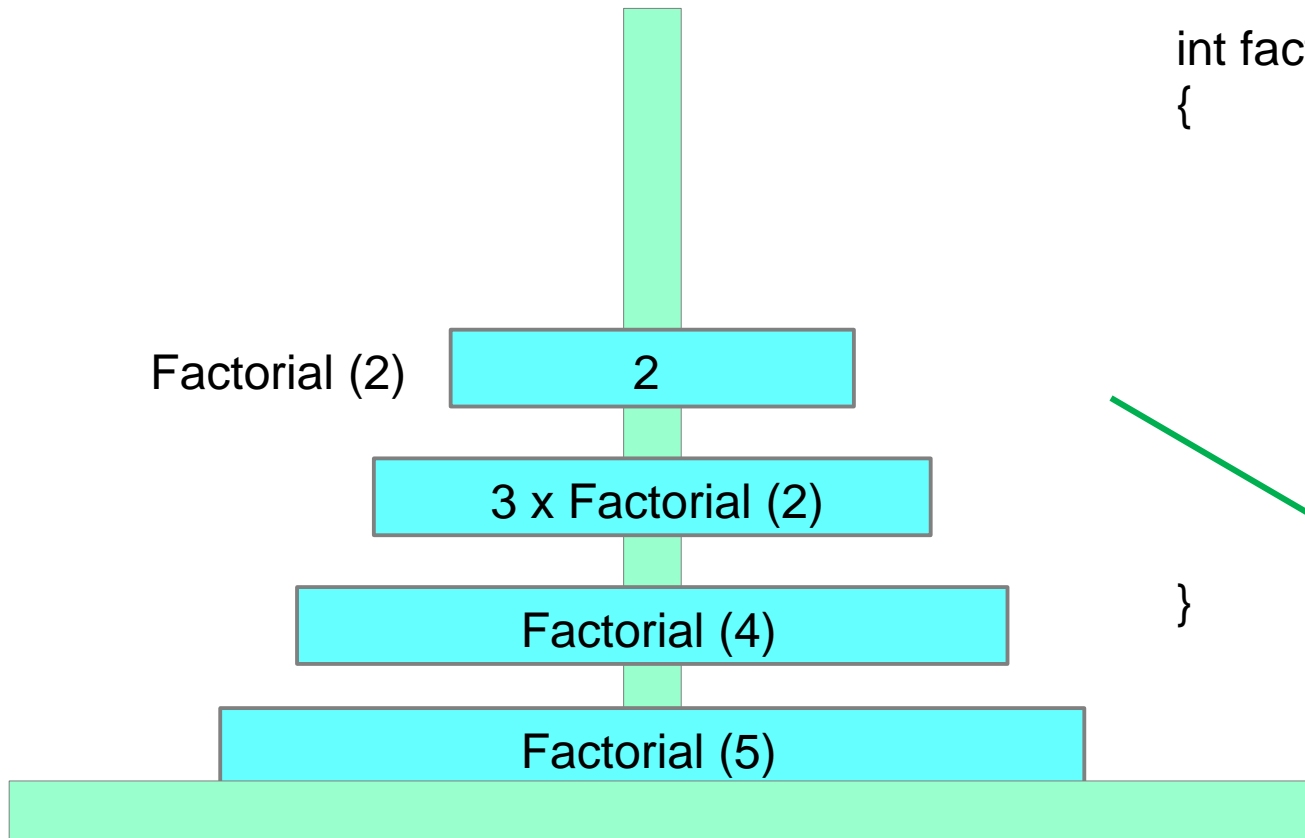
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

