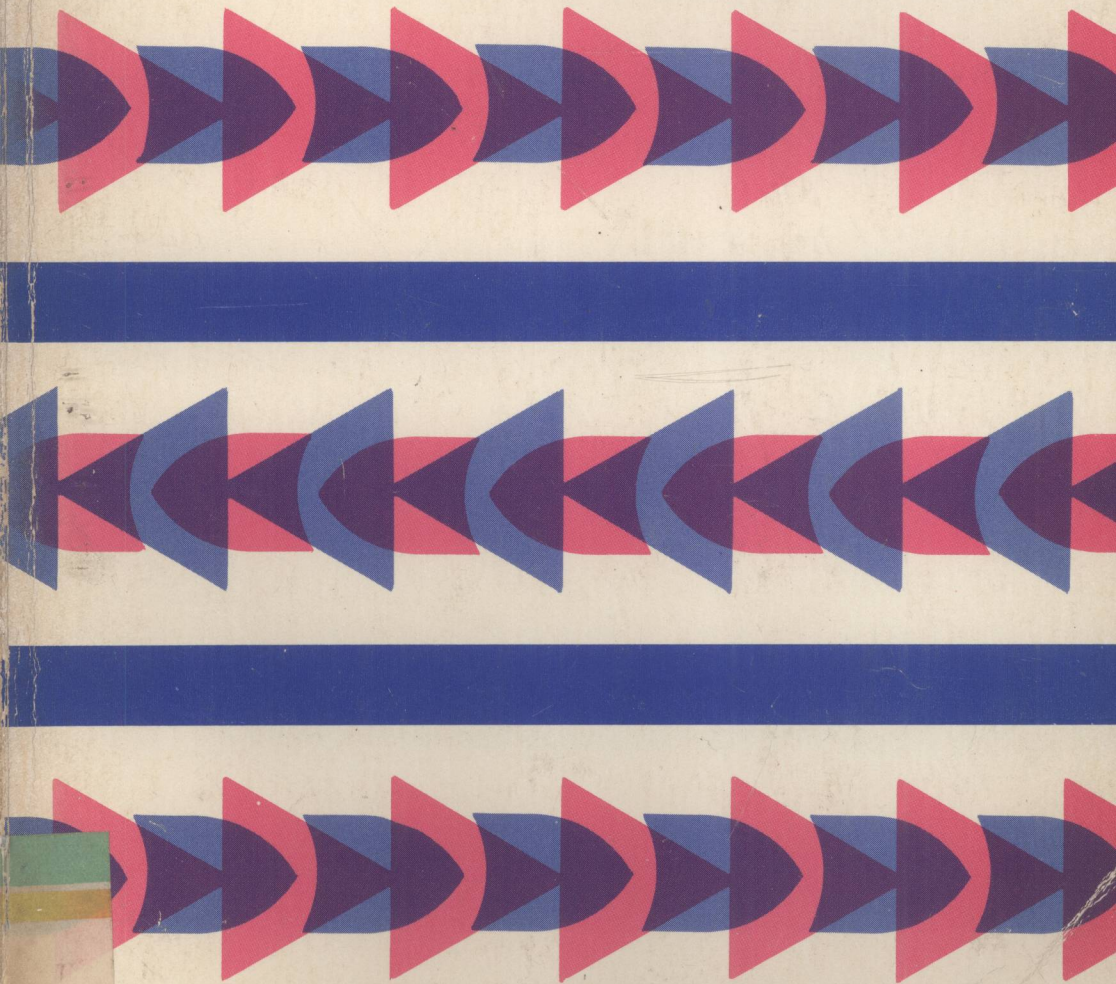
PASCAL FOR FORTRAN PROGRAMMERS

Ronald H. Perrott
Donald C. S. Allison



COMPUTER SCIENCE PRESS

PASCAL FOR FORTRAN PROGRAMMERS

Ronald H. Perrott
The Queen's University of Belfast

Donald C. S. Allison
Virginia Polytechnic Institute

COMPUTER SCIENCE PRESS

Computer Science Press, Inc.
11 Taft Court
Rockville, Maryland 20850

1 2 3 4 5 6 87 87 85 84

Library of Congress Cataloging in Publication Data

Perrott, Ronald H., 1942-
Pascal for FORTRAN programmers.

