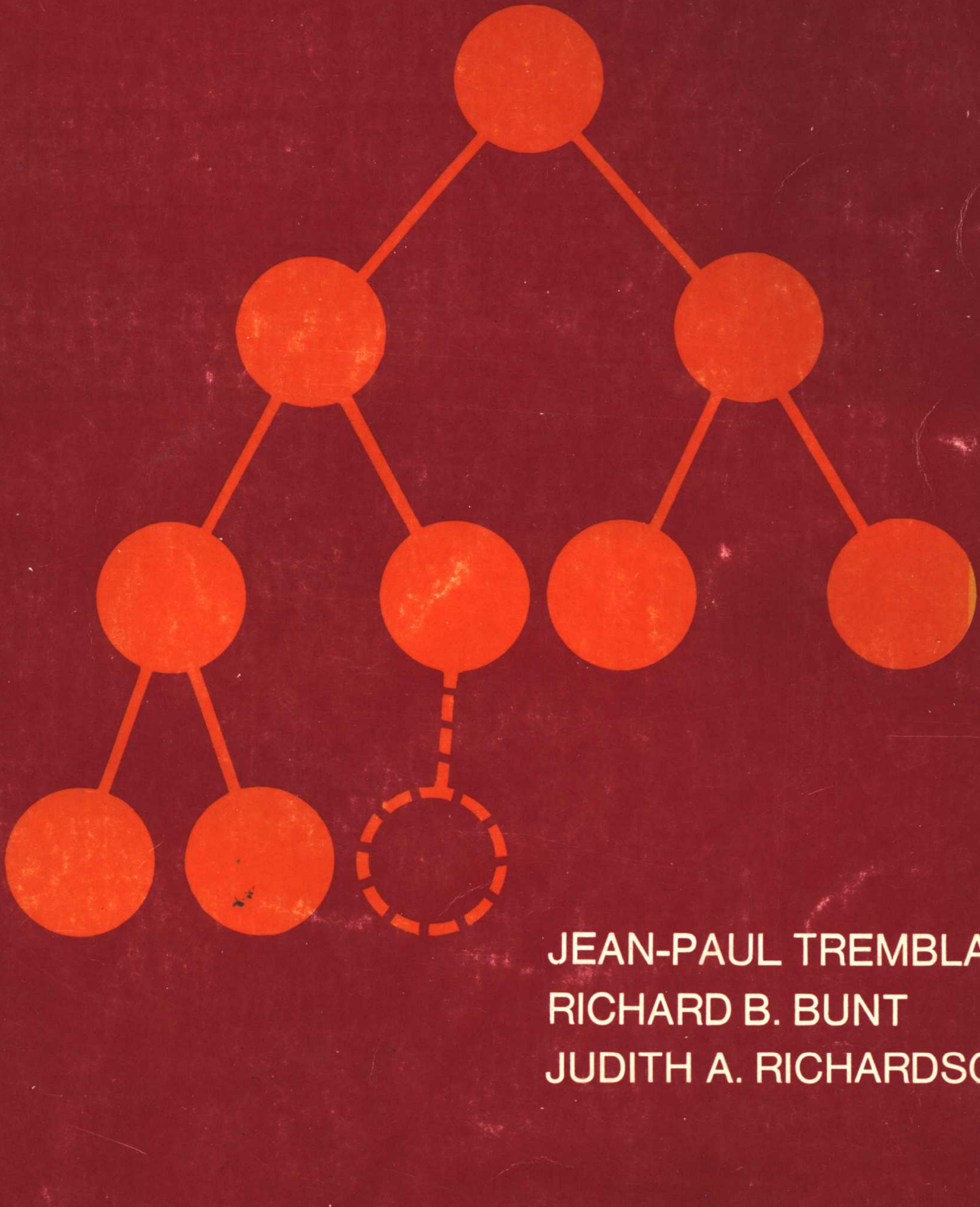
STRUCTURED PL/1 (PL/C) PROGRAMMING



JEAN-PAUL TREMBLAY
RICHARD B. BUNT
JUDITH A. RICHARDSON

STRUCTURED PL/I (PL/C) PROGRAMMING

Jean-Paul Tremblay
Richard B. Bunt
Judith A. Richardson

Department of Computational Science
University of Saskatchewan, Saskatoon
Canada

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Bogotá Düsseldorf
Johannesburg London Madrid Mexico Montreal
New Delhi Panama Paris São Paulo Singapore Sydney Tokyo Toronto