    return ans; 2
}
```

A green arrow points from the `return ans;` line in the code block to the `2` value in the Factorial (2) call of the call stack diagram.

Call Stack Factorial (5)

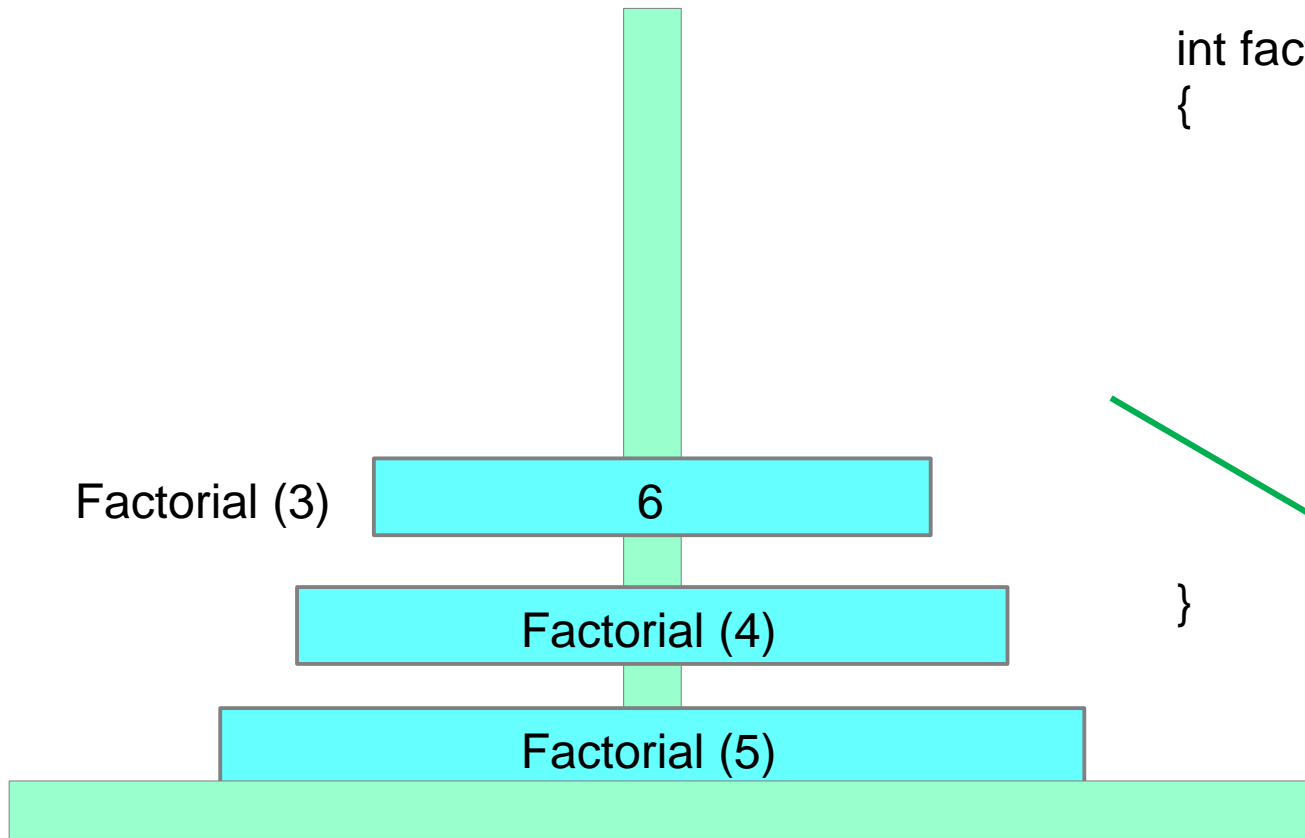


