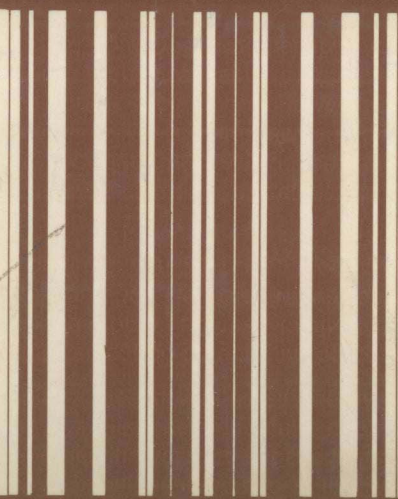


**CONCEPTUAL
PROGRAMMING
USING**

BASIC



**Allen Baker
Kathy Hamrick**

CONCEPTUAL PROGRAMMING USING BASIC

ALLEN BAKER
KATHY HAMRICK

Augusta College

PRENTICE-HALL, Englewood Cliffs, New Jersey 07632

Baker, Allen.

Conceptual programming using BASIC.

Includes index.

1. Basic (Computer program language) 2. Electronic digital computers—Programming. 3. Problem solving—Data processing. I. Hamrick, Kathy. II. Title. III. Title:

Conceptual programming using B.A.S.I.C.

QA76.73.B3B34 1984 001.64'24 83-9533

ISBN 0-13-166678-9

Editorial/production supervision and

interior design: *Service to Publishers, Inc.*

Cover design: *20/20 Services Inc. (Mark Berghash)*

Manufacturing buyer: *Gordon Osbourne*

© 1984 by Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

All rights reserved. No part of this book
may be reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-166678-9

Prentice-Hall International, Inc., *London*

Prentice-Hall of Australia Pty. Limited, *Sydney*

Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Prentice-Hall Canada Inc., *Toronto*

Prentice-Hall of India Private Limited, *New Delhi*

Prentice-Hall of Japan, Inc., *Tokyo*

Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*

Whitehall Books Limited, *Wellington, New Zealand*

**CONCEPTUAL
PROGRAMMING
USING BASIC**

*Dedicated to the memory of
Dr. Jerry Sue Townsend*

PREFACE

This text departs from the traditional approach used to teach BASIC. Its emphasis is on teaching the student a generalized approach to problem solving with BASIC as an added bonus rather than teaching BASIC with some problem-solving skills thrown in. We first teach the student to solve a problem by decomposing it into separately solvable components and examining the relationships each component has with the others. BASIC instructions are introduced after the student has mastered these problem-solving techniques, and even then, they are introduced simply as a means of translating the solution into a language the computer can understand.

The data flow diagram is the tool for problem decomposition. It is not a flowchart and should not be viewed as one. It is, rather, a pictorial representation of the disjoint components required to solve a problem and the relationship each component has with the others. Using this picture as a guide, the student can describe (using a pseudolanguage) the transformations that convert input data flows into the output data requirement of the problem. The pseudolanguage is mechanically translated into BASIC to produce the final BASIC program.

We have observed that students are most comfortable initially with simple problems that perform a single, nonrepetitive task. Beginning students appear to think in terms of single-line statement relationships. As the student gains experience, however, he or she appears to progress to another level where the thought process involves instruction blocks. These blocks may contain complex selection or repetitive processes.

Chapters 1 through 7 deal with nonrepetitive problems. Chapter 8 introduces repetition and procedurally related blocks of code. Beginning with Chapter 8, the student must begin thinking in terms of program modules and blocks of code. The instructor should provide encouragement during this period.

We have used minimal BASIC throughout the text, not because we like the standard, but rather because of system compatibility. You are encouraged, however, to expand the syntax to take advantage of your particular computer system. Expanded data names, IF/THEN/ELSE, and CASE statements (if your system supports these) should be substituted and would provide a welcome improvement.

