UMAP

MODULES AND MONOGRAPHS IN UNDERGRADUATE MATHEMATICS AND ITS APPLICATIONS

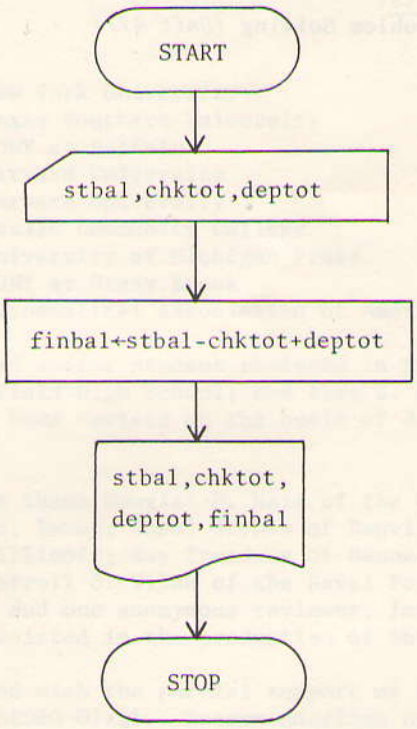
Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω α ϑ

Birkhäuser Boston Inc.
380 Green Street
Cambridge, MA 02139

MODULE 478

Iteration and Computer Problem Solving

Herbert L. Dershem



Computer Science/Algorithms

Intermodular Description Sheet: UMAP Unit 478

Title: ITERATION AND COMPUTER PROBLEM SOLVING

Author: Herbert L. Dershem
Department of Computer Science
Hope College
Holland, Michigan 49423

Review Stage/Date: IV 6/30/81

Classification: COMPUTER SCI/ALGORITHMS

Prerequisite Skills:
1. Completion of UMAP Unit 477, "Computer Problem Solving."

Output Skills:
1. Be able to improve algorithms by enhancing them.
2. Be able to read, interpret, and follow through (as a computer would) algorithms that involve while, until, and variable-controlled iteration.
3. Be able to describe and apply the top-down appraoch to algorithm design.

Related Units:
Computer Problem Solving *(Unit 477)*

MODULES AND MONOGRAPHS IN UNDERGRADUATE

MATHEMATICS AND ITS APPLICATIONS PROJECT (UMAP)

The goal of UMAP is to develop, through a community of users and developers, a system of instructional modules in undergraduate mathematics and its applications that may be used to supplement existing courses and from which complete courses may eventually be built.

The Project is guided by a National Advisory Board of mathematicians, scientists, and educators. UMAP is funded by a grant from the National Science Foundation to Education Development Center, Inc., a publicly support, nonprofit corporation engaged in educational research in the U.S. and abroad.

PROJECT STAFF

Ross L. Finney	Project Director
Solomon A. Garfunkel	Executive Director, COMAP
Felicia M. DeMay	Associate Director
Barbara Kelczewski	Coordinator for Materials Production
Paula M. Santillo	Assistant to the Directors
Donna DiDuca	Project Secretary/Production Assistant
Janet Webber	Word Processor
Zachary Zevitas	Staff Assistant

UMAP ADVISORY BOARD

Steven J. Brams	New York University
Llayron Clarkson	Texas Southern University
Donald A. Larson	SUNY at Buffalo
R. Duncan Luce	Harvard University
Frederick Mosteller	Harvard University
George M. Miller	Nassau Community College
Walter Sears	University of Michigan Press
Arnold A. Strassenburg	SUNY at Stony Brook
Alfred B. Willcox	Mathematical Association of America

This unit was field-tested and/or student reviewed in preliminary form by Harold Baker of Litchfield High School; and Alex G. Bennett of Bremerton High School and has been revised on the basis of data received from these sites.

The Project would like to thank Douglas F. Hale of the University of Texas-Permian Basin, Odessa, Texas; Carol Stokes of Danville Area Community College, Danville, Illinois; Ray Treadway of Bennett College, Greensboro, North Carolina; Carroll O. Wilde of the Naval Postgraduate School, Monterey, California; and one anonymous reviewer, for their reviews, and all others who assisted in the production of this unit.

This material was prepared with the partial support of National Science Foundation Grant No. SED80-07731. Recommendations expressed are those of the author and do not necessarily reflect the views of the NSF or the copyright holder.

ITERATION AND COMPUTER PROBLEM SOLVING

by

Herbert L. Dershem
Department of Computer Science
Hope College
Holland, Michigan 49423

TABLE OF CONTENTS

1. INTRODUCTION

One of the major advantages in using a computer for problem solving is that a process can be explained to the computer once and the computer can repeat that process as many times as is necessary to solve the problem. In fact, without this computers would be of limited use as problem solving machines because it generally takes longer to explain a process to the computer than to carry it out by hand. Repeated execution of a single set of instructions on a computer is often called *iteration*. The term "iteration" also refers to any single execution of a process that is carried out more than once; the sense in which the term is used should be clear from the context.

The module describes ways in which you can use computer iteration effectively in problem solving and, in addition, ways in which you can use two other forms of iteration in constructing the algorithm. The second form of iteration involves repeated tracing through of the problem solving steps, improving your algorithm with each iteration. We call this process *iterative improvement* in algorithm design. Each improvement of the algorithm might add features to the previous version.

The third form of iteration in problem solving is found in an approach to algorithm design called the *top-down approach*. This approach for carring out step 4 of the problem solving process (see Unit 477) might best be called *iterative refinement*. It differs from iterative improvement in that the algorithm is not changed at each iteration, but rather, more detail is provided.

2. A MORTGAGE PROBLEM

In order to illustrate the iteration technique, we shall solve the following problem: given an amount of

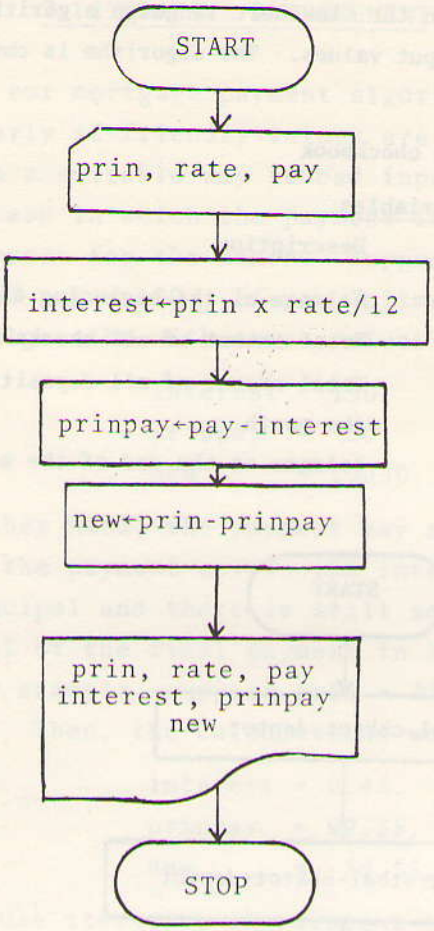
borrowed principal, a yearly interest rate, and a monthly payment, determine the new principal after one monthly payment has been made and determine the interest for one month.

The input for this algorithm consists of the principal at the beginning of the month, the yearly interest rate expressed as a decimal, and the amount of the payment. The output will consist of all values input plus the part of the payment which will be used to pay interest, the part of the payment which will be used to pay off the principal, and the new principal balance at the end of the month.

Algorithm 1. Mortgage payment - Version 1.

Variables

Name	Description
prin	The principal balance at the beginning of the month.
rate	The yearly rate of interest expressed as a decimal.
pay	The amount of the payment.
interest	The part of the payment which goes toward interest.
prinpay	The part of the payment which goes toward the principal.
new	The new principal balance at the end of the month.



The interest is calculated by multiplying the principle, prin, times the interest rate divided by 12 because the given rate is for a year and the period used is a month. The value of prinpay is the amount left from the payment pay after the interest is paid. Finally, new is the principal balance after prinpay is paid.

Exercises:

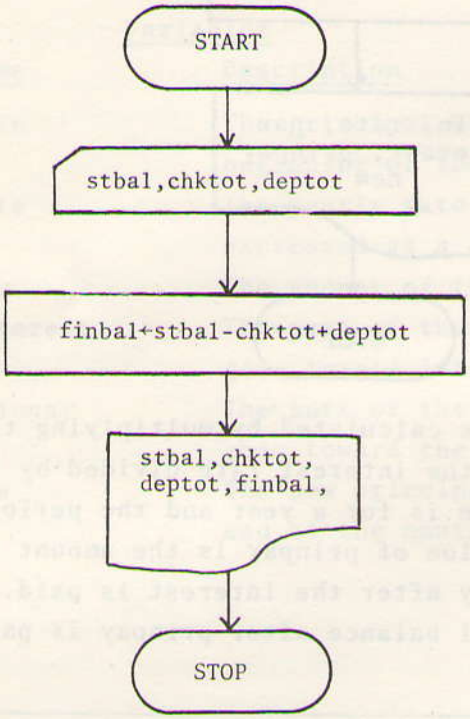
1. Choose sample values for prin, rate, and pay, and follow through algorithm 4 as a computer would.