Bibliography: p.
Includes index.
1. PASCAL (Computer program language) 1. FORTRAN (Computer program language) I. Allison, Donald C. S., 1983- . II. Title.
QA76.73.P2P47 1983 001.64'24 82-7253
ISBN 0-914894-09-9 AACR2

PREFACE

During the sixties the cost of software production and its subsequent use began to exceed the cost of the hardware on which it was executed. Also, the reliability of the programs was suspect while their adaptation or modification to a similar but slightly different situation was a difficult and at times an impossible task. However, the development of new programming methods under the headings of stepwise refinement or structured programming led to the introduction of languages and techniques which produced software of improved quality and reliability at a reasonable cost. These improvements do not appear to have been widely disseminated among the engineering and scientific communities. Naturally, programmers and managers were reluctant to discard and rewrite existing applications.

These communities have traditionally used FORTRAN for their applications. Hence, it is the objective of this book to examine some of the advances which have been made, as represented by the language PASCAL, and to show (where possible) how these can be expressed in FORTRAN. The design and implementation of PASCAL is one of the major contributions to programming language development over the last decade. PASCAL gained immediate recognition particularly as a teaching language for illustrating the concepts of structured programming, but now it has penetrated the majority of computer applications. It is currently being offered by most of the major manufacturers and on many small machines.

When FORTRAN was first introduced in 1957, the majority of programmers used machine code and felt that this was the best way to obtain efficient use of their machine. PASCAL today meets the same attitude from many FORTRAN programmers.

The approach used in this text was conceived during a sabbatical year at NASA's Ames Research Center by one of the authors. During that time questions were frequently asked about PASCAL; how it differed from FORTRAN, and what were the benefits to be gained by using it. The majority of the questioners were programmers, system analysts, scientists, and engineers who were currently engaged on projects which used FORTRAN; they were not inclined or motivated under such circumstances to make the effort to learn another language like PASCAL.

This book is intended to make such a transition as easy as possible, or at the very least, answer the above questions. It explains the program and data

structures which are available in PASCAL and shows (where appropriate) how to express them in FORTRAN and how to apply the process known as stepwise refinement.

Very compelling reasons are necessary to invest time and energy to learn a new language like PASCAL. However, the investment can be very worthwhile. The rich data and control structures and good debugging facilities of PASCAL mean that a program can be designed and written in less time, with less effort, requiring less coding, resulting in less opportunity to make errors and, therefore, requiring less time to find, diagnose, and correct other errors.

PASCAL encourages the process of stepwise refinement so that larger problems can be divided up into more manageable portions or components. The components can then be refined with a minimum of concern for the other components. Such an approach enables the program's documentation to be included at the design phase. If an error is discovered either in the refinement or at the testing stage, it is possible to identify how much of a component, or indeed the program, must be corrected. This also enables a program to be easily modified for a similar type of application or simply maintained by a different programmer.

The maintenance and adaptation of a program is an important property for a scientific or engineering application program. As more knowledge is gained, both from theoretical studies and practical experiments, more comprehensive models are required. This invariably means modifications to an existing program. In such circumstances the readability and clear structure of a PASCAL program is a decided advantage.

Another reason for using a language like PASCAL is its portability. There is one document which has been widely used for its definition and its implementation, and there are very few machine dependent features. The result has been that the movement of programs between various machines, even of different architecture, has been relatively easily achieved.

This widely referenced document on PASCAL, which is used in this text, is the *PASCAL User Manual and Report* by K.Jensen and N.Wirth. More recently a proposed PASCAL Standard has been under consideration by the International Standards Organization, and reference is made to this document also.

FORTRAN can give rise to syntactic subtleties which a compiler will fail to detect. For example, the first American space probe to Venus was aborted after it went off course. The reason was traced to a FORTRAN program statement which was syntactically correct (confusion between a comma and a period). It occurred in a statement of the form:

```
DO 6 I = 1, 25
```

The comma had been mistakenly written/punched as a period. This caused the FORTRAN compiler to create a new variable name DO 6 I and to give this

variable the value 1.25. This error was undetected during the program reading, debugging, and testing phases.

Such an error would have been detected by a PASCAL compiler on the first attempt at compilation. In this case the savings would have been immense, in time, effort, and cost of human and financial resources.

A PASCAL compiler subjects a program to a series of rigorous compile-time and (optional) run-time checks. These checks are applied to both program and data structures and substantially reduce the chance of programmer error. This completely abolishes the very kind of error which, in FORTRAN, gives rise to the most resistant programming errors. This is not to say that PASCAL is a language without defects, and the last chapter of this book has been devoted to the consideration of the disadvantages of PASCAL.

