DOCUMENT RESUME

ED 218 097 SE 037 897

AUTHOR	Dershem, Herbert L.
TITLE	Computer Problem Solving. [and] Iteration and Computer Problem Solving. Computer Science/Algorithms. Modules and Monographs in Undergraduate Mathematics and Its Applications Project. UMAP Units 477 and 478.
INSTITUTION	Education Development Center, Inc., Newton, Mass.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	80
GRANT	SED-76-19615-A02
NOTE	75p.
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	*Algorithms; Answer Keys; *College Mathematics; Computers; *Computer Science Education; Higher Education; Instructional Materials; *Learning Modules; *Mathematical Applications; *Problem Solving; Supplementary Reading Materials

ABSTRACT

These modules view aspects of computer use in the problem-solving process, and introduce techniques and ideas that are applicable to other modes of problem solving. The first unit looks at algorithms, flowchart language, and problem-solving steps that apply this knowledge. The second unit describes ways in which computer iteration may be used effectively in problem solving, and shows ways in which two other forms of iteration may be applied in algorithm construction. Both modules include exercises, and each has a model exam. Answers to all problems presented are provided. (MP)

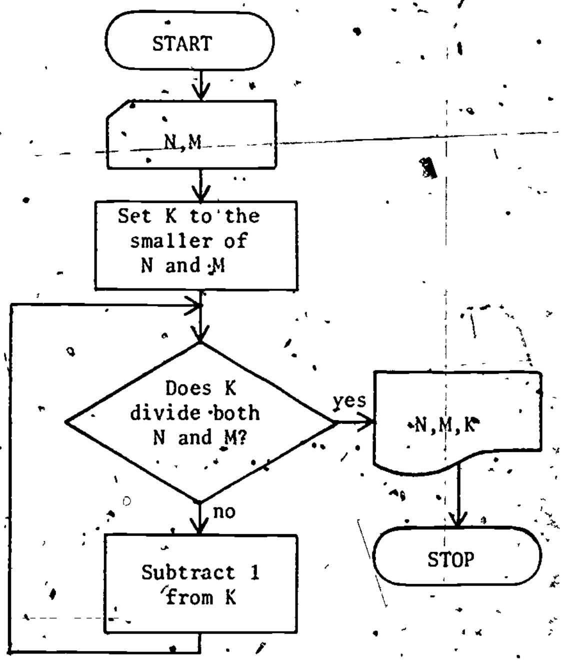
umap

UNIT 477

MODULES AND MONOGRAPHS IN UNDERGRADUATE MATHEMATICS AND ITS APPLICATIONS PROJECT

COMPUTER PROBLEM SOLVING

by Herbert L. Dershem



COMPUTER SCIENCE/ALGORITHMS

edc/umap/55 chapel st./newton, mass. 02160

COMPUTER PROBLEM SOLVING

by

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

TABLE OF CONTENTS

Intermodular Description Sheet: *UMAP Unit 477*

Title: COMPUTER PROBLEM SOLVING

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

Review Stage/Date: III 9/30/80

Classification: COMPUTER SCI/ALGORITHMS

Prerequisite Skills: None

Output Skills:

1. Be able to explain the concept of algorithm and be able to distinguish algorithms from non-algorithmic solution descriptions.
2. Be able to read and explain algorithms written in the flow-chart language.
3. Be able to state the seven steps in computer problem solving and be able to apply them in solving simple problems.

Related Units:
Iteration and Computer Problem Solving *(Unit 478)*

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 which may be used to supplement existing courses and from which complete courses may eventually be built.

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

The Project would like to thank Carol Stokes of Danville Area Community College, Danville, Illinois; Ray Treadway of Bennett College, Greensboro, North Carolina; Douglas F. Hale of the University of Texas-Permian Basin, Odessa, Texas; 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. SED76-19615 A02. Recommendations expressed are those of the author and do not necessarily reflect the views of the NSF or the copyright holder.

1. INTRODUCTION

Much of a person's life is spent in solving problems. Usually, we use a tool or a set of tools to assist us in solving those problems. For example, when faced with the problem of transporting myself from where I am now to a location across town where I would like to be, I might use an automobile as a tool. If I am faced with the problem of removing the contents of a capped bottle, a bottle-opener would be an appropriate tool. In some cases, I need to use a combination of tools. For example, if I had to remove the contents of a capped bottle that was in a store across town, I would use both the automobile and the bottle-opener.

For many problems that people need to solve, the computer is an appropriate tool. But the computer is just a tool; it will not solve a problem by itself. The computer can assist you in solving a problem only if you know how to utilize it correctly. This module introduces techniques and ideas which you may apply when using the computer to solve problems.

Although we are primarily concerned with the use of computers in the problem solving process, the techniques are applicable to other modes of solving problems.

2. ALGORITHMS

The first task in solving a problem is finding appropriate ways of representing the solution process. The more carefully this is done, the easier the problem solving process. An algorithm is a common way of representing the solution process, so we begin by defining this term.

