Ekaba Bisong
Programming in C++
University of Calabar

Lesson Note #34
June 09, 2015

Adapted from C++ How To Program edited for our own purposes

Function Call Stack and Activation Records

To understand how C++ performs function calls, we first need to consider a data structure (i.e., collection of related data items) known as a stack. Think of a stack as analogous to a pile of dishes. When a dish is placed on the pile, it's normally placed at the top (referred to as pushing the dish onto the stack). Similarly, when a dish is removed from the pile, it's normally removed from the top (referred to as popping the dish off the stack). Stacks are known as last-in, first-out (LIFO) data structures—the last item pushed (inserted) on the stack is the first item popped (removed) from the stack.

One of the most important mechanisms for computer science students to understand is the function call stack (sometimes referred to as the program execution stack). This data structure—working "behind the scenes"—supports the function call/return mechanism. It also supports the creation, maintenance and destruction of each called function's automatic variables.

As each function is called, it may, in turn, call other functions, which may, in turn, call other functions—all before any of the functions returns. Each function eventually must return control to the function that called it. So, somehow, we must keep track of the return addresses that each function needs to return control to the function that called it. The function call stack is the perfect data structure for handling this information.

Each time a function calls another function, an entry is pushed onto the stack. This entry, called a stack frame or an activation record, contains the return address that the called function needs in order to return to the calling function.

If the called function returns, instead of calling another function before returning, the stack frame for the function call is popped, and control transfers to the return address in the popped stack frame.

The stack frames have another important responsibility. Most functions have automatic variables—parameters and any local variables the function declares. Automatic variables need to exist while a function is executing. The called function's stack frame is a perfect place to reserve the memory for the called function's automatic variables. That stack frame exists as long as the called function is active.

If more function calls occur than can have their activation records stored on the function call stack, an error known as stack overflow occurs.