View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Research Online

University of Wollongong

Research Online

Department of Computing Science Working Paper Series

Faculty of Engineering and Information Sciences

1982

A semantically-based formatting discipline for Pascal

Paul A. Bailes
University of Wollongong

Antonio Salvardori
University of Wollongong

Follow this and additional works at: https://ro.uow.edu.au/compsciwp

Recommended Citation

Bailes, Paul A. and Salvardori, Antonio, A semantically-based formatting discipline for Pascal, Department of Computing Science, University of Wollongong, Working Paper 82-19, 1982, 26p.
https://ro.uow.edu.au/compsciwp/70

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

THE UNIVERSITY OF WOLLONGONG

A SEMANTICALLY-BASED FORMATTING DISCIPLINE FOR PASCAL

by

Paul A. Bailes and Antonio Salvadori

Preprint No. 82/19

DEPARTMENT OF COMPUTING SCIENCE

P.O. BOX 1144, WOLLONGONG, N.S.W 2500, AUSTRALIA

A SEMANTICALLY-BASED FORMATTING DISCIPLINE FOR PASCAL

by

Paul A. Bailes and Antonio Salvadori

Preprint No. 82-19 November 12, 1982

P.O. Box 1144, WOLLONGONG N.S.W. 2500, AUSTRALIA
tel (042)-282-981
telex AA29022

A Semantically-based Formatting Discipline for Pascal

Paul A. Bailes and Antonio Salvadori[*]

Department of Computing Science

The University of Wollongong

Wollongong N.S.W.

AUSTRALIA

ABSTRACT

The abstract (or semantic) syntax of the Pascal language is identified, and a linear representation for the trees so formed within the framework of the concrete syntax for that language is imposed. The indentation scheme so formed, augmented with a small number of pragmatic considerations, is compared with several previously proposed formatting schemes for Pascal and an example of the use of this new method is given.

November 12, 1982

---

[*] permanent address: University of Guelph, Ontario, Canada N1G 2W1.

A Semantically-based Formatting Discipline for Pascal

Paul A. Bailes and Antonio Salvadori *

Department of Computing Science

The University of Wollongong

Wollongong N.S.W.

AUSTRALIA

SUMMARY

The abstract (or semantic) syntax of the Pascal language is identified, and a linear representation for the trees so formed within the framework of the concrete syntax for that language is imposed. The indentation scheme so formed, augmented with a small number of pragmatic considerations, is compared with several previously proposed formatting schemes for Pascal and an example of the use of this new method is given.

KEY WORDS Programming languages Formatted languages Program readability Pascal Abstract syntax

INTRODUCTION

Recognition that a program is not just an example of person-machine communication but also, and very importantly, one of person-person communication, has developed ever since the appearance of the first "high-level" languages. Such communications include those between program writer and program maintainer during the life cycle of an item of software, and those between members of a team involved in either development or maintenance.

* permanent address: University of Guelph, Ontario, Canada N1G 2W1.

Methodological and associated language developments such as constructs supporting structured programming[1] and data abstraction[2] take cognizance of this fact. Nevertheless, their mere presence in a program, considered as a string of symbols written line by line does not immediately convey to the reader an understanding of its semantic structure.

This paper attempts to provide a remedy to this situation with emphasis on Pascal[3], by proposing a layout or formatting discipline to clearly display such structures. This layout can be achieved either by formatting or "pretty printing" programs or by individuals when preparing text in the absence of such an aid, in which case, the term "discipline" becomes particularly meaningful. The choice of Pascal is motivated by its widespread use in teaching environments, where

(a) it is used extensively to develop and teach algorithms to students.

(b) in both reading and writing programs, students need all the help they can get, in which case, well-defined and uniform layout rules should be of considerable benefit.

