

Subroutines and Control Abstraction

8.3.2 Call by Name

Call by name implements the normal-order argument evaluation described in Section 6.6.2. A call-by-name parameter is re-evaluated in the caller's referencing environment every time it is used. The effect is as if the called routine had been textually expanded at the point of call, with the actual parameter (which may be a complicated expression) replacing every occurrence of the formal parameter. To avoid the usual problems with macro parameters, the “expansion” is defined to include parentheses around the replaced parameter wherever syntactically valid, and to make “suitable systematic changes” to the names of any formal parameters or local identifiers that share the same name, so that their meanings never conflict [NBB⁺63, p. 12]. Call by name is the default in Algol 60; call by value is available as an alternative. In Simula call by value is the default; call by name is the alternative.

To implement call by name, Algol 60 implementations pass a hidden subroutine that evaluates the actual parameter in the caller's referencing environment. The hidden routine is usually called a *thunk*.² In most cases thunks are trivial. If an actual parameter is a variable name, for example, the thunk simply reads the variable from memory. In some cases, however, a thunk can be elaborate. Perhaps the most famous occurs in what is known as *Jensen's device*, named after Jørn Jensen [Rut67]. The idea is to pass to a subroutine both a built-up expression and one or more of the variables used in the expression. Then by changing the values of the individual variable(s), the called routine can deliberately and systematically change the value of the built-up expression. This device can be used, for example, to write a summation routine:

EXAMPLE 8.68

Jensen's device

² In general, a thunk is a procedure of zero arguments used to delay evaluation of an expression. Other examples of thunks can be seen in the `delay` mechanism of Example 6.84 (page 276) and the `promise` constructor of Exercise 10.11.

```

real procedure sum(expr, i, low, high);
  value low, high;
  comment low and high are passed by value;
  comment expr and i are passed by name;
  real expr;
  integer i, low, high;
begin
  real rtn;
  rtn := 0;
  for i := low step 1 until high do
    rtn := rtn + expr;
    comment the value of expr depends on the value of i;
  sum := rtn
end sum

```

Now to evaluate the sum

$$y = \sum_{1 \leq x \leq 10} 3x^2 - 5x + 2$$

we can simply say

```
y := sum(3*x*x - 5*x + 2, x, 1, 10);
```

Label Parameters

Both Algol 60 and Algol 68 allow a label to be passed as a parameter. If a called routine performs a goto to such a label, control will usually need to escape the local context, unwinding the subroutine call stack. The unwinding operation depends on the location of the label. For each intervening scope, the goto must restore saved registers, deallocate the stack frame, and perform any other operations

DESIGN & IMPLEMENTATION

Call by name

In practice, most uses of call by name in Algol 60 and Simula programs serve simply to allow a subroutine to change the value of an actual parameter; neither language offers call by reference. Unfortunately, call by name is significantly more expensive than call by reference: it requires the invocation of a thunk (as opposed to a simple indirection) on every use of a formal parameter. Call by name is also prone to subtle program bugs when a change to a variable in a surrounding scope unintentionally alters the value of a formal parameter. (Call by reference suffers from a milder form of this problem, as discussed in Section 3.23 [page 145].) Such deliberate subtleties as Jensen's device are comparatively rare, and can be imitated in other languages through the use of formal subroutines. Call by name was dropped in Algol 68, in favor of call by reference.

normally handled by epilogue code. To implement label parameters, Algol implementations typically pass a *thunk* that performs the appropriate operations for the given label. Note that the target of the label must generally lie in some surrounding scope, where it was visible to the caller under static scoping rules.

Label parameters are usually used to handle *exceptional conditions*—conditions that prevent a subroutine from performing its usual operation, and that cannot be handled in the local context. Instead of returning, a routine that encounters a problem (e.g., invalid input) can perform a *goto* to a label parameter, on the assumption that the label refers to code that performs some remedial operation, or prints an appropriate error message. In more recent languages, label parameters have been replaced by more structured exception handling mechanisms, discussed in Section 8.5.

✓ CHECK YOUR UNDERSTANDING

63. What is *call by name*? What language first provided it? Why isn't it used by the language's descendants?
64. What is *call by need*? How does it differ from call by name?
65. How does a subroutine with call-by-name parameters differ from a macro?
66. What is a *thunk*? What is it used for?
67. What is *Jensen's device*?

DESIGN & IMPLEMENTATION

Call by need

Functional languages like Miranda and Haskell typically pass parameters using a *memoizing* implementation of normal-order evaluation, as described in Section 6.6.2. This *lazy* implementation is sometimes called *call by need*. Memoization calculates and records the value of a parameter the first time it is needed, and uses the recorded value thereafter. In the absence of side effects, call by need is indistinguishable from call by name. It avoids the expense of repeated evaluation, but precludes the use of techniques like Jensen's device in languages that *do* have side effects. Among imperative languages, call by need appears in the scripting language R, where it serves to avoid the expense of evaluating (even once) any complex arguments that are not actually needed.

