

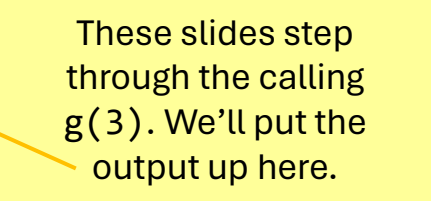
Tracing a Double-recursive Function

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

Output

nothing yet

These slides step through the calling `g(3)`. We'll put the output up here.



```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

Output

nothing yet

g(3)

First call is at the top of the tree, called the **root** of the tree.

```
void g(int n)
{
    if (n > 0)                // line 1
    {
        cout << n << " ";    // line 2
        g(n - 1);             // line 3
        g(n - 1);             // line 4
                                // line 5
    }
}
```

Output
3

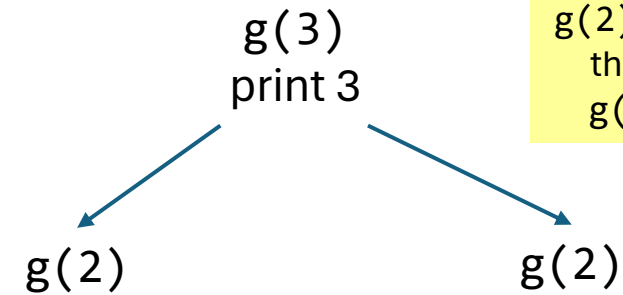
g(3)
print 3

First call is at the top of the tree, called the **root** of the tree.

g(n) always first prints n, if n > 0

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

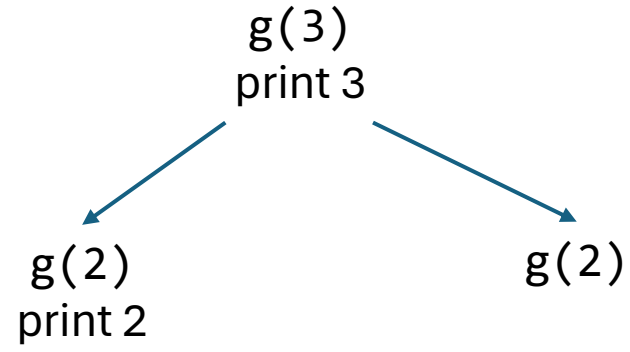
Output
3



After printing 3, it calls
g(2) on **line 3** of g(3), and
then afterwards it calls
g(2) on **line 4** of g(3).

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

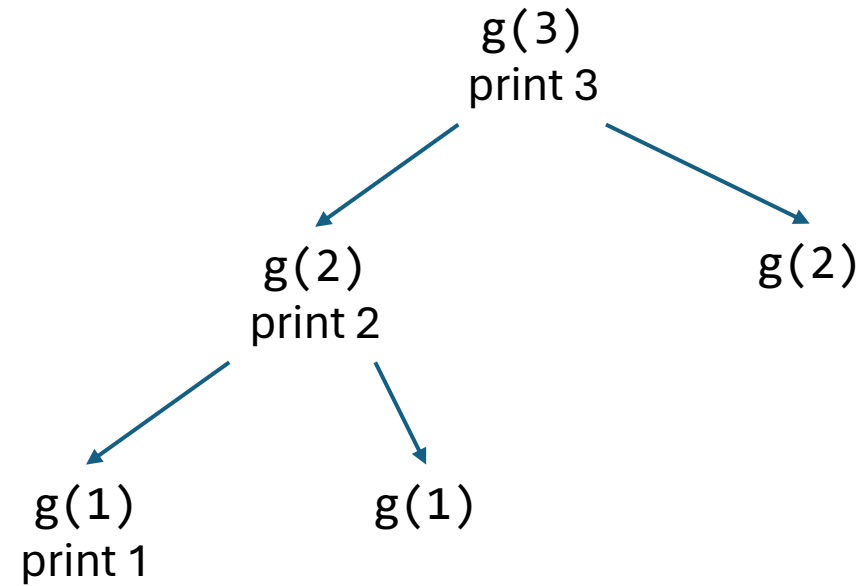
Output
3 2



The left-most function of the tree is always called first. Here it prints 2.

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}
```

Output
3 2 1

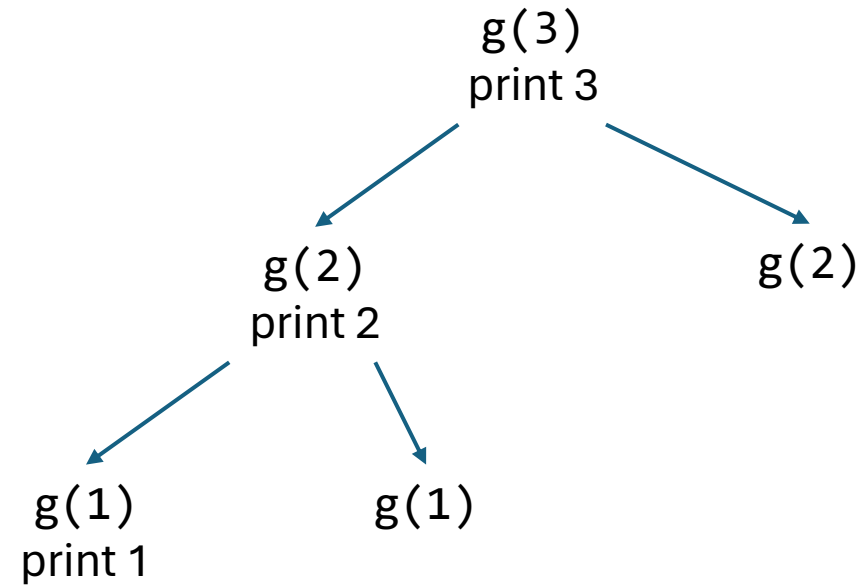


The left-most function of the tree is always called first. Here g(1) prints 1.

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

Output

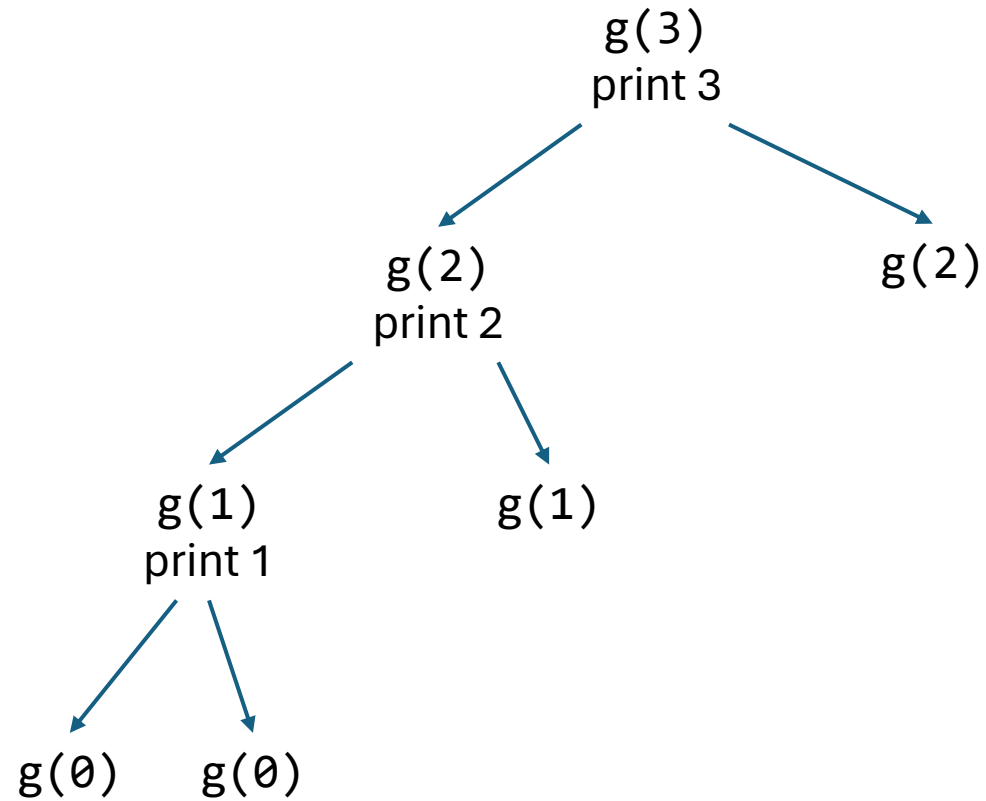
3 2 1



Then `g(1)` from line 3 of `g(2)` is called, and afterwards it will call `g(1)` from line 4 of `g(2)`.


```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}
```

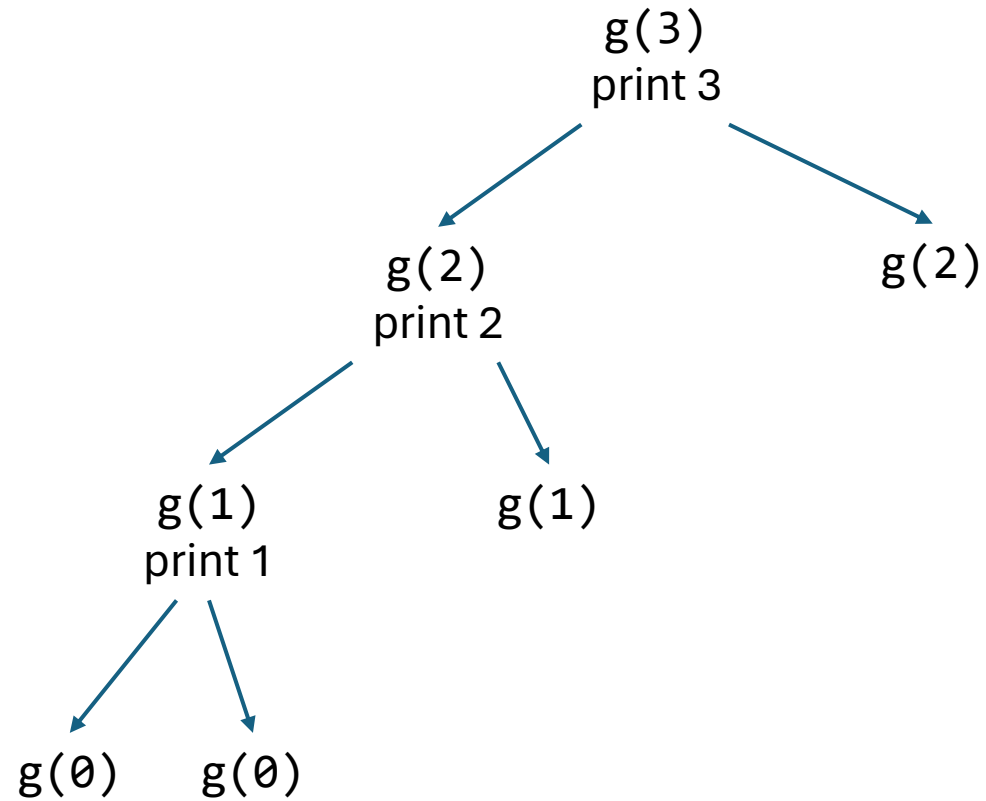
Output
3 2 1



Now `g(1)` calls `g(0)` on line 3 of `g(1)`, and afterwards calls `g(0)` on line 4 of `g(1)`.