This is a book primarily about PASCAL and programming for those who know some (or all of) FORTRAN and want to learn about PASCAL. All the features of PASCAL are examined and explained by identifying and comparing them with their corresponding features in FORTRAN. In many places there is no direct FORTRAN equivalent, and the book shows what a FORTRAN programmer must do. The translation methods are illustrated in terms of *FORTRAN 77*, although differences with earlier versions of FORTRAN are also mentioned.

Hence, not all the features of *FORTRAN 77* are considered, only those which are required to illustrate the PASCAL features. Also there are a few features in FORTRAN which have no equivalent feature in PASCAL, for example, the double precision data type and the statement function. Syntax or railroad diagrams are used to illustrate the features of both languages in the text. The complete set of syntax diagrams for each language is given in the Appendices, which can be used for reference purposes.

Several case studies are given throughout the chapters to illustrate the benefits of and the use of the different program and data structures. The emphasis has been on non-numerical problems. By using this approach to PASCAL, it is hoped that a scientist or engineer or any one familiar with FORTRAN can learn about PASCAL and structured programming and recognize the advances which have been made in programming language development.

Finally, we would like to thank all those people who read drafts of the manuscript and pointed out mistakes and ambiguities and made valuable comments. In particular, thanks are due to J.A.N. Lee, J. Elder and P. Dhillon.

Ron Perrott
Don Allison
September 1982

TABLE OF CONTENTS

Chapter 1

HISTORICAL OVERVIEW

1.1 THE HISTORY OF FORTRAN

In the early days of programming, before 1954, most programs were constructed in machine language or assembly language. The task of the programmer involved not only finding a solution of the problem under consideration but also involved contending with the characteristics of the machine. The hardware was regarded as the most important component in the programming process because of its expense. A high percentage utilization of this expensive piece of equipment was expected, and many programmers were prepared to invest time and energy to achieve this.

At that time, the attempts at introducing some form of high level language had not been encouraging. The claims made for such systems did not measure up to their actual performance. This led to the widespread belief, within the programming community, that efficient programming could not be achieved by using a high level language translator or compiler.

This situation might not have changed but for the fact that the cost of programming (programmers) began to equal the cost of the computer itself. In addition, a large proportion of the computer's time was being spent in simply debugging programs. These factors began to work in favor of high level language programming.

It was into this environment that John Backus of IBM in 1953/54 proposed the introduction of FORTRAN. A major concern of the FORTRAN team was, therefore, to demonstrate that a system could be built which would produce programs as efficient as hand coded programs and for a wide range of submitted programs. Only in this way would the largely hostile and skeptical programming community be convinced of the benefits of such a language.

Backus [1] felt that if their system translated a program which executed only half as fast as an equivalent hand coded program, then their translator would not be accepted. Hence, the design of the translator was the real challenge, not the task of designing the high level langauge.

In retrospect this was a good decision, otherwise the introduction of high level language programming would have been seriously delayed. Backus' description of the early days of the project indicates that the features of the FORTRAN language were made up as the project progressed. Language design was not considered to be the major problem. Rather the construction of the translator or compiler was considered to be the major problem. The name of the language, being an abbreviation of '*FOR*mula *Tran*slator', reflects much of the thinking on language design at that time.

FORTRAN was intended for a particular IBM machine and was never envisaged as being available on any other machine. Thus, it was felt that including features in the language which made the task of translation easier was acceptable. Hence, the approach to language design was casual in the sense that features were included or dropped as the project proceeded.

FORTRAN, therefore, had the objectives of making programming on the IBM 704 computer much faster, cheaper, and more reliable. As a consequence, the language reflected the hardware constraints of the IBM 704. No special provisions were made for program debugging, as it was assumed optimistically that FORTRAN would eliminate debugging.

It was also assumed that other manufacturers would provide their own similar language on their machines. As it turned out, in order to compete with IBM, the other major computer vendors decided to provide a FORTRAN compiler for their own machines. Their versions of FORTRAN usually included extensions which were tailored to their own hardware.

The description of FORTRAN [2] and its translator program was presented in February 1957 at the Western Joint Computer Conference in Los Angeles. Only at this stage was widespread interest shown in FORTRAN at other IBM 704 sites. By 1958 there were some 66 IBM 704 computers in operation which were using FORTRAN.