Definition of an Algorithm

An ordered set of rules for solving a problem is called an *algorithm* if it has the following properties:

1. The rules are unambiguous.
2. The rules are in a proper sequence.
3. The procedure specified by the set of rules solves the problem.
4. The procedure terminates after a finite number of steps, or actions specified by the rules.

We will "back into" an understanding of what an algorithm is by first considering what one isn't.

Nonalgorithm 1. Directions for getting to the hospital.

1. Go west on Tenth Street until you come to a stoplight.
2. Turn onto River Avenue at the stoplight.
3. Make a right at the Y on River Avenue.
4. The hospital will be on your right a few blocks past the Y.

Unfortunately, we have all been the recipients; and probably the givers as well, of nonalgorithmic directions like these. I hope your illness is not too acute in this case because if it is, you may terminate before the algorithm does. The problem with this nonalgorithm is that it violates condition 1 by being ambiguous. In rule 1, which stop light is meant? In rule 2, which way should I turn? At the Y described in rule 3, there is a right fork and a sharp right turn possible. Which should I take? How many blocks are a few in step 4?

Nonalgorithm 2. Directions for passing an exam.

1. Get lots of sleep the night before the exam.
2. Outline the material.
3. Read the material.
4. Listen attentively in class.
5. Take the examination.

This nonalgorithm contains rules that might be adequate to solve the problem, but they cannot be carried out in the specified order. Thus the rules are not in proper sequence, and condition 2 of the algorithm definition is violated.

Nonalgorithm 3. Directions for passing a true-false test.

1. Bring a coin to the test.
2. Flip the coin for each item on the test.
3. If the coin lands heads respond with true, if the coin lands tails, respond with false.

There is no ambiguity in this nonalgorithm, and steps are in proper sequence. The only difficulty is that unless you have an unusually intelligent (lucky?) coin, this procedure may not solve the problem at hand, which is to pass the test. Hence, condition 3 of the definition is violated.

Nonalgorithm 4. Directions for making a million dollars.

1. Get 10 million dimes.
2. Go to Las Vegas.
3. Play the dime slot machines until either you have 20 million (or more) dimes or until you run out of them.
4. If you run out of dimes, go back to step 1.
5. If you reach 20 million dimes, quit while you're ahead!

Even if you obtain the supply of dimes needed in step 1 and the cranking power needed in step 3, this procedure is still a nonalgorithm because it may never terminate: there is no guarantee that you will ever obtain the desired result at step 3. Hence, condition 4 in the definition is violated.

Now we are ready to consider an example which does qualify as an algorithm under our definition.

Algorithm 1. Find the greatest common divisor of two given numbers, N and M.

1. Let K be the smaller of N and M.
2. If K divides both N and M, then K is the greatest common divisor.
3. Otherwise, subtract 1 from K.
4. Go to step 2.

This algorithm is unambiguous, its rules are in proper sequence, it solves the problem, and it does so in a finite number of steps. It is also very simple and easy to follow. However, it may take a long time to solve the problem using this algorithm. For example, if 15798433 and 566832 are used for N and M, step 2 would be executed 566783 times before the correct answer of 49 is found. A more efficient algorithm, that is, one that can solve the same problem with much less effort, is known and we shall present it next.

Algorithm 2. Given two numbers, N and M, find their greatest common divisor.

1. If N is smaller than M, then exchange the two.
2. Divide N by M and call the remainder R.
3. If the remainder R is zero, then M is the GCD.
4. Otherwise, set N to the value of M and M to the value of R.
5. Go to step 2.

If we follow this algorithm for N = 15798433, M = 566832, we obtain the successive values of M as follows:

M = 566832
M = 493969
M = 72863
M = 56791
M = 16072
M = 8575
M = 7497

$M = 1078$
$M = 1029$
$M = 49$

In this case we execute step 2 only nine times, a significant improvement over Algorithm 1.

The phenomenon which we observe here occurs often in algorithms. Algorithm 1 is simple, straightforward, and solves the problem in an inefficient manner. Algorithm 2 is much more efficient but the cost is additional difficulty in understanding it. This tradeoff between simplicity and efficiency of algorithms frequently forces the algorithm designer to make a decision on the subject of priorities.

Exercises

1. Modify nonalgorithm 2 to make it an algorithm.
2. Determine whether each of the following are algorithms or nonalgorithms. For each nonalgorithm, find the rule or rules it violates and rewrite it as an algorithm.
   (a) How to place a telephone call.
      1. Pick up the receiver.
      2. Listen for a dial tone.
      3. Dial the number.
      4. Conduct the conversation.
   (b) How to find a word in a dictionary or determine that it is not listed.
      1. Turn to any page.
      2. If word at the top left of the page occurs alphabetically after the desired word, go to step 4.
      3. Turn ahead 10 pages and go to step 2.
      4. Turn back 2 pages.
      5. If word at the top right is before the desired word, go to step 3.
      6. Scan page for desired word.
      7. If it is found, write a definition.
      8. If it is not found, write, "not in dictionary."

