Programmable Pocket Calculators

Henry Mullish and Stephen Kochan

Novus Mathematician PR • Sinclair Scientific Programmable
Hewlett-Packard 25 • Hewlett-Packard 25C • Hewlett-Packard 55
Hewlett-Packard 65 • Hewlett-Packard 67 • Hewlett-Packard 19C
Hewlett-Packard 29C • Hewlett-Packard 33E

HAYDEN

Programmable Pocket Calculators

HENRY MULLISH and STEPHEN KOCHAN

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

Library of Congress Cataloging in Publication Data

Mullish, Henry.
Programmable pocket calculators.

SUMMARY: Examines in detail programmable pocket calculators, pointing out their architecture, special features, and programming techniques for the reader with no previous knowledge of programming.

1. Programmable calculators. [1. Programmable calculators. 2. Calculating machines] I. Kochan, Stephen, joint author. II. Title.
QA75.M79 001.64'2 80-11088
ISBN 0-8104-5175-1

Printed in the United States of America

2 3 4 5 6 7 8 9 PRINTING

81 82 83 84 85 86 87 88 YEAR

Programmable Pocket Calculators

PREFACE

A veritable calculator revolution erupted in 1971 when a million pocket calculators were sold in the United States alone. Overnight, the slide rule became obsolete and gave way to these electronic marvels which, in 1971, could do no more than add, subtract, multiply, and divide—albeit with phenomenal speed and accuracy.

Since then, the prices of pocket calculators have fallen from their initial $400 to the low price of $10, which some four-function calculators command today.

On the heels of the pocket calculator revolution now emerges yet another revolution—the programmable pocket calculator.

The origin of the programmable pocket calculator revolution dates back to December 1973 when Hewlett-Packard, a leading United States computer and calculator manufacturer, introduced to the world its stunning HP-65 "superstar" programmable calculator in pocket size. In addition to its programmability, the HP-65 provided the user with almost all the mathematical and scientific functions he could use. There were two reasons this particular model had such an impact on the calculator world. It made its mark in technological history by being the first pocket calculator produced that was *programmable.* Secondly, it permitted the user to record the program on a thin strip of metal-oxide coated plastic. Although the HP-65 opened up a completely new market, it was restricted by its somewhat prohibitive cost of $800, the price it commanded until the model was finally superseded in 1976 by its successor, the HP-67.

Apparently in an effort to capture an even greater share of the market, Hewlett-Packard introduced in December 1974 yet another programmable pocket calculator, the HP-55. Though not nearly so versatile as the HP-65, it was programmable and sold for half the price of the HP-65.

Inevitably, other leading manufacturers such as National Semiconductor entered the field of programmable pocket calculators and in January 1975 announced four different models: the 4515, the "Programmable Mathematician"; the 4524, the "Programmable Scientist"; the 1625, the "Programmable Financier"; and the 6035, the "Programmable Statistician." Although these four Novus brand models represented a somewhat lower level of sophistication than the Hewlett-Packard or Texas Instruments machines of their day, their price range of

around $100 made them extremely attractive for students and professionals in education, science, and business.

In June 1975, a British calculator manufacturer, Sinclair Radionics, introduced its 19-key, 24-step scientific programmable calculator, selling initially for $80.

Once again, in July 1975, Hewlett-Packard astounded the calculator world with its introduction of the HP-25, a programmable pocket calculator weighing a mere 6 ounces and selling initially for $195.

In July 1976, Hewlett-Packard announced a successor to its HP-65. The name given to their new 224-step card-programmable calculator was the HP-67. This feature-packed calculator was released with a price tag of $450. At this time Hewlett-Packard also announced a novel improvement to their highly popular HP-25. The model was called the HP-25C and it distinguished itself by being the first programmable calculator with a "continuous memory."

In December 1977, Hewlett-Packard introduced the HP-19C and the HP-29C. The HP-19C was the first programmable pocket calculator to become commercially available with a built-in thermal printer, which partially accounted for its retail price of $325. Its companion model, the HP-29C, similar in functions except for the printer, sold for $185.

