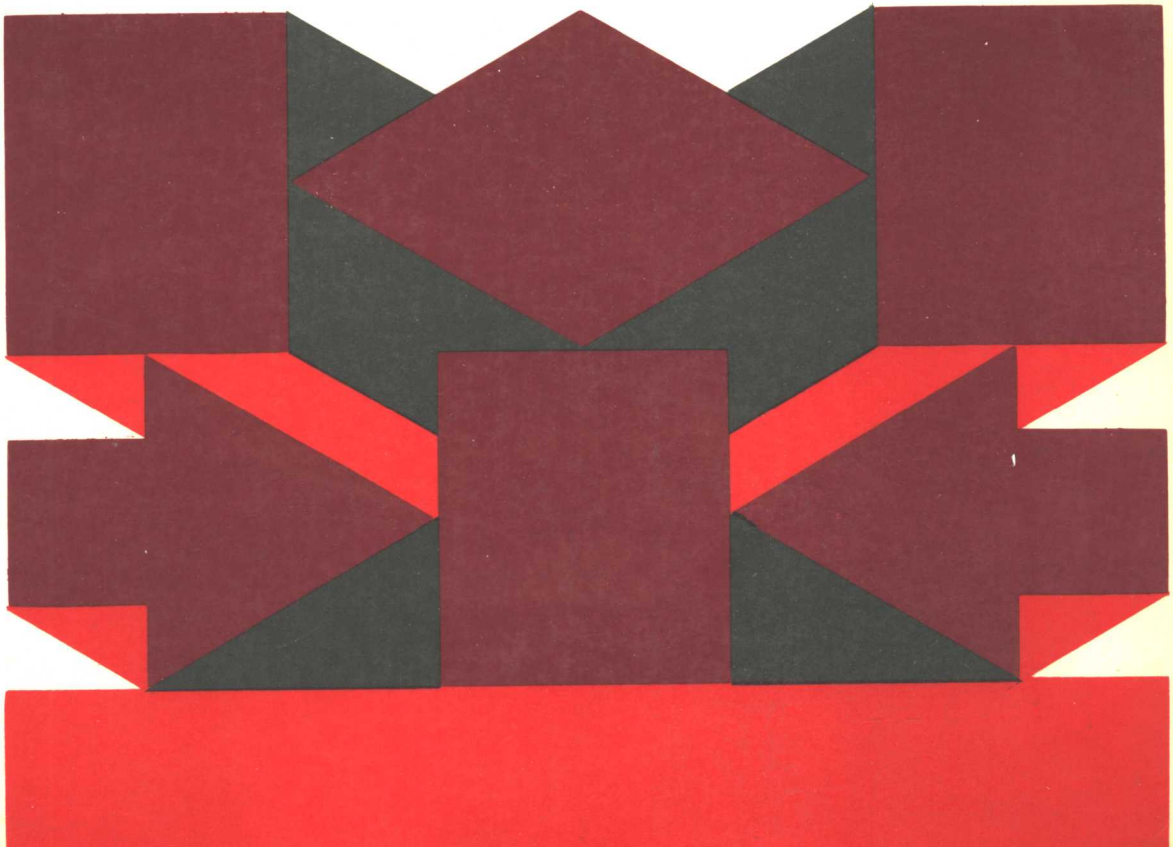
LAURA SARET

DATA PROCESSING LOGIC



DATA PROCESSING LOGIC

Laura Saret

Oakton Community College

McGraw-Hill Book Company

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

DATA PROCESSING LOGIC

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

ISBN 0-07-054723-8

This book was set in Times Roman by University Graphics, Inc.
The editors were Eric M. Munson and Jonathan Palace;
the designer was Rafael Hernandez;
the production supervisor was Charles Hess.
The drawings were done by Danmark & Michaels, Inc.
R. R. Donnelley & Sons Company was printer and binder.

Library of Congress Cataloging in Publication Data

Saret, Laura.
Data processing logic.

Includes index.
1. Electronic digital computers—Programming.
2. COBOL (Computer program language) 3. Flow charts.
I. Title.
QA76.6.S265 1984 001.6'1 83-24865
ISBN 0-07-054723-8

Preface

This book is one of the few available on structured data processing logic. The book was developed from lecture notes and handouts which have been used successfully in a course on data processing logic and programming techniques at Oakton Community College.

Although there are many books available on computer programming languages, most do not teach students how to program. The available books concentrate on a particular programming language, with a minimal amount of text being devoted to problem solving. This book concentrates on problem solving, with little emphasis placed on programming languages.