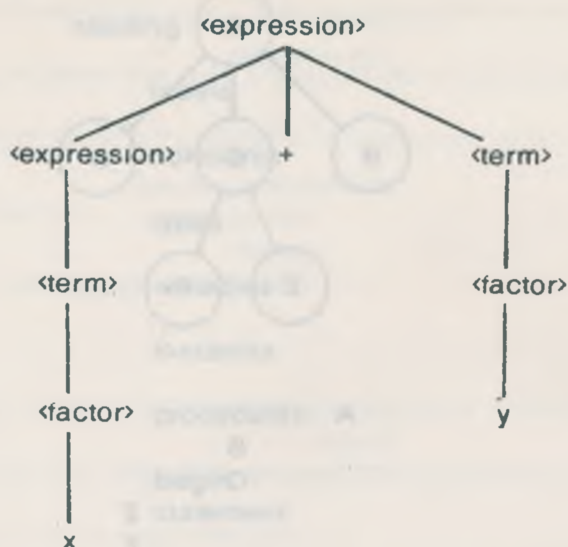
Of course, such benefits can also be seen to apply generally.

PROGRAM STRUCTURE

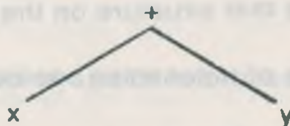
The semantic structure of a program i.e. the structure in terms of which its meaning is deduced, can be said to be its abstract syntax[2]. This may be represented by a tree structure with "operators" as nodes and "operands" as sub-trees. For example, the Pascal expression

$$x + y$$

has concrete syntax tree



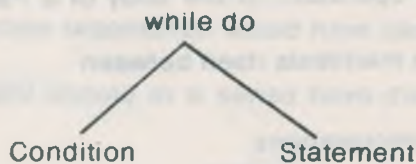
derived according to the syntax rules of the language, but has the abstract tree



conveying the meaning of the addition of x and y. Such treatment can of course be given to more "interesting" constructs. For example,

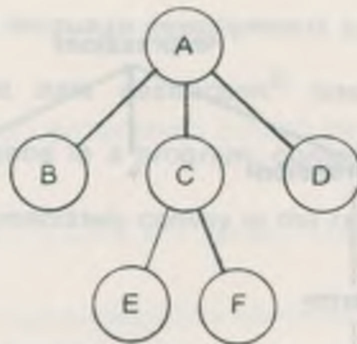
while Condition **do** Statement

can be said to have the abstract or semantic tree

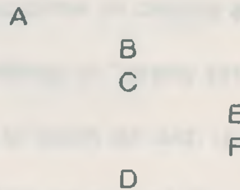


in which the "while do" operator has two operands "Condition" and "Statement" with the obvious meaning.

This capturing of the meaning of a program is fortunate, because the environment of program presentation, by lines on a page, admits to the clear and easy representation of tree structures by relative indentation. For example, the tree



appears as



The methodology for our treatment of Pascal, then, shall be to recognize a suitable abstract syntax, and to impose that structure on the concrete syntax as manifested in a particular program by the use of indentation. In detail, the various headers and keywords of the concrete syntax shall take the place of the operators of the abstract syntax.

PASCAL PROGRAM STRUCTURE

Siblings in the abstract tree are in the same way equivalently related under the influence of their associated operator. In the body of a Pascal program or procedure or function, this relationship manifests itself between

- label declarations
- constant declarations
- type declarations
- variable declarations
- procedure and function declarations
- the statements of the body

suggesting the following indentation scheme

```
heading
        labels
        constants
        types
        variables
        functions
        procedures
        begin
        statement
        .
        .
        .
        end.
```

Note that

(a) Function declarations are placed prior to procedures as a disciplined aid to program reading. "Forward" declarations can be used to overcome difficulties with the definition order.

(b) There is no indentation brought about by the presence of **begin** and **end**, which are meaningful only as (syntactic) delimiters that are chosen to group all the statements as a node in this level of the tree. Supposing that such a grouping was at all meaningful, then indentation would have been appropriate. However in our view, each statement should in a sense have direct access to the declarations.

(c) When a program or procedure or function contains sub-procedures or sub-functions, their bodies will be indented, so that only the user accessible part, the heading, will be available at the outer i.e. interesting, level of indentation e.g.

```
procedure first;
declarations
.
.
.
procedure second;
        declarations
        begin
        Statements for second
        .
        .
        .
        end;
begin
Statements for first
.
.
.
end;
```

so that reading the "Statements for first" to discover the meaning of accessed names, the reader has only to look along the current level of indentation, and only those declarations at that level will be worthy of interest, while those inside second, which are of no concern, are clearly moved out of the line of sight by indentation.

