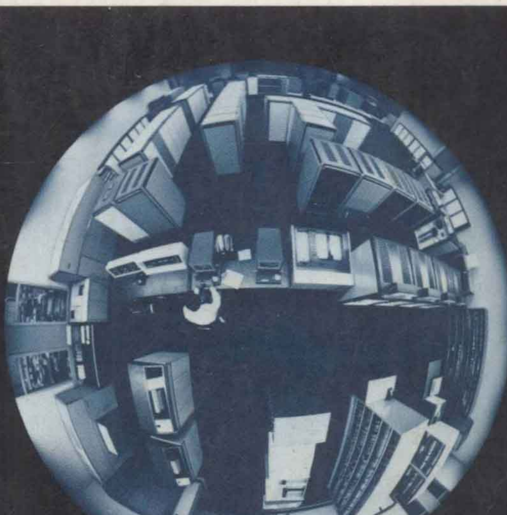
Fundamentals of the Computing Sciences

Supplementary Volume

Kurt Maly
Allen R. Hanson



Fundamentals of the Computing Sciences

Supplementary Volume

Kurt Maly
University of Minnesota

Allen R. Hanson
Hampshire College

Prentice-Hall, Inc.
Englewood Cliffs, New Jersey 07632

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

PREFACE

The material covered in this volume is intended to complement and extend the material in Fundamentals of the Computing Sciences. It is divided into four sections:

A. Introduction to FORTRAN
B. Introduction to SNOBOL
C. Advanced Topics
D. Solutions to the Exercises

As a companion to the basic text, it is designed to provide coverage of the languages used there and to extend a number of the topics covered. The contents of each section and the relationship of that content to the main text are summarized below. Each of the sections is self-contained and may be used independently of each other.

The language sections (A and B) are basic introductions to FORTRAN and SNOBOL, respectively. The level of these introductions is sufficient to provide the background necessary to understand the programs in the text and to do the exercises. In the case of the FORTRAN section, the presentation conforms to ANSI Standard FORTRAN; deviations from the standardized language are clearly marked as such, for example, the processing of non-numeric information. For SNOBOL, the presentation follows closely the SNOBOL4 language as described in Griswold, Poage, and Polonsky, The SNOBOL4 Programming Language, 2nd Edition (Prentice-Hall, 1971). Since each section is designed to follow the text, certain features of the languages may be emphasized more heavily than others; examples tend to emphasize features useful for understanding the main text. These sections can be used to provide the basic material for a programming laboratory accompanying the main course, and can be supplemented with additional material and/or local programming manuals.

The Advanced Topics section extends a number of chapters in the main text to more advanced material, parts of which require some mathematical maturity. The structure of this section follows the structure of the text; sections are numbered according to the chapter and section in the main book to which they pertain. In some sections derivations of results stated in the text are provided, while in others new material is presented. In many cases, natural extensions of the material covered here may be used as the basis for group projects.

The solutions to a large fraction of the exercises presented in the text are provided in the final section. In most cases, a complete solution is given; in others, solutions are either started or hints are given as to how to start.

Appendix 1 lists the ASCII character set and control codes, along with their decimal, binary, and octal equivalents, Appendix 2 contains a summary listing of the procedures, algorithms, and programs appearing in this volume and the main text.

KURT MALY
ALLEN R. HANSON

Table of Contents for Supplemental Manual

PART A   INTRODUCTION TO FORTRAN

A.0 A Historical Perspective

Unlike AL (cf. Chapters 1-3), which was not specifically designed as a *computer* language, FORTRAN (from FORmula TRANslation) is an actual programming language, whose name refers to two distinct entities. The first of these is the language itself, comprising the set of instructions and their meanings. The second is the *compiler*, which is a program that accepts as input FORTRAN programs and which produces as output an equivalent set of machine language instructions which are directly executable on the specific computer being used. In this sense, the compiler is equivalent to the Turing machine programs underlying AL; it provides the semantics of the FORTRAN statements. It is an interesting fact that not only is FORTRAN one of the most widely used programming languages today but also almost every computer that has ever been built has included FORTRAN in its software library. As a result, the term FORTRAN is actually a generic name, representing many different, though markedly similar, languages. There are many dialects of FORTRAN, almost as many as there are computing installations since all make some changes to the basic language. As a result, programs written in FORTRAN at one installation often do not run at another installation. Furthermore, the restrictions on what can and cannot be done in the language differ from installation to installation. To understand these differences, and the very nature of FORTRAN itself, it will be instructive to digress for a moment and consider the historical development of programming languages, particularly FORTRAN.

