DESIGN ISSUES FOR FUNCTIONS

The following design issues are specific to functions:

- Are side effects allowed?
- What types of values can be returned?
- How many values can be returned?

Functional Side Effects

Because of the problems of side effects of functions that are called in expressions, parameters to functions should always be in-mode parameters. In fact, some languages require this; for example, Ada functions can have only in-mode formal parameters.

This requirement effectively prevents a function from causing side effects through its parameters or through aliasing of parameters and globals. In most other imperative languages, however, functions can have either pass-by-value or pass-by-reference parameters, thus allowing functions that cause side effects and aliasing. Pure functional languages, such as Haskell, do not have variables, so their functions cannot have side effects.

Types of Returned Values

Most imperative programming languages restrict the types that can be returned by their functions. C allows any type to be returned by its functions except arrays and functions. Both of these can be handled by pointer type return values.

C is like C but also allows user-defined types, or classes, to be returned from its functions. Ada, Python, Ruby, and Lua are the only languages among current imperative languages whose functions (and/or methods) can return values of any type. In the case of Ada, however, because functions are not types in Ada, they cannot be returned from functions. Of course, pointers to functions can be returned by functions.

In some programming languages, subprograms are first-class objects, which means that they can be passed as parameters, returned from functions, and assigned to variables. Methods are

first-class objects in some imperative languages, for example, Python, Ruby, and Lua. The same is true for the functions in most functional languages.

Neither Java nor C# can have functions, although their methods are similar to functions. In both, any type or class can be returned by methods. Because methods are not types, they cannot be returned.

Number of Returned Values

In most languages, only a single value can be returned from a function. However, that is not always the case. Ruby allows the return of more than one value from a method. If a return statement in a Ruby method is not followed by an expression, nil is returned. If followed by one expression, the value of the expression is returned. If followed by more than one expression, an array of the values of all of the expressions is returned.

Lua also allows functions to return multiple values. Such values follow the return statement as a comma-separated list, as in the following:

return 3, sum, index

The form of the statement that calls the function determines the number of values that are received by the caller. If the function is called as a procedure, that is, as a statement, all return values are ignored. If the function returned three values and all are to be kept by the caller, the function would be called as in the following example:

a, b, c = fun()

In F#, multiple values can be returned by placing them in a tuple and having the tuple be the last expression in the function.