   (c) How to fill a ditch with sand.
      1. Obtain a shovel.
      2. Start shoveling sand into the ditch.
      3. If you run out of sand, go get more and go to step 1.
      4. When ditch is full, stop.
   (d) How a doctor heals a patient.
      1. Learn about patient's problem.
      2. Consider the symptoms.
      3. Initiate a treatment.
      4. Examine and test patient.
      5. If patient is cured, then send the bill.

3. FLOWCHART LANGUAGE

We next ask how we should express algorithms. In other words, is there a convenient language for communicating an algorithm from one person to another or from a person to a computer?

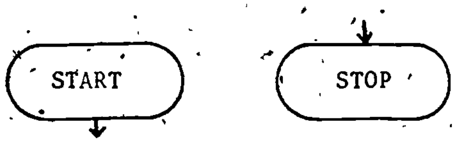
The language we have used above in communicating Algorithms 1 and 2 is English, which is very appropriate because it is understood by a reasonably large subset of the people with whom we communicate. English has two disadvantages, however. First, it is ambiguous as far as meaning is concerned, and therefore, even a carefully worded algorithm can suffer from the ambiguity of the language. Second, English is not an appropriate language for communicating with computers, since it is too complicated for them to understand.

It would appear then, that a computer programming language may be an answer to our dilemma. Programming languages are, of necessity, unambiguous, and are understandable to the computer. And indeed one of our goals is to express algorithms in this form so that the computer can solve the problem. But people have difficulty expressing algorithms directly in programming languages. For this reason we develop an intermediate language between English and the programming language, which we call *Flowchart Language*.

The expression of an algorithm in Flowchart Language consists of the rules of the algorithm written in abbreviated English and pictured graphically in "boxes." These boxes are connected by arrows which indicate the sequencing of the steps. We will use this Flowchart Language to express all of our algorithms.

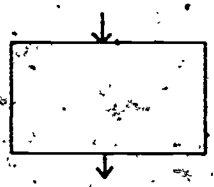
There are five different kinds of boxes that are used in representing algorithms.

1. Terminal box



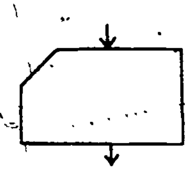
Terminal boxes are used to identify the starting point and stopping point of an algorithm. They will always contain either the word START or the word STOP.

2. Processing box

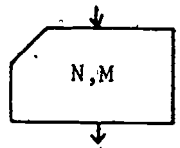


The work of the algorithm is specified in processing boxes. It contains an English statement which describes the action that is to be taken.

3. Input box

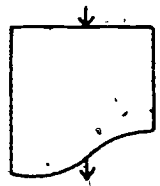


Each input box contains the names of variables whose values are to be obtained from a known source. For example, the box

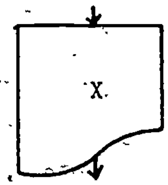


represents the obtaining of two values, the first called N and the second M.

4. Output box

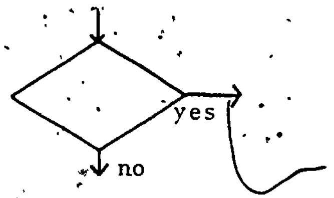


Like an input box, each output box contains the names of variables. The values of these variables are to be displayed in some form. For example, the box



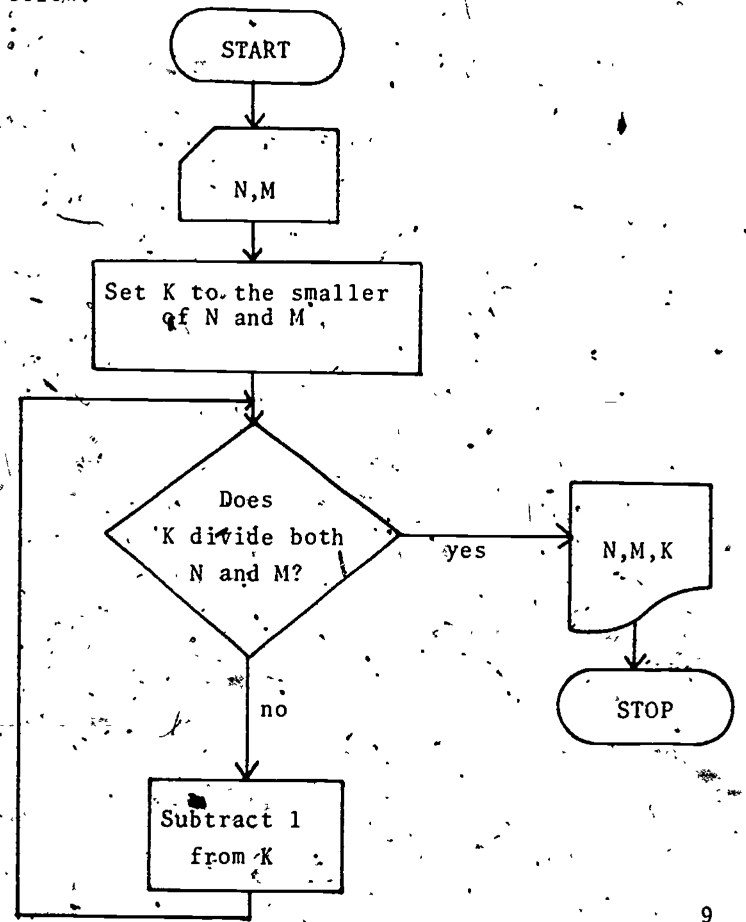
indicates that the current value of X is to be displayed.