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}
```

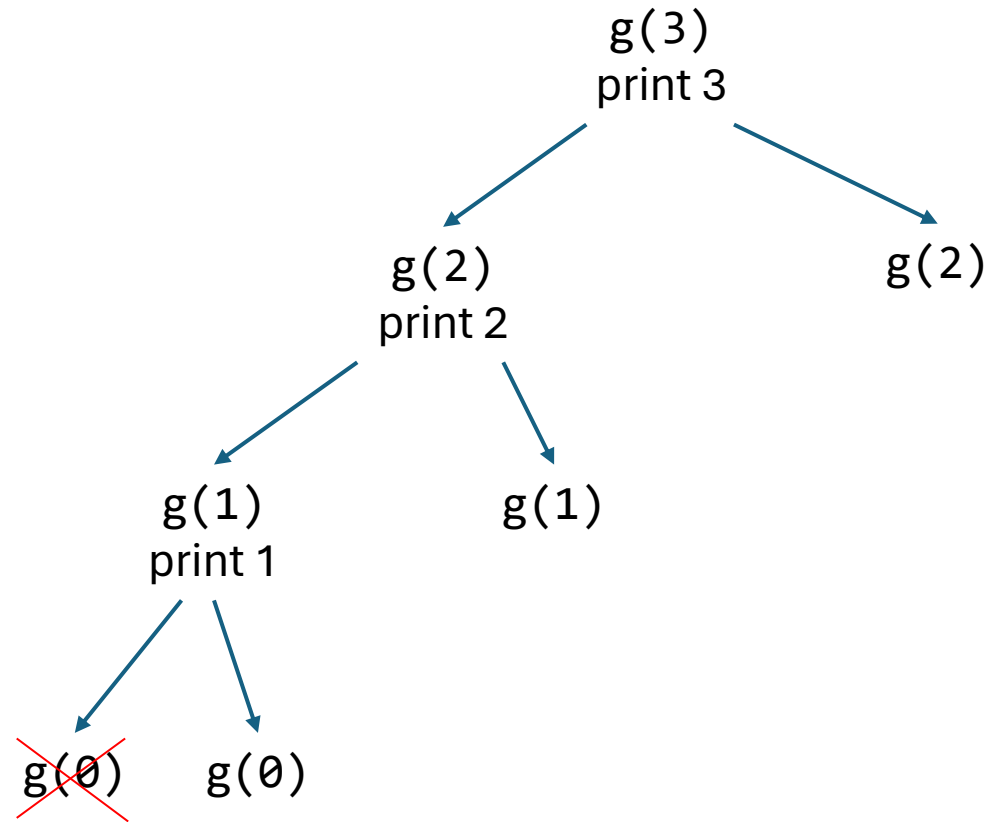
Output
3 2 1



`g(0)` does **not** print anything because of the if-statement on line 1 of `g(n)`.

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

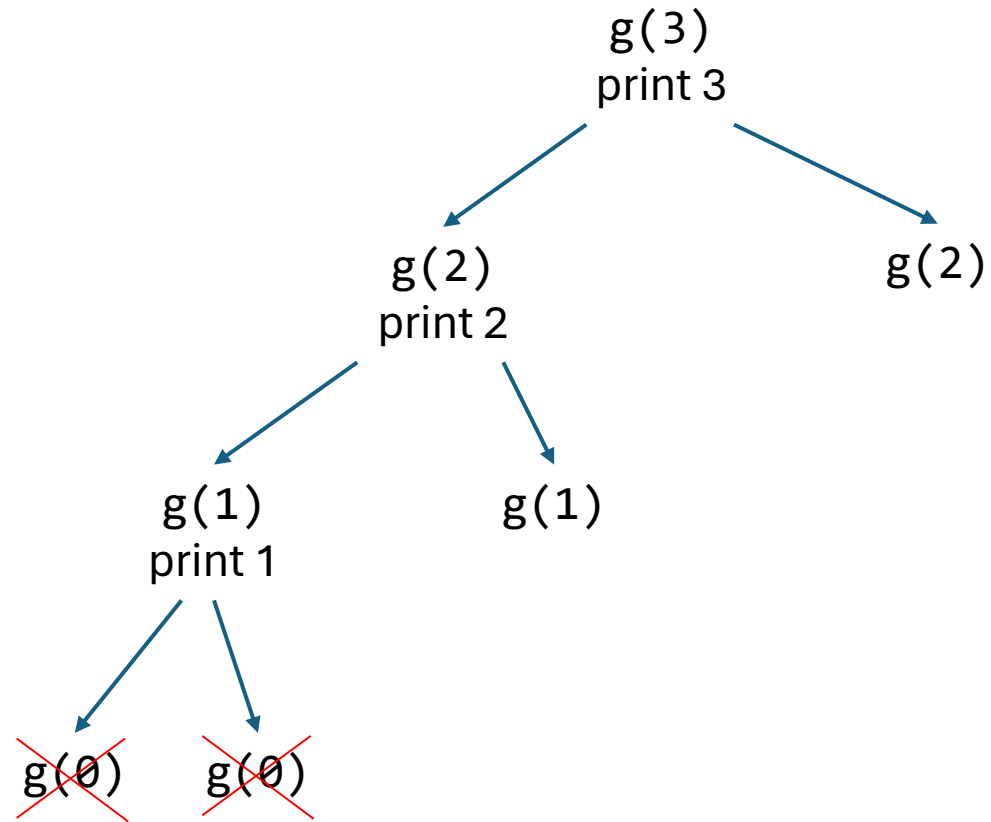
Output
3 2 1



The left `g(0)` ends without printing anything, and the right `g(0)` is called.

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

Output
3 2 1

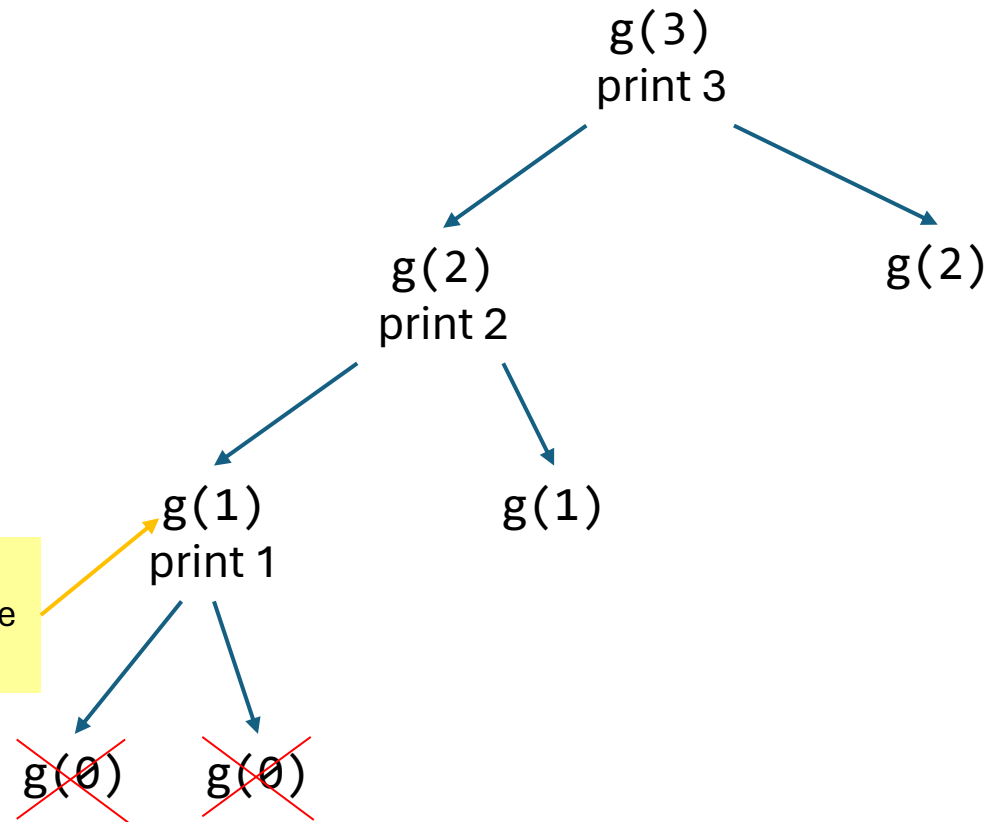


The right `g(0)`
ends without
printing anything.

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}
```

Output
3 2 1

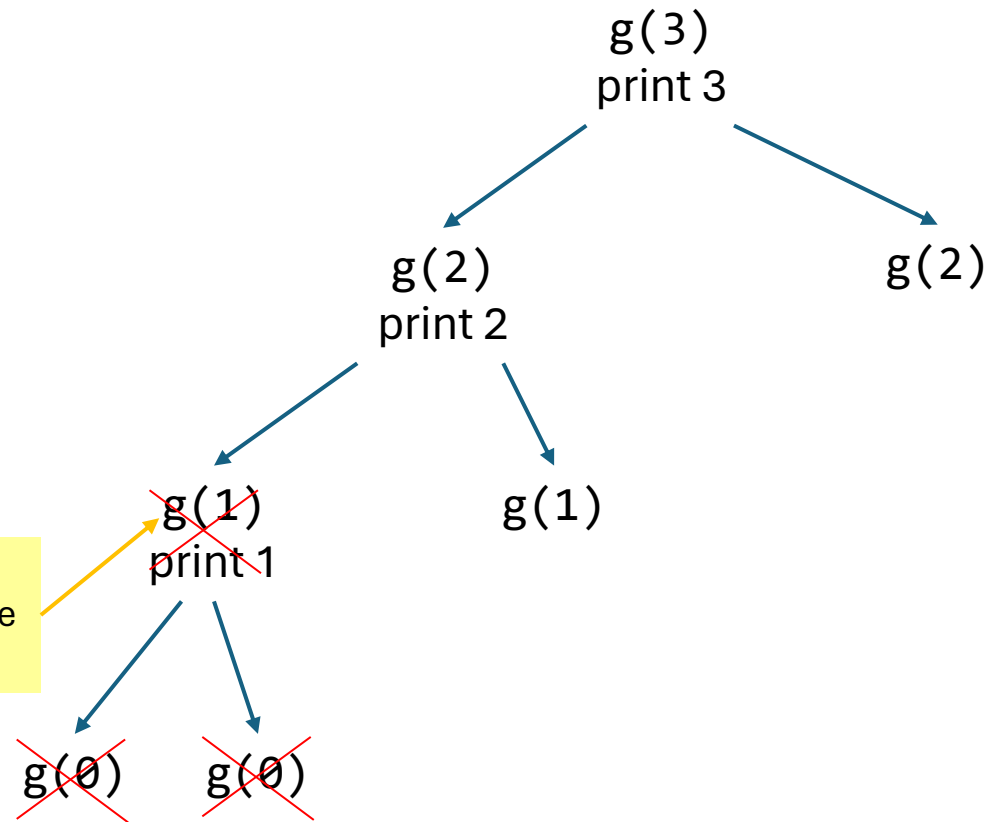
Now the call to
this g(1) is done
and it returns.



```
void g(int n)
{
    if (n > 0)                // line 1
    {
        cout << n << " ";    // line 2
        g(n - 1);             // line 3
        g(n - 1);             // line 4
                                // line 5
    }
}
```

