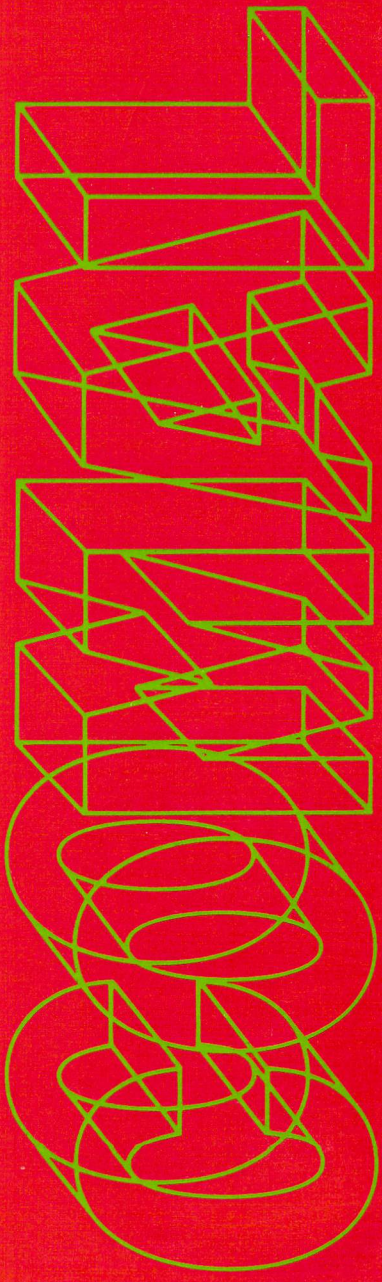
STARTING WITH COMAL

Ingvar Gratte



STARTING WITH COMAL

NOTE TO AUTHORS

Do you have a manuscript or a software program related to personal computers? Do you have an idea for developing such a project? If so, we would like to hear from you. Prentice-Hall produces a complete range of books and applications software for the personal computer market. We invite you to write to Giles Wright, Senior Editor, Prentice-Hall International (UK) Ltd, 66 Wood Lane End, Hemel Hempstead, Herts HP2 4RG, England.

STARTING WITH COMAL

Ingvar Gratte



ENGLEWOOD CLIFFS, NEW JERSEY LONDON MEXICO NEW DELHI
RIO DE JANEIRO SINGAPORE SYDNEY TOKYO TORONTO WELLINGTON

Library of Congress Cataloging in Publication Data

Gratte, Ingvar. (1943-)
Starting with COMAL.

Bibliography: p.
Includes index.
1. COMAL (Computer program language) I. Title.
QA76.73.C26G7 1985 001.64'24 83-9725
ISBN 0-13-843003-9 (pbk.)

British Library Cataloguing in Publication Data

Gratte, Ingvar
Starting with COMAL.
1. COMAL (Computer program language)
2. Microprocessors
I. Title
001.64'24 QA73.C/

ISBN 0-13-843003-9

© 1985 by Ingvar Gratte.

All rights reserved. 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 permission of Prentice-Hall International, (UK) Ltd. For permission within the United States contact Prentice Hall, Inc., Englewood Cliffs, New Jersey 07632.

Prentice-Hall, Inc., *Englewood Cliffs, New Jersey*
Prentice-Hall International, UK. Ltd., *London*
Prentice-Hall of Australia Pty., Ltd., *Sydney*
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*
Editora Prentice-Hall Do Brasil Ltda., *Rio de Janeiro*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Whitehall Books Limited, *Wellington, New Zealand*

Printed in Great Britain by A. Wheaton & Co. Ltd., Exeter

10 9 8 7 6 5 4 3 2 1

Contents

Preface *vii*

Chapter 1 **Introduction** **1**
1.1 Starting Programming 1

Chapter 2 **More Programming Ideas** **11**
2.1 Using INPUT and PRINT 11
2.2 Introduction to Strings 18
2.3 Mathematical Expressions, Parenthesis and Built-in Functions 19

Chapter 3 **Alternatives** **25**

Chapter 4 **Loops** **47**
4.1 FOR . . . NEXT 47
4.2 Other Constructions 54
4.3 Maximum and Minimum 60

Chapter 5 **Procedures** **65**
5.1 Introduction 65
5.2 Random Numbers 80

Chapter 6 **Arrays** **89**
6.1 Introduction 89
6.2 Sorting 100
6.3 Project: Mini-Market 105

Chapter 7 Using Sequential Files 113
7.1 Sequential Files 113
7.2 The Triangle Computer Club 117
7.3 Project: CHRS 126
7.4 Project: Simple Educational Pograms 128

Chapter 8 DATA Statements, Random Files and Functions 131
8.1 DATA Statements 131
8.2 Random Files 132
8.3 Functions 137
Conclusion 144

Solutions to Exercises and Problems 145

STANDARD COMAL List of Commands 198

Index 201

Preface

What makes computers incredibly useful is the simple fact that by using different programs, one computer can do many different things.

Programs are written in programming languages. There are a large number of such languages. This book uses one called COMAL-80, which was invented by the Danish teachers Benedicte Lofstedt and Børge Christensen.

COMAL-80 is a very structured version of BASIC, with many of the good qualities of advanced languages like Pascal and Ada. But at the same time COMAL is very simple to use – as simple as any ordinary BASIC. COMAL makes it possible to code quite complicated programs in a readable and concise way.

But perhaps its greatest advantage is that it can help to teach good programming habits. It makes the step to more complex languages, such as Pascal and Ada, natural and smooth.

Another advantage of COMAL-80 is that the variations between different versions are very small. (In this it is unlike BASIC. Different machines have very different versions of BASIC.)

In this book, where there are variations between machines, space has been left for "system notes". You are asked to fill them in with the correct information for your own particular system. This book is therefore independent of any particular implementation.

COMAL is the best introduction to programming for students of all ages. I hope you will enjoy learning it.

Ingvar Gratte

Stockholm
January 1985

1 Introduction

1.1 STARTING PROGRAMMING

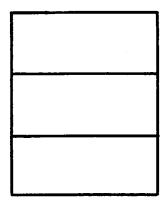
Here is a short COMAL program.

```
10 X:=3
20 Y:=4
30 Z:=X*Y
40 PRINT Z
```

If this program is in the internal memory of your computer and you type RUN (i.e. first type R, then U and then N) on the keyboard, and then press the key usually marked RETURN or ↵ [1] the number 12 will appear on your screen. When you type RUN you *execute* the program.

However, as the computer only understands 0s and 1s, why did anything happen at all? The answer is that there is another (complicated) program also in the computer's internal memory which translates the COMAL program, line by line, into sequences of 0s and 1s. This program is called the COMAL interpreter.

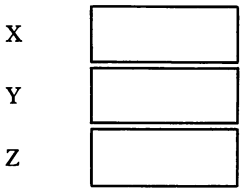
In the internal memory there are at least the following:



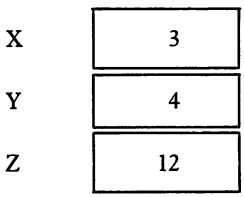
What are the data for this program? There are three *variables*, X, Y and Z.

1. The symbol ↵ is used from here onwards to mean "press the RETURN key".

A variable is something that can be assigned a value. Imagine that in the area called data you have three boxes, one for each variable, and that before the program is executed the boxes have no contents:



However, after the program is executed they have the following contents:




```
X:=3
```

is an example of an *assignment* and simply means, "put number 3 in the box marked X".

```
Z:=X*Y
```

is a slightly more complicated assignment which means the following: First compute the right-hand side, i.e. multiply the value of X by the value of Y, and then put the result in the box marked Z.

Exercise 1.1*[2]

If the current program is extended with

```
50 Z:=Z+X
60 PRINT Z
```

what will happen when this new program is executed?

Exercise 1.2*

What is the role of the PRINT statements (lines 40 and 60)? What would happen if they were deleted?

---

2. An asterisk after a question or exercise number means that there is a hint, or more or less complete solution, at the end of the book.

Exercise 1.3*

In a program line 100 reads

```
100 Q:=Q+A
```

Where just before line 100 the values of Q and A are given as:

Q	27
A	14

What are the values of Q and A after this line?

Before you enter the current program into the computer's memory, first make certain that there isn't a program already. The command

```
LIST
```

shows you all program lines (if any) in the memory. The command

```
NEW
```

takes away all program lines (if any) from the memory. When you have no program lines in the memory you are ready to start. (Note that you must press the RETURN (↵) key at the end of each line of typing, in order to enter the information into the computer: it will not respond to your commands unless you do so.)

Now that the program area is empty, type in AUTO. The system will provide you with line numbers and you may start typing (the underlined parts are typed by the user):

```
10 X:=3
20 Y:=4
30 Z:=X*Y
40 PRINT Z
50
```

When this is done and you want to leave AUTO, in some systems you just need to press the RETURN key again, while in others you must press ESC . . . . Check with your user-manual to find out how you do this on your machine.

SYSTEM-NOTE

To leave AUTO .....

.....

Now check that the program has been entered correctly by typing LIST, then execute the program by typing RUN.

To make changes in a program

There are two ways to change a program. One way is simply to type in the complete new line,

```
20 Y:=5
```

which will replace the old line 20.

The other way varies very much from system to system. The square on the screen which indicates where the next character on the screen will be placed is called *cursor*. In some systems the cursor can be moved up or down by pressing keys marked ↑ (up) and ↓ (down), and you can change the contents of a line (in this case line 20) only when it is on the screen. You can always make line 20 appear on the screen by typing

```
EDIT 20
```

(or, if the command EDIT doesn't exist for your system, LIST 20.)

If the cursor can only be moved left or right you must usually start by typing EDIT 20. The system will then place the cursor on that line. Suppose that you now have the cursor on the line

```
20 Y:=4
```

(In some systems EDIT places the cursor immediately to the right of the writing.) Use the keys marked → (to move the cursor one position right) and ← (to move the cursor one position left) in such a way that the cursor is "over" the figure 4. Then simply type 5 and press the return key. You can check that your change has been entered by typing LIST (or LIST 20 if you prefer to list only the current line). The modified program is ready to be executed.

Hints:
Learn how to make changes (as neatly as possible) in your programs.
Try to avoid using the first ("brute force") method to rewrite whole lines – in the long run this will save you a lot of time.

We will now extend the program with two lines. Type AUTO 50 and then continue just as before:[3]

3. In some systems AUTO automatically gives you the next free line, i.e. in this case line 50.

```
50 Z:=Z+X
60 PRINT Z
70
```

and leave AUTO.

Exercise 1.4

Check that you have typed each line correctly and execute the new program.

As you will soon appreciate, this is in many respects quite a bad program. For example, the first lines in a program should always start with name of the program, the version, the date and the author. You might also include a short description of what the program accomplishes. In this case (with the above program) one could type:

```
1 // DEMONSTRATION PROGRAM
2 // 20 AUG 1983 VERSION 1.0
3 // BY BILL COMPSON
```

and then list the program.

Version 1.0 means basic version 1, number 0. If you make minor changes to the program the next version will be called 1.1. If you make more fundamental changes, the next version will be called 2.0 and so on. The first version is always 1.0.

// means that the rest of the line is only a comment, meant for the reader of the program. // can make an entire line a comment, but it can also be placed to the right of a line to be included in a program.

Example

```
30 Z:=X*Y //MULTIPLY X BY Y
```

Comments should be used frequently in order to make programs *readable* both for the author and others. Comments do not affect the execution of programs – they are only for the convenience of the reader. When you execute the program with comments you will get exactly the same result as you obtained previously, when the program was run without comments.

List the program. Usually a program has line numbers 10, 20, 30 and so on. However, this one does not. To get it into that form type RENUM or RENUMBER (the question of which to use is system-dependent – sometimes both are acceptable). Then list the program, to see if this has worked.

Inserting Lines

A typical problem is that you have line numbers 10, 20, 30 etc., and you want to insert for instance five lines between lines 30 and 40. You can accomplish this by the command

```
AUTO 31,1
```

The system then generates the line numbers 31, 32, 33 and so on.

Exercise 1.5*

You should not insert more than nine line this way. Why?
Which line numbers would AUTO 70,3 generate?
Describe how to insert 43 lines between 40 and 50.

Deleting lines

Suppose you want to delete one or several lines using the command DEL.

Examples:
DEL 60 deletes line number 60
DEL 60-90 deletes lines 60-90 inclusive
DEL 60- deletes line 60 and all subsequent lines
DEL -90 deletes line 90 and all preceding lines

Note that typing:

```
50 ↵
```

does *not* delete line 50. Note also that in some systems a comma (,) is used instead of a dash (–), i.e.

```
DEL 60, 90
```

Exercise 1.6

Remove the last two lines in the current program.

Listing lines

To list the whole program you merely need to type LIST. But you can also list parts of a program: LIST 20 simply lists this line.

Exercise 1.7

What do you think
(a) LIST 20-70
(b) LIST -130
(c) LIST 30-
will accomplish?
Note that with LIST (just as with DEL) some systems use a comma (,) instead of a dash (–).

Storing and retrieving programs

Before you store a program on a disk the program must be named. How this is done depends on your system.

SYSTEM-NOTE

The rules for naming files are .....

.....

.....

.....

Common alternatives are:

DEMOPROG
"DEMOPROG"
"1:DEMOPROG"

There is usually a maximum length for the name of a file.

To save a program use the command SAVE (filename).[4] For example:

SAVE DEMOPROG
SAVE "DEMOPROG"
SAVE "O:DEMOPROG"

4. The third example, SAVE " O:DEMOPROG" should be used in Commodore systems. If the second way is used, serious problems can occur with the disk. This error is not trapped by COMAL.

Exercise 1.8

Save the current program, then check that it has been stored by using the command CAT or LOOKUP or DIR (depending on your system). These may be followed by some kind of identification.

SYSTEM-NOTE

To display the contents of disks make the following commands .....

.....

.....

Clear the internal memory by typing NEW, then retrieve the program you have just stored by using the command LOAD. Depending on your system you may need to type in one of the following:

```
LOAD DEMOPROG
LOAD "DEMOPROG"
LOAD "1:DEMOPROG"
```

Using the printer

If you want the program now in internal memory to be listed on paper, type one of the following:

```
SELECT OUTPUT PRINTER
SELECT OUTPUT "LP"
SELECT "LP"
```

or something similar, depending on your particular system.

Then type LIST. What happens? What happens if you type LIST once more? Where is the second output?

Now load the program. Type one of the commands above and then execute the program. What happens? What happens in a second execution? In some systems you automatically return to the screen. In others you must type SELECT OUTPUT "DS" or something similar.

These two commands may also be used inside programs.

SYSTEM-NOTE

To direct output to the printer use the command .....

.....

Return to the screen is automatic/not automatic. To return to the screen use the command .....

.....

Exercise 1.9

What will the following program do? Try to find out without actually running the program, then check your guess by executing the program.

```
 40 X:=3
 50 PRINT X
 60 SELECT "LP"
 70 PRINT X
 80 X:=X+1
 90 PRINT X
100 SELECT "DS"
110 X:=X—5
120 PRINT X
```

Note: In this book the Commodore commands SELECT "LP" and SELECT "DS" are used. Don't forget to make the small changes that may be necessary to make them suit your system.

QUESTIONS

1.1 Describe the role of the interpreter.

1.2 What is the effect of the command NEW?

1.3 Suppose there is no program in the internal memory. On a disk you believe you have a program named MAYBE.
(a) How can you check whether there is such a program?
(b) If there is such a program, describe how to list it on paper.

1.4 What is the point of using // in a program? What is the effect?

1.5

```
10 X:=7
20 X:=X+1
30 // STUPID PROGRAM
40 // X:=X/2
50 PRINT X
```

(a) What will this program accomplish?
(b) In the above program three lines are missing at the beginning. Suggest what they might look like and insert them (try to use the AUTO command in some form). Then amend the line numbers so that they are 10, 20 etc.
(c) How can you delete the line with the comment X:=X/2?
Do this and list the program.
(d) Describe how to change X:=X+1 to X:=X+X as smoothly as possible on your machine.

1.6* Given a part of a program:

```
140 A:=A+B+C
150 B:=A+B+C
160 C:=A+B+C
```

Just before line 140 you have the following contents:

A	47
B	−36
C	15

Describe the contents
(a) just after line 140;
(b) just after line 150;
(c) just after line 160.

1.7 You may check question 1.6 by inserting the three program lines below at suitable places. Use an AUTO command.

```
PRINT A
PRINT B
PRINT C
```

If you try to execute this program with 12 lines altogether, what will happen? You can cure this with three assignments in lines 100, 110 and 120. How? How can you modify the program to get the result on paper?

1.8 Insert the three lines you have forgotten, giving them line numbers 10, 20 etc. Now save the program, and check that it really has been saved.

2 More Programming Ideas

2.1 USING INPUT AND PRINT

One reason why computers are so useful is that by using a large number of different programs they can accomplish an incredible variety of tasks! Another reason is that good programs, when they are fed with different data, can do different things. In this respect all the programs used so far in this book have failed: they simply do the same thing, time after time.

The following program looks similar to the program at the start of Chapter 1, but is much more useful. (It's up to you to compose lines 10, 20 and 30.)

```
40 INPUT X
50 INPUT Y
60 Z:=X*Y
70 PRINT Z
```

If you execute this program the following happens (the underlined parts are typed by the user). Don't forget ↵.

```
RUN
?5
?7
35

RUN
?11
?3
33
```

But there are two large groups of people who might still be quite dissatisfied with this program. First, those who are to *use* it without reading the program. Second, those who must *read* your program, perhaps in order to modify it. The program can be made more "friendly" to users by inserting the following lines:

```
31 PRINT "THIS PROGRAM COMPUTES PRODUCT OF TWO NUMBERS"
32 PRINT "WHAT IS THE FIRST NUMBER"
41 PRINT "WHAT IS THE SECOND NUMBER"
69 PRINT "THE PRODUCT IS"
```

Insert these lines and renumber them, then execute the program. It could look like this:

```
RUN
THIS PROGRAM COMPUTES THE PRODUCT OF TWO NUMBERS
WHAT IS THE FIRST NUMBER
?5
WHAT IS THE SECOND NUMBER
?7
THE PRODUCT IS
35
```

It would be nicer to have the result printed on one line. In order to do this, change the line

```
PRINT "THE PRODUCT IS"
```

to

```
PRINT "THE PRODUCT IS",
```

and execute the program. Do not rewrite the whole line.

Exercise 2.1

The output from this is

```
THE PRODUCT IS35
```

Modify the PRINT statement to

```
PRINT "THE PRODUCT IS   ",
```

What is the modification? What is the effect of ending a PRINT statement with a comma?

Exercise 2.2

Modify the lines

```
PRINT "WHAT IS THE FIRST NUMBER"
```

and

```
PRINT "WHAT IS THE SECOND NUMBER"
```

so that the following happens:

```
RUN
THIS PROGRAM COMPUTES THE PRODUCT OF TWO NUMBERS
WHAT IS THE FIRST NUMBER?5
WHAT IS THE SECOND NUMBER?7
THE PRODUCT IS 35
```

The statement PRINT has the effect that the next PRINT statement starts at the beginning of the next line on the screen or printer.

Exercise 2.3

Without executing the following programs try to describe their output, i.e. what is printed and how.

(a)
```
10 A:=7
20 B:=5
30 PRINT A
40 PRINT
50 PRINT B
```

(b) change line 30 in (a) to 30 PRINT A,
(c) delete line 40 from the program in (a)
(d) delete line 40 from the program in (b).

Exercise 2.4

Modify the program in Exercise 2.2 so that the following result is obtained:

```
THIS PROGRAM COMPUTES THE PRODUCT OF TWO NUMBERS
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

WHAT IS THE FIRST NUMBER?5
WHAT IS THE SECOND NUMBER?7
THE PRODUCT IS 35
```

Instead of having two lines, for instance:

```
140 PRINT "THE PRODUCT IS ",
150 PRINT Z
```

it's possible to have one line,

```
140 PRINT "THE PRODUCT IS ", Z
```

Instead of having three lines, for instance:

```
240 PRINT A,
250 PRINT B,
260 PRINT C,
```

it's sufficient to use one line, for instance:

```
240 PRINT A,B,C,
```

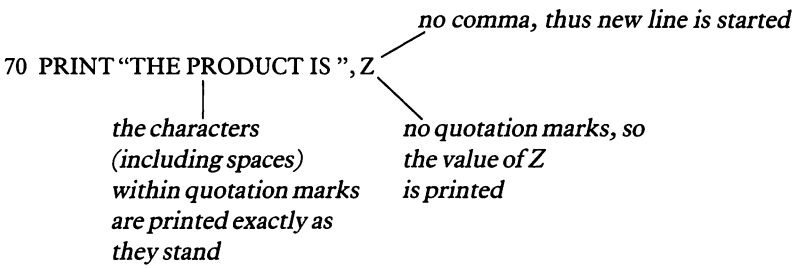
Exercise 2.5

What is the output of the very small program:

```
40 PRINT "BUFFALO"
50 PRINT "BILL"
```

What happens if you add a comma to line 40? Is the output correct then? Modify line 40 so that it becomes correct. Finally, make one PRINT statement of the two lines.

This section is finished with an example:



The TAB Function

On each line of the screen there is a definite number of positions (normally 80), and most often they are denoted 1, 2, 3 . . . 80.

However, sometimes there are only 40 positions. (Other values such as 64 do also exist.) Sometimes they are denoted starting with 0, 1, 2 . . . .

A PRINT statement

```
PRINT TAB(5), "JOHN"
```

will print JOHN with J in the position denoted 5; O in position 6; H in position 7; and N in position 8.

Exercise 2.6

How are the positions on the screen denoted on your system? Find out by using this small program:

```
10 PRINT "0123456789"
20 PRINT TAB(5),5
30 PRINT "123456789"
```

SYSTEM-NOTE

The number of positions on one line is ..... They are denoted .....

```
PRINT TAB(15), "!", TAB(30), "?"
```

will appear on the screen as:

!	?
position 15	position 30

The spaces on a line printed by a printer are also numbered in the same way, but the number of positions can be different from the number of positions on the screen.

Exercise 2.7

Try to imagine what will happen if you execute the line:

```
10 PRINT "BUFFALO", TAB(4), "BILL"
```

Then run it to see if you were right.

Readable Programs

One way to make programs more readable is to use names for variables which help to explain their use. The rules for forming variable names are system-dependent, but in any case the names may contain up to 16 letters or digits of which the first must be a letter.

Permitted: X6780MM SUMOFMANY SUM2
Not permitted: 2SUM 6780MM

To make the first program in this chapter more readable you could replace X with FIRST,Y with SECOND and Z with PRODUCT.

Exercise 2.8*

Start without any program lines and write a user-friendly, readable program that enters three numbers via INPUT and prints their sum.

Exercise 2.9

Write a program just like you have done for Exercise 2.8, but using four numbers instead of three.

Improving Stored Programs. Deleting Programs from Secondary Memory

On Page 8 (Exercise 1.8) you stored an old version of the demonstration program. Now load that program into the internal memory and replace X with FIRST,Y with SECOND and Z with PRODUCT in order to make it more readable.

Exercise 2.10

What happens if you try to save this program using the same file name as before? Will the new version replace the old version, or will nothing happen at all? To be absolutely certain that the new version gets stored you can start by deleting the old version from the disk, using the command DELETE.

Exercise 2.11

Try to delete the old version. Check that it really has been deleted.

Note: In some systems (for example Commodore) it is compulsory to mention the disk drive number when deleting a file, but not when saving a file.

SYSTEM NOTE

The rules for deleting files on secondary storage are .....

.....

.....

.....

A Few Words About INPUT

You are now used to this construction:

```
60 PRINT "WHICH NUMBER",
70 INPUT NUMBER
```

which appears on the screen as:

```
WHICH NUMBER? 57
```

(With this construction a question mark *always* occurs.) However, you could use just one line:

```
60 INPUT "WHICH NUMBER?":NUMBER
```

Note:
(a) that you must supply the question mark yourself;
(b) the use of the colon (:).

To make the following appear on the screen:

```
ENTER A NUMBER,PLEASE! 57
```

you can only use the second type of input:

```
60 INPUT "ENTER A NUMBER,PLEASE!":NUMBER
```

You can also enter two or several items via one INPUT statement, for example:

```
60 PRINT "WHAT ARE A AND B",
70 INPUT A,B
```

or

```
60 INPUT "WHAT ARE A AND B?":A,B
```

In both cases this appears on the screen as:

```
WHAT ARE A AND B? 67 83 (or 67,83)
```

Note that your answers must be separated by a space or by a comma.

2.2 INTRODUCTION TO STRINGS

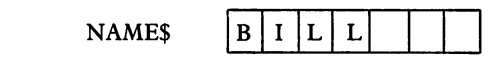
So far only one kind of variable has been used – those pointing at boxes containing numbers, for instance:



However you have also been working with strings – for instance:

```
"THE PRODUCT IS"
"BUFFALO"
"BILL "
```

To identify strings you use variables ending with a dollar sign ($)[1] for instance:



Such a box can contain at most a definite number of letters or digits – in this case 7. The actual number of characters is obtained using the key-word LEN. In the case above LEN(NAME$)=4 you decide the maximal number of characters it can contain by writing in your program, usually in the beginning.

```
DIM NAME$ OF 7
```

This is called a *declaration*.

Exercise 2.12

Investigate what happens if you execute the following program a few times (try to enter some long names).

```
10 DIM NAME$ OF 10
20 PRINT "WHICH NAME",
30 INPUT NAME$
40 PRINT
50 PRINT NAME$
```

How could you improve this program?

---

1. In some systems a "sun" (☉) is used instead of $.

Exercise 2.13

Execute the following program:

```
40 PRINT "MORNING,DAY,NIGHT OR EVENING",
50 INPUT ALTERNATIVE$
60 GREETING$:= "GOOD" + ALTERNATIVE$ + "!"
```

What happens? Now correct the program.

Exercise 2.14*

Write a program to print a membership card where you have to enter the member's name and membership number on the card, like this:

```
= = = = = = = = = = = = = = = = = = =

THE TRIANGLE COMPUTER CLUB

+ + + + + + + + + + + + + + + + + + +

MEMBERSHIP CARD FOR BILL COMPSON
1982     5 DOLLARS     NO 347

.....
(USER'S SIGNATURE)

= = = = = = = = = = = = = = = = = = =
```

Exercise 2.15

Check that the program in Exercise 2.14 is simple to amend for other years and other fees.

Exercise 2.16

How would you combine the strings "BUFFALO" and "BILL" into one string, "BUFFALO BILL"?

Exercise 2.17

What is the role of the DIM statement for strings? Give an example with your answer.

2.3 MATHEMATICAL EXPRESSIONS, PARENTHESES AND BUILT-IN FUNCTIONS

Expressions are read by the computer from left to right. Therefore 4/8 + 1 is equal to 1.5. But shouldn't 1 + 4/8 be 5/8 (i.e. 0.625)? No, because multiplication and division are "stronger" than (they have higher priority than)

addition and subtraction. However, multiplication and division have the same priority. Thus 12/4*3 = 9, but 12/(4*3) = 1, because numbers in parentheses are always computed first. Another example: 12/4+2 = 5, but 12/(4+2) = 2.

Exercise 2.18

What is (a) 12/4/3 (b) 12/(4/3) (c) (12/4)/3

Exercise 2.19

Should $\frac{1}{x+2}$ be written (a) as $1/x + 2$ or (b) $1/(x+2)$? Explain your answer.

Exercise 2.20

The average of the three numbers $x1$, $x2$ and $x3$ is $\frac{x1+x2+x3}{3}$ which is *not* the same as $x1+x2+x3/3$. Why? Correct this expression.

SQR

In COMAL the square root is written SQR. E.g.: SQR(9)=3; SQR(6.25)=2.5.

$\sqrt{x+2}$ is written SQR(x+2).
$\sqrt{2x-7}$ is written SQR(2*x−7)



(You must always use the multiplication sign when you multiply.)

There are three other functions in COMAL that you should know.

INT

INT gives the integer (or whole number) part of a number, for instance

INT(6.45)=6
INT(−6.45)=−7
INT(−3)=−3
INT(6)=6.

Thus the number I is an integer when INT(I)=I.
A number is a multiple of 4 when INT(I/ 4)*4=I.
The INT function can also be used instead of MOD (see page 42), and instead of DIV.

SGN

The function SGN simply tells you whether a number is positive, negative or zero. If the number is negative, this function will return -1; if the number is positive, it will return 1; and if the number is zero, it will return 0. I.e.:

SGN(456.34)=1
SGN(−98)=−1
SGN(0)=0

ABS

The ABS function returns the *value* of a number. This means that whether the number is positive or negative, it is returned as positive by this function. (The usual mathematical expression for this is ||.)

MOD

There is another expression, called MOD, which is used in this way:
A MOD B (assume that A and B are integers)
and gives the remainder after A is divided by B.

Examples: 37 MOD 8 = 5 Note the spaces around the word MOD.
1956 MOD 4 = 0

Thus A MOD B equals zero when A is divisible by B.

Exercise 2.21

Work out (without using your computer) what the following would be:

(a) 8 MOD 5
(b) 100 MOD 11

Then check your answers by typing

PRINT 8 MOD 5 ↵ and after that
PRINT 100 MOD 11 ↵

(No line numbers, you give direct commands.)

Exercise 2.22

Using INT instead of MOD, how could you decide whether a year is a leap year or not?

QUESTIONS

2.1 Give some reasons why computers are useful.

2.2 Mention two ways of making programs more readable.

2.3 You have now met two ways to get values into variables. What are they? Explain with examples.

2.4 What is the effect of a comma in PRINT statements? What is the effect of quotation marks in PRINT statements? Explain with examples.

2.5* Must the statement PRINT always produce a blank line? Why?

2.6 Explain what the TAB function does.

2.7* Write an easy-to-read program that makes the following appear on the screen:

```
WHICH NUMBER? 5
5*5 = 25
```

2.8* You could make the above program somewhat more "user-friendly". How?

2.9 Write a program (both readable and user-friendly) that prints the sum, difference and product of two arbitrary numbers entered via input.

2.10* Write a program that does the following on the screen:

```
NUMBER OF UNITS? 40
PRICE PER UNIT? 3
```

and then prints out on paper:

```
BILL
= = = =

NUMBER OF UNITS      PRICE PER UNIT
40                   3
TOTAL COST
120
```

2.11 Modify the last program so that the currency (pounds, dollars, crowns etc.) also appears – both in the question and in the bill.

2.12* Write a program that starts by printing a rectangle, and then computes the surface area of the rectangle, i.e.:

```
I----------I
I          I
I          I
I----------I

THIS PROGRAM COMPUTES THE SURFACE AREA OF A RECTANGLE.
THE SIDES ARE DENOTED A AND B.

HOW LONG IS A? 4
HOW LONG IS B? 5
THE AREA IS 20
```

2.13* In the program in Question 2.11 it is difficult to change currency. Why? Modify the program so that one of the first lines reads:

MONEYUNIT$:= "DOLLARS" or MONEYUNIT$:="POUNDS"

(or some other currency). Then use the variable MONEYUNIT$ in the rest of the program. Why is this version much simpler to change than the previous one?

2.14 Modify the rectangle program (Question 2.12) so that one of the first lines is:

LENGTHUNIT$:="INCHES" or LENGTHUNIT$:="METERS"

(or some other unit of length (in the plural form)). The program should pose the following questions:

```
HOW MANY METERS IS A?4.5
HOW MANY METERS IS B?6
THE SURFACE AREA IS 27 SQUARE METERS
```

2.15 Why is 1/3+4 different from 1/(3+4)?

2.16* Write as COMAL expression (|| is the mathematical expression for ABS)

(a) $\frac{1}{x+2}$

(b) $\frac{1}{2x-7}$

(c) $\sqrt{\frac{1}{3x+5}}$

(d)[2] $\left|\frac{3x-5}{5x-7}\right|$

(e)[2] $\sqrt{\frac{3x-5}{|2x+5|}}$

2.17 How can the INT function be used to check whether a number is a multiple of 5 or not?

2.18 If I=4637, what is

(a) I–INT(I/100)*100
(b) I-INT(I/1000)* 1000

Now answer the same questions, changing the value of I to −4637.

2. You don't need to solve (d) or (e) to be able to understand what follows. Leave them if they are too difficult.

3 Alternatives

The values you have entered via INPUT have not influenced *which* computations the computer should perform, but it is possible to make the computer do different computations for different numbers, or to avoid doing some things for some numbers. The program:

```
40 PRINT "WHICH NUMBER"
50 INPUT NUMBER
60 ROOT:=SQR(NUMBER)
70 PRINT "THE SQUARE ROOT OF ",NUMBER,"IS",ROOT
```

computes and prints the square roots of numbers entered with INPUT, but what happens if you try to enter a negative number? (Try this yourself.)

The previous program works only if the number is greater than or equal to 0. Change the program to:

```
40 PRINT "WHICH NUMBER",
50 INPUT NUMBER
60 IF NUMBER>=0 THEN
70 ROOT:=SQR(NUMBER)
80 PRINT "THE SQUARE ROOT OF ",NUMBER,"IS",ROOT
90 ENDIF
```

Lines 60-90 mean that if the number is greater than or equal to 0 then all the lines between 60 and 90 (ENDIF) are executed. If that is not the case (which means that the number is negative), nothing happens. If you list the program you should get the following:

```
40 PRINT "WHICH NUMBER",
50 INPUT NUMBER
60 IF NUMBER>=0 THEN
70    ROOT:=SQR(NUMBER)
80    PRINT "THE SQUARE ROOT OF ",NUMBER,"IS",ROOT
90 ENDIF
```

The lines between lines 60 and 90 (the line with the IF . . . THEN statement and the line saying ENDIF), are usually indented two spaces (depending on your system). This indentation shows quite well how more complicated programs are built up. The COMAL system provides the indentation.

If you input a negative number it would be nice to get some kind of warning, for instance:

```
YOU CANNOT CALCULATE THE SQUARE ROOT
OF A NEGATIVE NUMBER. TRY AGAIN!
```

To include this in the last program type AUTO 90 and then:

```
 90 ELSE
100 PRINT "YOU CANNOT CALCULATE THE SQUARE ROOT"
110 PRINT "OF A NEGATIVE NUMBER. TRY AGAIN!"
120 ENDIF
```

Then leave AUTO and list the program to see where the indentation is now. Execute this program a few times with different values to check that it works.

These two cases can be described, somewhat formally, by the following:

```
IF condition THEN
    xxxxxxxxxxxxxxxxxxx   }
    xxxxxxxxxxxxxxxxxxx   } an arbitrary number of lines
ENDIF

IF condition THEN
    xxxxxxxxxxxxxxxxxxx   }
    xxxxxxxxxxxxxxxxxxx   } an arbitrary number of lines
ELSE
    xxxxxxxxxxxxxxxxxxx   }
    xxxxxxxxxxxxxxxxxxx   } an arbitrary number of lines
ENDIF
```

Conditions

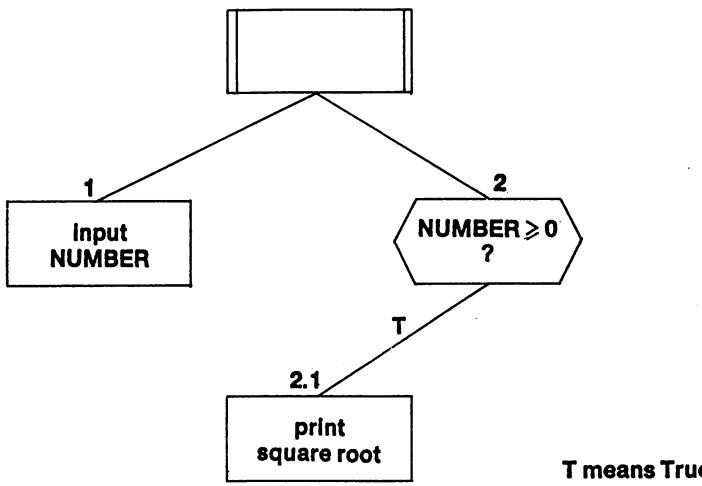
What is meant by a *condition*? Simply something that is either true or false.

Examples:

SUM > X	SUM is greater than X
SUM > = Z7 − 5	SUM is greater than or equal to Z7 − 5
NUMBER < SUM2/5	NUMBER is less than SUM2/5
FIRST < = SECOND + THIRD	FIRST is less than or equal to SECOND + THIRD

X< >2	X is not equal to 2
ALPHA = BETA	ALPHA is equal to BETA

You can use the six expressions > >= < <= <> = to form conditions.



(*Above:* structure diagram for the program at the bottom of page 25. *Below:* structure diagram for the program obtained in the middle of page 26.)

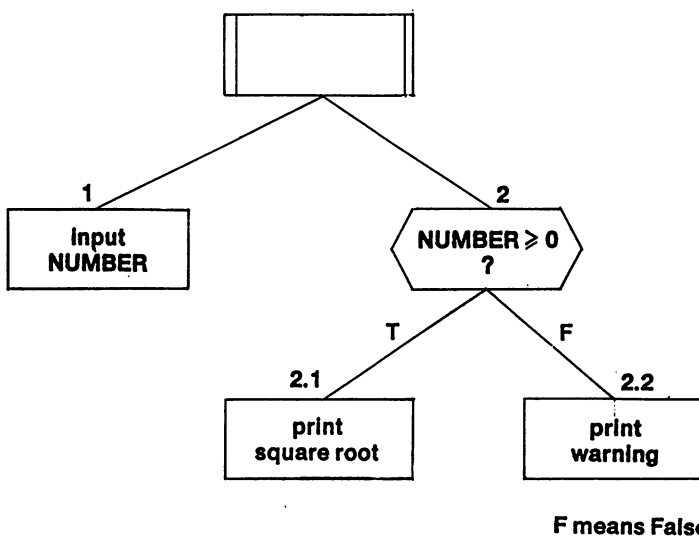
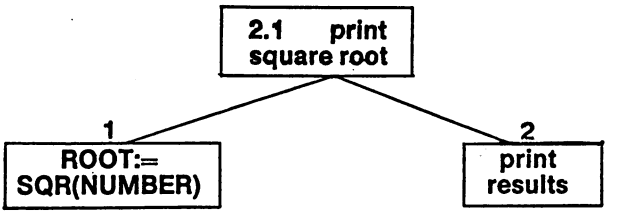


Figure 3.1

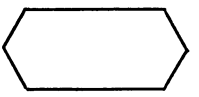
Figure 3.1 shows a way of illustrating programs. The two figures are called structure diagrams. They will be used frequently from now on. Note that lower-case letters are used inside boxes for things which could be further explained, while capitals are used for parts of the code. The letters T and F next to the lines stand for true and false. Note also the manner of numbering boxes. Box 2.1 can be further explained (or "broken down") in a separate structure diagram in this way:



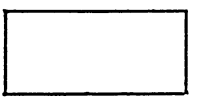
The correct numbers for the boxes in the figure above should actually be 2.1.1 and 2.1.2, since they are subdivisions of 2.1.

It is a matter of personal choice how far you break down a problem, but you should at least eliminate all uncertainties.

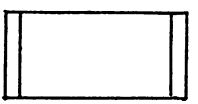
Symbols for use in structure diagrams



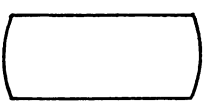
The symbol above is used to describe alternatives, while the symbol below is used elsewhere.



For the uppermost box in the structure diagram you should use the following symbol.



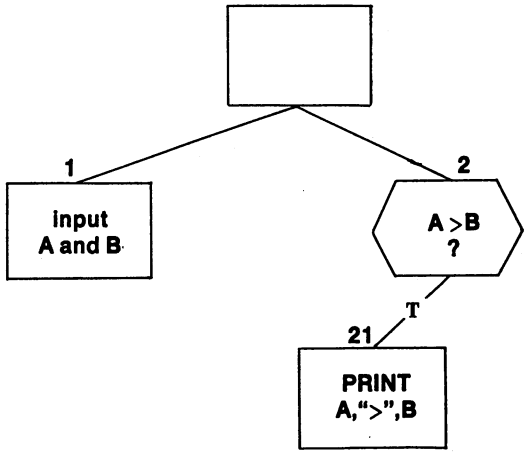
In the next chapter you will make acquaintance with the symbol



which is used to describe loops.

