Chapter 3

Recursion:

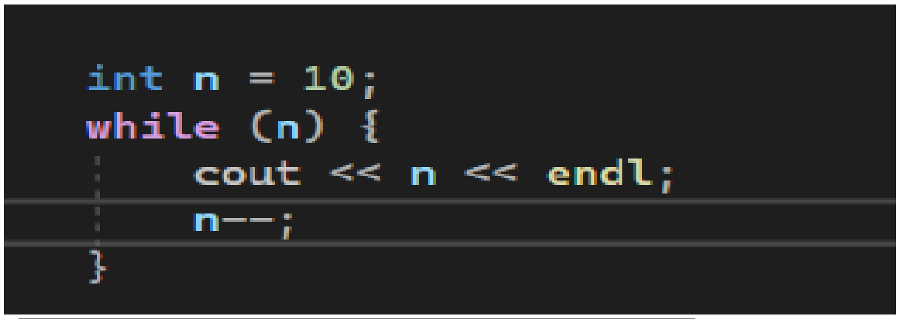
Imagine that you are looking for a key inside a box. When you open the box you find other boxes, some are empty and others have even more boxes, how do you find the keys?

There is more than one approach, and here is one:

1. Look through the box.
2. If you find a box, go to step 1.
3. If you find a key, you're done!

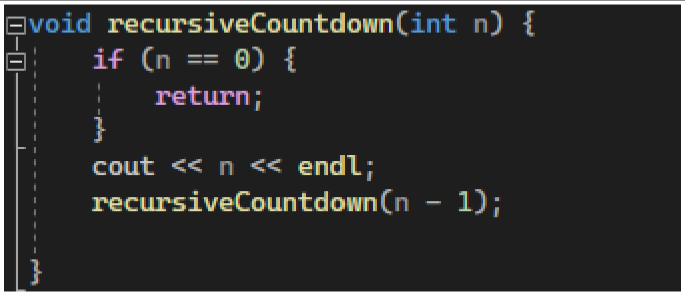
This method is a showcase of what a recursion is. But still, what is recursion in programming? Recursion is basically when you have a function that calls itself and such a function is called a recursive function.

Let's say we want to print numbers from 10 to 1 how will you do it? The firs thought would be to use loops like this:



```
int n = 10;
while (n) {
    cout << n << endl;
    n--;
}
```

But we can also use recursion to solve this problem:



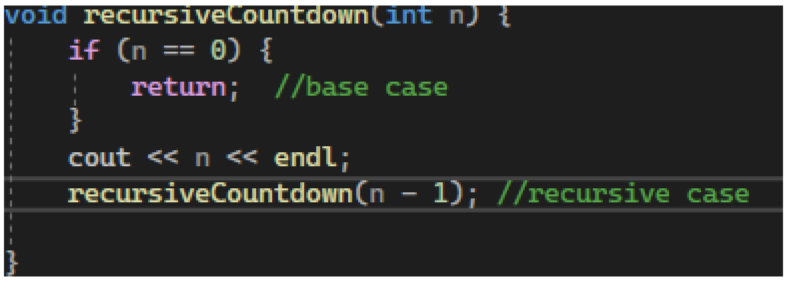
```
void recursiveCountdown(int n) {
    if (n == 0) {
        return;
    }
    cout << n << endl;
    recursiveCountdown(n - 1);

}
```

You might ask "we have loops, why do we need recursion? Does it have better performance?" well the answer is no, in fact in some cases loops are the ones who have better performance, recursion just makes the solution clearer this quote by Leigh Caldwell on Stack Overflow explains it: “Loops may achieve a performance gain for your program. Recursion may achieve a performance gain for your programmer. Choose which is more important in your situation!”

Recursive function:

recursive functions consist of two parts:

1. Recursive case: when you the function calls itself.
2. Base case: when the function doesn’t call itself again, so it doesn’t go into an infinite loop.



```
void recursiveCountdown(int n) {
    if (n == 0) {
        return;  //base case
    }
    cout << n << endl;
    recursiveCountdown(n - 1); //recursive case

}
```

The Stack:

The stack is a data structure that works LIFO (Last In First Out) , to better understand this lets imagine you have a pile of books one above the other , if you want to add or remove any book you can only do it at the topmost of the pile , that’s exactly how the stack works , if you have 10 books the only way to add another book is to add it in the 11th position , and if you want to remove the 9th you need to remove the 10th first.