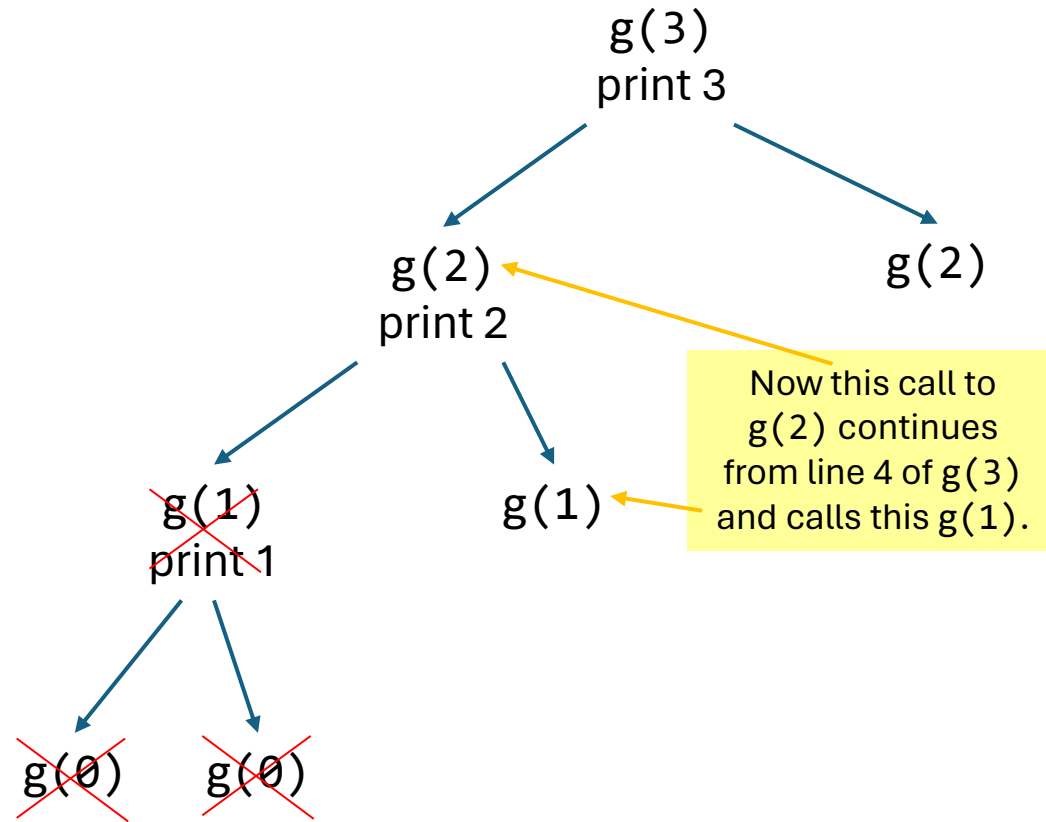
Output

3 2 1



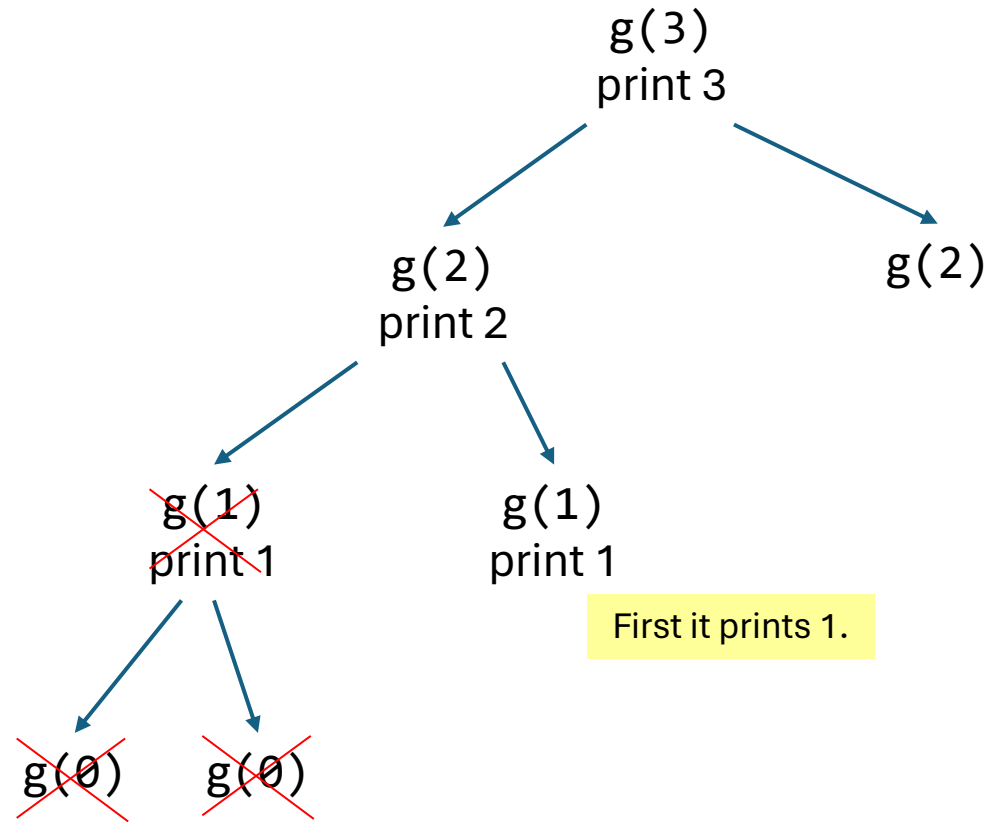
```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

Output
3 2 1



```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}
```

Output
3 2 1 1

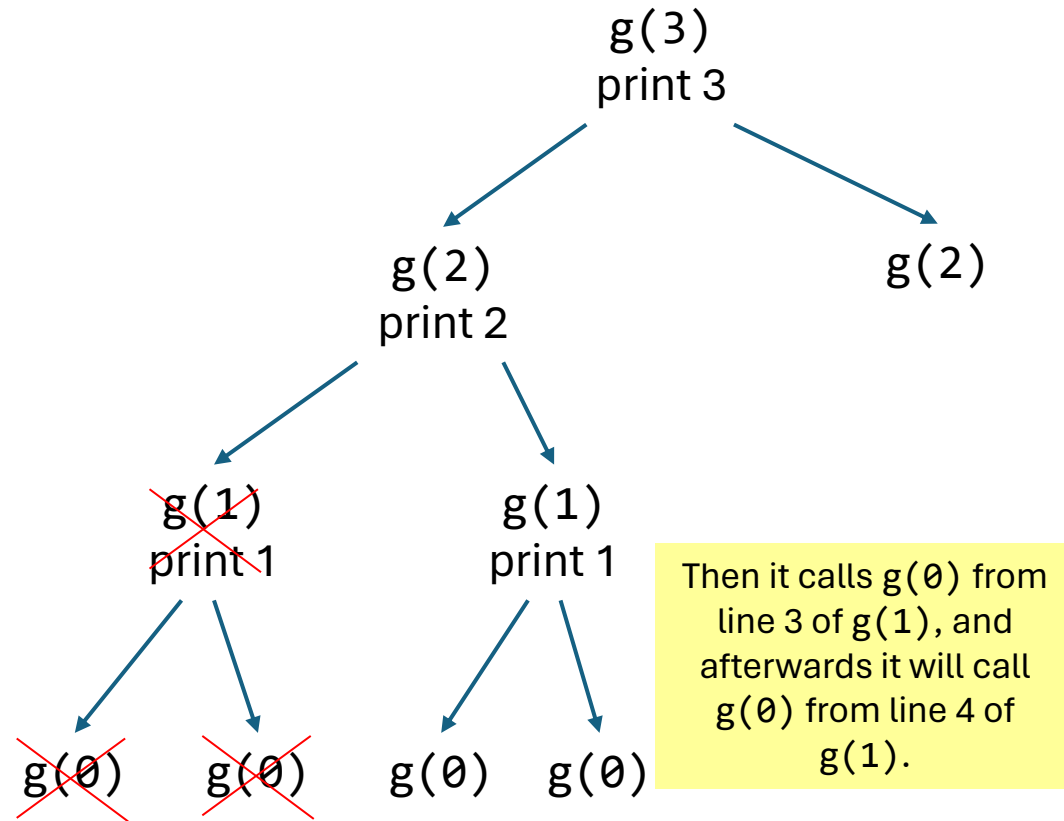