STRUCTURED PL/I (PL/C) PROGRAMMING

3 4 5 6 7 8 9 0 EBEB 8 3 2

This book was set in Souvenir by the authors.
The editors were Charles E. Stewart and Annette Hall;
the designer was Robin Hessel
the production supervisor was Dominick Petrellese
Edwards Brothers Incorporated was printer and binder.

Library of Congress Cataloging in Publication Data

Tremblay, Jean-Paul, date
Structured PL/I (PL/C) programming.

Bibliography: p.
Includes index.
1. PL/I (Computer program language) 2. PL/C
(Computer program language) I. Bunt, Richard B.,
date joint author. II. Richardson, Judith A.,
joint author. III. Title.
QA76.73.P25T73 001.6'424 79-17904
ISBN 0-07-065173-6

PREFACE

The first course in a computer science curriculum is certainly one of the most important. For most students this constitutes their initial exposure to fundamental notions such as the algorithm, and to the description of solutions in a manner sufficiently precise for computer interpretation. It is important that these notions be properly taught for, as the ancient Roman poet Horace observed, "A new cask will long preserve the tincture of the liquor with which it was first impregnated."

To this end we have prepared a package of instructional materials which reflects our own view of how the first course should be organized and taught. The cornerstone of this package is a book entitled "An Introduction to Computer Science: An Algorithmic Approach" (Tremblay/Bunt, 1979). This book presents computer science concepts in an algorithmic framework, with a strong emphasis on problem solving and solution development. We feel this to be particularly important for the first course.

Clearly the use of a programming language is an important part of the first course too. For that reason we have prepared a series of supplementary integrated programming guides (of which this is one) to provide the needed support. The supplementary guides are not intended to re-teach the ideas of the main book, but rather to *supplement* them with the programming concepts required to implement them in a particular programming language (here, PL/I), and thereby provide the student with the practical programming framework that we feel to be important.

Any book on the PL/I language must set careful terms of reference. The "language" for any programmer is, in fact, defined by the compiler that he or she is using. One of the original PL/I compilers was IBM's F-level PL/I compiler, now superseded by the newer Checkout and Optimizing compilers. Several other computer manufacturers offer PL/I-like languages in their software support: Honeywell, for example, has a language called EPL for its MULTICS operating system, and Digital Equipment Corporation offers a language called CPL with its DECSYSTEM 10 and 20 operating systems. A number of student-oriented compilers have emerged from universities: prominent among these are the University of Toronto's SP/k, Brooklyn Polytechnic Institute's PLAGO, the University of Maryland's PLUM, and Cornell University's PL/C. Our discussion in this book centers primarily on the PL/I programming language as it is implemented in the PL/C compiler. On occasion, however, we venture into full PL/I as supported by IBM, when language features that we require are not supported by PL/C.

It has been our experience that students learn by "viewing." This is particularly true in the case of programming where it seems that there are immense barriers of bewilderment for many students at the outset. To try to flatten these barriers we present worked-out sample programs, in many cases complete with actual run output. These have been programmed using the compilers available to us at the University of Saskatchewan; namely, release 7.6 of the PL/C compiler and version 5.4 of the IBM F-level PL/I compiler. In addition to examples presented for the sake of illustration, most chapters end with a number of more detailed applications that attempt to draw together the material presented in the chapter. These are the same applications that are discussed in the main book; their choice reflects our emphasis on the nonnumeric aspects of computing. As in the main body, this same bias is carried over into the exercises as well. Exercises are found at the end of most sections and at the end of most chapters.

Much has been said and written in the past few years about the merits of an approach to programming loosely termed "structured programming." Studies of the programming task itself have shown that adherence to certain basic principles can result in the production of better quality programs. Our approach is based on many of these principles, and our presentation and examples are designed accordingly. Chapter 7, on programming style, examines the process of programming itself in more depth.

