1. INTRODUCTION

That written material expected to be read by humans should be laid out with thought and care is widely appreciated. Yet the layout of many computer programs is poor. To make matters worse, programs written in modern programming languages have many nested levels of control structures and declarations. While compilers for these languages accept 'free-format' input and can distinguish the nesting regardless of how the text input is laid out, most humans are yet to adapt themselves in this fashion.

Laying out the text of a program so that its structure is readily apparent has come to be called 'pretty-printing'. Many sets of rules for pretty-printing exist (e.g. References 1–3). These rules range from such typewriting conventions as always following a comma by a blank and flanking an equality sign by blanks to insisting that reserved words such as **goto** appear only at the beginning of a line and never hidden somewhere in the middle of a line. Much of the work in the layout of a program text is routine once a set of systematic pretty-printing rules is chosen. In fact, several programs that pretty-print the given input exist.

In this paper, we limit ourselves to programs written for Pascal, and use the less pretentious word 'indenting' in preference to 'pretty-printing'. We develop the basic mathematical functions required to specify precisely the input-to-output transformation performed by a class of indenting programs. The companion paper[4] proves the correctness of an indenting program meeting the specifications developed here, and Reference 5 discusses global issues about the program.

The indentation scheme embodied in our specifications below has evolved over a period of years accommodating and adapting the many schemes proposed in the literature. The author finds it satisfactory but is aware of others who do not. The goal of this paper is not to promote this scheme but to show that specification for such programs can be developed with sufficient precision employing simple mathematical notions. Section 2 discusses our expectations of indenting programs. Section 3 establishes notation. Section 4 gives the input-to-output transformation performed by these programs using the syntax definition of Pascal. Section 5 specifies the transformation independently of this syntax definition. Section 6 shows that if the input file contains a legal construct of Pascal then the specifications of Sections 4 and 5 are equivalent.

2. WHAT SHOULD INDENTING PROGRAMS DO?

The specifications of a program are simply our requirements and expectations of it but stated precisely without ambiguity. We ignore certain specifications of a program such as that its length be so much, or that it be written in language X without **goto**s. Instead we will concentrate only on the relationship between the input and output of indenting programs. Such specifications are called functional specifications.[6]

We list some of our expectations of indenting programs below.

1. The most obvious and yet oft-forgotten requirement is that the output of an indenting program should be 'lexically equivalent' to the text input given. Should indenting programs accept only syntactically correct text? No. We believe that indenting programs must accept any text input; if the input happens to be a syntactically correct program, we then expect its output to be properly indented. And if the input is not syntactically correct, the output text should be indented as reasonably as possible. A notion of reasonableness underlies our low