```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

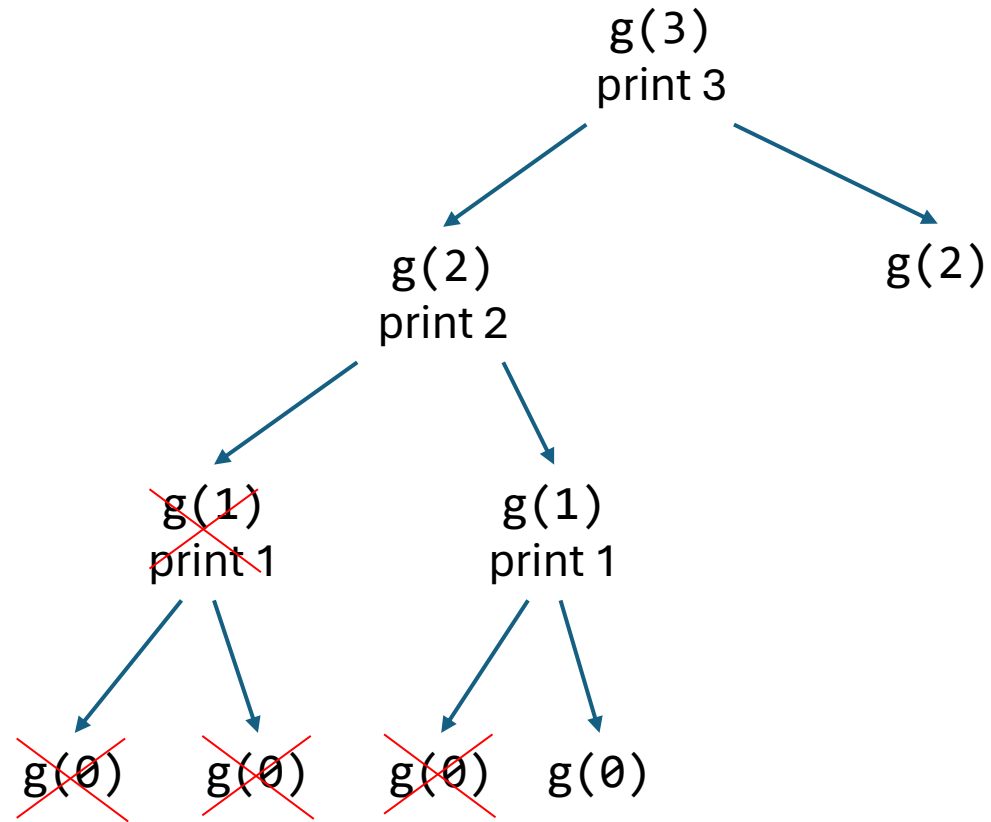
```

Output
3 2 1 1



```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}
```

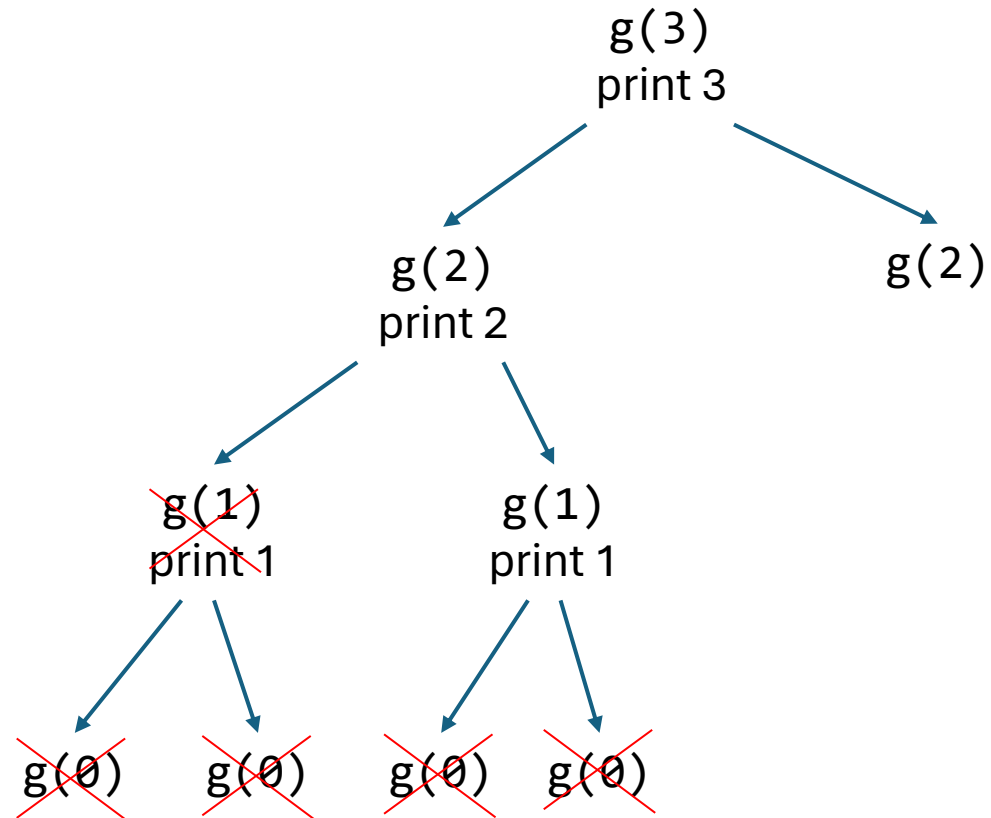
Output
3 2 1 1



Calling the left
`g(0)` does nothing.

```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}
```

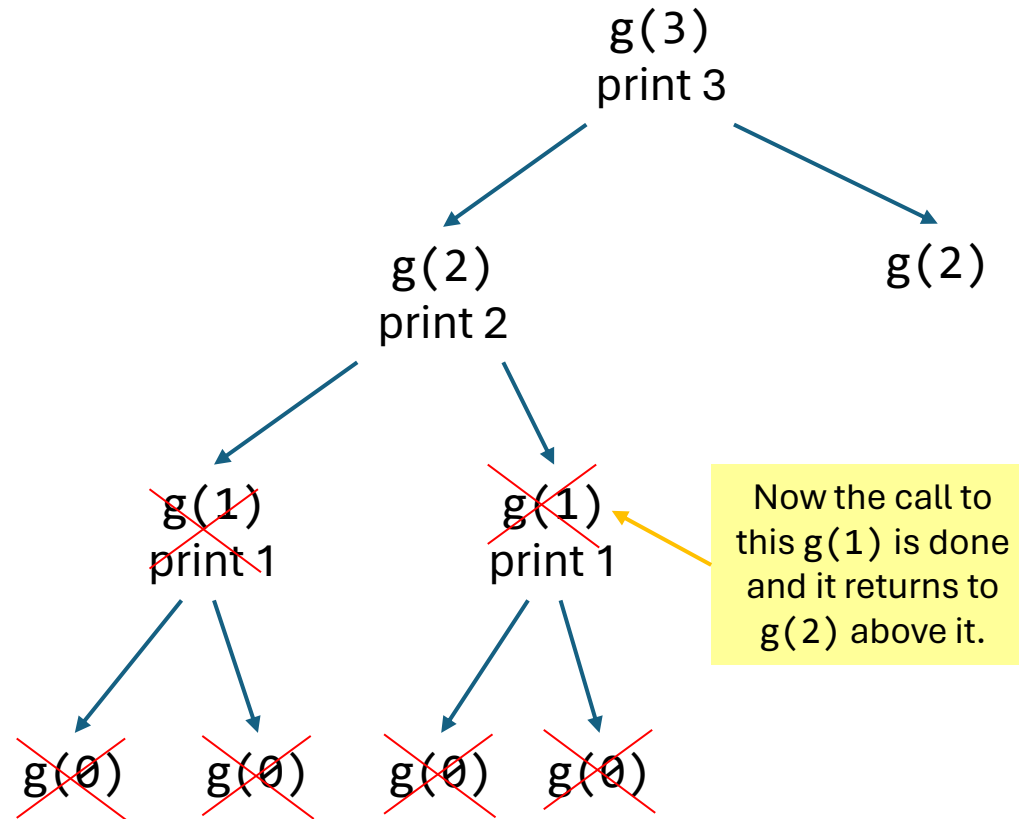
Output
3 2 1 1



Then the `g(0)` on the right is called, and does nothing.

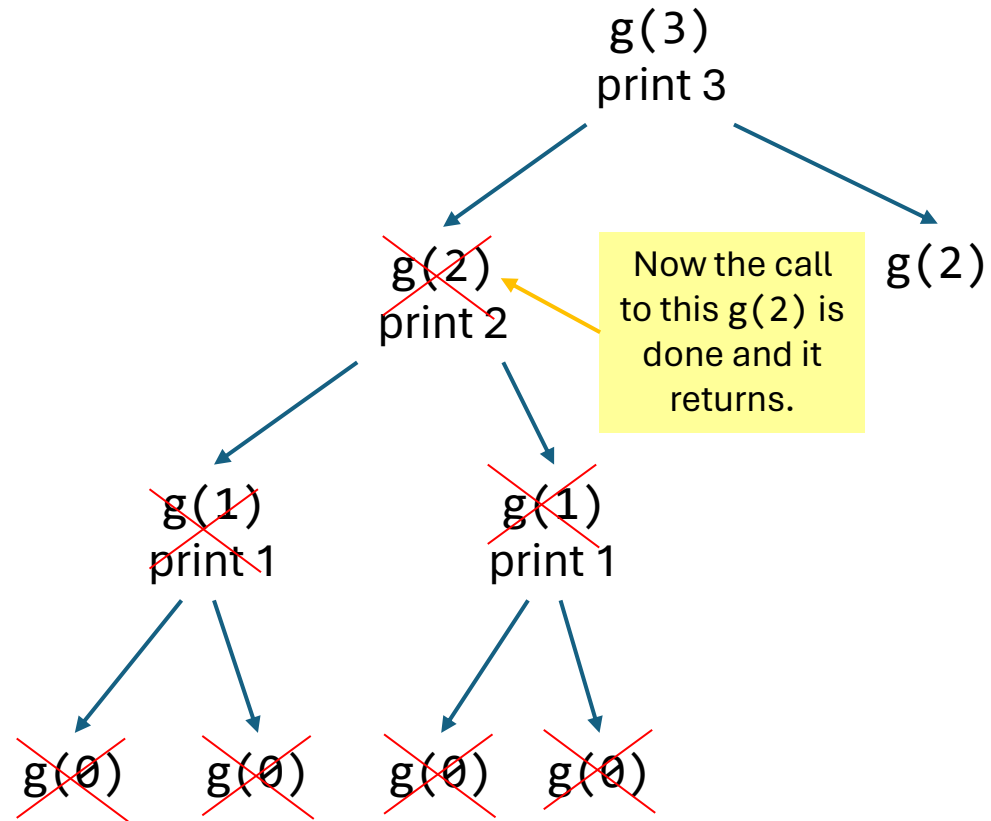
```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

