



RPG

A Programming
Language
for Today

Doris Cable

RPG

A Programming Language for Today

Doris Cable

Ventura College



Wm. C. Brown Publishers
Dubuque, Iowa • Melbourne, Australia • Oxford, England

Book Team

Editor *Earl McPeck*
Developmental Editor *Linda M. Meehan*
Publishing Services Coordinator (Production) *Julie A. Kennedy*
Publishing Services Coordinator (Design) *Barbara J. Hodgson*

Wm. C. Brown Publishers

A Division of Wm. C. Brown Communications, Inc.

Vice President and General Manager <i>George Bergquist</i>	Managing Editor, Production <i>Colleen A. Yonda</i>
National Sales Manager <i>Vincent R. Di Blasi</i>	Manager of Visuals and Design <i>Faye M. Schilling</i>
Assistant Vice President, Editor-in-Chief <i>Edward G. Jaffe</i>	Publishing Services Manager <i>Karen J. Slaght</i>
Marketing Manager <i>Elizabeth Robbins</i>	Permissions/Records Manager <i>Connie Allendorf</i>
Advertising Manager <i>Amy Schmitz</i>	

Wm. C. Brown Communications, Inc.

Chairman Emeritus *Wm. C. Brown*
Chairman and Chief Executive Officer *Mark C. Falb*
President and Chief Operating Officer *G. Franklin Lewis*
Corporate Vice President, Operations *Beverly Kolz*
Corporate Vice President, President of WCB Manufacturing *Roger Meyer*

Cover and section images by COMSTOCK Inc./Michael Stuckey

Copyediting and Production by *Lachina Publishing Services, Inc.*

Copyright © 1993 by Doris Cable. All rights reserved

Library of Congress Catalog Card Number: 92-70065

ISBN 0-697-11475-9

Forms reprinted by permission from International Business Machines Corporation.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Printed in the United States of America by Wm. C. Brown Communications, Inc.,
2460 Kerper Boulevard, Dubuque, IA 52001

10 9 8 7 6 5 4 3

Preface

This book is a text and learning guide for students for whom RPG will be a first introduction to programming languages. It is also a guide for the experienced programmer who wishes to add a useful tool to his or her knowledge base. The book will guide the reader through the differences between RPG II and RPG III. Finally, the book provides exercises and projects together with sample data to provide a sound basis for learning the language.

RPG II is available on a wide variety of minicomputers as well as mainframes and even personal computers. The fundamental concepts presented in this book apply equally to all forms of RPG, regardless of the hardware on which it will be run. The specific orientation will be toward RPG as it is run on IBM minicomputer hardware, but in general RPG II differs very little when used on different computer hardware.

The material on RPG III, however, is hardware-specific. RPG III, in its most complete form, is available only on the IBM System/38 and on the IBM AS/400. Some of the basic tools of RPG III, however, are available on the IBM System/36 and other hardware.

This book does not attempt to be all things to all people. It is not a general overview of computer programming languages. It is specific to the RPG language and its uses in a business environment.

OBJECTIVES FOR THIS BOOK ■

This book was written to help the student learn RPG II, RPG III, and RPG/400. All three versions of the language are based on similar concepts and methods. Combining discussion of the three versions of RPG into one text makes it possible to demonstrate their differences and similarities.

The book is divided into two sections. The first section (Chapters 1–11) covers all the fundamental concepts of RPG II and is intended for the first-time programming student. It assumes no prior background in programming.

The second section (Chapters 12–18) is dedicated entirely to RPG III/400 and assumes a knowledge of the fundamentals of RPG II coding. It offers a modern approach to programming with the use of externally defined files and structured programming methods. This is one of the few texts available today that provides complete coverage of RPG III/400.

Besides the usual textbook rules of programming in RPG, there are many unwritten conventions used on a day-to-day basis by professional RPG programmers. These conventions are not required by any computer hardware or by any compiler. They are standards that are generally used across industry that make the programmer's job easier and are only learned on the job. They represent methods that programmers have discovered (sometimes the hard way) to be easier to implement, more user friendly, or more universally understood. These programming standards and conventions are included in this book so that the student can enter the working world with more than just the programming theory found in the manuals.

OVERVIEW ■

This book is intended for the first-time programming student. The fundamental concepts are described in detail using a building-block approach. Each chapter presents a different type of problem that can be solved using the programming technique introduced in that chapter. Each chapter reinforces skills learned in earlier chapters. The student will be able to write a complete RPG program after completion of the first chapter. On completing the first section of the book, the student will be able to code programs proficiently in RPG II.

Each chapter begins with an outline listing the main topics to be covered in the chapter. At the end of each chapter, all new information is reviewed in a chapter summary. This summary lists new RPG rules, terms, and features. Students will find the summary useful when studying for tests.

Following the summary in each chapter are review questions. These questions relate to the chapter material and include fill-in answers which can provide a good basis to promote class discussion. These review questions also allow the student to review and study the concepts learned.

Program Debugging Exercises at the end of each chapter provide a means for the student to discover typical programming errors. Students are asked to find the errors, give possible explanations for their occurrence, and offer a solution. These projects may be assigned for outside study and are excellent for class discussion. Students who complete these exercises should be able to avoid making the same types of errors in their own programs. Program debugging is a skill that students will find useful in their future programming careers.