At this point Hewlett-Packard changed gears slightly and made an attempt to capture the lower end of the pocket calculator market. In April 1978 they released their Series E calculators. The HP-33E, retailing for $100, and the HP-38E, retailing for $120, are the two programmable models of this series.

The purpose of this book is to examine in detail these programmable pocket calculators and to point out their architecture, special features, and programming techniques designed to maximize their use. At no time will it be assumed that the reader has any previous knowledge of programming, since to do so would put him at the mercy of the calculator manuals that all too often leave so much to be desired.

Today it is not unusual to find a wide range of assorted programmable and nonprogrammable calculators on sale. Making a sensible selection is difficult because of the bewildering variety of calculator features available. But this is only part of the problem—from the consumer's point of view. The salesman has his problems, too. He is expected to be conversant on all the various models, many of which differ from each other in subtle ways. How is he supposed to know the advantages and disadvantages of so many machines, particularly the programmable models, since he probably has never had the training or experience necessary to understand the principles of programmables, let alone to answer intelligently incisive questions on the subject?

It is an interesting commentary on our times that the state of the art has progressed so far and so rapidly in so short a period of time that many department store calculator counters now provide a free telephone con-

nection directly to the manufacturer, of whom technical questions may be asked.

This book has been written to assist both the consumer and the salesman. Every program for each calculator is incorporated in a schematic showing *precisely* how to enter the program and to put the calculator to work. In this way the salesman and consumer alike will be at liberty to key in any program step by step, watch it calculate and display the final results without having to get involved with the various programming philosophies, logic, or the particular calculator architecture.

HENRY MULLISH
STEPHEN KOCHAN

CONTENTS

Chapter One
THE ART OF PROGRAMMING **1**
Some Fundamental Programming Concepts *1*
Flowcharting a Mathematical Problem *5*
Debugging and Editing Programs *5*

Chapter Two
AN OVERVIEW OF PROGRAMMABLE POCKET CALCULATORS **9**
The Logic Used in Programmable Pocket Calculators *9*
Calculator Displays *11*
Programmable Pocket Calculator Features *13*

Chapter Three
THE ECONOMY-LEVEL PROGRAMMABLE POCKET CALCULATORS **19**
The Novus Programmable Calculators *19*
The Sinclair Scientific Programmable *29*

Chapter Four
THE HEWLETT-PACKARD PROGRAMMABLE POCKET CALCULATORS **39**
Features Common To All the Hewlett-Packard Models *39*

Chapter Five
THE HEWLETT-PACKARD 25 AND 25C **51**
The HP-25 *51*
The HP-25C *51*
Manual Operation of the HP-25 and HP-25C *53*
Programming the HP-25 and HP-25C *55*
Debugging and Editing Programs on the HP-25 and HP-25C *81*

Chapter Six
THE HP-55 PROGRAMMABLE SCIENTIFIC POCKET CALCULATOR **84**
Manual Operation of the HP-55 *85*
The Digital Timer *89*
Programming the HP-55 *93*
Debugging and Editing Programs on the HP-55 *106*

Chapter Seven
THE PROGRAMMABLE HEWLETT-PACKARD 65 **108**
Manual Operation of the HP-65 *108*
Programming the HP-65 *117*
Debugging and Editing Programs on the HP-65 *149*

Chapter Eight
THE HEWLETT-PACKARD 67 **151**
Manual Operation of the HP-67 *151*
Programming the HP-67 *160*
Debugging and Editing Programs on the HP-67 *200*

Chapter Nine
THE HEWLETT-PACKARD 19C AND 29C **202**
Programming the HP-19C/29C *203*
Debugging and Editing Programs on the HP-19C/29C *229*

Chapter Ten
THE HEWLETT-PACKARD 33E **231**
Programming the HP-33E *233*
Debugging and Editing Programs on the HP-33E *248*
PPC—Formerly the HP-65 Users Club *248*

Chapter Eleven
IMPLICATIONS OF THE PROGRAMMABLE POCKET CALCULATOR IN SCIENCE, INDUSTRY, AND EDUCATION **250**

INDEX **253**

CHAPTER ONE
THE ART OF PROGRAMMING

Computer programming is not only a well-paid profession demanding considerable expertise of the individual but is also one of the most satisfying and challenging of professions. The profession itself is a mere child in terms of its beginning, but since the 1950s when computers came on the scene, it has grown by leaps and bounds. Universities around the world now offer a plethora of courses on the subject, and it is becoming increasingly common for high schools to offer courses in computer languages such as FORTRAN, ALGOL, PL/I, BASIC, COBOL, and so on.