Finally, it should be noted that a program would conceivably appear as

```
program
    declarations
    begin
    Statements
    .
    .
    .
    end.
```

where the indentation of all but one line of the text is clearly redundant. Unless **goto** statements and labels are being used (see below), it is of course reasonable to omit the indicated indentation in this particular situation.

PASCAL DECLARATIONS

Noting that we have already covered procedures and functions, we see that we have recognized in our semantic tree the Pascal syntax scheme of grouping labels, con–

stants, type and variable declarations. This is simply because we hold that this is a matter of the sensible organisation and classification of declaratives whose individual texts are physically insignificant for purposes of comprehension.

Label declarations appear as

```
label
        l1, l2, l3, ... ln;
```

where each l_i is a valid label. The keyword appears as a line by itself, corresponding to the semantic operator, the operands are on the succeeding line and are indented. Canonically, they should be on separate lines i.e.

```
label
        l1,
        l2,
        l3,
        .
        .
        .
        ln;
```

but the brevity of each label makes the previous proposal the more pragmatically sound.

The canonical arrangement is more correctly shown with respect to constant definitions:

```
const
        Name1        = C1;
        .
        .
        .
        Namen        = Cn;
```

We further make the pragmatic observation that the "=" delimeters be aligned for clarity, which may easily be achieved by using a <tab> character or characters, just as can be used to effect indentation. In this regard it is felt that all indentations should be of equal width. Capital letters may also be used for the "name" so that constants may be clearly identifiable wherever they may appear. For example

```
const
      BLANKS          = '       ';
      MAXHEIGHT       = 6;
      MINWEIGHT       = 100;
```

With types, schematically

```
type
      Name1        = T1;
      .
      .
      .
      Namen        = Tn;
```

a new item of interest is introduced: record definition, including variant records. Because the expression of a type definition, the T_i above, is already indented, there need be no further indentation for **record** ... **end** e.g.

```
type
      man  = record
             age        : integer;
             address    : array [1..60] of char
             end;
```

The discussion of variant records shall be treated under the "case" statement below. In a similar manner the schematic for "var" declarations would be:

```
var
      Name1       : T1;
      .
      .
      .
      Namen       : Tm;
```

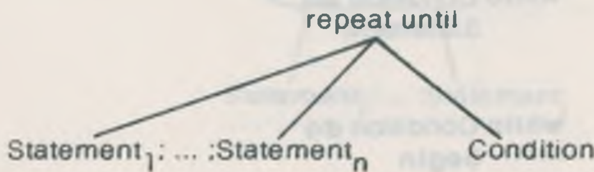
To clearly distinguish variable names from constants lower case letters only should be used e.g.

```
var
      eyecolor     : string;
      height       : integer;
      weight       : integer;
```

PASCAL STATEMENTS

We have already shown how the compound statement which is the body of a program, procedure or function is not indented with respect to **begin** ... **end** implying that for consistency the structuring operations which can contain compound statements be regarded as n-ary operators.

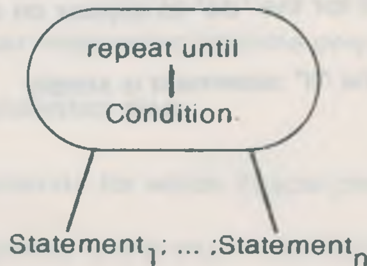
This is shown rather well when contemplating the Pascal iterative constructs. For the "repeat" statement, we may perhaps have semantically



representable by

repeat
    $Statement_1$;
    .
    .
    .
    $Statement_n$
until
    Condition

Note how the keywords representing the semantic operator are not indented, but that the operands are. By no great stretch of the imagination, it is possible to conceive of the "repeat until" operator as being "curried" i.e. it is applied to the "Condition", producing a new semantic operator, which can be then applied to $Statement_1$, ... ,$Statement_n$. Diagrammatically



which becomes

```
repeat
    Statement1;
    .
    .
    .
    Statementn
until Condition
```