2. Follow the steps in the flowchart language algorithm below for some sample input values. The algorithm is one to balance a checkbook.

Algorithm: Balancing a checkbook

Variables

Name	Description
stbal	Balance at the beginning of the month.
chktot	Total amount of all checks written.
deptot	Total amount of all deposits made during the month.
finbal	Balance at the end of the month.



3. ITERATIVE IMPROVEMENT

Although our mortgage payment algorithm 1 will function properly if friendly values are input, it will not respond in a suitable way to bad input. For example, consider the case in which the payment is too small to cover the interest for the month. Suppose prin = 20000., rate = .09, and pay = 100. If you follow through the steps of the algorithm with these values as input you get

```
interest = 150.
prinpay  = -50.
new      = 20050.
```

On the other hand, the payment may also be too large. In this case, the payment covers the interest and the remaining principal and there is still some extra left. This is typical of the final payment in a pay-back schedule. For example, suppose prin = 60., rate = .09, and pay = 100. Then, the calculations would be

```
interest = 0.45.
prinpay  = 99.55.
new      = -39.55.
```

We shall use iterative improvement to correct these two minor flaws in our original algorithm. If the payment is too small, we inform the user and terminate the program; if it is too large, we pay off the loan and notify the user that he or she is entitled to a refund.

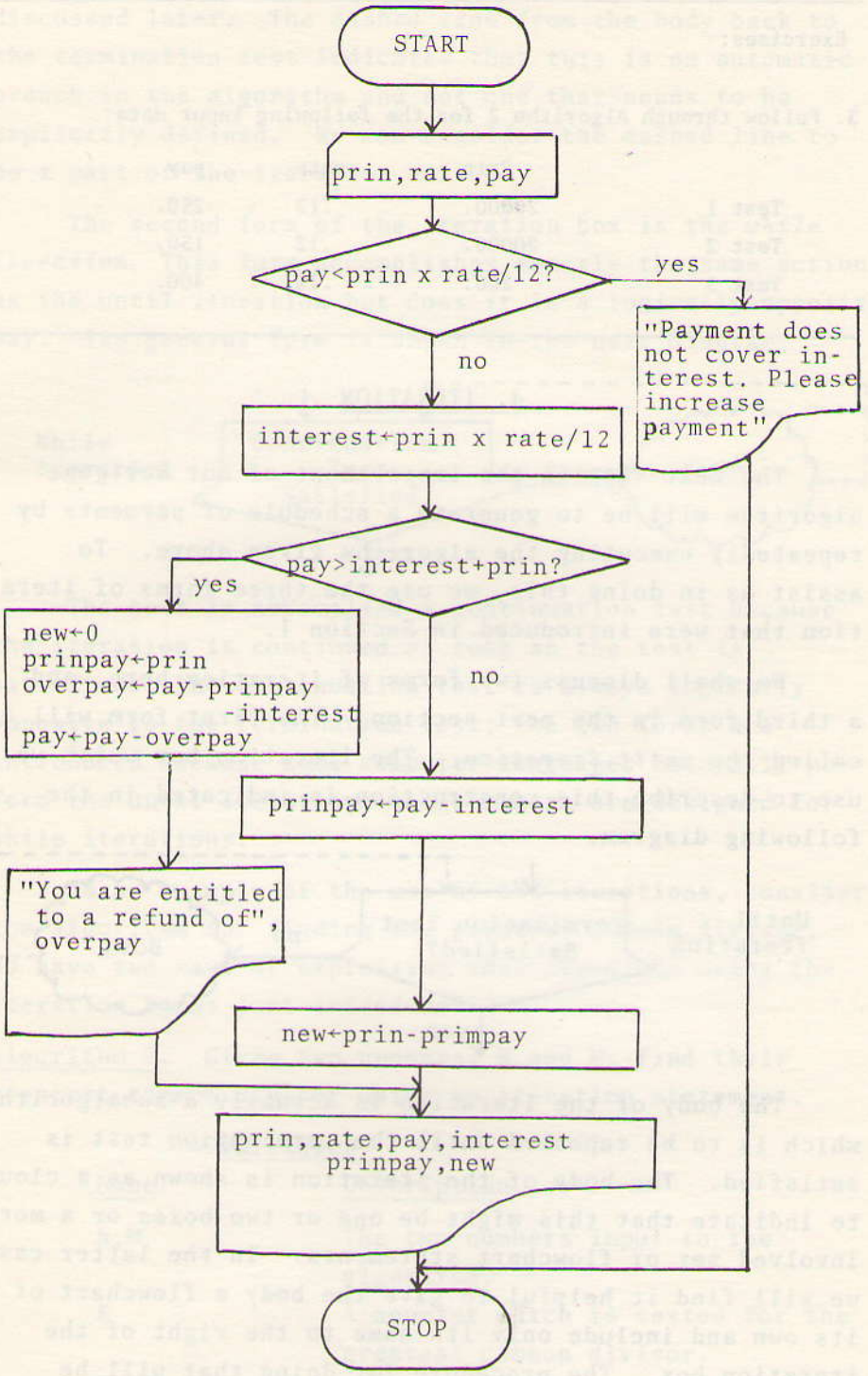
These modifications to Algorithm 1 are indicated in Algorithm 2. The decision box included immediately after the input box tests for a payment too small to cover the interest. When this is the case, we provide a message to the user and then immediately halt the algorithm. This action represents a policy of disallowing payments which are inadequate to pay the interest.

After the interest payment is calculated, we add another decision box which tests for payment greater than the principal. In this case, we calculate the new principal and amount paid on principal in a different way and adjust prinpay to exactly pay the remaining principal. We also include a message which notifies the user that he or she will receive a refund.

Algorithm 2. Mortgage payment - Version 2.

Variables

Name	Description
prin	The principal balance at the beginning of the month.
rate	The yearly rate of interest expressed as a decimal.
pay	The amount of the payment.
interest	The part of the payment which goes toward interest.
prinpay	The part of the payment which goes toward the principal.
new	The new principal balance at the end of the month.
overpay	The amount of overpayment for the final payment.


START
prin,rate,pay
pay<prin x rate/12?
yes
"Payment does not cover interest. Please increase payment"
no
interest←prin x rate/12
pay>interest+prin?
yes
new←0
prinpay←prin
overpay←pay-prinpay
-interest
pay←pay-overpay
no
prinpay←pay-interest
"You are entitled to a refund of",
overpay
new←prin-primpay
prin,rate,pay,interest
prinpay,new
STOP

Exercises:

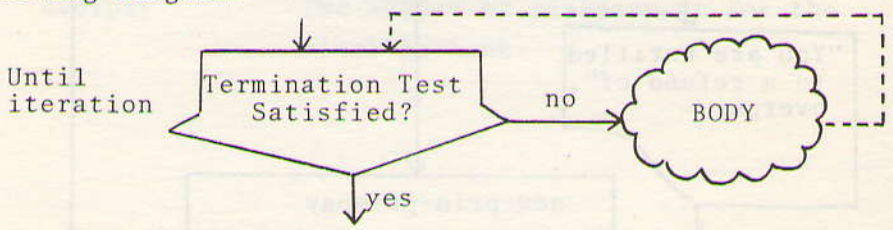
3. Follow through Algorithm 2 for the following input data:

	Prin	rate	pay
Test 1	20000.	.12	250.
Test 2	20000.	.12	150.
Test 3	200.	.12	400.

4. ITERATION

The next step in the improvement of our mortgage algorithm will be to generate a schedule of payments by repeatedly executing the algorithm given above. To assist us in doing this, we use the three forms of iteration that were introduced in Section 1.

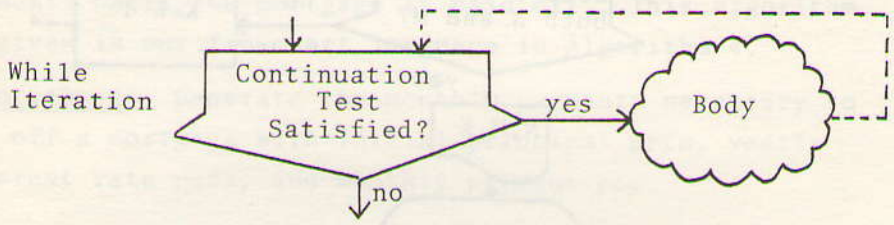
We shall discuss two forms of iteration here, and a third form in the next section. The first form will be called the *until iteration*. The iteration box which we use to describe this construction is indicated in the following diagram.