Why are computers so much in demand? The reason is that for the first time in our history we are able to solve problems with the speed of electricity. These electronic marvels have no inherent intelligence of their own. It is up to us to write appropriate instructions for them in order to arrive at the solutions. Such a sequence of instructions is called a *program* for the simple reason that the sequence is a planned one, exactly like a theatrical dramatic presentation follows its program in terms of acts and scenes.

Does one have to be a genius to be a programmer? Certainly not. Anybody with a modicum of intelligence and a slight sense of logic can program, without any previous training whatever. Of course, the greater one's sense of logic the easier it will be. It seems that certain people—particularly those who excel at puzzle solving and game playing—prove to be excellent programmers. What's more, such people tend to get "turned on" to programming almost with a passion. Programming is, like so many other things in life, improved by success. Once one has written a program, no matter how elementary it is, it seems to provide the kind of thrust to propel one to greater heights. The success feeds on itself. For some people, programming becomes a kind of addiction—happily one without any known negative effects, however.

Some Fundamental Programming Concepts

One is often faced with the problem of evaluating a complicated, perhaps lengthy mathematical expression using any one of the many available calculators. A sequence of keystrokes is then decided upon, and

each individual step is performed until the final solution is reached and displayed. Now, if there is a need to evaluate the same mathematical expression, one would have to physically repeat each one of the steps in the sequence as used to find the first solution. If this whole sequence of instructions must be repeated hundreds or even thousands of times, this would become a formidable chore. Ideally one would want some method by which the calculator could "remember" the sequence of instructions used in evaluating the expression the first time, permitting the user to reinitiate the sequence of instructions in the calculator's "memory" for each new set of data to be operated upon.

The basic feature of the modern programmable calculator provides precisely this ability: to store and execute a sequence of instructions known as a *program* in the calculator's memory and to have those stored instructions process as many data sets as are necessary.

An important variation of this sequential operation is the ability to automatically repeat a group of instructions when a particular part of the program is reached. This provides what is known as a *loop,* one of the fundamental properties of programming. This ability to alter normal

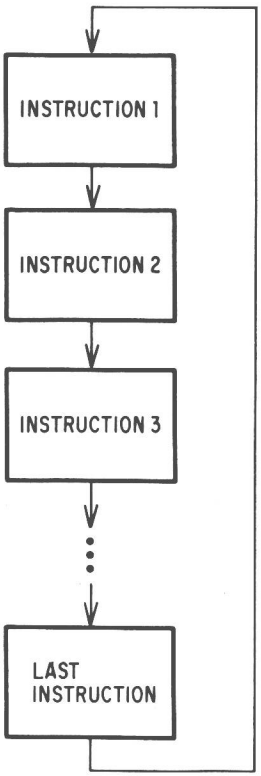

Fig. 1-1

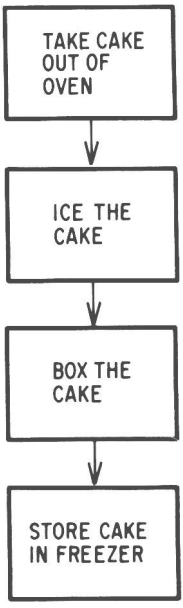

Fig. 1-2

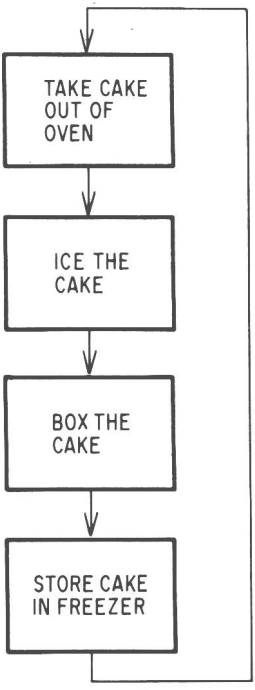

Fig. 1-3

sequential flow of a program is accomplished by what is referred to as an *unconditional jump,* as illustrated in Fig. 1-1.

