Automatic Indention Versus Program Formatting

Amiram Yehudai
Computer Science Division
Department of Mathematical Sciences
Tel-Aviv University
Tel-Aviv, ISRAEL

Indention has been widely recognized as an aid to improve program readability [1]. Various different schemes for indention have been proposed recently (see e.g. [2,3,4] and some of the papers cited in them); preprocessors for them have been written and some compilers produce their listing in an indented layout.

In [3] it is argued that indention is a matter of style. It is therefore not surprising that no indention scheme has gained overwhelming acceptance by the programming community, even if we restrict ourselves to Pascal users.

The debate about what is a good indention scheme can be fruitful only if people do not aim at finding one scheme that will become a standard. As is advocated in [2], we should be able to deviate from our formal rules of layout for purpose of readability. Therefore the thought that eventually every program will go through one accepted preprocessor and come out looking right cannot be accepted.

Since indention reflects the programmer's style, he or she should choose the indention scheme. Moreover, it is not clear that any automated indention scheme will be adequate, as I may choose different ways to lay out the same construct in different parts of my program.

A case in point is an if-then-else statement. Consider the following two program segments.

segment 1

```
if x=y
    then action 1
else if x+y<2*z
    then action 2
else if x+y+z=10
    then action 3
else        action 4;
```

segment 2

```
if x=y
   then if x<z
           then if  z<u
                    then action 111
                    else action 112
           else if  x<u
                    then action 121
                    else action 122
   else if x<y
           then action 21
           else action 22 ;
```

Note how the different indention schemes reflect the different roles of the if-then-else constructs. One may argue that if we had an elseif keyword [5] the first segment would use it and elseif may then use an indention scheme other than that of else and if. This would not solve the problem for the last else in that segment. Another solution might be to have a guarded command construct [6] that would let us write segment 1 without using if-then-else.

These solutions, and others, do not address the general problem. In Pascal, and in fact in any language, no matter how rich in language constructs, there will be different ways to use each construct. It would be impossible for an indention program to understand all these different styles, even if it actually parses the program.

Rather than force programmers into one scheme of indention, I propose that programmers be encouraged to indent their programs with the help of a program formatter. A program formatter (such as Spruce [7]) provides the user with various simple directives that may be inserted (as comments) into the program to dynamically set some of the indention parameters.

In Spruce one can direct the formatter to collect more than one statement per line, in certain parts of the program. One can change the indent tab, making it small where code is heavily nested etc.

It may be desirable to include in such a system the ability to redefine indention schemes for various language constructs, although these macro definitions will make both use and implementation of the formatter more complex.

In conclusion we argue that programmers be encouraged to indent their programs to reflect their own style. Like authors of any other type of text they should be provided with a

mechanism to conveniently format their text according to their own style, and not forced into one scheme of indention.

We hope the flood of proposed indention systems will be followed by suggested program formatters that will gain recognition and accepted as essential software development tools.

References

1. Zelkowitz, M.V. "Perspectives in Software Engineering", Computing Surveys 10:2 (June 1978), 197-216.

2. Grogons, P. "On Layout, Identifiers and Semicolons in Pascal Programs", SIGPLAN Notices 14:4 (April 1979), 35-40.

3. Bond, R. "Another note on Pascal Indention", SIGPLAN Notices 14:12 (December 1979), 47-49.

4. Gustafson, G.G. "Some Practical Experiences Formatting Pascal Programs", SIGPLAN Notices 14:9 (September 1979) 42-49.

5. Preliminary ADA Reference Manual, SIGPLAN Notices 14:6 (June 1979).

6. Dijkstra, E.W. "Guarded Commands, Nondeterminism, and Formal Derivation of Programs", Comm. ACM 18:8 (August 1975), 453-457.

7. Spruce, a Pascal Program formatter, M.N. Condict, R.L. Marcus and A. Mickel, University Computer Center, University of Minnesota. Included in Pascal 6000 Release 3.