The approach of the book is to teach via examples. All the rules for programming can be listed, but unless a student can see how to apply the rules, he or she will not learn. There are 41 realistic business examples used in this book to illustrate the various topics. Unlike other logic books, all examples are completely solved, so the student can see the entire solution process. All examples in the book relate to business problems. Data names used in this book are COBOL names, because they are more "English-like" than the names allowed by other languages. All examples, however, can easily be converted to FORTRAN or BASIC by changing names to adhere to FORTRAN or BASIC rules.

The book is organized by topic. Chapters 1 to 4 should be covered in sequence. After Chapter 4 is completed, however, the remaining chapters can be completed in any order. After each chapter, exercises and logic problems are included for the student to reinforce the concepts introduced in the chapter.

The author has found that there is sufficient material in this book for a 3-hour 1-semester college course in programming logic. In addition, the book can be used:

1. As a supplement to a programming language course. Many programming students and instructors have indicated that the examples in this book relate to the types of problems assigned in a beginning programming class.
2. As a high school course in the fundamentals of computer programming.
3. In an industry course to train computer programmers.
4. In short courses for business and management people who desire an introduction to computer programming and how a computer works.
5. For programmers in industry desiring to improve their programming skills using structured techniques.
6. For the person desiring to purchase a computer for his or her business and who is interested in learning how to describe the solutions to problems in a way that makes them easy to program.

This book assumes no previous knowledge of data processing. It is expected that the reader can use simple arithmetic tools, but no previous formal training in advanced mathematics is required.

With almost all books, there are people who contribute thoughts, constructive criticism, time, energy, and emotional support to the author, and yet do not receive the recognition they deserve because of space limitations. This book is no exception. Nevertheless, I would like to thank Fran Perlman, a former student, who reviewed the book as a favor to me; my other reviewers; McGraw-Hill for seeing this as a worthwhile project and publishing the book; and my students at Oakton Community College whose questions, comments, and academic successes and failures did much to develop this book. Finally, I would like to thank my husband, Larry, my children, Marla and Jeffrey, my sister, Sheri, and my parents, Marilyn and Harvey, who always expressed substantial interest in the book and encouraged me to start and finish it, probably so that they could see their names in this preface.

Laura Saret

To the Student

You will find this book useful whether you are a beginning data processing student, a manager in a corporation, a data processing professional wishing to update your skills and learn structured programming techniques, or a person curious about how to get the computer to do what you want it to.

Learning programming logic is not easy. For many of you, this is your first experience in learning to "think like a computer." You must unlearn the complicated thought processes you have become used to in dealing with everyday life and learn to tell the computer, in simplified terms, exactly what to do step by step.

The intent of this book is to teach you how to program without emphasizing any programming language. You will find this book useful in any programming language class you take, since writing a computer program merely involves translating programming logic into a language that the computer can understand.

This book assumes no previous knowledge of data processing. It is assumed that you can use simple arithmetic tools, but no training in advanced mathematics is required. If you have had an introductory course or previous experience in data processing, many of the concepts in the beginning of the book will be a review.

I hope that you will enjoy learning to program, and whether you end up as a data processing professional, as a manager, or as a user of video games, word processors, and other packaged programs, that you will appreciate the time, effort, and creativity involved in generating the logic of a computer program.

Laura Saret

Contents

CHAPTER 1

Data Processing Terminology

DATA PROCESSING

Data processing involves taking raw data as input to a system, processing the data, and producing information as output.

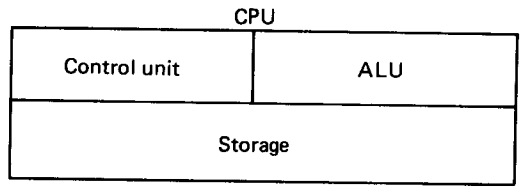
Input data → process data → output information

The basic requirements, therefore, of any data processing system (computerized or not computerized) are input, processing, and output. Consider, as an example, a system used to compute student grades at the end of a semester. Input consists of exam and assignment scores for each student in a class. The scores for each student are processed to compute his or her average and grade. Output will consist of a grade for each student.

Input exam and assignment scores → process scores to compute average and grade → output grades

HARDWARE

A computer is a machine used for data processing. It is a system of devices capable of processing data which has been entered and supplying the resulting information as output. As such, a computer system must consist of input devices, output devices, and a processing device. These devices are referred to as *hardware.* The processing device used by the computer is called the *central processing unit (CPU).* The CPU is what is usually thought of when you hear the term "computer." It consists of a control unit, an arithmetic logic unit (ALU), and a storage unit.