Another important dimension is added to this sequential operation (Fig. 1-1) when a jump is made to another point in the program when a *special condition is met.* This is known as a *conditional jump* and provides the means for highly sophisticated decisions to be made within a program.

As an analogy to both these different types of situations, imagine that a baker apprentice has a job of taking cakes out of the oven. He must then ice the cake, place it in a box, and store it in the freezer. The individual steps may be represented simply, as shown in Fig. 1-2.

Naturally there are many cakes that are produced by a bakery, and our apprentice's job is to repeat the same sequence of operations for each cake.

This may be represented schematically, as shown in Fig. 1-3. Here we have the concept of the loop connecting the end of the sequence of operations with the beginning.

Suppose now that the apprentice ends his day at 5:00 P.M. and has no wish to work overtime. After storing each cake in the freezer, he might ask himself whether it is 5:00 P.M. yet. If it is, he puts on his coat and leaves work. Otherwise he takes out the next cake, completing the sequence once

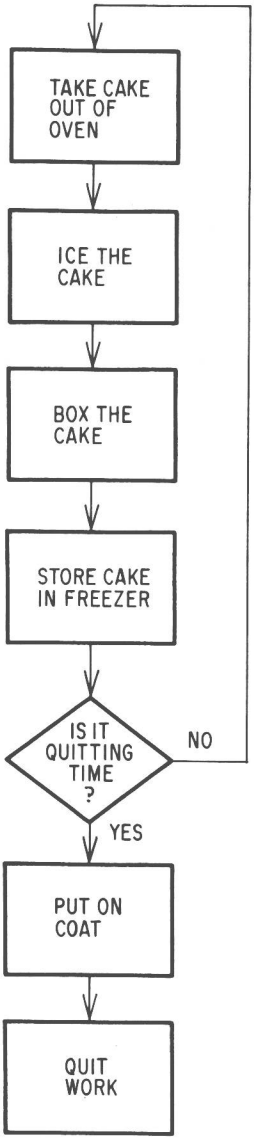

Fig. 1-4

again. It will be noticed that in this situation a decision is being made each time the process is executed; namely, is it quitting time or not? This may be represented schematically, as shown in Fig. 1-4.

This kind of schematic is usually referred to in programming as a *flowchart* and can be an extremely useful method of indicating the flow of control through a program.

Let us now amend our baker analogy so that we can illustrate further important principles of programming. Let us assume that in accordance

with union rules our apprentice finishes his workday after he has processed 200 cakes, regardless of what time of the day it happens to be. If he finishes at 3:00 P.M., then he leaves at 3:00 P.M., but if he does not finish until 7:00 P.M., then he works until 7:00 P.M. without further remuneration.

This new situation implies that a count of the number of cakes processed must be kept throughout the working day. At the beginning of the day, of course, this count is zero. As soon as a cake is stored in the freezer, he adds one to the count. Since we may assume that he is anxious to quit work as soon as possible, each time he adds a cake, he checks the count to see whether it has yet reached 200. If it has, he puts on his coat and quits work. If the count has not yet reached 200, he stays to process the next cake. The flowchart in Fig. 1-5 represents this new situation.

The concept of keeping a counter is of primary importance in programming. Here, the counter is the only means by which we know when to exit from the loop. In other words, a critical decision is being made based upon the value of the counter.

Notice that in Fig. 1-5 where the question, "Does count equal 200?" is asked (it is customary to write such questions within diamond-shaped "decision" boxes) if the answer is NO we do not go back to the very beginning, where the counter is set to zero, but rather to the following step.

Flowcharting a Mathematical Problem

Let us now take a simple problem in which a basic decision has to be made. We shall examine a series of integer numbers and determine how many of them are even and how many of them are odd.

By definition, an integer number is even if it is divisible by two with no remainder. If there is a remainder, that number is considered to be odd. In this particular problem there are two counters involved: one to keep a tally of the even numbers and the other to count the number of odd integers.