Provided at the end of each chapter is a programming project that allows the student to obtain "hands-on" experience with solving typical programming problems. These projects include such business applications as sales reports, employee lists, and inventory reports. Additional projects are provided in the Instructor's Manual.

Students are given an overview of the project, the input format, processing specifications, and the output format. The data for each program is included in Appendix D and is available on diskette. The appendixes also contain additional information concerning RPG programming.

SUPPLEMENTS ■

Adopters of *RPG: A Programming Language for Today* receive an Instructor's Manual that will provide insights into the material covered in each chapter. The manual contains chapter outlines, teaching tips, more detailed descriptions of some of the more difficult concepts, answers to the Review Questions, and solutions to the Debugging Exercises and Programming Projects.

A test bank will be provided containing approximately 25 questions per chapter. These test questions are also available in computerized format, on Wm. C. Brown's TestPak 3.0. TestPak is a computerized testing service that provides you with a call-in/mail-in testing program and the complete test item file on diskette for use with your IBM PC. In addition to random test generation, TestPak allows for new questions to be added, or existing questions to be edited. TestPak 3.0 is available free to adopters of *RPG: A Programming Language for Today*.

All data needed for the programming assignments is available on an IBM 3.5- or 5.25-inch diskette. The data can easily be uploaded to your computer. This allows students to spend less time entering data and more time learning programming.

This text can be used for a single-semester class in RPG to give students a good overview of a programming language as it is used in business. Using this approach, all 18 chapters could be covered.

It is also possible to use the first 11 chapters in a first-semester class and use the second section of the book for advanced students in a second semester. This would allow time to cover each chapter thoroughly and to ensure a complete understanding of the material.

It is hoped that this text will provide a solid foundation for programmers and will prepare them for the next generation of minicomputers using RPG, the language for today and tomorrow.

ACKNOWLEDGMENTS

I would like to thank the following reviewers who offered many helpful suggestions and comments: Barbara E. Koedel, Atlantic Community College; Thomas J. Abromovich, Retired from Black Hawk College, Moline, IL; Robert S. Landrum, Jones Junior College; P. Gapen, Laramie County Community College; Russell K. Lake, Parkland College—Champaign, IL; Steve Backe; Willard H. Keeling, Blue Ridge Community College; Thomas N. Latimer, Lansing Community College/Precision Computer Systems; William C. Fink, Lewis and Clark College; Rod B. Southworth, Laramie County Community College; and Catherine D. Stoughton, CIS Instructor, Laramie County Community College.



Introduction to RPG and Programming

The data processing industry is relatively young and in many ways is still searching for guidelines and standards. Hardware has gone (and continues to go) through many changes in type and architecture. At the same time, software is experiencing similar changes and continues to evolve into something more accessible for the user.

PROGRAMMING LANGUAGES ■

Many programming languages, such as COBOL, FORTRAN, PL/1, and RPG, have been around for many years. Newer languages, such as fourth-generation languages, have been developed to make the programmer's job easier. These are all known as high-level languages because they are designed to be easy for the programmer to use (unlike low-level languages, such as assembly or machine languages, which are understood better by the computer). Most high-level languages share fundamental features such as the ability to read files, do computations, and write reports. They are nearly all compiled languages, which means that a programmer must first write the program and then compile (convert) it on the computer into machine language.

OVERVIEW OF THE RPG LANGUAGE ■

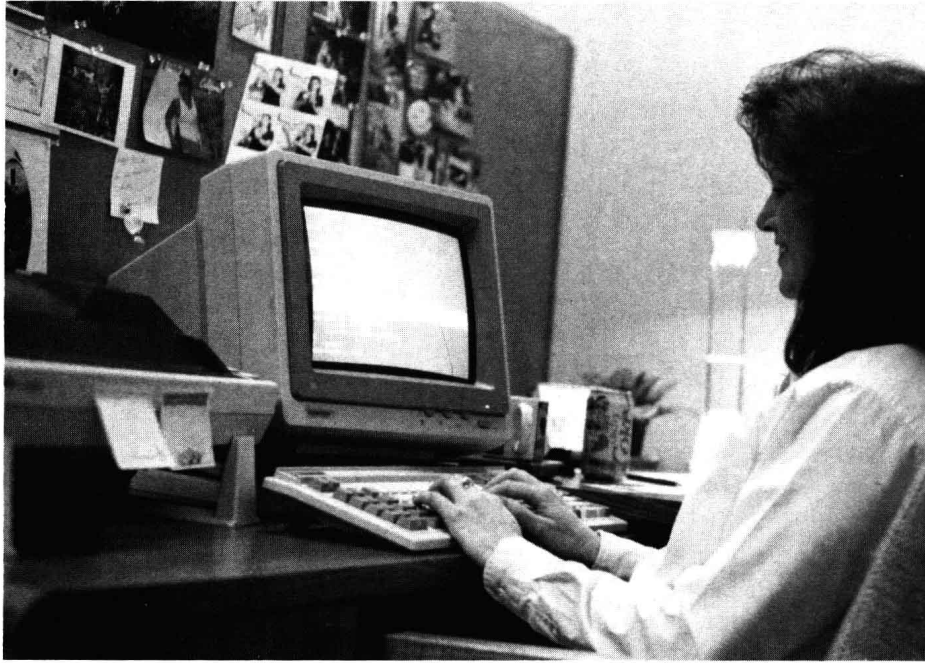
Of the many programming languages used on computers today, RPG (for Report Program Generator) is unique. Although it originated as a mere report writer in the 1960s, it has grown up to become a powerful full-service language for use in business applications. It is more widely used than any other language on minicomputers today. Interestingly, RPG is programmed in English worldwide.