Output
3 2 1 1



```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}
```

Output
3 2 1 1

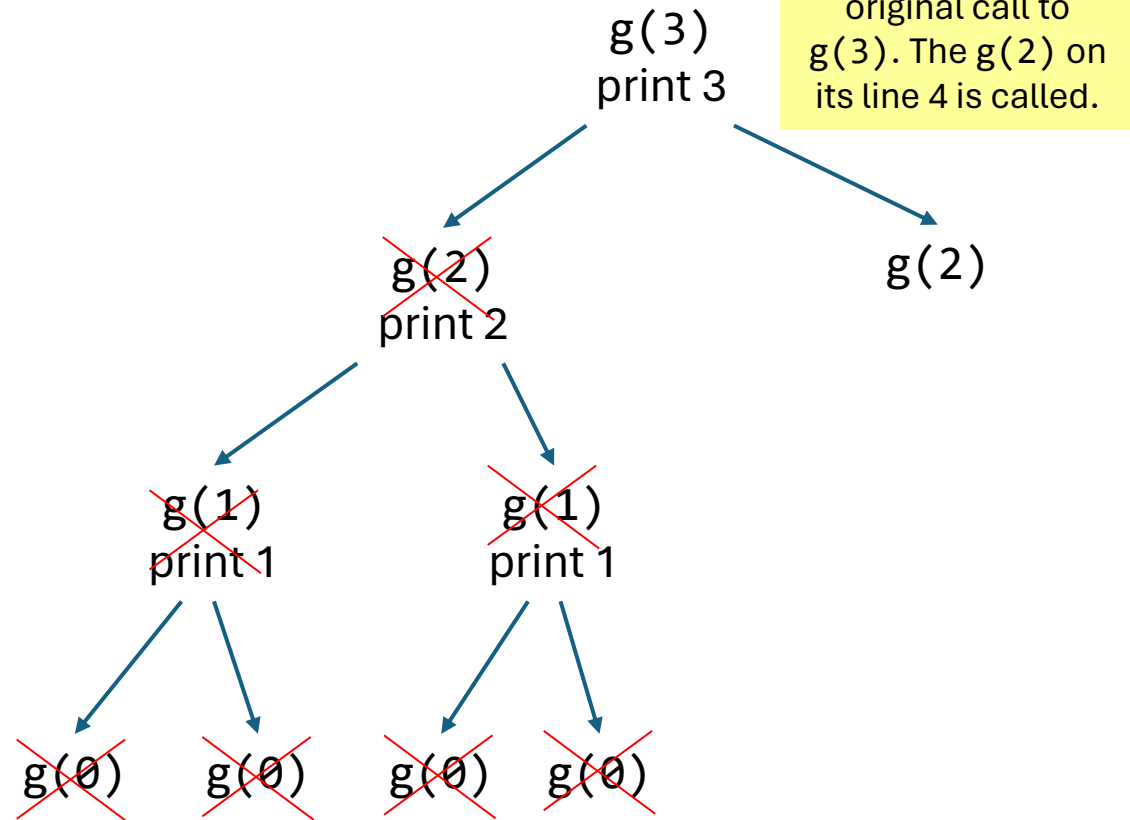


```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1

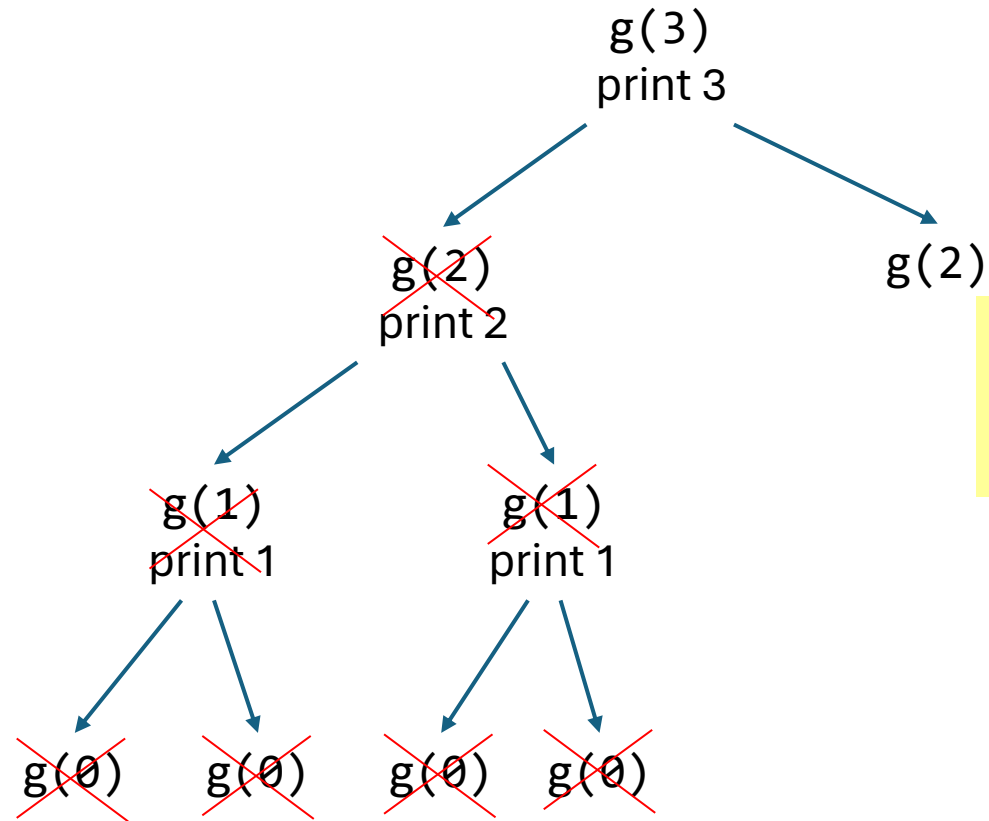


```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1



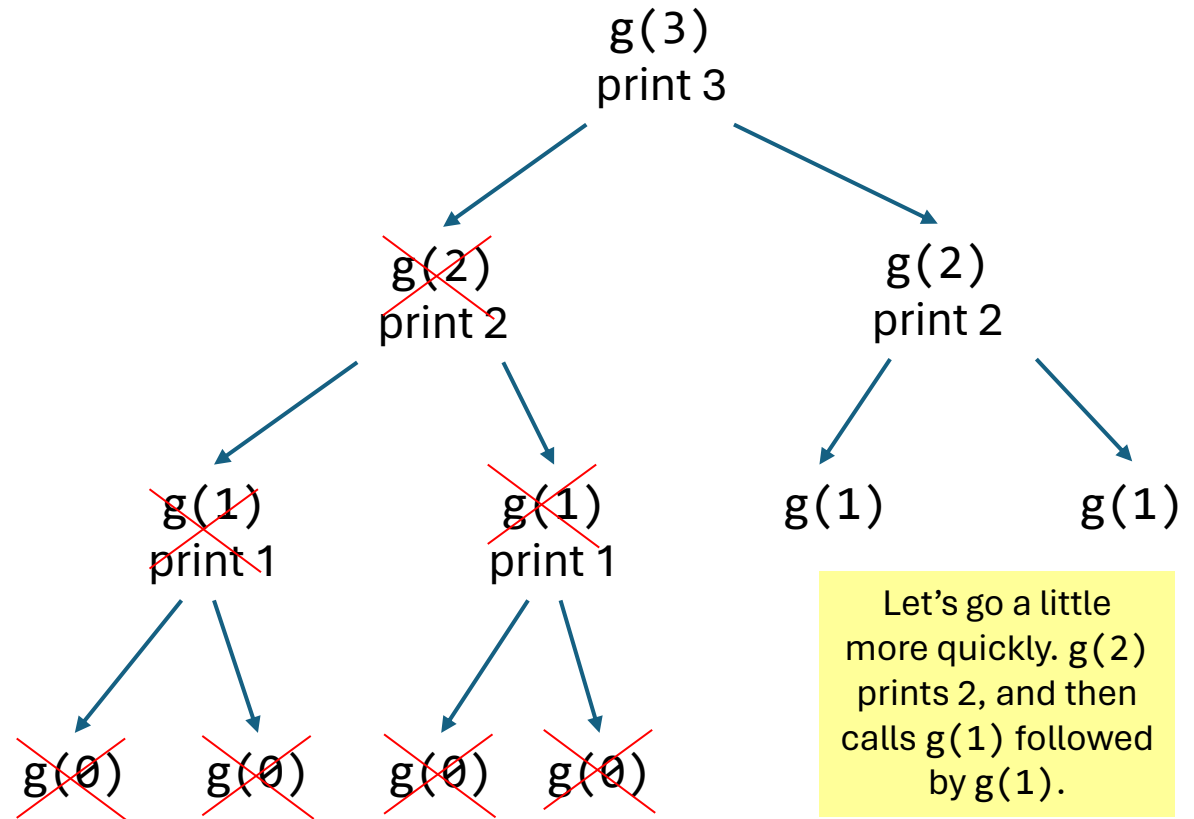
We have to call `g(2)` again. We're going to get the same tree as on the left.

```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1 2

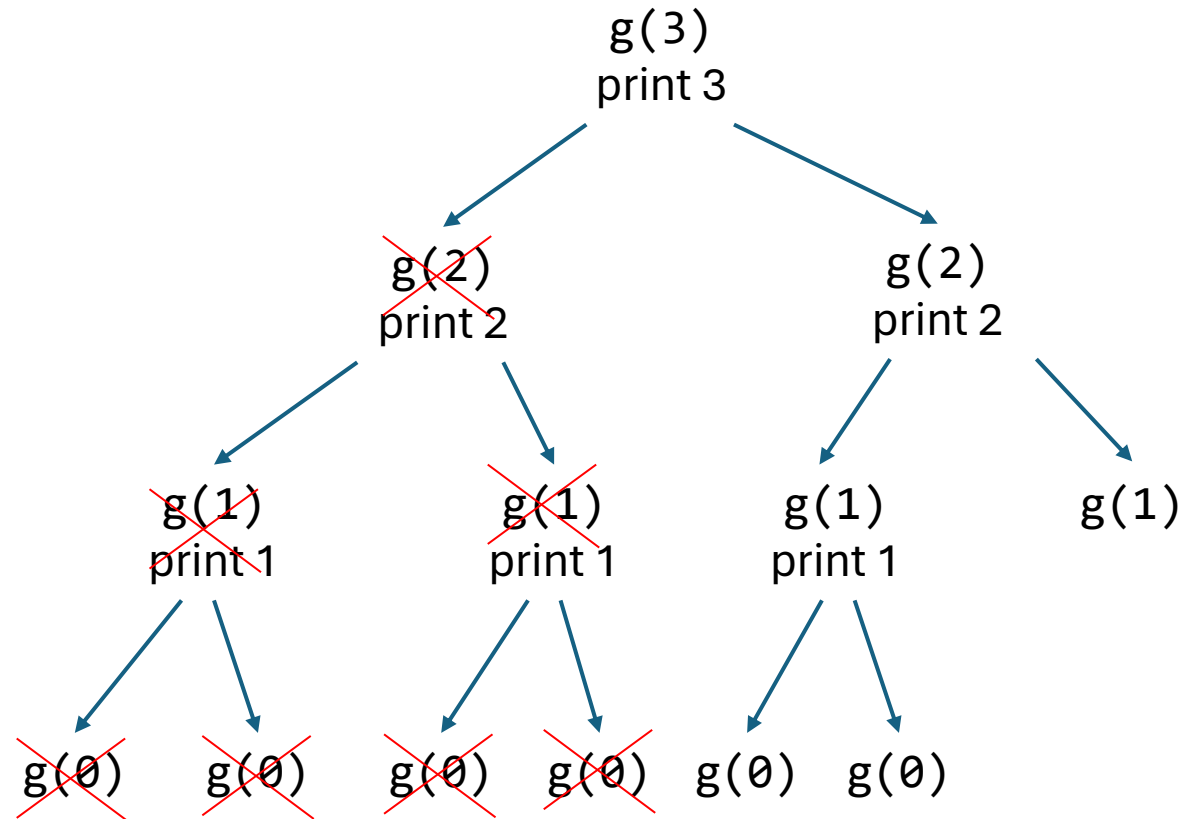



