This chapter will introduce you to a powerful software engineering paradigm called Design by Contract and a method for applying it to PL/SQL programming.

Design by Contract

The following quote struck me as a revelation of software-truth when I first read it sometime in the year 2000:

*Design by Contract is a powerful metaphor that...makes it possible to design software systems of much higher reliability than ever before; the key is understanding that reliability problems (more commonly known as bugs) largely occur at module boundaries, and most often result from inconsistencies in both sides' expectations. Design by Contract promotes a much more systematic approach to this issue by encouraging module designers to base communication with other modules on precisely defined statements of mutual obligations and benefits, not on vague hopes that everything will go right.*

—Bertrand Meyer, *Object Success*

I had written quite a lot of PL/SQL code as part of developing an Oracle performance diagnostic tool for DBAs and had come to loathe chasing down runtime bugs, many of which were of precisely the kind noted by Meyer, which is to say misuse or confusion regarding APIs between modules. The promise of engineering such bugs out of my code motivated me to learn more about Design by Contract and apply it somehow to PL/SQL programming.

Software Contracts

Design by Contract makes the observation that software modules have client-supplier relationships that can be modeled after legal contracts where two parties enter into an agreement of mutual self-interest
and obligation. Each party expects to benefit somehow from the contract, and each party is usually also under some obligation from the contract. In the world of software, contracts can be thought of as the rules under which the APIs between calling and called modules must function. The calling module provides some input values or other system state when the API is invoked, and the called module is expected to reliably compute some output or resultant system state upon completion. If the rules governing these inputs and outputs are broken, there is a contract violation and the software has a defect, or bug.

The concept of software contracts governing APIs is powerful for many reasons, but most especially because a high percentage of bugs are due to confusion or misuse at API boundaries, as suggested in the quote from Meyer. If we can enforce API contracts in some way such that contract violations are exposed immediately, we can discover an entire class of bugs quickly and improve the reliability of software greatly.

Basic Contract Elements

Three basic formal elements are used to define the terms of software contracts: preconditions, postconditions, and invariants. These abstract contract elements are documented and enforced in the code by using software mechanisms typically called assertions.

Preconditions

Preconditions are conditions or states that must be true in order for a module to compute correct results. They represent obligations on the callers of a module under the contract and accrue benefits to the module itself. Preconditions benefit the module because they represent hard facts that the module’s algorithms can rely on. Preconditions oblige callers because it is the responsibility of callers to make sure the preconditions are met prior to calling the module.

It is a contract violation to call the module while failing to satisfy the preconditions; therefore precondition violations indicate bugs in the calling code. The module is not even under any obligation to compute a result in this case, as the “terms of the deal” have not been met.

At the code level, preconditions are ideally checked prior to module entry by the execution environment itself. Since PL/SQL does not offer native support for contract elements, it is recommended that all modules enforce their preconditions immediately upon module entry, especially those that govern the validity of input variable values.

Postconditions

Postconditions specify conditions on the output or computed results that the module guarantees will be true when it completes its computation, provided the preconditions have been met. They represent a fundamental obligation on the module to compute correct results. Postconditions are a benefit to the callers of the module precisely in that they allow callers to trust the module’s output.

Failure to satisfy postconditions is a contract violation by the module itself, and indicates defects are present in the module. As a practical matter at the code level, it can be very difficult to assert the full correctness of a module’s outputs at runtime either from within or without the module, as doing so can imply having independent implementations of the module’s computational requirements to compare against each other for equality.

However, it may often be the case that limited or partial postconditions can be imposed on the module’s outputs that are sufficient to indicate a contract violation. For instance, if a function is