5. Decision box



Each decision box contains a question which can be answered yes or no. If the answer is "yes," the "yes" path is taken from the box. If the answer is "no," the "no" path is taken.

As an example, Algorithm 1 in Flowchart Language is shown below.



Names that are used in an algorithm to represent numerical values and whose values may change during the execution of the algorithm, are called *variables*. Three variables, N, M, and K, are used in Algorithm 1. It is easier to understand what an algorithm does if there is some explanation of the meanings of the variables included with the algorithm. For example, for Algorithm 1, our variable descriptions would be as follows:

Variables

Name	Descriptions
N,M	The two given numbers.
K	A counter used in searching for the greatest common divisor. It will contain the greatest common divisor when the algorithm terminates.

Setting a variable to the value of another variable or to the result of some arithmetic expression is such a common algorithm step that a special symbol, the left pointing arrow, is used to represent it. For example,

$$A \leftarrow B$$

means "set A to the value of B." This notation can also be used to represent the incrementing or decrementing of variables. The step "add 1 to X" is given by

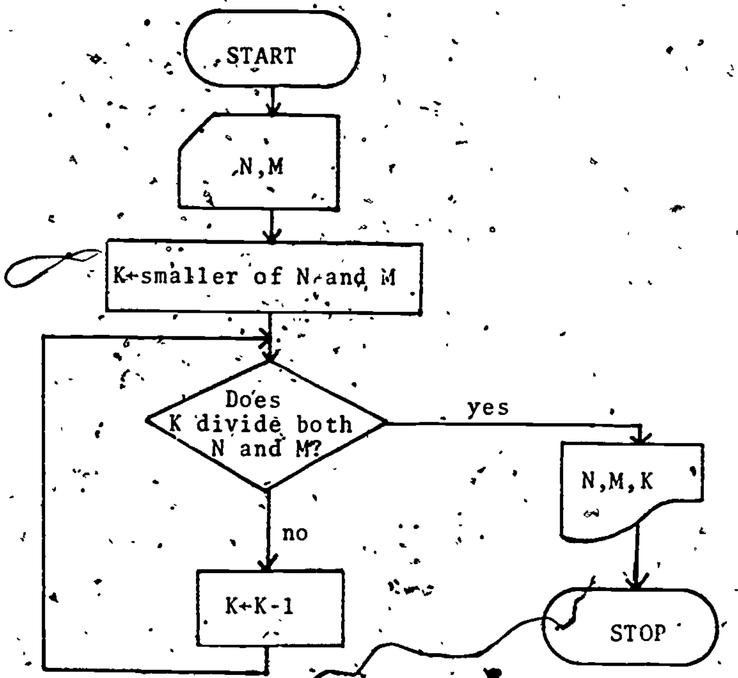
$$X \leftarrow X+1$$

Using this new symbol and the variable descriptions, we can now restate Algorithm 1.

Algorithm 1 - Second Version. Given two numbers, N and M, find their greatest common divisor.

Variables

Name	Description
N,M	The two given numbers.
K	A counter used to search for the greatest common divisor. It will contain the greatest common divisor when the algorithm terminates.

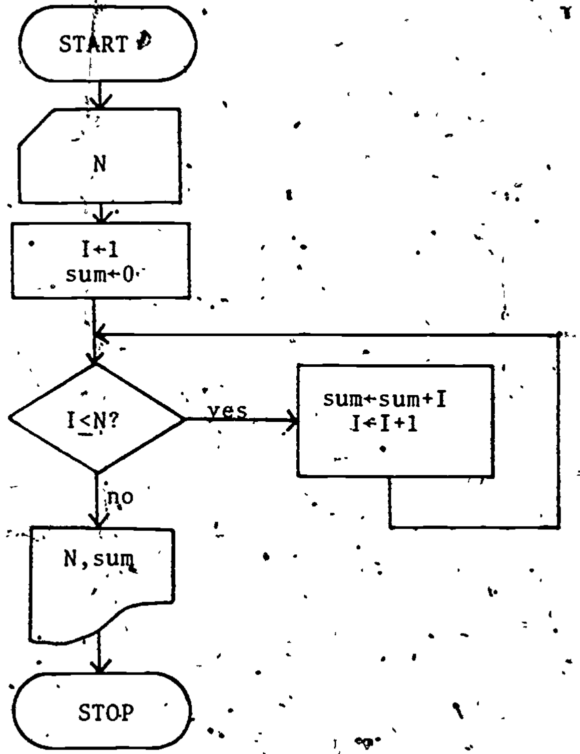


Exercises

3. Write algorithm 2 in the Flowchart Language.
4. Follow the algorithm described in the following flowchart and give the resulting output when the input value is 5.