Acknowledgments

There are many people who have directly or indirectly provided support to this project. We first wish to thank M. Edward Pettit for his thorough review of the manuscript and helpful comments. Second, we wish to thank Beth Bryan and Fred Maynard for their comments and their help and Bill Bompert for his support. We also wish to thank our families, Karen, Greg, Richard, and David, and Gayle, Beth, and Ginger, for putting up with us while we pondered over these words. Finally, we thank Jerry Sue Townsend, whose encouragement and support made this project possible.

ALLEN BAKER
KATHY HAMRICK

CONTENTS

Preface	xiii
1 DECOMPOSING THE PROBLEM	1
1.1 Problems and the Computer	2
1.2 Problem Decomposition	2
1.3 The Data Flow Diagram	8
1.4 A Final Problem	9
1.5 Exercises	10
2 DATA FLOW REQUIREMENTS	13
2.1 Decomposing the Output Data Flows	13
2.2 Concentrating on the Problem Structure	18
2.3 Problems in the Data Flow Diagram	19
2.4 Exercises	22
3 ARITHMETIC TRANSFORMATIONS	27
3.1 Data Flows and Variables	28
3.2 Numeric and Character Constants	29

3.3	The Arithmetic Transformations	29
3.4	Describing the Transformations	32
3.5	Summary	34
3.6	Exercises	35
4	CONDITIONAL TRANSFORMATIONS	36
4.1	Relational Expressions	37
4.1.1	Simple Relational Expressions	37
4.1.2	Compound Relational Expressions	38
4.2	Describing Conditional Transformations	42
4.2.1	The Logical IF Instruction	42
4.2.2	The CASE Instruction	44
4.3	Exercises	47
5	PROBLEM DYNAMICS	49
5.1	Background	49
5.2	Input and Output Operations	50
5.2.1	Fields and Records	50
5.2.2	READ and WRITE Pseudoinstructions	51
5.2.3	Input and Output in Data Flow Diagrams	52
5.3	Transformation Ordering	54
5.4	A Final Problem	60
5.5	Exercises	62
6	INTRODUCTION TO BASIC	64
6.1	The BASIC Statement	64
6.2	Variables and Constants in BASIC	65
6.3	The REMARK Statement	66
6.4	Translating the Assignment Instruction	67
6.5	Translating the Logical IF Instruction	68
6.6	Translating the CASE Instruction	72

6.7	Translating the READ and WRITE Instructions	74
6.7.1	Translating READ	74
6.7.2	Translating WRITE	75
6.7.3	The DATA and READ Statements	78
6.8	The END Statement	78
6.9	Exercises	79
7	THE COMPLETE PROGRAM	81
7.1	A Payroll Problem	82
7.2	A Grading Problem	89
7.3	A Minimum-Charge Problem	92
7.4	Exercises	97
8	REPETITIVE PROCESSES	99
8.1	Repetition and the Data Flow Diagram	99
8.2	Using Repetition	100
8.2.1	The Pseudolanguage Description of a Repetitive Process	101
8.2.2	Describing the Repetitive Transformation	103
8.3	Additional Examples	104
8.4	Translating the Repetitive Transformation into BASIC	109
8.5	Exercises	111
9	COUNTING, ACCUMULATION, AND SEARCHING	112
9.1	The Counting Process	112
9.2	Selective Execution of the Count	114
9.3	Multiple Counts in the Same Process Block	115
9.4	The Accumulation Process	117
9.5	Combining the Count and Accumulation Processes	119
9.6	The Searching Process	122
9.7	Combining the Processes	123
9.8	Exercises	125