RPG is defined as a problem-oriented language. Multitudes of fourth-generation languages have been developed in recent years to achieve exactly what RPG has been doing all along—solving the problems of business with a minimum of programming effort or expertise.

A Brief Glance Backward

In the early days of computers (circa 1950–1968) all business computers were large mainframes. (Minicomputers and personal computers had not yet been invented.) Programming languages were complicated, difficult to learn and to use. Often management needed only a report printed—perhaps just a listing of some file. In the languages available at that time, however, there was no such thing as a simple program.

At last, in the 1960s, IBM decided to create a language that could quickly meet the need for these simple reports. RPG was that language. It could read a data file, keep a few running totals, and write a nicely formatted report. This early version of RPG was simple to learn and easy to use,



but it had many limitations. It could not handle arrays of data, or make decisions, or update files. It could merely read and print. It did serve its purpose, but it was not a serious language.

Some people still think of RPG as this type of limited tool. On some mainframes, RPG remains in its original form doing only simple report writing.

RPG II—Enter the Minicomputer

In 1969 IBM announced the first of its minicomputers for the business world—the System/3. Minicomputers had been around since 1964 for engineering uses, but none had been available for business applications. The System/3 was the right machine for small businesses that could not afford roomfuls of expensive programmers. IBM, therefore, decided to introduce an easy language on the System/3. It would be a programming tool that anyone could learn and use quickly. That language was an updated version of RPG called RPG II.

Along with RPG II came many improvements over the original version of RPG. It became possible to make decisions, to control the actions of the program, to update files, and to use data arrays. RPG II could perform every function needed in the business environment at that time (using batch processing) and could do it faster, easier, and cheaper than any language of its time. Small businesses could have all the advantages of a large computer at far less cost.

Before long, other computer vendors realized that they would need to make versions of RPG II available for their customers if they were to compete with IBM. RPG II had become the standard of the minicomputer industry for business applications.

On-Line Interactive Processing

By the mid-1970s, management was asking for more than RPG II could provide. They didn't like to wait for reports, and they didn't want mountains of paper on their desks. They wanted on-line, interactive, instant, available data on their terminals. Most languages had been designed long before terminals had become commonplace. No language made this type of programming easy.

As a temporary solution to this problem, IBM designed a utility called CCP which could be used in conjunction with RPG II to facilitate the writing of interactive application programs.

Everyone knew that this was not a complete answer to the problem, but they also knew that the System/3 was rapidly becoming obsolete. IBM then developed the System/34 and later the System/36 as further solutions. These machines used RPG II with an on-line capability—a big improvement over CCP.

RPG III—A Truly On-Line Language

In 1979, with the System/38, IBM introduced RPG III, a language that was specifically intended to be used for programming interactive on-line applications. RPG III is a fully functional language, providing all the benefits of a completely modern structured programming language for use in the business environment. It continues to support all of the functions of RPG II with none of the aggravations of a limited language. RPG III is a language designed for communicating with terminals. It combines the best of the more modern languages (such as PASCAL) with the ease of use of a fourth-generation programming tool. Like PASCAL, it encourages the use of completely structured programming methods.

RPG III is native to the IBM System/38 architecture for which it was designed. This means that it cannot be readily adapted for use on other computers. Versions of RPG III are now available for other minicomputers (such as the IBM System/36), but there are certain limitations. The greatest benefits of RPG III can only be realized on the IBM System/38 and AS/400.

The Future

With the announcement of IBM's AS/400 model in 1988, the functions of the System/36 and the System/38 are molded into a single framework. The language used on the AS/400 is RPG/400. RPG/400 is really RPG III with some additional operations and enhancements. With IBM's commitment to integrating the AS/400 into its planned system architecture for the 1990s, it is probable that RPG II, RPG III, and RPG/400 will share a united future in the 1990s and well into the twenty-first century.

OVERVIEW OF PROGRAMMING ■

Programming consists of planning and designing the program, documenting the plan, writing the source code, compiling the code into object form, testing the program until it is free of errors, saving the final copy of the program, and finally documenting the program for future users or programmers.

Before a programmer begins writing a program, it is necessary to determine what the program is to do. The first step is to develop a plan. This may be done by a systems analyst or a programmer/analyst. It consists of discussing the project with the user (or manager) and finding out what is needed.

The second step will consist of documenting these findings by means of flowcharts, written narrative, and sample report formats so that the user can see what to expect. If all of this is acceptable, the programmer can then begin coding in the language available. Some typical documentation is shown in Figures I.2(a), I.2(b), I.2(c), and I.2(d).

Coding the Program

The language in which programmers usually write their instructions for the computer is called a source language, or source code. It consists of words and phrases that are recognizable as human language (rather than binary symbols that have meaning only to the computer). The programmer must code with great attention to detail and accuracy to ensure that the final product will be free of errors. The programmer will write the entire program in this source language before submitting it to the computer to be compiled, or translated into machine language.

Program coding today is normally entered on a terminal using a special utility program called a text editor utility or source entry utility. Figure I.3 shows what a source program might look like on a display screen using a common utility.