```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1 2 1



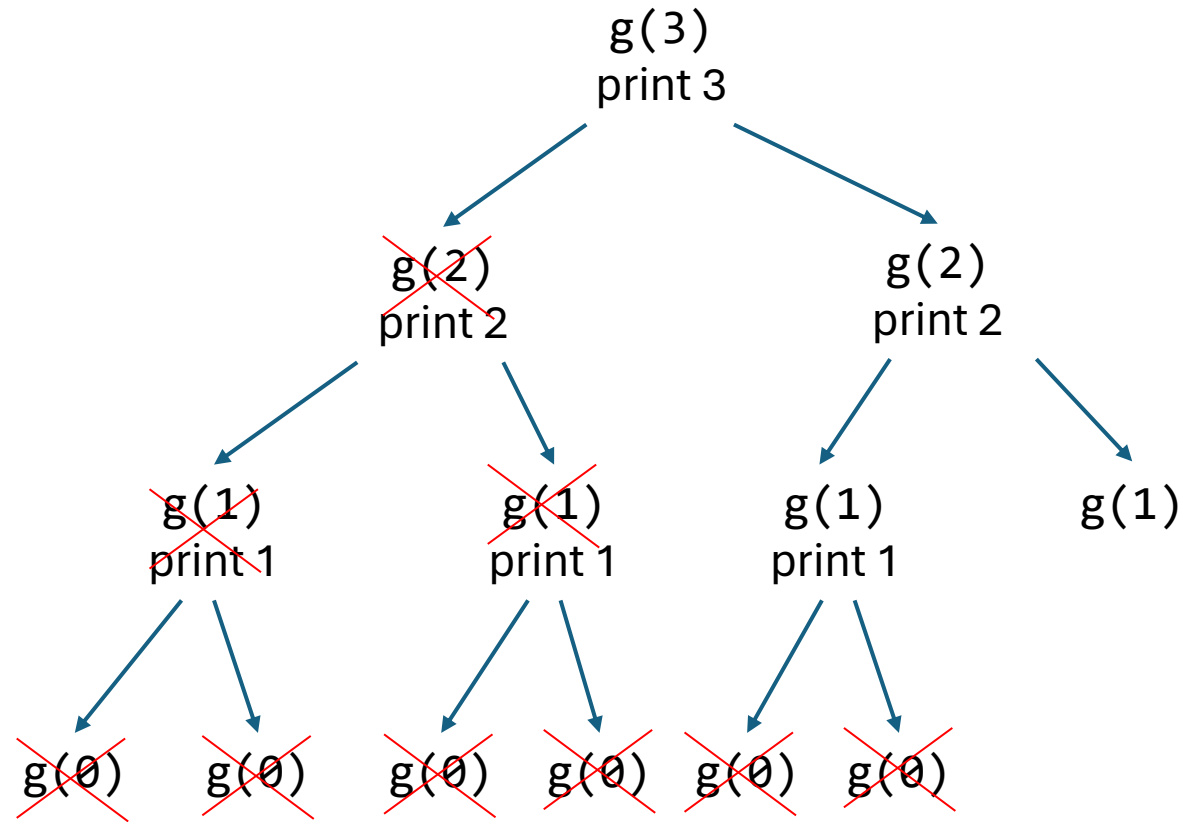
The g(1) on the left is called first. It prints 1 and then calls g(0) followed by g(0).

```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1 2 1



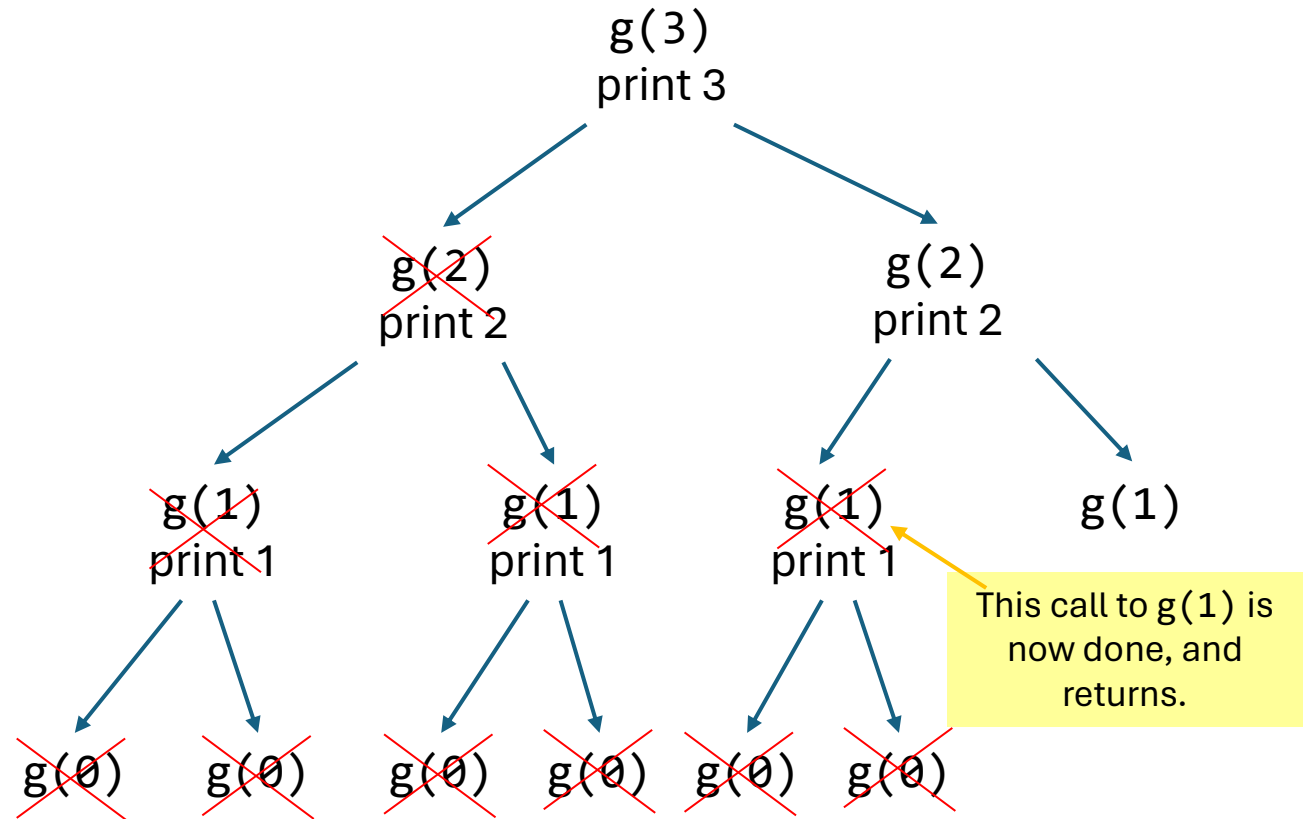
The calls to g(0)
both do nothing.

```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1 2 1

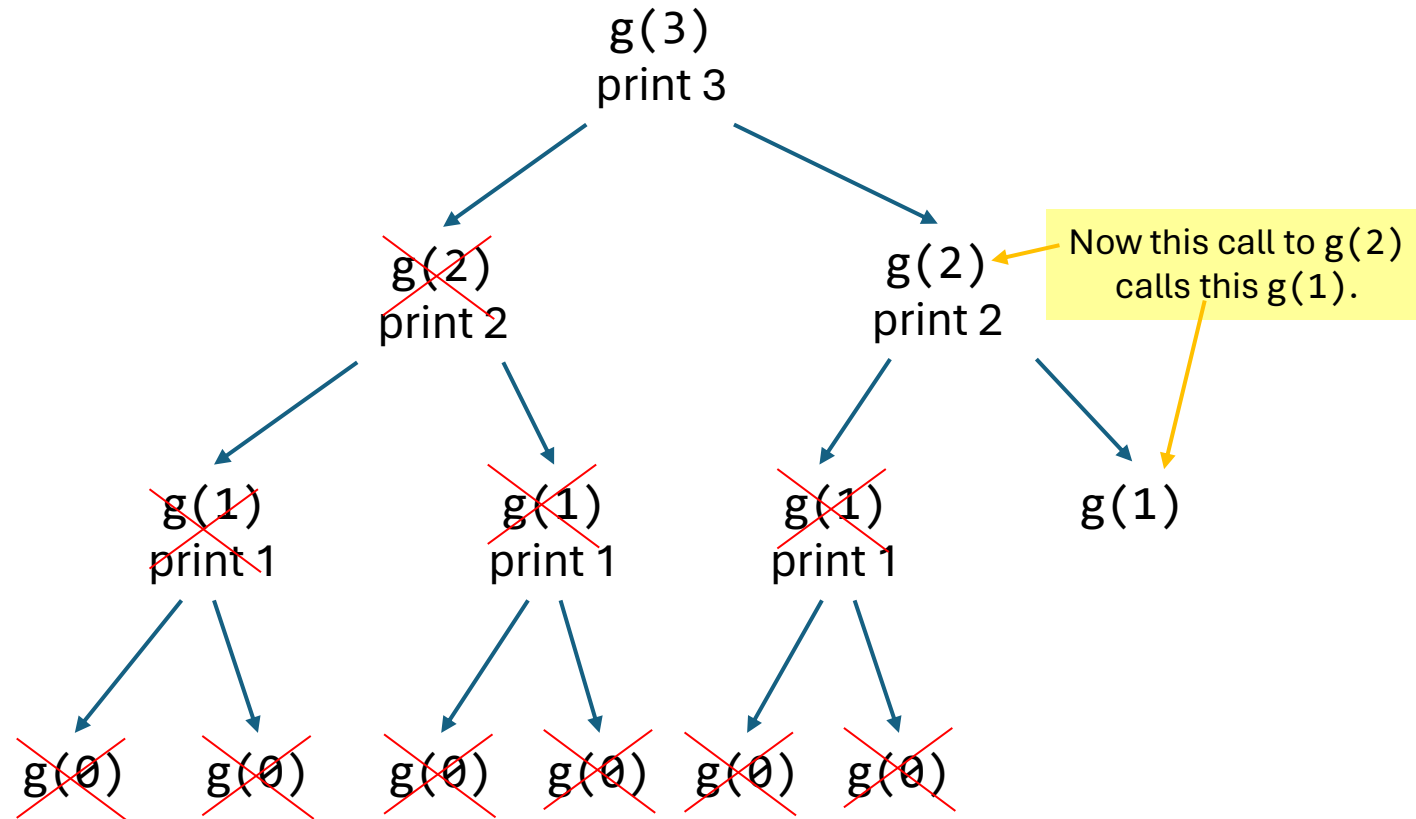


```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                          // line 5
    }
}

