In this third lecture about recursion, we explain how recursive functions work.

This is the typical structure of recursive functions. A recursive function has one or several arguments. Inside this function, it first checks whether the argument, or arguments, have met the stop condition, or conditions. If the stop conditions are met, the function may perform some work to finalize the computation. If the stop conditions are not met, the function makes necessary changes to the arguments and calls the function again.

This lecture will go through a recursive function thoroughly. This recursive function has only one argument and it is an integer. At the top of this function, it checks whether the input argument’s value is zero or negative. If this condition is met, the function returns zero.

If the argument is not zero or negative, the function calls itself with n – 1. This is the change. The value of n becomes smaller.

The function’s return value is stored in a local variable called x. .

The value of x is added with the value of n. . The result is stored in another local variable y. . The function returns the value of y. .

The main function calls this recursive function with 3 as the argument. The function’s return value is stored in a local variable called ei. .

The program prints the value of ei and then exits.

To understand recursive functions, we need to understand stack memory. Some students do not understand recursive functions because they do not fully understand stack memory.

Thus, this lecture explains how the stack memory changes when running a recursive function.

Let’s start at the main function. This function calls f of 3 and stores the results in ei.

This is the stack memory when the program is about to call function f. The value of ei is unknown right now.

This slide shows the stack memory right after entering the function F. .

The argument is 3. The value address is the address of ei in the main function. The return location is the line after calling F . .

This slide adds the line numbers so that we can mark the correct return locations. The input argument n is not negative or zero. Thus, the function calls f itself again in line 10. Please notice that the local variables x and y are also in the stack memory.

It is important to distinguish the two return locations in the stack memory. When F is called the first time by the main function, the return location is line 18. When F is called the second time by F, the return location is line 11.

It is very important to keep track of the return locations.

We also need to pay attention to the value address. When F is called the first time by the main function, the value address is the address of the local variable ei inside the main function. The value address is 100. When F is called the second time by F, the value address is the address of X inside F. . The value address is 204.

Please remember that every local variable and argument has an address and the addresses are always different.

When F is called the first time by the main function, the argument N is 3. When F is called the second time, the argument is 3 - 1 and it is 2.

At this moment, F has been called twice. The first time by the main function and the second time by F. . There are two frames in the stack memory.

Some students asked me whether the frames for recursive functions would be merged into only one frame.

There is no such thing as merging frames in the stack memory. When a function is called, a frame is pushed to the top of the stack memory. This always happens without exception. This happens regardless whether a function is recursive or not.

In the second call of the function F, the input argument is 2. This does not satisfy the stop condition.

Thus, the function calls F again at line 10. The argument is n - 1 and it is 1. The value address is x’s address and it is 304.

Please be careful about the value addresses in these two calls. The addresses refer to two different locations in the stack memory.

The input argument n is not zero or negative; thus, the stop condition is not met and the function calls itself again in line 10.

The argument is N - 1 and it is zero.

As usual, we have to write the value address and return location in the top frame.

Please pay attention to the value address and the return location.

The stop condition is met and the function returns zero.

The value is written to the address 404. Then, the top frame of the stack memory is popped. The program continues from line 11.

Line 11 adds the values of x and n. In this frame, the value of x is 0 and the value of n is 1. Thus, the value of y is 1.

The program moves to line 12 and returns the value of y.

The value address is 304. Thus, the value at that address is modified to 1.

The return location is line 11 and the program continues at this line. The top frame is popped.

Line 11 adds the values of x and n. Their values are 1 and 2 respectively. The value of y is 3.

Line 12 returns the value of y. The value address is 204. Thus, the value at address 204 is modified to 3.

Then, the program continues from line 11 and the top frame is popped.

Line 11 adds the values of x and n. The value of y becomes 6.

Line 12 returns the value of y. The value address is 100. The value at address 100 is changed to 6.

The program continues from line 18 and the top frame is popped.

The program prints the value of ei and it is 6. .