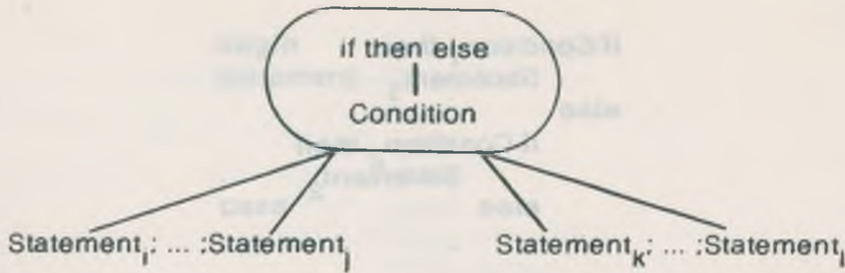
i.e. the simple initial application of **repeat** ... **until** to the Condition deserves no special indentation.

Formats for the "while" and "for" statements are then easily derived:

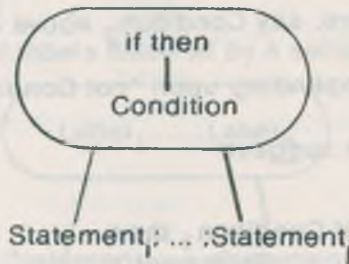
```
while Condition do
    Statement


while Condition do
    begin
    Statement1;
    .
    .
    .
    Statementn
    end


for Index := Expression1 to Expression2 do  [or downto]
    Statement


for Index := Expression1 to Expression2 do
    begin
    Statement1;
    .
    .
    .
    Statementn
    end
```

Note that there is no reason for the "do" to appear on an individual line.

With regard to branching, the "if" statement is simply



or



yielding

```
if Condition then
        Statement_i;
        .
        .
        .
        Statement_j
        end
```

and perhaps

```
else
        begin
        Statement_k;
        .
        .
        .
        Statement_l
        end
```

with **begin** ... **end** optional in the case of a single Statement. Clearly the tree structure has two levels, whereas our indentation scheme only has one, because of the fortuitous placement of the non-indented **else**.

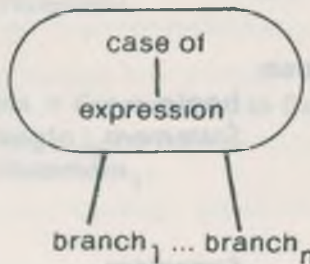
An important semantic phenomenon for which Pascal provides only in the limited way of the "case" statement (see below) is the multi-way branch, which must be simulated in Pascal by a series of nested "if" statements:

```
if Condition₁ then
          Statement₁
else
          if Condition₂ then
                    Statement₂
          else
                    .
                    .
                    .
```

Therefore, in the case where, say Condition_2 above does not represent a condition of the state of the machine depending upon "not Condition_1" but represents an alternative to "Condition_1", then we suggest:

```
if Condition₁ then
          Statement₁
else
if Condition₂ then
          Statement₂
else
          .
          .
          .
```

The "case" statement is semantically



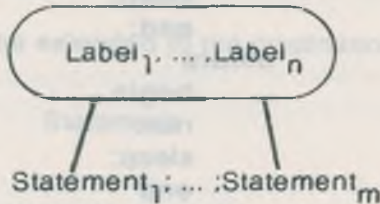
and therefore gives

```
case Expression of
          branch₁
          .
          .
          .
          branchₙ
          end
```

the closing "end" is indented to avoid

```
begin
Statement;
.
.
.
case
   .
   .
   .
end
end
```

Each branch in detail is a list of labels followed by a series of statements



i.e.

```
Label1, ... ,Labeln :
         begin
         Statement1;
         .
         .
         .
         Statementm
         end
```

An "others" or "default" branch that some versions of Pascal provide may be treated as one of these e.g.

```
others :
         begin
         Statement1;
         .
         .
         .
         Statementk;
         end
```

For example

```
case day of
          monday, tuesday, wednesday, thursday :
                    begin
                    rise;
                    work;
                    sleep;
                    end;
          friday :
                    begin
                    rise;
                    work;
                    drink;
                    sleep;
                    end;
          others :
                    begin
                    rise;
                    sleep;
                    end
          end;
```

The format of the variant record declaration may be derived from the format of the "case" statement. The overall structure is

```
case tag_field type_identifier of
          Variant_1;
          .
          .
          .
          Variant_n
```

(note the absence of a terminating **end**). Each variant appears likewise as

```
label_1, ... ,label_n :
          body
```

where body is structured as the declaration of procedure or function formal parameters:

```
(Name_1 : Type_1; ... ;Name_n : Type_n)
```

The potential problem of exceeding the right margin is discussed below in the context of parameter declarations.