The body of the iteration is actually a subalgorithm which is to be repeated until the termination test is satisfied. The body of the iteration is shown as a cloud to indicate that this might be one or two boxes or a more involved set of flowchart statements. In the latter case, we will find it helpful to give the body a flowchart of its own and include only its name to the right of the iteration box. The procedure for doing that will be

discussed later. The dashed line from the body back to the termination test indicates that this is an automatic branch in the algorithm and not one that needs to be explicitly defined. We can consider the dashed line to be a part of the iteration box.

The second form of the iteration box is the *while iteration*. This form accomplishes exactly the same action as the until iteration but does it in a logically opposite way. Its general form is shown in the next diagram.



The test is now called a continuation test because the iteration is continued as long as the test is satisfied. The continuation test is always logically opposite to the termination test; the two forms are introduced because some computer languages naturally perform the until iterations while others are designed for while iterations.

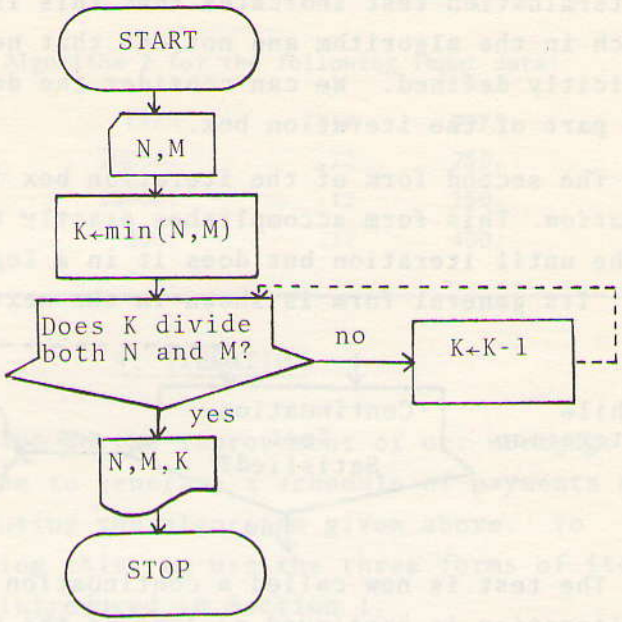
As an example of the use of the iterations, consider the algorithm for finding the greatest common divisor. We have two ways of expressing this algorithm using the iteration boxes just introduced.

Algorithm 9. Given two numbers, N and M, find their greatest common divisor using an iteration statement.

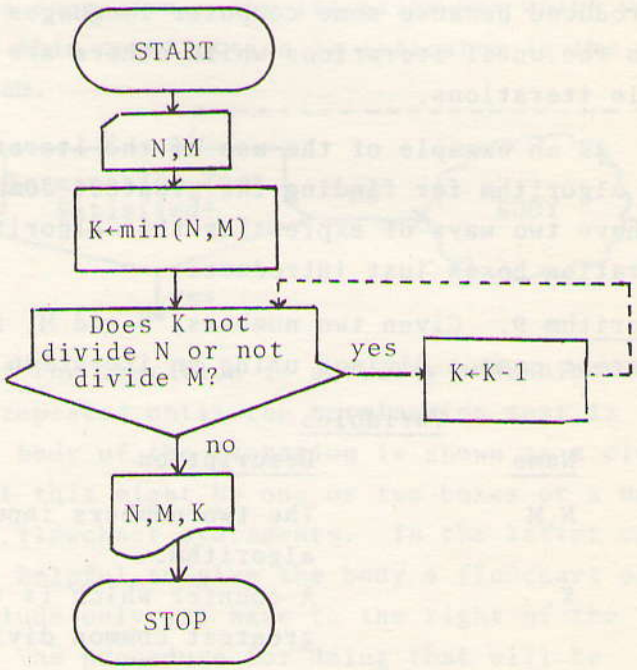
Variables

Name	Description
N,M	The two numbers input to the algorithm.
K	A counter which is tested for the greatest common divisor.

a. Using until iteration



b. Using while iteration



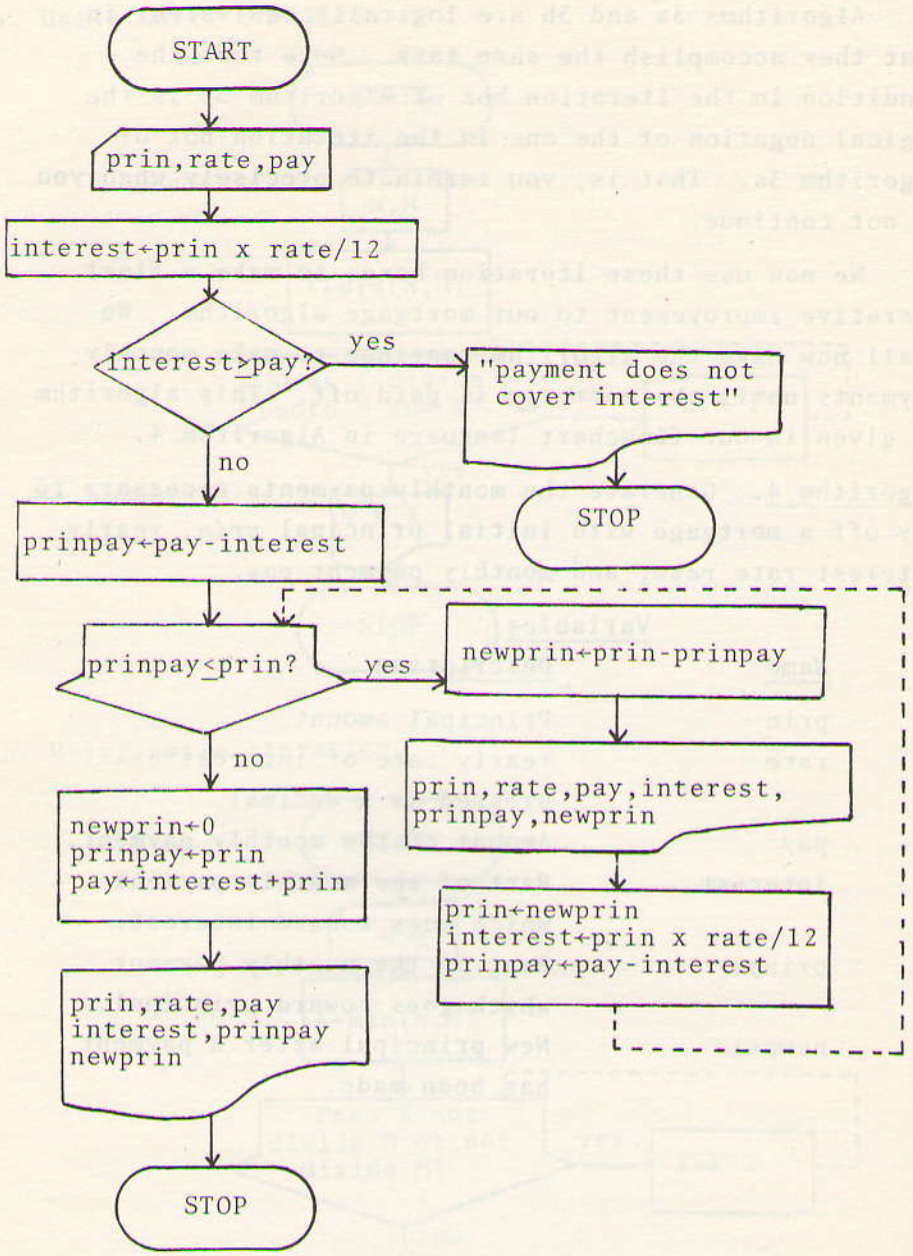
Algorithms 3a and 3b are logically equivalent in that they accomplish the same task. Note that the condition in the iteration box of Algorithm 3b is the logical negation of the one in the iteration box of Algorithm 3a. That is, you terminate precisely when you do not continue.

We now use these iteration boxes to make a final iterative improvement to our mortgage algorithm. We shall now have the algorithm continue to make monthly payments until the mortgage is paid off. This algorithm is given in our flowchart language in Algorithm 4.

Algorithm 4. Generate the monthly payments necessary to pay off a mortgage with initial principal *prin*, yearly interest rate *rate*, and monthly payment *pay*.