Finally, since we view this guide both as an instructional vehicle and a reference document, we have included as an appendix a lengthy reference summary of the PL/I language (and, in particular, the PL/C language).

SUMMARY BY CHAPTERS

The book begins with a brief introduction to programming from a PL/I perspective.

Chapter 2 provides an introduction to basic concepts of computing and programming as well as the first examples of complete PL/I programs. Some simple applications are described.

The notion of "flow of control" is introduced in Chapter 3, along with two fundamental PL/I control structures: the IF and the DO. Solutions to several fairly elaborate applications are developed.

The concept of the array is the topic of Chapter 4. Processing of single-dimensional arrays, or vectors, is discussed first. The chapter then moves to a consideration of arrays of higher dimension. Some typical applications of vectors and arrays are discussed. Among these are the important applications of searching and sorting.

String processing is the topic of Chapter 5. The representation of strings in a computer and basic PL/I operations on strings are described. A number of simple applications involving string processing are developed. More advanced topics are deferred to Chapter 9. Chapter 5 also deals for the first time with the concepts of formatted I/O — in particular the GET EDIT and PUT EDIT statements of PL/I.

Chapter 6 deals with functions and procedures. Topics discussed include the correspondence of arguments and parameters, the way in which functions and procedures are invoked and values are returned in PL/I, and the general question of scope in programs. Three applications involving the use of functions and procedures are considered.

Programming style is the topic of Chapter 7. This we feel to be an important chapter of the main book. In this book we try to consider the effects of style on the production of PL/I programs. Examples of actual programs are included to illustrate the points made.

Chapter 8 deals with the subject of numerical computation. PL/I programs are given for the solution of problems discussed in the main book. These include root finding, numerical integration, the solution of simultaneous linear equations, and curve fitting. For some of the material in this chapter, familiarity with elementary calculus would be an asset.

Chapter 9 returns to the topic of string processing, with the presentation dealing with more advanced applications such as KWIC indexing and text editing.

Chapter 10 offers an introduction to the support of linear data structures in PL/I. Simple structures such a linear lists, stacks, and queues are discussed, as are PL/I capabilities to manage these structures. Important PL/I features such as pointer variables, controlled and based storage, structures, and recursion are presented for the first time. A number of important applications are described. These include the compilation of expressions, the symbolic manipulation of polynomials, and simulation. Also discussed in this chapter are hash-table techniques.

Chapter 11 considers the PL/I support for the most important non-linear data structure — the tree. Topics include the representation of trees in PL/I and the application of trees to problems such as the symbolic manipulation of expressions, searching, and sorting.

As already mentioned, the book concludes with an appendix containing a detailed reference summary of the PL/I language.

HOW TO USE THIS BOOK

This book is intended for use in conjunction with the book by Tremblay and Bunt entitled "An Introduction to Computer Science: An Algorithmic Approach" (Tremblay/Bunt, 1979). The material covered by these two books encompasses courses CS1 and CS2 in the revised curriculum proposals of the Association for Computing Machinery (Austing et al., 1979).

As was done in the main book, we make assumptions as to the nature of available computing facilities. For convenience of presentation we assume a card reader/line printer environment throughout. Since we recognize that this may not be the case for many students, the dependency on such matters is minor. Should an alternative environment exist, a simple comment from the instructor should suffice to overcome any possible problems of comprehension.

ACKNOWLEDGMENTS

A project of this scale cannot be completed without the able assistance of many people. We are, of course, indebted to all those who assisted us both directly and indirectly. Marilyn Archibald was an active participant in the early stages of the project, and contributed significantly to the first four chapters. Grant Cheston devoted a great deal of time to the reading of our notes and contributed many valuable suggestions. Our proof readers, including Lyle Opseth, Murray Mazer, Cheryl Ernewein, and Dave Hrenewich, showed patience and diligence. Lyle Opseth also programmed many of the examples in the book. Murray Mazer and Guy Friswell assisted in the programming of some examples. We are grateful for the support and comments of our colleagues and students in the Department of

Computational Science at the University of Saskatchewan, who have class-tested preliminary versions of our books over the past three years. We appreciate the efforts of the Department of Printing Services at the University of Saskatchewan, and in particular Mr. Bill Snell who provided us with an automatic typesetting capability that made it possible to meet a difficult schedule. Finally, one of the authors (Bunt) owes a large debt of thanks to the Research Division of the IBM Corporation at Yorktown Heights, New York, for a very enjoyable and productive sabbatical leave during which this project was completed.

