Subroutine Closures

Deep binding is implemented by creating an explicit representation of a referencing environment and bundling it together with a reference to the subroutine. The bundle as a whole is called a **closure**. Usually the subroutine itself can be represented in the closure by a pointer to its code. In a language with dynamic scoping, the representation of the referencing environment depends on whether the language implementation uses an **association list** or a **central reference table** for run-time lookup of names.

One might be tempted to think that the binding time of referencing enviornments would not matter in a language with static scoping. After all, the meaning of a statically scoped name depends on its lexical nesting, not on the flow of execution, and this nesting is the same whether it is captured at the time a subroutine is passed as a parameter or at the time the subroutine is called. A program may have more than one *instance* of an object that is declared within a recursive subroutine. A closure in a language with static scoping captures the current instance of every object, at the time the closure is created. When the closure's subroutine is called, it will find these captured instances, even if newer instances have subsequently been created by recursive calls.

It should be noted that binding rules matter with static scoping only when accessing objects that are neither local nor global, but are defined at some intermediate level of nesting. If an object is local to the currently executing subroutine, then it does not matter whether the subroutine was called directly or though a closure; in either case local objects will have been created when the subroutine started running. If an object is global, there will never be more than oneinstance, since the main body of the program is not recursive. Binding rules are irrelevant in languages like C, which has no nested subroutine, thus ensuring that any variable defined outside the subroutine is global.

Suppose that we have a language with static scoping in which nested subroutines can be passed as parameters, with deep binding. Tp represent a closure for subroutine S, we can simply save a pointer to S's code together with the static link that S would use if it were called right now, in the current environment. When S is finally called, we temporarily restore the saved static link, rather than creating a new one. When S follows its static chain to access a nonlocal object, it will find the object instance that was current at the time the closure was created. This instance my not have the *value* it had at the time the closure was created, but its identity, will reflect the intent of the closure's creator.