The computer was developed during the 1940s as the result of a need for devices which would carry out the solutions to numerical problems. These problems were characterized by vast numbers of repetitive operations and very little input/output. The first computers were programmed in the language of the machine itself, often binary sequences representing instructions and data. As we have seen from the considerations of Chapters 1 and 2, such a language is not ideally suited for human use; a single misplaced digit in one of the sequences is sufficient to destroy program functioning. It was a reasonable quest to consider methods by which communication between man and machine could take place at a level more suited to the needs of the human, rather than the machine. As a result, several projects were begun to develop *symbolic* programming languages.

Early in the 1950s, such a project was begun at IBM under the direction of John Backus. In 1956, the first reference manual describing the FORTRAN language for the IBM 704 was released; shortly after, the first commercial version of FORTRAN was made available. This was followed by FORTRAN II in 1958. For the IBM 704, FORTRAN II contained some additional features, most notably subroutines and functions (procedures). After an initial period of skepticism and hostility, the concept of FORTRAN caught on, and other manufacturers began supplying versions of FORTRAN for their own machines. Soon FORTRAN became generally accepted and widespread. It is interesting to note that since FORTRAN was reasonably easy to learn and use, and since it did not require an intimate knowledge of the machine on which it ran, the use of languages such as this extended access to computers to scientists, engineers, and computer specialists alike.

Because of the rapid proliferation of FORTRAN, little thought was given to standardization; however, in the early 1960s the state of affairs had gotten out of hand. A committee was formed to standardize the basic features of the language and provide some semblance of machine independence. In 1962, a report was released describing a language now known as FORTRAN IV (a version of FORTRAN known as FORTRAN III had a very short life). In 1962, the American Standards Association (ASA) standardized two versions of FORTRAN, known as FORTRAN and Basic FORTRAN, such that Basic FORTRAN formed a proper subset of FORTRAN. This means that any Basic FORTRAN program will execute under a FORTRAN compiler (but not vice versa). FORTRAN as we know it today corresponds to FORTRAN IV, while Basic FORTRAN is roughly equivalent to FORTRAN II.

An important point to consider is the structure of a computing system as it existed during the formation of FORTRAN. A computer installation typically consists of the computer proper (including the central processing unit, arithmetic unit, and main memory) and various peripheral devices. The peripherals usually consist of input/output and bulk storage devices. Typical I/O devices include card readers and card punches, high-speed line printers, and teletype terminals of some kind. Bulk storage devices consist of magnetic tape units, magnetic drums, and others. Modern computer installations are considerably different in terms of device technology and type, but this simple model will suffice. As a result of this structure, FORTRAN was very early locked into an instruction set which represented those operations normally required by the complement of their available peripheral devices. Consequently, FORTRAN at times appears considerably out of touch with modern developments, particularly time-shared computer systems. On the other hand, it is remarkable how successful the language has been and how adaptable it has become.

A.1 FORTRAN: An Overview

The FORTRAN which will be presented in this chapter is the standard FORTRAN mentioned in the previous section. In any actual programming language, the programmer must make concessions to the structure of the machine. Usually these concessions include a fixed format for program statements, a restricted alphabet, and a small set of statements from which to build up programs. In addition, the types of data which may be used and the ranges of each are very often fixed. For example, the maximum and minimum integer values that can be stored are a function of the *word size* of the particular machine which will be used to execute the program. Word size refers to the number of bits an addressable memory location in the machine contains. Once these concessions become familiar, they do not present any real obstacle to the development of programs for a specific problem. They may, however, require us to alter the algorithm developed in order to translate it into a program.

Firmly embedded in the structure of FORTRAN is the idea of a *card image*. All program statements and data must be tailored to fit this concept. This restriction is not surprising since FORTRAN was originally developed as a *batch* language; all programs which are to be run on the computer are in the form of card decks. These decks are submitted to the computer in batches--hence the name. A large percentage of modern computing is still batch oriented. Figure A.1 illustrates the standard FORTRAN punched card with the FORTRAN character set punched; this character set contains the 26 capital letters, the 10 digits, and the characters +, -, *, /, (,), =, .(period), , (comma), blank, and $. The card itself is 80 columns wide, each column can contain one symbol from the character set.

