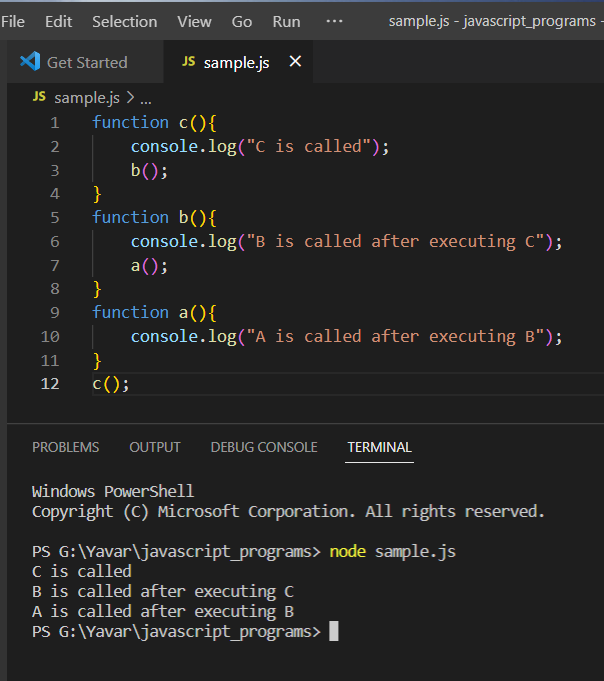
**Chapter 22- Call stack**

The call stack keeps track of executing functions in the program. Once a function is called it is placed in the call stack and once it is returned it is removed from the call stack.

Placing a function in the call stack is called pushing and removing it from the call stack is called popping.

Ex.



In the above example, the function call c() is executed first. As soon as the program gets executed in the call stack it creates a memory and a lexical environment which has the link to its parent.So it will be placed in the call stack first. On executing it will call the function b(), then it will be placed on top of c(), on executing this function will call a() eventually, then a() will be placed on top of b() in the call stack. This process is called pushing into the call stack. First a() will be executed and after that a() will be popped out and then b() will be popped out after execution and finally c() also will be popped out after execution.

**Execution context:**

The call stack is the stack of execution contexts. When a program is executed a global execution context is created. It has a memory context and the code execution context. The memory context has the variables and the functions in the program. After the execution, the content and memory used will be popped out and erased from the memory.

**Stack overflow:**

Every time a function is called, new context is created in memory. But memory is not infinite. Stack overflow occurs when the memory required to build the call stack exceeds the address space allocated for the stack. This is determined and managed internally by the browser.