FIGURE I.2(a) Flowchart

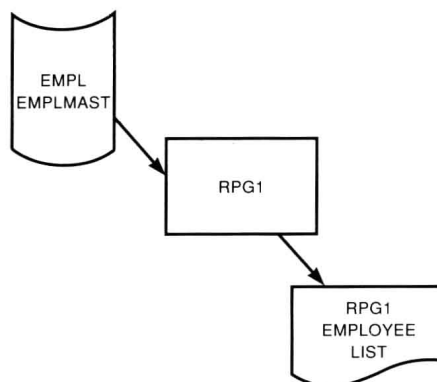


FIGURE I.2(b) Record Layout

IBM		RECORD LAYOUT	
DATE	2/9	Record Layout Application Program	FORMAT NAME EMPL
1	EMPLOYEE NAME	EMPL ID#	PAY RATE
51			
101			
151			
201			

FIGURE I.2(c) Program Narrative

Program Narrative

Input: This program uses the following input files.

EMPLMAST Employee Master List
File Length 128

EMPL Employee Time Transactions
File Length 50

Process:

EMPL records are in sequence by department. Each EMPL record is read and the employee number compared with the EMPLMAST file to determine if it is a valid employee transaction. If the employee number is invalid, the hours in the record will not be added to the totals. The invalid employee number will be printed on the report with the message "Invalid Employee Number." Processing will continue with the next record.

If the employee number is valid, the record will be printed on the report showing department, employee name, regular hours, overtime hours, double-time hours, holiday hours, sick time hours, and a total of all hours. Total hours are computed from the hours in the input record by adding all the hour fields together.

Each type of hours is accumulated (i.e., all regular hours, all overtime hours, etc.) and a total is printed for each department. This total will be printed at the end of each group of records for a department after advancing one line.

A grand total for all departments will be printed at the end of the report.

FIGURE I.2(d) Printer Spacing Chart

150/10/6 PRINT CHART

(SPACING: 150 POSITION SPAN, AT 10 CHARACTERS PER INCH, 6 LINES PER VERTICAL INCH)

PROGRAM TITLE _____

PROGRAMMER OR DOCUMENTALIST _____

CHART TITLE _____

CARRIAGE CONTROL
TAPE CHAN

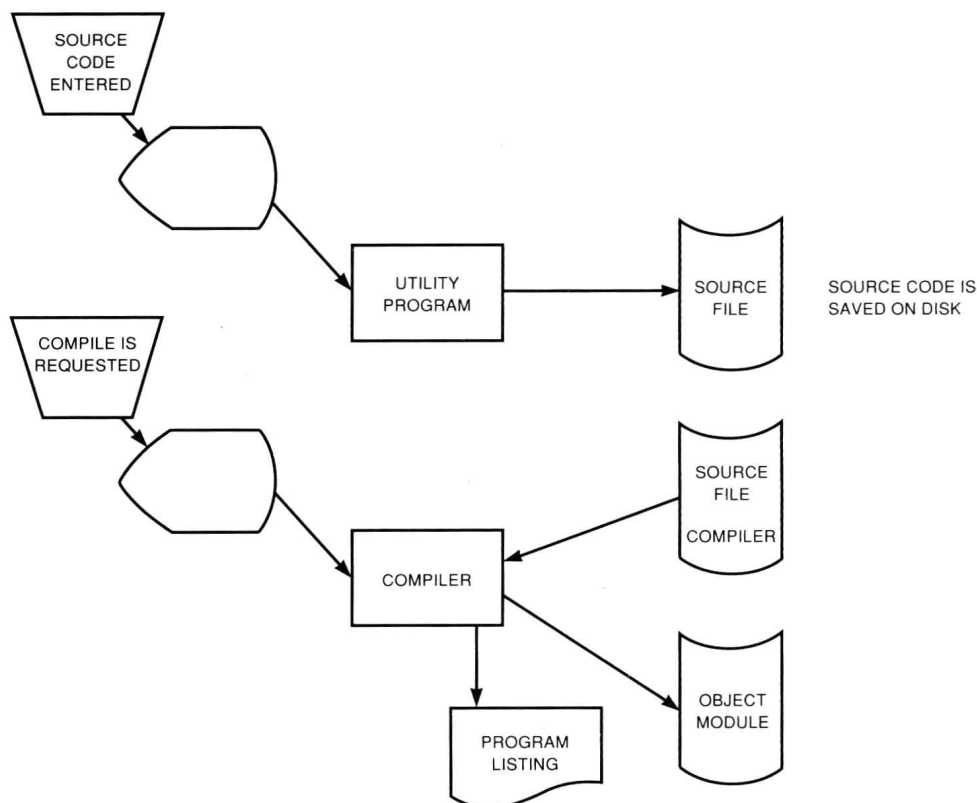
PROG ID _____

PAGE _____

DATE _____

← Fold back at dotted line

1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

FIGURE I.4 Compilation Flowchart

will show every line of code as it was coded by the programmer. The source listing will also note any errors and show them at the bottom of the listing. Note that the compiler will find only syntax errors or errors in spelling of key words or placement of the fields. It is not capable of finding errors in logic. If the errors are too severe to permit compilation, then it will say so and no object code will be produced at all. If all goes well, the source listing will state that, and the object code (an object module) will be created.