Variables

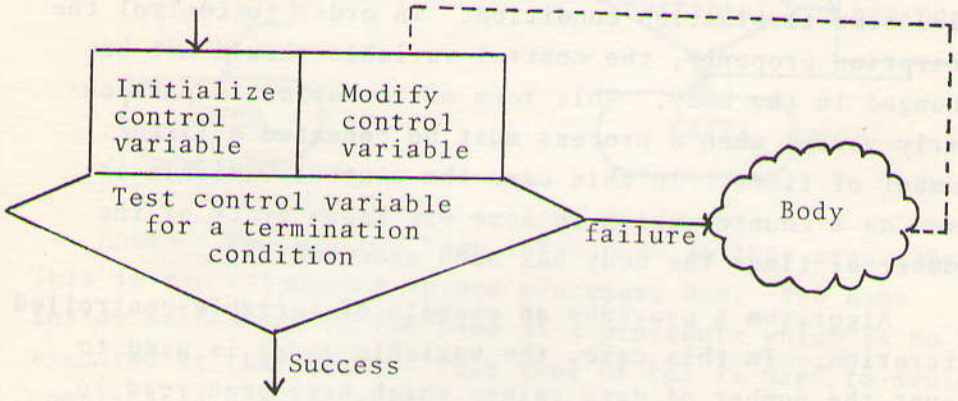
Name	Description
prin	Principal amount.
rate	Yearly rate of interest expressed as a decimal.
pay	Amount of the monthly payment.
interest	Part of the monthly payment which goes toward interest.
prinpay	Part of the monthly payment which goes toward principal.
newprin	New principal after a payment has been made.


START
prin,rate,pay
interest←prin x rate/12
interest≥pay?
yes
"payment does not cover interest"
STOP
no
prinpay←pay-interest
prinpay≤prin?
yes
newprin←prin-prinpay
prin,rate,pay,interest,
prinpay,newprin
prin←newprin
interest←prin x rate/12
prinpay←pay-interest
no
newprin←0
prinpay←prin
pay←interest+prin
prin,rate,pay
interest,prinpay
newprin
STOP

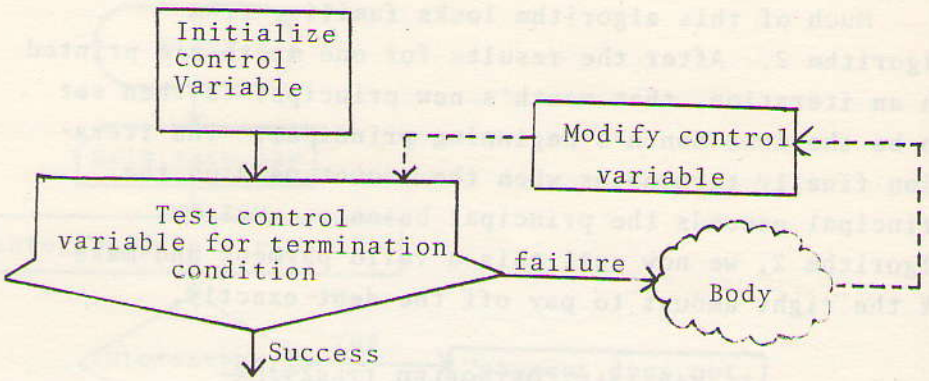
Much of this algorithm looks familiar from Algorithm 2. After the results for one month are printed in an iteration, that month's new principal is then set to be the next month's beginning principal. The iteration finally terminates when the amount paid on the principal exceeds the principal balance. Unlike Algorithm 2, we now call this a valid payment and make it the right amount to pay off the debt exactly.

5. VARIABLE-CONTROLLED ITERATION

In the last section we learned about forms of iteration which repeated a subalgorithm until termination conditions were met or continuation conditions were not. Now we introduce a slightly different form of iteration. This is iteration controlled by a variable. The general form of such an iteration is



It should be noted that this form of iteration is a special case of the form we studied in the preceding section. This iteration can be stated in terms of the previous one as indicated in the following diagram.

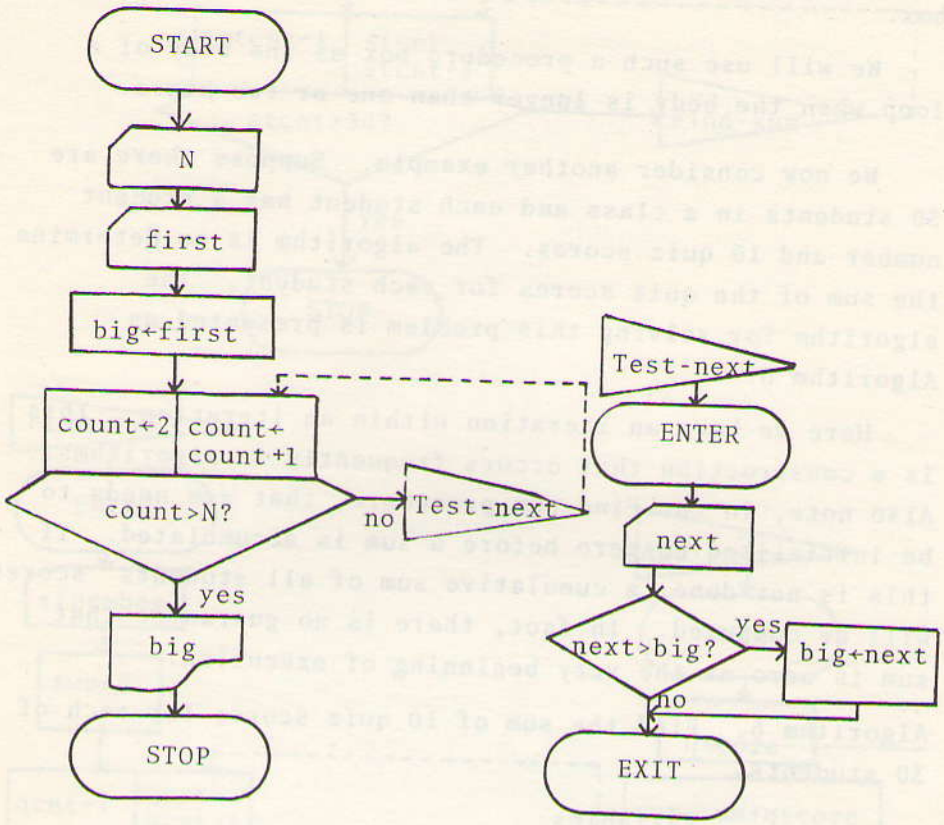


The reason we give the variable-controlled iteration its own form is that it is commonly used and is directly implemented in most programming languages.

The variable which controls the iteration is set to some initial value at the beginning of the iteration. It is then modified after each execution of the body and the iteration is terminated when the control variable satisfies some termination condition. In order to control the iteration properly, the control variable should not be changed in the body. This form of iteration is particularly useful when a process must be repeated a fixed number of times. In this case the control variable is used as a counter which in some way keeps track of the number of times the body has been executed.

Algorithm 5 provides an example of variable-controlled iteration. In this case, the variable *count* is used to count the number of data values which have been read in. It is initialized to 2 to indicate that the second value is read during the execution of the body. It is incremented by 1 each time, and when the increment makes count larger than N, then N values have been read, so the iteration is terminated.

Algorithm 5. Find the largest of N values.



Another new box has been introduced in this algorithm. This is the triangular shaped procedure box. The name inside such a box is the name of a procedure which is to be executed at that point. This type of box is used to avoid complicated constructions in the body of an iteration.

The flowchart for the procedure is then found elsewhere. A procedure is actually a subalgorithm. Instead of beginning and ending with START and STOP boxes, its termination boxes are ENTER and EXIT. The ENTER box of a procedure has a pennant attached to it which gives the name by which it is called into action. When we arrive at the EXIT box of the procedure, we automatically cease execution of the procedure

and begin execution in the algorithm which called the procedure at the next sequential box after the procedure box.

We will use such a procedure box as the body of a loop when the body is longer than one or two boxes.

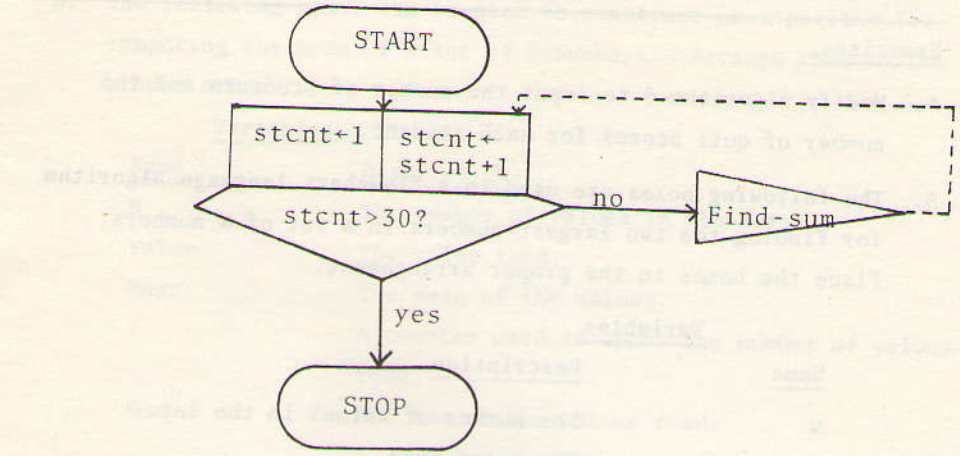
We now consider another example. Suppose there are 30 students in a class and each student has a student number and 10 quiz scores. The algorithm is to determine the sum of the quiz scores for each student. The algorithm for solving this problem is presented as Algorithm 6.

