FUNDAMENTALS OF STRUCTURED PROGRAMMING USING FORTRAN WITH SF/k AND WATFIV-S

R. C. HOLT
J. N. P. HUME

7-3

FUNDAMENTALS OF STRUCTURED PROGRAMMING USING FORTRAN WITH SF/k AND WATFIV-S

R. C. HOLT
J. N. P. HUME
Department of Computer Science
University of Toronto

RESTON PUBLISHING COMPANY, INC., Reston, Virginia
A Prentice-Hall Company

Library of Congress Cataloging in Publication Data

Holt, R C
Fundamentals of structured programming using FORTRAN with SF/k and WATFIV-S.

Bibliography: p. 337
Includes index.
1. FORTRAN (Computer program language)
2. Structured programming. I. Hume, J. N. P., joint author. II. Title.
QA76.73.F25H85 001.6'424 76-20593
ISBN 0-87909-303-X
ISBN 0-87909-302-1 pbk.

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

PREFACE

This book is intended to form the basis of an introductory course in computing. No particular mathematical background beyond basic arithmetic is assumed; examples are taken largely from everyday life. In this way, the focus is on programming and problem solving, rather than on mathematics. It is our strong conviction that the foundation of computer programming must be carefully laid. Bad habits once begun are hard to change. Even for those who do not continue to study computer science, an experience in the systematic analysis of problems from the statement of "what is to be done" to the final algorithm for "doing it" can be very helpful in encouraging logical thinking.

The programming language presented here is Fortran, extended in such a way that it permits structured programming. This language is introduced in a series of subsets that we call SF/1, SF/2, SF/3, and so on. The SF stands for Structured Fortran. We hope that a student will learn the concepts of structured programming by following this step-by-step presentation of extended Fortran subsets.

Just as a program provides a list of instructions to the computer to achieve some well-defined goal, the methodology of structured programming provides a list of instructions to persons who write programs to achieve well-defined goals. The goals of structured programming are to get a programming job done correctly and in such a form that later modifications can be done easily. This means that programs must be understood by people other than their authors.

The Standard Fortran language is not a language that encourages structured programming but fortunately it has recently been extended to include several new features that are essential. One compiler that supports these extensions is the Watfiv-S compiler. All programs written in this book are compatible with Watfiv-S.

As each extended Fortran subset is learned, new possibilities open up. Even from the first subset SF/1, it is possible to write programs that do calculations and print. By the time the subset SF/4 is reached, a student has learned how to handle alphabetic information, as well as to do numerical calculations and structure the control flow of the program.

Extended Fortran was chosen because it contains control structures that make structured programming easy. It is, however, possible to produce the same result using Standard Fortran and this is described in detail. The reason that Standard Fortran is not used in the first place is that it obscures the elegance of the control structures and does not permit format-free input-output or direct character handling.

Structured programming is especially important when working on larger programs; a detailed discussion of the techniques of modular programming and top-down design accompanies the introduction of Fortran subprograms in SF/7.

Many examples in the book are from data processing. General concepts of data structures, searching, and sorting fit well into this important area that touches all our lives. The book ends with examples of scientific calculations and the translation of a high-level programming language into machine language.

At all times we have tried to present things in easy to understand stages, offering a large number of program examples and exercises to be done by the student. Each chapter has a summary of the important concepts introduced in it.

The subsets of extended Fortran that we call SF/k are based on subsets for the PL/1 language called SP/k. THe SP/k subsets were designed by R. C. Holt and D. B. Wortman of the University of Toronto.

This book was prepared using a text editing system on a computer. Each program was tested using the Watfiv-S compiler. The job of transcribing the authors' pencil scrawls into the computer was done with great care and patience by Inge Weber. We are indebted to the many people who offered constructive criticism. In particular we would thank Jim Horning, Bob Cherniak, Brian Clark, Dave Barnard, Les Mezei, Rudy Schild, and Laurie Johnston for their detailed critiques. We have sprinkled through the book names of other people who have helped us.

The time taken to write a book comes at the expense of other activities. Since most of the time was in the evenings or on weekends we must end with grateful thanks to our wives Marie and Patricia.

R. C. Holt
J. N. P. Hume

CONTENTS

CHAPTER 1

INTRODUCTION TO PROGRAMMING

The name of this book is somewhat of a mouthful! Perhaps it would help if we took it piece by piece and introduced you to the name slowly. We hope that it is no secret that the book has to do with computers and particularly with the use of computers rather than their design or construction. To use computers you must learn how to speak their language or a language that they can understand. We do not actually speak to computers yet, although we may some day; we write messages to them. The reason we write these messages is to instruct the computer about some work we would like it do for us. And that brings us to programming.

WHAT IS PROGRAMMING?