As an example:

```
type
          planet = (venus, mars, pluto);

          alien   = record
                      age    : integer;
                      weight  : real
                      case origin : planet of
                             venus, mars :
                                     (arms, heads, legs : integer;
                                      married : boolean);
                             pluto :
                                     (wheels, gears, cogs : integer)
                      end;
```

The indentation of labels can be extended to the destination of a "goto" e.g.

```
              Statement
              .
              .
              .
       99 :   Statement
              .
              .
              .
              goto 99
              .
              .
              .
```

We are theoretically guaranteed to have the space to make the label outstanding (the body of an entire program may need to be indented in this case).

Finally, of structured statements, is the "with" statement, which is obviously

```
       with Record do
              begin
              Statement1;
              .
              .
              .
              Statementn
              end
```

The remaining statements are the "goto", "assignment", and "procedure call" which are not part of this discussion because they do not involve control structure semantic operators.

MISCELLANY

A sound pragmatic point concerns the lines which exceed some extreme right margin, determined by paper, screen or physical device width, which occur often in the context of procedure and function parameter specifications and calls, and whose frequency increases as the left margin is indented further.

For procedure and function parameters, the following scheme is proposed. Instead of an horizontal decomposition, we propose

```
procedure P ( formal-parameter1,
              .
              .
              .
              formal-parametern );
```

for a procedure heading, and

```
function F ( formal-parameter1,
             .
             .
             .
             formal-parametern )
             : function-type;
```

for a function heading, and

```
PF ( actual-parameter1,
     .
     .
     .
     actual-parametern);
```

for a procedure or function call. This is done purely for the sake of aesthetics. For example

```
procedure readyfile( filename          : namestr;
                     var numberoffields : integer;
                     var field          : fieldtype);
```

Should an entity (typically, part of a long expression, or perhaps part of a formal parameter specification) exceed the right margin, a break is simply made at the last blank part of the right margin and continuation proceeds from the current indent position on the next line e.g.

```
verylong := ( longer * verylong ) /
                          summationofvariables;
```

Note how the continued expression lines align from the left, and not the position of the control line.

Finally, we remark that the advantages of horizontal spacing, particularly between groups of declarations, cannot be overestimated, and that similarly mandatory should be a comment explaining the nature of a procedure, function or program after its heading.

PREVIOUSLY PROPOSED SCHEMES

The merits of our proposal, apart from its semantical foundation and beauty, can be demonstrated by discussing some previously proposed schemes.

Chronologically, the first published formatting scheme for Pascal is that which could be deduced from the pages of Wirth[4], which is that presumably called the "classical" style[5]. While the examples presented are rather inconsistent in indentation, nonetheless the following trends are observed:

(a) the general structure of a program/procedure/function is

```
heading
     declarations
begin
     statements
end
```

which vaguely corresponds to ours save for the irrelevant prominence of the **begin** ... **end** brackets.

(b) the bodies of loops and trailers tend to be indented; if they are single statements they sometimes appear on the same line as the semantic operator e.g.

```
if Condition then Statement
else Statement
```

and the indentation of a compound statement body is apparently a consequence of **begin** ... **end** influence e.g.

```
if Condition then
begin
          Statement
             .
             .
             .
end
```

However, we repeat that such rules are deductions, and require clarification and improvement – the currently described piece of work can be thought of as such. Some concrete proposals can be found in Singer, Hueras and Ledgard[6]. Their merit is the emphasis on use of vertical spacing to aid clarity, and on the definition of various disciplines regarding interfacing of sub-programs to their environments (such issues are regarded as being outside the scope of the present work). On the other hand