```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

    return ans;
}
```

Call Stack Factorial (5)



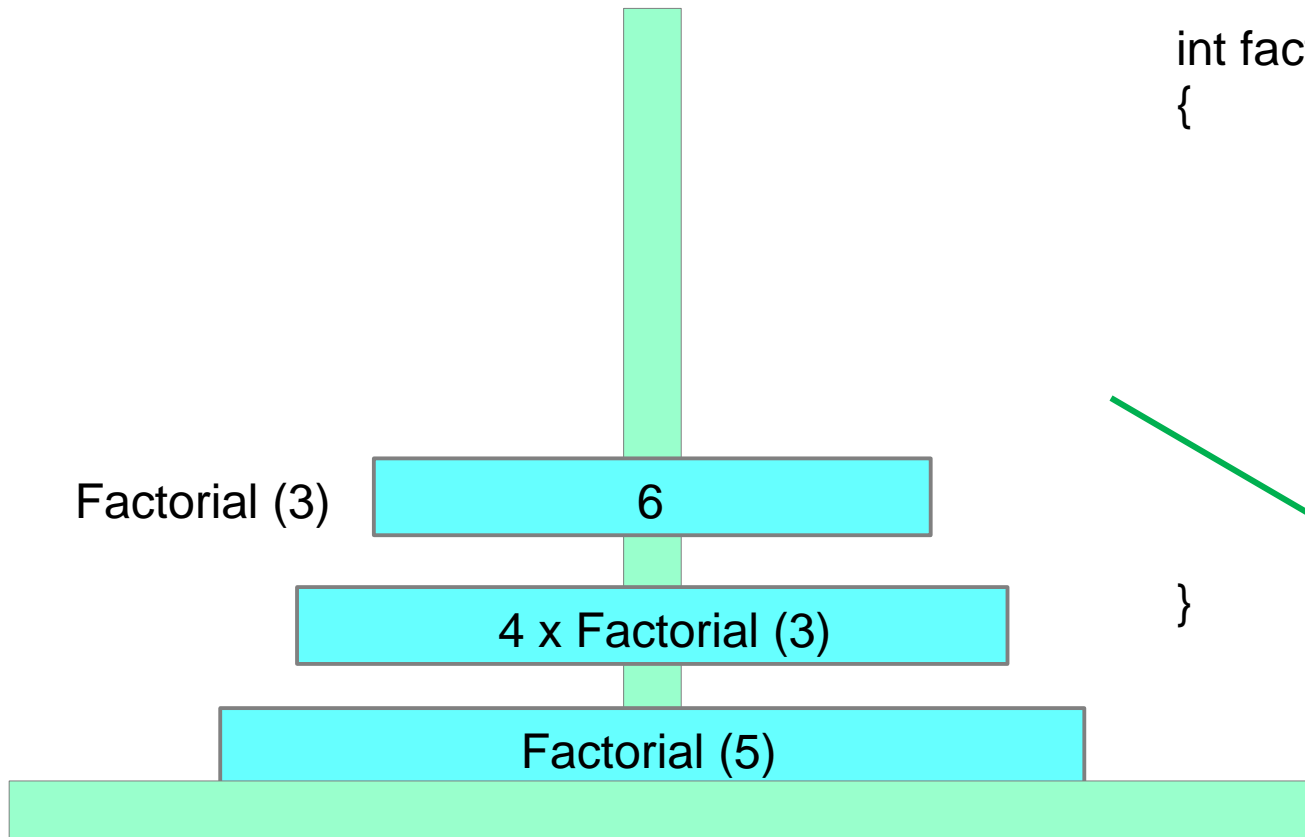
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

    return ans;
}
```

