Stylistics in Languages With Compound Statements

By Arthur Sale*

This short communication discusses a stylistic problem which arises in languages, such as PASCAL, which use both statement separators, such as semicolons, and begin-end bracketting structures. It suggests that an alternative to the traditional rules which have evolved from Algol 60 is preferable.

KEYWORDS AND PHRASES: Programming stylistics, indentation, compound statements.
CR CATEGORIES: 4.20.

1. INTRODUCTION

Programming languages of reasonably modern origin usually have two features which are of interest to this paper:

(a) the language is made up of *statements* separated by some lexical token, usually a semicolon, and

(b) groups of statements can be treated together by enclosing them by distinctive tokens, for example **begin** and **end**.

Such languages are common; examples are Algol 60, PL/1, Algol 68, and PASCAL.

The problem of stylistics arises from the basically line-oriented structure of program source text, and the increasingly prevalent practice of emphasising this structure by indentation. This structure cuts across the syntax structure to create mismatch problems which are not present in langauges like FORTRAN and BASIC.

Some rules for stylistically formatting programs in such languages have been developed from the requirements specified by the editors of the (now discontinued) Algorithms section of the Communications of the ACM originally for Algol 60. These rules, which I shall call the *classical rules*, can be incompletely summarised as follows:

(a) Every **begin** and **end** shall be on a line by itself, and the enclosed text shall be indented one level deeper than either the **begin** or the **end**.

(b) Every internal statement of a statement (such as *s* in "*while b do s*"), shall be indented one level deeper than the context structure.

(c) Semicolons are inserted only when necessary to separate two adjacent statements.

To focus attention, the following program fragment has been modified from a recent program I wrote and put into classical style as shown in Figure 1.

2. PROBLEMS WITH THE CLASSICAL STYLE

The classical style has three problems; their relative importance being subjective:

(i) **wastage of page space**

vertically: The layout has many lines of little information content. The extreme case is "**end else begin**" in the middle of an **if**, which borders on the ludicrous.

horizontally: If most compound constructs contain **begin-end** pairs, inner statements are indented by two levels. This means that programs containing much structuring detail rapidly move across the page, to be wrecked at the right margin (as in the distributed version of PASCAL-P).

(ii) **editing inconvenience**

Since most editing facilities (be they terminal-oriented, or punched cards) work on a source line, the classical placement rules for semicolons make insertion or deletion of lines a real headache. Such editing can often have consequent effects on the lines surrounding the change; in the extreme case a four line change is required to add a second line to an internal statement which is not already enclosed in a **begin-end**. (Interestingly, the declaration part of PASCAL avoids this problem by good design.)

(iii) **understanding**

It is apparently quite difficult to reconcile two different structuring rules which cut across each other, and beginning students seem to have much difficulty in deciding where to put semicolons. The evidence seems to show that if they were never told anything about layout, and could write one long line of program, they would have no problems with semicolons. But introduce these topics, and all is confusion. To help, they may devise line-oriented

```
begin
    {classical style}
    while (i ≤ M) do
        begin
            ch:=text[i];
            if (ch=lastchar) then
                begin
                    checkoccurrence;
                    i:=i+D[ch]
                end
            else
                begin
                    increment(usecount[stride[ch]]);
                    i:=i+stride[ch]
                end
        end
end;
```

Figure 1

* *Department of Information Science, University of Tasmania.* *Manuscript received 17th January, 1978. Revised version received 7th March, 1978.*

rules-of-thumb which imply the stream-oriented rules underlying the syntax. For example, one such rule is:

"Never put a semicolon after a **begin** or an **else**"

The difficulty about this lies in when to put semicolons on ordinary statements (it depends on what follows), or on **ends**. Such rules are complex.

3. ALTERNATIVES

Clearly the problem revolves around internal statements, **begin-end**, and semi-colons. The possibilities are legion, as I have discovered by polling my colleagues on what they consider to be best practice, and by looking at real programs. Though few programmers seem to understand why they write programs the way they do, there are a lot of style variations around.

The following three examples should illustrate the variety that is possible:

```
while b do
begin
    s1;
    s2
end;

while b do
    begin
    s1;
    s2
    end;

while b do
    begin s1;
    s2 end;
```

4. A PERSONAL STYLE

I would like here to promote a personal style which I use for all programs I write, and which I believe has some important morals for designers of programming languages. This personal style consciously tries to minimize the problems I identified earlier while retaining the advantages of syntactic regularity and obvious indentation structure. Let me first re-write my example in my own style and then justify it.

```
begin
    {personal style}
    while (i ≤ M) do begin
        ch:=text[i];
        if (ch = lastchar) then begin
            checkoccurrence;
            i:=i+D[ch];
        end else begin
            increment(usecount[stride[ch]]);
            i:=i+stride[ch];
        end;
    end;
end;
```

The rules for this style can be summarised as follows:

RULE 1: Put semicolons at the end of every line, except after a begin (and even here they don't matter), or if you have to break up something you'd normally put on one line over several (such as an assignment or an expression).

RULE 2: Remember each statement construct as a line oriented template, as for example:

```
while condition do begin
    statements;
end;
```

COROLLARY: Always use **begin-end** even if only one statement is controlled.

COROLLARY: The template for an **if** includes '**end else begin**', so no confusing errors due to **elses** are possible.

This style is more compact, though it is possible to reduce vertical space usage even further, and I maintain that it does not compromise the basic purposes of layout at all. The problem of where to put semicolons is greatly simplified, and editing insertions and deletions always involve only the one line concerned. Beginning students should also find that the regular use of syntactic templates of the kind illustrated (which have no exceptions to remember) is quite easy, and they do not have to reconcile some stream-oriented syntax rules with another set of layout rules.

Why is this not a universal style then? Probably the answer lies in two aspects: *habit* and *syntax*. We are all familiar with the resistance to change when we try to alter some deeply ingrained and habitual behaviour, and it is quite apparent in some conversations I have had that layout is a learned response of this type. Some programmers remark that they find my examples harder to follow, and are taken aback when I say that I find their style similarly awkward (because of relative unfamiliarity).

The other resistance arises because the classical style rules directly mirror the syntactic constructs of Algol 60. Apply a new syntax production rule, and you indent one level, at least down to the statement productions. Of course Algol 60 totally ignored the existence of lines, and I argue now that this was unfortunate, and should not force us to therefore adopt clumsy habits. To illustrate this we can look at two constructs which were imported into PASCAL but were not in Algol 60. The **case** and **repeat** constructs do not require **begins**, and have defined closing terminators (**end** and **until**). Curious that in the two new importations, the language is better than the carry-overs . . .

Example:

```
case expression of
    statements
end;
```

5. MORALS

1. Designers of programming langauges should be more aware that the languages they design are not simply collections of syntactic rules, but are going to be mapped onto lines, and the language should acknowledge this fact.
2. The **begin** ought to die a graceful death; compound constructs ought to embrace the many-statement enclosure as the normal case, not as an exception.
3. All compound constructs should have an explicit termination; perhaps chosen to match the construct (as in **repeat-until**, or the less felicitous **if-fi** of Algol 68).
4. We should all ask ourselves more often: "why do I lay programs out the way I do?"