(a) not much is said about the indentation of nested procedure definitions. Is the following forbidden?

```
procedure A
procedure B
begin
          Statement
             .
             .
             .
end;
begin
          Statement
             .
             .
             .
end
```

(b) while the bodies of loops etc are to be indented, so are the subordinates of a **begin** ... **end** which grouping we have shown to be semantically insignificant, always occurring as part of a more meaningful set. The present scheme allows double indentation

```
while Condition do
begin
        Statement
           .
           .
           .
end
```

which can only lead to a quicker than necessary confrontation with the right margin. Similarly superfluous is the required indentation of the detail of a record definition between **record** and **end**. The detail of the indentation of the bodies of a "case" statement with respect to the case labels (and labels in general) is unspecified. Finally, the **then** and **else** branches of the "if" are given yet more indentation e.g.

```
if Condition
  then
        begin
              Statement
                 .
                 .
                 .
        end
  else
        begin
              Statement
                 .
                 .
                 .
        end
```

which we suggest demonstrates the lack of a formal correspondence between this scheme and the abstract syntax of its object programs.

A further set of rules in Peterson[7] which does not address the general methodological questions we felt the latter did so well echoes its mistakes with regard to indentation. Notable is a lack of specification of a number of concepts and relationships such as nested procedure declarations (which is an absolutely vital issue in the environment of top-down development) and the layout of some statements and labels. Present again is the idea that **begin** ... **end** is *semantically* significant (which significance is represented by indentations). Even more obscure is the way in which multiple levels of

indentation are required in the bodies of the control constants, for example

```
if Condition
   then begin
                    Statement
                    .
                    .
                    .
        end
   else begin
                    Statement
                    .
                    .
        end
```

and

```
while Condition
   do begin
                 Statement
                 .
                 .
                 .
      end
```

which is faulty in that

(a) there is an excess of syntactic detail and complex indentation to introduce some essentially simple concepts.

(b) indentation to the right proceeds excessively quickly.

(c) the right-justification of **while** and **do** must be hard to effect, similarly, the indentations are not uniform, but depend upon the relevant semantic operator.

An interesting scheme[5] addresses the issue of indentation in addition to the irritation caused to the Pascal programmer by having to remember the trivial details of placement of semicolons and the need to insert **begin** ... **end** around compound statements as bodies of such as **while**'s and **if**'s. The latter is solved by the elegant expedient of making **begin** and **end** part of every structured construct, and, because Pascal allows an empty statement, placing a semicolon after every statement (i.e. before "**end**" - we see that a semicolon can never occur before an else, though). For example, we write

```
if Condition then begin
    Statement
end else begin
    Statement
end
```

where the Statement can be a simple or multiple statement without fear. The scheme admittedly addresses only part of the layout problem, and could be incorporated into our framework as an alternative to our particular formulations in this regard. It is after much consideration that we choose not to, because

(a) the fate of the outer **begin** ... **end** for the body of a program/procedure/function is not simply and uniformly accounted for (it causes an indent of its own in the scheme referred to)

(b) we are in a sense changing the language, hypothesising a new concrete syntax along the lines of

```
if Condition then
    Statement
    .
    .
    .
else
    Statement
    .
    .
    .
fi
```

and implementing it in terms of combinations of elements of the Pascal syntax; we choose not to impose this burden on the programmer believing our scheme aids layout just as effectively and in a more familiar and natural way.

The work of Singer et al[6] and Peterson[7] is further discussed by Mohilner[8] with respect to the "right margin" problem. We feel that our solution to this problem is on firmer foundations. Oppen[9] discusses pretty printing from an implementation viewpoint rather than discussing a basic formatting policy for particular styles of language.

Our final example is an analysis by Rose and Welsh[10] of a version of Pascal with inbuilt formatting rules as part of the syntax, avoiding the need for **begin** ... **end**

delimiters and semicolons and the **end** terminator (of **case** and **record** entities). We see the basic principles of our design appearing, but of course without the need to consider the placement of **begin** and **end** (the possibility of the omission of which, as this work demonstrates, proves their uselessness as a semantically meaningful construct). Details with which we quibble are

(a) the layout of the "if" statement as

```
if Condition
then
        Statement
else
        Statement
```

there being no reason for the placing of the "then" on a single line.