Programming is writing instructions for a computer in a language that it can understand so that it can do something for you. You will be learning to write programs in one particular programming language called Fortran. When these instructions are put on to some medium that a computer can read such as punched cards then they can be fed into the machine. They go into the part of the computer called its memory and are recorded there for as long as they are needed. The instructions could then be executed if they were in the language the computer understands directly, the language called machine language. If they are in another language such as Fortran they must first be translated, and a program in machine language compiled from the original or source program. After compilation the program can be executed.

Computers can really only do a very small number of different basic things. For example, an instruction which says, STAND ON YOUR HEAD, will get you nowhere. The repertoire of instructions that any computer understands usually includes the ability to move numbers from one place to another in its memory, to add, subtract, multiply, and divide. They can, in short, do all kinds of arithmetic calculations and they can do these operations at rates of up to a million a second. Computers are extremely fast calculating machines. But they can do more; they can also handle alphabetic information, both moving it around in their memory and comparing different pieces of information to see if they are the same. To include both numbers and alphabetic information we say that computers are data processors or more generally information processors.

When we write programs we write a sequence of instructions that we want executed one after another. But you can see that the computer could execute our programs very rapidly if each instruction were executed only once. A program of a thousand instructions might take only a thousandth of a second. One of the instructions we can include in our programs is an instruction which causes the use of other instructions to be repeated over and over. In this way the computer is capable of repetitious work; it tirelessly executes the same set of instructions again and again. Naturally the data that it is operating on must change with each repetition or it would accomplish nothing.

Perhaps you have heard also that computers can make decisions. In a sense they can. These so-called decisions are fairly simple. The instructions read something like this:

```
IF JOHN IS OVER 16 THEN PLACE HIM ON THE HOCKEY TEAM
     ELSE PLACE HIM ON THE SOCCER TEAM
```

Depending on the condition of John's age, the computer could place his name on one or other of two different sports teams. It can decide which one if you tell it the decision criterion, in our example being over sixteen or not.

Perhaps these first few hints will give you a clue to what programming is about.

WHAT IS STRUCTURED PROGRAMMING?

Certain phrases get to be popular at certain times; they are fashionable. The phrase, "structured programming" is one that has become fashionable recently. It is used to describe both a number of techniques for writing programs as well as a more general methodology. Just as programs provide a list of instructions to the computer to achieve some well-defined goal, the methodology of structured programming provides a list of instructions to persons who write programs to achieve some well-defined goals. The goals of structured programming are, first, to get the job done. This deals with how to get the job done and how to get it done correctly. The second goal is concerned with having it done so that other people can see how it is done, both for their education and in case these other people later have to make changes in the original programs.

Computer programs can be very simple and straightforward but many applications require that very large programs be written. The very size of these programs makes them complicated and difficult to understand. But if they are well-structured, then the complexity can be controlled. Controlling complexity can be accomplished in many different ways and all of these are of interest in the cause of structured programming. The fact that structured programming is the "new philosophy" encourages us to keep track of everything that will help us to be better programmers. We will be cataloguing many of the elements of structured programming as we go along, but first we must look at the particular programming language you will learn.

WHAT IS FORTRAN?

The name Fortran is short for FORmula TRANslation and Fortran is a language that has been developed to be independent of the particular computer on which it is run and oriented to the problems that persons might want done. We say that Fortran is a high-level language because it was designed to be relatively easy to learn and use. As a problem-oriented language it is particularly concerned with problems of numerical calculations such as occur in scientific and engineering applications but it has been extended so that it can be useful in alphabetic information handling required by business and humanities applications.

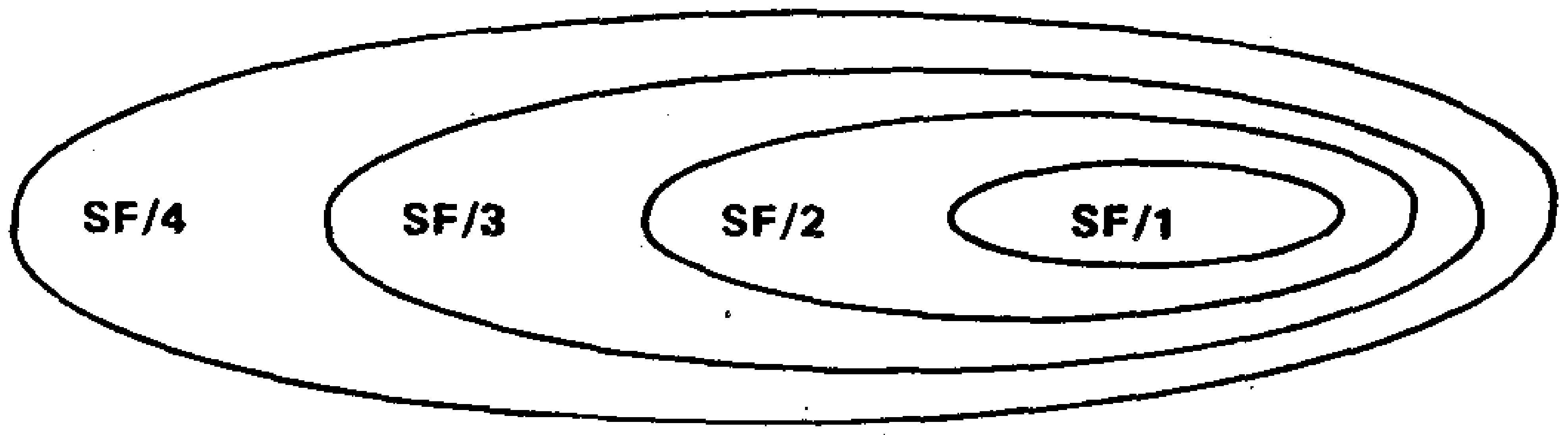
Fortran as it has been extended is a very extensive language, so that although each part is easy to learn, it requires considerable study to master. Many different computer installations have the facilities to accept programs written in Fortran. This means that they have a Fortran compiler that will translate programs written in Fortran into the language of the particular machine that they have. Also many programs have already been written in Fortran; in some installations a standard language is adopted, and Fortran is often that standard language.