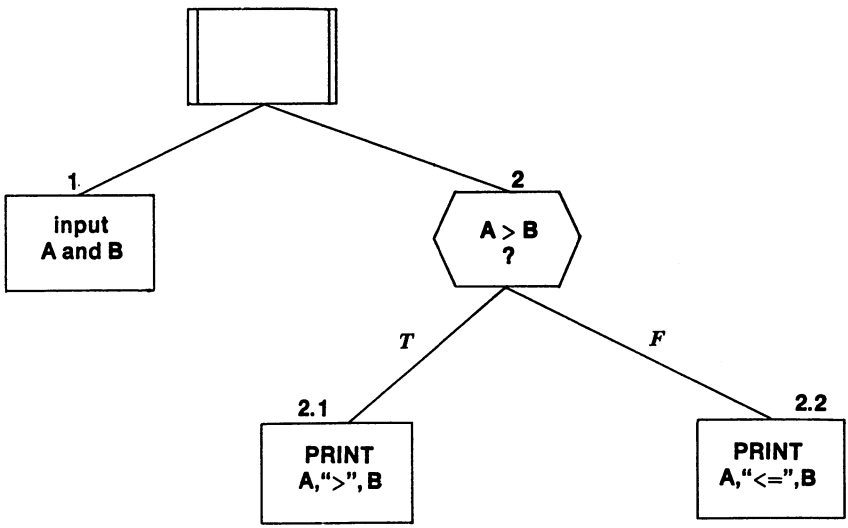
Exercise 3.1

Code the small program that is described by the following structure diagram:

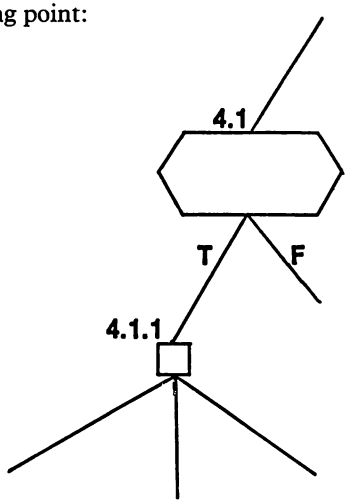


Exercise 3.2

Now code this program:



Sometimes several boxes are needed under an alternative-box, so that you can get a figure like this, where the small square box is simply serving as a branching point:



Exercise 3.3*

Write a program that enters two numbers via INPUT and prints their product and quotient, for instance this way:

```
THIS PROGRAM COMPUTES THE PRODUCT A*B AND QUOTIENT
A/B OF TWO NUMBERS A AND B ENTERED VIA INPUT.
WHAT IS A?3.5
WHAT IS B?2
THE PRODUCT OF 3.5 AND 2 IS 7
THE QUOTIENT OF 3.5/2 IS 1.75
```

Draw structure diagrams to describe the differences between the following three cases:

(a) If the value 0 is entered for B the only answer will be
THE PRODUCT OF 3 AND 0 IS 0

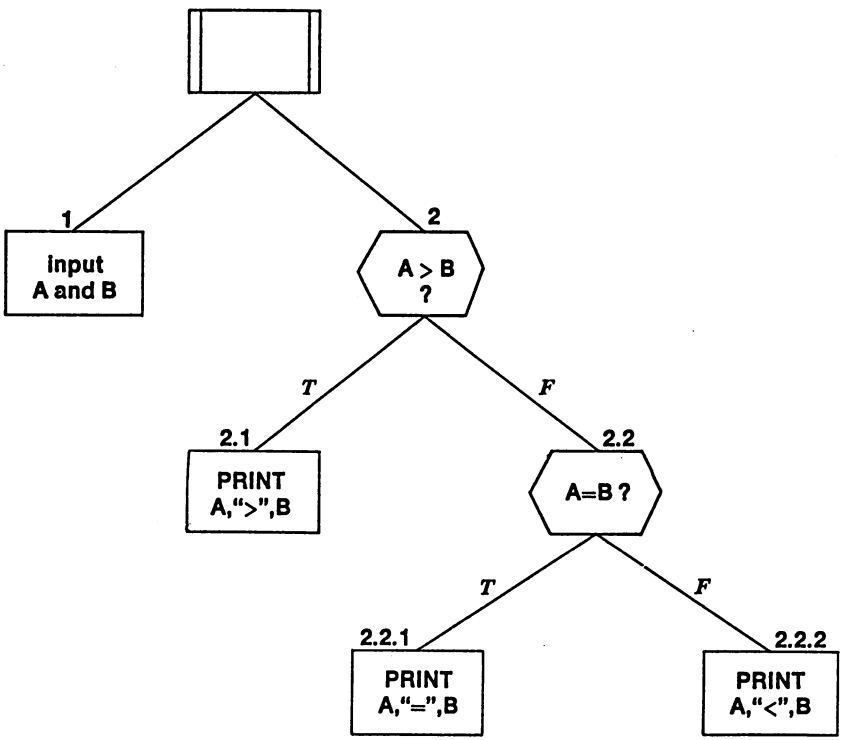
(b) If the value 0 is entered for B the answer will be
THE PRODUCT OF 3 AND 0 IS 0
YOU CANNOT DIVIDE BY ZERO.

(c) If the value 0 is entered for B the answer will be
THE PRODUCT OF ZERO IS ALWAYS ZERO!
YOU CANNOT DIVIDE BY ZERO.

When you have two numbers, A and B, you have three cases: A>B; A=B; and A<B. To find out which of these applies you can use a construction with ELIF (short for ELSE IF):

```
 40 INPUT "WHAT ARE A AND B?":A,B
 50 IF A>B THEN
 60   PRINT A,">",B
 70 ELIF A=B THEN
 80   PRINT A,"=",B
 90 ELSE
100   PRINT A,"<",B
110 ENDIF
```

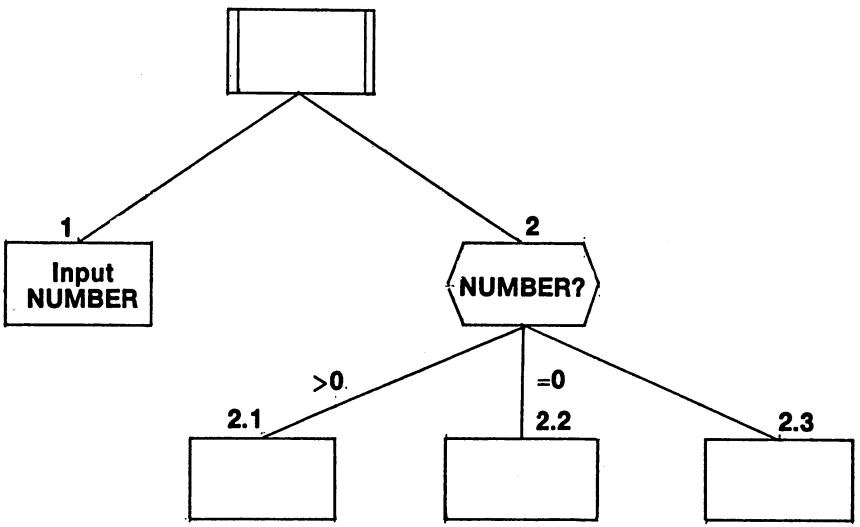
Remember that you don't need to consider the indentation when entering a program: the system does that for you. In this case you have two choices and the structure diagram looks like this where alternative-box 2 corresponds to line 50, and alternative-box 2.2 corresponds to line 70:



If you have a single number and you want to find out whether it is greater than zero, less than zero, or zero, you can draw a similar structure diagram and use similar code.

Exercise 3.4*

Draw such a structure diagram and construct the code. In this case you can use a simpler structure diagram:



Remember to read the alternatives from left to right.

Exercise 3.5

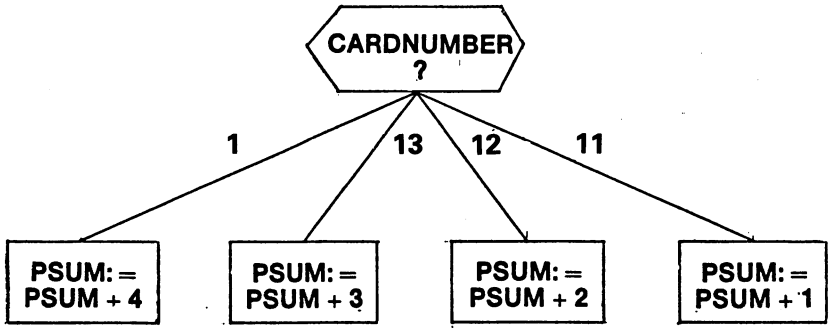
Complete the structure diagram above.

Exercise 3.6*

In most ball-games the winning team gets two points or, in the case of a draw both teams get one point each. Draw a structure diagram to get the following on the screen:

```
GOALS SCORED
HOME TEAM?3
AWAY TEAM?2
RESULT: HOME TEAM 2 POINTS
        AWAY TEAM 0 POINTS
```

In a set of playing cards each card has a color plus a number. For instance, knaves have the number 11. In Bridge you count one point for a knave (number = 11), two for a queen (number = 12), three for a king (number = 13) and four for an ace (number = 1). The structure diagram opposite describes how to add the number of points for one card to the sum PSUM:



The code becomes:

```
100 IF CARDNUMBER = 1 THEN
110    PSUM:=PSUM+4
120 ELIF CARDNUMBER = 13 THEN
130    PSUM:=PSUM+3
140 ELIF CARDNUMBER = 12 THEN
150    PSUM:=PSUM+2
160 ELIF CARDNUMBER = 11 THEN
170    PSUM:=PSUM+1
180 ENDIF
```

In general, you have

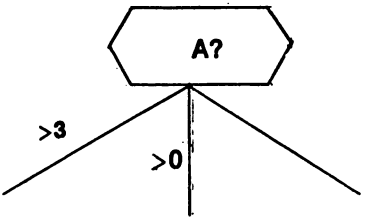
```
IF condition THEN
    xxxxxxxxxxxxxxxxxx
    xxxxxxxxxxxxxxxxxx        } an arbitrary number of lines
ELIF condition THEN
    xxxxxxxxxxxxxxxxxx
    xxxxxxxxxxxxxxxxxx        } an arbitrary number of lines
ELIF condition THEN
    xxxxxxxxxxxxxxxxxx
    xxxxxxxxxxxxxxxxxx        } an arbitrary number of lines
ELSE
    xxxxxxxxxxxxxxxxxx
    xxxxxxxxxxxxxxxxxx        } an arbitrary number of lines
ENDIF
```

In this example there are two ELIF statements, but, of course, the number of ELIFs is entirely dependent on the content of the program. Sometimes you have one ELSE and sometimes no ELSE whatsoever.

Exercise 3.7

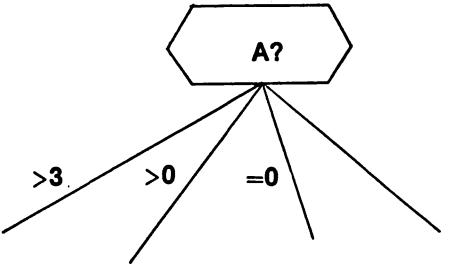
```
 40 INPUT "WHAT IS A?":A
 50 IF A>3 THEN
 60    PRINT A,">3"
 70 ELIF A>0 THEN
 80    PRINT A,"IS SMALL BUT POSITIVE"
 90 ELSE
100    PRINT A,"IS < = 0"
110 ENDIF
```

The program above is obtained from a structure diagram, part of which is given below. Complete the diagram.



Exercise 3.8*

What program is obtained if this part of the structure diagram looks like this:



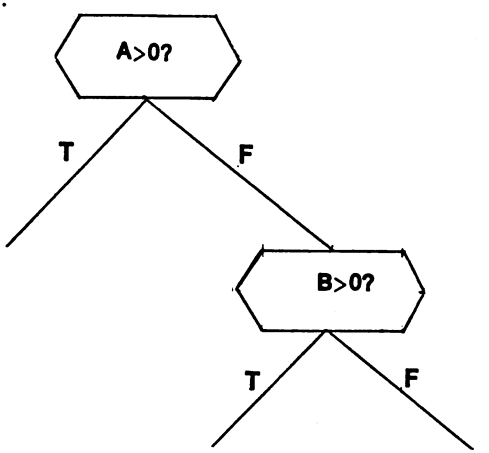
Exercise 3.9

Complete the following program with suitable PRINT statements on lines 70, 90 and 110.

```
 40 INPUT "WHAT IS A?":A
 50 INPUT "WHAT IS B?":B
 60 IF A>0 THEN
 70
 80 ELIF B>0 THEN
 90
100 ELSE
110
120 ENDIF
```

Exercise 3.10

Part of the structure diagram for the program in Exercise 3.9 is given below. Complete it.



Exercise 3.11*

Enter two numbers A and B with INPUT. Construct a program that finds out which of the following six cases is correct:

A>B>0 B>A>0 A>0>B B>0>A 0>A>B 0>B>A

(It is assumed that neither A nor B is equal to 0 and that A≠B.) Start by drawing a structure diagram. The screen could look like this:

```
WHAT IS A? -3
WHAT IS B? 2
RESULT:2>0>-3
```

Hint: If A>B then start by exchanging their values, i.e.

```
50 IF A>B THEN
60    G:=A
70    A:=B
80    B:=G
90 ENDIF
```

Then you have only three cases to consider.

Exercise 3.12

Why doesn't the following exchange the values of A and B?

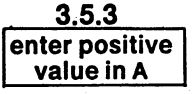
```
50 A:=B
60 B:=A
```

Exercise 3.13*

Much programming effort is spent on checking that values in a program are reasonable, for instance that a value for the sidelength of a square is positive. Write a part of a program that accomplishes the following:

```
.
.
.
WHAT IS A? -5
!!! WARNING - A MUST BE POSITIVE
!!! TRY AGAIN
WHAT IS A? 3
.
.
.
```

Assume that in a structure diagram this corresponds to



Now draw the rest of the structure diagram below box 3.5.3. Explain why this isn't a fool-proof way to get a positive value for A!

Exercise 3.14*

Modify the program on page 22 to allow for the following:

(a) A general price reduction of 10% if the number sold is greater than or equal to 100.

(b) A general price reduction of 20% if the number sold is greater than or equal to 400, or 10% if the number sold is greater than or equal to 100.

Exercise 3.15

Modify the program in Exercise 3.14 (b) so that a check ensures that NOFUNITS is positive, then make a structure diagram for this program.

Exercise 3.16

Enter the member's name, membership number and age via INPUT, and PRINT on paper, a membership card similar to that in Exercise 2.16. If the member's age is less than or equal to 18 the person is a junior member and there should be a line on the card like this:

```
1982    2 DOLLARS    NO 347 (JUNIOR)
```

If the person is older than 18 years but not older than 40 he/she is considered

a senior with a membership fee of $5. Otherwise the person is considered a supporter with a fee of $4.

Exercise 3.17

Using the same membership card program (see Exercise 3.16), why not enter the categories (for instance J, SE or SU) via INPUT? (Make no mention of age!) Write a program that accomplishes this.

Exercise 3.18

Write a program that accomplishes the following:

```
WHAT TIME OF DAY IS IT NOW?
IS IT MORNING, DAY, EVENING OR NIGHT? MORNING
GOOD MORNING!
HAVE YOU SLEPT WELL? I HAVE!
```

If you answer DAY the output should be:

```
GOOD DAY!
WHAT WOULD YOU LIKE FOR LUNCH? A BOILED EGG?
```

And something similar for the other alternatives.
If, by chance, you answered AFTERNOON the computer should answer:

```
AFTERNOON? I DO NOT UNDERSTAND AFTERNOON!
```

Exercise 3.19

Computers are very often used to produce advertisements with a "personal touch". The company NEW IDEAS ABOUT SOAP has the following information about a group of people: forename, surname, age, sex, marital status. Write a program to produce the following kind of letter, using INPUT to enter the information.

DEAR **MRS COMPSON**
HAVE YOU TRIED OUR NEW, INCREDIBLE COMPUTER-SOAP? **IT WILL MAKE YOU LOOK MUCH YOUNGER THAN YOU ARE!** THIS IS A SPECIAL OFFER FOR YOU, **MRS COMPSON**. JUST BRING THIS PERSONAL LETTER TO YOUR STORE AND YOU WILL GET 20 PER CENT REDUCTION IN THE PRICE! IN EVERY SOAP STORE!

YOURS SINCERELY
BOB BRIGHT
DIRECTOR, NEW IDEAS ABOUT SOAP LTD

It should be possible to alter the parts written in bold type according to the circumstances of the recipient of the letter, for example: a girl aged 18 needs better motivation than to be told that the soap makes her look much younger; and why not use her forename? What about a more informal ending?

The CASE Construction

A case construction is used where the answer to be input (which may be a string variable or a numeric variable) could be one of a number of things.

This very useful construction is best illustrated with an example:

Example:

```
.
.
.
210 CASE MONTH$ OF
220 WHEN"JAN","MAR","MAY","JUL","AUG","OCT","DEC"
230   NOFDAYSINMONTH:=31
240 WHEN"APR","JUN","SEP","NOV"
250   NOFDAYSINMONTH:=30
260 WHEN"FEB"
270   IF YEAR MOD 4 = 0 THEN
280     NOFDAYSINMONTH:=29
290   ELSE
300     NOFDAYSINMONTH:=28
310   ENDIF
320 ENDCASE
```

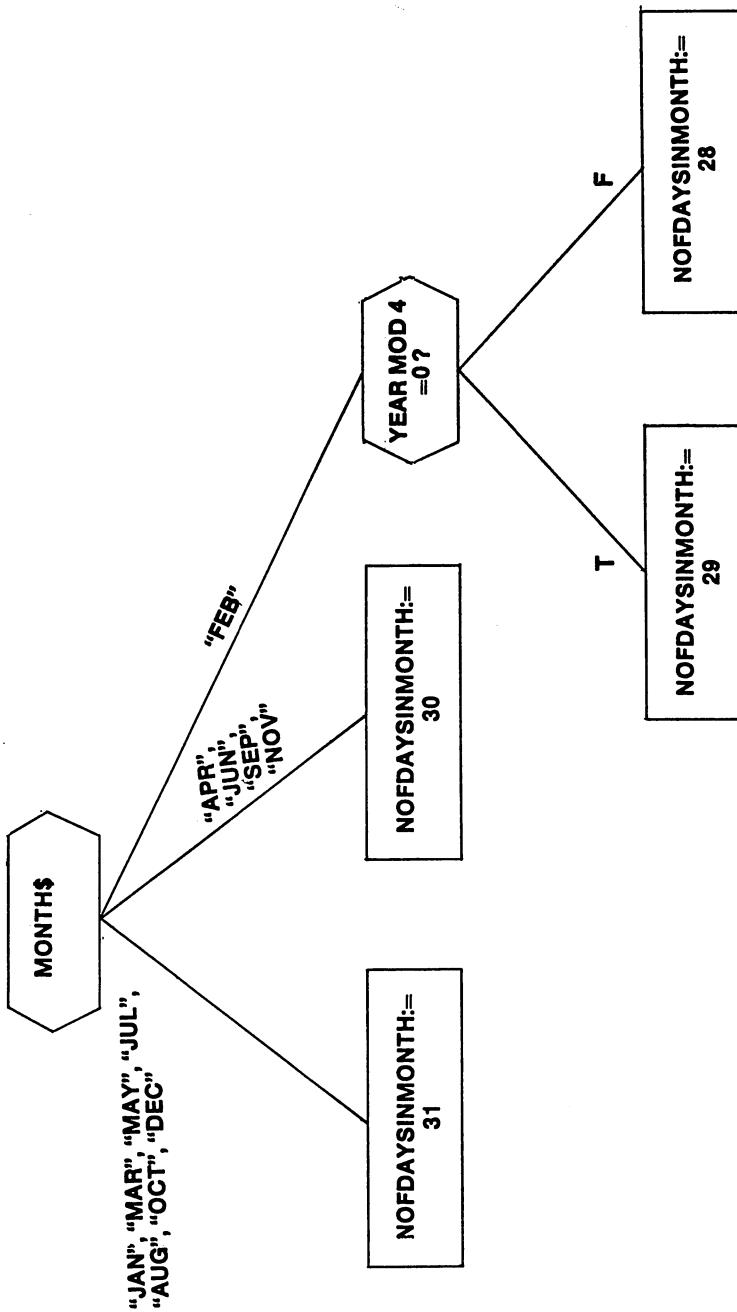
The structure diagram that describes this program-segment looks like this:

YEAR is the actual year, for instance 1982.
YEAR MOD 4 gives the remainder when YEAR is divided by 4.

Example:
1982 MOD 4 = 2
1984 MOD 4 =0

therefore 1984 is a leap year.

MONTH$ is declared with three characters.


MONTH$
"JAN", "MAR", "MAY", "JUL",
"AUG", "OCT", "DEC"
"APR",
"JUN",
"SEP",
"NOV"
"FEB"
NOFDAYSINMONTH:=
31
NOFDAYSINMONTH:=
30
YEAR MOD 4
=0?
T
NOFDAYSINMONTH:=
29
F
NOFDAYSINMONTH:=
28

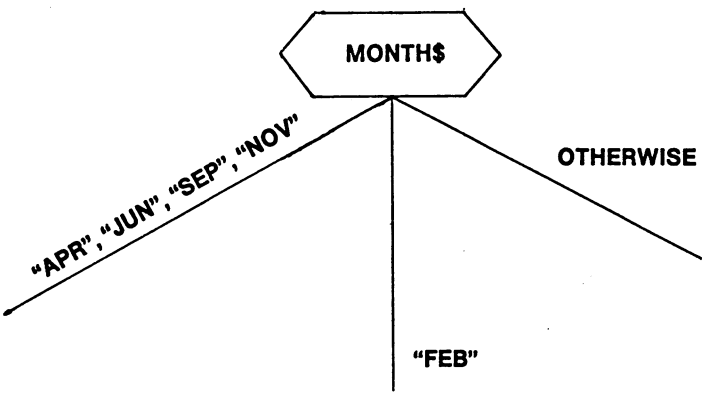
Exercise 3.20

Investigate what happens if MONTH$="BUH".

If MONTH$ really denotes the first three letters in an existing month the following program has the same effect (note that lines 250-290 are the same as for the previous program):

```
210 CASE MONTH$ OF
220 WHEN "APR","JUN","SEP","NOV"
230   NOFDAYSINMONTH:=30
240 WHEN "FEB"
250   IF YEAR MOD 4=0 THEN
260     NOFDAYSINMONTH:=29
270   ELSE
280     NOFDAYSINMONTH:=28
290   ENDIF
300 OTHERWISE
310   NOFDAYSINMONTH:=31
320 ENDCASE
```

The structure diagram now looks like this:



Exercise 3.21

Complete the structure diagram above.

Exercise 3.22

What happens in this case if MONTH$="BUH"?

Exercise 3.23

Modify the introductory CASE construction (lines 210-320 on page 38), so that if for instance MONTH$="BUH" a warning is printed on the screen, for instance,

ERROR! NO MONTH STARTS WITH BUH!

Exercise 3.24*

Try to rewrite the programs in Exercises 3.17 and 3.18 using CASE.

Instead of using MONTH$, i.e. a string as a selector of alternatives, one might use numbers.

Example:

```
210 CASE MONTHNR OF
220 WHEN 4,6,9,11
```

and so on.

Exercise 3.25

Complete the program segment (lines 210 and 220) above.

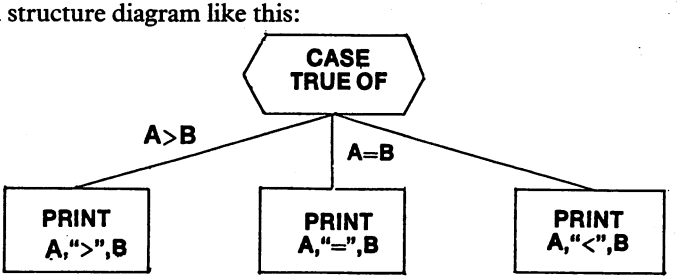
More About CASE

The CASE construction is so general that you can do without ever using IF . . . THEN constructions. The question of which to use in a particular case is a matter of convenience, taste and habit. (Don't forget the result of Exercise 3.20!)

Example:
The code on pages 30-31 could instead be formulated:

```
 40 INPUT "WHAT ARE A AND B?":A,B
 50 CASE TRUE OF
 60 WHEN A>B
 70   PRINT A,">",B
 80 WHEN A=B
 90   PRINT A,"=",B
100 OTHERWISE
110   PRINT A,"<",B
120 ENDCASE
```

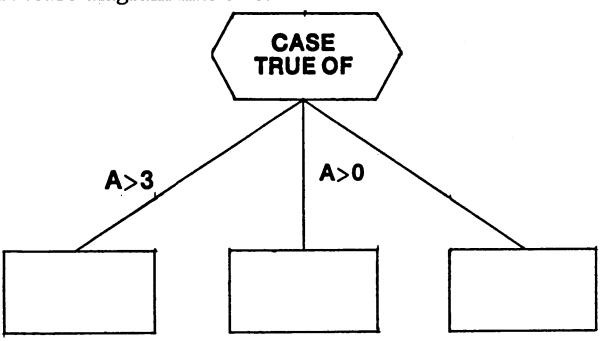
with a structure diagram like this:



The code of Exercise 3.7 can be formulated with CASE thus:

```
 40 INPUT "WHAT IS A?":A
 50 CASE TRUE OF
 60 WHEN A>3
 70   PRINT A,">3"
 80 WHEN A>0
 90   PRINT A,"IS SMALL BUT POSITIVE"
100 OTHERWISE
110   PRINT A,"IS<=0"
120 ENDCASE
```

with a structure diagram like this:



In fact you do not *need* to use IF, but if you only have IF . . . THEN . . . or IF . . . THEN . . . ELSE . . . (plus ENDIF), it is usually pointless to use CASE. Each time you need to use one or several ELIFs, try to find out if it would be better to use CASE instead. Go through Exercises 3.1–3.19 to find out when it would be better to use CASE instead.

More About MOD[1]

A MOD B gives the remainder after A is divided by B. In one sense, MOD is unnecessary, because you can always use INT instead:

$$\frac{73}{23} = 3 + \frac{4}{23}$$

Therefore

```
73 MOD 23 = 4
INT (73/23) = 3
```

But

```
73 = 23*3+4
```

or

```
4 = 73 - 23*3
```

1. It is not necessary to understand this section to continue reading this book.

With 73 replaced by A and 23 by B you will obtain

A MOD B = A − B*INT(A/B)

Another expression, which can be used when working with integers, is DIV, where simply:

A DIV B = INT(A/B)

Usually MOD and DIV are only used with positive integers.

Exercise 3.26

As you know, X is an integer precisely when INT(X)=X.
Use this to write a program segment that does the following:

```
WHICH INTEGER?4.7
4.7 IS NOT AN INTEGER
TRY AGAIN!
WHICH INTEGER?5
```

Exercise 3.27*

Write a program which enters two integers via INPUT, then checks that they really are integers, and after that finds out if the first divides into the second, or the second divides into the first, or neither of these alternatives. *Warning:* You must provide for the case where some or all of the numbers are equal to 0. Start by drawing structure diagrams.

QUESTIONS

> *Note:* All programs except the very simplest ones, must be constructed by first drawing structure diagrams and, after that, coding. If your structure diagrams have been in reasonable detail, the coding will be much easier.

3.1 Give examples to show how structure diagrams look when using the following:

IF . . . THEN . . .
IF . . . THEN . . . ELIF . . . ELSE . . .
CASE.

3.2 Try to describe what a condition looks like. Use some illustrative examples.

3.3 Customers of the Great Shop are entitled to different discounts.
Category "ORD" – ordinary customers – get 10% discount if the value of their purchases is $100 or more.
Category "EMP" – employees – get 10%, and 20% when the value of their purchases is greater than or equal to $50, and 30% when it is greater than or equal to $100.
Category "FIR" – firms – get 25% discount when the value of their purchases is greater than or equal to $1,000.
Write a program (using, among other things, CASE) that computes what the customer actually must pay (start by drawing structure diagrams).

3.4 How does the output of the following program depend on the value of A?

```
IF A<=0 THEN
  PRINT "A SHOULD BE POSITIVE"
ELIF A<>INT(A) THEN
  PRINT . . . . . .
ELSE
  CASE A OF
  WHEN 1
    PRINT "ONE"
  WHEN 2
    PRINT "TWO"
  WHEN 3
    PRINT "THREE"
  OTHERWISE
    PRINT "TOO BIG!"
  ENDCASE
ENDIF
```

Fill in the missing line and draw a structure diagram.

3.5 When should one use lower-case letters, as opposed to capitals, in structure diagrams?

3.6* Write a program that starts by drawing a right-angled triangle

```
*
**
* *
****
```

and then asks:

WHAT IS KNOWN?
(1) ONE SHORTER SIDE AND THE LONGEST ONE
(2) THE TWO SHORTER ONES

If something other than 1 or 2 is answered the computer should answer:

!!! NO SUCH ALTERNATIVE

In case 1 the answer could look like:

THE UNKNOWN SIDE (I.E. THE OTHER SHORT SIDE)
IS 4.76 METERS LONG

and similarly in case 2.

3.7* Construct a program that computes and prints out the next day's date. For instance:

YEAR? 1982
MONTH? 9
DAY? 23

Answer in the form
(a) 24/9/1982
(b) 24 SEPTEMBER 1982
Hint: When DAY is >27 you need to watch out for problems.

3.8 Construct a program which gives, for instance, $5789.23, prints:

+++FIVETHOUSANDSEVENHUNDRED AND EIGHTYNINE DOLLARS, 25 CENTS

Note that the number of cents is rounded to the nearest multiple of 5. Sums greater than or equal to $10,000 are rejected.

Note: Questions 3.7 and 3.8 are difficult.

4 Loops

4.1 FOR . . . NEXT

As you may already know, a computer can produce thousands of similar bills, i.e. it can repeat essentially the same operation a large number of times. In COMAL you can make the computer execute some lines again and again, in a few different ways.

Use AUTO and type the following lines:

```
10 FOR I:=1 TO 4 DO
20 PRINT "I=",I
30 NEXT I
```

and then leave AUTO and list the program.[1]

Exercise 4.1

Run the program above and describe what it does. What will happen if you change line 10 to

```
FOR I:=5 TO 11 DO
```

Insert the following lines:

```
25 SQUARE:=I*I
26 PRINT "I*I=",SQUARE
```

and renumber, list and execute the program. Describe what happens.

---

1. In some versions (notably later Commodores) ENDFOR is used instead of NEXT. If, when working with such versions, you happen to type NEXT it doesn't matter because the system will automatically replace it with ENDFOR.

Example:

```
FOR I:=4 TO 10 DO
  xxxxxxxxxxxxxx
  xxxxxxxxxxxxxx
NEXT I
```

} an arbitrary number of lines

The computer carries out the program, taking a different value for I, from 4 to 10, each time.

Zone

There is a variable, called ZONE, which belongs to the system.[2] Usually its value is 0. If its value is 15, the line:

```
PRINT "I AM","A","BOY"
```

gives the output

```
I AM            A               BOY

position
1               16              31
```

If the value of ZONE is 0 you'll get the output

```
I AMABOY
```

SYSTEM NOTE

The system variable ZONE is assigned a value in the following way: .....

.....

---

2. In at least the earlier versions of Metanic COMAL TAB:=12 has the same effect as, for instance, ZONE 12.

Exercise 4.2

```
10 // ZONE PROGRAM
20 // 5 NOVEMBER 1982 VERSION 2.0
30 // BY BILL COMPSON
40 //
50 FOR I:=1 TO 10 DO
60   PRINT "ZONE=",I
70   ZONE I
80   PRINT "BUFFALO","BILL","WAS","OLD"
90   PRINT
100  ZONE 0
110  FOR J:=1 TO 40 DO
120    PRINT "="
130  NEXT J
140  PRINT
150  PRINT
160 NEXT I
```

Figure 4.1

Before executing the program above on your computer try to run it "by hand", and write the output yourself. Then execute it to check your work.

Exercise 4.3*

Write a program that prints tables of squares and cubes of integers. The following should appear, first on the screen:

```
THIS PROGRAM PRODUCES TABLES OF THE SQUARES AND
CUBES OF INTEGERS
FROM WHICH VALUE? 1
TO WHICH VALUE? 5
```

Then, on paper, a table like this should be printed:

```
***TABLES OF SQUARES AND CUBES***
**********************************
INTEGER      SQUARE       CUBE
=======      =======      ====
      1            1          1
      2            4          8
      3            9         27
      4           16         64
      5           25        125
```

You should have done a structure diagram for this program, but if you haven't already done so, try drawing it now.

Sometimes it is beneficial to have a program which will work through the values in reverse order.

Exercise 4.4

Try to figure out what this small program does:

```
10 FOR STARTYPE:=10 TO 3 STEP -1 DO
20    PRINT STARTYPE
30 NEXT STARTYPE
```

STEP is most commonly used as being equal to +1, but it can have any positive or negative value, integer or non-integer. However, you may find that you get some strange results if you do use non-integer step lengths.

Exercise 4.5

Execute this small program. Which values are printed?

```
10 FOR STARTYPE:=3 TO 10 STEP 3 DO
20    PRINT STARTYPE
30 NEXT STARTYPE
```

Supposing you wanted the computer to add four numbers $x1$, $x2$, $x3$ and $x4$ and call their sum SUM, the computer can do this by the calculation:

SUM:=$X1$+$X2$+$X3$+$X4$

but it can also be done in this way:

First, the sum is 0	:	SUM:=0
then add the first number	:	SUM:= SUM+$X1$
then add the second number	:	SUM:=SUM+$X2$
then add the third number	:	SUM:= SUM+$X3$
and finally the fourth number	:	SUM:=SUM+$X4$

Note that essentially the same calculation is repeated four times. The following program should add four numbers entered via INPUT:

```
.
.
.
40 SUM:=0
50 FOR I:=1 TO 4 DO
60    PRINT "WHICH NUMBER",
70    INPUT NUMBER
80    SUM:=SUM+NUMBER
90 NEXT I
```

Now write line 100 and run the program. An illustration of how this program works is given in Fig. 4.2.

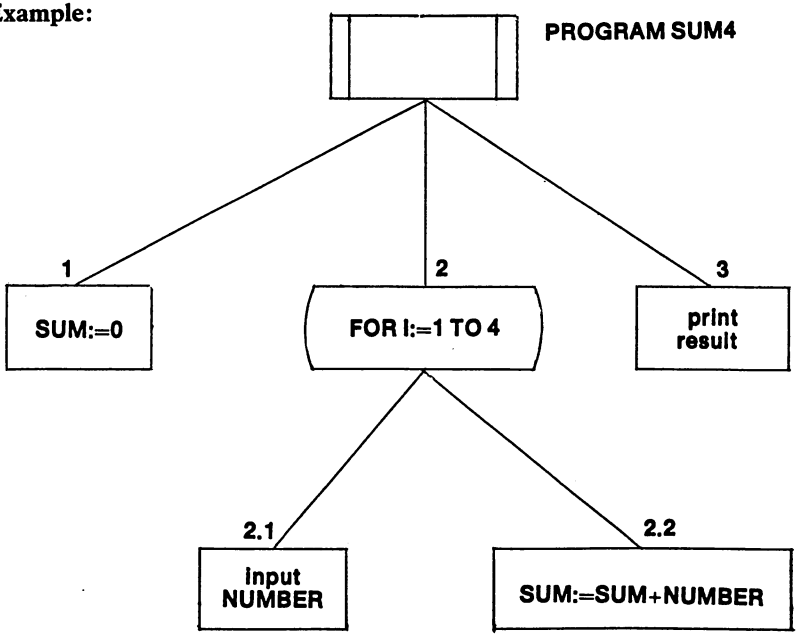
	SUM	NUMBER	I
SUM:=0	0		
I:=1			1
INPUT NUMBER		11	
SUM:=SUM+NUMBER	11		
I:=2			2
INPUT NUMBER		47	
SUM:=SUM+NUMBER	58		
I:=3			3
INPUT NUMBER		16	
SUM:=SUM+NUMBER	74		
I:=4			4
INPUT NUMBER		15	
SUM:=SUM+NUMBER	89		

Fig. 4.2

The symbol for FOR . . . NEXT (and the other ways to repeat executions) is:



Example:



Remember that lower-case letters are used for things that need further explanation, while capitals are used for parts of the actual code.

Don't forget:
All programs, except the very simplest ones, should be written in the following way:
(a) Draw structure diagrams – reasonably detailed
(b) Code.
If the structure diagrams are reasonably detailed, the coding is more or less routine!

Exercise 4.6*

Write a program that computes and prints the sum of an arbitrary but known set of numbers.

Exercise 4.7*

Write a program that computes and prints the product of an arbitrary but known set of numbers.

Exercise 4.8

Complete the program in Exercise 4.6 in such a way that the average of the numbers is also computed and printed.

Exercise 4.9*

Construct a program that draws rectangles. The following should appear on the screen:

```
THIS PROGRAM DRAWS RECTANGLES.
HEIGHT? 5
WIDTH? 10
```

The rectangle should then be printed on paper:

```
* * * * * * * * * *
*                 *
*                 *
*                 *
* * * * * * * * * *
```

Exercise 4.10

Construct a program that produces an arbitrary but known number of membership cards of the type discussed in Exercise 2.14.

Exercise 4.11

Each string has a length which is denoted by LEN. For instance,

LEN("ABCD") is equal to 4. Modify each membership card in such a way that the member's name is written this way:

```
= = = = = = = = =
BILL COMPSON
= = = = = = = = =
```

Hint:
If A$="CHARLES", then the program segment:

```
270 L:=LEN(A$)
280 PRINT
290 PRINT A$
300 FOR I:=1 TO L DO
310    PRINT "+",
320 NEXT I
330 PRINT
```

gives the output:

```
CHARLES
++++++
```

Exercise 4.12

Write a program (using FOR . . . NEXT) that produces bills on paper. On the screen:

```
CUSTOMER'S NAME? BILL COMPSON
NOFLINES? 3
LINE 1
======
ARTICLE NUMBER? 31
PRICE? 3.47
QUANTITY 5
LINE 2
======
```

and so on. The bill looks something like this:

CUSTOMER'S NAME: BILL COMPSON

ARTICLE NUMBER	PRICE	QUANTITY	COST
31	3.47	5	17.35
....	....	..	.....
....	....	..	.....

TOTAL COST xxxx DOLLARS

Exercise 4.13*

Construct a program that produces an arbitrary but known number of bills like the one above.

Exercise 4.14*

In a group of a known number of people, you have for each of these the following details: (a) their recorded weight (in an arbitrary weight unit); and (b) their gender. Construct a program that produces the following statistics, for example:

```
30 PEOPLE
MEAN WEIGHT 69.7 KILOGRAMS
OF THESE, 17 FEMALES WITH MEAN WEIGHT xxxx KILOGRAMS
AND 13 MALES WITH MEAN WEIGHT xxxx KILOGRAMS.
```

If there are no females print only the first two lines plus:

```
NO FEMALES
```

or, if there are no males, print the first two lines plus:

```
NO MALES
```

(Information about gender and weight are entered via INPUT.)

Exercise 4.15*

Construct a program that draws calendars. Enter the month, year and the first day of the month (for example Thursday). Then draw a calendar like the one below, including the month and the year in the heading.

SUN	MON	TUE	WED	THU	FRI	SAT
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

4.2 OTHER CONSTRUCTIONS

Execute the following small program:

```
10 REPEAT
20   INPUT NUMBER
30 UNTIL NUMBER=7
```

You can go on entering numbers for ever with this program, until you enter the number 7, so that the condition NUMBER=7 in line 30 is fulfilled.