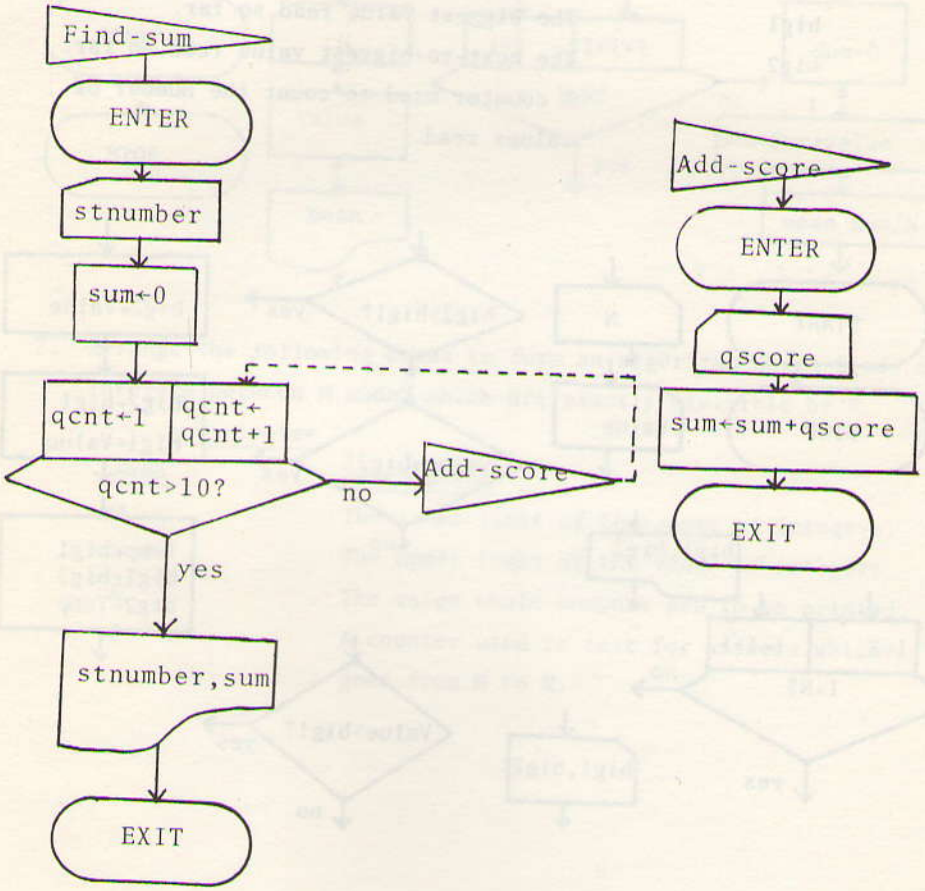
Here we have an iteration within an iteration. This is a construction that occurs frequently in algorithms. Also note, in the Find-sum procedure, that *sum* needs to be initialized to zero before a sum is accumulated. If this is not done, a cumulative sum of all students' scores will be computed. In fact, there is no guarantee that sum is zero at the very beginning of execution.

Algorithm 6. Find the sum of 10 quiz scores for each of 30 students.

Variables

Name	Description
stcnt	The counter for students which goes from 1 to 30.
stnumber	The student number read for each student.
sum	The sum of each student's quiz scores.
qcnt	The counter for quizzes which goes from 1 to 10.
qscore	The quiz score read for each quiz and student.


START
stcnt←1
stcnt←
stcnt+1
stcnt>30?
no
Find-sum
yes
STOP


Find-sum
ENTER
stnumber
sum←0
qcnt←1
qcnt←
qcnt+1
qcnt>10?
no
Add-score
yes
stnumber,sum
EXIT
Add-score
ENTER
qscore
sum←sum+qscore
EXIT

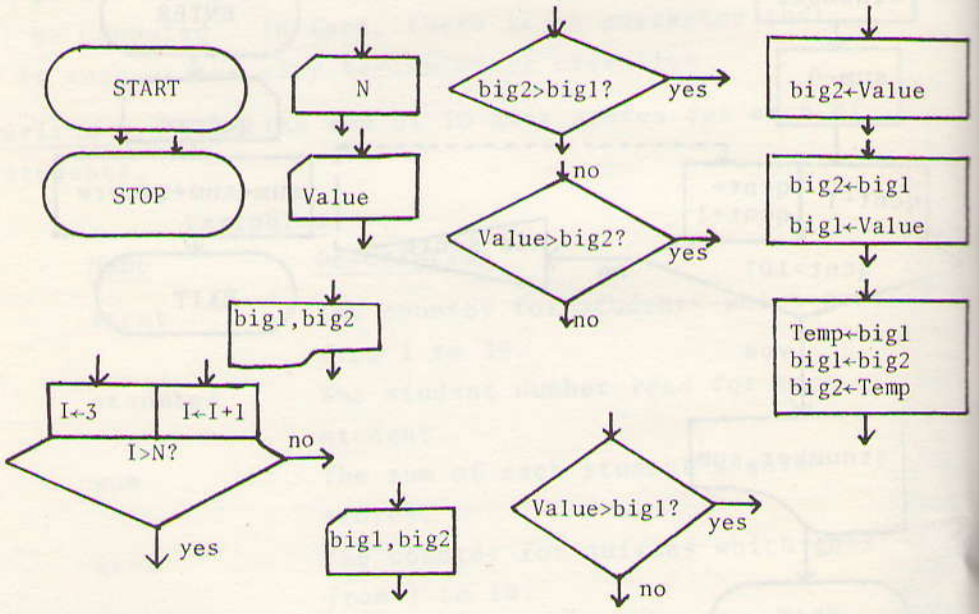
Exercises.

4. Modify algorithm 6 to input the number of students and the number of quiz scores for each student.

5. The following boxes are used in a flowchart language algorithm for finding the two largest numbers in a set of N numbers. Place the boxes in the proper arrangement.

Variables

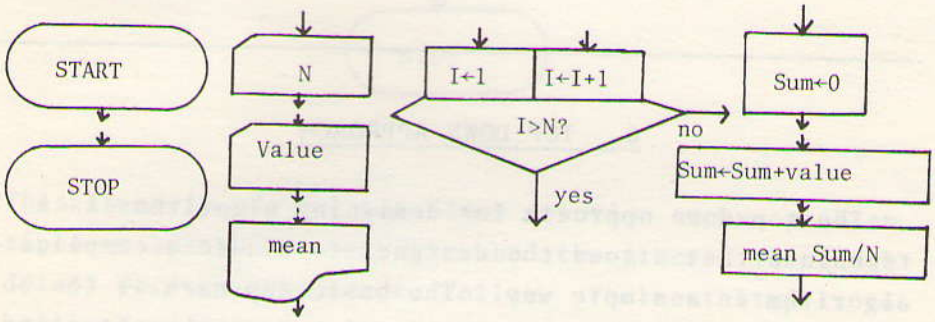
Name	Description
N	The number of values in the set.
Value	The value read.
big1	The biggest value read so far.
big2	The next-to-biggest value read so far.
I	A counter used to count the number of values read.



6. The following boxes can be used to construct an algorithm for computing the mean of a set of N numbers. Arrange them in the proper order.

Variables

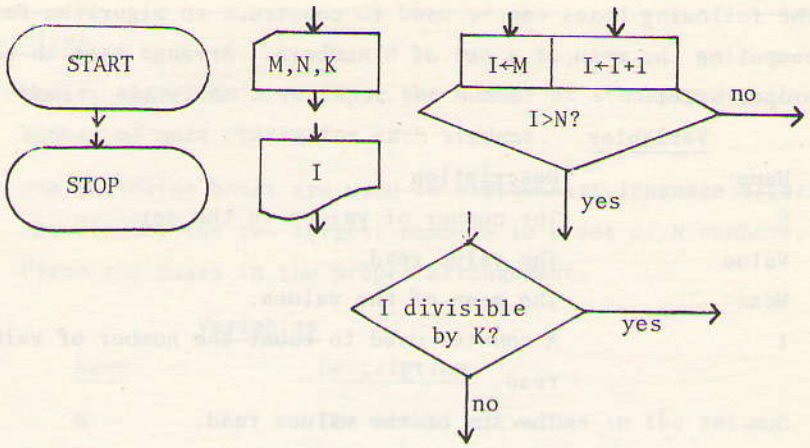
Name	Description
N	The number of values in the set.
Value	The value read.
Mean	The mean of the values.
I	A counter used to count the number of values read.
Sum	The sum of the values read.