The input data to the flowchart in Fig. 1-6 are the individual numbers themselves. Each number is examined to determine if it is odd or even, and one is added to the appropriate counter. Once all the numbers have been examined, the contents of the odd counter and that of the even counter are displayed. This represents the output to the problem.

With few exceptions, the concepts of *input* and *output* are common to all programs.

Debugging and Editing Programs

When examining programs such as those included in this book, one should not be misled into thinking that they were written this way the first time. Few programs work the first time. Moreover, even if a program appears to be working the first time, the chances are that it will not work for

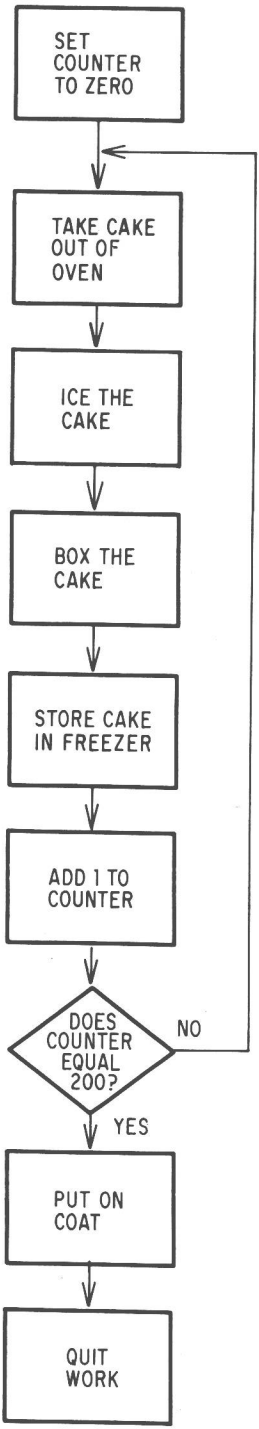
SET COUNTER TO ZERO
TAKE CAKE OUT OF OVEN
ICE THE CAKE
BOX THE CAKE
STORE CAKE IN FREEZER
ADD 1 TO COUNTER
DOES COUNTER EQUAL 200?
NO
YES
PUT ON COAT
QUIT WORK