Exercise 4.16

What does this small program accomplish? Rewrite it using FOR . . . NEXT.

```
10 I:=5
20 SUM:=0
30 REPEAT
40   SUM:=SUM+I
50   I:=I+1
60 UNTIL I=10
```

The difference between REPEAT . . . UNTIL and FOR . . . NEXT is that REPEAT . . . UNTIL can be used when the number of loops is unknown.

Example:

Assume that an arbitrary number of positive numbers are to be entered via INPUT on the screen, for instance in this way:

```
ENTER AN ARBITRARY NUMBER OF POSITIVE NUMBERS.
THE PROGRAM STOPS WHEN A NUMBER <=0 IS ENTERED.
THEN IT PRINTS THE SUM OF ALL THE POSITIVE NUMBERS ENTERED.
WHICH NUMBER? 5
WHICH NUMBER? 7
WHICH NUMBER? -3
THE SUM IS 12
```

A program that does this can look the following way:[3]

```
10-30 (the usual identifying information)
40-60 (informative PRINT statements)
 70 SUM:=0
 80 REPEAT
 90   INPUT"A NUMBER, PLEASE!":NUMBER
100   IF NUMBER > 0 THEN SUM:=SUM+NUMBER
110 UNTIL NUMBER <=0
120 PRINT"THE SUM IS ",SUM
```

3. In some older versions, one-line IF . . . THEN statements are not permitted. In this case, use the following:

```
IF NUMBER >0 THEN
  SUM:=SUM+NUMBER
ENDIF
```

In this example, until the value of NUMBER is <=0 the lines between REPEAT and UNTIL is not limited to two – you can use as many as you need:

```
REPEAT
  xxxxxxxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxxxxxxx      } an arbitrary number of lines
UNTIL condition
```

Note that the condition is tested *after* the lines have been executed. This means that all the lines are executed at least once.

Exercise 4.17

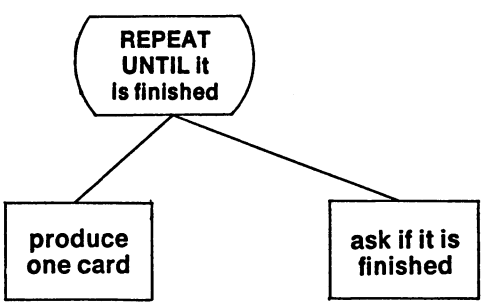
What does the following program segment accomplish?

```
480 REPEAT
490   PRINT"WHICH POSITIVE NUMBER",
500   INPUT Q
510 UNTIL Q>0
```

Exercise 4.18

Construct a similar segment that only lets through a number I if it is an integer.

Suppose that the cashier of the Triangle Computer Club wants to be able to produce several membership cards in the same execution. You know how to write a program that produces *one* card. What your new program should do is described in the following structure diagram (the symbol for REPEAT . . . UNTIL is the same as that for FOR . . . NEXT):



One way to finish the program would be:

```
PRINT
PRINT"DO YOU WANT TO FINISH - Y(ES) OR N(O)",
INPUT ANSWER$
```

These lines appear just before UNTIL. The answer you input will be one character only, because somewhere earlier in the program (probably at the beginning) you will have declared:

```
DIM ANSWER$ OF 1
```

Exercise 4.19

Using the code from the previous version of this program (see Exercises 4.10 and 2.14), construct the new version of this program.

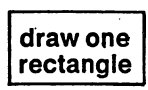
Exercise 4.20

Extend the structure diagram in Exercise 4.19 so that the following kind of output is obtained on paper when it is run:

```
5 CARDS      TOTAL FEE 21 DOLLARS
```

Exercise 4.21

Modify the rectangle program (see Exercise 4.9) in such a way that it can be used for an arbitrary number of rectangles. Start by drawing a structure diagram in which one box is denoted:



Exercise 4.22

Improve the last program so that only positive values are accepted as input. (Even if the user tries to enter negative values several times!)

Exercise 4.23

Execute the small program:

```
10 I:=7
20 IF I=7 THEN STOP
30 PRINT "I=",I
```

What is the effect of STOP? You can resume the execution by typing CON (short for *continue*). Try doing so now.

Exercise 4.24*

Modify the rectangle program (see Exercises 4.21, 4.22) in such a way that only five attempts are permitted for each entered value. At all attempts after the first a warning should be given, and at the fifth attempt there should be a final warning.

Instead of REPEAT . . . UNTIL you can always use the construction WHILE . . . ENDWHILE. The first example with REPEAT . . . UNTIL would now become:

```
10 INPUT NUMBER
20 WHILE NUMBER <>7 DO
30   INPUT NUMBER
40 ENDWHILE
```

As long as the numbers entered are different from 7 the program will go on.

Exercise 4.25

Will the following program accomplish the same as the program given in Exercise 4.16, using REPEAT . . . UNTIL?

```
10 I:=5
20 SUM:=0
30 WHILE I<10 DO
40   SUM:=SUM+I
50   I:=I+1
60 ENDWHILE
```

The program that totals an arbitrary number of positive numbers entered via INPUT now becomes:

```
.
.
.
 70 SUM:=0
 80 INPUT "A NUMBER, PLEASE! ":NUMBER
 90 WHILE NUMBER >0 DO
100   SUM:=SUM+NUMBER
110   INPUT "A NUMBER, PLEASE!":NUMBER
120 ENDWHILE
130 PRINT "THE SUM IS ",SUM
```

As long as the value of NUMBER is positive the lines between WHILE and ENDWHILE (i.e. lines 100-110) will be executed again and again.

Exercise 4.26
Why isn't there any need, this time, of an IF . . . THEN statement?

Exercise 4.27
Try to rewrite the small program segments in Exercises 4.18 and 4.19 using WHILE . . . ENDWHILE instead of REPEAT . . . UNTIL.

```
WHILE condition DO
  xxxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxxx
ENDWHILE
```

} an arbitrary number of lines

Note that with WHILE . . . ENDWHILE the condition is tested *before* the lines are executed. This means that sometimes the lines aren't executed at all.

Exercise 4.28
What does the following small program do:

```
 40 PRINT "HOW MANY EGGS",
 50 INPUT NOFEGGS
 60 NOFBOXES:=0
 70 WHILE NOFEGGS>=12 DO
 80    NOFEGGS:=NOFEGGS-12
 90    NOFBOXES:=NOFBOXES+1
100 ENDWHILE
```

Exercise 4.29
Rewrite the program in Exercise 4.28 above using REPEAT . . . UNTIL.

Exercise 4.30
The problem solved by the program in Exercise 4.28 can also be solved without loops. How?

Exercise 4.31
In a group of an unknown number of people you have for each of these: (a) their recorded weight; and (b) their gender. The values are entered in this way:

WHICH SEX – M(ALE) OR F(EMALE) – ENTER C(LEAR) TO FINISH?

The statistics should be presented in the same way as in Exercise 4.14.
(a) Use REPEAT . . . UNTIL.
(b) Use WHILE . . . ENDWHILE.
(Start (as usual) by drawing structure diagram(s).)

Loops or Iterations

The use of loops or iterations allows certain lines of program to be repeated. Three forms of loops are discussed in Figure 4.3.

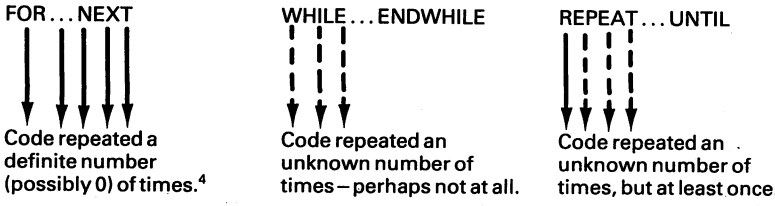


Figure 4.3

Exercise 4.32

Rewrite the programs of Exercises 4.3, 4.4 and 4.5 using (a) REPEAT ... UNTIL, and (b) WHILE ... ENDWHILE.

Check that you get the same results with both methods.

4.3 MAXIMUM AND MINIMUM

Very often, usually as a part of a larger program, you have to find the maximum or minimum of two, three, or any number of numbers. If you just have two numbers it's quite simple to find the maximum. If the numbers are x and y:

```
IF x>y THEN
  MAX:=x
ELSE
  MAX:=y
ENDIF
```

Exercise 4.33

Write a program to find the minimum of two numbers.

How can you find the maximum of three numbers, $X1$, $X2$ and $X3$ for instance?

---

4. For example,

```
FOR I:=5 TO 3 DO
  PRINT"SOMETHING"
NEXT I
```

doesn't produce anything at all. Note that this is in marked contrast to almost all ordinary BASIC dialects, where a FOR ... NEXT loop is executed at least once.

Start with MAX:=*X*1, then let MAX be the greatest of MAX and *X*2. MAX will therefore be the greatest of *X*1 and *X*2, then let MAX be the greatest of MAX and *X*3. MAX will now be the greatest of *X*1 and *X*2 and *X*3!

The code for this might look like:

```
MAX:=X1
IF MAX>X2 THEN MAX:=X2
IF MAX>X3 THEN MAX:=X3
```

You can write the IF . . . THEN statement in two ways. Either:

```
IF MAX>X2 THEN MAX:=X2
```

or (if one-line IF . . . THEN statements are not permitted on your system):

```
IF MAX>X2 THEN
  MAX:=X2
ENDIF
```

Exercise 4.34

Write the code of a program to find the minimum of:

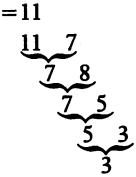
(a) three numbers A, B and C
(b) four numbers, FIRST, SECOND, THIRD and FOURTH.

If the computer is to find the smallest of five numbers, for instance:

11 7 8 5 3

The following happens:

MIN:=11



Exercise 4.35

Write a program that accepts via INPUT a variable but definite number of numbers and makes the following:

```
HOW MANY NUMBERS? 4
WHAT IS NUMBER 1? 5
WHAT IS NUMBER 2? 7
WHAT IS NUMBER 3? 6
WHAT IS NUMBER 4? 9
MAXIMUM OF THE 4 NUMBERS IS 9
```

Exercise 4.36*

Modify your program to give the following output:

MAXIMUM OF THE 4 NUMBERS IS 9 (NUMBER 4)

Exercise 4.37

Modify your program (use IF . . . THEN . . . ELSE) to give this output instead:

MINIMUM OF THE 4 NUMBERS IS 5
MAXIMUM OF THE 4 NUMBERS IS 9

Exercise 4.38*

In one whole year a cigarette smoker writes down his weekly consumption of cigarettes. Call the weeks 1, 2, 3 . . . 52. Enter these values via INPUT.

(a) Write a program that gives the following information:

MINIMUM CONSUMPTION IN ONE WEEK 27 CIGARETTES
MAXIMUM CONSUMPTION IN ONE WEEK 83 CIGARETTES

(b) How should you modify the program if this information is required:

MINIMUM CONSUMPTION IN ONE WEEK 27 CIGARETTES
THIS HAPPENED IN WEEK NR 3
MAXIMUM CONSUMPTION IN ONE WEEK 83 CIGARETTES
THIS HAPPENED IN WEEK NR 37

(c) How would you get the following information?

FROM WEEK NR 3 TO WEEK NR 4 THE INCREASE WAS MAXIMAL – 26 CIGARETTES!

(d) Make the program provide some statistics, at least the mean value.
(e) If the minimal consumption occurs in more than one week note this by:

THIS HAPPENED IN WEEK NR 3 (PLUS 2 OTHERS)

Do not use the number 52 in the code but use instead the variable NOFWEEKS, which should have been initialized (i.e. given a value) early in the program.

```
NOFWEEKS:=52
```

(f) What is the point of this?
(g) If you – for the purpose of testing the program – change only this line to for instance:

```
NOFWEEKS:=10
```

you can execute your programs without much effort. Try it!

You have seen many examples of comparisons between numbers, but you can also compare strings:

"BILL"<"BUFFALO" (alphabetical order)
"BILL"<"BILLMAN" (alphabetical order)
"BILL4"<"BILLMAN" (this is system-dependent – try it on your computer)
"1234"<"47" (1 comes before 4 (don't confuse the *number* 1234 with the *string* "1234"))[5]

Exercise 4.39

Construct a program that does the following:

```
HOW MANY EXAMPLES(ENTER 0 TO STOP) 3
ENTER FIRST STRING:BUFFALO
ENTER SECOND STRING:BILL
BILL<BUFFALO
ENTER FIRST STRING:     BUFFALO
ENTER SECOND STRING:BILL
  BUFFALO<BILL
.
.
.
```

Exercise 4.40

The last answer in Exercise 4.40 above seems strange: can you explain why

```
BUFFALO<BILL?
```

Warning:
Never use code such as:

```
INPUT"ENTER TWO STRINGS:":ASTR$,BSTR$
```

Avoid using INPUT with more than one variable if strings are involved: depending on your system this *can* have various strange effects!

---

5. There is a function STR$ in COMAL so that

```
STR$(456)="456"
```

(a number) (a string)

There is also another function, VAL, which is the inverse of STR:

```
VAL("456")=456
```

(Some versions of COMAL – mainly the older versions – do not support these functions.)

5 Procedures

5.1 INTRODUCTION

Suppose that you have a "tool" called STARS which you can use as a "building-block" in your programs in such a way that executing:

```
170 EXEC STARS(7) or 170 STARS(7)[1]
```

draws 7 stars:

```
*******
```

and executing (the value of Q is 11)

```
130 EXEC STARS(Q)
```

draws 11 stars:

```
***********
```

You could then use this tool to construct more complicated figures:

```
210 FOR I:=1 TO 5 DO
220    EXEC STARS(I)
230 NEXT I
```

will print this figure:

```
*
**
***
****
*****
```

1. What concerns the keyword EXEC there are 3 alternatives. 1) It's necessary to include it when using procedures, 2) It's impossible to include it, 3) You can choose from time to time (usually with the command SETEXEC) cross over the two inapplicable alternatives. If your alternative is 2) or if it is 3) and you prefer not to use the key-word EXEC just cross it over in the following.

Exercise 5.1*

Write the code to draw

(a)

```
****
***
**
*
```

(b)

```
*
***
*****
```

STARS is an example of a *procedure*: it has one *parameter*. A parameter is a variable which can have different values each time the procedure is used. A procedure can have any number of parameters. But how does the computer know what to do when encountering a line such as

```
170 EXEC STARS(7)
```

The answer is, of course, that the procedure must be defined somewhere, like this (see Figure 5.1):

```
 50 PROC STARS(NOFSTARS)
 60   FOR STARNR:=1 TO NOFSTARS DO
 70     PRINT "*",
 80   NEXT STARNR
 90   PRINT
100 ENDPROC STARS
```

or like this:

```
2070 PROC STARS(Z)
2080   FOR I:=1 TO Z DO
2090     PRINT "*",
2100   NEXT I
2110   PRINT
2120 ENDPROC STARS
```

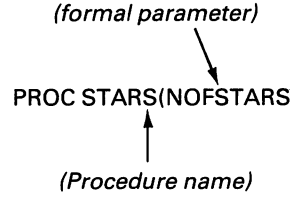


Figure 5.1 *Procedure heading*

Note that whether you denote a formal parameter by NOFSTARS or Z is entirely unimportant from the computer's point of view – but extremely important from a reader's point of view! Note also that from the computer's point of view it is unimportant *where* (in the program) the code for a procedure is placed.[2]

In a program you should start with the identifying information, follow with most of the declarations (i.e. DIM statements), then the procedures and finally the main program. For example:

```
10 // PROGRAM PROCDEMO1
20 // VERSION 1.1   9 DEC 1982
30 // BY BILL COMPSON
40 //
50 // The program demonstrates how to use procedures.
60 // ===== PROCEDURES =====
70 PROC STARS(NOFSTARS)
80   FOR STARNR:=1 TO NOFSTARS DO
90     PRINT "*",
100  NEXT STARNR
110  PRINT
120 ENDPROC STARS
130 // ===== MAIN PROGRAM =====
140 FOR I:=1 TO 5 DO
150   EXEC STARS(I)
160 NEXT I
```

Suppose that instead you had a more advanced procedure called

DRAWSTARS

such that

```
350 EXEC DRAWSTARS(5,7)
```

prints 7 stars starting from 5, i.e.:

```
*******
↑
(pos. 5)
```

Exercise 5.2

What will the program lines below accomplish?

```
200 FOR M:=0 TO 10 DO
210   EXEC DRAWSTARS(20-M,2*M+1)
220 NEXT M
```

2. Unless the numbering makes it appear inside another procedure.

A complete program might look like this:

```
10 //PROGRAM PROCDEMO2
20 //VERSION 1.0  9 DEC 1982
30 //BY BILL COMPSON
40 //
50 //The program demonstrates how to use procedures.
60 //=====PROCEDURES=====
70 PROC DRAWSTARS(STARTPOS,NOFSTARS)
80   PRINT TAB(STARTPOS),
90   STARNR:=0
100  REPEAT
110    STARNR:=STARNR+1
120    PRINT "*",
130  UNTIL STARNR:=NOFSTARS
140  PRINT
150 ENDPROC DRAWSTARS
160 //=====MAIN PROGRAM===
200 FOR M:=0 TO 10 DO
210   EXEC DRAWSTARS (20-M,2*M+1)
220 NEXT M
```

Exercise 5.3

(a) Now execute the program in Exercise 5.2 to see what it actually does,
(b) save it
(c) then change lines 200-220 to:

```
200 FOR STARNR:=1 TO 4 DO
210   EXEC DRAWSTARS(10,5)
220 NEXT STARNR
```

(d) What do you think will be obtained on the screen?
(e) Execute the program to check your guess.

Instead of a rectangle five stars across and four stars high something else is obtained. The reason that something goes wrong is simply that the same variable STARNR is used both in the main program and inside the procedure. When the procedure is executed the value of STARNR is changed, and the main program is influenced in an unintentional way. This is called a side-effect and usually such side-effects have to be eliminated. In COMAL you can avoid such side-effects by closing procedures. This is done by adding the word CLOSED to the procedure head. For example:

```
70 PROC DRAWSTARS(STARTPOS,NOFSTARS) CLOSED
```

Exercise 5.4

Run the above program, making that small change to close the procedure. In this case you have two *different* boxes named STARNR, one *inside* the procedure and one *outside* the procedure (in the main program).

The only way for a *closed* procedure to exchange information with the rest of the program is via the formal parameters.[3]

The following example might show the effect of closing a procedure:

```
 40 PROC AHA1 CLOSED
 50   A:=7
 60 ENDPROC AHA1
 70 PROC AHA2
 80   A:=9
 90 ENDPROC AHA2
100 //
110 A:=5
120 PRINT "A=",A
130 EXEC AHA1
140 PRINT "AFTER AHA1 A=",A
150 EXEC AHA2
160 PRINT "AFTER AHA2 A=",A
```

Exercise 5.5

(a) What will be the result of running the program above? Try to figure this out without actually executing it.

(b) What will happen if lines 110 and 120 are deleted?

Now load the program you saved on the previous page (PROCDEMO2). Check that lines 200–220 are:

```
200 FOR M:=0 TO 10 DO
210   EXEC DRAWSTARS(20-M,2*M+1)
220 NEXT M
```

Exercise 5.6

What happens if, for instance, you change 10 to 15 and 20 to 17?

Obviously, you can generalize this to a procedure:

```
PROC STARTRIANGLE(MIDPOS,HEIGHT)
```

A program to test this procedure might look like this:

---

3. Later in this book you will learn how to get round this problem by "importing" variables in a *controlled* fashion (i.e. without unwanted side-effects).

```
10 //PROGRAM TRIANGLEDRAWING
20 //VERSION 1.2 11 DEC 1982
30 //BY BILL COMPSON
40 //
50 //This program draws triangles of different sizes and locations.
60 //=====PROCEDURES=====
70 PROC DRAWSTARS(STARTPOS,NOFSTARS) CLOSED
80   PRINT TAB(STARTPOS),
90   STARNR:=0
100  REPEAT
110    STARNR:=STARNR+1
120    PRINT "*",
130  UNTIL STARNR:=NOFSTARS
140  PRINT
150 ENDPROC DRAWSTARS
160 //-----------------------------------
170 PROC STARTRIANGLE(MIDPOS,HEIGHT)
180   FOR M:=0 TO HEIGHT - 1 DO
190     EXEC DRAWSTARS(MIDPOS-M,2*M+1)
200   NEXT M
210 ENDPROC STARTRIANGLE
220 //=====MAIN PROGRAM=====
230 PRINT "VALUE<=0 FOR HEIGHT FINISHES THE EXECUTION"
240 REPEAT
250   INPUT "HEIGHT?":H
260   IF H>0 THEN
270     INPUT "MIDPOSITION?":MP
280     EXEC STARTRIANGLE(MP,H)
290   ENDIF
300 UNTIL H<=0
```

Exercise 5.7
Execute the program above a few times.

If you close the procedure STARTRIANGLE (i.e. in this case extend line 170 with the word CLOSED) different things will happen, depending on which system you use. In most COMAL versions this is not permitted because the *procedure* DRAWSTARS is *defined outside* but *used inside* STARTRIANGLE. Therefore you need an extra line:

```
151 IMPORT DRAWSTARS
```

to make DRAWSTARS accessible inside the closed procedure STARTRIANGLE.

In some versions closing a procedure only closes it for *variables* but *not for procedures*, i.e. you may freely use all other procedures within a closed procedure.

In some versions instead of IMPORT the word GLOBAL is used (usually to make variables inside a closed procedure accessible from outside).

SYSTEM NOTE

A closed procedure may / must not freely use other procedures defined outside the procedure. The effect of GLOBAL / IMPORT is to .....

.....

.....

.....

(delete inapplicable alternatives)

If you want to make STARTRIANGLE "self-contained", you can in most systems define DRAWSTARS inside STARTRIANGLE. (The drawback is that you cannot then use DRAWSTARS outside STARTRIANGLE unless you duplicate the code.) Lines 70-220 now become:

```
 70 PROC STARTRIANGLE(MIDPOS,HEIGHT)CLOSED
 80   PROC DRAWSTARS(STARTPOS,NOFSTARS) CLOSED
 90   PRINT TAB(STARTPOS),
100   STARNR:=0
110   REPEAT
120     STARNR:=STARNR+1
130     PRINT "*",
140   UNTIL STARNR:=NOFSTARS
150   PRINT
160   ENDPROC DRAWSTARS
170   FOR M:=0 TO HEIGHT-1 DO
180     EXEC DRAWSTARS(MIDPOS-M,2*M+1)
190   NEXT M
200 ENDPROC STARTRIANGLE
```

Exercise 5.8

Using the procedure STARTRIANGLE make a program that draws a Christmas tree:

```
   *
  ***
   *
  ***
 *****
   *
  ***
 *****
*******
```

Exercise 5.9

Using the procedure DRAWSTARS, construct a closed procedure STARTRIANGLED (D stands for DOWN) that draws triangles like this one:

```
*****
 ***
  *
```

Exercise 5.10

If you wanted to draw upright triangles made up of arbitrary characters, such as +, for instance, you might use the procedure

TRIANGLE(MIDPOS,HEIGHT,CHARACTER$).

This functions just like STARTRIANGLE but with the star replaced by the character CHARACTER$. Construct such a procedure using

DRAW(STARTPOS,NOFSIGNS,CHARACTER$)

as a building block. (*Hint:* see the examples shown in Figure 5.2.)

```
 10 PROC TEXT1(N$)CLOSED
 20   L:=LEN(N$)
 30   PRINT N$
 40   FOR I:=1 TO L DO
 50     PRINT"=",
 60   ENDFOR I
 70   PRINT
 80 ENDPROC TEXT1
 90 //--------------------------------
100 PROC TEXT2(N$,S$)CLOSED
110   L:=LEN(N$)
120   PRINT N$
130   FOR I:=1 TO L DO
140     PRINT S$,
```

```
150   ENDFOR I
160   PRINT
170 ENDPROC TEXT2
180 //--------------------------------
190 PROC TEXT3(N$,STARTPOS)CLOSED
200   L:=LEN(N$)
210   PRINT TAB(STARTPOS),N$
220   PRINT TAB(STARTPOS),
230   FOR I:=1 TO L DO
240     PRINT"=",
250   ENDFOR I
260   PRINT
270 ENDPROC TEXT3
280 //--------------------------------
290 PROC TEXT4(N$,S$,STARTPOS)CLOSED
300   L:=LEN(N$)
310   PRINT TAB(STARTPOS),N$
320   PRINT TAB(STARTPOS),
330   FOR I:=1 TO L DO
340     PRINT S$,
350   ENDFOR I
360   PRINT
370 ENDPROC TEXT4
380 //--------------------------------
390 EXEC TEXT1("CHARLES")
400 EXEC TEXT2("CHARLES","+")
410 EXEC TEXT3("CHARLES",20)
420 EXEC TEXT4("CHARLES","!",20)

CHARLES
=======
CHARLES
++++++

                    CHARLES
                    =======
                    CHARLES
                      !!!!!!!
```

Figure 5.2 ***Four Different Procedures***

You can now see how procedures extend COMAL-80 and make it more powerful: a much larger and more useful language (see Figure 5.3).

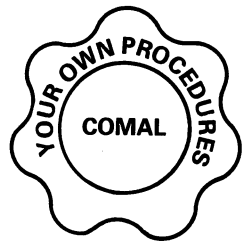


Figure 5.3

The advantage of *closing* procedures is that you get clean tools which can be used as building blocks in many different programs.

Exercise 5.11*

Construct a procedure

DRAWRECT(LEFTPOS,WIDTH,HEIGHT)

that draws a rectangle of width equal to WIDTH, height equal to HEIGHT and the leftmost vertical side in position LEFTPOS.

Exercise 5.12*

Using this procedure, it is quite simple to construct a procedure DRAWSQUARE(LEFTPOS,SIDELENGTH) that draws a square. Try doing so.

Exercise 5.13

If you want to be able to draw rectangles and squares with characters other than the star you may use a formal parameter WHICHCHARACTER$, which contains the character to be used, for instance +. Call them DRAWRECTO and DRAWSQUAREO.

To use the procedure DRAW you only need to know the values of the two formal parameters.

Which information can differ from one membership card to another in the Triangle Computer Club?

```
====================================
THE TRIANGLE COMPUTER CLUB
++++++++++++++++++++++++++++++++++++
MEMBERSHIP CARD FOR[BILL COMPSON]
[1982]  [5]DOLLARS        CARD NO.  [347]
.....
(USER'S SIGNATURE)
====================================
```

The boxes marked above can be filled with different contents.

NAME$	(in this case BILL COMPSON);
YEAR	(in this case 1982);
FEE	(in this case $5)
NO.	(in this case 347).

A procedure CARD 1(NAME$,YEAR,FEE,NO.) which prints such a card might be defined this way (line numbers omitted):

```
PROC CARD1(NAME$,YEAR,FEE,NO)
   SELECT OUTPUT "LP"
   EXEC ONESIGN("=",40)
   EXEC BLANKLINES(1)
   PRINT "THE TRIANGLE COMPUTER CLUB"
   EXEC BLANKLINES(1)
   EXEC ONESIGN("+",40)
   EXEC BLANKLINES(1)
   PRINT "MEMBERSHIP CARD FOR",NAME$
   EXEC BLANKLINES(1)
   PRINT YEAR,TAB(12),FEE," DOLLARS CARD NO.",NO.
   EXEC BLANKLINES(3)
   EXEC ONESIGN(".",30)
   PRINT "(USERS SIGNATURE)"
   EXEC BLANKLINES(2)
   EXEC ONESIGN("=",40)
   EXEC BLANKLINES(2)
   SELECT OUTPUT"DS"
ENDPROC CARD1
```

Exercise 5.14*

Define BLANKLINES and ONESIGN. Both these are to be *closed*.

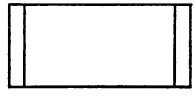
Exercise 5.15

Can you make CARD1 closed?

Exercise 5.16

Using the structure diagram in Figure 5.4 (on the next page) construct the corresponding program. Before you start coding, decide whether you find the structure diagram sufficiently detailed. If you think it is not, you must first break it down. For example, doesn't box 4 need a thorough refinement?

Note that when you use a procedure inside a program, this is indicated by double bars:



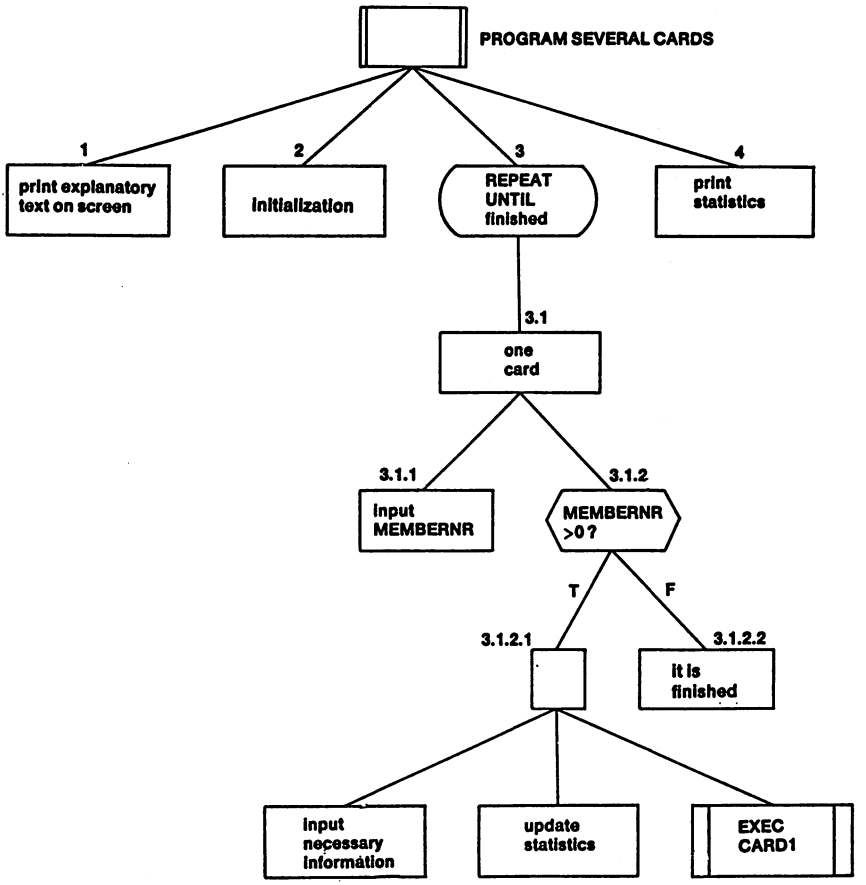


Figure 5.4

It is up to you to decide how many parameters to use in the box: it depends upon how detailed your other boxes are.

A subtle point: Very often you need to exchange the values of two variables.

Example:

A	47
B	73

to be changed to:

A	73
B	47

If this is a common operation, why not construct a procedure:

```
PROC EXCH(X,Y)CLOSED
  GARB:=X
  X:=Y
  Y:=GARB
ENDPROC EXCH
```

A program to check this procedure might look like this:

```
 80 PROC
.
.
.
190 //=====MAIN PROGRAM=====
200 INPUT"A? ":A
210 INPUT"B? ":B
220 EXEC EXCH(A,B)
230 PRINT"THE NEW VALUES ARE"
240 PRINT"A=",A
250 PRINT"B=",B
```

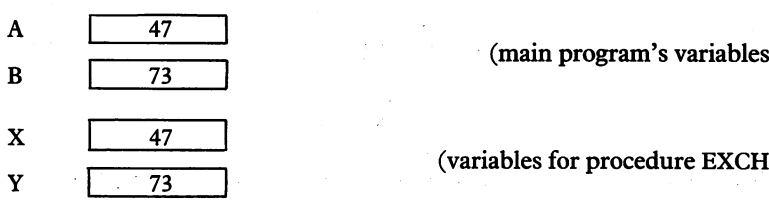
Execute this program. Why doesn't it work?

When EXCH(A,B) is executed the first thing that is done is:

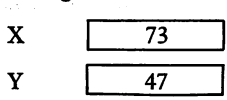
```
X:=A
```

```
Y:=B
```

(i.e., the values of A and B are copied into the procedure's own variables, called X and Y).



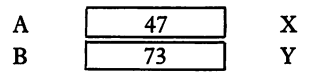
When the procedure is executed, *only* the contents of the boxes X and Y are exchanged:



Nothing has happened to A or B.
However if, instead, you write:

```
PROC EXCH(REF X,REF Y)
```

and don't change the rest of the code for EXCH, then the following happens when the procedure is started:



One could say that X becomes a "nick-name" for box A, and that Y becomes a "nick-name" for box B. Otherwise expressed, X and A denote the same box; and Y and B denote the same box. To exchange the values of X and Y also means to exchange the values of A and B.

Exercise 5.17

Execute the program with the indicated change to check that it works.

Exercise 5.18

Try to figure out – without executing the program – what happens if you have PROC EXCH(REF X,Y).

Using formal parameters information is "imported" to a procedure. To "export" information from a procedure a formal parameter must be used with REF.

Exercise 5.19

What ways are there in your system for a closed procedure to exchange information with the main program?

Hint: What is the effect of the IMPORT (or GLOBAL) statement?

Exercise 5.20

What is wrong with the procedure MAX2(A,B,M)? Execute the following program. (As usual, you compose lines 10-30 by yourself.)

```
 40 PROC MAX2(A,B,M)
 50   IF A>B THEN
 60     M:=A
 70   ELSE
 80     M:=B
 90   ENDIF
100 ENDPROC
110 //=================================
120 INPUT "X? ":X
130 INPUT "Y? ":Y
140 EXEC MAX2(X,Y,Z)
150 PRINT "THE GREATEST VALUE=",Z
```

Explain in detail why this doesn't work, and correct the error(s).

Exercise 5.21

```
PROC P(X)
  X:=X+1
ENDPROC P

PROC PC(X) CLOSED
  X:=X+1
ENDPROC PC

PROC PR(REF X)
  X:=X+1
ENDPROC PR

PROC PRC(REF X) CLOSED
  X:=X+1
ENDPROC PRC
```

(a) What will result from an execution of this small program:

```
100 Z:=7
110 PRINT Z
120 EXEC P(Z)
130 PRINT Z
```

Answer without executing the program, then enter these lines, plus the code for the procedure, into your computer and check your guess.

Change line 120 in the above program to:

(b) 120 EXEC PC(Z)
(c) 120 EXEC PR(Z)
(d) 120 EXEC PRC(Z)

What will result from these changes? Now rerun the program to see what actually happens.

Exercise 5.22

Now enter a line Y:=4711 into the four procedures above. Change the program by entering

```
105 Y:=0
115 PRINT Y
125 PRINT Y
```

Then answer questions (a)–(d) above.

5.2 RANDOM NUMBERS

In each COMAL system it is possible to generate sequences of random numbers using RND or RND( ). In some versions (for instance RCs) the function RND (no parenthesis) can be used to generate random numbers >0 and <1, for instance the following way:

```
10 // THIS PROGRAM GENERATES 10 RANDOM NUMBERS
20 FOR I:=1 TO 10 DO
30   RANDOMNUMBER:=RND
40   PRINT RANDOMNUMBER IF
50 NEXT I
```

If you execute this program (provided your system has this facility) a few times you will probably get the same sequence each time. If that is the case, insert:

```
15 RANDOMIZE
```

(depending on your system you may have to use RANDOM rather than RANDOMIZE) and execute the modified program a few times to see that it works.

In this program instead of lines 30 and 40 you can use one line, for instance:

```
30 PRINT RND
```

Exercise 5.23 (for use on systems with RND, I.E. without ())

```
10 //THIS PROGRAM GENERATES AND SUMS 10 RANDOM NUMBERS
20 SUM:=0
30 RANDOMIZE
40 FOR I:=1 TO 10 DO
50   RANDOMNUMBER:=RND
60   PRINT RANDOMNUMBER
70   SUM:=SUM+RANDOMNUMBER
80 NEXT I
90 PRINT "SUM=",SUM
```

Does this program give new sequences of random numbers each time it is used?

In this program you cannot replace lines 50-70 with

```
50 PRINT RND
60 SUM:=SUM + RND
```

The reason for this is that each RND is a new random number.

In other COMAL versions you use RND( ), i.e. RND with a value. If you have such a system try the following program for some different values of X and Y.

```
10 INPUT "X?":X
20 INPUT "Y?":Y
30 PRINT RND(X)
40 FOR I:=1 TO 10 DO
50   RANDOMNUMBER:=RND(Y)
60   PRINT RANDOMNUMBER
70 NEXT I
```

Exercise 5.24 (for use on systems with RND( ))
For which values of X and Y (if any) do you get the same sequence of numbers in two (or more) consecutive executions of the program above.

Exercise 5.25
What corresponds to RANDOM or RANDOMIZE in a system with RND()?

In most COMAL systems there is also a function RND(,), where the numbers in brackets must be integers.[4]

Example:
RND(1,6) should with equal probability, i.e. on average every sixth time, assume each one of the values 1, 2, 3, 4, 5, 6.

SYSTEM NOTE

The rules for generating random numbers in the interval ..

.....

.....

The rules for generating integer random numbers are .....

.....

.....

---

4. If your system hasn't got the RND(,) facility, you can get round this in the following way: assume for example that you want RND(1,6). If R is a random number of the usual kind i.e. 0<R<1 then 6*R is randomly situated in the interval 0<X<6. I.e. INT(6*R) has the same effect as RND(1,6), where 1 is the lowest possible value of the number, and 6 is the number of available alternatives.

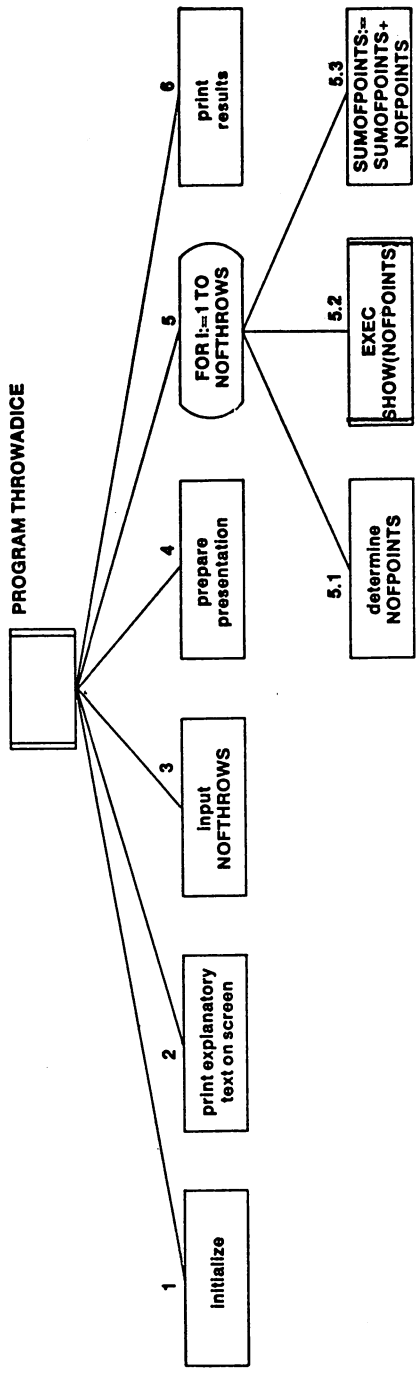

PROGRAM THROWADICE
1
initialize
2
print explanatory text on screen
3
input NOFTHROWS
4
prepare presentation
5
FOR I:=1 TO NOFTHROWS
5.1
determine NOFPOINTS
5.2
EXEC SHOW(NOFPOINTS)
5.3
SUMOFPOINTS:= SUMOFPOINTS+ NOFPOINTS
6
print results

Figure 5.5

Now the problem is to construct a program that simulates throwing a dice a definite number of times. Every throw should be illustrated on the screen. The first structure diagram might look like the one in Figure 5.5. The box marked initialize should appear in all, or almost all, structure diagrams. In this case it might include a line with RANDOM or RANDOMIZE if that is necessary to get differing sequences of throws at different executions.

```
 10 //PROGRAM THROWADICE
 20 //1 SEP 1982 VER 1.0
 30 //BY BILL COMPSON
 40 //THE PROGRAM SIMULATES FIXED NUMBER OF THROWS
 50 //==========DECLARATIONS==========
 60 DIM ANSWER$ OF 1
 70 //==========PROCEDURES==========
 80 PROC SHOW(OUTCOME)
 90   PRINT OUTCOME
100 ENDPROC SHOW
110 //==========MAIN PROGRAM==========
120 //INITIALIZATION
130 SUMOFPOINTS:=0
140 //PRINT EXPLANATORY TEXT ON SCREEN
150 PRINT
160 PRINT"PROGRAM SIMULATES THROWING A DICE A",
170 PRINT"FIXED NUMBER OF TIMES."
180 PRINT
190 //
200 INPUT"ENTER NUMBER OF THROWS":NOFTHROWS
210 //PREPARE THE PRESENTATION
220 INPUT"ENTER MEDIA(SCREEN OR PAPER)":ANSWER$
230 IF ANSWER$="P"THEN SELECT OUTPUT"LP"
240 //
250 PRINT CHR$(147)//----ERASES THE SCREEN---
260 PRINT"*****************************************"
270 PRINT"SIMULATION OF",NOFTHROWS,"THROWS WITH A
    DICE"
280 PRINT
290 //
300 FOR I:=1 TO NOFTHROWS DO
310   NOFPOINTS=RND(1,6)
320   EXEC SHOW(NOFPOINTS)
330   SUMOFPOINTS:=SUMOFPOINTS+NOFPOINTS
340 NEXT I
350 //PRINT RESULTS
360 PRINT
370 PRINT"****RESULTS****"
```

```
380 PRINT "TOTAL NUMBER OF POINTS=",SUMOFPOINTS
390 MEAN:=SUMOFPOINTS/NOFTHROWS
400 PRINT "MEAN VALUE=",MEAN
410 //
420 IF ANSWER$="P" THEN SELECT OUTPUT"DS"
430 //BACK TO SCREEN!
```

Figure 5.6

Important Note:
As you know, all programs, except the very simplest ones, must be constructed by first *drawing structure diagrams* – reasonably detailed – and then *coding*. The coding should follow the pattern of the program in Figure 5.6.