```

Output
3 2 1 1 2 1

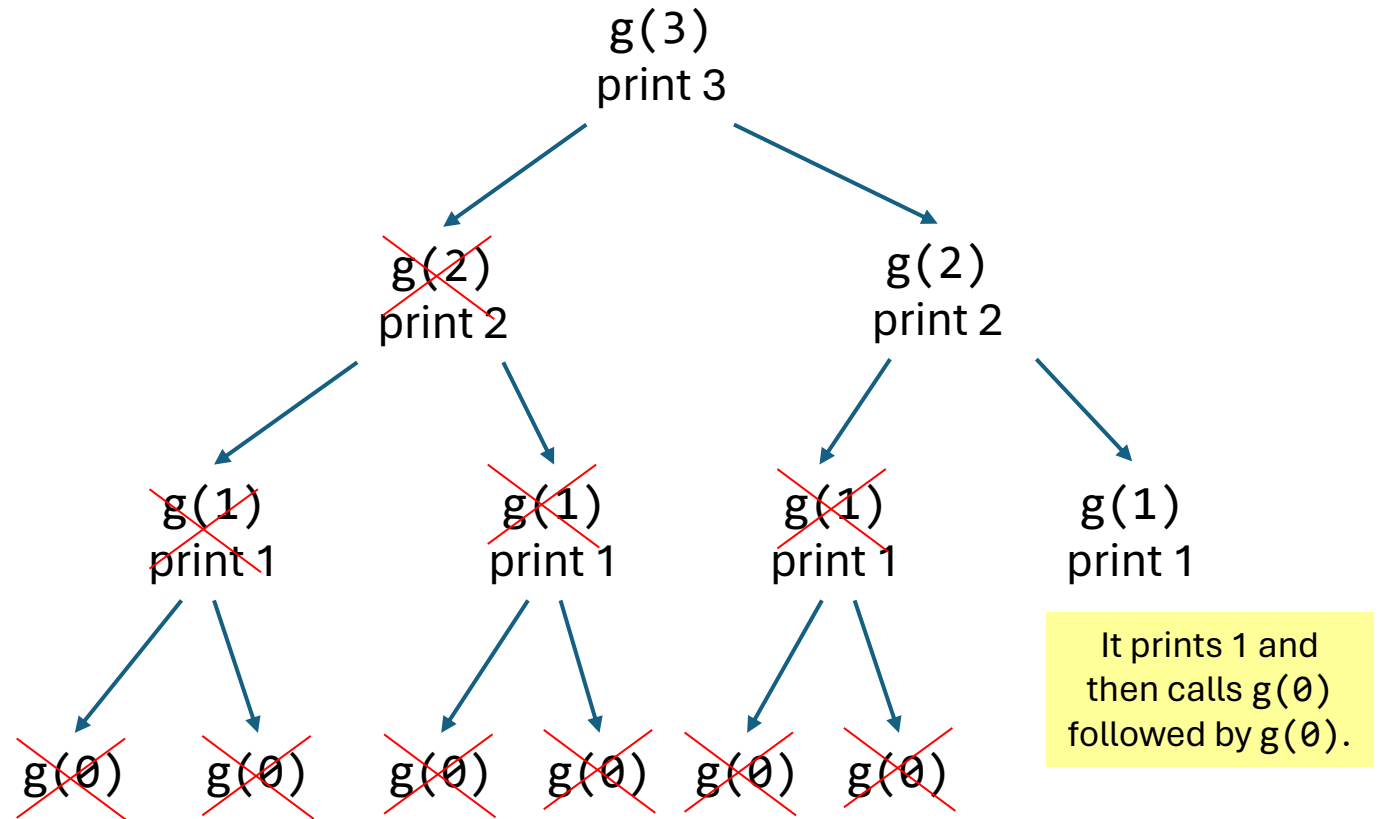


```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1 2 1 1

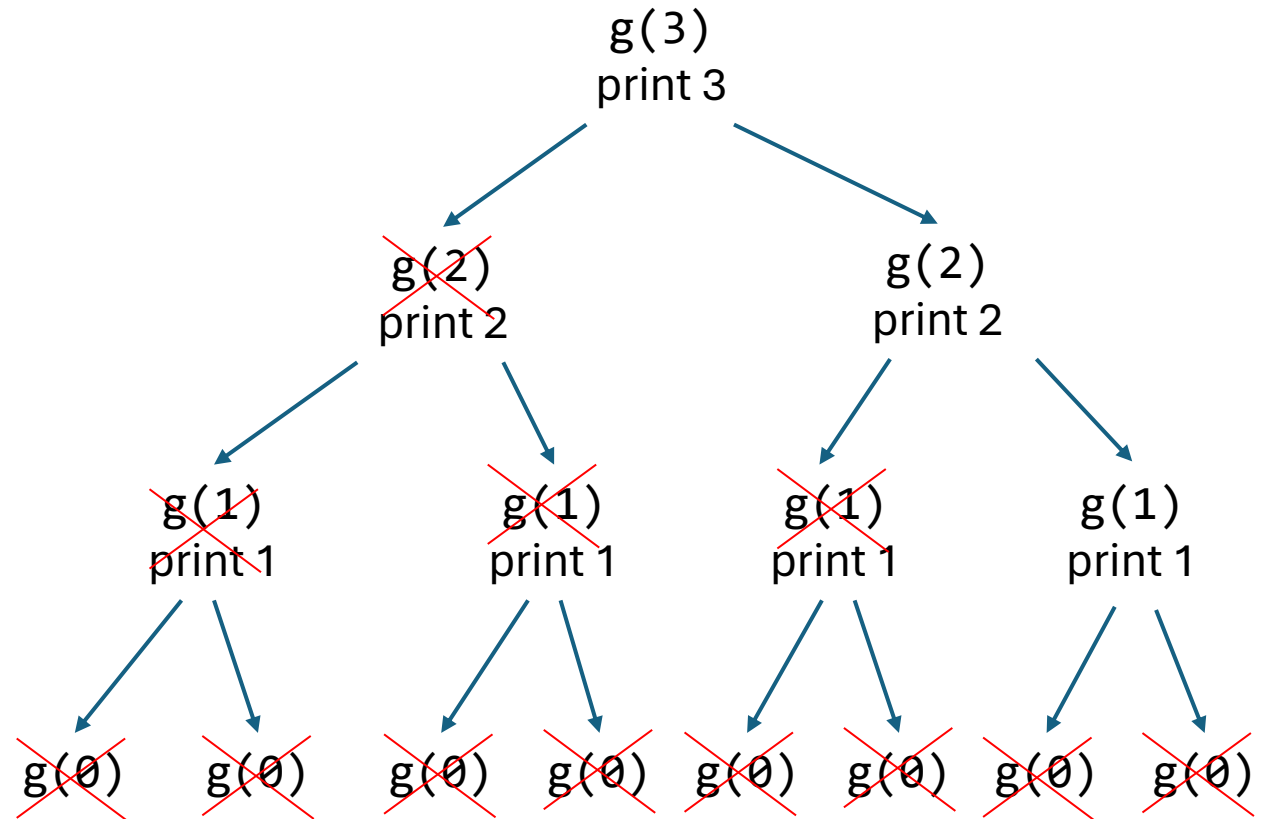


```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                           // line 5
    }
}

```

Output
3 2 1 1 2 1 1



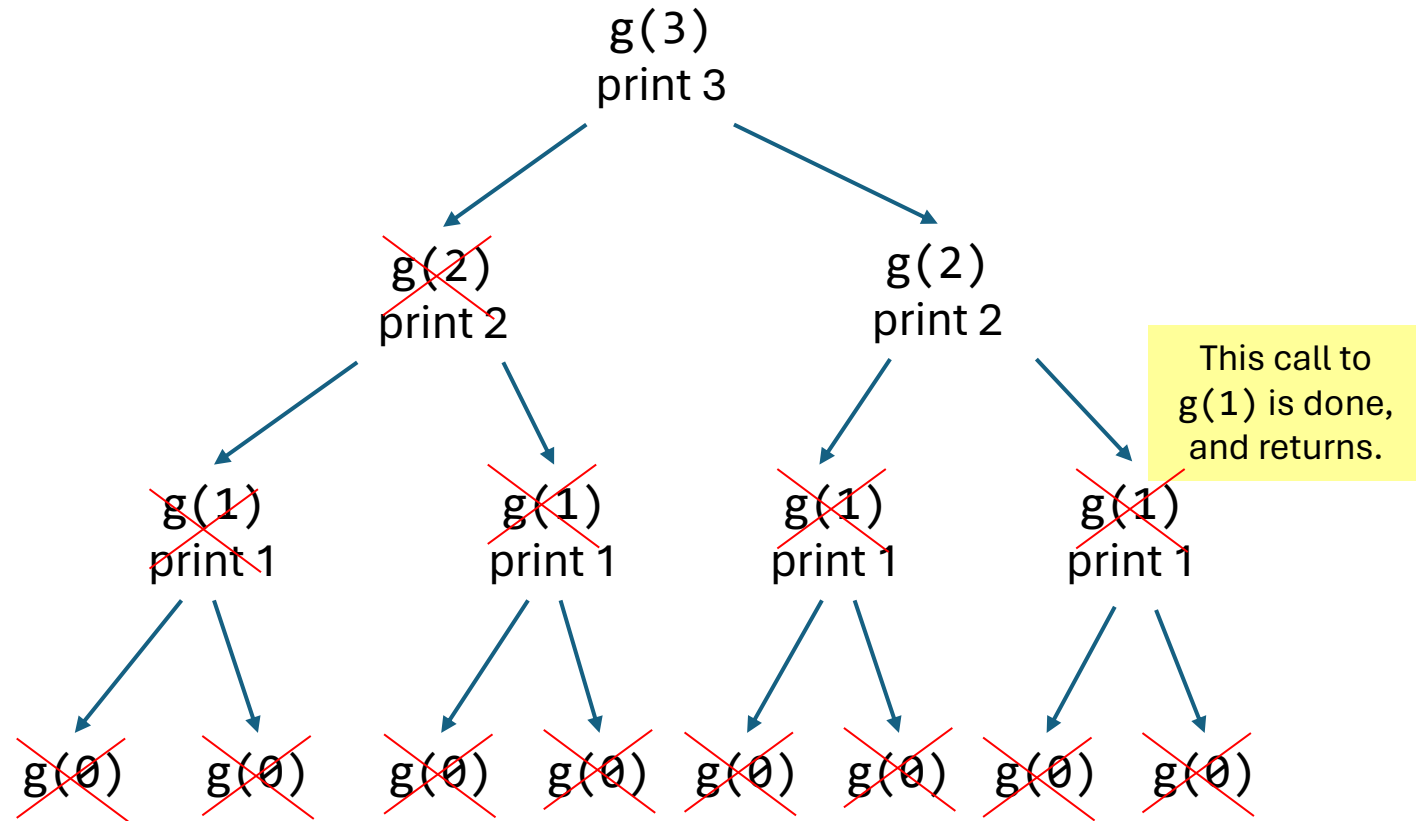
g(1) calls g(0),
both calls returning
with doing nothing.