10	REPETITION AND BASIC	127
10.1	Counting, Accumulation, and Searching Translations	127
10.2	BASIC Translation Using the FOR/NEXT Instruction	130
10.3	The Counter Method of Detecting the End of the File	133
10.4	Exercises	136
11	ARRAYS	138
11.1	The Need for the Set Variable—The Array	138
11.2	The Concept of the Array	139
11.3	Loading the Array	141
11.4	Manipulating Arrays	143
11.4.1	Searching an Array	143
11.4.2	Counting Values in an Array	144
11.4.3	Searching for the Minimum or Maximum Value in an Array	145
11.4.4	Accumulating the Elements within an Array	146
11.5	Printing the Elements of an Array	147
11.6	Translating the Array into BASIC	147
11.7	Exercises	151
12	SORTING	153
12.1	The Sorting Concept	153
12.2	Sorting Algorithms	154
12.2.1	The Selection Interchange Sort	154
12.2.2	The Adjacent Interchange Sort	157
12.3	Translating the Sort into BASIC	159
12.4	Exercises	161
13	FUNCTIONS AND SUBROUTINES	162
13.1	Mathematical Functions	162
13.2	User-Defined Functions	164

13.3	Subroutines	166
13.4	Exercises	171
14	STRING PROCESSING	172
14.1	Review of String Constants and Variables	172
14.2	String Comparisons	173
14.3	String Concatenation	174
14.4	String Functions	174
14.4.1	The Function VAL	175
14.4.2	The Function STR\$	175
14.4.3	The Function LEN	175
14.4.4	The Functions LEFT\$, RIGHT\$, and MID\$	175
14.5	Exercises	179
15	INTERACTIVE PROGRAMS: GAMES AND TUTORIALS	180
15.1	Main Program Design	181
15.2	The Guessing-Game Problem	183
15.3	A Tutorial Program	186
15.4	Exercises	188
Appendix 1:	PRIMITIVE TRANSFORMATIONS AND TRANSLATIONS	189
Appendix 2:	BASIC INSTRUCTIONS USED IN THIS TEXT	200
INDEX		203

1

DECOMPOSING THE PROBLEM

If you were asked to examine the computer system shown in Figure 1.1 and to identify the most important component, you would probably respond, “The computer itself.” Although the computer is an extremely important part of any automated system, it is not the most important. We can identify the important component by removing the computer from the picture. What remains? The problem to be solved.

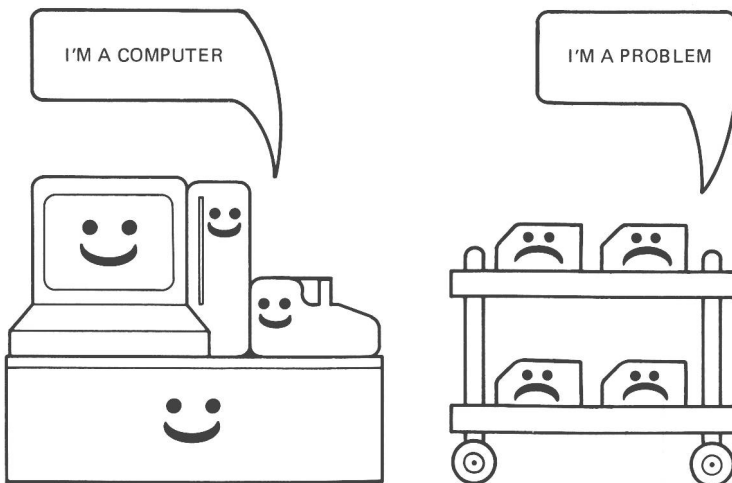


Figure 1.1 A COMPUTER SYSTEM

1.1 PROBLEMS AND THE COMPUTER

Not all problems can be solved by a computer. We can calculate utility bills, evaluate income tax returns, and even regulate the temperature in office buildings. We cannot, however, evaluate the quality of life, calculate the physical attraction of two people in love, or even regulate the economy of the United States. The characteristic separating the last three problems from the first three is that the first three can be precisely and thoroughly defined. If a solution to a problem can be defined precisely, then it can be implemented on a computer.

Since the problem and its solution are the most important components of any computer system, we should begin our study of computers by solving problems. Without properly developed problem-solving skills, the programmer cannot describe the problem and, therefore, cannot implement it on the computer.

