This lecture explains what is stack memory and how it works.

Stack memory is also called Call Stack.

As explained in the previous lecture about computer storage, programs are stored in memory. Every statement in a program has a corresponding location in the program memory. This example shows that every line of the function has a unique location. In reality, it is more complex than that but this simplified conceptual model is sufficient for our purpose now.

When a function is called, the location after this call is stored in the stack memory.

This slide illustrates this concept. The statement after the function is the return location of the function call.

Why is it called the “return location”? When a function finishes its work, the program returns to the calling function. Thus, this is called the return location.

Consider this example. Function f1 calls function f2 at location 4. The location after this call, in other words location 5, is the return location of this function call.

This location is stored in the stack memory. Thus, location 5 is stored in the stack memory.

Next, imagine that function f2 calls f3 at location 11. The return location, 12, is stored in the stack.

By convention, new items are added to the top of the stack memory. When f3 finishes, the program continues from location 12.

When f2 finishes, the program continues from location 5.

The stack memory stores where the program should continue from.

It is the location after the function call. This is always true and there is no exception. Please remember this rule clearly. It will be one of several rules about the stack memory.

Let’s be a little more precise. What is a “location”? We can think of it as the line number of a statement. This is still a simplified conceptual model but it is a little closer to what computers actually do. Imagine that each statement in a program occupies a line. The line number after the function call is stored in the stack memory.

The real term is “program counter” but we do not need to be that precise here. Thus, I am going to use line number throughout this course since the line number is easier to understand.

Let me explain two terms used for the stack memory. When a new item is added to the top of a stack, the action is called “push”. We say an item is pushed onto the stack. When an existing item is removed from the top of a stack, we say this item is “popped” from the stack. Obviously, we cannot pop a stack if the stack is empty.

Using these terms, when a function call occurs, the line number after this call is pushed onto the stack memory. When the called function finishes, the line number is popped from the stack memory and the program continues from that line number.

This pushing-popping occurs during program’s execution. In other words, whether a line number is pushed to the stack memory may depend on the run-time conditions. In this example, calling function f2 is enclosed by a condition and the condition is not true. Thus, function f2 is not actually called. Since f2 is not called, the line number 7 is not pushed onto the stack memory.

It is important to understand that each call site has a specific line number. Thus, each call site has a corresponding return location. In this example, function f2 is called in two locations: line 4 and line 6. When f2 is called at line 4, the corresponding return location is line 5. When f2 is called at line 6, the corresponding return location is line 7. Do not mix them.

The stack memory can store four different types of information. The return location is the first type. Please remember the rule: when a function is called, the line number after this function call is stored in the stack memory.

The next lecture explains another type of information stored in the stack memory.