18 Summary: Contracts, and Why They are Important

We have introduced *contracts*, using the example of an algorithm for integer exponentiation.

Contracts are expressed in form of annotations, started with `//@`. These annotations are checked when the program is executed if it is compiled or interpreted with the `-d` flag. Otherwise, they are ignored.

The forms of contracts, and how they are checked, are:

`@requires`**:** A precondition to a function. This is checked just before the function body executes.

`@ensures`**:** A postcondition for a function. This is checked just after the function body has been executed. We use `\result` to refer to the value returned by the function to impose a condition on it.

`@loop_invariant`**:** A loop invariant. This is checked every time just before the loop exit condition is tested.

`@assert`**:** An assertion. This is like a statement and is checked every time it is encountered during execution.

Contracts are important for two purposes.

Testing: Contracts represent a kind of generic test of a function. Rather than stating specific inputs (like `f(2,8)` and testing the answer `256`), contracts talk about expected properties for *arbitrary* values. On the other hand, contracts are only useful in this regard if we have a good set of test cases, because contracts that are not executed with values that cause them to fail cannot cause execution to abort.

Reasoning: Contracts express important properties of programs so we can *prove* them. Ultimately, this can mathematically verify program correctness. Since correctness is *the* most important concern about programs, this is a crucial aspect of program development. Different forms of contracts have different roles, reviewed below.

The proof obligations for contracts are as follows:

`@requires`**:** At the call sites we have to prove that the precondition for the function is satisfied for the given arguments. We can then assume it when reasoning in the body of the function.

`@ensures`**:** At the return sites inside a function we have to prove that the postcondition is satisfied for the given return value. We can then assume it at the call site.

`@loop_invariant`**:** We have to show:

Init: The loop invariant is satisfied initially, when the loop is first encountered.

Preservation: Assuming the loop invariant is satisfied at the beginning of the loop (just before the exit test), we have to show it still holds when the beginning of the loop is reached again, after one iteration of the loop.

We are then allowed to assume that the loop invariant holds after the loop exits, together with the exit condition.

`@assert`**:** We have to show that an assertion is satisfied when it is reached during program execution. We can then assume it for subsequent statements.

Contracts are crucial for reasoning since (a) they express what needs to be proved in the first place (give the program's *specification*), and (b) they *localize* reasoning: from a big program to the conditions on the individual functions, from the inside of a big function to each loop invariant or assertion.