Jean-Paul Tremblay

Richard B. Bunt

Judith A. Richardson

REFERENCES

AUSTING, R. H., BARNES, B. H., BONNETTE, D. T., ENGEL, G. L., and STOKES D. G.: "CURRICULUM '78: Recommendation for the Undergraduate Program in Computer Science," *Communications of the ACM*, Vol. 22, No. 3, March 1979, pp. 147-166.

TREMBLAY, J. P. and BUNT, R. B.: *An Introduction to Computer Science: An Algorithmic Approach,* McGraw-Hill Book Co., New York, 1979.

CONTENTS

CHAPTER 1 INTRODUCTION TO PL/I PROGRAMMING

Interactions involving humans are most effectively carried out through the medium of language. Language permits the expression of thoughts and ideas, and without it, communication, as we know it, would be very difficult indeed.

In computer programming, a programming language serves as the means of communication between the person with a problem and the computer used to help solve it. Languages are said to affect the thought and culture of those who use them. Eskimos, for example, have a large vocabulary simply on the subject of snow. An effective programming language enhances both the development and the expression of computer programs. It must bridge the gap between the too often unstructured nature of human thought and the precision required for computer execution. The programming language shapes the thought processes of the programmer, and the quality of the language has a large effect on the quality of the programs produced with it.

1-1 INTRODUCTION

This book is intended to supplement the text *An Introduction to Computer Science: An Algorithmic Approach.* Its purpose is to provide an introduction to the programming language PL/I, sufficient to enable the reader to implement the algorithms of the main text.

As much as possible, we attempt to parallel the presentation of the main text. In our presentation, we assume that the pertinent sections of the main text have been read. The algorithmic language of the main text has been designed for easy translation into several popular programming languages, including PL/I.

The purpose of this chapter is to define a perspective for the material in this book by providing a brief overview of the development and use of the PL/I language.

1-2 A SHORT HISTORY OF THE PL/I LANGUAGE

In the early days of computing, programming was a very formidable task. Many of the early computers were "hard wired" to perform a specific task: to change the "program" required rewiring components. John von Neumann was the first to propose the concept of the stored program, that is, the idea that the instructions of the program be stored in the memory of the computer along with the data. Before long, people began to look for more convenient ways of specifying these instructions, moving from low level (i.e., machine-oriented) symbolic assembly languages through to higher level problem-oriented programming languages.

One of the first general purpose problem-oriented programming languages was FORTRAN (FORmula TRANslator), introduced in 1954 and designed for the solution of scientific numerical problems. FORTRAN was instrumental in demonstrating the value and cost effectiveness of problem-oriented programming languages, and soon other such languages began to appear, with COBOL (COmmon Business Oriented Language) for business applications, ALGOL (ALGOrithmic Language) for problems in numerical mathematics, LISP for list processing applications primarily in artificial intelligence, and SNOBOL for applications involving string manipulation, proving to be the most enduring.

Late in 1963, a committee consisting of IBM personnel and customers began to consider the design of a new programming language, intended originally as a major FORTRAN enhancement. During the design process, features were borrowed from the other languages mentioned previously and incorporated, perhaps in slightly different form, into the new language, which was eventually named PL/I. The first official PL/I manual was published in 1965, and the first compiler was available in 1966.

PL/I is a very large and a very general language, with features appropriate to the solution of a very wide range of problems. Its use as a teaching language has been enhanced by the development of student-oriented compilers, perhaps the most popular of which is PL/C, developed at Cornell University. Such compilers support only those features of PL/I appropriate to student programmers and offer powerful error diagnostic capabilities to aid in learning the language. Also, the cost of running programs under PL/C is low, thus permitting its use in large classes.