```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1 2 1 1

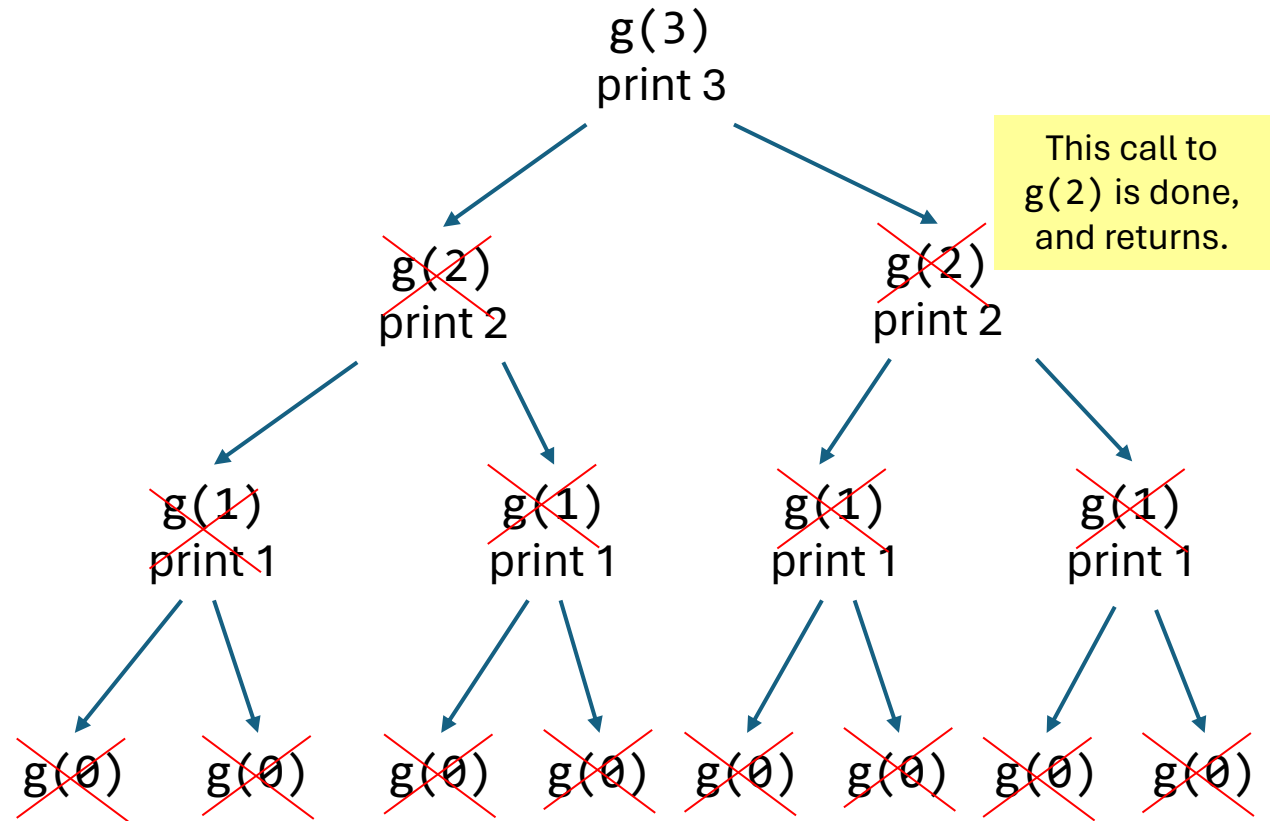


```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output
3 2 1 1 2 1 1




```

void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

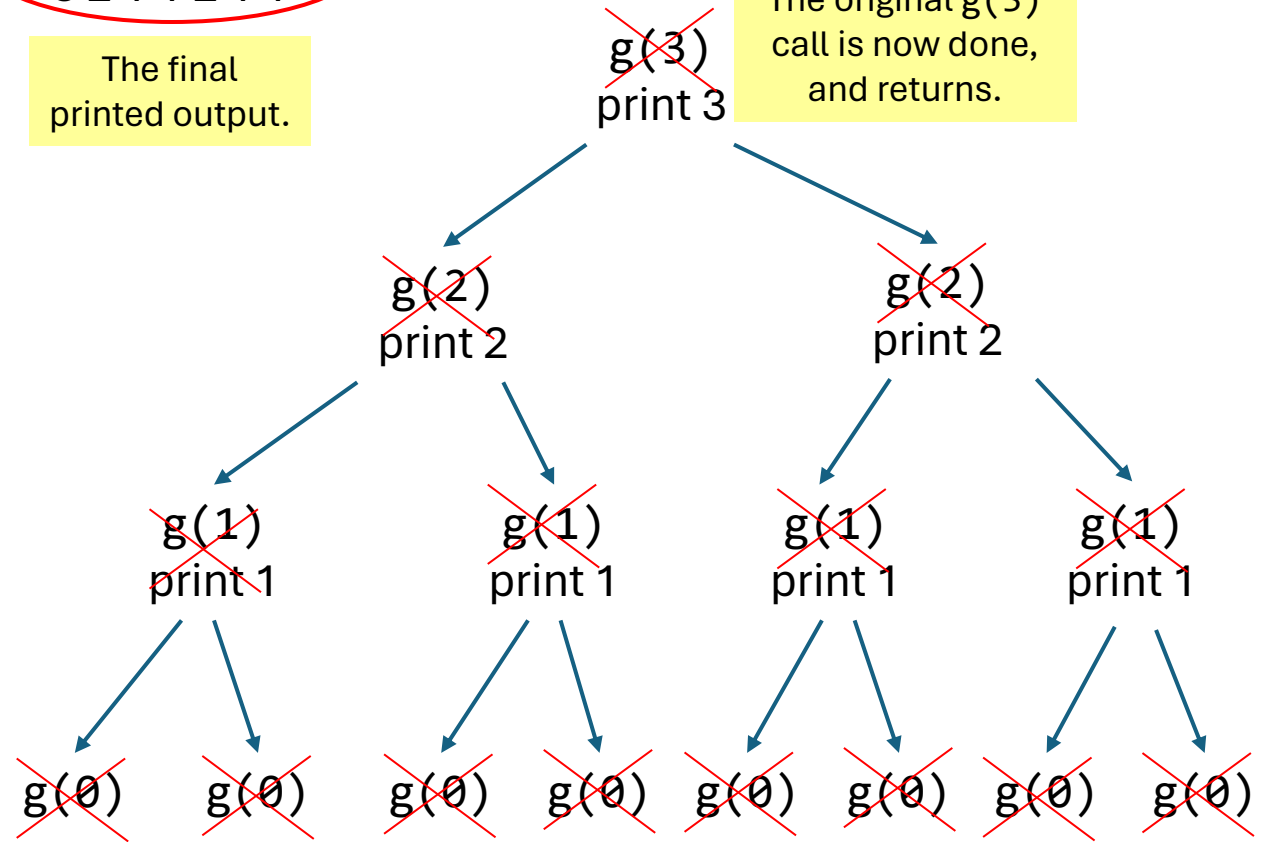
```

Output

3 2 1 1 2 1 1

The final
printed output.

The original g(3)
call is now done,
and returns.



```

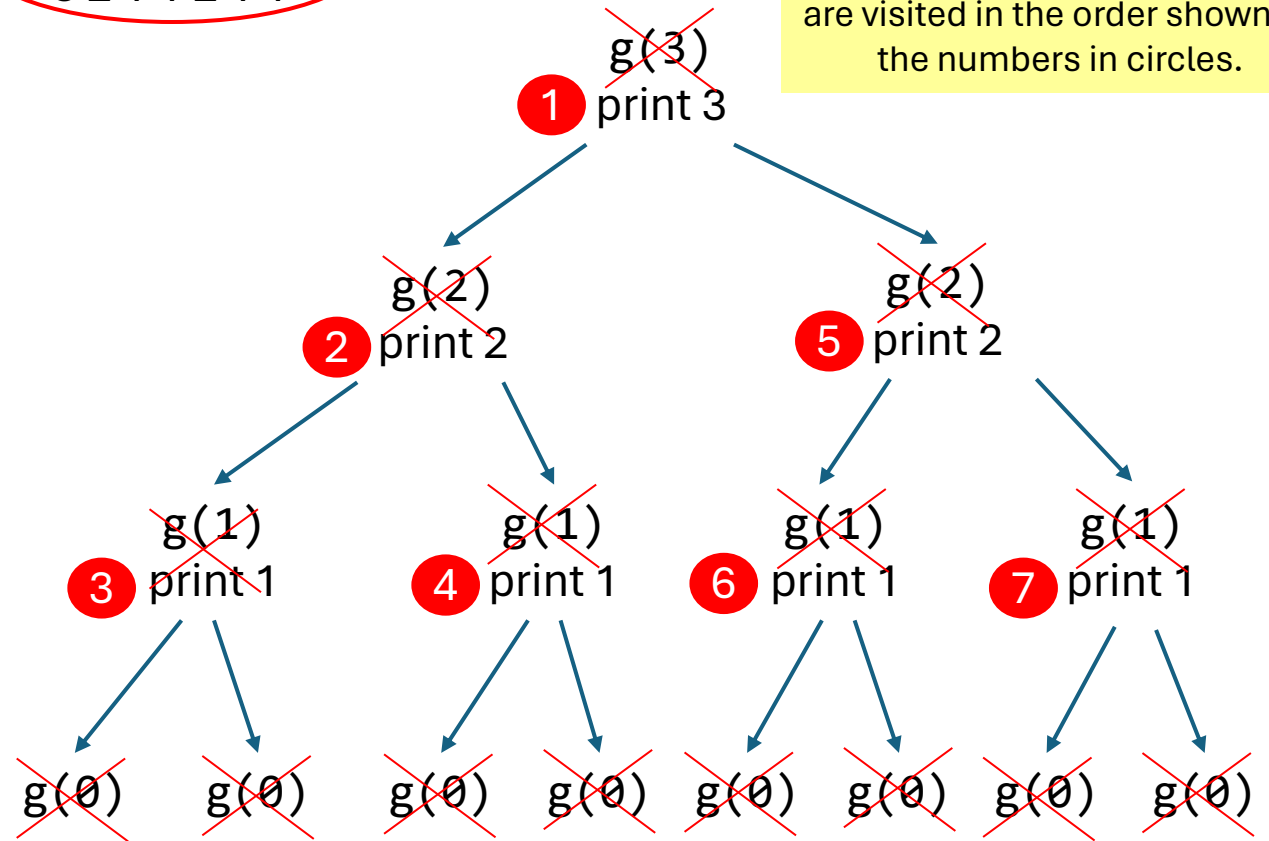
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);         // line 3
        g(n - 1);         // line 4
                        // line 5
    }
}

```

Output

3 2 1 1 2 1 1

The print statements in the tree are visited in the order shown by the numbers in circles.



```
void g(int n)
{
    if (n > 0)           // line 1
    {
        cout << n << " "; // line 2
        g(n - 1);          // line 3
        g(n - 1);          // line 4
                          // line 5
    }
}
```

For completeness, here is the final recursion tree with no other markings.