Variables

Name	Descriptions
N	The number input.
I	A counter.
Sum	The sum of the first N integers.



4. PROBLEM SOLVING STEPS

Now that we have the concept of an algorithm and the flowchart language available, we are ready to learn how to use these tools in solving problems. It would be simple if we could put the statement of a problem into a machine and get out an algorithm for solving it. Unfortunately, no such machine has ever been discovered. Problem solving is accomplished only through careful work: before the algorithm is constructed, detailed planning and analysis must occur; after the algorithm is constructed, extensive testing must take place to verify that it does indeed solve the problem.

In this section we outline a step-by-step procedure that is useful in the construction of algorithms which solve problems. You might call this an algorithm for constructing algorithms. This process consists of seven steps which are described below.

1. Precisely define the problem.

At this step, the problem solver determines what the problem is. This step, in practice, is usually quite difficult. In many courses we are not concerned with this step since the problems are precisely formulated by the textbook author or the instructor. An important part of this step is determining exactly what form the output of the problem should have.

2. Identify the inputs to the problem.

At this step you ask, "What are the pertinent facts that are given in this problem?" The answer will depend on what is available and what is needed. One important aspect of this task is determining the appropriate form for the input.

3. Identify the outputs of the problem.

Here you determine the results that are desired. By considering the result of step 1, you should be able to determine what is needed and in what form.

4. Construct an algorithm for the solution.

This is the key step in the process and one which can cause the most trouble. Too many problem-solvers try to skip this step or combine it with the next in an effort to obtain a solution quickly. This is a situation where haste really does make waste: time invested in careful formulation here can save much more time at steps 6 and 7.

5. Implement the algorithm for the solution.

In computer problem solving, this step is known as programming. If step 4 is done carefully, it can be carried out in a straightforward way once the basics of a programming language are mastered.

6. Test the procedure constructed in step 4.

Once the implementation is complete, we test the procedure using inputs for which the correct outputs are known, and compare the results with our expectations. If they differ, then the existence of an error has been discovered. There is a temptation to assume the algorithm is correct and rush through this step. As you gain more experience with computer problem solving, you will learn (probably the hard way) that you should never assume anything is correct. That kind of skeptical attitude makes for the best testing.

7. Locate and correct errors uncovered by testing and go back to step 6.

Errors may originate at step 4 or step 5. Usually those which originate at step 5 are fairly easy to detect and correct because they require some minor modification to the program. Errors which origniate at step 4 are much more difficult. Such errors may require you to completely rethink your algorithm and, in some cases, discard all you have done and start over. Again, a careful job at step 4 can avoid this waste.

5. AN EXAMPLE

Let's follow this process through for a simple problem. The problem is to read a set of numbers and determine how many are positive and how many are negative.

1. Precise formulation - The problem as stated above leaves out some necessary information. First, what are we to do with zeros? Should they count as positive,

negative, or not at all? Secondly, how will we know when we have all of the numbers?

We can solve both of these problems at once by saying a zero will not be counted as positive or negative but rather a zero will signify the end of the data. Therefore, our more precise formulation of the problem now reads as follows:

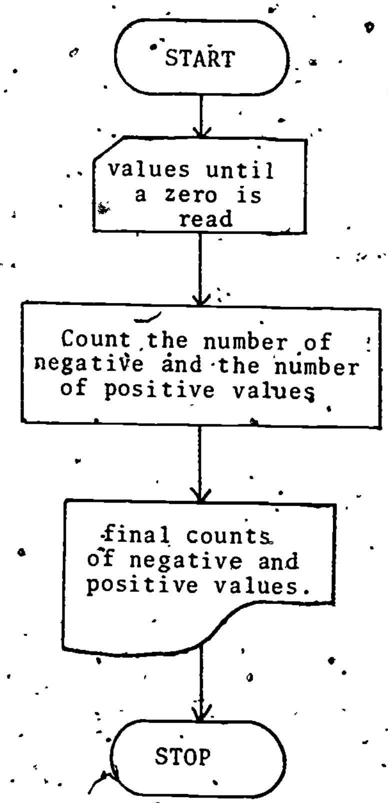
> Read a set of numbers until a zero is encountered and count the number of positive and negative numbers read.

2. Inputs - The inputs will be the given set of numbers, all non-zero, followed by a zero. Note that any input not ending with a zero is invalid for this problem.

3. Output - The output of our problem consists of two numbers, the number of positive values and the number of negative values.