Once the object module is created, it should be saved on disk and the program will be ready for testing. The source code should probably also be saved on disk so that changes can be made later if needed.

In learning environments, it is often useful to do a test run immediately after the compiler has finished. This is referred to as a "load and go" method. Figure I.5 shows a source listing of an RPG program with its accompanying error messages.

Program Testing

Once the program is compiled successfully, it must be tested to determine whether it will produce the desired results. Acceptance by the compiler does not necessarily mean the output will be correct. Steps in testing are as follows:

1. *Preparing Test Data.* When a program is to be tested, it should be tested with data which was prepared for testing the various routines within the program. Test data should contain every possible combination of bad and good data to be sure all functions of the program are exercised. It is better to find the problems before the program goes into "live" productive use than to wait and let the errors be found by users, bosses, or instructors. Program testing should be extensive and thorough. The preparation of good test data is a difficult and tedious task, but it is an important part of programming.

2. *Desk Checking.* Sometimes problems are not easy to find. Sometimes the most careful programmer will find an obscure bug or error in the program that refuses to allow the program to perform correctly. The only solution to this dilemma is for the programmer to sit down with a listing of the program and "think" some data through the program line by line. This process, called "playing computer," is slow, but it is one of the best ways to discover what has gone wrong with the program.
3. *Program Debugging.* Programs containing bugs (or errors) should never be released into a production environment. Nothing will give computers a bad name faster than invalid data showing up on someone's report or screen. Unexpected program failures greatly reduce the credibility of the data processing department. It is the responsibility of every programmer to make certain that no program is released until it has been completely and thoroughly tested and found to be absolutely bug free.

FIGURE I.5 Source Listing

SEQUENCE NUMBER	1	2	3	4	5	6	7
	67890123456789012345678901234567890123456789012345678901234						
100	F*	TITLE:	SALES ANALYSIS REPORT				
200	F*	DATE:	3/11	AUTHOR:	D.CABLE		
300	F*	DESCRIPTION:	PRINT REPORT OF ALL SALES FOR MONTH				
400	F*****						
500	F*	MODIFICATIONS:					
600	F*	NO.	DATE	INIT	DESCRIPTION		
700	F*	XX/XX/XX	XXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX			
800	F*****						
NAME OF PROGRAM WILL BE ILLO5 IN LIBRARY CABLE							
	H						
* 1019		ALL DECIMAL DATA ERRORS IGNORED					
900	FSALEMASTIP	F	512	DISK			
1000	FREPORT	O	F	132	PRINTER		
1100	F*****						
1200	ISALEMASTAA	O1					
1300	I			1	2	SACMPY	
1400	I			3	50	SALOC	
1500	I			6	6	SATYPE	
1600	I			7	21	SAPNO	
1700	I			22	230	SAMTH	
1800	I			24	280	SAQTY	
1900	I			29	352	SAPRC	
2000	C		TIME		TYME	60	
2100	OPRINT	H	101	1P			
* 6001	6001-*****	.					
* 6035	6035-*****						
2200	O			6	*ILLO05*		
2300	O			75	*P.C. SOLUTIONS*		
2400	O			124	*PAGE*		
2500	O			132			
2600	O	H	2	1P			
* 6001	6001-*****	.					
* 6035	6035-*****						
2700	O			8	*RUN DATE*		
2800	O		UPDATE Y	17			
2900	O		TYME	26	* : : *		
3000	O			60	*MONTHLY SALES ANALYSIS*		
3100	O	H	2	1P			
* 6001	6001-*****	.					
* 6035	6035-*****						

FIGURE I.5 Continued

SEQUENCE NUMBER	1	2	3	4	5	6	7
	6789012345678901234567890123456789012345678901234						
3200	0				10	*COMPANY*	
3300	0				20	*LOCATION*	
3400	0				26	*MONTH*	
3500	0				33	*QUANT*	
3600	0				42	*TOTAL SALES*	
3700	0	0	1	01			
* 6001	6001-*****	.					
* 6035		6035-*****					
3800	0		SACMPY	12			
3900	0		SALOC	20			
4000	0		SAMTH	25			
4100	0		SAQTY K	32			
4200	0		SAPRC K	42			
***** E N D O F S O U R C E *****							

* 7086 900 RPG PROVIDES BLOCK/UNBLOCK SUPPORT FOR FILE SALEMAST.
 * 7064 1000 REPORT FILE NOT REFERENCED FOR OUTPUT

CROSS-REFERENCE LISTING

FILE/RCD	DEV/RCD	REFERENCES (D=DEFINED)
PRINT	**UNDEF**	2100
02 REPORT	PRINTER	1000D
01 SALEMAST	DISK	900D 1200

FIELD	ATTR	REFERENCES (M=MODIFIED D=DEFINED)
PAGE	P(4,0)	2500
SACMPY	A(2)	1300D 3800
SALOC	P(3,0)	1400D 3900
SAMTH	P(2,0)	1700D 4000
* 7031 SAPNO	A(15)	1600D
SAPRC	P(7,2)	1900D 4200
SAQTY	P(5,0)	1800D 4100
* 7031 SATYPE	A(1)	1500D
TYME	P(6,0)	2000D 2900
UPDATE	P(6,0)	2800

