usable for any given function; it should not need to be rewritten for every function that must be integrated. It is therefore natural that the name of a program function that evaluates the mathematical function to be integrated be sent to the integrating subprogram as a parameter.

Although the idea is natural and seemingly simple, the details of how it works can be confusing. If only the transmission of the subprogram code was necessary, it could be done by passing a single pointer. However, two complications arise.

First, there is the matter of type checking the parameters of the activations of the subprogram that was passed as a parameter.

In C and C++, functions cannot be passed as parameters, but pointers to functions can. The type of a pointer to a function is the function's protocol. Because the protocol includes all parameter types, such parameters can be completely type checked.

Fortran 95 has a mechanism for providing types of parameters for subprograms that are passed as parameters, and they must be checked. Ada does not allow subprograms to be passed as parameters. The functionality of passing subprograms as parameters is instead provided by Ada's generic facility, which is discussed in Section 9.8.

In languages that allow nested subprograms, such as JavaScript, there is another issue related to subprogram names that are passed as parameters. The question is what referencing environment for executing the passed subprogram should be used. The three choices are:

1. The environment of the call statement that invokes the passed subprogram (shallow binding)
2. The environment of the definition of the passed subprogram (deep binding)
3. The environment of the call statement that passed the subprogram as an actual parameter (ad hoc binding)

The following example program, written with the syntax of JavaScript, illustrates these choices.

```javascript
function sub1() {
    var x;
    function sub2() {
        alert(x); // Creates a dialog box with the value of x
    }
    function sub3() {
        var x;
        var x = 3;
        x = x + 1;
        sub4(sub2);
    }
    function sub4(subx) {
        var x;
        subx();
    }
}

x = 4;
sub2();
}
```

Consider the execution of sub2 when it is called in sub4. For shallow binding, the referencing environment of that execution is that of sub4, so the reference to x in sub2 is bound to the local x in sub4, and the output of the program is 4. For deep binding, the referencing environment of sub2's execution is that of sub1, so the reference to x in sub2 is bound to the local x in sub1, and the output is 1. For ad hoc binding, the binding is to the local x in sub3, and the output is 3.

In some cases, the subprogram that declares a subprogram also passes that subprogram as a parameter. In those cases, deep binding and ad hoc binding are the same. Ad hoc binding has never been used because, one can assume, the environment in which the procedure appears as a parameter has no natural connection to the passed subprogram.

Shallow binding is not appropriate for static-scope languages with nested subprograms. For example, suppose the procedure Sender passes the procedure Sent as a parameter to the procedure Receiver. The problem is that Receiver may not be in the static environment of Sent, thereby making it very unnatural for Sent to have access to Receiver's variables. On the other hand, it is perfectly normal in such a language for any subprogram, including one sent as a parameter, to have its referencing environment determined by the lexical position of its definition. It is therefore more logical for these languages to use deep binding.

Some dynamic-scope languages use shallow binding.

### 9.7 Overloaded Subprograms

An overloaded operator is one that has multiple meanings. The meaning of a particular instance of an overloaded operator is determined by the types of its operands. For example, if the + operator has two floating-point operands in a Java program, it specifies floating-point multiplication. But if the same operator has two integer operands, it specifies integer multiplication.

An overloaded subprogram is a subprogram that has the same name as another subprogram in the same referencing environment. Every version of an overloaded subprogram must have a unique protocol; that is, it must be different from the others in the number, order, or types of its parameters, or