7. Arrange the following boxes to form an algorithm which finds all integers between M and N which are exactly divisible by K.

Variables

Name	Description
M	The lower limit of the range of integers.
N	The upper limit of the range of integers.
K	The value whole numbers are to be printed.
I	A counter used to test for answers which goes from M to N.



6. TOP-DOWN APPROACH

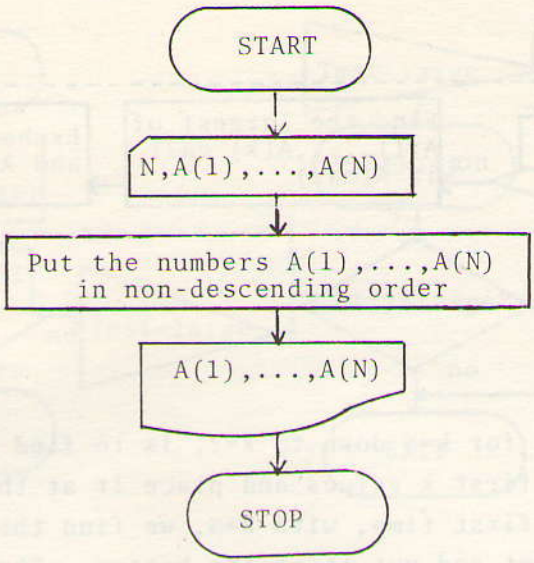
The top-down approach for designing algorithms is a technique that allows the designer to handle a complicated algorithm in a simple way. The basic approach is to design the algorithm using powerful boxes, then breaking those boxes down into flowcharts with less powerful boxes and continuing that process until you are at a level suitable for implementation on a computer.

In order to illustrate this technique, we look at an algorithm for arranging N numbers in natural order.

Algorithm 7. Read N numbers and print them in non-descending order.

Variables

Name	Description
N	The number of values in the set.
A	An array of N values to be ordered.

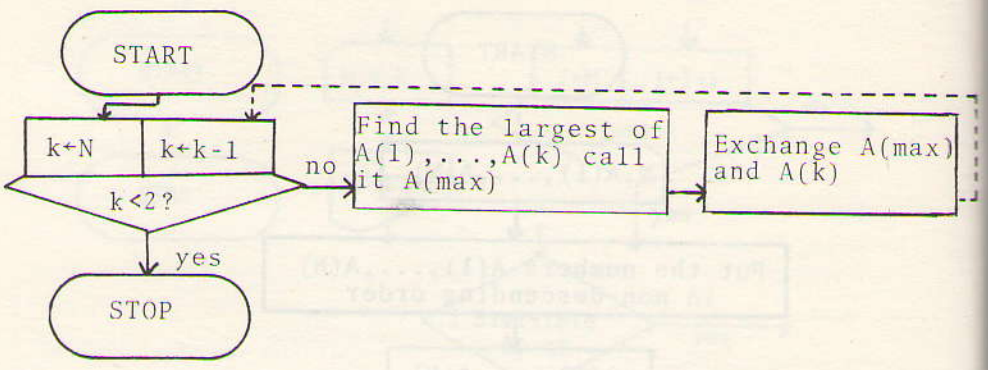


This is obviously not the final solution to our problem since we still have not described the process in enough detail for the computer to follow. The next step is to break the middle box itself down into an algorithm. This is the beginning of iterative refinement.

Algorithm 8. Put the numbers A(1),...,A(N) into non-descending order.

Variables

Name	Description
N	The number of values in the set.
A	An array of N values to be ordered.
k	A counter which goes from N to 2.
max	The index of the largest value from A(1) to A(k).



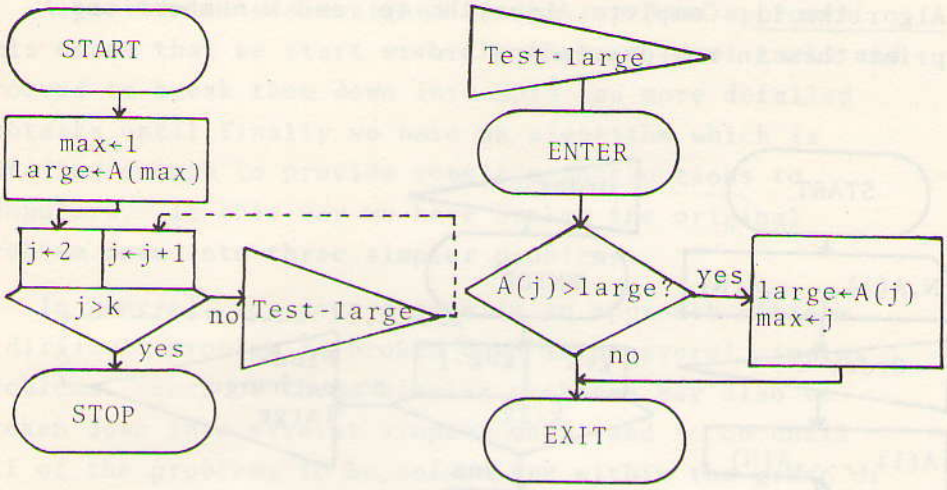
This procedure, for k=N down to k=2, is to find the largest of the first k values and place it at the kth position. The first time, with k=N, we find the largest in the entire set and put it at the bottom. The next pass through the iteration, with k=N-1, we ignore A(N) since it is already correct, and place the largest of the first N-1 values in the (N-1)st position. We continue the process until all are in order.

Next we break down the box "find the largest..." into a flowchart.

Algorithm 9. Find the largest of A(1),...,A(k), call it A(max).

Variables

Name	Description
A	An array of values.
k	The number of values in the set.
max	The index of the largest value from A(1) to A(k).
large	The value of A(max).
j	A counter which goes from 2 to k.

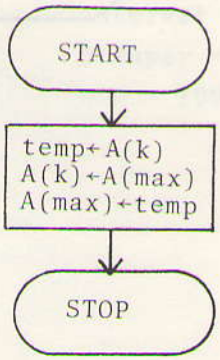


Finally, we expand the "Exchange..." box from Algorithm 9. This requires three data movements and one temporary location. It is given by the following algorithm.

Algorithm 10. Exchange A(max) and A(k).