INDICATOR	REFERENCES (M=MODIFIED D=DEFINED)
LR	900D
01	1200M 3700
1P	2100 2600 3100

MESSAGES

MSGID	SEV	NUMBER	TEXT
* QRG1019	00	1	IGNDECERR(*YES) SPECIFIED ON COMMAND. NO DECIMAL DATA ERRORS
* QRG6001	40	4	FILE/RECORD NAME NOT VALID, NOT DEFINED, IGNORED, OR MISSING.
* QRG6035	20	4	VALID TYPE FOUND
* QRG7031	00	2	SPACE OR SKIP MUST BE SPECIFIED ONLY FOR PROGRAM DESCRIBED FI
* QRG7064	40	1	ASSUMED.
* QRG7086	00	1	NAME OR INDICATOR NOT REFERENCED.
			PROGRAM FILE NOT REFERENCED. FILE IGNORED
			RPG WILL HANDLE BLOCKING FUNCTION FOR THE FILE. INFDS CONTENT
			ARE TRANSFERRED.

FIGURE I.5 *Continued*

```
MESSAGE SUMMARY

TOTAL      00      10      20      30      40      50
  13        4        0        4        0        5        0

  42 RECORDS READ FROM SOURCE FILE
SOURCE RECORDS INCLUDE  33 SPECIFICATIONS,      0 TABLE RECORDS, AND      9 COMMENTS

QRG0008 COMPILE TERMINATED. SEVERITY 40 ERRORS FOUND IN PROGRAM

***** END OF COMPI LATION *****
* QRG1020          ERROR OCCURRED CREATING OR UPDATING DATA AREA RETURN CODE. COMPI
```

Documentation

Documentation is the process of recording all information related to a given program (or system of programs) in such a way that users, managers, other programmers, or data processing auditors will be able to quickly understand exactly what it is that your program is supposed to be doing. At a minimum it should include:

1. Record layout forms
2. Printer (or screen) layouts
3. A program narrative describing the routines used in the program
4. A flowchart and/or pseudocode outline of the program logic
5. The final copy of the source listing (produced at final compilation time)
6. Sample reports (or screens)
7. User instruction manual

Documentation is an often-neglected part of a programmer's job but a crucial one if a company is to continue to function successfully over a period of time.

CONCLUSION ■

RPG is a programming language that can serve as an excellent introduction to programming. It is a language that is best suited for business applications and is widely used on mid-range computers. RPG has been used as a business programming language since the early 1960s and is predicted to remain popular with mid-range users for many years to come.

This text presents an introduction to both RPG II and RPG III. It explains the fundamentals of programming along with standard conventions practiced in a business programming environment.

Contents

Preface vii

Introduction to RPG and
Programming ix

SECTION 1 RPG II

1

Input and Output Processing 3

Program Processing Steps 4
RPG Coding Formats 6
Common Entries on Coding
Forms 8
The Control Specifications Form
(Header Format or
H Specification) 8
The File Description Specifications
Form (F Specification) 9
 *Defining the Input File 9 / File
 Type/File Designation Entries 10
 / File Format 10 / Record
 Length 10 / Device 10 /
 Defining the Output File 11*
The Input Specifications Form
(I Specification) 11
 *Form Type/Filename Entries 11 /
 Sequence Number Option Entries
 (Columns 15–16) 12 / Record
 Identifying Indicator Entry (Columns
 19–20) 12 / From-To Entries
 13 / Field Location Entries
 (Columns 44–47 and 48–51) 13
 / Field Name Entries (Columns
 53–58) 13*
The Output Specifications Form
(O Specification) 14
 *Filename Entry (Columns 7–14)
 14 / Type Entry (Column 15) 14
 / Space Entry (Columns 17–18)
 14 / Output Indicators (Columns
 24–25, 27–28, or 30–31) 14 /
 Field Name (Columns 32–37) 14
 / End Position (Columns 40–43)
 15*
The RPG Fixed Logic Cycle 15
Incorporating Comments Within a
Program 17

*Why Document? 17 / How Much
Documentation Is Enough? 18 /
How Do You Put Comments into an
RPG Program? 18 / Conventions
for Professional Program
Documentation 19*
Compiling a Program 20
Summary 23
Review Questions 24
Exercises 24
*DEBUGGING EXERCISES 25
PROGRAMMING PROJECT 28*

2

Arithmetic Operations and Report Formatting 29

General Rules for Coding
Calculation Specifications 29
 *Addition 30 / Subtraction 31 /
 Multiplication 31 / Division 32
 / Division Remainders 32 /
 Rounding a Result Field
 (Half-adjusting) 33 / Defining a
 Constant 34 / Calculating
 Cumulative Totals 35 / Summary
 of Arithmetic Functions in RPG 35*
Sample Program Using Arithmetic
Calculations 36
 *Input Data 36 / Output Data
 36 / File Description
 Specifications 37 / Input
 Specifications 37 / Calculation
 Specifications 38 / Output
 Specifications 40 / Field Editing
 (Edit Codes) 42 / Other Edit
 Codes 44 / Editing Dollar Signs
 44 / Printing Total Lines 45*
Summary 46
Review Questions 47
Exercises 47
*DEBUGGING EXERCISES 48
PROGRAMMING PROJECT 52*

