THE CASE FOR THE ASSERT STATEMENT

by

David Matuszek
Department of Computer Sciences
The University of Texas
Austin, Texas 78712

While the use of structured programming techniques can sharply reduce errors, debugging is still very much with us. Even when the programmer takes care right from the beginning to insert statements to print intermediate results, the central task of debugging remains that of locating errors on the basis of inadequate (or superabundant) information.

This situation can be mitigated by the use of a simple, transparent statement that can be introduced into any Algol-like language with relative ease:

```
assert <Boolean expression>
```

with semantics

```
if ¬(<Boolean expression>) then
     begin
        print ("Assertion not met:
               <text of Boolean expression>",
               <values of variables used in Boolean expression>);
        abort
     end
```

For example, the statement `assert x>0 & x≤xmax` would be equivalent to

```
if ¬(x>0 & x≤xmax) then
     begin
        print ("Assertion not met: x>0 & x≤xmax",
               "x = ", x, "xmax = ", xmax);
        abort
     end
```

Additional implementation-dependent information, such as a line number or machine address, could be added to the above print list. Traceback information, if available, would also be helpful.

The assert statement has the following virtues:

(1) It is very natural. A programmer is constantly aware while writing code of the assumptions he is making, and the assert statement is a trivial way of making many of these explicit.

(2) It effectively documents (and enforces) invariants which must be maintained as the code is debugged or modified.

(3) It provides a painless way to check inputs for reasonableness and to ensure that program limits (e.g. array bounds) are not violated.

(4) It can be added to or removed from a working program without affecting the behavior of that program (provided the <Boolean expression> does not invoke a function with side effects).

(5) If efficiency is considered essential, compiler or preprocessor flags could cause assert statements to be treated as comments.

(6) From an implementation point of view, all information required to construct the print list is readily available to the compiler or preprocessor.
(7) It is psychologically easier to assert that one's code works than it is to insert statements for the specific purpose of locating errors.
(8) Most important, the assert statement is so simple that it stands an excellent chance of being incorporated into programs right from the beginning, rather than merely being stuck in after something has already gone wrong.

As compared to a trace statement which causes a line of information to be printed each time the value of a traced variable is altered, the assert statement has the following advantages:
(9) There is no need to decide which variables to trace, or at what points to turn tracing on or off. Nor is there any danger that tracing never gets turned off.
(10) assert statements cannot produce reams of paper which show only that no errors have yet ocurred. Thus there is little need to be cautious about using them.
(11) If an assertion is not met, all relevant variables are printed, not just those which the programmer conjectures to be at fault.
(12) The assert statement does not modify the interpretation of other statements in the program (e.g. assignment statements), thus it is much simpler to add to an existing compiler than tracing facilities.
(13) The assert statement provides valuable documentation, so it is as appropriate in a working program, or in an algorithm intended for publication, as it is in a program under development.

It is easy, but not necessarily desirable, to invent extensions to the assert statement. As presented here the statement is simple, has no options to choose among, and can be jotted down by the programmer without distracting him overmuch from his task of writing the "real" code. Any extensions, even optional ones, increase the complexity of the statement and may reduce the frequency with which it is used.

A particularly alluring extension is the ability to make assert statements nonfatal, or to allow only a preset maximum number of errors to occur. This has the additional disadvantage of drawing attention to the debugging aspect of the statement rather than the documentary aspect. It might be better, though less flexible, to provide this capability by a control card parameter.

The assert statement is based directly on the pioneering work of Floyd [1]. While analogues to the statement as described here have occasionally appeared in debugging packages [2], assert has apparently never been included as an integral part of a programming language. Perhaps it is time that higher-level languages began including this and other debugging facilities in the language specification, rather than leaving it to local implementations to provide octal dumps as the primary debugging aid.

References:
[1] R.W. Floyd, "Assigning meanings to programs", Proc. Symp. Appl. Math., 19, 19-32 (1967).

[2] E. Satterthwaite, "Debugging tools for high level languages", Software - Practice and Experience, 2, No. 3, 197-217 (1972).