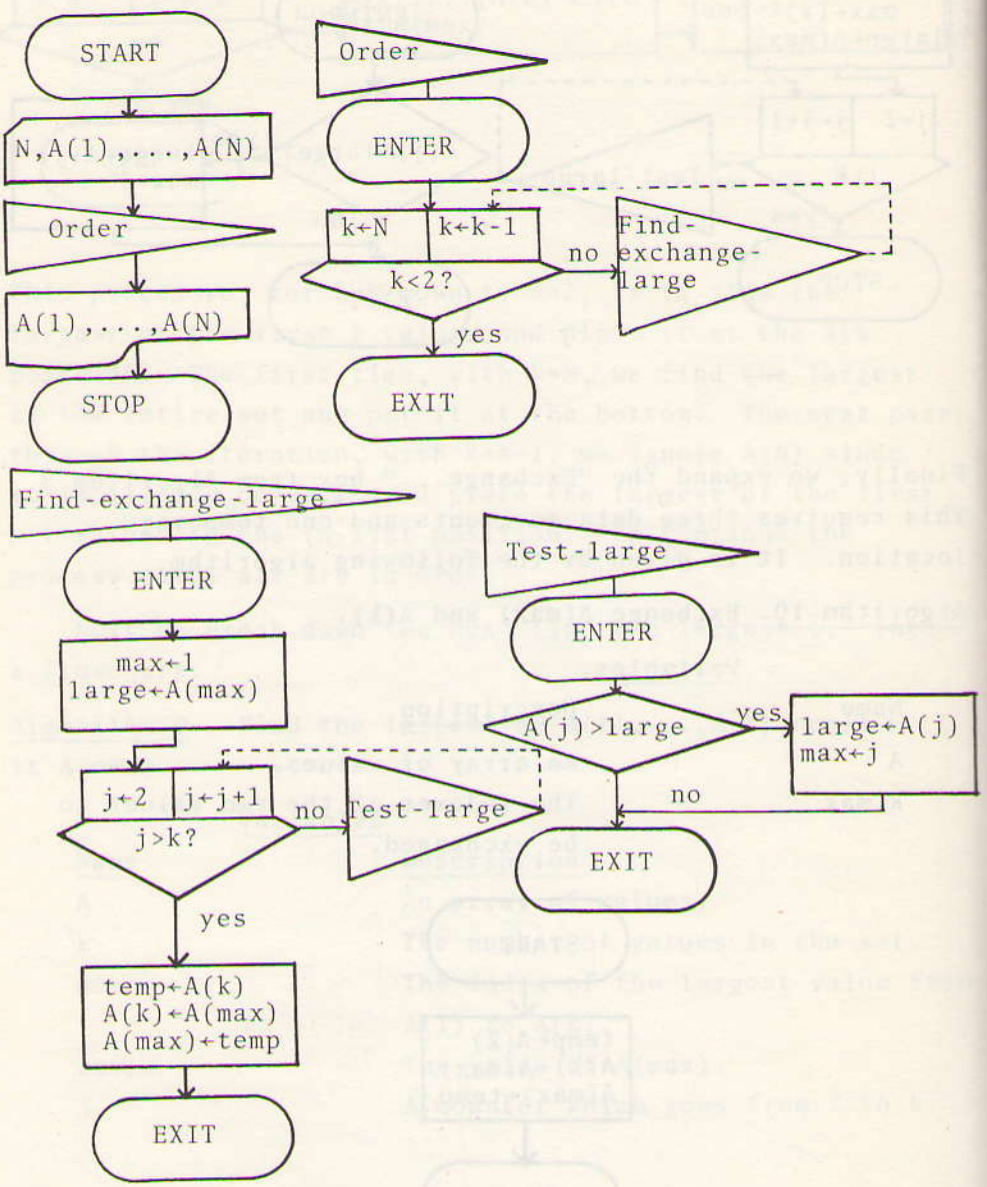
Variables

Name	Description
A	An array of values.
k,max	The indexes of the two values to be exchanged.



We now have, in Algorithms 7-10, all of the steps necessary to complete the task of placing the numbers in order. We combine these into one flowchart for Algorithm 11.

Algorithm 11. Complete Algorithm to read N numbers and print them in non-descending order.



Algorithm 11 was designed by the top-down approach. This means that we start with the highest level tasks and proceed to break them down into more and more detailed subtasks until finally we have an algorithm which is detailed enough to provide complete instructions to computers. In this way we have broken the original problem down into three simpler problems.

In general, top-down design is an approach whereby a difficult problem is broken down into several simpler problems. Each of these simpler problems may also be broken down into several simpler ones, and so on until all of the problems to be solved are within the grasp of the problem solver. This iterative refinement is a very important strategy in computer problem solving.

Exercises:

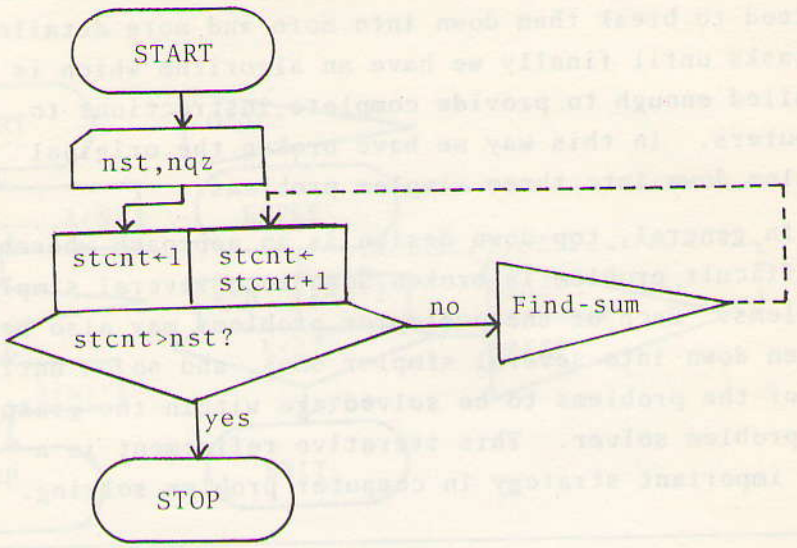
8. Using the set of numbers 5,3,9,6,2,7 follow through Algorithm 11 as a computer would.

---

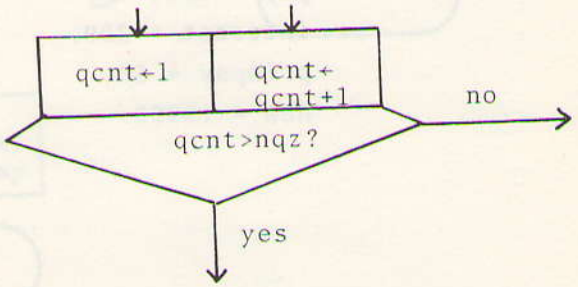
7. SOLUTIONS TO EXERCISES

3. For Test 1, result is prin = 20000.
rate = .12
pay = 250.
interest = 200.
prinpay = 50.
new = 19950.

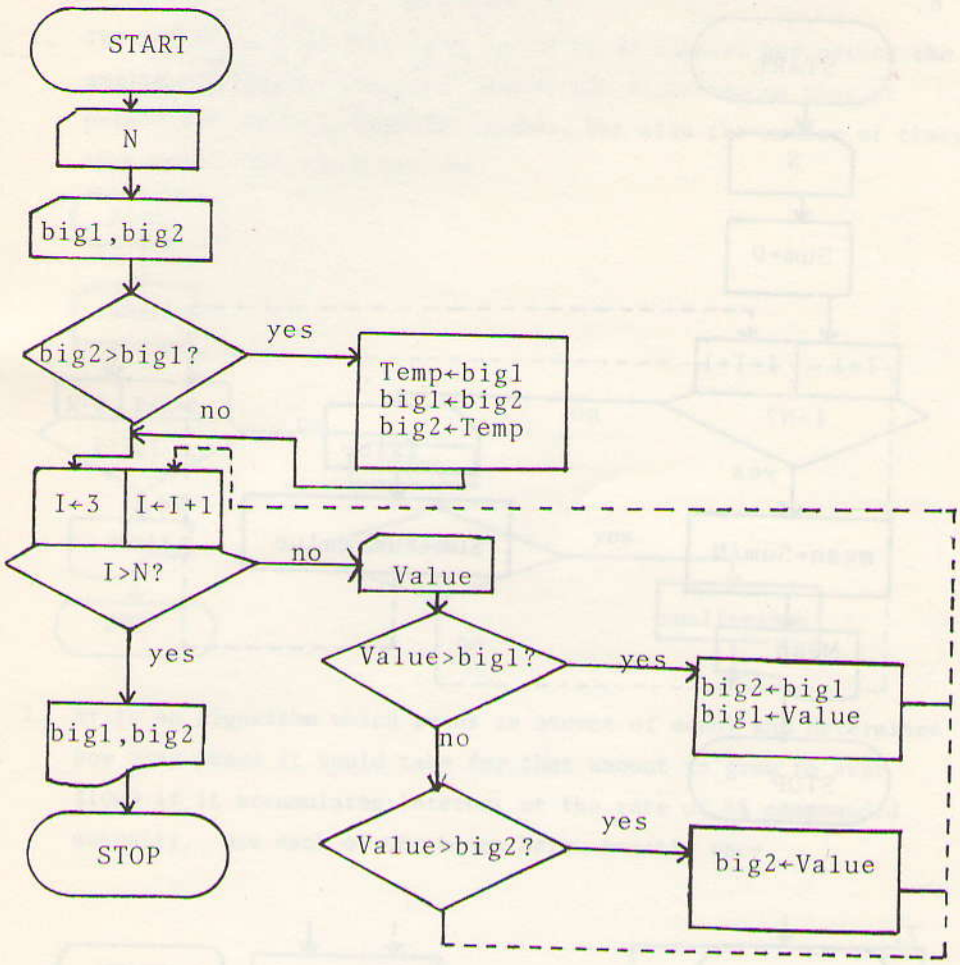
4. Change first flowchart to;



