SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 13, 163–179 (1983)

A Specification Schema for Indenting Programs

PRABHAKER MATETI*

Department of Computer Science, University of Melbourne, Parkville, Victoria, Australia

SUMMARY

A two level specification of the functional behaviour of a class of indenting programs for Pascal is presented. The transformation that these programs perform on the input text is a composition of splitting input lines, altering the blank space between lexical tokens and computing the margin required in front of each of the split lines. The high level specification is given as a stylized Pascal grammar in Extended BNF. In contrast, the low level specifications, which are operationally closer to a program, and which define how syntactically invalid text is dealt with, require several mathematical functions that capture the essence of these basic transformations. The specifications of an indenting program for Pascal are then obtained as a further elaboration of these functions. Most indentation styles appearing in the literature can be specified with precision using methods developed in this paper. Our experience in this case study indicates that although specifications for real-life programs can be given using simple mathematics, the effort required is still considerable.

KEY WORDS Functional specifications Correctness proofs Pretty-printing Pascal

PREFACE

The present paper is one of a triplet on an indenting program for Pascal. We undertook this exercise with three objectives in mind:

1. The literature sadly lacks real-life programs whose correctness is established by proof rather than by testing. On the other hand, those who have practised proving correctness have been raising the hopes of the readers to such an extent that a single mistake in a published proof gets the widest adverse publicity. We hope that our indenting program and its specifications and proof will serve as examples in this regard.
2. The practising programmer, we find, often uses the lowest level of formalism whereas a student who has just been through correctness methods employs formidable notation and an excess of formalism. The right level for a given program escapes both. It is not easy to say what is a right level. This can only be communicated through examples.
3. There is a myth that giving precise specifications for 'real-life' programs is often not possible. We are quite willing to accept this as a definition of 'real-life' programs but not as a corollary. Another myth is to equate precision with formalism. We hope that these papers will serve as examples where sufficient precision is attained with very little formalism.

Only the reader can tell how far we succeed in fulfilling our objectives.

* Present address: Department of Computer Engineering, Case Western Reserve University, Cleveland, OH 44106, U.S.A.

0038–0644/83/020163–17$01.70 *Received 15 September 1980*