The *control unit* can be thought of as the "brain" of the computer. Among other things, it determines which program instruction to execute, interprets the instruction, and causes the instruction to be executed. The *arithmetic logic unit* is responsible for doing arithmetic computations (for example, ADD A TO B), data transfers (moving data from one area of storage to another), and logical comparisons (such as IS A = 3?) as directed by the control unit. The *storage unit* contained within the CPU is referred to as *primary* or *main storage.* It is used to store the data and instructions (programs) needed by the computer. Another term used for storage is *memory.* The memory size of a computer refers to the storage capacity of the CPU.

Because of space and cost constraints, main memory is not the only storage used

by a computer. Imagine the size of the CPU that would be needed if the Internal Revenue Service, for example, stored all its programs and data in main memory! *Secondary* or *external storage* media are used as additional storage or memory for the computer. However, in order for the CPU to use data or instructions, they must be in main memory. The computer can't process data or use instructions that are on a secondary storage medium.

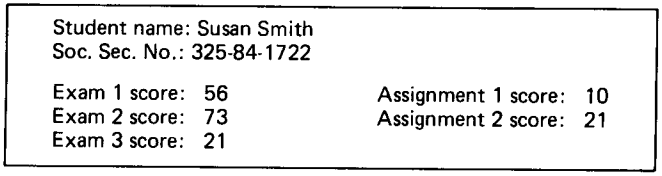
Examples of secondary storage media include punched cards, magnetic disks, and magnetic tapes. *Input/output (I/O) devices* are hardware used to get data and instructions from secondary to primary storage and vice versa. I/O devices include such things as card readers, magnetic tape drives, magnetic disk drives, and printers.

DATA ORGANIZATION

Data are organized as files in secondary storage. A *file* consists of a group of related *records*. Examples of files include payroll files, student files, and accounts receivable files. A payroll file, for example, will probably have a record for each employee of the company. A student file may have a record for each student in the school. Records are composed of fields. A *field* is a group of consecutive storage positions reserved for a specific type of data, e.g., name, address, or social security number. Fields are composed of *characters*. In the name field which contains the name JONES, there are five characters: J, O, N, E, and S.

Fields are said to be alphabetic, numeric, or alphanumeric (alphameric), depending on the characters stored in the field. An *alphabetic field* can contain the characters A through Z, and spaces or blanks. A *numeric field* can contain the characters 0 through 9, a decimal point (.), and an optional sign (+ or −). An *alphanumeric field* can contain alphabetic characters, numeric characters, and special characters such as a dollar sign ($), comma (,), quote ("), apostrophe ('), percent sign (%), hyphen (-), etc.

Consider again the earlier example of computing students' grades. Suppose we keep track of the scores on an index card file. We will have an index card (record) for each student (see sample below). The fields in the record include student name, social security number, Exam 1 score, Exam 2 score, Exam 3 score, Assignment 1 score, and Assignment 2 score.



The student name consists of letters and a blank between the first and last name. It is an alphabetic field. The social security number contains numbers and hyphens, and is therefore an alphanumeric field. Each of the "score" fields is numeric.

SOFTWARE

Software is the term used to describe computer programs. A *computer program* is a set of instructions written in a specific sequence to direct the computer to perform certain predetermined tasks. In the student grade example, we can write instructions that cause the computer to input a record for a particular student, compute the grade based on the contents of the exam and assignment score fields, and print (output) the student's name and his or her grade. Then we can write an instruction to have the computer process the next record and continue doing so until we run out of records.

THE PROGRAMMING PROCESS

In general, the systems analyst working on a particular project will provide program specifications to be used by the programmer. The specifications will include such things as a statement of the problem, a description of all inputs and outputs, and details of the processing that is to take place. The programmer can then write a program which will meet the specifications.

There are seven steps in the programming process:

1. *Understand the program specifications.* Before doing anything else, the programmer needs to become familiar with what the program is trying to accomplish and with the input and output files designed by the systems analyst.
2. *Plan the program logic.* Suppose you are going to drive from Chicago to New York on a trip. Most people would not just get into the car and start driving; they would plan an itinerary, figure out what to pack, phone ahead for reservations, etc. In much the same way, a programmer should not begin by coding. During the planning step, a programmer decides how best to meet the program specifications set forth by the analyst. The programmer usually divides the program into modules or subroutines and uses design techniques, such as flowcharts, pseudocode, and hierarchy charts, to help plan the solution.