Call stack:

The call stack is a stack that holds the functions you have called, suppose you called a function called add, the call stack will save it at the bottom.

add

Inside the function add you call another function that is called display what will happen is that when you are in the body of add you go perform display while add is still incomplete and display is pushed into the stack

display
add

When display is finished it is popped from the call stack and we return to add again which is also popped when it is finished

add

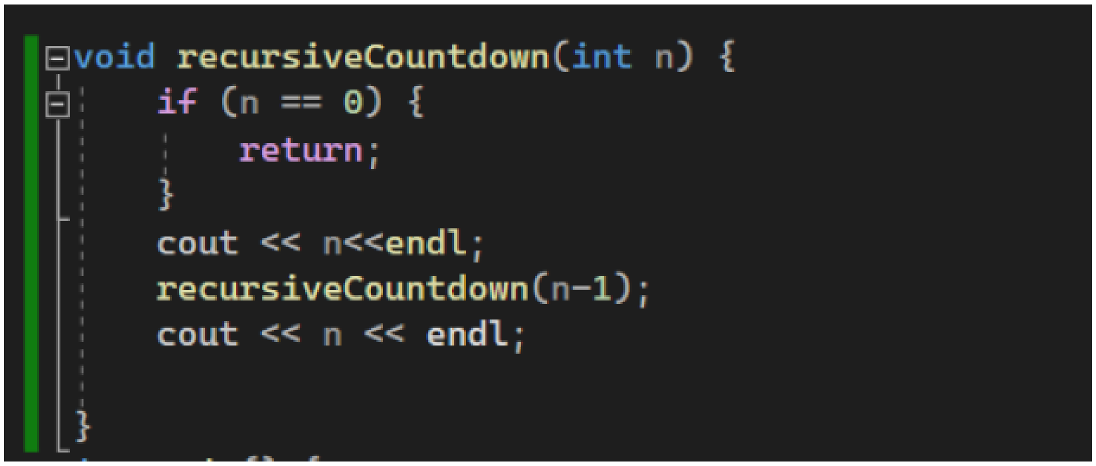
Let’s see how the call stack works with the recursive function we saw above when we count from 3 to 1.

recursiveCountDown(1)
recursiveCountDown(2)
recursiveCountDown(3)

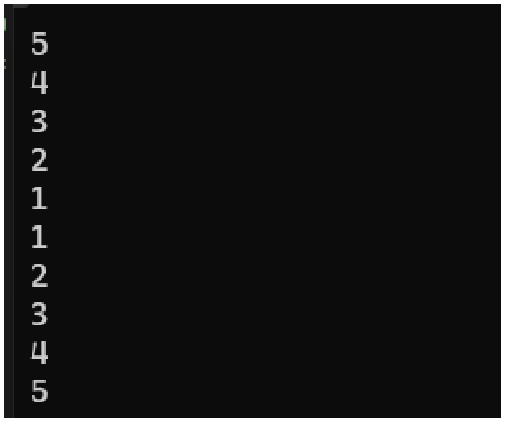
The first time you call the function with n (3) so it is pushed in the stack then you print the number and before you end the function it calls itself with n-1 (2) until you reach the base case, so the recursion stops, and the functions are finished then pops them

Notes:

You can see that in each recursion the function deals with the variable n, but you must know that each function has its own version of n that is not affected by what happens in the other recursions here is an example to clarify:



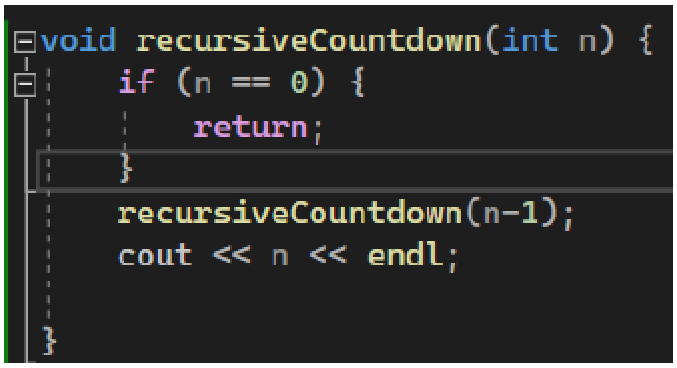
```
void recursiveCountdown(int n) {
    if (n == 0) {
        return;
    }
    cout << n<<endl;
    recursiveCountdown(n-1);
    cout << n << endl;

}
```



```
5
4
3
2
1
1
2
3
4
5
```

The n is not affected unless you purposely change it using return statement of the other recursions.

What if I want to print the numbers ascendingly? You can do that by simply interchanging the print line with the recursive case line.



```cpp
void recursiveCountdown(int n) {
    if (n == 0) {
        return;
    }
    recursiveCountdown(n-1);
    cout << n << endl;

}
```

Finally, I have added some simple recursion function in the project if you need more examples.