(a) Four Parts:
  (i) General information (in this case lines 10-40)
  (ii) Declarations (in this case line 60)
  (iii) Procedures (in this case lines 80-100)
  (iv) Main program

(b) You should use remarks, i.e. lines beginning with //, showing the rough outline of the structure diagrams.

Figure 5.6 is a listing of a program written according to the structure diagram in Figure 5.5. Observe how the structure diagram influences the use of comments. Figure 5.7 shows the result of an execution, which shows that the program does work in principle.

```
**********************************************************
********SIMULATION OF 10 THROWS WITH A DICE********
4
1
3
4
4
5
5
2
2
2
****RESULTS****
TOTAL NUMBER OF POINTS=32
MEAN VALUE=3.2
```

Figure 5.7

Now it is time to improve the procedure SHOW to get an illustration of each throw of the dice on the screen. You can change the code for SHOW without touching the rest of the program: simply replace one "black box" with another, hopefully better, one. Or, to put it another way, consider SHOW as a tool, and try to improve that tool.

Exercise 5.26*

Improve SHOW so that it gives a result such as that shown below.

```
EXEC SHOW(6) gives
**
***
```

(and so on).

Exercise 5.27*

How much must you change the structure diagram on p. 82 if at each throw two dice are to be thrown? (Don't forget to change the name of the program.) *Hint:* Use a procedure SHOW2(OUTCOME 1,OUTCOME 2). At this stage don't bother about what it should do in detail.

Exercise 5.28

Construct the program code. First construct SHOW 2 in a trivial way, just to check that the entire program works in principle, then rebuild SHOW 2 to a more sophisticated procedure.

Tossing a Coin Using Random Numbers

Suppose that you want to write a program to simulate the tossing of a coin, but that instead of playing fair, you want to influence the probability of "heads" occurring. You can simulate the tossing of a biased coin in the following way. Determine a random number, for instance

```
R:=RND(1)
```

If R<0.40 then you have "tails", otherwise you have "heads". Call the number which you are comparing with R, SKEWNUMBER. In the beginning of the program you initialize SKEWNUMBER, for instance by

```
SKEWNUMBER:=0.55
```

Exercise 5.29
For which value of SKEWNUMBER should you have an unbiased ("fair") coin?

Exercise 5.30
Draw a structure diagram for this program.
Hint 1: Use the earlier one (Figure 5.3) as much as possible.
Hint 2: Use two procedures SHOWTAIL and SHOWHEAD.

Exercise 5.31
Construct a program according to your structure diagram, first with trivial versions, and then more sophisticated versions of these two procedures.

QUESTIONS

5.1 What is the point of using procedures?

5.2 When is it necessary to use REF in a procedure head?

5.3 Why is it favorable to close procedures?

5.4 Why (at least for more complex problems) is it necessary to draw structure diagrams *first*?

5.5 Rectangular boxes in structure diagrams can be done in two ways. When is each sort used?

5.6 Construct a procedure:[5]

```
TOMORROW(REF YEAR,REF MONTHNR,REF DAYNR)
```

that changes the date by one day. (Compare Question 3.7.) Why is it necessary that all variables are used with REF? Use this to construct a procedure

```
PROC DAYSFROMNOW(NOFDAYS,REF YEAR,REF MONTHNR,REF DAYNR)
```

5.7 Construct a procedure:

```
DATECHECK(YEAR,MONTHNR,DAY,REF OK)
```

5. In this century every year (except 1900) that is a multiple of 4 is a leap year.

so that OK:=1 if there is such a date, and OK:=0 otherwise.

5.8 Construct a procedure:

DAYTYPE(YEAR,MONTHNR,DAY,REF TYPE)

(Sunday has TYPE=0, Monday has TYPE=1,.., Saturday has TYPE=6) so that TYPE denotes the actual day on that date. If there is no such date then TYPE=−1 (in this century). Use the information that January 1 1900 was a Monday.)

6 Arrays

6.1 INTRODUCTION

Very often you need to keep a lot of numbers and/or strings in the computer's internal memory, and to be able to use all these they must be named. So far you have used variable names such as SUM,X,NAME$, and many others, but how would you name the six results from the tossing of a dice in order to count the number of times each number occurs?

One way to do this would be to call them NOF1, NOF2, NOF3, NOF4, NOF5, NOF6. You would then have to initialize them at the beginning of your program, i.e.:

```
NOF1:=0
NOF2:=0
NOF3:=0
NOF4:=0
NOF5:=0
NOF6:=0
```

Suppose that instead you wanted to simulate the tossing of two dice. You would then need twelve different names, and twelve different initialization statements. You would also need a CASE statement to update the result each time the dice is thrown.

```
CASE OUTCOME OF
WHEN 1
  NOF1:=NOF1+1
WHEN 2
  NOF2:=NOF2+1
.
.
.
```

Thus with two dice (i.e. twelve alternatives) the CASE statement will be very long.

To avoid all this trouble you could use an *array*. This should be declared early in the program, for example:

```
DIM NOF(6)
```

This statement reserves six boxes, which are denoted:

```
NOF(1)
NOF(2)
NOF(3)
NOF(4)
NOF(5)
NOF(6)
```

Now the initialization is quite simple:

```
FOR I:=1 TO 6 DO
  NOF(I):=0
NEXT I
```

Instead of the long CASE statement you'll get

```
NOF(OUTCOME):=NOF(OUTCOME)+1
```

Exercise 6.1*

Suppose that you would like to be able to "trace" the simulation, for instance in such a way that on the screen is printed ONE, TWO, SIX, etc. Do this using a CASE statement. *Hint:* To put one space between consecutive words use a semicolon in your PRINT statements.[1]

Just as you can have arrays of numbers, you can also have arrays of strings. These should be declared, for example:

```
DIM THROWNAME$(6) OF 5
```

Then after the following initialization:

```
THROWNAME$(1):="ONE"
THROWNAME$(2):="TWO"
THROWNAME$(3):="THREE"
THROWNAME$(4):="FOUR"
THROWNAME$(5):="FIVE"
THROWNAME$(6):="SIX"
```

1. This does not work in all COMAL versions: investigate the effect of a semicolon in your system.

you have the following boxes:

THROWNAME$(1)	O	N	E		
THROWNAME$(2)	T	W	O		
THROWNAME$(3)	T	H	R	E	E
THROWNAME$(4)	F	O	U	R	
THROWNAME$(5)	F	I	V	E	
THROWNAME$(6)	S	I	X		

Exercise 6.2*

Which very short construction can now replace the rather long CASE construction in the last exercise?

One possible end result of a simulating throwing a dice is given in Table 6.1

Table 6.1
RESULTS
=====

Points	*No of Occurrences*	*Percentage*
===	=========	=====
1	43	21.5
.	.	.
.	.	.
.	.	.
6	37	18.5
TOTAL	200	

Exercise 6.3

Write the code for the program which would produce the table in Table 6.1.

Obviously, tables like these are very common. If, for instance, you simulate the throwing of two dice, this table might then look like Table 6.2.

Table 6.2
RESULTS
=====

Points	*No of Occurrences*	*Percentage*
====	=========	=====
1	57	11.4
.	.	.
.	.	.
.	.	.
12	63	12.6
TOTAL	500	

Exercise 6.4

Write a procedure

```
PROC TABLE1(NOFLINES)
```

that, in the way indicated in Tables 6.1 and 6.2, tabulates

```
NOF(1),NOF(2),... NOF(LINES).
```

You may also need tables for other purposes, such as in Table 6.3. What differs in each table is:

(a) the heading;
(b) number of lines in the table;
(c) name of the array, for instance NOF or NOFPUPILS.

Table 6.3

RESULTS

=====

Marks	*No of Pupils*	*Percentage*
====	=======	======
1	3	12
2	5	20
3	6	24
4	4	16
5	7	28
TOTAL	25	

Exercise 6.5

Make a procedure

```
PROC TABLE(REF VECTOR(),NOFLINES) CLOSED
```

that prints a table without headings, for instance:

1	3	12
2	5	20
3	6	24
4	4	16
5	7	28
TOTAL	25	

The use of parentheses after the array name in a PROC statement indicates that the parameter is an array. Note that the COMAL standard requires that REF must be used in an array. However in some versions, for instance some of Commodore's, this isn't necessary. The advantage of using REF is that much less internal memory is then occupied, because there is no duplication since the actual array is used in the procedure. The disadvantage is that

unintentionally the contents of the array may be changed.

To use this procedure for the table that illustrates the throwing of a dice write:

```
EXEC TABLE(NOF,6)
```

Note that brackets are only used in the definition of the procedure, and that NOF must be an array to "match" that definition. The code for such a procedure (plus a small main program to test it) might look something like the version given in Figure 6.1 Try running this program a couple of times.

```
 10 PROC TABLE(REF VECTOR(),NOFEL)CLOSED
 20   ZONE 12
 30   SUM:=0
 40   FOR I:=1 TO NOFEL DO
 50     SUM:=SUM+VECTOR(I)
 60   NEXT I
 70   FOR I:=1 TO NOFEL DO
 80     PERCENTAGE:=VECTOR(I)/SUM
 90     PRINT I,VECTOR(I),PERCENTAGE
100   NEXT I
110   PRINT "TOTAL",SUM
120 ENDPROC TABLE
130 //=======MAIN PROGRAM=======
140 PRINT "***TESTPROGRAM***"
150 INPUT "WHICH DIMENSION?": D
160 DIM TESTVECTOR(D)
170 FOR I:=1 TO D DO
180   TESTVECTOR(I):=RND(100,900)
190 NEXT I
200 EXEC TABLE(TESTVECTOR,D)
```

Figure 6.1

Exercise 6.6

Obviously there is (at least) one error. Now correct that error and run the program a few times more.

For procedures, just as for programs, you should supply some kind of identifying information at the beginning.[2] In this case, the amended version of the program is shown in Figure 6.2. (Note that the identifying information

2. Such identifying information is essential for procedures that could be used in several different programs.

for the program is still missing.) This procedure can be used as a building block in other programs.

```
 10 PROC TABLE(REF VECTOR(),NOFEL)CLOSED
 20   //5 NOVEMBER 1982 VERSION 1.1
 30   //BY BILL COMPSON
 40   //
 50   ZONE 12
 60   SUM:=0
 70   FOR I:=1 TO NOFEL DO
 80     SUM:=SUM+VECTOR(I)
 90   NEXT I
100   FOR I:=1 TO NOFEL DO
110     PERCENTAGE:=VECTOR(I)/SUM*100
120     PRINT 1,VECTOR(I),PERCENTAGE
130   NEXT I
140   PRINT "TOTAL",SUM
150 ENDPROC TABLE
160 //=======MAIN PROGRAM=======
170 PRINT "***TESTPROGRAM***"
180 INPUT "WHICH DIMENSION?": D
190 DIM TESTVECTOR(D)
200 FOR I:=1 TO D DO
210   TESTVECTOR(I):=RND(100,900)
220 NEXT I
230 EXEC TABLE(TESTVECTOR,D)
```

Figure 6.2

List

You are now used to using commands like LIST20-70 (or LIST 20,70) to list part of the program in the internal memory on screen or paper. But you can also use the command LIST 20-70 filename (or LIST 20,70 filename) to list the actual lines not on screen or paper, but in a file in background storage.

In some systems (notably the last Commodore version) there is one method of composing programs (Method 1 below). In most of the others one uses a slightly different method (Method 2).

Method 1
Assume that the program in Figure 6.2 is in the computer's memory. Simply type:

```
LIST 10-150 "TABLE"
DEL 10-150
```

```
RENUM 1,1
LIST
LIST "MAINTABLE"
```

Check what there is in the background memory, then give the following commands:

```
NEW
MERGE "TABLE"
LIST
MERGE "MAINTABLE"
LIST
```

Now you will have got an idea of what happens. Try typing these commands:

```
NEW
MERGE "MAINTABLE"
LIST
MERGE "TABLE"
LIST
```

Exercise 6.7

Explain the effect of the MERGE command.

Warning: If you use the command ENTER here it has the same effect as NEW followed by MERGE.

Method 2

Supposing that in your system line numbers can be all positive integers up to and including 9999, assume that the program in Figure 6.2 is in the internal memory and give the command:

```
RENUM 5000,1
```

You will then obtain the program in Figure 6.3.

```
5000 PROC TABLE(REF VECTOR(),NOFEL)CLOSED
5001   //5 NOVEMBER 1982 VERSION 1.1
5002   //BY BILL COMPSON
5003   //
5004   ZONE 12
5005   SUM:=0
5006   FOR I:=1 TO NOFEL DO
5007     SUM:=SUM+VECTOR(I)
5008   ENDFOR I
5009   FOR I:=1 TO NOFEL DO
```

```
5010      PERCENTAGE:=VECTOR(I)/SUM*100
5011      PRINT I,VECTOR(I),PERCENTAGE
5012    ENDFOR I
5013    PRINT "TOTAL",SUM
5014 ENDPROC TABLE
5015 //=======MAIN PROGRAM=======
5016 PRINT "***TESTPROGRAM***"
5017 INPUT "WHICH DIMENSION?": D
5018 DIM TESTVECTOR(D)
5019 FOR I:=1 TO D DO
5020    TESTVECTOR(I):=RND(100,900)
5021 ENDFOR I
5022 EXEC TABLE(TESTVECTOR,D)
```

Figure 6.3

Now give the commands:

```
LIST 5000-5014 "TABLE"
DEL 5000-5014
RENUM 5000,1
LIST
```

The following will then appear on the screen:

```
5000 //=======MAIN PROGRAM======
5001 PRINT "***TESTPROGRAM***"
5002 INPUT "WHICH DIMENSION?": D
5003 DIM TESTVECTOR(D)
5004 FOR I:=1 TO D DO
5005    TESTVECTOR(I):=RND(100,900)
5006 ENDFOR I
5007 EXEC TABLE(TESTVECTOR,D)
```

Now list this into the file "MAINTABLE".

Exercise 6.8

Examine the effect of the command ENTER by executing the following commands, and then describe its effect in your own words.

(a)
```
NEW
ENTER "TABLE"
LIST
ENTER "MAINTABLE"
LIST
```

(b)
```
NEW
ENTER "MAINTABLE"
LIST
ENTER "TABLE"
LIST
```

(c)
```
NEW
ENTER "TABLE"
RENUM
LIST
ENTER "MAINTABLE"
LIST
```

(d)
```
NEW
ENTER "MAINTABLE"
RENUM 9000
LIST
ENTER "TABLE"
LIST
```

Sometimes it is very informative to draw a histogram, for example:

```
 1 :******************************** 31
 2 :******************************** 32
 3 :****************************************** 40
 4 :********************************** 34
 5 :****************************************** 40
 6 :************************************* 38
 7 :****************************************** 39
 8 :******************************** 32
 9 :************************************* 38
10 :*********************************** 37
11 :********************************** 35
12 :*********************************** 37
```

Exercise 6.9*

Write the code for a procedure

```
PROC HISTO(REFVECTOR(),NOFEL)CLOSED
```

that draws such a horizontal histogram.

Exercise 6.10*

Test the code from the above exercise using a small test program that you have constructed yourself.

Exercise 6.11

Once you are satisfied that the procedure in Exercise 6.9 works, store the procedure in a file named HISTOH (the last H stands for horizontal), in the same way that you stored the TABLE procedure.

Exercise 6.12

Construct a procedure that computes the largest value of an array.

The procedure HISTO you defined earlier isn't always very good: for example, what happens if the value is 93 and there should be 93 stars?

Exercise 6.13

Try to improve HISTO in such a way that even 93 stars could be shown in some reasonable way.

Sometimes it's preferable to use vertical histograms like:

```
*
*  *     *
*  *  *  *  *
*  *  *  *  *  *
1  2  3  4  5  6
```

Exercise 6.14*

Try to construct a reasonably detailed structure diagram for the above procedure. When you draw structure diagrams for procedures always start with an uppermost box like this:

PROC HISTO(REF VECTOR(),NOFEL)

The advantage of a histogram of this kind is that it can be drawn to scale: by using a scale marked along the side of the histogram the values can be as large as desired.

Exercise 6.15*

Amend the structure diagram to include a scale up the side of the histogram and code the procedure. After testing the program, list it into a file called HISTOV (standing for vertical histogram).

Exercise 6.16

This procedure gives quite a strange result when NOFEL is too large. Try to explain why. Can you correct it in some way?

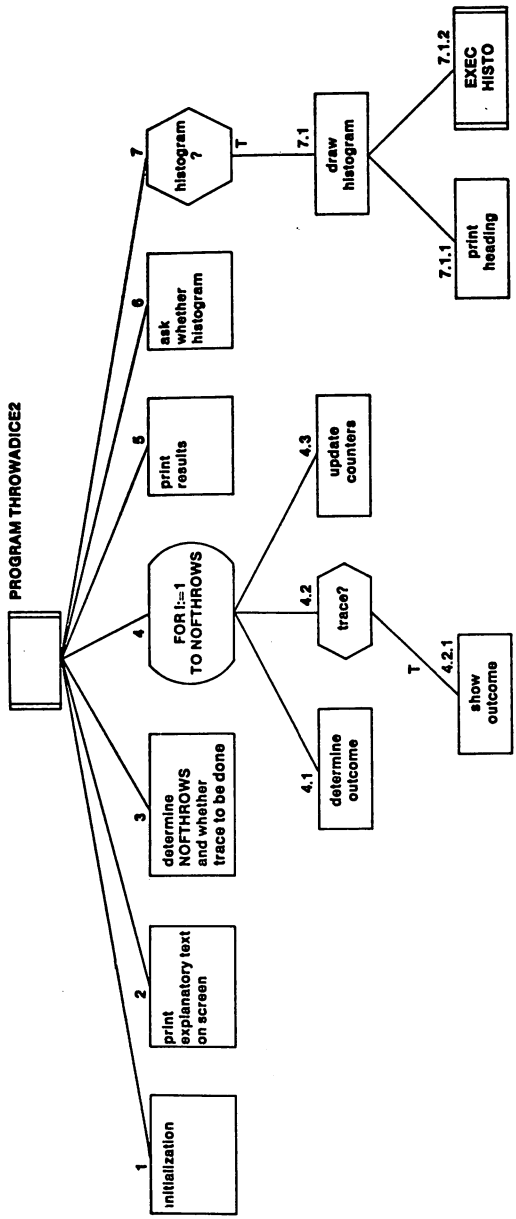

PROGRAM THROWADICE2
1
initialization
2
print explanatory text on screen
3
determine NOFTHROWS and whether trace to be done
4
FOR I:=1 TO NOFTHROWS
4.1
determine outcome
4.2
trace?
T
4.2.1
show outcome
4.3
update counters
5
print results
6
ask whether histogram
7
histogram ?
T
7.1
draw histogram
7.1.1
print heading
7.1.2
EXEC HISTO

Figure 6.4

Exercise 6.17

The program in this exercise completes program THROWADICE detailed in Figure 5.5 If you haven't already done so, list the procedures SHOW, HISTO and TABLE into your secondary memory.

Now break down the coarse structure diagrams in Figures 6.4 and 6.5 into greater detail, using the three procedures above. Then code the whole program.

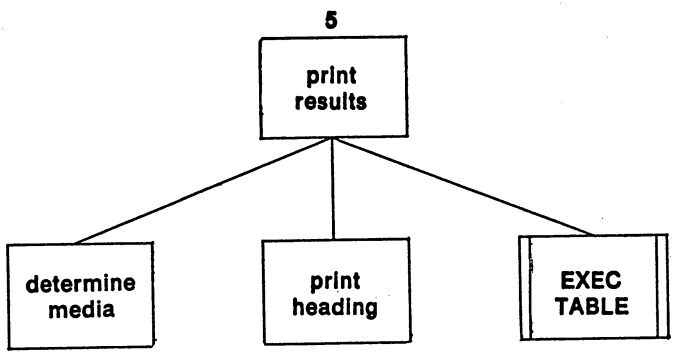


Figure 6.5

6.2 SORTING

A common problem is the sorting of numbers or strings (numbers into ascending or descending order; strings into alphabetical order). A very simple example might look like this:

```
HOW MANY NUMBERS? 5
NUMBER 1? 113
NUMBER 2? 82
NUMBER 3? 61
NUMBER 4? 47
NUMBER 5? 98
SORTED IN ORDER
47 61 82 98 113
```

In order to manipulate these five numbers they are placed in the array A, i.e.:

A(1)	113
A(2)	82
A(3)	61
A(4)	47
A(5)	98

These numbers are to be sorted, i.e. the smallest in A(1), the second smallest in A(2) and so on. *One* method of doing this is to find out in which box the largest of the numbers A(1), A(2), A(3), A(4) and A(5) is situated (in this case it is box number 1). Exchange the contents of A(1) and A(5): now A(5) is correct.

Now find out in which box the largest of the numbers A(1), A(2), A(3) and A(4) is situated (in this case it is box number 1). Exchange the contents of A(1) and A(4): now both A(4) and A(5) are correct. Continue this process until all boxes are correct.

Exercise 6.18*

Write a closed procedure

```
SORT(REF VECTOR(),NOFEL)
```

according to this strategy. Start by drawing a structure diagram.

Exercise 6.19*

Test the above procedure with a program that you have written yourself.

Hint 1: If you use the lines:

```
INPUT "HOW MANY NUMBERS?":NOFNUMBERS,
```

followed by a declaration

```
DIM A(NOFNUMBERS)
```

you can determine the size of the array each time you run the program (compare the code in figure 6.1 on page 93). Then list the procedure into the file named SORTM.

Hint 2: Define and use in this program a procedure PROUT(REF VECTOR(), NOFEL), that prints an array. Use it before and after the sorting program.

After the test, list the procedure into the file named SORTM.

Exercise 6.20*

Using the ideas behing this procedure draw a structure diagram for the following problem:

```
HOW MANY PUPILS? 30
FOR EACH PUPIL ENTER NAME AND WEIGHT
PUPIL NR 1:
NAME? COMPSON BILL
WEIGHT? 87.5
```

and so on, until

```
PUPIL NO 30:
NAME? WATERMAN FRED
WEIGHT? 86.3
```

As a result there will be two lists, one of pupils sorted according to weight, the other of pupils' names in alphabetical order.

To Analyze a Test[3]

In a test there are 12 questions. For each question there are five alternative answers, of which only one is correct. The questions are numbered 1,2,3,...,12. The alternatives are numbers 1,2,3,4,5.

Exercise 6.21*

Construct a program, using the structure diagram in Figure 6.6 as a guide, that via a terminal enters the answers to the questions for all the 30 pupils and then prints out a list of all the pupils, sorted according to their total scores, with the highest score first. (The correct answer gives one point, i.e. the maximum score is 12.)

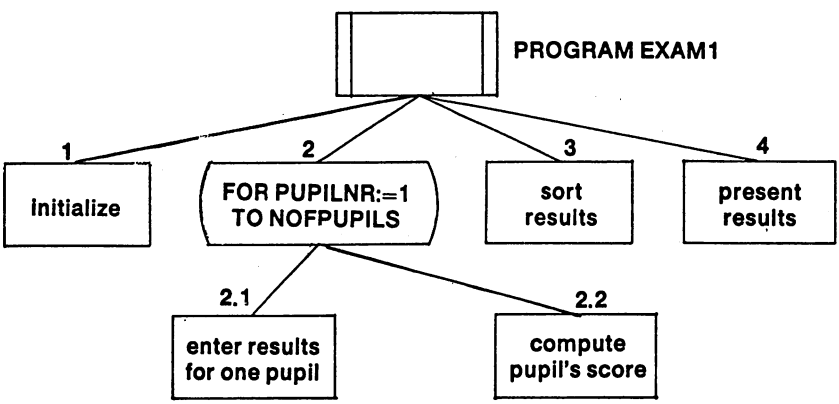


Figure 6.6

3. The section "To Analyze a test" is in some ways more advanced than the rest of the book. You do not need to do this section, in order to do what follows after it.

Initialize means for example, to provide the program with the correct answers to the questions, and a statement such as NOFPUPILS:=30.

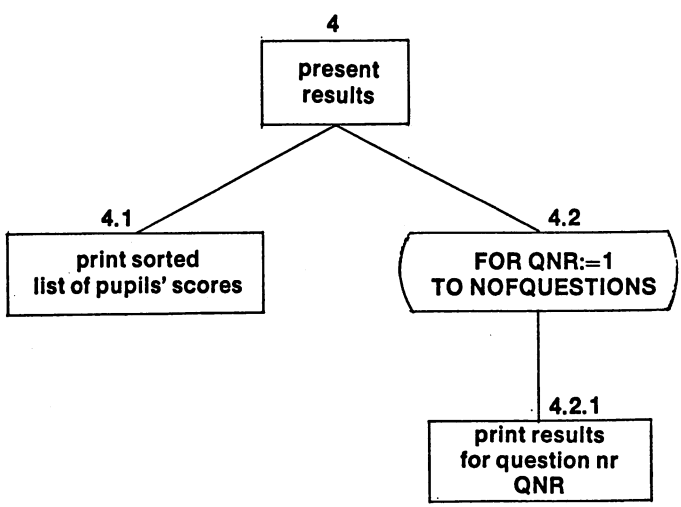
Extension: suppose that the person who organizes the test wants statistical information about each question, for example a histogram. Then the entered results must be stored in a *two-dimensional array* (which is denoted ANSWER). (See Figure 6.7.)

```
Question 1 2 3 4 5 6 7 8 9 10 11 12
Pupil
1
2
3                          x <-ANSWER(3,9)
.
.
.
30
```

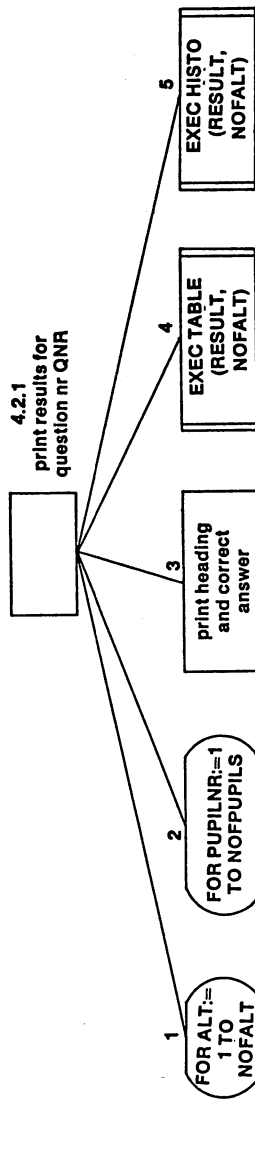
Figure 6.7 The answer from pupil number 3 to question number 9

Exercise 6.22*

With this extension, refine box 2.1 (Figure 6.6) down to code level. With the same structure diagram box 4 becomes:



In this case NOFALT=5 and RESULT is a vector with NOFALT elements.



Note: **that in this case NOFALT=5. RESULT is a vector with NOFALT elements.**

Figure 6.8

Exercise 6.23

Which parts of the structure diagrams do you think need further refinement?

Exercise 6.24

Make the amendments you consider desirable and, after that, code the program.

Exercise 6.25

Test the program for NOFPUPILS=4 and NOFQUESTIONS=6. *Note:* At the beginning of the code you must initialize:

```
NOFQUESTIONS:=12 (originally)
NOFPUPILS:=30    (originally)
```

And you must declare the array ANSWER by:

```
DIM ANSWER(NOFPUPILS,NOFQUESTIONS).
```

6.3 PROJECT: MINI-MARKET

A project is taken here to mean a larger exercise that usually can have several differing solutions.
Suppose that a (very small) firm sells a small number of different products. The products can be described using two arrays:

NAME$ (which contains the names of the products)
PRICE (which contains the prices of the products).

NAME$(7)	HAMMER
PRICE(7)	5.80

This means that the seventh product is a hammer, and that it costs $5.80.

To use these vectors they must first be declared, for instance in the following way:

```
NOFPRODUCTS:=30
DIM PRICE(NOFPRODUCTS)
DIM NAME$(NOFPRODUCTS) OF 20
```

The last line declares the maximum number of characters in boxes NAME$(1), NAME$(2), and so on.

A statement like:

```
PRICE(7):=5.90
```

changes the price of a hammer to $5.90.

This project entails devising a "mini-market" with 10 products. Use the variable NOFPRODUCTS, and initialize it by

```
NOFPRODUCTS:=10
```

Start with a loop:[4]

```
FOR I:=1 TO NOFPRODUCTS DO
  PRINT "PRODUCT NR",I
  INPUT "NAME? ":NAME$(I)
  INPUT "PRICE? ":PRICE(I)
NEXT I
```

What is the price of a screwdriver? Is there any such product? To find out, construct a procedure:

```
PROC SEARCH(Q$, REF WHICH)
```

so that WHICH becomes equal to 0 if there is no product named Q$ and otherwise it becomes equal to the number of the relevant product.

Example:

```
EXEC SEARCH("HAMMER",RESULT)
```

should produce the result

```
RESULT:=7.
```

First try:

```
FOR I:=1 TO NOFPRODUCTS DO
  IF NAME$(1)=Q$ THEN WHICH:=I
NEXT I
```

Exercise 6.26*

There is an error in the above example. Correct it. *Hint:* What happens if there is no such product?

One disadvantage with the corrected code is that if, for instance, the correct value of WHICH is 1, then many lines are unnecessarily executed. How could you avoid that?

Second try (see Figure 6.9):

```
WHICH:=0
FOR I:=1 TO NOFPRODUCTS DO
```

4. When you have finished the next chapter you'll know how to store information like this in a file.

```
    IF NAME$(I)=Q$ THEN
      WHICH:=I
      GOTO DONE
    ENDIF
  NEXT I
  DONE:  //NO UNNECESSARY EXECUTIONS!
```

DONE is an example of a *label*. The same naming rules apply for both labels and variables.

Third try (this doesn't work on all COMAL versions)

```
WHICH:=0
FOR I:=1 TO NOFPRODUCTS DO
  IF NAME$(I)=Q$ THEN
    WHICH:=I
    I:=NOFPRODUCTS   //ENSURES NO MORE LOOPS DONE!
  ENDIF
NEXT I
```

Rules for the use of GOTO

1. The only permitted construction is GOTO label.
2. Try to use GOTO only as a last resort – never by routine. It should only be used when the usual control structures

   ```
   FOR . . . NEXT
   REPEAT . . . UNTIL
   WHILE . . . ENDWHILE
   ```

   give too cumbersome solutions.
3. Only use GOTO to jump out of a loop forwards – *never* jump backwards.

Exercise 6.27

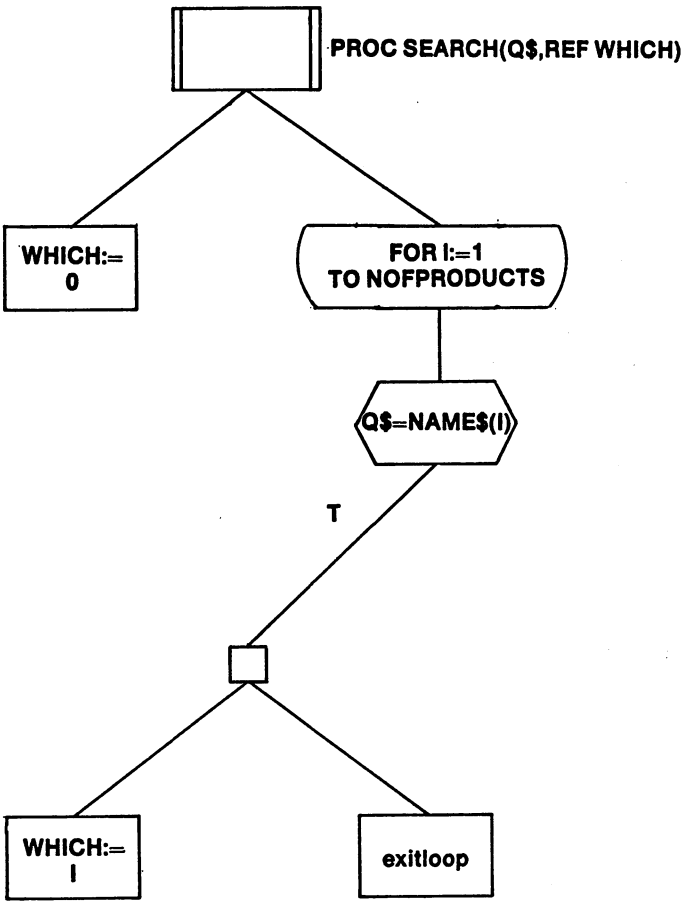
Construct a procedure PRICELIST, that prints the current price list.

Exercise 6.28*

Construct a procedure

```
FINDPRICE(Q$,REF WHICHPRICE)
```

so that if there is an article named Q$ then WHICHPRICE is equal to its price, otherwise WHICHPRICE is equal to 0. *Hint:* Use the procedure SEARCH defined earlier.



Note: **exitloop means that the lowest loop will be left. This can be implemented by the GOTO label.**

Figure 6.9 *Structure Diagram for the Second Try*

Exercise 6.29

Construct a procedure

WHATTOPAY(INAMOUNT,CUSTTYPE$,REF TOPAY)

that computes how much the different types of customers must actually pay. Use the information in Question 3.3 (page 44) as a basis.

Exercise 6.30
Construct a procedure BILL that produces a bill for instance in this way:

```
CUSTOMER? BILL COMPSON
CUSTOMER CATEGORY? ORDINARY
TO FINISH ENTER STOP FOR ARTICLE NAME
ARTICLE NAME? CHAIR
!!! NO SUCH ARTICLE !!!
ARTICLE NAME? SCREWDRIVER
HOW MANY? 5
```

and so on. Using this information and the procedure WHATTOPAY (see Exercise 6.29), print a bill on paper.

Now use these procedures to write a program that assists the owner of the mini-market. The program should center around a "menu" like this:

```
MINI-MARKET MENU
================
(1) CHANGE PRICES
(2) LOOKUP PRICES
(3) PRINT PRICELIST(ON SCREEN OR PAPER)
(4) PRINT A BILL
(5) PRINT (ON SCREEN OR PAPER) ACCUMULATED SALES FOR TODAY
(6) END
CHOOSE ALTERNATIVE
```

Note: it is assumed that the computer is switched on all day long and that alternative (6) is chosen when the day ends. Alternative (5) should contain information about the number of bills, accumulated sales, in total and per article. To find out how many of each article have been sold use the vector SOLD-NUMBER. In both (1) and (2) the information to be input is the name of products.

Example:
Alternative (1) *could* look like this:

```
CHANGE PRICES
==============
ENTER NAME OF PRODUCT. ENTER STOP WHEN READY
ARTICLE NAME? CHAIR
!!! NO SUCH ARTICLE !!!
ARTICLE NAME? SCREWDRIVER
CURRENT PRICE IS 3.25
NEW PRICE (ENTER NUMBER <=0 FOR NO CHANGE)? 3.35
ARTICLE NAME? STOP

===BACK TO MAIN MENU===
```

The structure of the program can be described in this way:

```
initialization
REPEAT
  print menu
  input ALTERNATIVE
  CASE ALTERNATIVE OF
  WHEN 1
    erase screen
    print headline and first explanation
    REPEAT
      change one price
    UNTIL no more prices to change
  WHEN 2
    erase screen
    print headline and first explanation
    REPEAT
      look up one price
    UNTIL no more prices to look up
  WHEN 3
    input WHERE$
    EXEC PRICELIST(WHERE$)
  WHEN 4
    EXEC BILL
  WHEN 5
    input WHERE$
    EXEC STATISTICS(WHERE$)
  WHEN 6
    ask whether statistics on paper are desired
    IF yes EXEC STATISTICS("P")
  ENDCASE
UNTIL ALTERNATIVE=6
```

This way of describing what a program will do is fully equivalent to using structure diagrams: it is called *pseudocode* (note the indentation used in the pseudocode).

Exercise 6.31
Transform the pseudocode from the last page into a structure diagram.

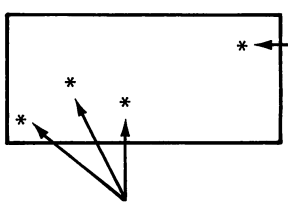
Exercise 6.32
Transform three or four of your earlier structure diagrams into pseudocode.

Exercise 6.33
Now break down your structure diagrams or pseudocode so that the coding becomes straightforward, then perform the coding and execute the program.

Exercise 6.34
Try to split this large problem between a group of people, so that each person solves one part of it, and then "patching" it together. Use either method 1 with MERGE or method 2 with ENTER (compare p. 94–97).

This project illustrates the problems you will come across in ordinary life when trying to solve more complex problems using a computer. For example, you must break down your initial problem into manageable parts which can then be solved using procedures.

In BASIC a program that accomplished the same as our last program would be exceedingly difficult to construct, to read and to modify. Let's illustrate the last point.



In BASIC a program too often looks like a cobweb.

Changing one part of a program

may modify it elsewhere
(with or without your knowing).

Using procedures, however, it is much easier to isolate the changes that always need to be done.

What is programming in reality? First of all you need to know *what* the problem is that you have to solve (and this is often the hardest part!). Second, you need to design a solution strategy, either in the form of reasonably detailed structure diagrams or in reasonably detailed pseudocode. After this the coding – i.e. the writing of the program lines – should be more or less routine.

Drawing structure diagrams forces you to divide your problem (often several times) into smaller, more manageable problems using procedures. Doing this requires insight and hard work: it is a measure of a programmer's creativity and experience.

6.4 *External procedures*

In many modern systems, like Commodore's version 2.00, a closed procedure without IMPORT statements that is SAVED to disk may be used as an *external procedure*.