Thus, FORTRAN was the first high level language introduced on a large scale. It proved the viability of using a high level language to solve many problems. However, the spread and acceptance of FORTRAN was not without resistance. The majority of programmers at that time saw no reason to change because they felt that machine code was the best way to obtain efficient use of a computer. Also, it takes time and effort to learn a new language and a lot of courage to abandon working programs or to start a project in a new language, especially when there is no obvious guarantee of improvement or success.

In 1962, a working group (X3) of the American National Standards Institute (ANSI) was given the task of producing a specification of FORTRAN. ANSC-X3 produced two documents known as FORTRAN and Basic FORTRAN, with the latter language being a subset of the former. We will refer to

these documents as *FORTRAN 66*. This *Standard* was subsequently accepted by the International Standards Organization (ISO).

Some of the main considerations of the standardization committee were to facilitate the movement of programs between machines and to insure that the language would be upward compatible with previous versions of FORTRAN.

A compiler writer was free to add other features to the language provided the rest of the features still remained within the *Standard*. Many manufacturers took advantage of this flexibility which probably resulted in the spread of FORTRAN. Manufacturers were able to incorporate features which their particular hardware was good at exploiting.

In April 1978 the earlier *Standard* for both Basic FORTRAN and FORTRAN was withdrawn and a new *Standard* approved [3]. This is widely known as *FORTRAN 77*. It also contained two parts, FORTRAN, and a subset FORTRAN. Any program written in the subset would execute when submitted to a compiler for the full language. It is the full *FORTRAN 77* language which has been used in the following chapters.

FORTRAN 77 tried to maintain upward compatibility with *FORTRAN 66*, where possible. A feature of *FORTRAN 66* would only be eliminated if there were good reasons for doing so. Several changes were also made to encourage portability; for example, the Hollerith constant was replaced by the character data type.

The *FORTRAN 77 Standard* is approximately 200 pages in length, much larger than the earlier *Standard*. The increased size was intended to make the document easier for users to understand and at the same time to give a compiler writer more freedom. *FORTRAN 77* can still include extra features provided it executes programs adhering to the *Standard*.

FORTRAN in its original form was a simple language which was easy to learn and efficient to implement. The interaction of its features was well understood, but the regular addition of new features as each new version was produced has undermined these properties. FORTRAN is now a large language with few means of structuring data, but with many features whose interaction in some instances is ambiguous.

This ambiguity has given rise to different interpretations by different implementers, which have made the movement of programs between different processors more difficult.

1.2 THE HISTORY OF PASCAL

In 1957 as a result of pressure from user groups, John Carr III, President of the Association for Computing Machinery, appointed a committee to estab-

lish a universal programming language. There was to be collaboration with a European Committee (GAMM) which was already engaged in a similar task. It is interesting to note that FORTRAN was regarded as unacceptable for this language as it was the property of IBM.

The deliberations of this group produced the algorithmic language known as ALGOL 60. Unlike FORTRAN, the design and development of this language proceeded in parallel. The syntax (grammar) and semantics (meaning) were described in a report which used a special notation to describe the syntax. This notation or metalanguage is known as the Backus-Naur Form[4].

ALGOL 60 made little impact in the United States but was more successful in Europe. As the years passed and implementations of ALGOL 60 became available, a successor to ALGOL 60 known as ALGOL 68 was produced. ALGOL 68 turned out to be a controversial language in that many people regarded it as too complex. As a result, Niklaus Wirth of Eidgenossische Technische Hochschule in Zurich embarked on the design and implementation of what we now know as PASCAL.

Wirth had designed and implemented at least two other languages before commencing his work on PASCAL. He was, therefore, one of the leading authorities on the design and implementation of programming languages. He had started to name his languages after mathematicians, hence the choice of the name PASCAL.

There were two principal aims in the development of PASCAL [5]:

i) to make available a language suitable to teach programming as a systematic discipline based on certain fundamental concepts clearly and naturally reflected by the language (structured programming),

ii) to develop implementations of this language which are both reliable and efficient on presently available computers.

The language benefited from the experience gained in the design and use of other high level languages:

i) by identifying what is missing, and what is necessary,

ii) by using more modern techniques to produce a reliable and efficient compiler.