It has been the experience over the past years that a high-level language lasts much longer than machine languages, which change every five years or so. Fortran began its existence nearly twenty years ago and it has had numerous extensions. As each new version of Fortran was created an attempt was made to keep it compatible with previous versions. This is because once an investment has been made in programs for a range of applications, an installation does not want to have to reprogram when a new Fortran compiler is acquired.

Because of the long life-span of programs in high-level languages it becomes more and more important that they can be adapted to changes in the application rather than completely reconstructed. A high-level language has the advantage that well-constructed and well-documented programs in the language can be readily modified. It is our aim to teach you how to write such programs. To start your learning of Fortran we will study subsets of the extended Fortran language called SF/k. SF/k was developed at the University of Toronto.

WHAT IS SF/k?

The name SF/k stands for Structured Fortran subset k. There really is a series of subsets beginning at SF/1, then SF/2, and going on up. The first subset contains a small number of the language features of extended Fortran, but enough so that you can actually write a complete program and try it out on a computer right away. The next subset, SF/2, contains all of SF/1 as well as some additional features that enlarge your possibilities. Each subset is nested inside the next higher one so that you gradually build a larger and larger vocabulary in the extended Fortran language. At each stage, as the special features of a new subset are introduced, examples are worked out to explore the increased power that is available.



THE SF/k SUBSETS

In a sense, the step-by-step approach to learning Fortran is structured and reflects the attitude to programming that we hope you learn.

There is no substitute for practice in learning to program, so as soon as possible and as often as possible, submit your knowledge to the test by creating your own programs.

WHY LEARN JUST A SUBSET?

The Fortran language is very extensive; some features are only used rarely or by a few programmers. If you know exactly what you are doing, then these features may provide a faster way to program; otherwise they are better left to the experts. A beginner cannot really use all the features of the complete Fortran language and will get lost in the complexity of the language description. With a small subset it is much easier to pick up the language and then get on with the real job of learning programming.

For learners we need a fast compiler because, for many programs, compiling is nearly all that happens; the execution is sometimes omitted or is very short because there are errors in the program. A special compiler might be used for the SF/k language and this could be small enough to run on some very inexpensive computers often called minicomputers. But it is not necessary to have an SF/k compiler to use the SF/k language since it is a subset of the extended Fortran language of the Watfiv-S compiler.

But perhaps most important, the SF/k language has been selected from the extended Fortran language so as to provide features that encourage the user to produce well-structured programs. This is why it is appropriate as a means of learning structured programming.

CORRECTNESS OF PROGRAMS

One of the maddening things about computers is that they do exactly what you tell them to do rather than what you want them to do. To get correct results your program has to be correct. When an answer is printed out by a computer you must know whether or not it is correct. You cannot assume, as people often do, that because it was given by a computer it must be right. It is the right answer for the particular program and data you provided because computers now are really very reliable and rarely make mistakes. But is your program correct? Are your input data correct?

One way of checking whether any particular answer is correct is to get the answer by some other means and compare it with the printed answer. This means that you must work out the answer by hand, perhaps using a hand calculator to help you. When you do work by hand you probably do not concentrate on exactly how you are getting the answer but you know you are correct (assuming you do not make foolish errors). But this seems rather pointless. You wanted the computer to do some work for you to save you the effort and now you must do the work anyway to test whether your computer program is correct. Where is the benefit of all this? The labor saving comes when you get the computer to use your program to work out a similar problem for you. For example, a program to compute telephone bills can be checked for correctness by comparing the results with hand computations for a number of representative customers and then it can be used on millions of others without detailed checking. What we are checking is the method of the calculation.

We must be sure that our representative sample of test cases includes all the various exceptional circumstances that can occur in practice, and this is a great difficulty. Suppose that there were five different things that could be exceptional about a telephone customer. A single customer might have any number of exceptional features simultaneously. So the number of different