For example: In the program, in Figure 6.2 (page 94) delete lines 16-230 and SAVE the rest, i.e. the very procedure itself, for instance by the command SAVE "1:XPROC". (Or something similar depending on your system.) Then the program

```
 10 PROC TABLE (REF VECTOR(),NOFEL) EXTERNAL "1:XPROC"
160 //=======MAIN PROGRAM=======
```

and so on (lines 160-230 exactly as in Figure 6.2) will accomplish exactly the same as the program in Figure 6.2.

Exercise 6.35

Solve exercise 6.17 to making SHOW, HISTO and TABLE external procedures saved in three different files.

Exercise 6.36

Could external procedures be used in your solution for the MINI-MARKET project?

Try to use file names starting with an X for external procedures. Usually you can have exactly *one* external procedure in one file.

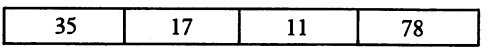
If your system doesn't have this facility simply cross over this section!

7 Using Sequential Files

7.1 SEQUENTIAL FILES

Instead of entering numbers or strings from your keyboard (using INPUT), a program can obtain them by reading a *file*. There are two types of files in COMAL: *sequential* files and *random* files. Random files are discussed in Section 8.2.

With a sequential file the program obtains ("reads") the strings and numbers in order. A simple sequential file that only contains four numbers can be illustrated like this:



the file
starts here

Suppose that this file is called SEQDEMO. The program in Figure 7.1 demonstrates how to read these numbers.

```
 10 // PROGRAM READFROMSEQDEMO
 20 // 1 NOVEMBER 1982 VERSION 1.0
 30 // BY BILL COMPSON
 40 // DEMONSTRATES HOW TO READ FROM A SEQUENTIAL FILE
 50 OPEN FILE 3, "SEQDEMO",READ
 60 SUM:=0
 70 REPEAT
 80    READ FILE 3: NUMBER
 90    PRINT NUMBER
100    SUM:=SUM+NUMBER
110 UNTIL EOF(3)
120 CLOSE FILE 3
130 PRINT
140 PRINT "THE SUM OF THE NUMBERS IN SEQDEMO=",SUM
```

Figure 7.1

The number 3 in lines 50, 80, 110 and 120 is called a *channel number*. The question of which values you may use for channel numbers depends on which system, you are using: if in your system 4 is a permitted channel number you can use the program in Figure 7.1 by simply replacing number 3 by number 4 in lines 50, 80, 110 and 120. Don't forget to follow the file-naming conventions in your system. (See the system note in Chapter 1: page 7.)

(*Note:* In a few versions, for instance the RC version, EOF is set to true first when an attempt is made to READ *after* the last element. With such a system the program READFROMSEQDEMO above prints out the last element *twice*; and the sum is not correct because the last element is added to the sum twice. However, if you enter the lines

```
 85 IF NOT EOF(3) THEN
105 ENDIF
```

the program will work for such versions of COMAL.)

SYSTEM NOTE

Permitted channel numbers are .....
When the end-of-file is reached EOF for that file is set to TRUE. / When an attempt is made to READ after the last element EOF is set to TRUE. (Delete inapplicable alternative.)

Note that before a file can be used it must be opened: this is done in line 50. When you have finished using a file you must close it: see line 120 – the command CLOSE closes all files now open.

Note: in some versions you may just need to write, for example, CLOSE 7 instead of CLOSE FILE 7, and OPEN 3, instead of OPEN FILE 3. Try to use the longer versions, because they make your program more readable.

Exercise 7.1*

If the file SEQDEMO does not contain any number at all, the program READSEQDEMO gives an error message. Why? Suggest how to correct this.

How could you create a sequential file? The program in Figure 7.2 illustrates one possible way.

```
10 // PROGRAM CREATESEQDEMO
20 // NOVEMBER 1982 VERSION 1.0
30 // BY BILL COMPSON
40 // DEMONSTRATES HOW TO CREATE A SEQUENTIAL FILE
50 OPEN FILE 3, "SEQDEMO",WRITE
```

```
 60 PRINT "THIS PROGRAM ENTERS NUMBERS INTO A"
 70 PRINT "SEQUENTIAL FILE NAMED SEQDEMO"
 80 REPEAT
 90   INPUT "HOW MANY NUMBERS? (0 TO FINISH)": NOFNUMBERS
100   FOR I:=1 TO NOFNUMBERS DO
110     INPUT "A NUMBER,PLEASE!": NUMBER
120     WRITE FILE 3: NUMBER
130   NEXT I
140 UNTIL NOFNUMBERS=0
150 CLOSE FILE 3
```

Figure 7.2

Exercise 7.2

Execute the program in Figure 7.2 twice and enter differing sequences. Examine the contents after each execution (by using READFROMSEQDEMO in Figure 7.1). What happens if the file SEQDEMO already exists when you execute the program in Figure 7.2?

Exercise 7.3

Perhaps you could insert a line (for instance number 45) that deletes the previous file SEQDEMO (if there is such a file). Try this and see if it works.

Suppose that you would like to extend SEQDEMO, for instance with the numbers 56 and 37, to obtain:

35	17	11	78	56	37

You can do this in the way described in Figure 7.3.

```
 10 // PROGRAM APPENDTOSEQDEMO
 20 // 10 NOVEMBER 1982 VERSION 1.0
 30 // BY BILL COMPSON
 40 // DEMONSTRATES HOW TO APPEND TO A SEQUENTIAL FILE
 50 OPEN FILE 3, "SEQDEMO",APPEND
 60 PRINT "THIS PROGRAM ADDS NUMBERS TO A"
 70 PRINT "SEQUENTIAL FILE NAMED SEQDEMO"
 80 REPEAT
 90   INPUT "HOW MANY NUMBERS? (0 TO FINISH)": NOFNUMBERS
100   FOR I:=1 TO NOFNUMBERS DO
110     INPUT "A NUMBER,PLEASE!": NUMBER
120     WRITE FILE 3: NUMBER
130   NEXT I
140 UNTIL NOFNUMBERS=0
150 CLOSE FILE 3
```

Figure 7.3

Exercise 7.4

You can open a sequential file in three different ways: what are they, and what do they mean?

Exercise 7.5*

Standardize these last three programs (Figure 7.1, 7.2 and 7.3) in such a way that they are applicable for all sequential files that only contain numbers. *Hint:* Enter the value of FILENAME$ in some way, and use constructions like:

```
OPEN FILE 4,FILENAME$,APPEND
```

Exercise 7.6*

Write a procedure:

```
XCOPY(F$,T$)[1]
```

That copies the content of the file named F$ (which only contains numbers) into a file named T$. Start by first deleting the previous file T$ (if any).

Exercise 7.7

In the old days, when internal memory was very expensive, you had to use a lot of tricks to sort sequential files. Today (when the file isn't very large), you can sort a file this way (assuming that the file only contains numbers):

(1) count the number of numbers in the file;
(2) reserve space in the internal memory by a DIM-statement;
(3) read the file into the array;
(4) sort the array using a procedure discussed in Section 6.2;
(5) delete the old file;
(6) write the array into the file.

Write a procedure SORTFILE that sorts files in this way. Start by drawing structure diagrams.

Exercise 7.8

It is useful to automatically obtain a back-up copy of your sequential file before trying to sort it. Suggest how this could be included in the procedure. *Hint:* if for example the filename is NICE, save a copy called NICEX.

---

1. In many systems there exists a procedure or command called COPY that copies files. This is why this procedure is denoted XCOPY and not simply COPY.

In many applications you enter values from time to time. In Exercise 4.31 values of gender and weight were entered via input, but all values were entered at the same execution.

Exercise 7.9*

Write a program that makes it possible to enter, at arbitrary times, the values of gender and weight of one or several people, and also to produce the same statistics as mentioned in Exercise 4.31.

Exercise 7.10

Construct a program that splits the file WEIGHT into two different files, one named WEIGHTM for male values and another named WEIGHTF for female values.

Exercise 7.11*

It would be very convenient to have a closed procedure ANALYZE(F$) to analyze sequential files that only contain numbers. Write such a program which for instance produces the following information:

```
INFORMATION ABOUT THE FILE NAMED WEIGHTM
NUMBER OF NUMBERS = 50
MEAN VALUE = 78.3
LEAST VALUE = 65.7
LARGEST VALUE = 82.5
```

7.2 THE TRIANGLE COMPUTER CLUB

This club uses a sequential file called MEMBERS for its members. (In this club all members pay the same fee.)

...	347 BILL COMPSON	348 DICK WATERMAN	350 JAMES F LONGWOOD	...

To enter the members' names use code similar to that in the program APPENDTOSEQDEMO (Figure 7.3). The main difference is that instead of

```
WRITE FILE 3 : NUMBER
```

you must have something like

```
WRITE FILE 3 : MEMBERNR,MEMBERNAME$
```

Exercise 7.12*

Write a program that can append new members to the file MEMBERS. Start by entering the number of the first new member to append. (Later you will make acquaintance with a smart way to store this number: see Exercise 8.3, p. 134).

Exercise 7.13

Make a program that lists the members on paper:

```
MEMBER NUMBER          MEMBER NAME
============          ==========
15                     JOHN JOHNSON
17                     DICK SWANSON
.
.
.
```

If, instead of printing the members' names on paper, the members are listed on the screen there will be some problems: for instance the first members will "disappear" from the screen too quickly. Perhaps it would be preferable to list the members 10 at a time, like this:

```
MEMBER NUMBER          MEMBER NAME
============          ==========
xxx                    xxx xxxxxxxxx
xxx                    xxxxxx xxxxxxxxxxx
.
.
.
xxx                    xxxx xxxxxxxxxx
PRESS RETURN FOR NEXT 10 MEMBERS.
```

Suppose that you want to always use lines 3-12 on the screen for the members. You can arrange this by using the command CURSOR or PRINT AT.

With Commodore computers, CURSOR 5, 7 moves the cursor to *line* 5, *position* 7. Instead of the two commands:

```
CURSOR 5,7
PRINT SUM
```

you can when using Commodore computers, have one command:

```
PRINT AT 5,7 : SUM
```

In some other versions CURSOR 5,7 moves the cursor to *line* 7, *position* 5; and in some others you just have PRINT AT and no cursor command.

SYSTEM NOTE

(for CURSOR / PRINT AT) .....

.....

.....

.....

Warning: this type of command(s) is highly system-dependent. In the following examples we will follow the Commodore rules. Remember to change the code so that it suits *your* system.

The purpose of the following program is to print on line 10 of the screen first:

```
11111111
```

and after that:

```
222
```

```
10 CURSOR 10,1
20 PRINT "11111111"
30 CURSOR 10,1
40 PRINT "222"
```

The end result is:

```
22211111.
```

If you want to obtain only:

```
222
```

you must first erase the old line. You could erase the whole screen of course, but if that is too much then you could use the ERASELINE procedure given below to erase one line.

The procedure:

```
PROC ERASELINE(LINENR) CLOSED
  STARTPOS:=1
  LASTPOS:=80
  // STANDARD SCREEN - CHANGE IF NECESSARY
  CURSOR LINENR,STARTPOS
  FOR P:=STARTPOS TO LASTPOS DO
    PRINT " ",
    NEXT P
  CURSOR LINENR,STARTPOS
ENDPROC ERASELINE
```

erases the whole line with linenr LINENR, and places the cursor at the beginning of that line.

Exercise 7.14
What is the command to erase the whole screen in your system?

Exercise 7.15*
Write a program that lists the members on the screen in the way indicated previously.

7.2.1 Project Triangle Computer Club (1)

Construct a program that helps the cashier of a club keep the membership records. Perhaps the program should center around a menu like this one (see Figure 7.4):

```
TRIANGLE COMPUTER CLUB

(1) ADD NEW MEMBERS
(2) LIST MEMBERS
(3) PRINT MEMBERSHIP CARDS
(4) END THIS SESSION
WHICH ALTERNATIVE?
```

Note: a project must always be documented with reasonably detailed structure diagrams. When these are ready, start the coding. Try to divide the work amongst several people (or if you work alone at least pretend to) and in the end "collate" the complete program using ENTER/MERGE and/or other commands. If your system has the facility for external procedures, try to use them as often as possible.

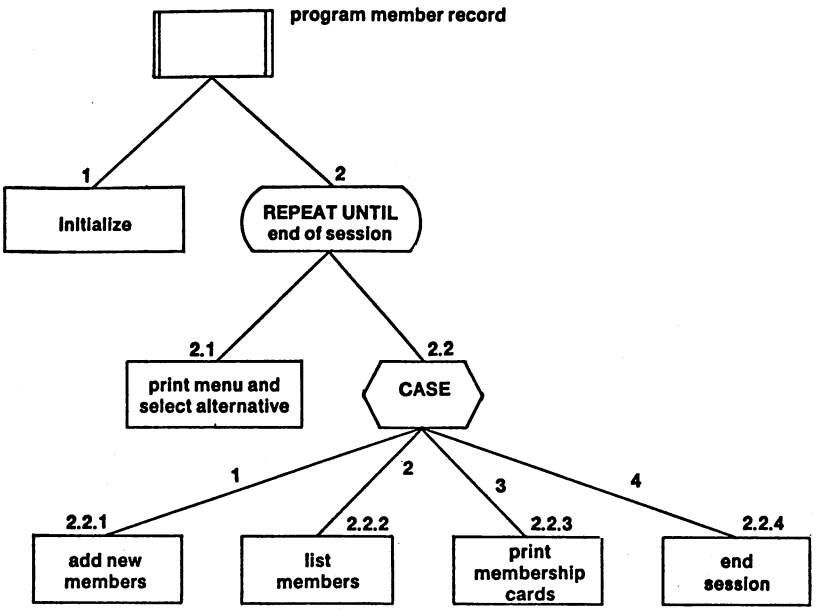


Figure 7.4

When you enter a new member there are three things to do (see Figure 7.5):

(a) give him/her a number;
(b) print a membership card; and
(c) append number and name to the file MEMBERS.

For inexperienced users it is often useful to have forms to fill in.

```
==============================
    THE TRIANGLE COMPUTER CLUB
    ++++++++++++++++++++++
    MEMBERSHIP CARD FOR □
    1982     5 DOLLARS     NO 546
    .....
    (USER'S NAME)
    ==============================
```

The cursor indicates where the next character will appear on the screen: i.e.

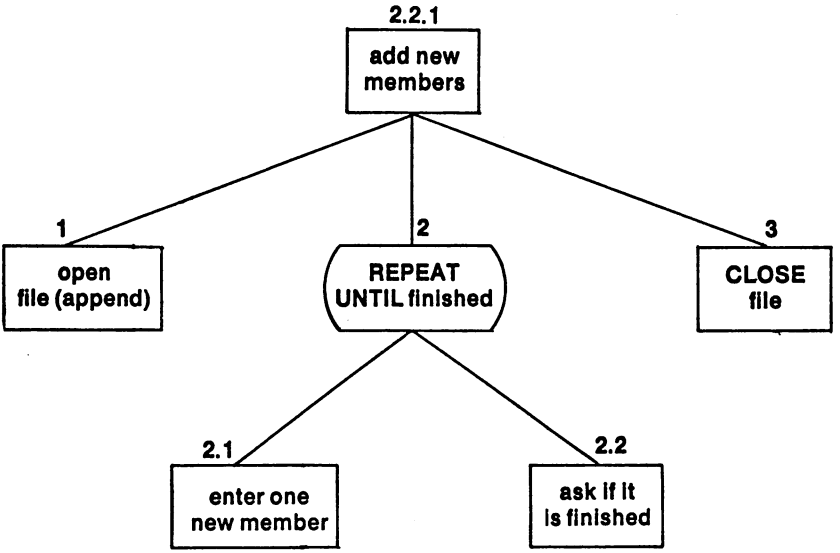


Figure 7.5(a)

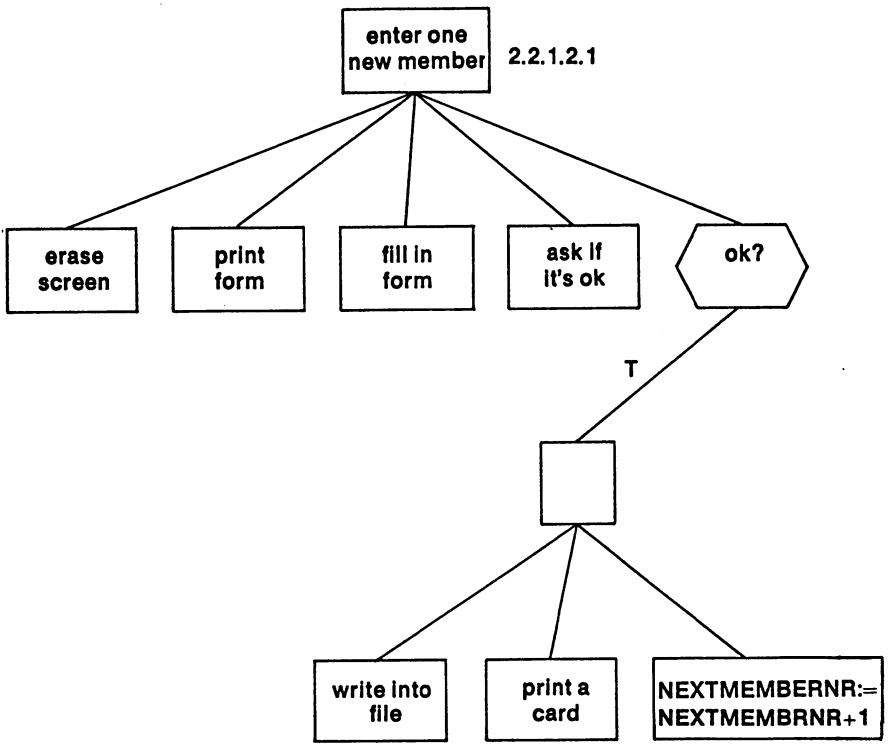


Figure 7.5(b)

where the user enters the new member's name, before pressing the RETURN key.

When this is done there are a few questions to the user. The questions are written in the lower half of the screen. For instance:

```
OK ( Y(ES) OR N(O) )? Y
ANOTHER MEMBER ( Y(ES) OR N(O))? N
```

Note that this is part of entering new members. Each year new membership cards are printed in December (alternative 3 on the menu).

When storing information (often in large amounts) there are two important factors to bear in mind: the first is that the user must be able to get the information he/she wants. The second is to ensure that the information is correct and up-to-date. However, in this case – as the file is quite small – it can be kept up-to-date by "semi-manual" means:

On the screen members' names are printed 10 at a time, but instead of printing

```
PRESS THE RETURN KEY TO CONTINUE
```

at the bottom of the screen, you have

```
WHICH ALTERNATIVE, C(ONTINUE) D(ELETE) or M(ODIFY)?
```

If the user chooses alternative C the next 10 members are shown. If the user chooses alternative D, the system asks for the line number, and then erases this member on the screen. After that the question is posed again. The command M should in this case be quite rare, and should only occur when a member changes his/her name in some way.

The following list tells you how to go about updating MEMBERS:

(i) take a safety copy of MEMBERS, in case something should go wrong (remember that in many systems there is a command for copying files);
(ii) copy MEMBERS into a file named GARB;
(iii) delete MEMBERS;
(iv) update GARB into a new file called MEMBERS;
(v) list the new version of MEMBERS on the screen;
(vi) if it is OK, delete GARB.

Some Useful String Commands

So far the string MEMBERNAME$ contains both the member's forename(s) and surname. How would you go about displaying the information on the screen like this:

MEMBER NR	SURNAME	FORENAME
========	=======	=========
.		
.		
.		
347	COMPSON	BILL
348	WATERMAN	DICK
350	LONGWOOD	JAMES F
.		
.		
.		

If NAME$="LARS FREDRIK SWANSON", then

NAME$(6:12)="FREDRIK" and

NAME$(14.20)="SWANSON".

The command NAME$ (1:4):="JOHN" alters NAME$ to

"JOHN FREDRIK SWANSON".

To scan a string character by character, use 1:1, 2:2, 3:3 etc., so that if

NAME$="BILL COMPSON"

NAME$ (1:1)="B";

NAME$ (2:2)="I"; and so on. In some versions NAME$(1:1) can be shortened to NAME$(1). However, this is not COMAL standard and can cause misunderstandings. NAME$ (I:J) is thus the substring consisting of character number I up to and including character number J.

A simple procedure to split a complete name like BILL COMPSON into surname and forename(s) is:

```
PROC SPLIT(NAME$ ,REF SURNAME$ ,REF FORENAME$) CLOSED
  // The object is to find where the last blank inside the name is.
  L:=LEN(NAME$)
  FOR I:=L TO 1 STEP-1 DO
    IF NAME$ (I:I)=" " THEN GOTO DONE
  NEXT I
  DONE:
  FORENAME$ :=NAME$ (1:I-1)
  SURNAME$ :=NAME$ (I+1,L)
ENDPROC SPLIT
```

Exercise 7.17*

This procedure can be coded in a much better way without GOTO, using REPEAT . . . UNTIL. Now try that.

Exercise 7.18

Using the procedure SPLIT, modify the list of members you displayed on the screen earlier.

Exercise 7.19*

In BASIC you have a confusing variety of different string commands.

(a) LEFT$ (A$,N) gives the leftmost N characters in A$. For example, LEFT$ ("BUFFALO",4)="BUFF". What would this command be in COMAL?

(b) MID$ (A$,7,3) gives three characters starting with the 7th. What would this be in COMAL?

(c) RIGHT$ (A$,N) gives the rightmost N characters in A$.

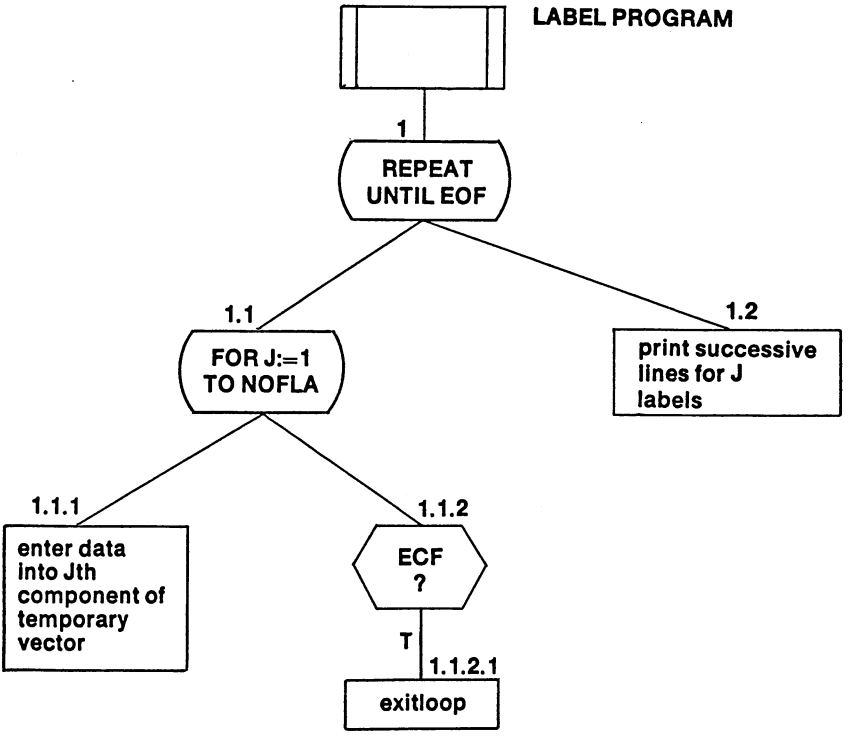
(d) L=LEN(A$) gives the length of the string A$. How could RIGHT$(A$,N) be expressed in COMAL using LEN?

7.2.2 Project: Triangle Computer Club(2)

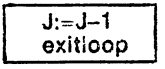
There is a record in the membership file of the member's address, and the last year each member paid his or her membership fee. Every year, in December, the membership fees are collected for the following year. Each member gets a computer-written letter requesting the member to pay next year's membership fee. The computer is used to produce address labels in a way that doesn't waste paper. When the fee is paid the member's number is entered into a file. At suitable intervals this file is emptied and the member file is updated. After a lapse of about two weeks, new requests are written to those who have not yet paid. Then, two weeks later, a final request is made. Later on, for instance in February, people who haven't paid the last three year's membership fees are excluded from the member file.

Construct a program (or programs) that helps the cashier of the club to keep the records in order. (Assume that new members do not need to pay before December.)

Hint: for the "label-printing" program:
if you have NOFLA labels beside each other you need some arrays of dimension NOFLA to store values of names and addresses etc.



(*Note:* If EOF is set to TRUE first when an attempt is made to READ beyond the file (for instance with RC), the last label will be printed twice. You can correct this by changing 1.1.2.1 to



7.3 PROJECT:CHR$[2]

We know that computers can only manipulate and store integers described in the computer's memory as sequences of 0s and 1s), yet in a lot of exercises the computer has obviously manipulated and stored strings. How is this possible? It's simply that internally the computer takes for instance A as the integer 65 and B as the integer 66. Every symbol entered via the keyboard (or in other ways) is translated into a number inside the computer. Conversely, when PRINT "A" is executed a pattern of 0s and 1s (65 in binary form = 1000001) makes an A appear for instance on the screen.

However, instead of PRINT "A" you can use the command PRINT CHR$(65).

2. This project uses two-dimensional arrays. You can skip it and go on with what follows if you do not wish to try using these.

SYSTEM NOTE

Some systems use only CHR (i.e. without the dollar sign). Find out what is used in your system .....

What is the point of this? One application is to obtain capital letters in a random fashion:

```
PRINT CHR$(RND(65,90))
```

produces one each of the letters A – Z. Another application is that for different values of P the command PRINT CHR$(P) can accomplish for instance a line feed, a clear screen or reverse field. (How this is done is dependent on your particular system. Look up your system manual for details.) A third application (in some systems) is where some values of P give interesting graphic patterns, called *semigraphic symbols*. For example:



Exercise 7.20

Try to find out, using suitable programs, the effect of PRINT CHR$(I) for different values of I, both on screen and on paper. *Warning:* sometimes you get different results on the screen and on the printer. For instance the printer may replace lower-case letters with semigraphic symbols. (This can usually be avoided by using a command such as POKE in Commodore systems.)

Exercise 7.21 (for systems with semigraphics)

Using some semigraphic symbols improves the procedures HISTOH and HISTOV. (These procedures are closed – thus you do not encounter problems when you try to modify them.)

In Chapter 5 you constructed programs that drew Christmas trees and other figures either on the screen or on paper. Suppose now that you want to be able to draw (on paper) figures like this:

```
=======
=     =
= +++ =
= +  + =
= +++ =
=======
 +   +
  + +
   +
```

To describe what is to be printed, use a two-dimensional array MARK$:

```
DIM MARK$(NOFROWS,NOFCOLUMNS) OF 1
```

(arbitrary) (fixed)

When starting a session and also later on you can enter a stored picture into MARK$. Such a picture is stored in a file whose two first items are the size of the picture, i.e. NOFROWS and NOFCOLUMNS. You can only enter a picture smaller than, or of the same size as, the dimensions you have specified.

At any time during this session you should be able to SHOW the whole or a suitable part of the present figure on screen and paper. It must also be possible to STORE the present figure. To erase parts of the figure must be simple to do. It must at least be possible to compose the picture with rectangles, squares, triangles and lines. The lines can point N, NE, E, SE, S, SW, W and NW. It must be possible to use semigraphic symbols (if they exist on your system).

Exercise 7.22*

Use these ideas to draw large numbers on paper, for example in such a way that 347.6 is printed:

```
XXX     X X     XXX             XXX
  X     X X       X               X
XXX     XXX       X             XXX
  X       X       X     XX      X X
XXX       X       X     XX      XXX
```

7.4 PROJECT: SIMPLE EDUCATIONAL PROGRAMS*

Using a file with the contents:

DENMARK COPENHAGEN SWEDEN STOCKHOLM NORWAY OSLO . . . ,

you could easily construct an educational program like this:

```
NAME THE CAPITAL OF THE FOLLOWING COUNTRIES
(ANSWER NOMORE TO STOP)
DENMARK? OSLO
SORRY THAT IS WRONG - TRY AGAIN
DENMARK? STOCKHOLM
SORRY - WRONG AGAIN! YOU ONLY HAVE 2 TRIES!
SWEDEN? STOCKHOLM
VERY GOOD - RIGHT FIRST TIME!
```

and so on. Use RND to get the questions in random order. It should not be permitted to have the same question twice in one session. Perhaps this dialog would be a bit better:

```
NAME THE CAPITAL OF THE FOLLOWING COUNTRIES
(ANSWER NOMORE TO STOP)
DENMARK? OSLO
SORRY - OSLO IS THE CAPITAL OF NORWAY!
DENMARK? NOTTINGHAM
SORRY - NO CAPITAL NAMED NOTTINGHAM AS FAR AS I KNOW!
```

and so on.

At the end of the session it should be possible to obtain some statistics on each student's answers.

Write a "skeleton" ("quizmaster") program where the basic coding is unchanged, but where new data can be entered, for the name of the file, the introductory question, and then the one-worded questions and answers. For example:

```
NAME THE GOALKEEPER OF THE FOLLOWING SOCCER TEAMS
(ANSWER NO MORE TO STOP)
ARSENAL?
```

and so on.

8 Data Statements, Random Files and Functions

8.1 DATA STATEMENTS

Frequently in traditional BASIC (and sometimes in COMAL-80) one uses DATA statements: these statements contain values which can be read into variables using READ statements.

Example:

```
.
.
.
100 FOR I:=1 TO 12 DO
110   READ MONTH$(I)
120 NEXT I
.
.
.
500 DATA "JAN","FEB","MAR"
510 DATA "APR","MAY","JUN","JUL","AUG","SEP"
520 DATA "OCT","NOV","DEC"
```

When the interpreter encounters a READ statement the next value(s) in DATA statements are read. The first READ statement starts with the first value(s). If the data in the DATA statements are too few an error message will be printed, and the execution will be aborted.

Just as when one is reading files, when executing DATA statements one can freely mix numbers and strings and read several variables in one statement. The only point to remember is that (just as for files) a string cannot be read into a variable without a dollar sign ($). If you want to read the DATA statements from the beginning of the program, you should give the command RESTORE.

DATA statements should be used only in two ways (if at all):

(1) During initialization, to get a shorter program – for instance lines 100-120 above.
(2) To "simulate" sequential files (which cannot be changed during an execution).

For example, instead of keeping the numbers 47, 56, 38, 94 and 35 in a sequential file keep them in DATA statement(s). However, you will not be able to change them via the program if you do this.

Even if you don't know how many numbers you have in your DATA statements you can, for instance, sum them. The corresponding problem for sequential files is solved using EOF, and the corresponding condition for DATA is denoted EOD (end of data).

Example:

```
100 // SUM:= sum of all numbers in DATA statements
110 SUM:=0
120 REPEAT
130   READ NUMBER
140   SUM:=SUM + NUMBER
150 UNTIL EOD
```

Exercise 8.1

Try and find out how you could put your knowledge of DATA and READ, from the exercises you have already done, to some good use.

8.2 RANDOM FILES

To read the 1000th item in a sequential file you must first read 999 items. This is obviously a waste of time, so it would often be useful to access a particular item directly, rather than searching through from the beginning of a file. With *random files* (also called *direct access files*) this is possible.

You can imagine such a file as consisting of boxes of the same size (but which can vary in size from file to file) numbered 1, 2, 3 . . . Each such box can be directly accessed, both for reading and writing. Just like sequential files, such files must be opened before they are used:

```
OPEN FILE 3, filename, RANDOM 30
```

3 denotes (just as for sequential files) the channel number. 30 is the size of each box measured in *bytes*. (One character needs exactly one *byte* of memory and usually a real number needs 5 bytes.)

Note: In some versions, notably that of RegneCentralen (RC), you must also explicitly create a random file the first time you use it, with the command CREATE.

SYSTEM NOTE

Rules (if any) when first using a random file are .....

.....

.....

A real number requires ..... bytes .....

To write into such a file use a command such as:

```
WRITE FILE 3,7,1 : NUMBER,NAME$
```

In this case, 3 is the channel number, 7 is the box number and 1 is starting position for writing. (The number of positions, i.e. the size of the box, can be found in the OPEN statement.)
The instructions to read, for instance, box 11 starting with position 17 would be:

```
READ FILE 3,11,17 : STREETADRESS$
```

When you have finished using a file like this it must be closed in the same way as a sequential file would be, i.e. use the command CLOSE FILE 3, or CLOSE to close all files now open.

Note that with a random file you may both WRITE and READ between the OPEN and CLOSE command. When you use a sequential file you open it for only one function: to read, write or append.

Example:
Suppose that the direct access file DADEMO contains boxes with a length of 20 bytes. The file is opened by:

```
OPEN FILE 5, "DADEMO",RANDOM 20
```

Note: Don't forget the naming conventions of your system.

Suppose that box 17 has these contents (where b indicates a blank):

```
FREDRICAbbERICSSONbb
```

If NAME$ is declared to contain 10 characters then the result of

```
READ FILE 5,17,11: NAME$
```

is

```
NAME$=ERICSSON
```

The result of

```
WRITE FILE 5,17,1:"bbbE"
```

is that the contents become

```
bbbERICAbbERICSSONbb
```

Exercise 8.2

Suggest how to change the contents of the above box to bbbERICbbbSWANSSONbb

Exercise 8.3*

Now you can complete The Triangle Computer Club Project (see p. 118) by storing the next member number to be used in box 1 in a random file called NXTMNR (that therefore only contains one box): try doing this.

8.2.1 Project: Simple-money Bank

The Simple-money Bank has only a few customers numbered 1, 2, 3 . . . Make a program that supports the following dialog on the screen:

```
!!! SIMPLE-MONEY BANK !!!
==================
D(EPOSIT), W(ITHDRAW) OR E(ND)? W
CUSTOMER NUMBER? 3
AMOUNT? 300
```

On the last line of the screen the following warning-text could appear:

```
! BALANCE IN ACCOUNT NUMBER 3 ONLY 234 !
```

After which the cursor returns to the position after AMOUNT? and asks again. When the sum for withdrawal is permissible the following will appear on the last line of the screen:

```
! ACCOUNT NUMBER 3 UPDATED ! PRESS RETURN TO CONTINUE
```

When the RETURN key has been pressed, all the information about the last transaction disappears from the screen and the cursor returns to the first question. If the first letter of the answer is neither D, W nor E but, for instance, X a warning text should appear on the last line of the screen:

```
! NO COMMAND NAMED X !
```

and the cursor should return to the first question.
After an E(ND) command the following statistics should be printed on paper:

```
NUMBER OF DEPOSITS xxxxxxxxx
SUM OF DEPOSITS xxxxxxxxx
NUMBER OF WITHDRAWALS xxxxxxxxxxx
SUM OF WITHDRAWALS xxxxxxxxxxxx
```

8.2.2 Project: Telephone Directory*

Suppose that you have a random file like this:

```
BOX  1:  ANDERSSON JIM   345698
BOX  2:  BROWN JOHN      137968
.
.
.
BOX 58:  ZYGMUND ERIC    984793
```

containing names and telephone numbers, and sorted alphabetically in the way indicated above. Now suppose that you want to find out the telephone number of Michael Johnson. Let MIDDLE:=INT(1+58)/2) i.e., MIDDLE:=29. If the name in box 29 is the one you are looking for, then you have finished the exercise. If the name in box 29 comes after JOHNSON, MICHAEL then you should repeat the procedure, but with 58 replaced by 28. If the name in box 29 comes before JOHNSON, MICHAEL then you should repeat the procedure, but with 1 replaced by 30. This procedure is illustrated in Figure 8.1.

Before continuing two assumptions are made; the number of boxes is NOFBOXES: i.e. the boxes are numbered 1,2,3 . . . NOFBOXES; each box contains 50 characters, 35 for the name and 15 for the number (which is a string).

Exercise 8.4

Write a procedure

SEARCH(NAME$,REF WHICHBOX)

so that if the name in a box is equal to NAME$ then WHICHBOX is set to the number of this box. Otherwise WHICHBOX:=0.

Exercise 8.5

If, for instance, LOW=1 and HIGH=2000, what is the maximum number of times the loop in the procedure SEARCH can be repeated?

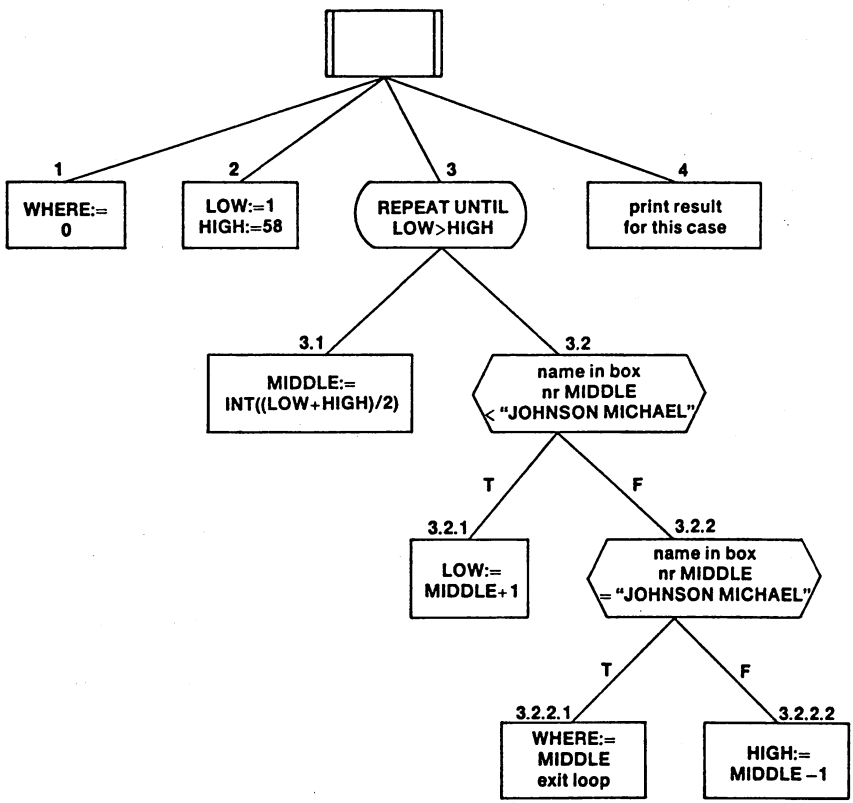


Figure 8.1 Structure Diagram to Illustrate Searching

Exercise 8.6

To keep such a file up to date you must be able to delete boxes, insert new boxes and change the contents of boxes (i.e. change the telephone number).

Construct three procedures which accomplish this and use them in a program that keeps the telephone directory up to date. Start by drawing structure diagrams.

8.2.3 Project: League Tables

Use these kinds of files to keep soccer, hockey or basketball (or any other) league tables in order. The contents in one box (for the National Hockey League) could be:

NEW YORK RANGES 10 7 2 1 64 19

where 10 is the number of fights so far, 7 the number won, 2 the number drawn and 1 the number lost. 64 means the number of goals scored and 19 the number of goals conceded.

When working with, for instance, the National Hockey league, one should be able to enter new results such as:

NEW YORK RANGERS – BUFFALO SABRES 4-1

One should also be able to obtain the current table (on paper or on the screen). The program should be able to administer an arbitary number of tables. The program should start by asking (only one table per session):

WHICH SPORTTABLE(TYPE HELP FOR HELP!)?

If HELP is typed, a list of all tables in the system should be obtained.

Exercise 8.7

How do you fix the problem that in England a soccer team gets three points for a win?

8.3 FUNCTIONS

The sole purpose of some (usually closed) procedures is to produce one number.

Example:

```
PROC MAX2(A,B, REF MAX) CLOSED
  IF A>B THEN
    MAX:=A
  ELSE
    MAX:=B
  ENDIF
ENDPROC MAX2
```

If you want to add the largest of X and Y to SUM this can be done in the following way:

```
.
.
.
430 EXEC MAX2(X,Y,V)
440 SUM:=SUM+V
```

However, in modern versions of COMAL you can do this instead. Define a so-called function this way:

```
FUNC MAX(A,B) CLOSED
  IF A>B THEN
    RETURN A
  ELSE
    RETURN B
  ENDIF
ENDFUNC MAX
```

and rewrite lines 430-440 to

```
430 SUM:=SUM+MAX(X,Y)
```

MAX(X,Y) is equal to the value returned above when A is replaced by the value of X and B by the value of Y.