The first version of PASCAL was drafted in 1968 with the first compiler available in 1970. Wirth initially intended, in order to ensure the portability of the compiler, to use FORTRAN as the implementation language. However, the program and data structures of FORTRAN were inadequate for such a task. Eventually, PASCAL was implemented using PASCAL as the implementation language [6]. After two years experience with the language some minor modifications were introduced.

The substantive publication and the one that we shall refer to frequently is the *PASCAL User Manual and Report* [7] which consists of a tutorial on the language as well as a formal report. It is this document which has been widely used as a reference document for the implementers of the language.

In 1973 Hoare and Wirth [8] attempted a formal definition of the semantics of PASCAL. This enabled areas of uncertainty and ambiguity to be identified and slight revisions to be made.

In 1977 a working group within the British Standards Institute [9] was formed to produce a standard for PASCAL. Their deliberations have since been widely distributed in the computer literature. Their proposed *PASCAL Standard* is now in the process of being accepted by the International Standards Organization [10]. This document will be frequently referred to in the text.

PASCAL gained widespread acceptance as a teaching language to illustrate the concepts of structured programming, and soon it was being used in other application areas. It is currently being offered by most of the major manufacturers and on many small machines.

PASCAL is a simple language with few features, eight data types and nine control structures whose interaction or combination is well understood. Both the data and control structures can be developed hierarchically. The fact that PASCAL has few machine dependent features has enabled the movement of programs between different machines to be relatively easily achieved.

One other major development has been the introduction of the programming language Ada[1] by the US Department of Defense [11] for use on its embedded computer systems. Of the four proposed languages which were submitted, all four chose as their base language PASCAL. PASCAL is, therefore, a prerequisite to understanding Ada.

References

[1] Backus, J.W., "The History of FORTRAN I, II and III", in *The History of Programming Languages*, R.W. Wexelblat ed., Academic Press, Los Angeles, 1981

[2] Backus, J.W. et al, "The FORTRAN Automatic Coding System", in *Proc. Western Joint Computer Conference*, Los Angeles, 1957.

[3] *American National Standard Programming Language FORTRAN (ANSI X3.9-1978)*, New York American National Standards Institute Inc., 1978.

[1]Ada is a registered trademark of the US Department of Defense.

[4] Naur, P., "Revised Report on the Algorithmic Language ALGOL 60", *Comm ACM* 6, (1) 1963, 1-17.

[5] Wirth, N. "The Programming Language PASCAL", *Acta Informatica* (1), 1971, 35-63.

[6] Wirth, N., "The Design of a PASCAL Compiler", *Software—Practice and Experience* (1), 1971, 309-333.

[7] Jensen, K. and N. Wirth, *PASCAL User Manual and Report*, Springer-Verlag, New York, 1978.

[8] Hoare, C.A.R. and N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL", *Acta Informatica* (3), 1973, 335-355.

[9] Addyman, A.M. et al. "A Draft Description of PASCAL", *Software—Practice and Experience* (9), 1979, 381-424.

[10] *Computer Programming language PASCAL*, ISO Draft International Standard (DIS) 7185, International Organization for Standardization 1982.

[11] *Reference Manual for the Ada Programming Language*, Washington, D.C., US Department of Defense, November, 1980.

Chapter 2

STRUCTURED PROGRAMMING

2.1 PROGRAM DESIGN

In this chapter the topic of program design and construction is considered. The method described forms the basis for the design and construction of the programs used in the case studies of the following chapters. This method is known as structured programming or stepwise refinement, and it is a technique which is independent of the programming language used. It has been the increasing cost of program construction, the suspect reliability, and the difficulty of adapting programs which has led to the spread and acceptance of techniques such as structured programming.

The design and construction of a solution by means of a program involves several distinct phases. In the case of a small program many of these phases may be carried out subconsciously by the programmer. However, if we think "big" and regard ourselves as embarking on a project which will take many years to complete and which involves many programmers, then each of the phases requires individual specification.

In the latter situation we can list the following phases as being involved in the production of the final product:

i) **Definition**. A clear and precise definition of the actual problem must be given and understood.

ii) **Choice of algorithm**. An algorithm must be chosen, perhaps from a set of algorithms, as the most appropriate to use. In fact, a choice of algorithm may be required at various stages of the solution.

iii) **Choice of programming language**. Ideally a programming language should be chosen which has the most suitable program and data structures. However, the number of programming languages available at an installation or mastered by the programmers involved will be limited so that this phase may not involve a major decision.