Fig. 1-5

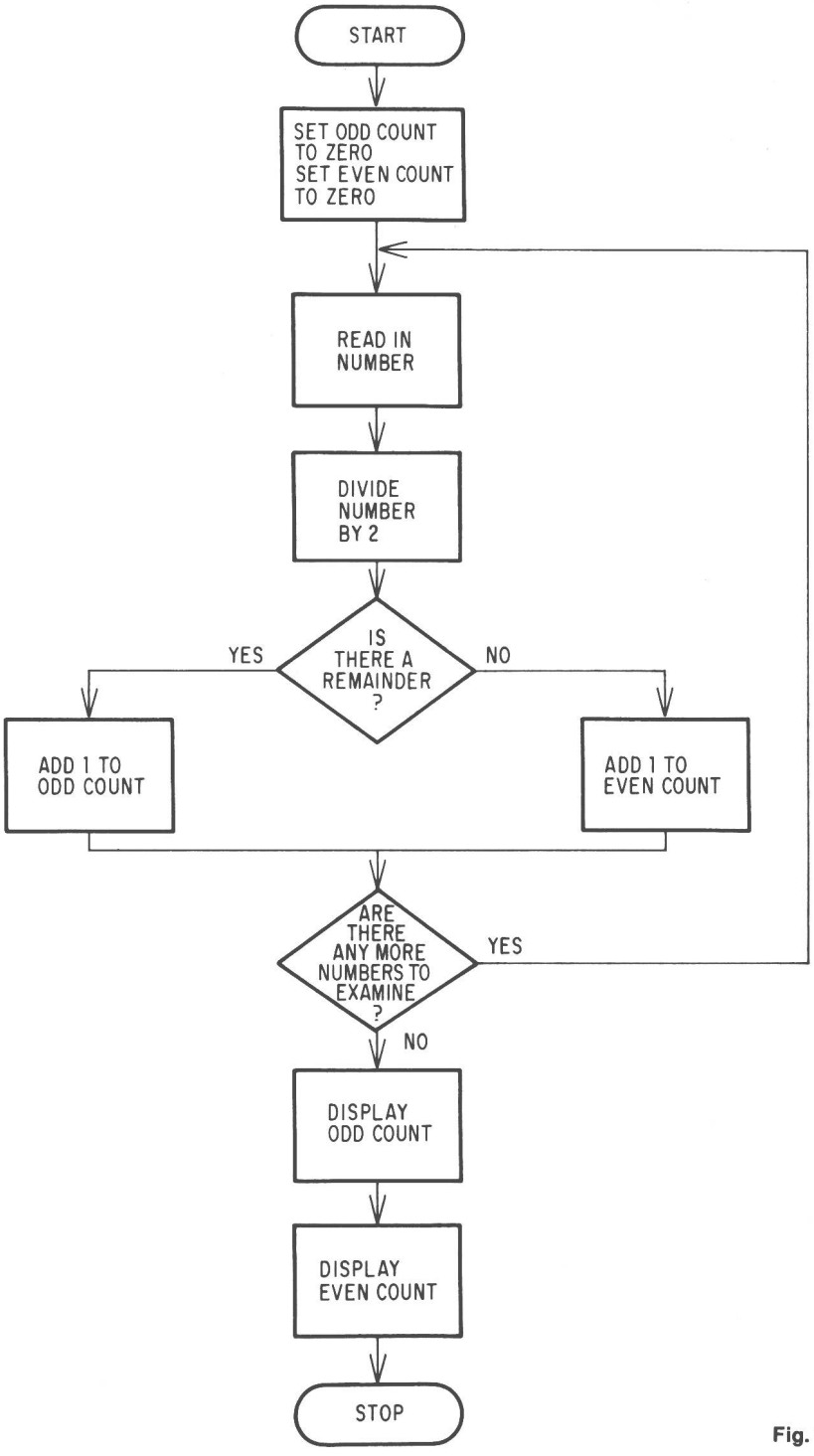
START
SET ODD COUNT TO ZERO
SET EVEN COUNT TO ZERO
READ IN NUMBER
DIVIDE NUMBER BY 2
IS THERE A REMAINDER ?
YES
NO
ADD 1 TO ODD COUNT
ADD 1 TO EVEN COUNT
ARE THERE ANY MORE NUMBERS TO EXAMINE ?
YES
NO
DISPLAY ODD COUNT
DISPLAY EVEN COUNT
STOP

Fig. 1-6

all possible data. Programming can be—and often is—quite a frustrating process. One needs to be proficient in the use of a particular machine and have a clear understanding of the steps to solve a given problem (algorithm) in order to arrive at the desired result.

To our knowledge the programmer is yet to be born who has not had to suffer the frustration of having diligently and carefully written a program that did not work the first time it was run. It seems to be a characteristic of programming that errors are made either in writing the program—that is, the program has an error of logic—or in entering the instructions into the computer. These errors are traditionally known as *bugs* and the finding and elimination of these bugs is known as *debugging*. Once having found the bug, correcting and modifying the program is generally spoken of as *editing*. In fact, the debugging and editing phase may take longer than the writing phase!

Should it be necessary to modify a program once it has been written—and this is almost always the case—it might be somewhat of a relief to the programmer to know that it is not always necessary to rekey in the program from the beginning, since much of the original program may be salvaged.

Before being convinced that a program is in perfect working condition, one should check it out using sample data and compare the output with known results, if this is at all possible. Naturally, if there is a conflict between these results something is wrong, and the program has to be suspect. The fault may lie in the incorrect keying in of the program, or there may be an error of logic.

In the former case, a careful comparison of the keyed in program against the original handwritten program will bring to light any inconsistencies. In the latter case, where a logical error is suspected, the following approaches are suggested:

1. Check the flowchart to insure that blocks are in logical sequence.
2. Compare the correspondence between the logic of the flowchart and the program itself.
3. Be sure that the instructions behave in the manner planned. This may mean going through the program on paper step-by-step, keeping track of the contents of each of the registers used by the program.
4. Make use of any additional debugging aids available on the particular calculator. This may include a *single step* key, which permits the user to proceed through the program one instruction at a time; a *pause* key, which halts the program temporarily to permit intermediate results to be viewed; or take advantage of any listed features that may be present.