1.2 PROBLEM DECOMPOSITION

A successful problem solver has learned that complex problems are only a composition of a certain number of simple problems. If he or she can decompose the complex problem into readily solvable simpler ones, the battle is won!

There are many ways to approach *problem decomposition*. You can divide the problem into components that are logically related, components that occur during the same period of time, components that form a control or procedural unit, or components that have other characteristics in common. The method you choose, however, has a tremendous effect on the complexity of your final solution.

The approach we will use to decompose a problem is based on the *flow of data* required to solve it. The data are traced from their input source, through appropriate transformations, to their output destination. This approach ignores the procedural aspect of the problem and concentrates on the flow of data.

The decomposition process starts with a statement of the whole problem to be solved. You must first determine what is known and what is required. We think of our known facts as *inputs* and our requirements as *outputs* in the problem-solving process. You can illustrate this idea graphically as shown in Figure 1.2.

The arrow into the *bubble* in Figure 1.2 represents the input data flow, and the arrow out of the bubble represents the output data flow. The bubble represents the *transformation* required to change the input into the output data flow. The graph is called a *data flow diagram*.

If the problem is a very simple one, you can describe the process

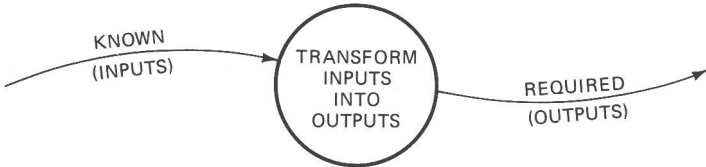


Figure 1.2 THE PROBLEM-SOLVING GRAPH

in a single bubble or transformation. Otherwise, you must decompose the problem into other, simpler transformations. In the remainder of this chapter, we examine problem decomposition using data flow diagrams.

Let us begin our study with a simple problem.

Problem 1.1

If Ed makes \$3.75 per hour and works for 30 hours, what is Ed's gross pay?

The first thing you want to do is to make sure that you understand the problem. Your understanding is often enhanced if you list the inputs in one column and the outputs in another column:

INPUTS		OUTPUTS	
\$3.75	ED'S RATE	?	ED'S GROSS PAY
30	ED'S HOURS WORKED		

The solution to the problem reduces to those processes or transformations that take the inputs and create the outputs. We can illustrate the solution to Problem 1.1 as shown in Figure 1.3.

Notice that bubble 1 in Figure 1.3 has not been defined. All we know is that if the problem can be solved (and in this case, it can), some transformation of the input items must occur to produce the output item. If we can describe that process, we can solve the problem, and we are done. Otherwise, we must decompose the process into simpler steps.

Let us look at the transformation as a kind of *black box*. The concept of a black box should be familiar to you even if you have never

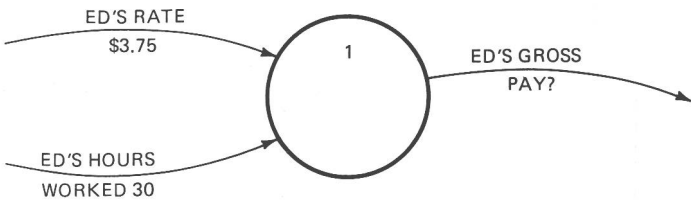


Figure 1.3 THE PROBLEM-SOLVING PROCESS

been exposed to the term. A television or an automobile can be considered a black box. You do not need to understand the workings of the inner components of either to use them. You simply input certain items (the turn of a knob or key) and experience certain outputs (a television picture or a running engine).

Using data flow diagrams as a decomposition tool allows you to defer examining the inside of the bubbles (black boxes) until later. In fact, we do not assign a name to the bubble until all input and output data flows have been established and the data flow names written above the arrows. We then choose a name that describes the transformation in terms of its input and output data flows. The name is formed by combining a strong action verb and a single object. From the input and output data flows shown in Figure 1.3, we could assign the name **CALCULATE ED'S GROSS PAY** to bubble 1.