(In some older versions one has this solution (the names of functions must in this case start with the letters FN):

```
DEF FNMAX(A,B)
  IF A>B THEN
    FNMAX:=A
  ELSE
    FNMAX:=B
  ENDIF
ENDDEF FNMAX
```

Line 430 would then be written:

```
430 SUM:=SUM + FNMAX(X,Y)
```

Another old-fashioned (and sometimes confusing solution) is this:

```
PROC MAX(A,B) CLOSED
  IF A>B THEN
    MAX:=A
  ELSE
    MAX:=B
  ENDIF
ENDPROC MAX
```

In this case line 430 would be:

```
430 SUM:=SUM + MAX(X,Y))
```

The first alternative (using FUNC) is today considered standard, and will be adhered to in this book.

Exercise 8.8

Find out which of all the procedures you have used so far could be rewritten using functions.

Exercise 8.9*

If you do not have the function ABS in your language you could easily define it. Do so, calling it XABS.

You can use different kinds of (closed) functions to produce random numbers, for example this way: Let P denote a prime number, for instance P=13. Start by determining a start value that must be an odd integer, called R, then determine successive random numbers by executing the following lines again and again:

```
R:=R*P
R:=R MOD 1000
```

R/1000 is then a random number.

Exercise 8.10*

Construct a function XRND(REF R) CLOSED using the above code. Why must R be used with REF?

Exercise 8.11*

Construct another function, TRND, that could be used instead of RND(,).

Exercise 8.12

Test this generation of "random" numbers by using the procedure HISTO.

Compare the numbers with those you can obtain in COMAL using the RND function.

A subtle point:
As you may already know, you can use the computer to compute more or less complicated mathematical expressions *without using line numbers*.

Examples:

```
PRINT 5+7
12
X:=6
IF X=6 THEN PRINT 7
7
Y:=2
PRINT X/Y
3
```

and so on.
But what happens if you type the following line?

```
PRINT 5>3
```

Shouldn't this produce an error message? Is it possible to print a condition? The answer is yes, and is explained by the fact that the interpreter converts a true condition to 1 and a false one to 0.

```
PRINT 5=8
0
PRINT TRUE
1
PRINT FALSE
0
```

What happens when you execute the following lines?

```
10 FOR I:=-3 TO 3 DO
20   PRINT "I=",I
30   IF I THEN
40     PRINT "I TRUE"
50   ELSE
60     PRINT "I FALSE"
70   ENDIF
80 NEXT I
```

If a number is different from 0 it is considered to be TRUE, otherwise it is considered to be FALSE. This can be used in the following way. First define a function:

```
FUNC OK(NUMBER) CLOSED
  IF INT(NUMBER)=NUMBER AND NUMBER>0 THEN
    RETURN TRUE
  ELSE
    RETURN FALSE
  ENDIF
ENDFUNC
```

Note that TRUE is only a way of writing 1 that makes programs more readable (the same applies for FALSE and 0).
In the third line of code above you could equally write

```
RETURN 1
```

This function can be used in the following way when entering numbers:

```
REPEAT
  INPUT "A POSITIVE INTEGER,PLEASE!": NUMBER
UNTIL OK(NUMBER)
```

The code for OK contains the word AND.
You can use AND, OR and NOT to compose conditions, for instance:

```
IF SUM>0 AND NOF=200 THEN . . .
IF NOT(X=0 OR Y>X) THEN . . .
```

A condition with AND is true when both the composing conditions are true. A condition with OR is true when at least one of the composing conditions is true and NOT makes a true condition false and a false condition true. NOT(3>5) is true. If you use AND and OR in the same condition you should use parentheses. For example, instead of:

```
IF A=0 AND B=0 OR C=0 THEN
```

use either:

(a) IF(A=0 AND B=0) OR C=0 THEN
or
(b) IF A=0 AND (B=0 OR C=0) THEN

Exercise 8.13
Investigate in which of the above examples the parentheses aren't necessary.

Exercise 8.14
Find values of A, B and C that
(a) make the condition in (a) above true, and that in (b) false;
(b) make the condition in (a) above false, and that in (b) true.

Exercise 8.15*
Rewrite the code for OK using OR instead of AND. (*Note:* several changes must be done.)

A function that only returns the values TRUE (i.e. 1) or FALSE (i.e. 0) is called a Boolean function.

Exercise 8.16*

Construct a closed Boolean function

```
BOUND(LOW,HIGH,NUMBER)
```

which is true precisely when LOW<NUMBER<HIGH. Write two versions – one with AND and the other with OR.

Exercise 8.17

Write a code segment that only lets through a number when it is an integer between the numbers 100 and 1 000.
Try two different kinds of solutions. In one answer, use the functions defined so far, and in the other use a Boolean function

```
GOOD(LOW,HIGH,NUMBER)
```

with suitable properties.

```
REPEAT
  "INPUT A POSITIVE INTEGER,PLEASE!": NUMBER
UNTIL GOOD (100,1000,NUMBER)
```

Matching Strings
If you type

```
PRINT "OH" IN "JOHN"
```

the answer 2 will appear on the screen. If you type

```
PRINT "N" IN "JOHN"
```

the answer 4 will appear, but if you type

```
PRINT "AH" IN "JOHN"
```

you'll get the answer 0, What would the result of

```
"COMPSON" IN "BILL COMPSON"
```

be?
By typing PRINT "CO" IN "COCONUT" you'll get the answer 6 (i.e. the first match!)

Exercise 8.18

From these examples, and others you invent yourself, explain how IN works.

Exercise 8.19

What is the effect of the following code segment:

```
NOF1:=0
NOF2:=0
L:=LEN(WORD$)
FOR I:=1 TO L DO
  CH$:=WORD$(I:I)  // ONE CHARACTER
  IF CH$ IN "AEIOU" THEN
    NOF1:=NOF1+1
  ELSE
    NOF2:=NOF2+1
  ENDIF
NEXT I
PRINT . . .
```

Find this out without running the program. Change the variable-names NOF1 and NOF2 to something more readable. Fill in the PRINT statement(s).

Exercise 8.20

Construct a procedure ROBBER(REF WORD$), so that if WORD$="ULLA", ULOLLOLA is printed on the screen. (I.e. each consonant is followed by an O and the same consonant again.)

Exercise 8.21 (difficult)

Construct a procedure that translates in the other direction, i.e. from the "robber" language back to ordinary English.

Exercise 8.22*

Construct a procedure ALLIN(SMALL$,BIG$) such that ALLIN ("CO","COCONUTCOAL") results in the printout:

```
1
3
8
```

but ALLIN("CO","HORSE") results in:

```
NO MATCH!
```

Exercise 8.23*

Construct a procedure:

```
MYIN(SMALL$,BIG$)
```

with precisely the same effect as the built-in function IN (without using IN).

CONCLUSION

Now that you have come this far it's time to round off.
I have tried to teach you what is called structured programming.

> *"Simply put, structured programming is a set of techniques that makes programs easier to write, easier to understand, easier to fix and easier to change."*
>
> *Gregg Williams (BYTE,* March 1981)

The main point with structured programming is to divide problems into subproblems again and again, until one gets manageable and reasonably clean problems. This is called top-down design, and this is exactly what you have been doing when you have been drawing successively more detailed structure diagrams. In structure diagrams there are essentially three different figures, one illustrating sequence, one illustrating repetition and one illustrating alternatives. These three fundamental ways of composing programs are also considered an essential part of structure programming.

The hardest part about programming is to find out *what* the problem the computer is to solve really is. After this has become clear you might design a solution method (*how* to solve it), for instance with structure diagrams. The main point with this method is that the structure diagrams can be used for many different programming languages, for instance Pascal or ADA. Once the structure diagrams have been drawn, the coding should be routine.

You should now have a feeling for what programming really is, and you should know enough to be able to attack larger, more realistic problems. Good luck!

Solutions to Exercises and Problems

Exercise 1.1

After line 40 you have

X	3
Y	4
Z	12

After line 50 the last box is instead

Z	15

and this value gets printed in line 60.

Exercise 1.2

If you take away the PRINT statements you won't see any results of what has been happening inside the computer.

Exercise 1.3

Q	41
A	14

Exercise 1.5

To insert 43 lines you can do as follows: Give the command AUTO 41,1. Insert 9 lines and leave AUTO. Renumber and insert another 9 lines, and so on. (There are smarter ways – you can use RENUMBER – look up your manual.)

Question 1.6

Construct a table like this:

```
          A      B      C
          47    -36     15
140
          26
150
                  5
160
                        46
```

Exercise 2.8

```
10 PRINT "WHAT IS THE FIRST NUMBER,
20 INPUT NUMBER1
30 PRINT "WHAT IS THE SECOND NUMBER",
40 INPUT NUMBER2
50 PRINT "WHAT IS THE THIRD NUMBER",
60 INPUT NUMBER3
70 SUM:=NUMBER1+NUMBER2+NUMBER3
80 PRINT "SUM OF THE THREE NUMBERS = ",SUM
```

Perhaps you answered something like this? Then you may have forgotten the important first three lines! What should they look like?

Exercise 2.14

To obtain for instance

```
1982        5 DOLLARS        NO 347
```

use

```
PRINT YEAR,TAB(12),FEE,"DOLLARS      NO ", MEMBERNR
```

and *not*

```
PRINT "1982   5 DOLLARS   No", MEMBERNR
```

The variables YEAR and FEE obtain their values in the beginning of the program. For instance.

```
60 YEAR:=1982
70 FEE:=5
```

Question 2.5

No. The statement PRINT simply causes the next printing to start at the first position on the left of the next line.

Question 2.7

```
40 PRINT "WHICH NUMBER",
50 INPUT NUMBER
60 SQUARE:=NUMBER * NUMBER
70 PRINT NUMBER,"*",NUMBER,"=",SQUARE
```

or with one line 60 instead of the two lines 60 and 70 above:

```
60 PRINT NUMBER,"*",NUMBER,"=",NUMBER *NUMBER
```

i.e. the value of the expression

```
NUMBER*NUMBER
```

is first computed and then printed out.

Question 2.8

For instance by having on the screen:

```
THIS PROGRAM COMPUTES THE SQUARE OF A NUMBER YOU
ENTER WHICH NUMBER?
```

and so on.

Question 2.10

Name the variables NUMBER, PRICE and COST. When the user has entered values for NUMBER and PRICE the cost is computed by:

```
COST:=NUMBER*PRICE
```

To print the bill use code like this:

```
100 PRINT "BILL"
110 PRINT "===="
120 PRINT
130 PRINT "NUMBER OF UNITS",TAB(25), "PRICE PER UNIT"
140 PRINT NUMBER,TAB(25),PRICE
150 PRINT
160 PRINT "TOTAL COST"
170 PRINT COST
180 PRINT
190 PRINT "--------------------"
200 PRINT
```

Question 2.12

To print a rectangle use code like this:

```
 40 PRINT "----------"
 50 PRINT "I",TAB(20),"I"
 60
 70
 80
 90
100
 .
 .
 .
```

Lines 60-90 are identical to line 50 and line 100 is identical to line 40.

Question 2.13

```
 10 // BILL—PROGRAM 3
 20 // 25 AUG. 1982 VERSION 1.1
 30 // BY BILL COMPSON
 40 // ==========
 50 DIM MONEYUNIT$ OF 12
 60 MONEYUNIT$:="DOLLARS"
 70 INPUT "NUMBER OF UNITS?": NUMBER
 80 PRINT "PRICE PER UNIT(IN ",MONEYUNIT$,")",
 90 INPUT PRICE
100 COST:=NUMBER*PRICE
110 SELECT OUTPUT"LP"
120 PRINT "BILL"
130 PRINT "===="
140 PRINT
150 PRINT "NO. OF UNITS",TAB(25),"PRICE UNIT IN ",MONEYUNIT$
160 PRINT NUMBER,TAB(25),PRICE
170 PRINT
180 PRINT "TOTAL COST"
190 PRINT COST,"(",MONEYUNIT$,")"
200 PRINT
210 PRINT "----------"
220 PRINT
230 SELECT OUTPUT "DS"
```

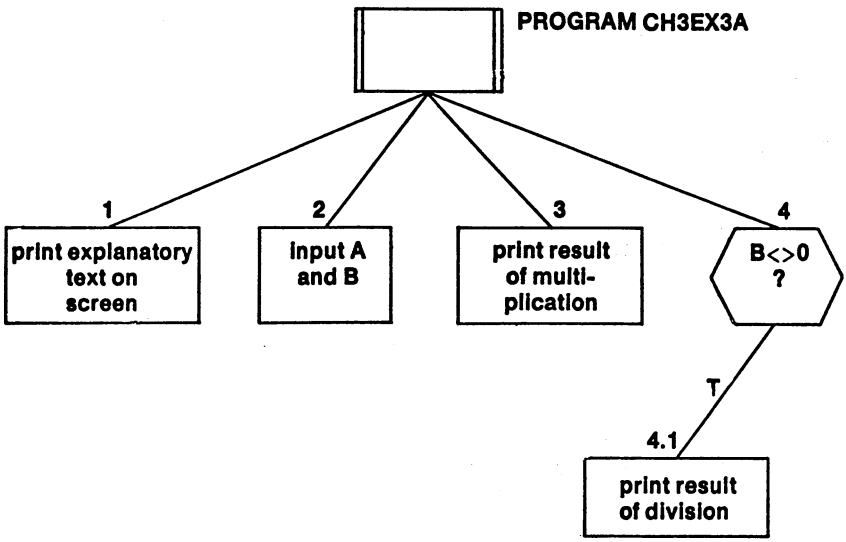
Note that the lines 110 and 230 are system-dependent.

Question 2.16

(a) 1/(*x*+2)
(b) 1(2**x*−7)
(c) SQR(1/(3**x*+5))
(d) ABS((3**x*−5)/(5**x*−7))
(e) SQR((3**x*−5)/ABS(2**x*+5))

Exercise 3.3

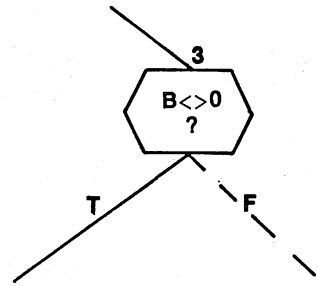
A structure diagram for (a) could look like this:



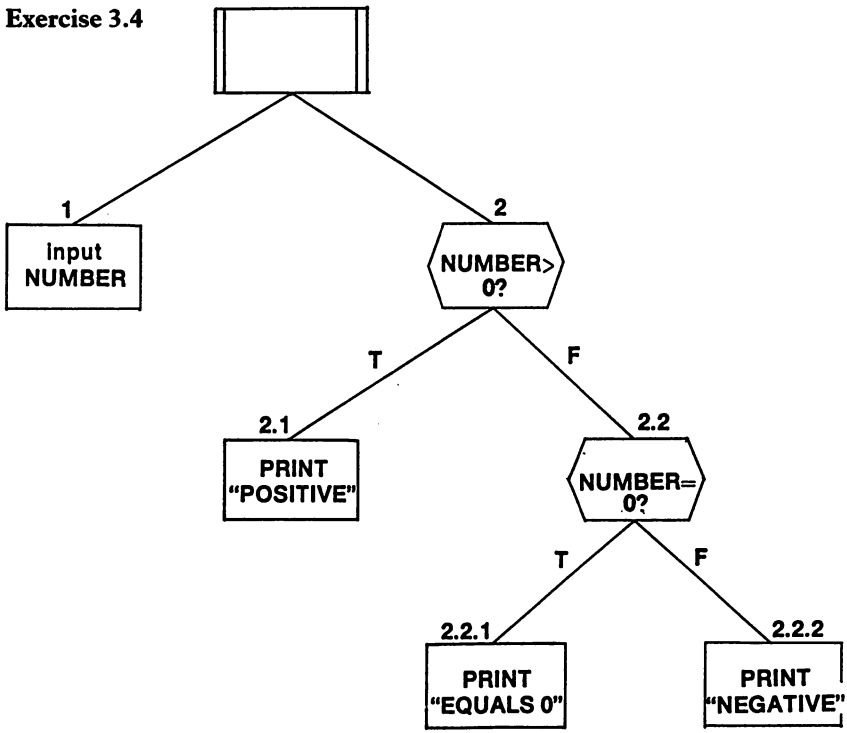
Hints: In (b) there must also somewhere be a box marked for instance



In (c) you should get this



Exercise 3.4

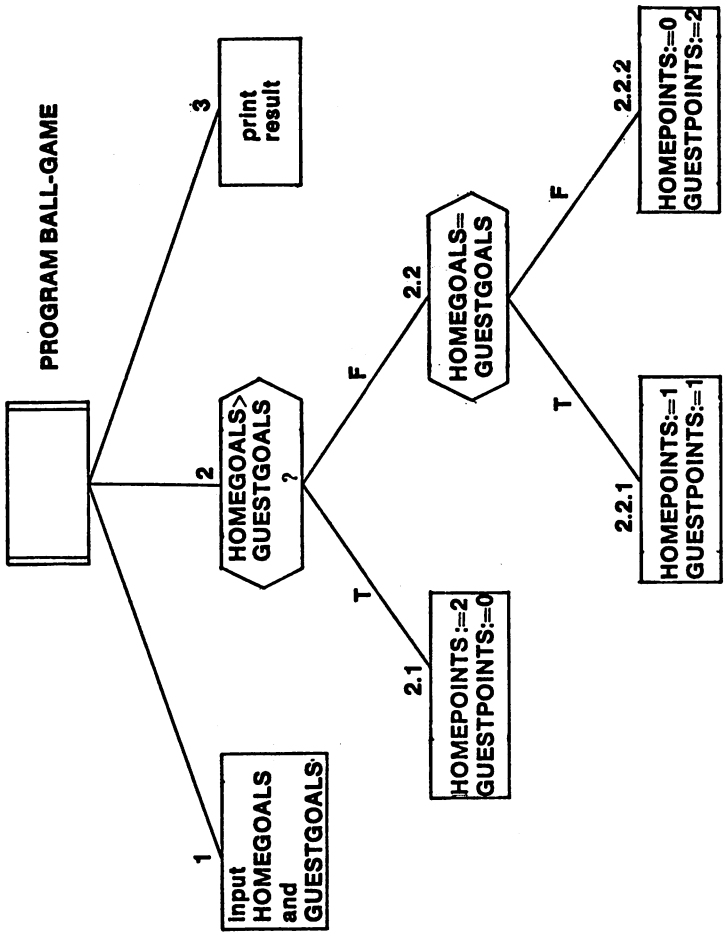


The code corresponding to box 2 is (without line-numbers)

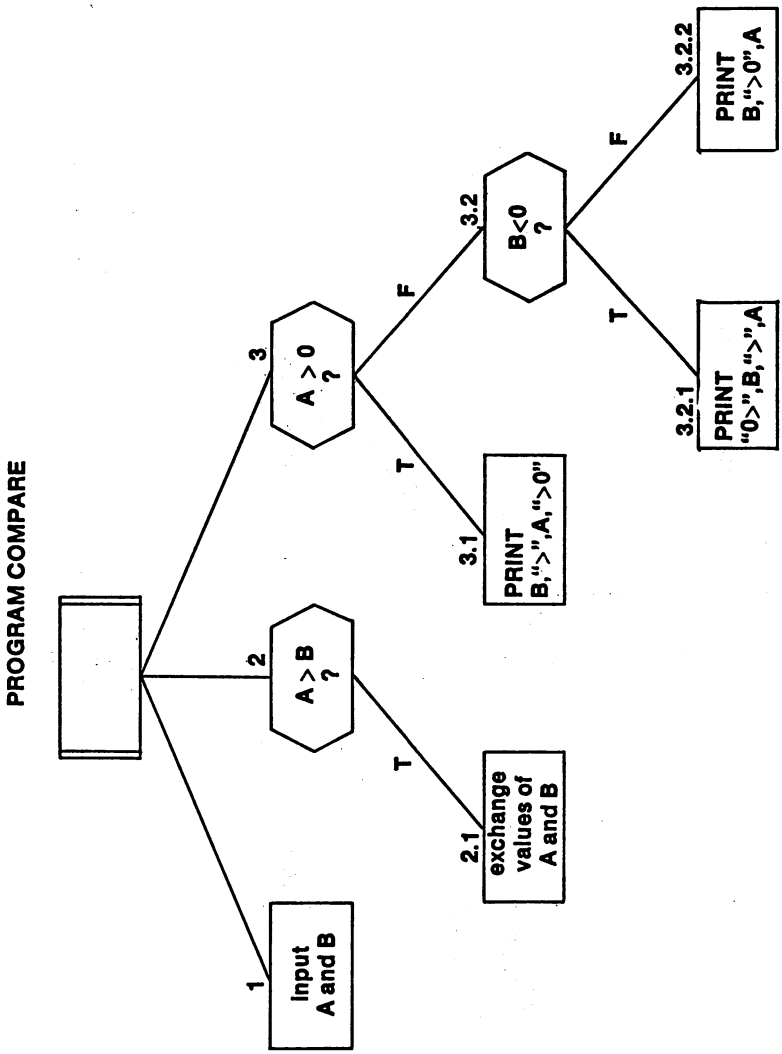
```
IF NUMBER>0 THEN
  PRINT "POSITIVE"
ELIF NUMBER=0 THEN
  PRINT "EQUALS 0"
ELSE
  PRINT "NEGATIVE"
ENDIF
```

Exercise 3.8

```
 40 INPUT "WHAT IS A? ": a
 50 IF a>3 THEN
 60    PRINT a," > 3"
 70 ELIF a>0 THEN
 80    PRINT a," is small but positive"
 90 ELIF a=0 THEN
100    PRINT a," = 0"
110 ELSE
120    PRINT a," < 0"
130 ENDIF
```

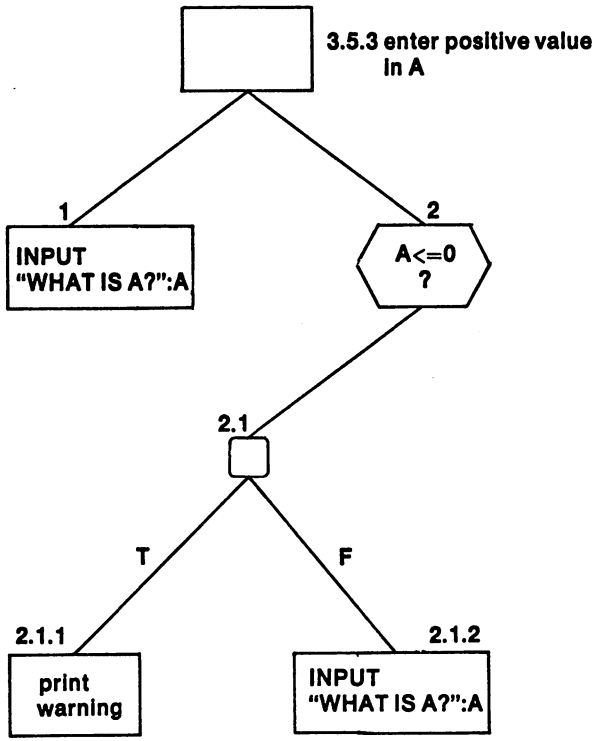

PROGRAM BALL-GAME
1
Input HOMEGOALS and GUESTGOALS
2
HOMEGOALS> GUESTGOALS ?
T
2.1
HOMEPOINTS :=2 GUESTPOINTS:=0
F
2.2
HOMEGOALS= GUESTGOALS
T
2.2.1
HOMEPOINTS:=1 GUESTPOINTS:=1
F
2.2.2
HOMEPOINTS:=0 GUESTPOINTS:=2
3
print result

Exercise 3.6



Exercise 3.11

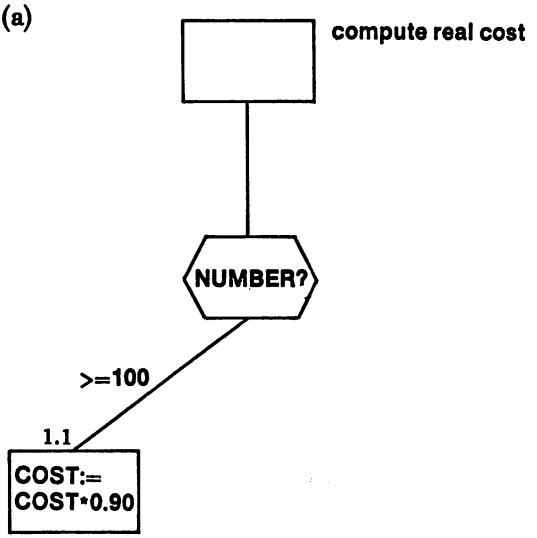
Exercise 3.13



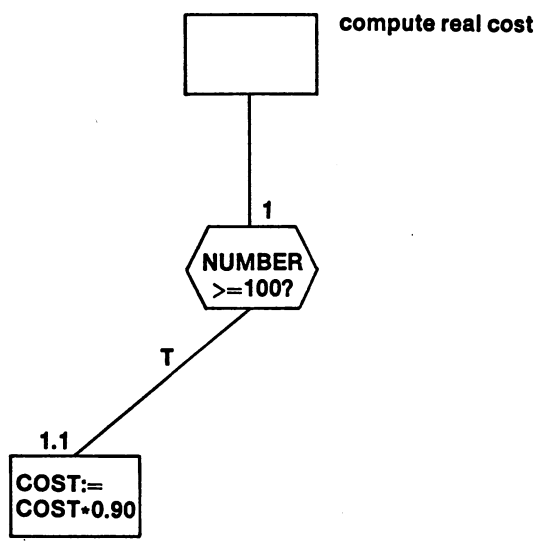
A better way of solving this exercise is explained in Chapter 4. Try this exercise again when you have read Chapter 4. (This is not a fool-proof way to get a positive value for A!)

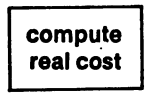
Exercise 3.14

(a)

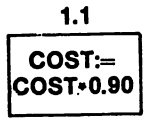


or



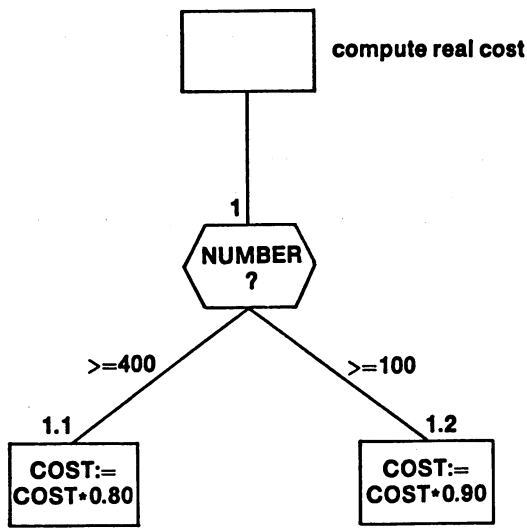


is a box in a previous structure. If this box is numbered 4.3 (for example) then the box



has the number 4.3.1.1

(b)



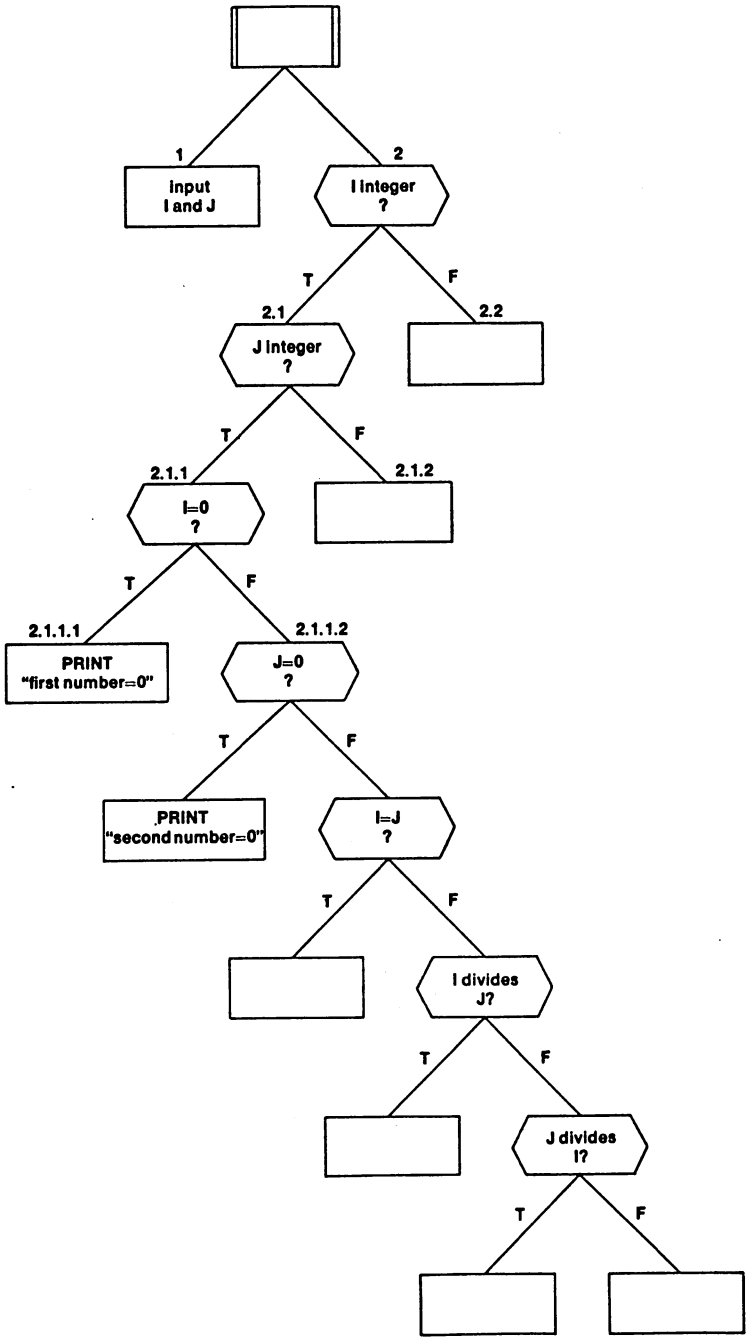
Exercise 3.24 (a)

```
CASE ANSWERS OF
WHEN "MORNING"
  xxxxxxxxxxxxxxx        an arbitrary number of lines to use when
  xxxxxxxxxxxxxxx        the answer is MORNING
WHEN "DAY"
  xxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxx
WHEN "EVENING"
  xxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxx
WHEN "NIGHT"
  xxxxxxxxxxxxxxx
  xxxxxxxxxxxxxxx
OTHERWISE
  PRINT ANSWER$,"    I DO NOT UNDERSTAND",ANSWER$
ENDCASE
```

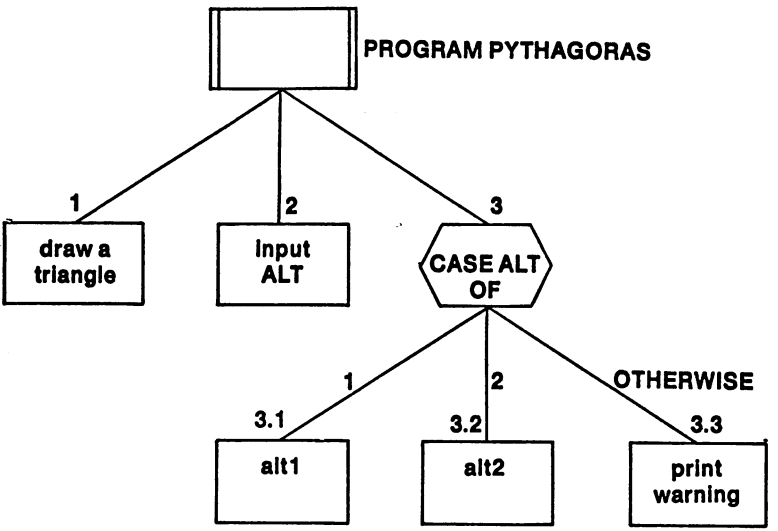
Exercise 3.27

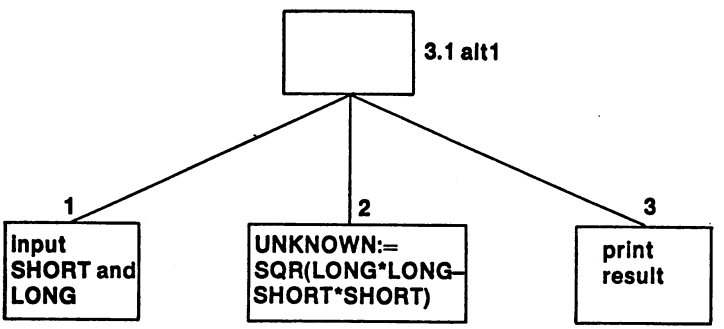
Give the empty boxes suitable contents. (Next page!)

As you can see, when there are many levels the numbering system becomes a bit hard to use. A better way of solving this exercise is explained in Chapter 4.


1
Input
I and J
2
I Integer
?
T
F
2.1
J Integer
?
2.2
T
F
2.1.1
I=0
?
2.1.2
T
F
2.1.1.1
PRINT
"first number=0"
2.1.1.2
J=0
?
T
F
PRINT
"second number=0"
I=J
?
T
F
I divides
J?
T
F
J divides
I?
T
F

Question 3.6

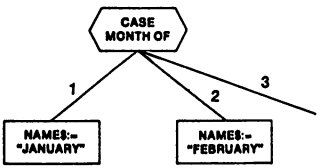




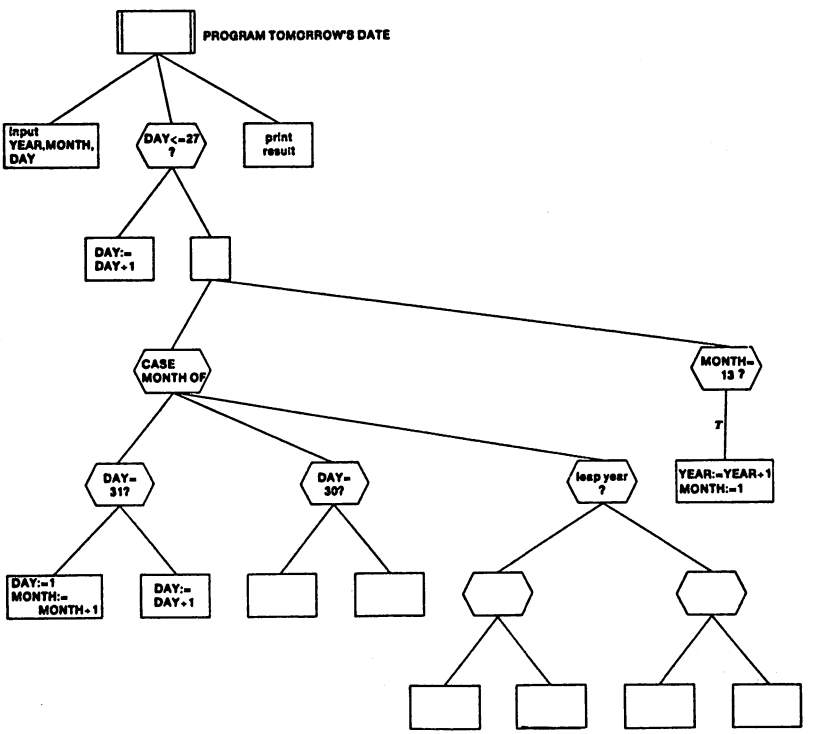
and analogously for alt2.

Question 3.7

A CASE statement is needed here for (b), such as



and so on (12 boxes – one per month)

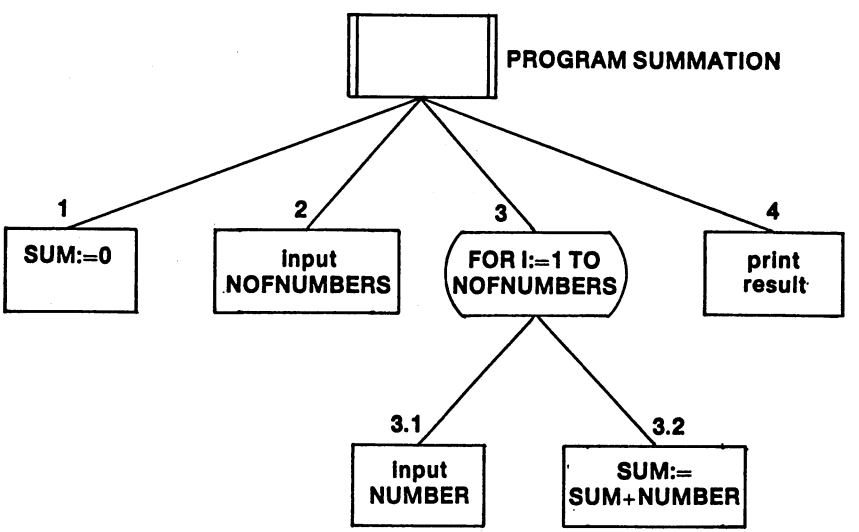


Complete the structure diagram.