(b) sometimes we then see

```
if Condition
then Statement
else
        Statement1;
        .
        .
        .
        Statementn
```

which masks the "then Statement" and similarly in the case of an "else Statement".

(c) multiple statements per line, which mars clarity

(d) similarly, (case) labels and statements on the same line.

In summary these proposals are sound, particularly with respect to the way in which they agree with ours about the nested structure of declarations, but we claim the relative merit of our work is in the accommodation of the actual Pascal syntax, and in the strictness of our approach with respect to the vertical separation of text.

CONCLUSION

Ease of understanding of a program is facilitated by a clear exposition of its semantic tree structure, which can be achieved by indentation. A formatting discipline for

Pascal based upon indentation to reflect the semantic structure of programs has been developed. A comparison with other schemes of varying quality has been attempted, the results of which comparison by themselves command the proposed scheme. The scheme is simple, straightforward and semantically elegant.

EXAMPLE

```
program example (input, output);
{ example -- reads a list of not greater than 1000 numbers
            and outputs them in ascending order }
const
        LIMIT   = 1000; { upper bound on number of items read }
type
        minrange       = 1..LIMIT; { index type for table }
var
        numbersread  : 0..LIMIT; { counter for numbers read }
        table            : array [minrange] of integer; { number storage }
        i                   : minrange; { loop index }

procedure smallfrom (base : minrange);
{ smallfrom -- select the smallest number in the range "base" to
               "numbers-read" }
        var
                min      : minrange; { index of smallest number found }

        function indexsmallfrom (base : minrange) : integer;
        { indexsmallfrom -- integer function to give the position of the
                            smallest number starting at position "base"}
                var
                        index   : minrange; { index of smallest number found so far }
                begin
                if base = numbersread then
                        indexsmallfrom := base
                else
                        begin
                        index := indexsmallfrom (base + 1);
                        if table [base] < table [index] then
                                indexsmallfrom := base
                        else
                                indexsmallfrom := index
                        end
                end;
        begin
        min := indexsmallfrom (base);
        writeln (table [min]);
        table [min] := table [base]
        end;
begin
numbersread := 0;
while not eof do
        begin
        numbersread := numbersread + 1;
        readln (table [numbersread])
        end;
for i := 1 to numbersread do
        smallfrom (i)
end.
```

ACKNOWLEDGEMENTS

One of the authors (A.S.) wishes to thank Professor J. Reinfelds, the staff and faculty of the University of Wollongong for their support and kind hospitality whilst he was a visitor in the department.

REFERENCES

1. O.J. Dahl, E.W. Dijkstra and C.A.R. Hoare, Structured Programming, Academic Press, 1972.
2. J.V. Guttag, E. Horowitz and D.R. Musser, 'The design of data type specifications', in Current Trends in Programming Methodology, **4**, 60–79(1978).
3. K. Jensen and N. Wirth, Pascal – User Manual and Report, 2nd edn., Springer-Verlag, New York, 1975.
4. N. Wirth, Algorithms + Data Structures = Programs, Prentice Hall, Englewood Cliffs, 1976.
5. A. Sale, 'Stylistics in Languages with Compound Statements', Aust. Comp. J., **10**, 2, 58–59(1978).
6. A. Singer, J. Hueras and H. Ledgard, 'A Basis for Executing Pascal Programmers', SIGPLAN notices, **12**, 7, 101–105(1977).
7. J.L. Peterson, 'On the Formatting of Pascal Programs', SIGPLAN notices, **12**, 12, 83–86(1977).
8. P. R. Mohilner, 'Prettyprinting Pascal Programs', SIGPLAN notices, **13**, 7, 34–40(1978).
9. D.C. Oppen, 'Prettyprinting', ACM Trans. on Prog. Lang. and Sys., **2**, 4, 465–483(1980).
10. G. Rose and J. Welsh, 'Formatted Programming Languages', Software – Practice and Experience, **11**, 651–669(1981).