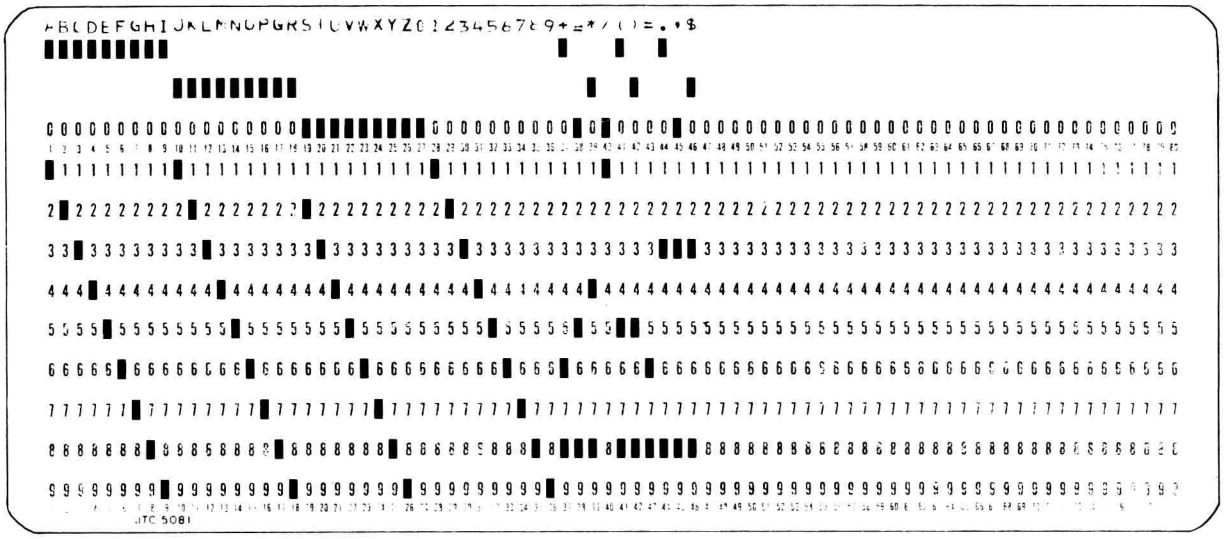


Figure A.1 The Punched Card

The holes in the card represent the *code* assigned to the symbol in that column. The holes are sensed by the card reader and converted to a sequence of electrical pulses which are sent to the computer. The printing on the top of the card is of no importance to the computer; it is only for our convenience. Certain groups of columns have significance. Columns 1-5 are reserved for a statement label, similar to the transfer addresses in the Turing machine language, which serves to uniquely identify the FORTRAN statement contained on the line if we so desire. Each statement label is an unsigned positive integer of from 1 to 5 digits. If the first column contains a C, then the remainder of the card is treated as a comment and is ignored by FORTRAN; this corresponds to the /* ... */ delimiters in AL. In the event a statement requires more than one card, it may be continued onto the next card by writing a nonzero, nonblank character in column 6 of the continuation card, up to a total of 19 continuations. The group of columns beginning at column 7 up to and including column 72 is reserved for the FORTRAN statement. In this field, blanks are completely ignored (with certain exceptions which will be discussed later); however, inclusion of blanks makes the statements more readable; hence, they always should be used to punctuate words and the like. Columns 73-80 are completely ignored by the FORTRAN compiler; they are commonly used for program identification and card sequencing. There are multiple horror stories concerning several 1000-card FORTRAN programs being accidentally dropped and shuffled without any practical way of restoring the order. Therefore, it is advisable to use these columns to number the cards sequentially.

When describing a programming language, there are two general categories of facilities which must be considered. One of these concerns the data types allowed by the language (e.g., integers, real numbers, complex numbers) and the ways in which the data may be structured (e.g., simple objects such as constants and compound objects such as n-tuples, tables). The second concerns the set of instructions and operators included in the language and their relation to the data types supported. One convenient format for the exploration of a language is to partition the instruction set into sets of logically equivalent instructions. Pursuing this format, we shall consider the following distinct sets in the listed order:

1. *The language elements:* These include the types of constants allowed, the definition of a variable, and the kinds of data structures supported.

2. *Expressions and basic statements:* Here we consider the structure of allowed expressions, the operators provided, and the hierarchy of operators.

3. *Specification statements:* Some statements in FORTRAN are not language elements as such; rather, they provide information to the FORTRAN compiler concerning the structure of variables, their types, locations, etc.

4. *Control statements:* These statements are the ones which provide us with control over the sequence of execution of statements. Normally execution is sequential beginning with the first statement in the program. Control statements allow us to alter this implied sequence.