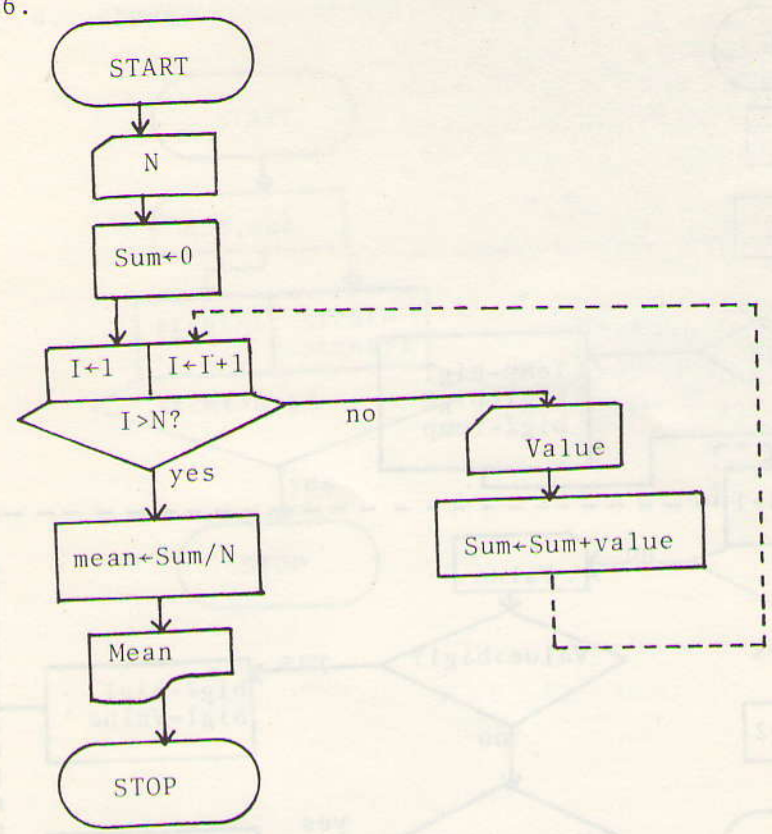
Change iteration box in Find-sum to



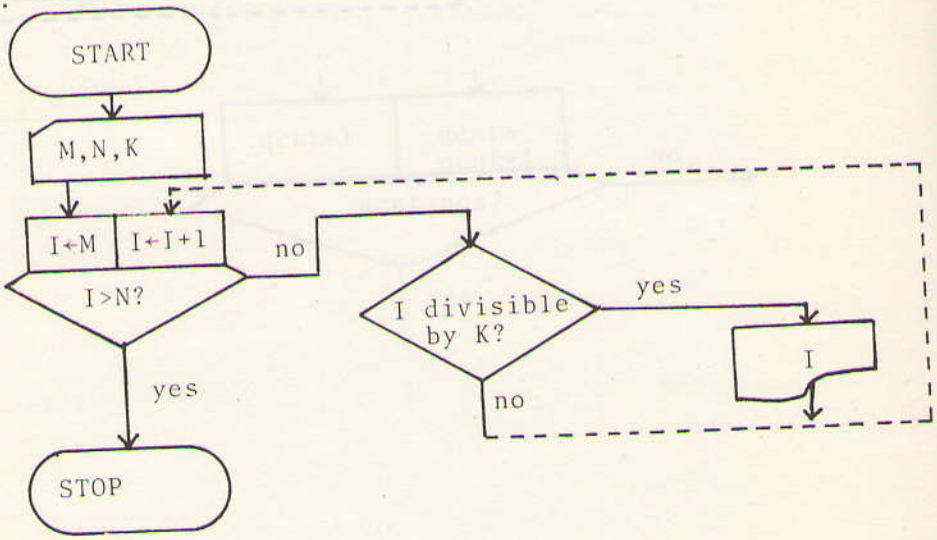
5.


START
N
big1,big2
big2>big1?
yes
Temp←big1
big1←big2
big2←Temp
no
I←3
I←I+1
I>N?
no
Value
yes
big1,big2
STOP
Value>big1?
yes
big2←big1
big1←Value
no
Value>big2?
yes
big2←Value

6.

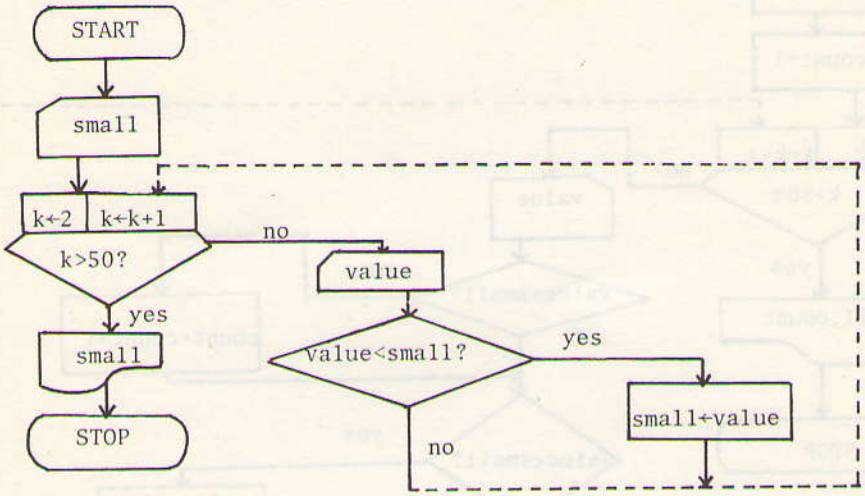

START
N
Sum←0
I←1
I←I+1
I>N?
no
yes
Value
Sum←Sum+value
mean←Sum/N
Mean
STOP

7.

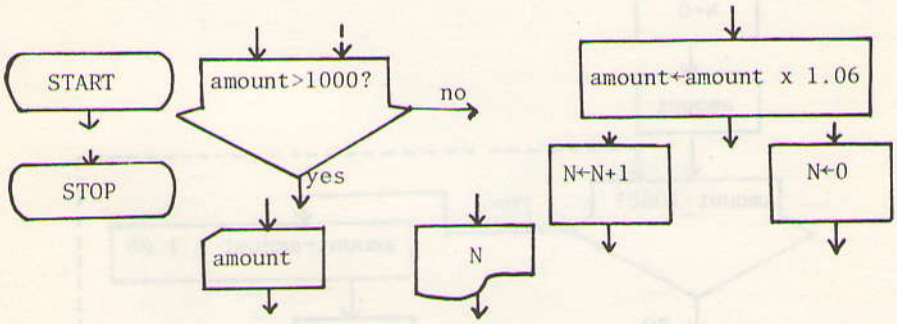

START
M,N,K
I←M
I←I+1
I>N?
no
yes
I divisible by K?
yes
no
I
STOP

8. MODEL EXAM

1. The following algorithm reads a set of 50 numbers and prints the smallest number in the set. Modify the algorithm so that it prints not only the smallest number, but also the number of times that number occurs in the set.



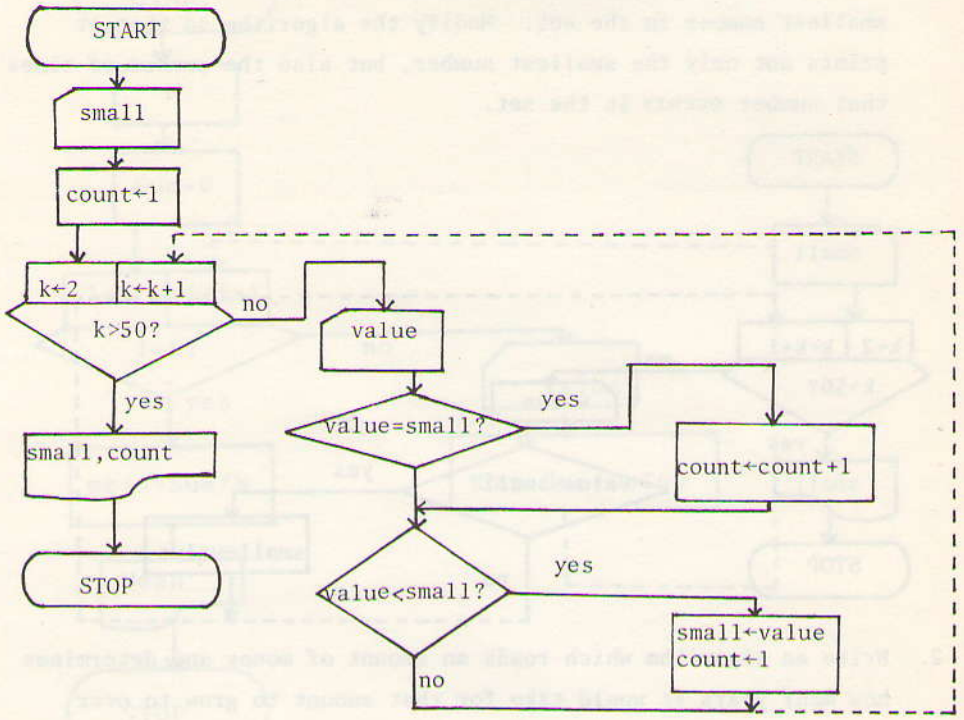
2. Write an algorithm which reads an amount of money and determines how many years it would take for that amount to grow to over $1000 if it accumulates interest at the rate of 6% compounded annually. Use each of the boxes below exactly once.



3. Explain in your own words the top-down approach to algorithm design. Discuss why it is important.

9. SOLUTIONS TO MODEL EXAM

1.



2.