1-3 THE USE OF A PROGRAMMING LANGUAGE

As described earlier, a programming language serves to aid in the transformation of a problem solution into an executable computer program. In fact, a language that is well-designed enhances not only the *expression* of the solution, but also its *development* as well.

Once a problem solution has been formulated in terms of a computer program in some programming language, it must then be translated into the machine language of the computer on which the program is to be run. Machine language is not programmer-oriented; machine language programs are nothing more than long strings of numbers, but written in such a way as to be meaningful to the computer. The translation of a program written in a high level programming language (sometimes called the *source program*) to its machine language equivalent (sometimes called the *object program*) is handled through a special program known as a *compiler*.

For a given programming language, there may be many compilers. For example, there will be a different compiler for every different type of machine that supports the language. Even on the same machine, there may be several compilers for the same language. For example, in addition to the regular compiler (say, the regular PL/I compiler) there may be one or more student compilers (such as PL/C) that support perhaps fewer features of the language but offer special instructional capabilities.

Often the action of a compiler appears transparent to the programmer, but never completely so. For example, the compiler can often detect errors made in the writing of the program that would prevent it from running correctly. Other errors may escape its detection, and not be discovered until the translated machine language program (or the object program) is actually in execution. The distinction between *compile-time* and *run-time* is described more completely in Chap. 2.

1-4 THE APPROACH OF THE BOOK

Although we are dealing with the *language* PL/I, where possible, we will be running our example programs under the PL/C *compiler*. There are some subtle differences that ought not to affect the beginning programmer, but may be of more concern as more progamming experience is acquired. A list of the major differences can be found in the Appendices.

When first learning the PL/I language, it is very easy to be overpowered by detail. Since PL/I is by design intended for a wide range of applications, almost every feature of the language comes with many variants and options, most of which are unimportant to the beginning programmer. We have tried to ease this problem through a layered presentation that matches the presentation of the main text. When a feature of the language is first introduced, it is described in such a way as to be useful immediately to the beginning programmer. Variations and additional options are deferred until motivated by actual problem requirements.

Much has been said and written in recent years on an approach to programming known as "structured programming." Structured programming is really little more than that application of a particular discipline to the practice of programming. The evidence seems clear that students produce better programs in a shorter time span with this philosophy. The presentation in this book is consistent with the teachings of structured programming.

With this short introduction, you are now ready to begin your study of the PL/I language.

BIBLIOGRAPHY

BACKUS, J.W., et al.: "The FORTRAN Automatic Coding System" (ed., S. Rosen), in *Programming Systems and Languages,* McGraw-Hill Book Company, New York, 1967.

CONWAY, R.W., and WILCOX, T.R.: *"Design and Implementation of a Diagnostic Compiler for PL/I", Communications of the ACM,* Vol. 16, No. 3, March 1973, 169-179.

IBM System 360 Operation System: PL/I Language Specifications, IBM Corp., C28-6571-0, Data Processing Division, White Plains, N.Y., 1965.

SAMMET, J.E.: *Programming Languages: History and Fundamentals,* Prentice-Hall Inc., Englewood Cliffs, N.J., 1969.

TREMBLAY, J.P., and BUNT, R.B.: *An Introduction to Computer Science: An Algorithmic Approach,* McGraw-Hill Book Company, New York, 1979.

CHAPTER 2

FUNDAMENTAL PL/I CONCEPTS

This chapter introduces several of the fundamental concepts of programming in the PL/I language. The presentation closely follows that of Chap. 2 in the main text. The chapter begins with a simple overview of solving problems in PL/I, including data, its representation and manipulation, and the use of variables. Simple input and output operations are discussed. This discussion should be sufficient to allow the novice programmer to write very simple PL/I programs. The process of preparing a program to run under the PL/C compiler is explained and some instruction on program execution, debugging, and tracing is given. The chapter concludes with complete PL/I programs for the applications developed in Chap. 2 of the main text.