5. *Input/output statements:* These provide means for communications between the program and the outside world.

6. *Program segmentation:* FORTRAN provides a variety of means for segmenting programs, including subroutines and functions.

7. *Miscellaneous features:* Many installations provide extensions above and beyond the standards set forth in the ASA definition of FORTRAN. We shall make an effort to studiously avoid such features, except when necessary.

A.2 FORTRAN Language Elements

The most primitive element in any programming language is the constant. Since different types of constants are represented in a different manner inside the computer, the different kinds of data types to be allowed have to be defined; they are

```
INTEGER
REAL
COMPLEX                                              (1)
DOUBLE PRECISION
LOGICAL
```

All constants (and, as we shall see, all variables) are required to be one of these five types. Each type has specific characteristics. The ranges of values for the first four types in (1) are not defined within FORTRAN proper but depend on the computer on which a FORTRAN program is to execute. They can, however, be expressed in terms of computer-dependent parameters.

1. *Integer constants:* An integer constant is a string of digits, without a decimal point, immediately preceded by a sign: + (optional) or - . The range of admissible values is usually expressed as

   $$-m_1 \leq i \leq n_1$$

   where m_1 and n_1 are positive integers. For a machine with 48 bits per word of storage, typically

   $$m_1 = n_1 = 2^{47} - 1$$

   since one bit is reserved for the sign.

2. *Real constants:* A real constant is represented as a string of decimal digits containing an embedded decimal point with a preceding sign: + (optional) or - . Examples of real numbers are

-.874563	2345672.98674625
1.67843	.00008623
100.	-78.987
+100	-1.0

   An alternative representation of real numbers is permitted: a real number as defined above, followed by the letter E (for exponent), followed by an integer constant. This is called the *engineering* (or *scientific*) representation; examples include

-28.3546E-10	+98.E7
1.E2	-98.E-7
.00089E93	-98.E+7
-9.457E+23	+98.E+7

   To provide a uniform description of real numbers, every real number is represented in terms of a *mantissa* and an *exponent*. The mantissa of any real number is composed of the relevant digits of the number preceded by the decimal point. The exponent is the number of places this decimal point must be shifted to obtain the number. For example, let the number be 256.12; the mantissa is then .25612, and the exponent is 3 since

   $$.25612 * 10^3 = 256.12$$

   Using this representation, the range of real values may be described using the three parameters m_2, n_2, and t:

$-m_1 \leq \text{exponent} \leq n_2$

number of relevant digits = t

3. *Complex constants:* A complex number is represented conceptually as $x + iy$ where x and y are the real and imaginary parts, respectively. A complex constant is represented as two real values enclosed in parentheses and separated by a comma. The range of each part of the complex number is the same as the range of a real constant; examples include

(1.3,-78.99)	representing	$1.3 - i78.99$
(3.8E-1,.85)	representing	$.38 + i.85$
(-8.,+6)	representing	$-8. + i6.$

4. *Double precision:* In some applications, the range of real values may be too small; inclusion of the double precision type allows us approximately to double the range of the real constant. In other words, type double precision provides a mantissa of approximately twice the number of relevant digits as that of a real number. When double precision numbers are written in engineering notation, the exponent symbol (E) is replaced by a D. Numbers of fewer than a minimum number of digits *must* contain the D in order to distinguish them from ordinary real constants. Examples are

   98.85D0

   9856362547.67483 (minimum number of digits for standard notation is machine dependent)

   .8746D-8

5. *Logical:* The range of values for the type logical is computer independent and consists of only the two values:

   .TRUE.

   .FALSE.

Variables in FORTRAN are similar in concept to AL variables, though highly restricted. They are the symbolic names given to a quantity which may change in value during the execution of a program. The value assigned to a variable must be one of the five types discussed above; the concept of a variable type is then well defined: integer variables may be assigned integer values, real variables may be assigned real values, etc. In addition to the type of a variable, we must take into consideration the different structures which a variable may have. FORTRAN allows both the simple variable whose value at any instant is a single constant (which may be computed) and the compound variable whose value is actually a set of values, each of the same type. The only compound variable which FORTRAN allows is the multidimensional array with from one to three dimensions. A one-dimensional array is similar to the tuple, a two-dimensional array corresponds to a table, while a three-dimensional array is best visualized as shown in Figure A.2. A compound variable is also known as a subscripted variable. As we shall see in Section A.4, various information concerning the types of