4. Construction of an algorithm - Our flowchart language is a useful tool here and we can nicely formulate our solution as follows:

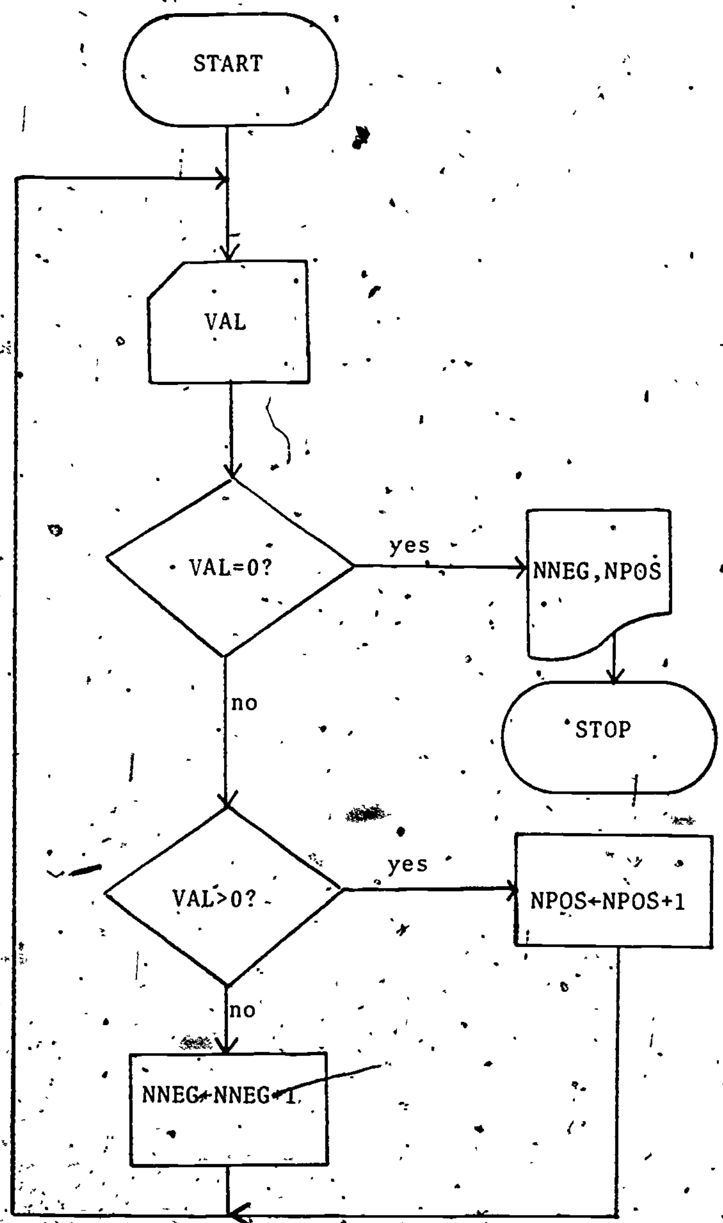
<u>Algorithm 2</u>. Count the number of positive and negative values in a data set - Version I.



This solution algorithm, though certainly correct, provides too little detail to be of sufficient use in implementation. Therefore, we refine it.

<u>Algorithm 3</u>. Count the number of positive and negative values in a data set - Version II.

Name	Description
VAL	The values input.
NNEG	The number of negative values.
NPOS	The number of positive values.



This algorithm appears to be correct. But as we warned earlier, never assume anything is correct.

5. Implementation - Algorithms such as this one are implemented translating them into a computer programming language such as FORTRAN or BASIC. Since this module is intended to be independent of programming languages, we will not discuss this step in algorithm development. If you are interested in pursuing this further, you may do so by learning to program in any programming language.

6. Testing - The above algorithm was implemented in a programming language and run with the following results.

   Input: 12,-5,15,-2,7,0
   Output: NNEG=1762699507
   NPOS=2001731581

   Obviously, there is some difficulty here. The strategy used to check for the presence of an error was to use an input for which we know what the output should be. When the output obtained differed from our expectations, we knew there was an error, and therefore we must proceed directly to step 7.

   Had we been successful in generating the expected output for this particular input, we should run additional tests until we have satisfied ourselves that the procedure is correct.

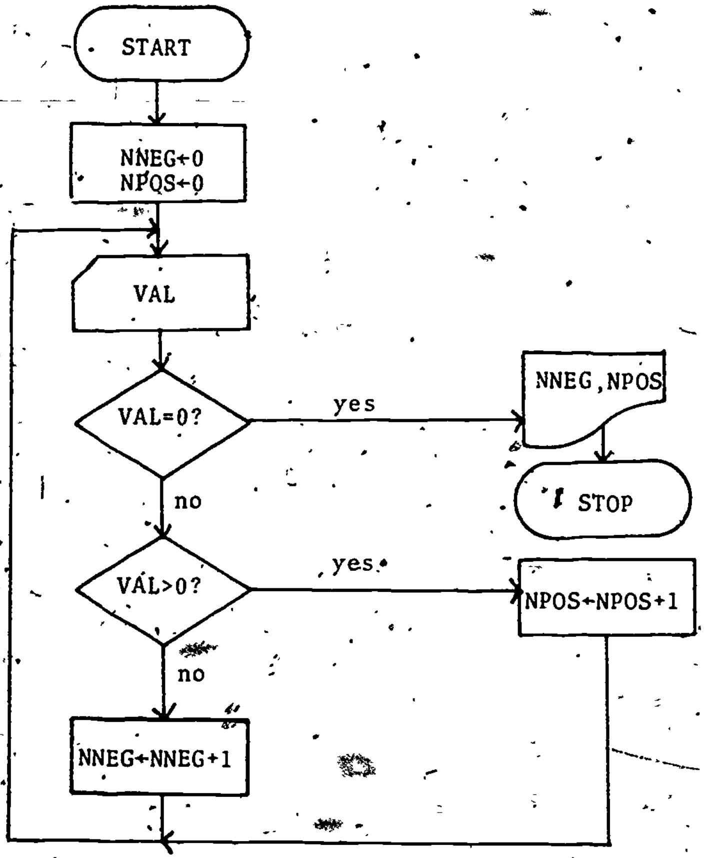
7. Locate and correct errors - This phase of the problem solving process is a difficult one which is learned only through experience. In this case, we notice that values of NNEG and NPOS are quite wild and after a little thought we suspect that these two variables were not given proper initial values, which in this case should be zeros. In many programming languages, variables like NNEG and NPOS are not automatically set to zero. In order to insure the correctness of our