   A *subroutine* or *module* is a set of instructions that will accomplish a specific function of the program, e.g., printing headings, doing totals, accessing a table, etc. The subroutine takes data from the main (calling) routine or program, executes the instructions of the subroutine, and sends the results back to the main program. The advantages of using subroutines include the following:
   a. A decrease in the cost and time of testing and modifying programs, because the programmer can test or change a subroutine without affecting the logic of the rest of the program.
   b. A programmer can write a standardized routine which can then be used in other programs. This reduces programming cost by eliminating the need to write duplicate code in every program.
   c. Several programmers are able to work on the same program simultaneously since each may be assigned a separate module to write.
3. *Code the program.* After the solution is planned, the next step is to code the solution in a programming language such as ASSEMBLER, COBOL, or FOR-

TRAN. Coding involves translating the solution into a language that the computer can understand. The coded program is called the *source program.*

4. *Get the program into machine readable form.* This is done by punching the program onto cards, entering it via a computer terminal, or using a key-to-tape or key-to-disk machine.
5. *Compile or assemble the program.* Computers cannot execute source programs. They work electronically and, therefore, must represent data and instructions in binary (on-off) form. Compilation or assembly involves the conversion of the source program to an *object* or machine language (binary) *program.* The conversion is done by a program called a *compiler* in the case of a high-level language such as COBOL or an *assembler* in the case of an assembly language program. The input to the compiler or the assembler is the source program. Outputs from the compiler or the assembler include the object program and a list of diagnostics (errors). The errors listed are *syntax* (or "language") *errors.* Examples of syntax errors include the use of a variable name that contains too many characters, writing GO TO STEP-10 when the programmer used the name STEP10 without a hyphen when labeling the instruction, and violating punctuation rules of a language. *Logic errors* are *not* listed during compilation. An example of a logic error is using an ADD instruction instead of a SUBTRACT instruction in a COBOL program. Even though both instructions are valid, very different results will be obtained during program execution. Logic errors can only be detected by testing the program.
6. *Test the program.* Once all the syntax (language) errors have been corrected, it is time to test the logic of the program. This involves using sample data as input, executing the program, and checking the results.
7. *Document the program.* Documentation is listed as the last step in the programming process, but documentation, i.e., providing a written commentary of steps 1 to 6, should be done as the steps are done. Some of the things to include in documentation are the specifications provided by the analyst, flowcharts and other planning aids, program listings, sample test data, and test results.

This book concentrates on the planning step. Program flowcharts, hierarchy charts, and pseudocode will be used to aid in the development of program logic.

ARITHMETIC AND LOGIC SYMBOLS

The following arithmetic and logic symbols will be used in this book:

Symbol	*Meaning*	*Example*
$=$	Equals	$A = B$ means A is equal to B.
$<$	Less than	$A < B$ means A is less than B.
$>$	Greater than	$A > B$ means A is greater than B.
$\leq$	Less than or equal to	$A \leq B$ means A is less than or equal to B.
$\geq$	Greater than or equal to	$A \geq B$ means A is greater than or equal to B.
$\neq$	Not equal	$A \neq B$ means A is not equal to B.

Symbol	*Meaning*	*Example*
:	Compare	$A : B$ means compare the value of A to the value of B.
+	Addition	$A + B$ means add A and B.
−	Subtraction	$A - B$ means subtract B from A.
*	Multiplication	$A * B$ means multiply A and B.
/	Division	A/B means divide A by B.
**	Exponentiation	$A ** B$ means A to the B power.

QUESTIONS AND EXERCISES FOR REVIEW

1. Define the following terms:
   a. Data processing
   b. Computer
   c. Hardware
   d. Primary storage
   e. Memory
   f. Secondary storage
   g. File
   h. Record
   i. Field
   j. Character
   k. Alphabetic field
   l. Numeric field
   m. Alphanumeric field
   n. Software
   o. Computer program
   p. Subroutine
   q. Source program
   r. Object program
   s. Compiler
   t. Documentation
2. What are the basic requirements of any data processing system?
3. What are the three parts of the CPU? Describe the function of each.
4. What are I/O devices?
5. Describe the relationship between files, records, fields, and characters. Give an example to illustrate.
6. What are the steps of the programming process? Describe each.
7. What is the difference between a logic error and a syntax error?
8. How does a source program become an object program?
9. How would you express each of the following using arithmetic and logic symbols?
   a. 5 is equal to C
   b. 7 is not equal to 12
   c. Add 3 and 6
   d. 4 is less than 11
   e. A is less than or equal to 15
   f. Subtract 11 from 23
   g. Raise 5 to the power of 2
   h. Divide 87 by 3
   i. Compare D to 6
   j. Multiply 4 and 15
   k. X is greater than or equal to 12
   l. 15 is greater than 3

CHAPTER 2

Introduction to Program Flowcharts ■