3

Computer Logic and Processing Multiple Record Types 54

Fundamental Logic Concepts
(Compare) 54

Using Indicators in RPG 55
The Compare Operation 56
 *Comparing Alphabetic or
 Alphanumeric Data Fields 56 /
 Comparing Alphabetic or
 Alphanumeric Fields of Different
 Lengths 57 / Comparing
 Numeric Data Fields 57 /
 Comparing Numeric Fields of
 Different Lengths 59 / Collating
 Sequence for Comparing 61 /
 Comparing Literal Values 61 /
 Summary of the Compare
 Operation 62*
Processing Multiple Record
Types 62
 *Use of Input Indicators 62 /
 Field Record Relation Indicators
 64 / Calculation Specifications
 66*
Branching Within Calculations 67
 *Using the GOTO Operation 67 /
 Using Subroutines 68 / GOTOs
 Revisited 69*
Summary 70
Review Questions 71
Exercises 71
*DEBUGGING EXERCISES 72
PROGRAMMING PROJECT 77*

4

Control Breaks (The RPG Cycle) 79

Sample Program Using Level
Breaks 79
 *Sequence of Input Data 79 /
 Level Break Totals 80 / Control
 Level Indicators 80*
The Effect of the RPG Cycle on
Level Breaks 85
Summary Reports 87
Summarized Output Files 87
Multiple Output Files 89
Summary 90
Review Questions 91
Exercises 91
*DEBUGGING EXERCISES 92
PROGRAMMING PROJECT 96*

5

Multiple Level Breaks 98

- Sample Program Using Multiple Level Breaks 98
 - Input Specifications* 98 /
 - Calculation Specifications* 100 /
 - Output Specifications* 103
- The RPG Cycle Revisited 103
 - Group Indication* 104 / *Zeroing Out a Total Field* 107
- Standards for Report Formatting 110
 - Report Headings* 110 / *Report Totals* 112
- Sample Program Defining Report Headings 112
 - Report Identification* 112 /
 - Report Title* 112 / *Page Number* 113 / *System Date and Edit Codes* 113 / *System Time and Edit Words* 114 / *Summary of Report Formatting* 115
- Summary 115
- Review Questions 116
- Exercises 116
- DEBUGGING EXERCISES 117
- PROGRAMMING PROJECT 122

6

Exception Output Processing 124

- Sample Program Using Exception Output Processing 125
 - File Description and Input Specifications* 125 / *Calculation Specifications (Program Loops)* 126 / *Output Specifications* 128
- Page Overflow 128
 - Fetch Overflow* 129 / *Sample Program* 130 / *Internal Control (Line Counting)* 134
- Moving Data (MOVE and MOVEL) 135
 - Moving Fields of Different Lengths with MOVE* 136 / *Moving Fields of Different Lengths with MOVEL* 136 / *Alphanumeric/Numeric MOVE or MOVEL* 138 / *Moving Literal Values* 138 / *Initializing a Field to Blanks* 139
- Summary 140
- Review Questions 141
- Exercises 141
- DEBUGGING EXERCISES 142
- PROGRAMMING PROJECT 146

7

Table Processing 148

- Types of Table Applications 148
- Benefits of Table Processing 149
- Extension Specifications 151
- Table File Formats 151
- Types of Tables (Internal and External) 151
- Internal (Compile Time) Table Entries 151
- Sample Program 1 Using Internal or Compile Time Tables 152
 - File Extension Specifications* 152 / *Record Layout* 154 /
 - Compiling Table Information* 155 / *Calculation Specifications for Processing Table Data* 156
- External (Pre-Execution Time) Table Entries 156
- Sample Program 2 Using an External or Pre-Execution Time Table 158
 - File Description Specifications* 160 / *Extension Specifications* 161 / *Record Layout* 161 /
 - Input Specifications* 161 / *Calculation Specifications* 162 /
 - Output Specifications* 162 / *Table Data File Maintenance* 162
- The Time at Which Table Data Is Available 163
- Avoiding Common Errors 163
- Alternating Tables 164
- Sample Program 3 Using Alternating Table Data 164
 - Input Specifications* 164 /
 - Calculation Specifications* 164 /
 - Output Specifications* 166 /
 - Multiple Tables* 166 / *Printing a Table* 166
- Summary 167
- Review Questions 168
- Exercises 168
- DEBUGGING EXERCISES 169
- PROGRAMMING PROJECT 174

8

Array Processing 176

- Rules for Constructing Arrays 176
- Examples of Arrays 177
- Extension Specifications for Arrays 177
- How an Array Is Loaded 177
 - Compile Time Arrays* 177 /
 - Pre-Execution Time Arrays* 179 /
 - Execution Time Arrays* 180
- Array Processing 181

Accessing an Entire Array 181 /

Accessing Individual Elements of an Array 181

- Sample Program 1 Using Three Types of Array 182
 - The Compile Time Array* 183 /
 - Printing the Entire Array* 184 /
 - Pre-Execution Time Array* 184 /
 - Execution Time Array* 184
- Sample Program 2 Showing Other Uses of Arrays 190
 - Compile Time Array* 191 /
 - Execution Time Arrays* 191 /
 - Calculation Specifications* 192 /
 - Using an Array Index* 193 /
 - Computed Index Using Exception Output* 193 / *Accessing an Array by Index (Looping)* 193
- Sample Program 3 Using LOKUP with an Array 197
 - Special Operations for Array Data* 201
- Sample Program 4 Using XFOOT 203
- Sample Program 5 Using MOVEA 205
- Summary 208
- Review Questions 209
- Exercises 209
- DEBUGGING EXERCISES 210
- PROGRAMMING PROJECT 215