program we need to include steps to do this.

After our algorithm is modified to include these initializations to zero, it takes the following form.

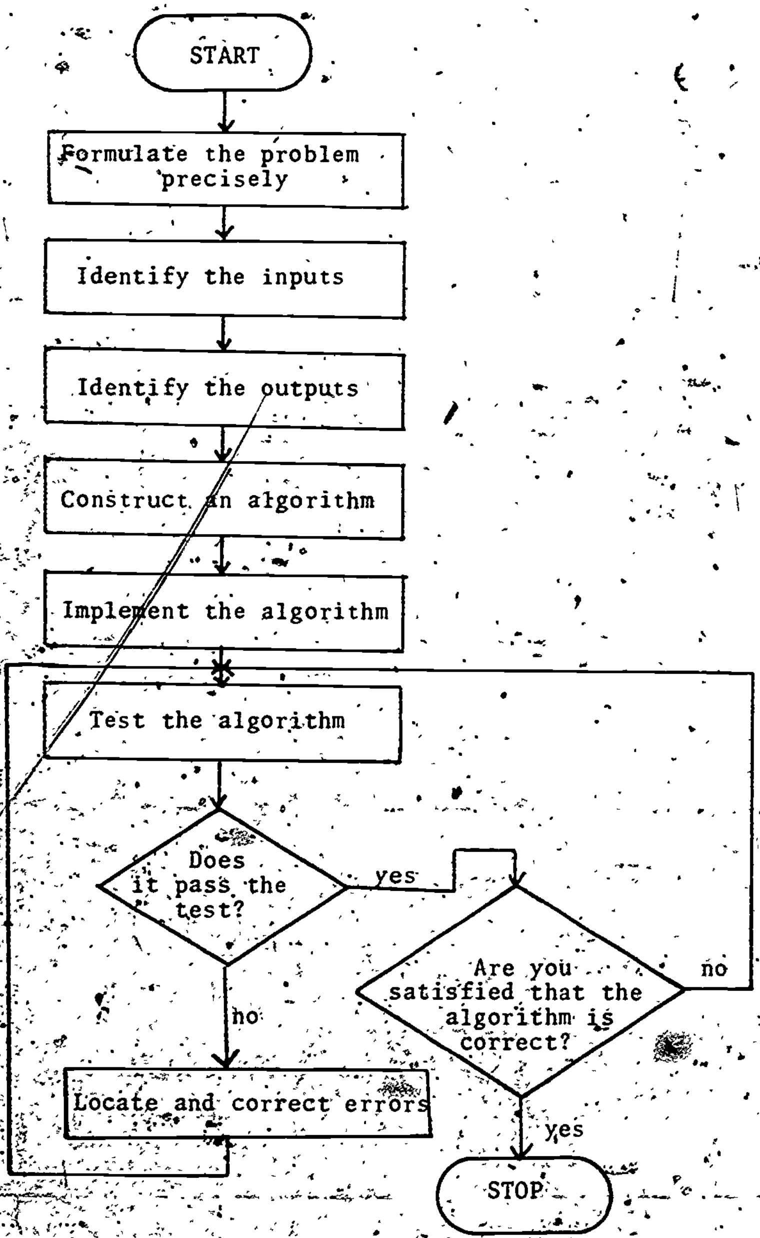
Algorithm 4. Count the number of positive and negative values in a data set - Version 3.

Name	Variables Description
VAL	The values input.
NNEG	The number of negative values.
NPOS	The number of positive values.



After making the correction which results in Algorithm 4, we must return again to step 6 for testing. Extensive testing of the implemented version of this algorithm will reveal no further errors. When we have tested enough to satisfy ourselves with the correctness of the algorithm, we accept it as providing the solution.

We have now outlined a basic procedure for solving a problem. This procedure can be put in the form of a flowchart as well.



Exercises

Follow through the first four of the seven problem solving steps with the following problems:

5. Put three numbers into non-descending order.
6. Calculate the mean of a set of numbers.
7. Input an integer and compute the sum of all integers from one to the integer input.
8. Input an amount A, and an interest rate I. Compute the principal resulting if A dollars are invested for a given time period with interest rate I percent per time period.
9. Input A and I as in 4, and N, the number of time periods. Compute the principal resulting if A dollars are invested for N time periods with interest rate I percent per time period compounded after each time period.