The transformation for Problem 1.1 is a simple multiplication of rate times hours worked:

$$\$112.50 = \$3.75 \times 30$$

where \$112.50 represents the gross pay that Ed would receive. The final data flow diagram showing the result of the transformation is illustrated in Figure 1.4.

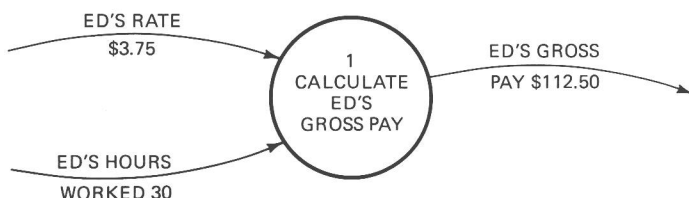


Figure 1.4 FINAL DATA FLOW DIAGRAM

The data flow diagram always represents a steady state of the problem. We are not concerned with iterations, controls, or simple error paths. We are concerned only with the flow of data as they move from the input source to the output. Looking inside the bubble will come later.

Let us consider a generalization of Problem 1.1. If we can work one case, we can work others.

Problem 1.2

If the hourly rate and hours worked are known, calculate the gross pay.

Again, let us list our inputs and outputs:

INPUTS	OUTPUTS
HOURLY RATE	GROSS PAY
HOURS WORKED	

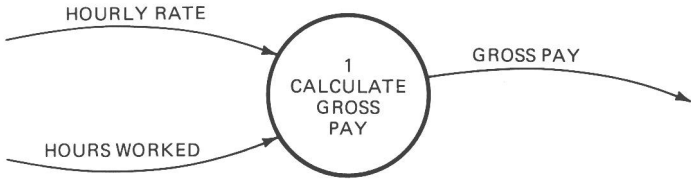


Figure 1.5 CALCULATE GROSS PAY

Our problem-solving process is to transform the inputs, HOURLY RATE and HOURS WORKED, into GROSS PAY. This process is illustrated in Figure 1.5. From our work with Problem 1.1, we know that GROSS PAY can be determined from HOURLY RATE and HOURS WORKED.

Suppose that we add a level of complexity to Problem 1.2 by requiring that net pay and gross pay be determined.

Problem 1.3

If a person’s hourly rate, hours worked, and deductions are known, determine his or her net pay and gross pay.

Let us make sure that we understand Problem 1.3 by working with a specific case. Assume that Fred works 40 hours and receives \$5.50 per hour. Assume further that Fred must pay \$50 in deductions. Listing our inputs and outputs, we have:

INPUTS		OUTPUTS	
\$5.50	FRED’S RATE	?	FRED’S NET PAY
\$50	FRED’S DEDUCTIONS	?	FRED’S GROSS PAY
40	FRED’S HOURS		

Again, our problem is to transform the inputs into the required outputs. Fred’s gross pay is $40 \times \$5.50$, or \$220. We must subtract his deductions of \$50 from \$220 to obtain his net pay of \$170.

We can diagram Problem 1.3 as illustrated in Figure 1.6.

The transformation, however, cannot be named using an action verb and a single object. Further decomposition is required. The output, NET PAY, can be determined from the inputs, DEDUCTIONS and

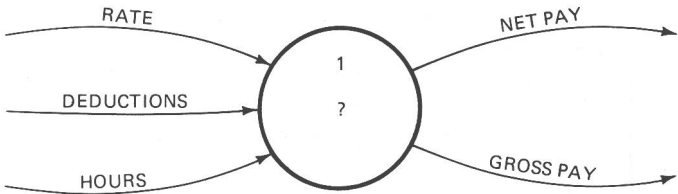


Figure 1.6 DIAGRAM OF PROBLEM 1.3