Programming Languages Should NOT Have Comment Statements

Michael J. Kaelbling
1386 Friar Lane, Columbus OH 43221, USA

Abstract

The purpose of this paper is to open discussion on the adequacy and design of commenting techniques in programming (and, indeed, all computer processing) languages. The argument is made that the common notion of comment statements is too simplistic, and that a powerful alternative exists. A distinction is made between simple comment statements and scoped comments. Two types of scoped comments are mentioned, and their weaknesses are considered. The paper ends with a call for language designers to address this issue.

Introduction

Programming languages should not have simple comment statements; they should have scoped comments. A simple comment statement is a self-standing construct, syntactically unrelated to other parts of a programming language. It is the semantics of a comment that relate it to a program and convey information. Thus one must understand a comment to know to what it refers. A scoped comment is a construct syntactically related to a part of a program. Without regard to actual meaning, the referent of a scoped comment can be quickly determined. An examination of popular programming languages will reveal that most languages contain comment statements, not scoped comments.

The major problem with comment statements is that one cannot automatically be sure to what they apply. A good deal of understanding is required to break the catch-22: to use the comments to understand the program, one must understand the program enough to use the comments. True, comment statements are usually located near to their referents, but this is a convention. Without recourse to semantics, one does not know if a comment statement refers to a preceding statement, or to a following statement, or to a group of statements, or to a part of a statement. This ambiguity problem defeats program transformation systems [1] and prettyprinters [2], which must make assumptions about the logical relationship of comments to code.

Scoped comments avoid the ambiguity problem that plagues comment statements. A scoped comment, much like a scoped variable, has a clearly defined area of application. The use of scoped comments would allow for enhancements to program editing and maintenance systems. With the cursor pointing to an identifier, one could query for comments associated with the declaration of that identifier. Then, one could query for comments that apply to the statement in which the identifier appears, and for comments that apply to the block in which the statement appears, and so on.

Scoped Comments in a Language (Comment Clauses)

One way to apply scoped comments to a program would be to insert a comment clause into each statement. For example, the partial language specification given below makes clear the syntactic relationship of comments to statements and groups of statements.

```
<statement>       ::= <assignment> | <compound stmt>
<assignment>      ::= <id> := <expr> <comment>
<compound stmt>   ::= BEGIN <comment> <statements> END
<comment>         ::= /* <characters> */
<statements>      ::= <statement> | <statements> ; <statement>
```

Such a grammar could be used by language-oriented editors (like those of ALOE [3]) to prompt for comments and increase the likelihood of well-documented code being written. Such an approach, however, has drawbacks: (1) a new, and perhaps incompatible, grammar would have to be specified, (2) the method of applying comments would probably be language-specific, and (3) the structure of the comments would have to follow the structure of the program. The first drawback could lead to compatibility problems with existing programs, compilers, and methodologies. The second drawback could create the burden of having to learn and support several different commenting techniques in a multilingual system. The third drawback might not allow for convenient comments, e.g., in the language of the partial grammar we saw above, it would be impossible to apply one comment to two consecutive statements without introducing a compound statement.

Simulating Scoped Comments (Making Do)

A second way to apply scoped comments to a program would be to make use of comment statements to simulate scoped comments. Such an approach avoids drawback (1) and minimizes drawbacks (2) and (3). One would, however, have to resort to convention to ensure that comment scoping was not corrupted by arbitrary comments. To implement the approach, one would define comment prefixes to indicate scope. The main problem with this approach would be its reliance on the goodwill and diligence of the user. Here are some sample prefixes:

`/* [: abc... */`	is a comment for the area starting here,
`/* ]: abc... */`	is a comment for the area ending here,
`/* p: abc... */`	is a comment for the preceding statement,
`/* =: abc... */`	is a comment for the preceding identifier.

Concluding Remarks

With all the attention given to the design of programming languages, it seems that the critical, but lowly, role of comments deserves more thought. Perhaps documentation does not belong in a program, but rather layered on top with its own syntax and conventions. Surely there is a better way to specify location-dependent information than scattering undirected, one-dimensional strings throughout a file. Two possible approaches have been mentioned in this paper in an effort to spark the interest of language designers. I await with anticipation their clever solutions to the problem of simplistic comment statements.

References

[1] Calliss, F.W. "Problems With Automatic Restructurers", *SIGPLAN Notices*, vol. 23, no. 3, pp. 13-21, March 1988.

[2] "indent" in Commands Reference Manual, Sun Microsystems, Inc., 1986.

[3] Habermann, N., Medina-Mora, R. et al. "A Compendium of Gandalf Documentation", Technical Report, Dept. of Computer Science, Carnegie-Mellon University, April 1981.