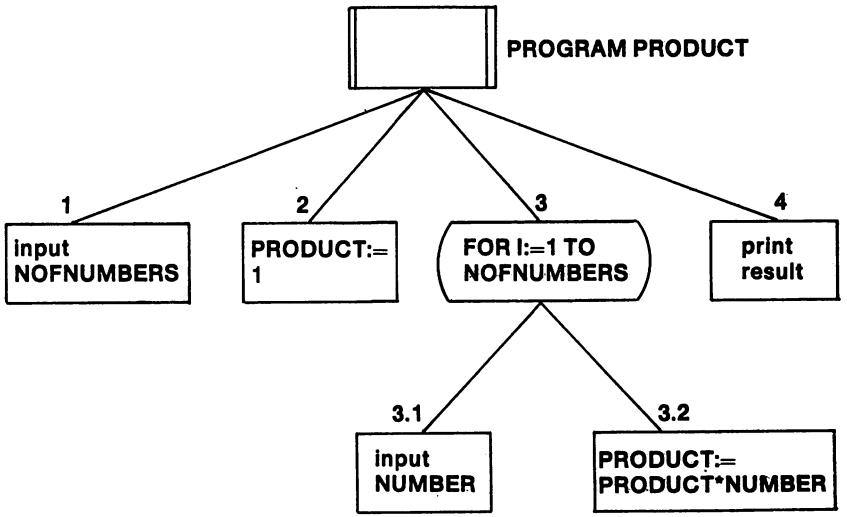
Exercise 4.3

```
10 // table of squares and cubes
20 // 5 november 1982 version 1.2
30 // by bill compson
40 //
50 PRINT "program produces tables of squares and cubes of integers
60 INPUT "from which value?": first
70 INPUT "to which value?": last
80 SELECT OUTPUT "lp"
90 PRINT "*** table of squares and cubes ***"
100 PRINT "**************************"
110 PRINT
120 ZONE 15
130 PRINT "integer","square","cube"
140 PRINT "=======","======","===="
150 FOR i:=first TO last DO
160   square:=i*i
170   cube:=i*i*i
180   PRINT i,square,cube
190 NEXT i
200 SELECT OUTPUT "ds"
```

Exercise 4.6



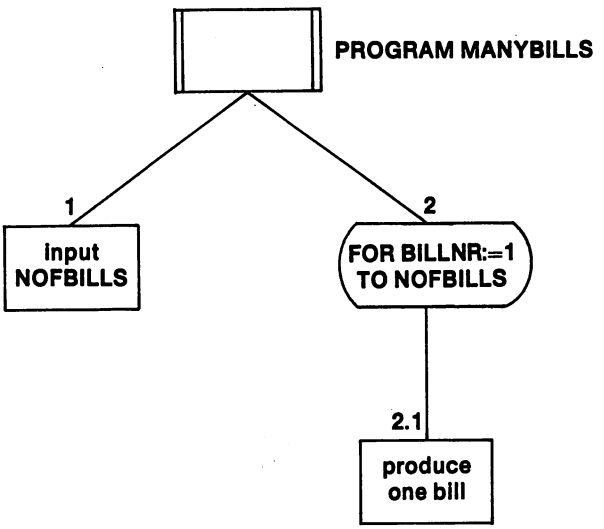
Exercise 4.7



Exercise 4.9

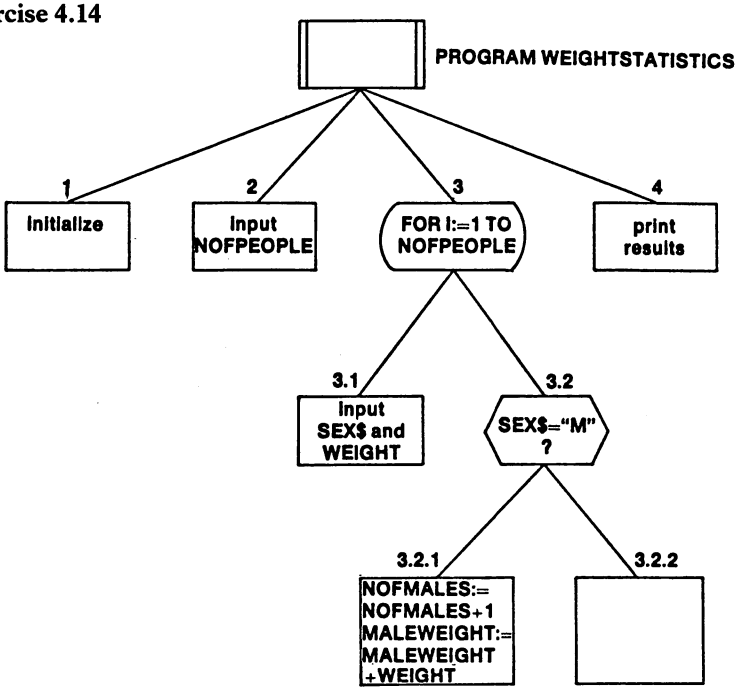
```
10 // rectangle program
20 // 5 november 1982 version 1.1
30 // by bill compson
40 //
50 PRINT "THIS PROGRAM DRAWS RECTANGLES!"
60 INPUT "HEIGHT?": height
70 INPUT "WIDTH": width
80 SELECT OUTPUT "lp"
90 FOR w:=1 TO width DO
100    PRINT "*",
110 NEXT w
120 PRINT
130 FOR h:=2 TO height—1 DO
140    PRINT "*",TAB(width),"*"
150 NEXT h
160 FOR w:=1 TO width DO
170    PRINT "*",
180 NEXT w
190 PRINT
200 SELECT OUTPUT "ds"
```

Exercise 4.13

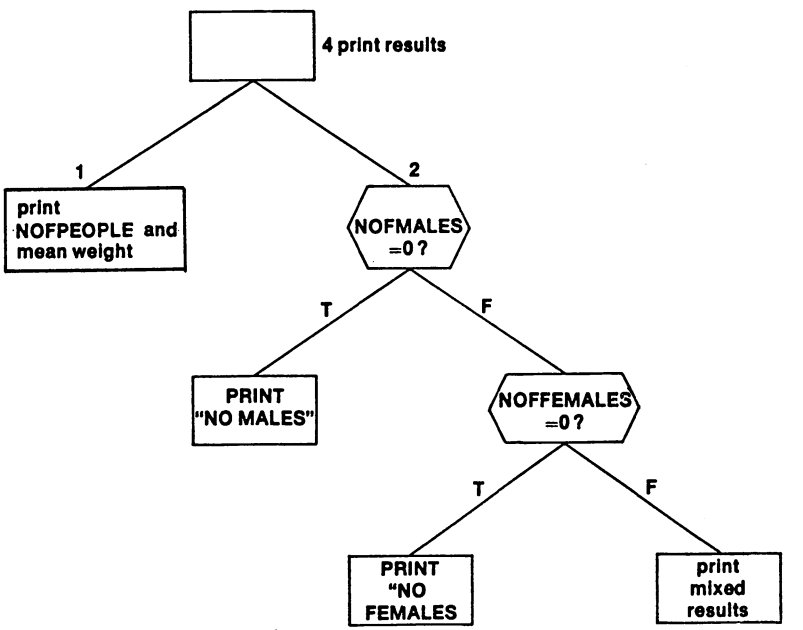


From doing the previous exercise you should now know how to produce one bill.

Exercise 4.14



Write the contents of box 3.2.2 yourself.



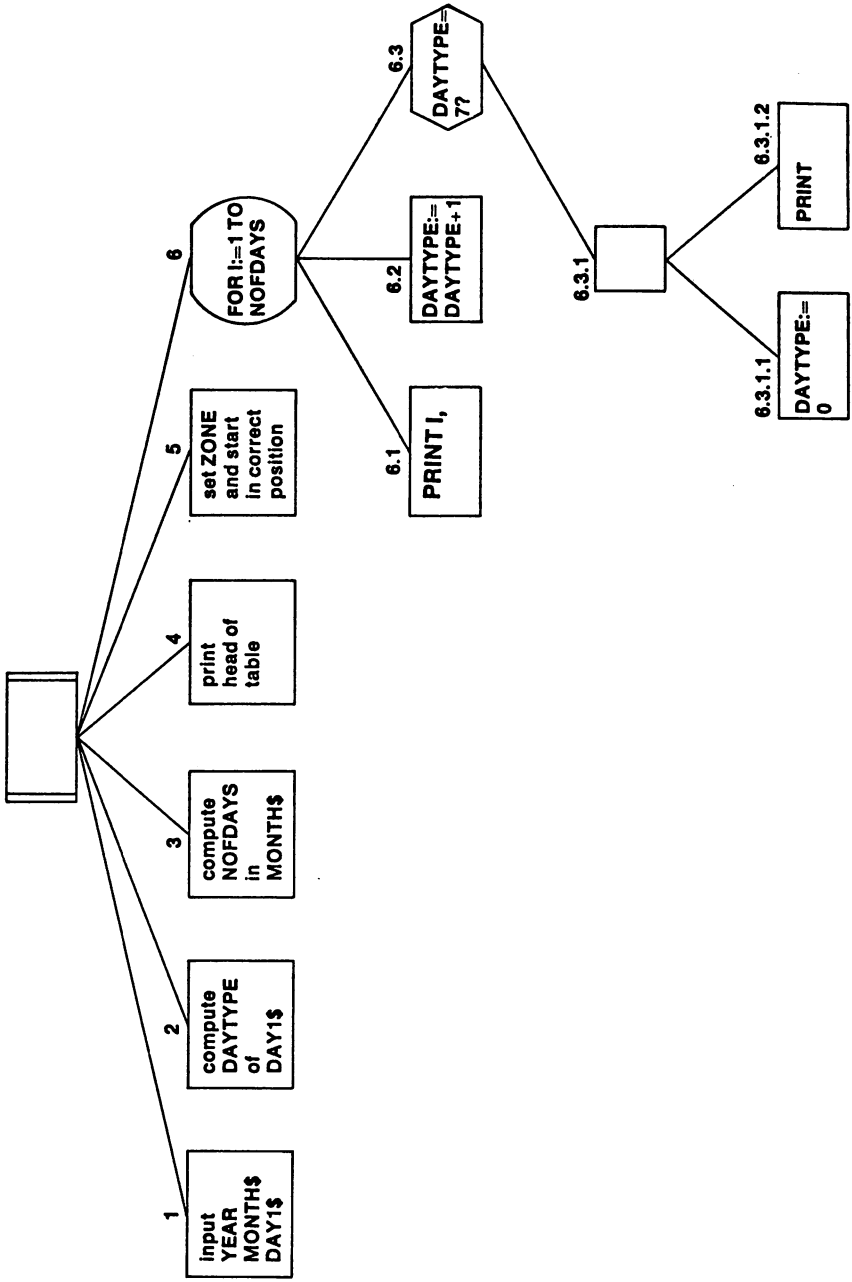
Exercise 4.15

Start by drawing the heading, then compute the number of days in the month.

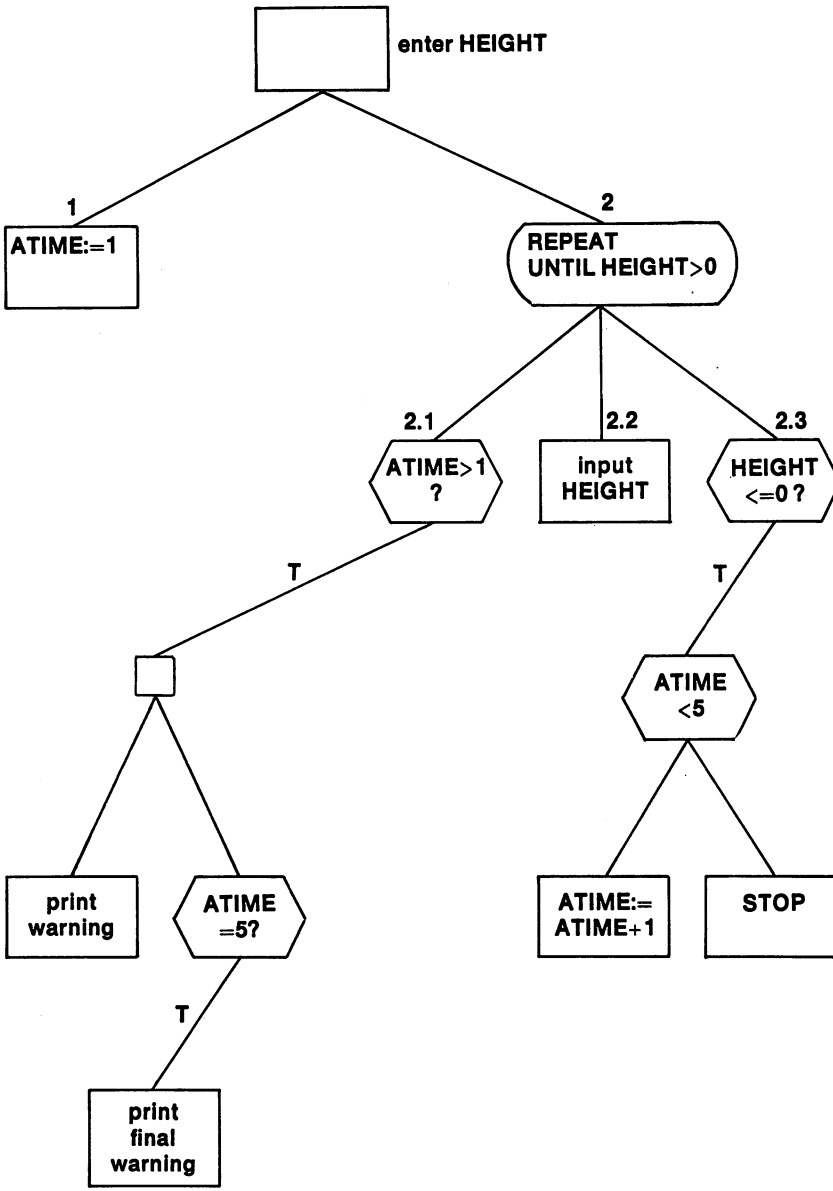
Introduce a variable DAYTYPE, so that

SUNDAY has DAYTYPE=0
MONDAY has DAYTYPE=1
.
.
.
SATURDAY has DAYTYPE=6

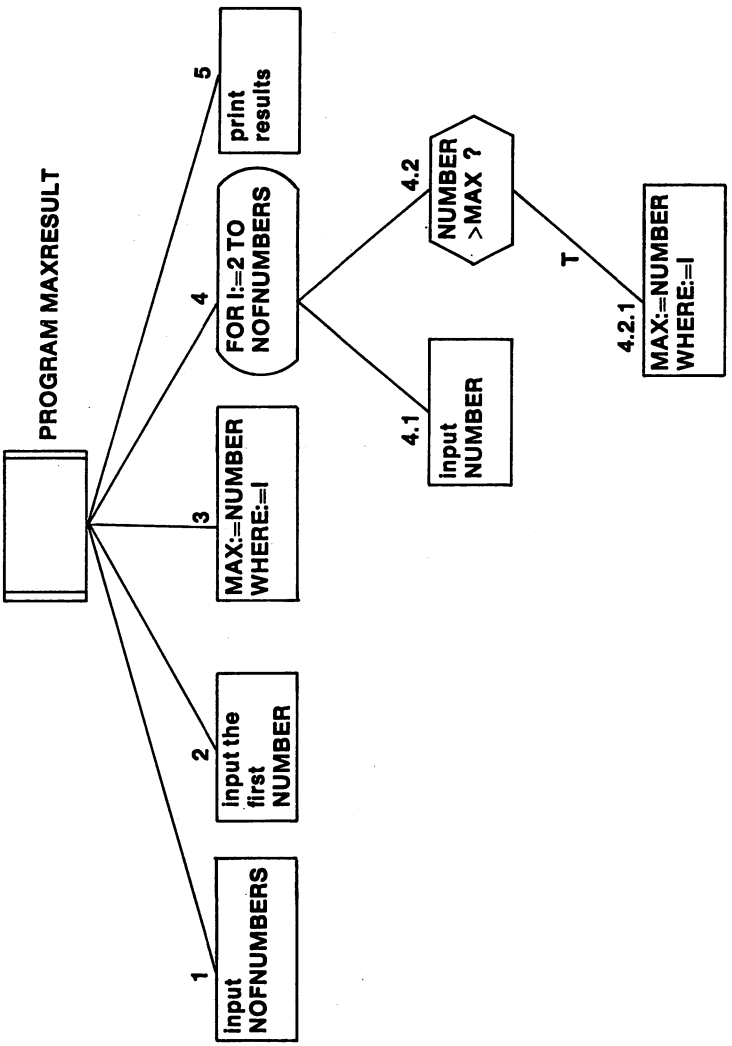
then you know in which position to start when printing the first day, i.e. 1. All the time you should update the value of DAYTYPE, so that when DAYTYPE=6 it is time for a new line.


1
input YEAR MONTH$ DAY1$
2
compute DAYTYPE of DAY1$
3
compute NOFDAYS in MONTH$
4
print head of table
5
set ZONE and start in correct position
6
FOR I:=1 TO NOFDAYS
6.1
PRINT I,
6.2
DAYTYPE:= DAYTYPE+1
6.3
DAYTYPE= 7?
6.3.1
6.3.1.1
DAYTYPE:= 0
6.3.1.2
PRINT

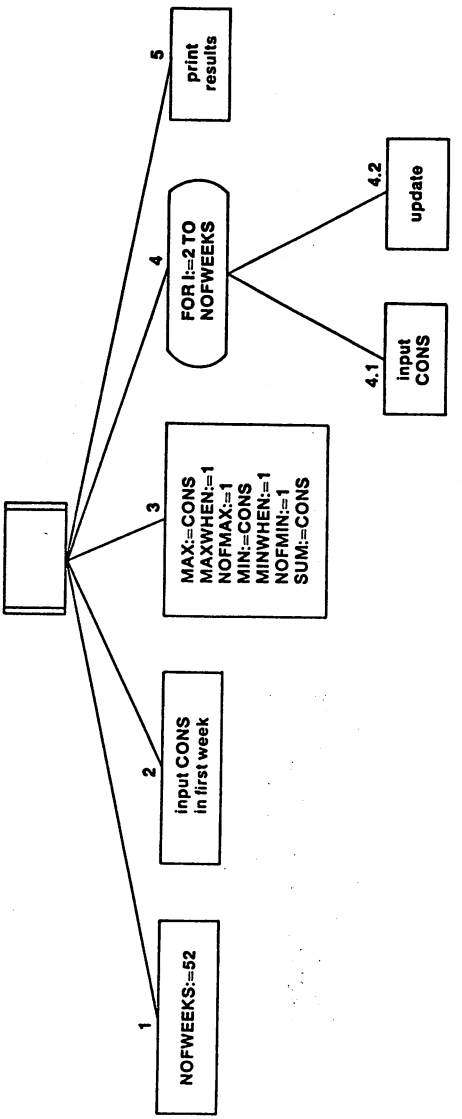
Exercise 4.24

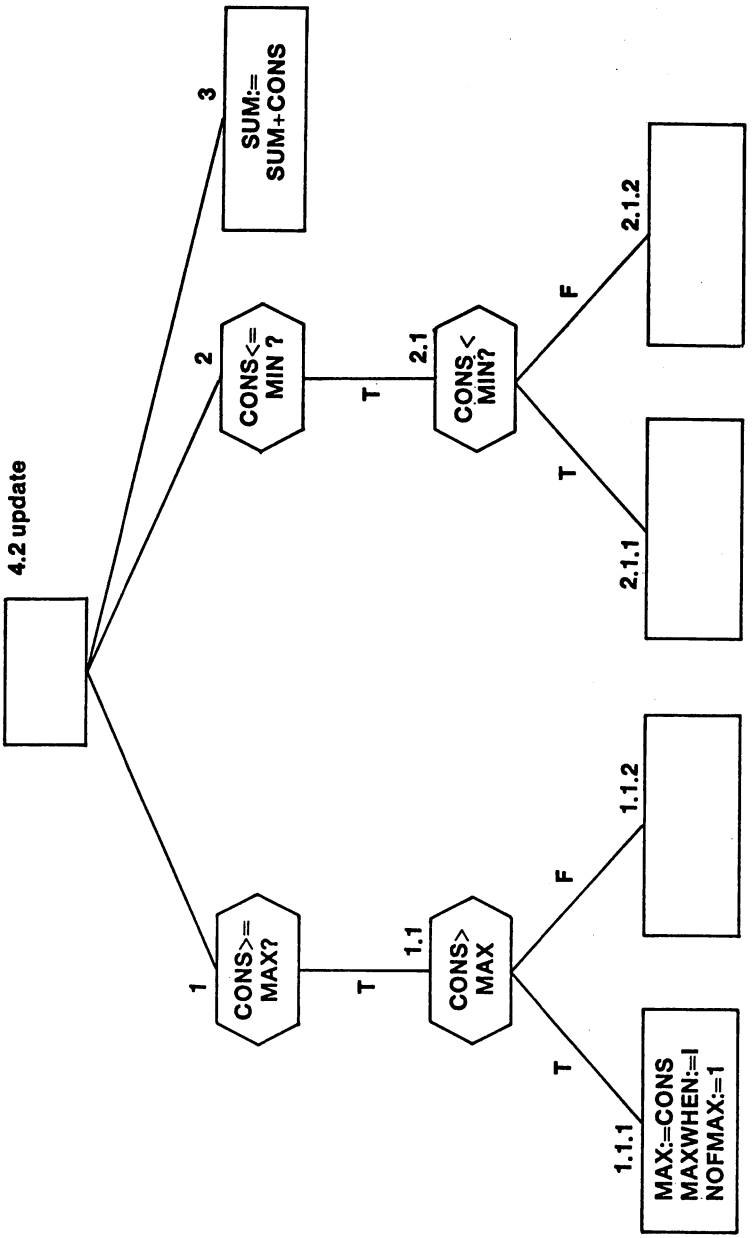


Exercise 4.36



Exercise 4.38(e)





This solution covers all of Exercise 4.38, except 4.38(c).

Exercise 5.1

(a)
```
FOR J:=4 TO 1 STEP -1 DO
    EXEC STARS(J)
NEXT J
```

(b)
```
FOR J:=1 TO 5 STEP 2 DO
    EXEC STARS(J)
NEXT J
```

Exercise 5.11

```
200 PROC drawrect(leftpos,width,height) CLOSED
210   PRINT TAB(leftpos),
220   FOR i:=1 TO width DO
230     PRINT "*",
240   NEXT i
250   PRINT
260   FOR j:=TO height -1 DO
270     PRINT TAB(leftpos),"*",TAB(leftpos+width),"*"
280   NEXT j
290   FOR i:=1 TO width DO
300     PRINT "*",
310   NEXT i
320   PRINT
330 ENDPROC drawrect
```

Exercise 5.12

```
370 PROC drawsquare(leftpos,sidelength) CLOSED
380   exec drawrect(leftpos,sidelength,sidelength)
390 ENDPROC
```

or (depending on your system)

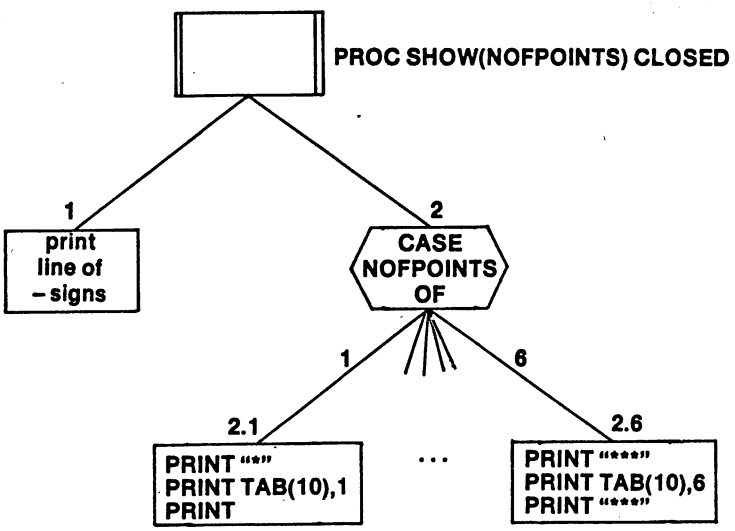
```
370 PROC drawsquare(leftpos,sidelength) CLOSED
380   IMPORT drawrect
390   exec drawrect(leftpos,sidelength,sidelength)
400 ENDPROC
```

Exercise 5.14

```
PROC BLANKLINES(NOFLINES) CLOSED
  FOR LINENR:=1 TO NOFLINES DO
    PRINT
  NEXT LINENR
ENDPROC BLANKLINES

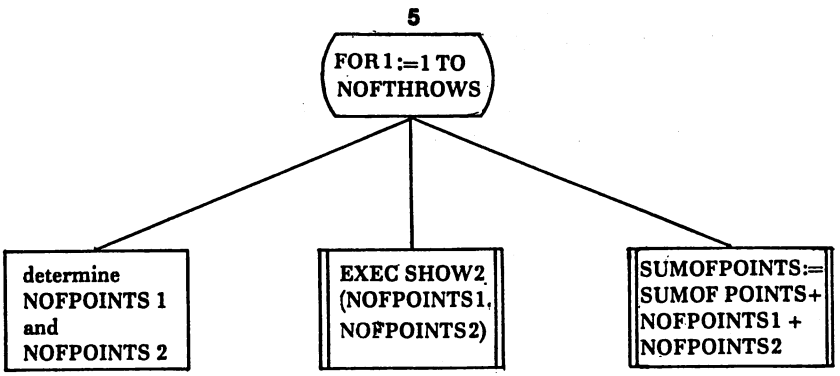
PROC ONESIGN(S$,NOFTIMES) CLOSED
  FOR I:=1 TO NOFTIMES DO
    PRINT S$,
  NEXT I
  PRINT
ENDPROC ONESIGN
```

Exercise 5.26



Exercise 5.27

The only change in the coarse structure diagram for PROGRAM THROWADICE (Figure 5.4), is:



Question 5.6

```
PROC DAYSFROMNOW(NOFDAYS,REF YEAR,REF
MONTHNR,REF DAYNR) CLOSED
  IMPORT TOMORROW
  FOR I:=1 TO NOFDAYS
    EXEC TOMORROW(YEAR,MONTHNR,DAYNR)
  NEXT I
ENDPROC DAYSFROMNOW
```

Note that in some systems there is no need to import procedures.

Exercise 6.1

```
CASE OUTCOME OF
WHEN 1
  PRINT "ONE";
WHEN 2
  PRINT "TWO";
WHEN 3
  PRINT "THREE"
WHEN 4
  PRINT "FOUR";
WHEN 5
  PRINT "FIVE";
WHEN 6
  PRINT "SIX";
ENDCASE
```

Exercise 6.2

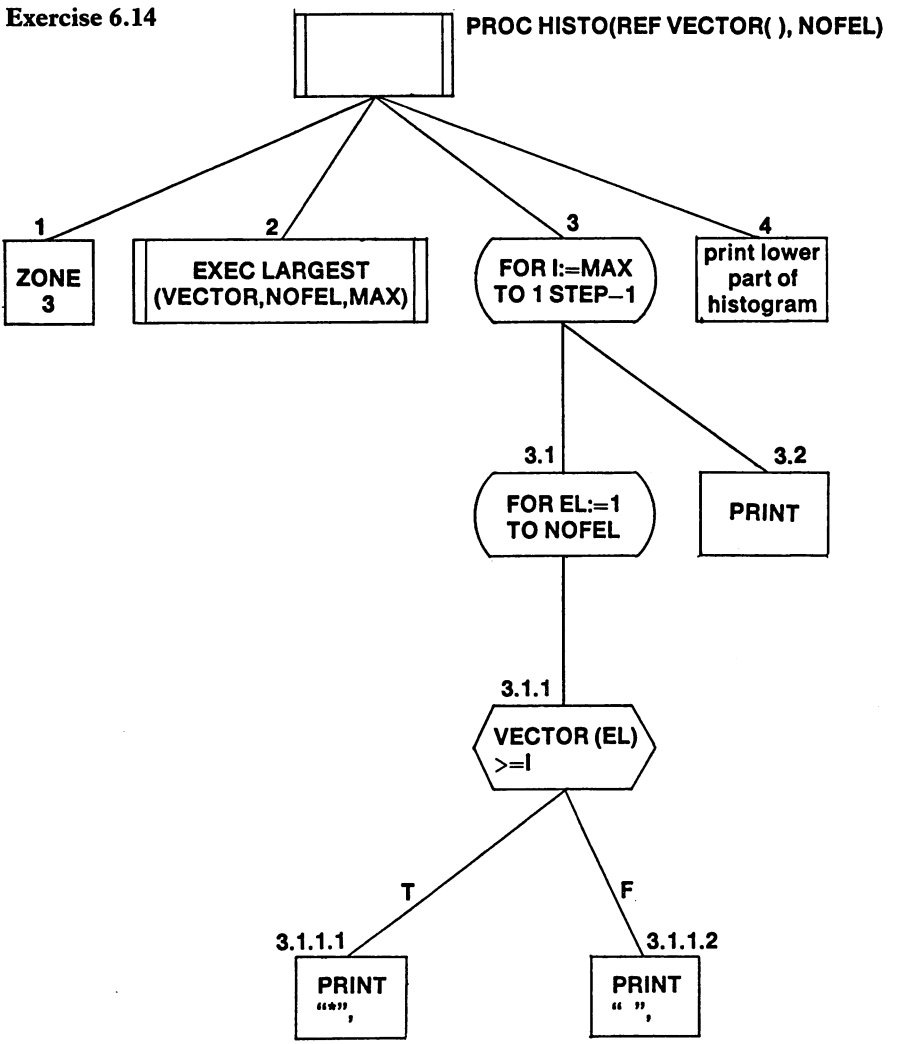
```
PRINT THROWNAME$(OUTCOME);
```

Exercises 6.9 & 6.10

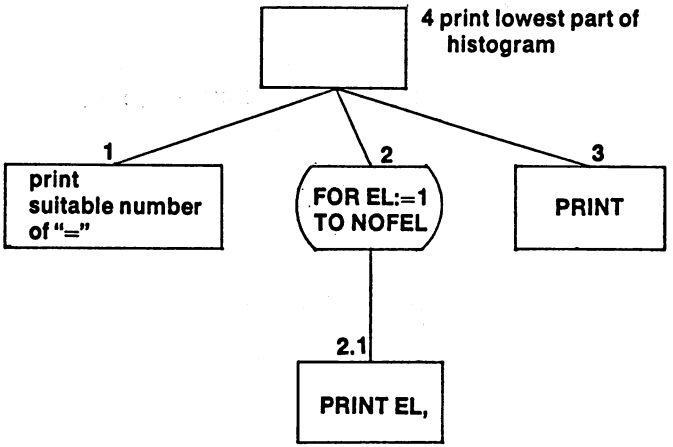
```
 50 PROC histo(REF vector(),nofel) CLOSED
 60 // version 1.3 19 Dec 1982
 70 // by Bill Compson
 80 //
 90   FOR i:=1 TO nofel DO
100     PRINT 1,TAB(8),":".
110     j:=vector(i)
120     FOR j1:=1 TO j DO
130       PRINT "*",
140     NEXT j1
150     PRINT "        ",j
170   NEXT i
```

```
180 ENDPROC histo
190 // ===== main program ======================
200 PRINT "*** testprogram ***"
210 INPUT "which dimension?": d
220 DIM testvector(d)
230 FOR i:=1 TO d DO
240    testvector(i):=RND(20,60)
250 NEXT i
260 EXEC histo(testvector,d)
```

Exercise 6.14

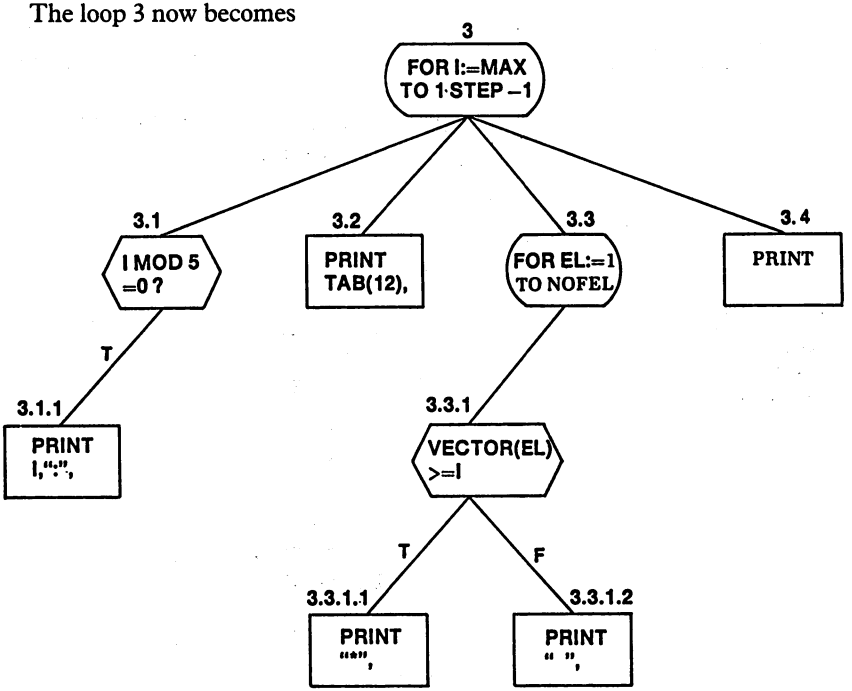


LARGEST is a procedure defined in Exercise 6.12.



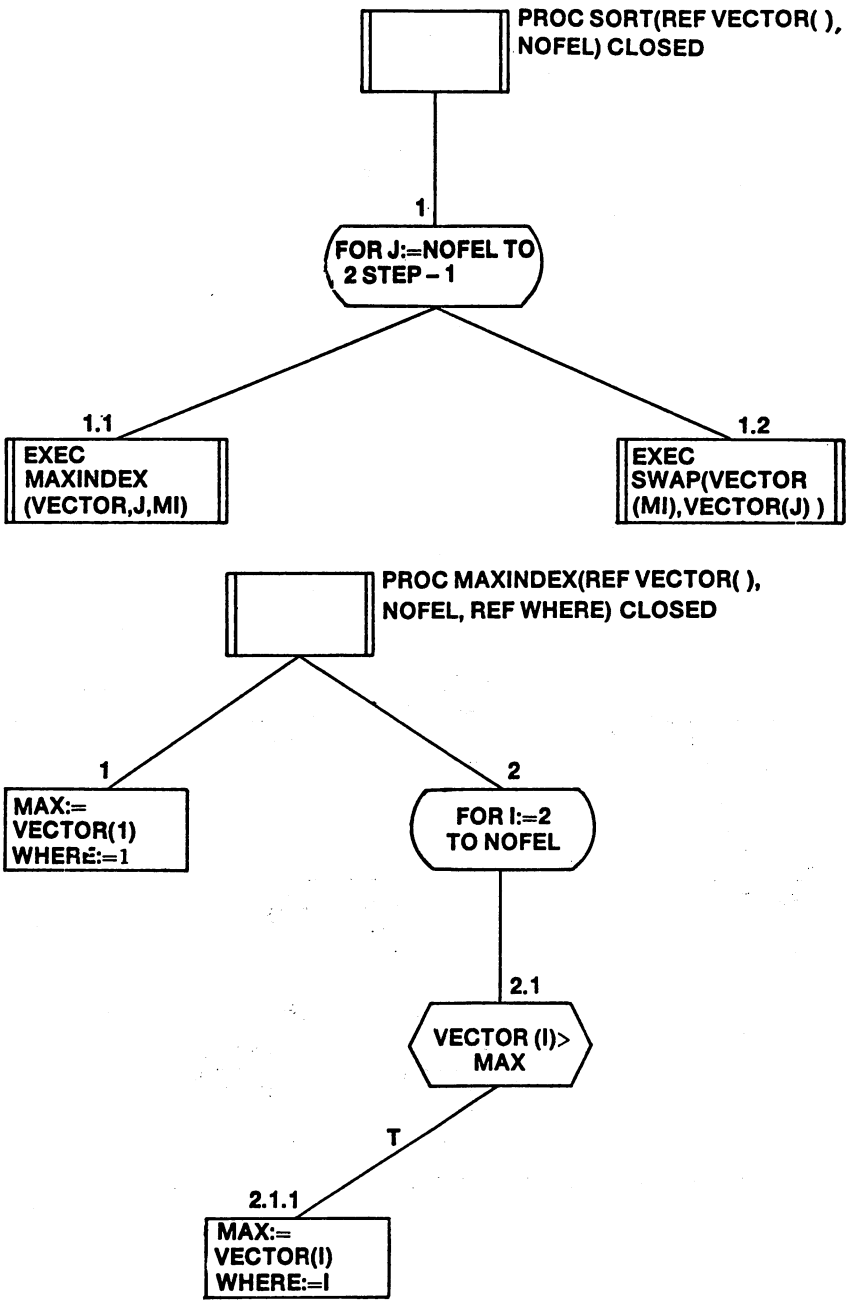
Exercise 6.15

The loop 3 now becomes



Box 4 must also be changed somewhat.

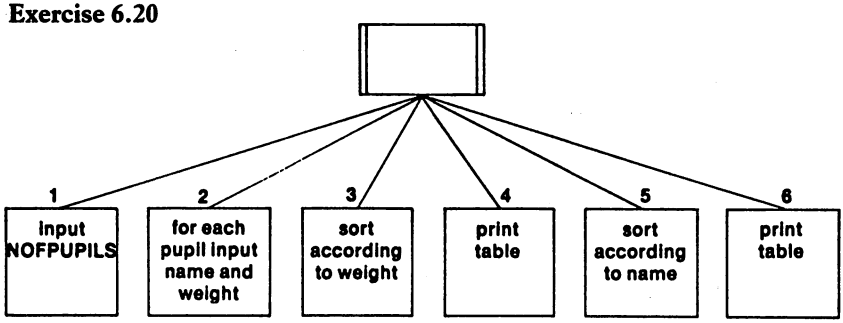
Exercises 6.18 and 6.19

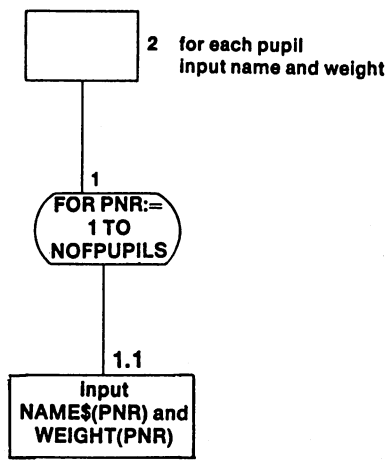


```
 50 PROC sort(REF vector(),nofel) CLOSED
 60   IMPORT maxindex.swap
 70   FOR j:=nofel TO 2 STEP -1 DO
 80     EXEC maxindex(vector,j,mi)
 90     EXEC swap(vector(j),vector(mi))
100   NEXT j
110 ENDPROC sort
120 //-------------------------------------
130 PROC maxindex(REF vector(),nofel,REF where) CLOSED
140   where:=1
150   max:=vector(1)
160   FOR j:=2 TO nofel DO
170     IF vector(j)>max THEN
180       max:=vector(j)
190       where:=j
200     ENDIF
210   NEXT j
220 ENDPROC maxindex
230 //-------------------------------------
240 PROC swap(REF x,REF y) CLOSED
250   g:=x
260   x:=y
270   y:=g
280 ENDPROC swap
290 //-------------------------------------
300 PROC prout(REF vector(),nofel) CLOSED
310   t:=ZONE
320   ZONE 10
330   FOR 1:=1 TO nofel DO
340     PRINT vector(i).
350   NEXT i
360   ZONE t
370 ENDPROC prout
380 //===== main program ====================
390 INPUT "which dimension?": d
400 DIM v(d)
410 FOR i:=1 TO d DO
420   v(i):=RND(10,99)
430 NEXT i
440 PRINT
```

```
450 PRINT "the vector before sorting: "
460 EXEC prout(v,d)
470 PRINT
480 PRINT "the vector after sorting: "
490 EXEC sort(v,d)
500 EXEC prout(v,d)
```

Exercise 6.20





Exercise 6.21

Set NOFQUESTIONS:=12. There must be a vector CORRECT with a dimension equal to NOFQUESTIONS, i.e. there must be a line

```
DIM CORRECT(NOFQUESTIONS)
```

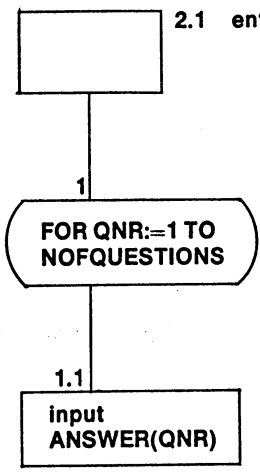
The box initialize also determines NOFPUPILS and contains a

```
DIM SCORE(NOFPUPILS)
```

In the initialization the vector CORRECT should be given its contents in this way:

```
FOR QNR:=1 TO NOFQUESTIONS DO
  PRINT "WHAT IS THE CORRECT ANSWER TO QUESTION NR ",
  QNR,   INPUT CORRECT(QNR)
NEXT QNR
```

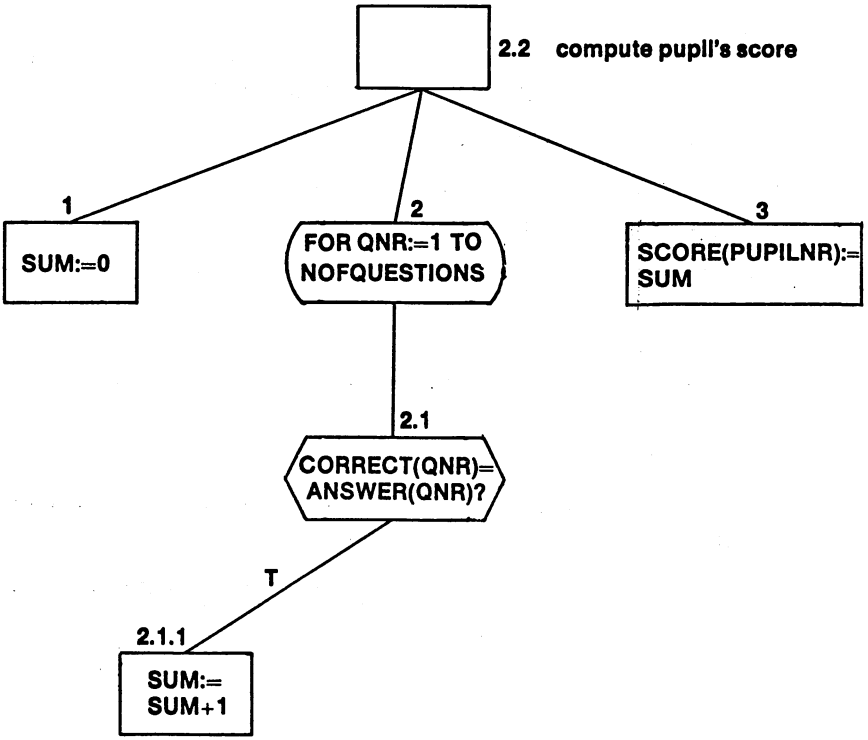
The results of one pupil are stored (temporarily) in the vector ANSWER, with NOFQUESTIONS number of elements.