9

Processing Multiple Sequential Files 217

- Types of Files 217
 - Master Files* 218 / *Transaction Files* 218 / *Table Files* 218 /
 - History Files* 219 / *Backup Files* 219
- Tape Media 220
 - Sequential File Processing* 220 /
 - Record Blocking* 221 /
 - Processing a Master File* 223
- Method for Updating Sequential Files 223
 - Transaction Types* 224 /
 - Matching Record Logic* 224
- Sample Program Using Sequential File Processing 225
 - Record Layouts* 225 / *File Description Specifications* 226 /
 - Input Specifications* 227 /
 - Sequence Checking* 228 /
 - Matching Record Indicator (MR)* 228 / *Output Specifications* 229
- Packed Decimal Data 231
 - Reading and Writing Numeric Fields in Packed Decimal Format* 232
- Summary 233
- Review Questions 234

Exercises 234
DEBUGGING EXERCISES 235
PROGRAMMING PROJECT 240

10

Multiple File Processing (Indexed Files) 242

Direct Access Storage Devices
(DASDs) 242
 Disk Access Concepts 243 / File
 Organization on DASDs 244
Sample Program 1 Creating an
Indexed File 246
 File Formats 246 / File
 Description Specifications 247 /
 Input Specifications 247 /
 Output Specifications 248 /
 Inquiry Programs 249
Sample Program 2 Using Inquiry
Programs 250
 Chaining to an Indexed File 250
 / Printed Output from an Indexed
 File 251 / File Maintenance
 (Data Updating) 252 /
 Reorganizing a File 253
Sample Program 3 Updating an
Indexed File 253
Sample Program 4 Demonstrating
Sequential Retrieval from an
Indexed File 258
Record Address Files 258
Summary 261
Review Questions 262
Exercises 262
DEBUGGING EXERCISES 263
PROGRAMMING PROJECT 267

11

Additional Subjects 269

Line Counter Specifications 269
Binary Data 270
Editing Data Operations 272
 Move Zone 272 / Test Zone
 272 / Test Numeric 273
Reversing a Sign 274
Testing the Value of a Bit 274
Special Indicators 276
 Halt Indicators 276 / External
 Switches 276
Indicators—A Review 279
 Record Identification Indicators
 279 / Field Record Relationship
 Indicators 279 / Resulting
 Indicators 279 / Overflow
 Indicators 279 / Level Break

Indicators 279 / Matching
Record Indicators 279 / Last
Record Indicator 280 / Halt
Indicators 280 / External
Indicators (Switches) 280 / Even
More Indicators 280

External Subroutines 280
Look-Ahead Feature 282
FORCE Operation 283
READ Operation 284
Debugging Methods 284
DSPLY Operation 286
Accessing Multiple Indexed Files
287
Naming Conventions 289
 Program and File Names 289 /
 Field Names 290
Summary 290
Review Questions 291
DEBUGGING EXERCISES 292

SECTION II RPG III/RPG 400

An Introduction to RPG III/ RPG 400 299

12

Overview of RPG III/400 File Concepts 301

Introduction to RPG III/400 File
Concepts 301
Benefits of Externally Defined
Files 302
 Programmer Productivity 302 /
 Standardized Naming Conventions
 302 / File Changes Made Easier
 303 / Automatic Documentation
 303
The Data Description Specification
(DDS) 303
Using Externally Defined Files
305
 File Description Specifications
 305 / Input Specifications 306
Explanation of Physical/Logical
Files 306
 Physical Files 306 / Logical
 Files 306 / Unique Keys in a
 File 308 / Multiple Indexes 309
File Format Names for Data File
Descriptions 309
File Format Names for Display File
Descriptions 311
File Members 311
Summary of File Organization 313

Redefinition of Fields 313
Data Structures 314
Summary 316
Review Questions 317
DEBUGGING EXERCISES 318
PROGRAMMING PROJECT 321

13

Calculation Enhancements in RPG III/400 323

Reserved Words for RPG III/400
324
Arithmetic Operations in RPG
III/400 324
Structured Programming Concepts
for RPG III/400 325
 Structured Selection
 (If...Then...Else) 325 /
 Structured Iteration (DO Loops)
 330 / Difference Between a Do
 While and a Do Until 332 / Use
 of Indicators with DOxxx 333 /
 Compare and Branch Operation
 (CABxx) 334 / Case Structure
 (CASxx) 334 / New Methods for
 Using Indicators (Setting Flags)
 336 / Calling a Program from
 Within an RPG III/400 Program
 336 / Defining Field Attributes in
 RPG III/400 338
Summary 340
Review Questions 341
DEBUGGING EXERCISES 342
PROGRAMMING PROJECT 347

14

File Management Techniques in RPG III/400 349

Full Functional File Processing
349
Reading a Sequential File Using
SETLL and READ Operations 350
Defining a File Key in RPG III/400
(KLIST and KFLD) 351
New Methods for Accessing Files
and Retrieving Records 353
 Using SETLL and READ Operations
 (with a Key) 353 / Using SETGT
 and READP Operations 354 /
 Using SETLL and READE
 Operations 355 / Differences
 Between CHAIN and READ
 Operations 355
Creating Your Own Processing
Cycle 356
Exception Output Labels 356