A green arrow points from the closing curly brace of the `factorial` function to the `return ans;` statement, which is followed by a yellow box containing the number 6.

Call Stack Factorial (5)



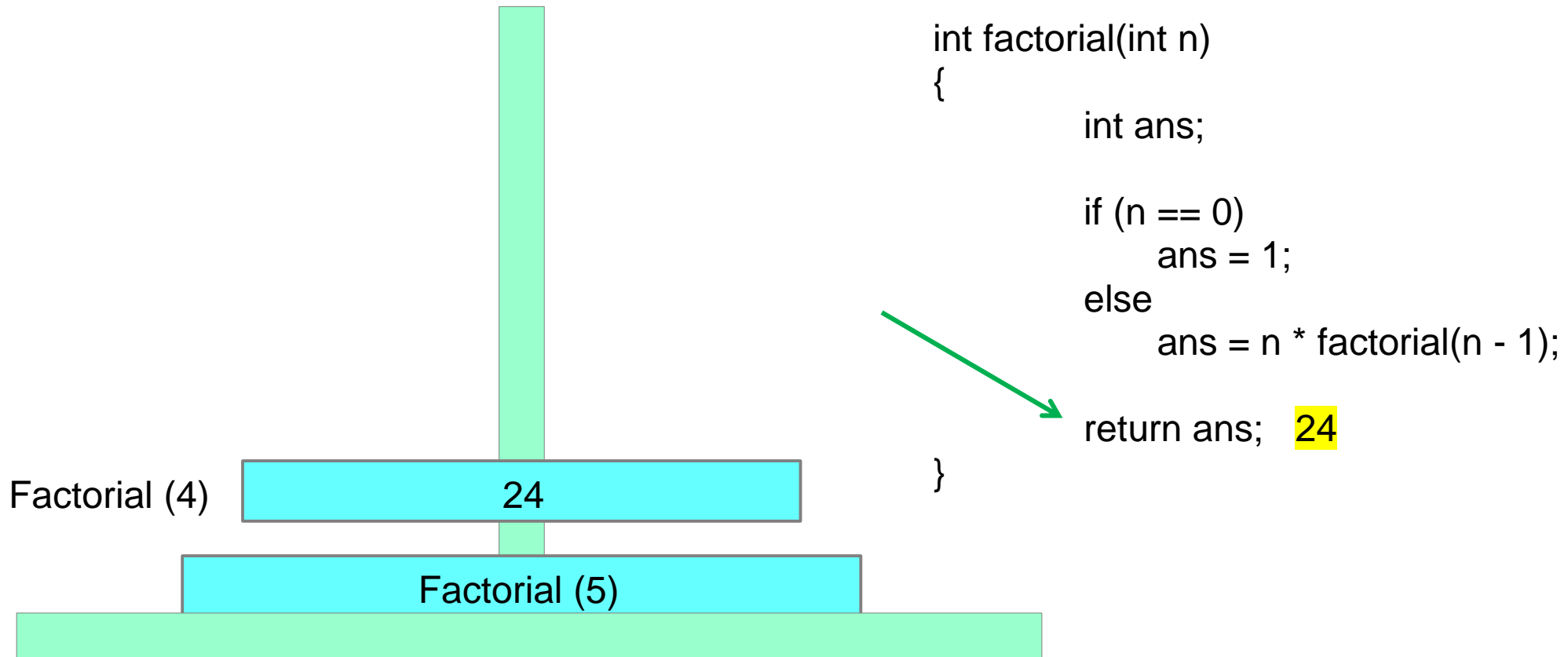
```
int factorial(int n)
{
    int ans;

    if (n == 0)
        ans = 1;
    else
        ans = n * factorial(n - 1);

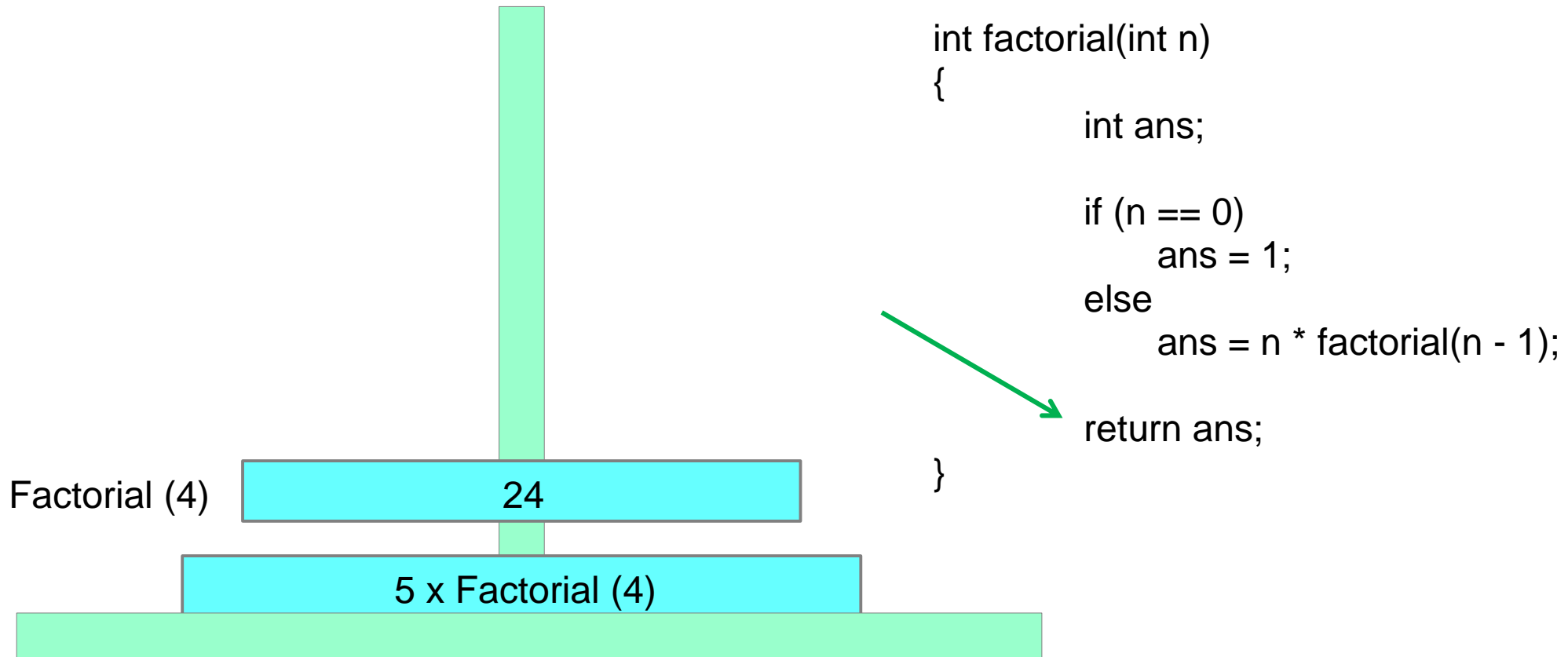
    return ans;
}
```

A green arrow points from the recursive call `factorial(n - 1)` in the code to the `Factorial (3)` call in the call stack diagram.

Call Stack Factorial (5)



Call Stack Factorial (5)



Call Stack Factorial (5)