In this way the answers are entered one by one. A smarter way is perhaps this:

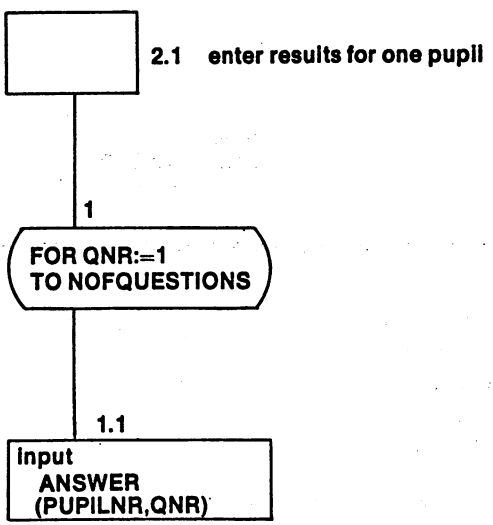
```
PRINT "ENTER RESULTS FOR ALL QUESTIONS. RETURN AFTER
EACH ANSWER."
FOR QNR:=1 TO NOFQUESTIONS DO
  INPUT ANSWER(QNR),
NEXT QNR
```

Note the comma in the input statement.



Exercise 6.22

Box 2.1 now becomes



Exercise 6.26

```
WHICH:=0
FOR I:=1 TO NOFPRODUCTS DO
. . . . . .
```

Exercise 6.28

```
PROC FINDPRICE(Q$,REF WHICHPRICE) CLOSED
  IMPORT SEARCH
  EXEC SEARCH(Q$,WHICH)
  IF WHICH=0 THEN
    WHICHPRICE:=0
  ELSE
    WHICHPRICE:=PRICE(WHICH)
  ENDIF
ENDPROC FINDPRICE
```

Note that in some systems there is no need to import procedures.

(Solutions below which involve EOF are for the first-mentioned type of system.)

Exercise 7.1

Enter the lines

```
 55 IF NOT EOF(3) THEN
115 ENDIF
```

or

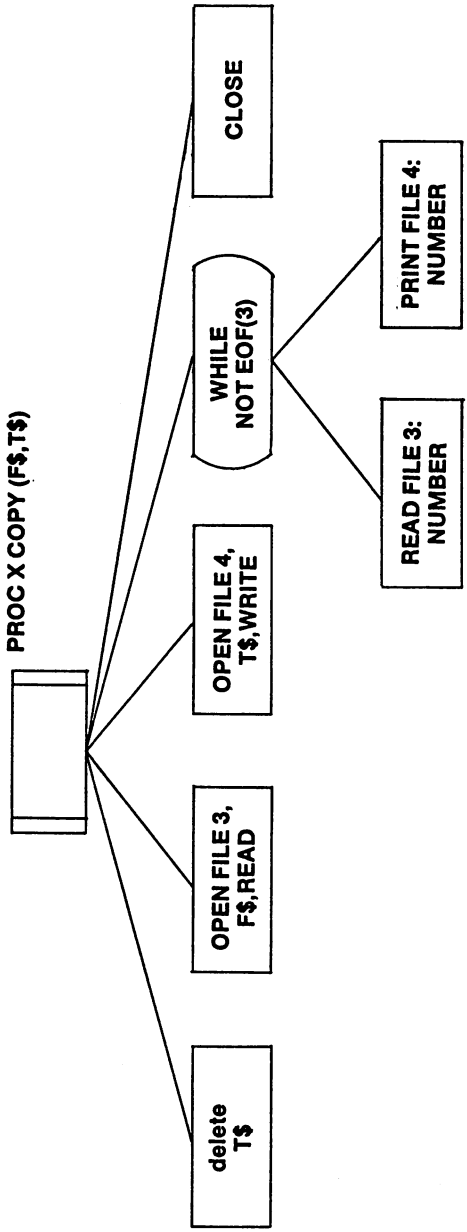
```
 55 IF EOF(3) THEN
 56    PRINT "EMPTY FILE!"
 57 ELSE
115 ENDIF
```

Exercise 7.5

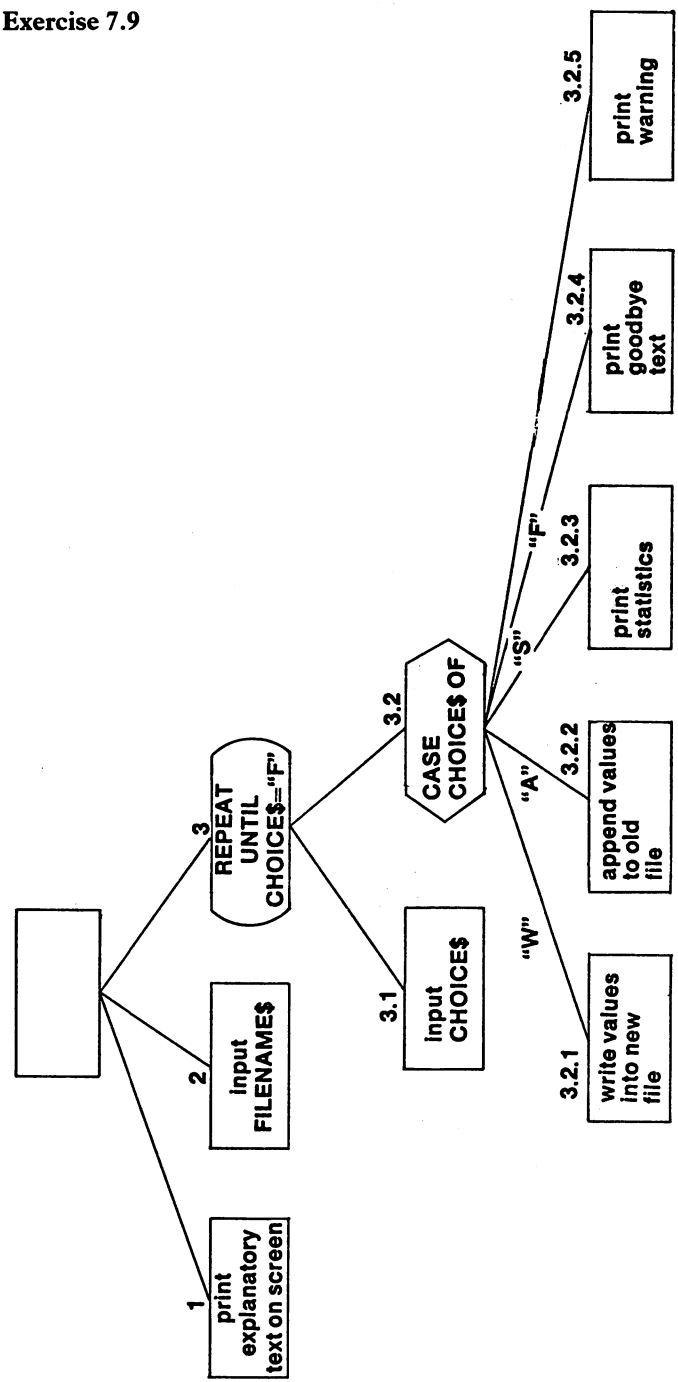
Example: Modification of Program APPENDTOSEQDEMO (Figure 7.3):

```
45 DIM FILENAME$ OF 8
46 INPUT "WHICH FILE?": FILENAME$
50 OPEN FILE 3,FILENAME$,APPEND
70 PRINT "NAMED",FILENAME$
```

Exercise 7.6


PROC X COPY (F$,T$)
delete T$
OPEN FILE 3, F$,READ
OPEN FILE 4, T$,WRITE
WHILE NOT EOF(3)
READ FILE 3: NUMBER
PRINT FILE 4: NUMBER
CLOSE

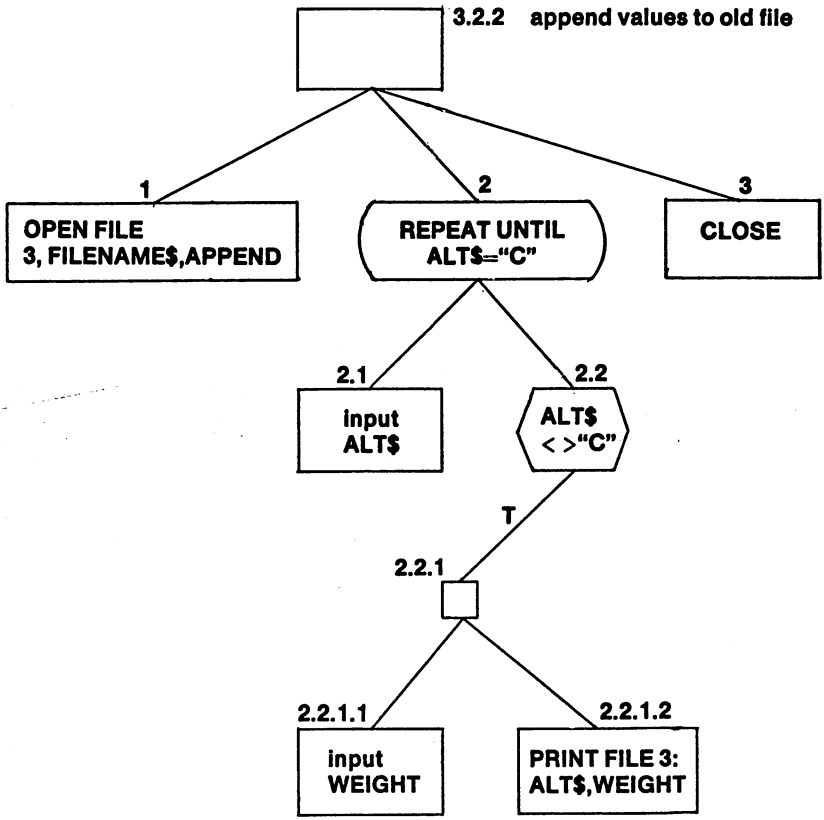
Exercise 7.9



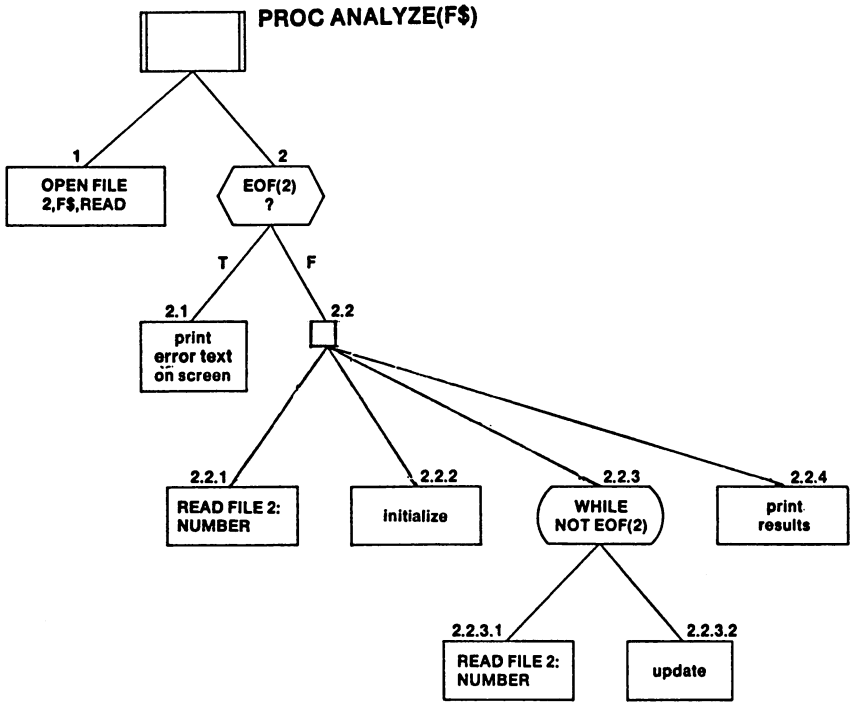
Perhaps it would be better to integrate boxeds 3.2.1 and 3.2.2. Why? How would you do this?

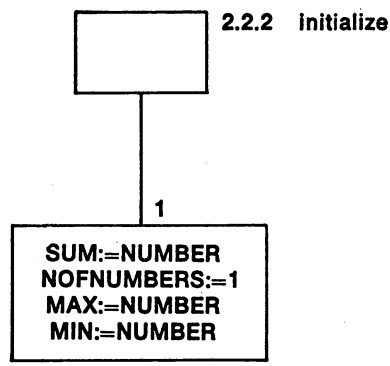
When adding new values use the question – just as you did earlier:

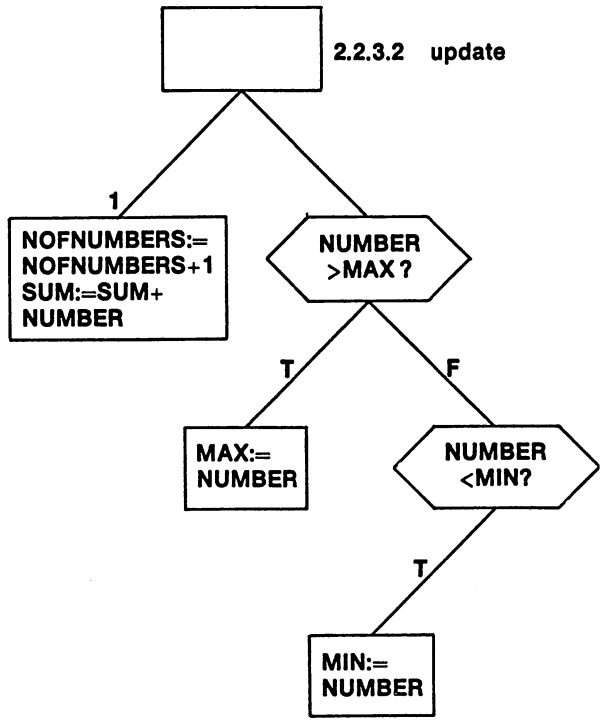
WHICH SEX – M(ALE) OR F(EMALE) – ENTER C(LEAR) TO FINISH?



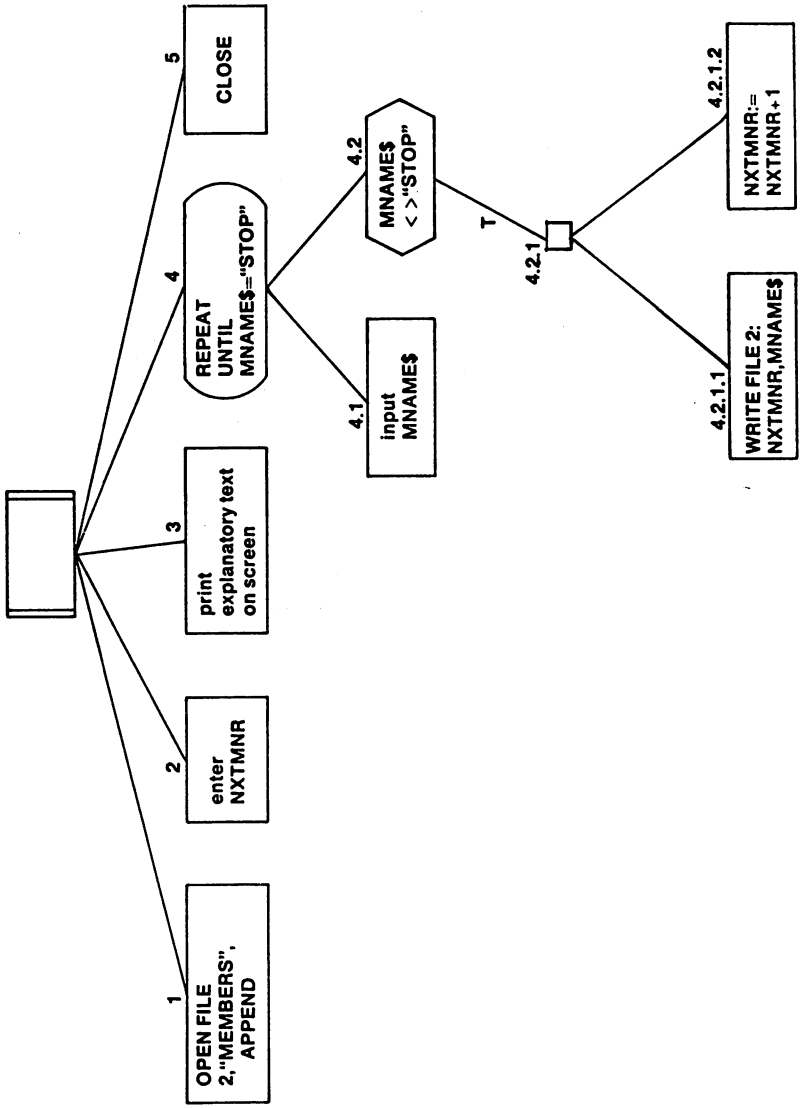
Exercise 7.11



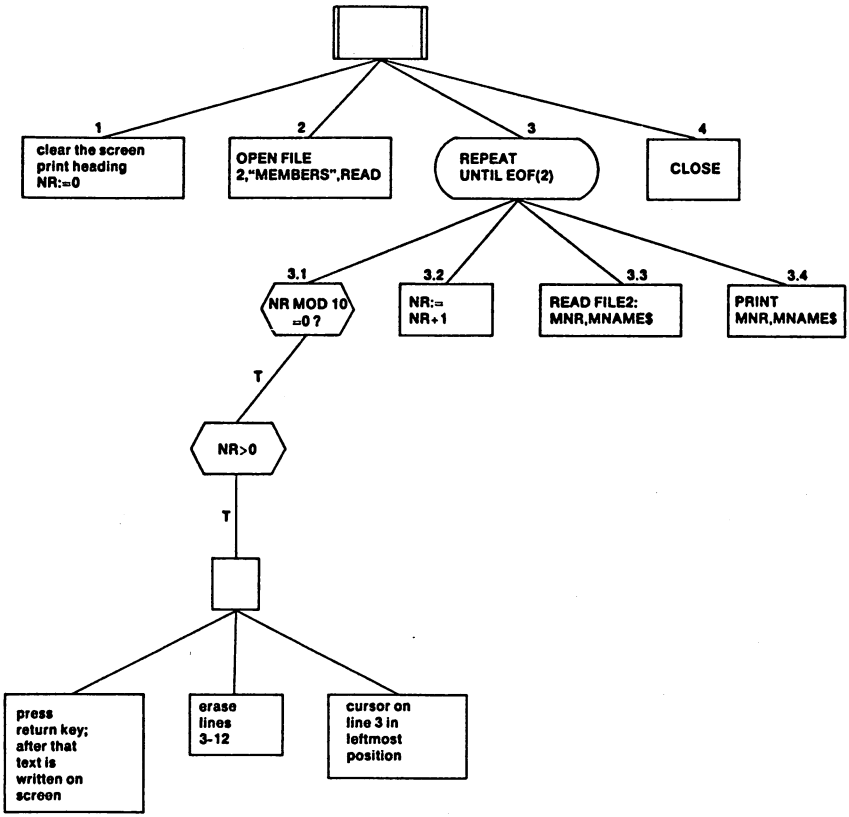



2.2.3.2 update
1
NOFNUMBERS:=
NOFNUMBERS+1
SUM:=SUM+
NUMBER
NUMBER
>MAX?
T
F
MAX:=
NUMBER
NUMBER
<MIN?
T
MIN:=
NUMBER

Exercise 7.12



Exercise 7.15



Exercise 7.17

```
PROC SPLIT(NAME$,REF SURNAME$,REF FORENAME$) CLOSED
  // The problem is to find where the last blank inside the name is
  L:=LEN(NAME$)
  I:=L
  REPEAT
    I:=I-1
  UNTIL NAME$(I:I)=" "
  FORENAME$:=NAME(1:I-1)
  SURNAME$:=NAME(I+1:L)
ENDPROC SPLIT
```

Exercise 7.19

(b) MID$(A ,FROM,HOWMANY) equals A$(FROM:FROM+HOWMANY-1)
(c) A$(L-N+1:L)

Exercise 7.22 : Hint

If you want to enter the digit 3

```
x x x
    x
x x x
    x
x x x
```

use a procedure THREE(COLNR,ROWNR) to enter *x* at the right places in MARK$. Use STR$ to convert a number to a string and after that examine each digit in the string.

Simple Educational Programs Project: Hint

In one sequential file, for instance called MAINFILE , you have

```
"NAME THE CAPITAL OF THE FOLLOWING COUNTRIES","CAPFILE",
"NAME THE GOALKEEPER OF THE FOLLOWING TEAMS","TEAMFILE"
```

and so on.

CAPFILE contains for instance:

```
"DENMARK","COPENHAGEN","SWEDEN","STOCKHOLM",. . . .
```

Suppose that you have chosen for instance CAPFILE. Then you will for instance have the following

```
QUESTION$(1)="DENMARK"
ANSWER$(1)="COPENHAGEN"
QUESTION(2)="SWEDEN" . . .
.
.
.
QUESTION$(20)="UNITED STATES"
ANSWER$(20)="WASHINGTON"
```

In this case NOFALT=20.

```
FOR I:=1 TO NOFALT DO
  select one question not asked before
  ask it
  mark that it has been asked
NEXT I
```

Use a procedure like CHOOSE below to select the next question.

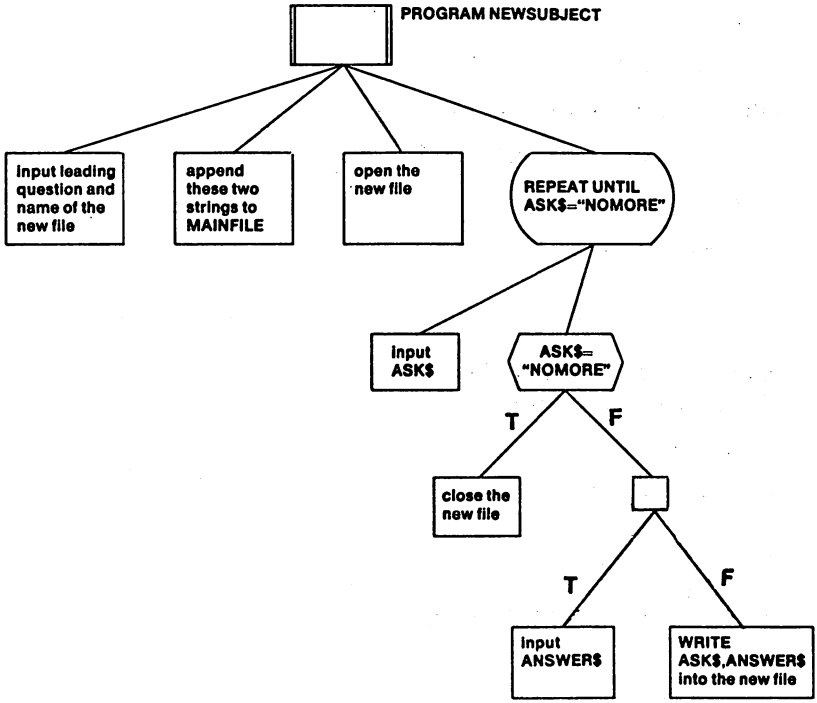
```
PROC CHOOSE(REF WHICH)
  REPEAT
    WHICH:=RND(1,NOFALT)
  UNTIL MARK(WHICH):=0
ENDPROC CHOOSE
```

MARK is a vector of dimension NOFALT.

You should use another program to enter a new subject to ask questions about. For instance like this:

```
WHICH LEADING QUESTION? . . .
NAME OF FILE? . . .
```

These two strings are appended to MAINFILE.


PROGRAM NEWSUBJECT
Input leading question and name of the new file
append these two strings to MAINFILE
open the new file
REPEAT UNTIL ASK$="NOMORE"
input ASK$
ASK$= "NOMORE"
T
F
close the new file
T
F
input ANSWER$
WRITE ASK$,ANSWER$ into the new file

Exercise 8.3

```
OPEN FILE 7,"NXTMNR",RANDOM 12
```

At the beginning of the program use the statement

```
READ FILE 7,1,1:NXTMNR
```

and at the end, use

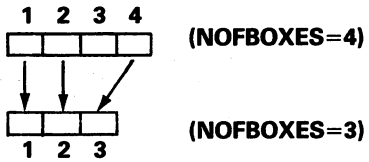
```
WRITE FILE 7,1,1:NXTMNR
```

Telephone Directory Project: Hint
You should define procedures:

```
DELBOX(I)
```

which deletes box nr *I*

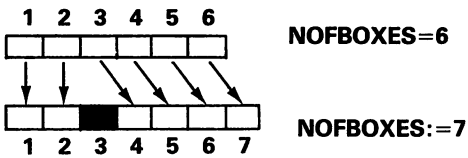
Example: DELBOX(3)




```
INSERTBOX(NAME$,TELNR$)
```

First decide WHERE it should be inserted.

Example: WHERE=3 NOFBOXES=6




```
UPDATE(NAME$,TELNR$)
```

changes the telephone number for the individual with name NAME$ to TELNR$.
With both DELBOX and INSERTBOX you have good use of a procedure

```
MOVEBOX(I,J)
```

which moves the contents of box I to box J.
All these procedures can and should be made *closed*.

This can be solved in many ways. Make certain that your program is user-friendly!
It could circle round a menu like this:

```
-D(ELETE)
-N(EW)
-C(HANGE)
-S(TOP)
```

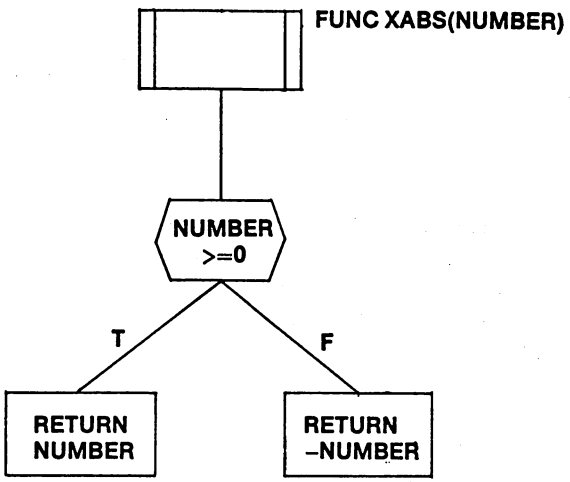
which alternative?

Pseudocode for part of the solution would look like this (enter NOFBOXES):

```
REPEAT
  choose ALT$
  CASE ALT OF
  WHEN "D" DO
    enter NAME$
    EXEC SEARCH(NAME$,WHICHBOX)
    IF WHICHBOX>0 THEN
      EXEC DELBOX(WHICHBOX)
    ELSE
      print warning
    ENDIF
  WHEN "N" DO
    enter NAME$ and TELNR$
    EXEC INSERTBOX(NAME$,TELNR$)
  WHEN "C" DO
    enter NAME$
    EXEC SEARCH(NAME$,WHICHBOX)
    IF WHICHBOX>0 THEN
      enter TELNR$
      change tel.nr. in box nr WHICHBOX
    ELSE
      print warning
    ENDIF
  WHEN "S" DO
    move NOFBOXES to its own place
    (for instance a file of its own)
    close all files
  OTHERWISE
    PRINT "**unrecognized command**"
    PRINT "TRY AGAIN!"
  ENDCASE
UNTIL ALT$="S"
```

Exercise 8.9

```
FUNC XABS(NUMBER) CLOSED
  IF NUMBER>=0 THEN
    RETURN NUMBER
  ELSE
    RETURN -NUMBER
  ENDIF
ENDFUNC XABS
```



Exercise 8.10

```
FUNC XRND(REF R) CLOSED
  P:=13
  R:=R*P
  R:=R MOD 1000
  RETURN R/1000
ENDFUNC XRND
```

Exercise 8.11

```
FUNC TRND(I,J,REF R) CLOSED
  IMPORT XRND
  RETURN I+INT(J-I+1)*XRND(R))
ENDFUNC TRND
```

Exercise 8.15

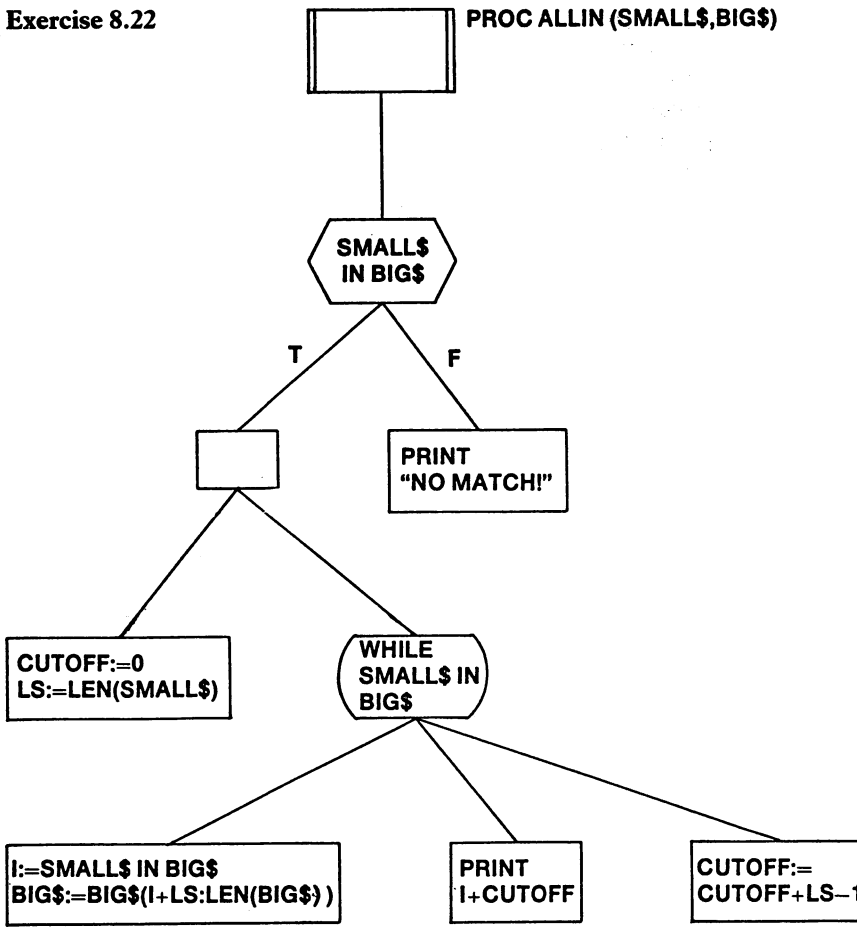
```
FUNC OK(NUMBER) CLOSED
  IF INT(NUMBER)<>NUMBER OR NUMBER<=0 THEN
```

```
    RETURN FALSE
  ELSE
    RETURN TRUE
  ENDIF
ENDFUNC OK
```

Exercise 8.16

```
FUNC BOUND(LOW,HIGH,NUMBER) CLOSED
  IF LOW<=NUMBER AND NUMBER<=HIGH THEN
    RETURN TRUE
  ELSE
    RETURN FALSE
  ENDIF
ENDFUNC BOUND
```

Exercise 8.22



Test the procedure manually with SMALL$ ="CO" and BIG$ = "COCONUTCOAL".

Exercise 8.23

```
FUNC MYIN(SMALL$,BIG$) CLOSED
  DIM S$ OF 1
  S$:=SMALL$(1:1)
    LB:=LEN(BIG$)
    LS:=LEN(SMALL$)
  J:=0
  FOR I:=1 TO LB DO
    IF S$=BIG$(I:I) THEN
      IF SMALL$=BIG$(I:I+LS-1) THEN
        J:=I
        GOTO DONE
      ENDIF
    ENDIF
  NEXT I
  DONE:
  RETURN J
ENDFUNC MYIN
```

STANDARD COMAL (The COMAL Kernal)

LIST OF COMMANDS

		Page
ABS	absolute value of a number	21
AND	logical operator	141
APPEND	file command	115
ATN	mathematical function – not mentioned	
AUTO	interactive command (i.e. not in programs)	3
CASE	branching construction, used together with ENDCASE, WHEN, OTHERWISE	38
CAT	interactive command to display the contents of secondary memory	8
CHR$	converts an integer to a string with one character; also used in commands for example to erase the screen, and in some systems can be used to produce semigraphic symbols	126
CLOSE and CLOSE FILE	file command	113
CLOSED	a procedure or function can be made closed to avoid side-effects	68
CON	resumes program execution after a STOP	57
COS	mathematical function – not mentioned	
DATA	stores information inside a program	131
DEL	interactive command to delete program line(s) in the internal memory	6
DELETE	deletes file(s) on secondary storage	16
DIM	reserves space in the internal memory	
	strings	18
	one-dimensional arrays	90
	two-dimensional arrays	105
	string arrays	90
DISPLAY	displays the specified program lines without line numbers – not mentioned	
DIV	mathematical function	43
ENTER	moves programs, procedures, functions etc, stored with LIST to the internal memory	96
EOD	end of data – used in connection with DATA – statements and READ	132

EOF	a condition that is true when all items in a sequential file have been read	113
EXP	mathematical function – not mentioned	
FALSE	another way to represent 0 that makes programs more readable	140
FOR	. . . NEXT control structure (in some systems FOR . . . ENDFOR	47
FUNC	. . . ENDFUNC function definition	138
GOTO	to jump forward in programs (only to labels)	107
IF	. . . THEN etc. (ELIF,ELSE,ENDIF)	25
IMPORT	for using procedures and external variables inside a closed procedure	70
IN	function that checks whether and where one text string is contained in another	142
INPUT	to enter values from the keyboard	11
INPUT FILE	not mentioned	
INT	mathematical, function – returns the integer part of a number	20
LABEL	lines can be labelled using a label	107
LEN	returns the length of a string	52
LIST	interactive command – to list part or the whole of a program on screen or paper	3
	or onto secondary storage	94
LOAD	interactive command – to read a program (or part of a program) stored by a SAVE command into the internal memory	8
LOG	mathematical, function – not mentioned	
MOD	mathematical, function	21
NEW	interactive command – to clean up the internal memory from program lines	3
NOT	logical operator	141
OPEN FILE	opens a file	113
OR	logical operator	141
ORD	the "inverse" of CHR$ – not mentioned	
PRINT	to print on screen or paper	1
PRINT FILE	not mentioned	
PRINT USING	not mentioned	
PROC	. . . ENDPROC procedure definition	66
READ	to read from DATA statements	131
READ FILE	to read from files created by WRITE FILE/ APPEND FILE	113
RENUM(BER)	interactive command – to renumber program lines in the internal memory	5
REPEAT	. . . UNTIL control structure	54
RESTORE	in connection with DATA and READ	131

RND	random number generator(s)	80
RUN	interactive command – to start the execution of the program now in the internal memory	1
SAVE	stores the program now in the internal memory on secondary storage	7
SELECT OUTPUT	specifies where PRINT shall take place	8
SGN	mathematical function, returns the sign of a number	21
SIN	mathematical, function – not mentioned	
SIZE	to display the size of the used part of the internal memory only as a direct command – not mentioned	
SQR	mathematical, function (square root)	20
STOP	stops the execution of a program	57
STR$	converts an arithmetic expression into a string	63
TAB	used in PRINT statements	15
TAN	mathematical, function – not mentioned	
TRUE	another way to represent 1 that makes programs more readable	140
VAL	is the "inverse" of STR$	63
WHILE	... ENDWHILE control structure	58
WRITE FILE	to write data into a file	114, 133
ZONE	the width of print zones	48

Index

ABS, 21
Ada, 144
AND, 141
APPEND, to a seq. file, 115
Arrays, 89 *ff*
    numeric, 90
    string, 90
    two-dim. numeric, 103
    two-dim. strings, 127
Assignment, 2
AUTO, 3, 6

Basic, 111–112, 125
Boolean function, 142

CASE, 38 *ff*
CAT, 8
Channel number, 114
CHRS, 126 *ff*
CLOSE (FILE), 113, 114, 133
CLOSED, 68
Comal interpreter, 1
Comment (in programs), 5
CON, 58
CREATE, 133
CURSOR, 118

DATA, 131 *ff*
DEL, 6
DELETE, 16
DIM
    strings, 18
    arrays, 90
    string arrays, 90
    two-dimensional arr., 103, 127
DIR, 8
Direct access file, *see* Random file
DIV, 43

EDIT, 4
ELIF, 30, 33
ELSE, 26, 33
ENDCASE, 38
ENDFOR, 47 (*foot note*)
ENDFUNC, 138
ENDIF, 25, 33
ENDPROC, 66
ENDWHILE, 58
ENTER, 96
EOD, 132
EOF, 113 *ff*
EXEC, 65 *ff*
EXTERNAL procedure, 112

File
direct access, *see* Random files
random, *see* Random files
sequential, *see* Sequential files
Filename, rules for, 7
FOR, 47 *ff.*, 60
Formal parameter, 66
FUNC, 138
Function
built-in, 19 *ff*
other, 137 *ff*

GLOBAL, 71
GOTO, 107

Histogram, 97 *ff*

IF, 25 *ff.*, 33
IMPORT, 70
IN, 142
INPUT, 11 *ff*
INT, 20, 42 *ff*

LABEL, 107
LEN, 18
LIST
editing command, 3, 6 *ff*
into sec. memory, 94
LOAD, 8
LOOKUP, 8
Loops, 47

Maximum, 60 *ff*
MERGE, 95
Minimum, 60 *ff*
MOD, 21, 42

NEW, 3
NEXT, 47
NOT, 141

OPEN FILE
sequential file, 113, 114
random file, 132
OR, 141
OTHERWISE, 40
OUTPUT, *see* SELECT

Pascal, 144
PRINT, 1, 11 *ff*
PRINT AT, 118
PROC, 66
Procedure, 65 *ff*
Procedure heading, 66
Pseudocode, 110

Random numbers, 80 *ff.*, 139
Random files, 132 *ff*
RANDOM(IZE), 80
READ from DATA statements, 131
READ FILE
sequential file, 113 *ff*
random file, 133
REF, 77 *ff.*, 92
REM, *see* Comment
RENUM(BER), 5
REPEAT, 54 *ff.*, 60
RESTORE, 131
RND, 80 *ff*
RUN, 1

SAVE, 7
SELECT (OUTPUT), 8
Sequential files, 113 *ff*
SGN, 21
Side-effect, 68
Sort, 100 *ff*
SQR, 20
STEP, 50
STOP, 57
String
introduction, 18 *ff*
comparison, 63
join, 19 (Ex. 2.13)
length (LEN), 18
matching, 142–143
substring, 124
commands in Basic, 125

STRS, 63 (*foot-note*)
Structure diagram, 28 *ff*

TAB, 14–15
Top-down design, 144
TRUE, 140

UNTIL, 54

VAL, 63 (*foot-note*)
Variable, 1

WHEN, 38
WHILE, 58 *ff.*, 60
WRITE FILE
    sequential file, 114, 115
    random file, 133

ZONE, 48 *ff*

COMAL is a programming language that is easy to use but at the same time is very powerful.

STARTING WITH COMAL provides a self-contained introduction to programming. It is built around a large number of examples and exercises, and gives the reader a working knowledge of how to solve problems using a computer. Structured programming methods are used throughout.

The variations between different versions of COMAL-80 are very small. This book is suitable for all versions.

Ingvar Gratte is a lecturer at the University of Uppsala, Sweden. He has many years' experience of teaching computer science and programming, at both school and university level.

Prentice/Hall PHI International

ISBN 0-13-843003-9