ACCESS TO NON-LOCAL DATA ON THE STACK

Access becomes more complicated in languages where procedures can be declared inside other procedures. Let us consider the simple case of C functions and then language ML, that permits both nested function declarations and functions as "first-class objects;" that is, functions can take functions as arguments and return functions as values.

Data Access Without Nested Procedures

A global variable v has a scope consisting of all the functions that follow the declaration of v, except where there is a local definition of the identifier v. Variables declared within a function has a scope consisting of that function only, or part of it, if the function has nested blocks.

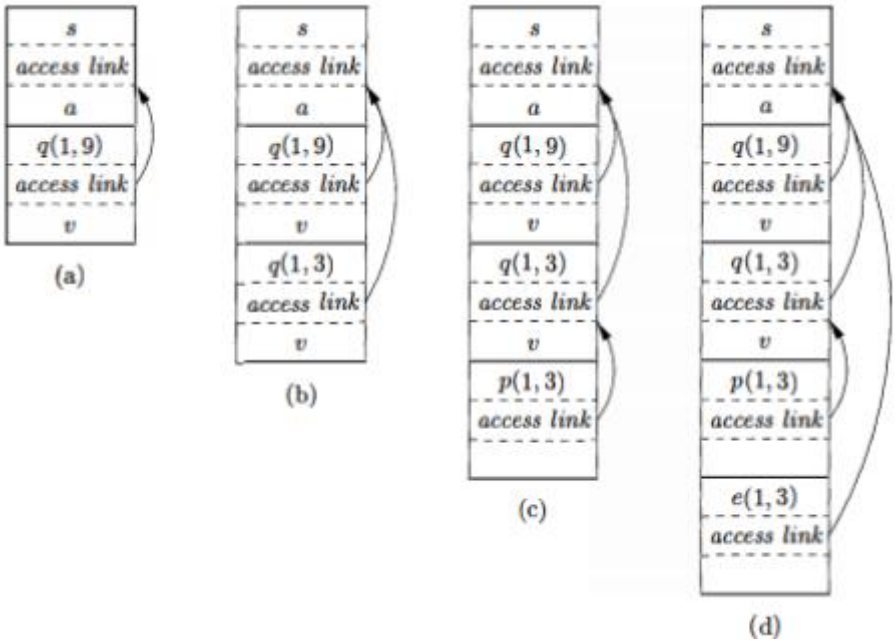
1. Global variables are allocated static storage. The locations of these variables remain fixed and are known at compile time. So to access any variable that is not local to the currently executing procedure, we simply use the statically determined address.
2. Any other name must be local to the activation at the top of the stack. These variables can be accessed through the top_sp pointer of the stack.

Issues with Nested Procedures

The access is more complicated if the language allows nested procedures and uses the normal static scoping rule. The reason is that knowing at compile time that the declaration of p is immediately nested within q does not tell us the relative positions of their activation records at run time. In fact, since either p or q or both may be recursive, there may be several activation records of p and/or q on the stack.

Access Links

- A direct implementation of the normal static scope rule for nested functions is obtained by adding a pointer called the access link to each activation record.
- If procedure p is nested immediately within procedure q in the source code, then the access link in any activation of p points to the most recent activation of q.
- The nesting depth of q must be exactly one less than the nesting depth of p. Access links form a chain from the activation record at the top of the stack to a lower nesting depths. Along this chain are all the activations whose data and procedures are accessible to the currently executing procedure.



Access Links for Procedure Parameters

- When a procedure p is passed to another procedure q as a parameter, and q then calls its parameter (and therefore calls p in this activation of q) , it is possible that q does not know the context in which p appears in the program.
- If so, it is impossible for q to know how to set the access link for p. The solution to this problem is as follows: when procedures are used as parameters, the caller needs to pass, along with the name of the procedure-parameter, the proper access link for that parameter.