---

6. ANSWERS TO EXERCISES

1. One possible solution might be:

   1. Read the material.
   2. Listen attentively in class.
   3. Outline the material.
   4. Gets lots of sleep the night before.
   5. Take the examination.

2. (a) Nonalgorithm because it may fail to solve the problem. For example, what happens if the line is busy?

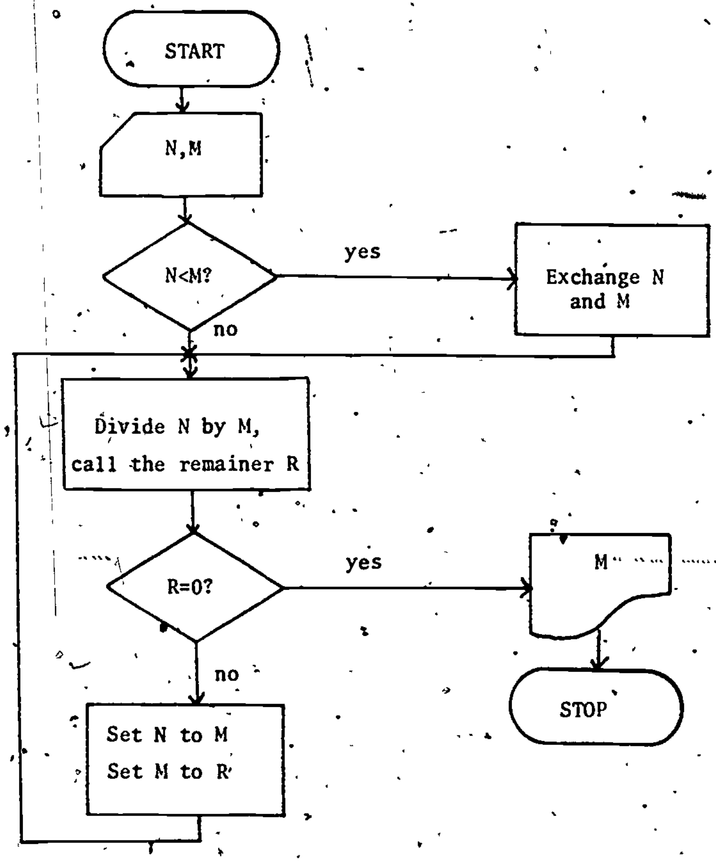
   (b) Nonalgorithm because it may not solve the problem. If the search begins on an even (odd) page, then only even (odd) pages will be scanned. If step 4 had read, "Turn back 1 page," the procedure would have been an algorithm.

   (c) Nonalgorithm because it is ambiguous. Step 3 does not explain how to get more sand.

   (d) Nonalgorithm because steps are out of order. If steps were changed to 1,4,2,3,5 then it would be better. Also, no action is specified if patient is not cured, so the procedure may not terminate. One could add the step:

      6. If patient dies, then stop treatment.

3.



4. This will print

   N=5

   sum=15

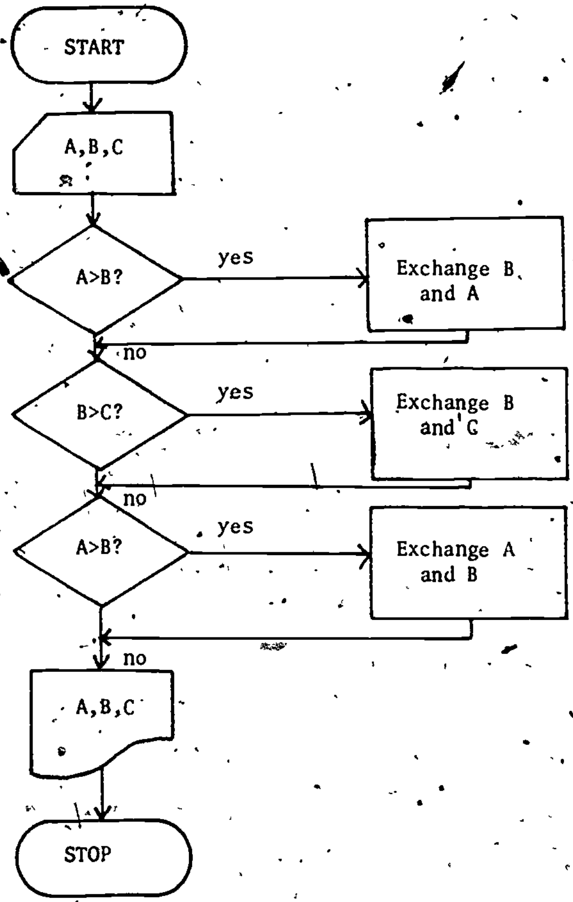
5. Precise formulation: Put three real numbers into non-descending order.

   Input: The three numbers in original order.

   Output: The three numbers in non-descending order.

   Algorithm: Variables

Name	Description
A,B,C	The three variables read and placed in order with A the smallest and C the largest.

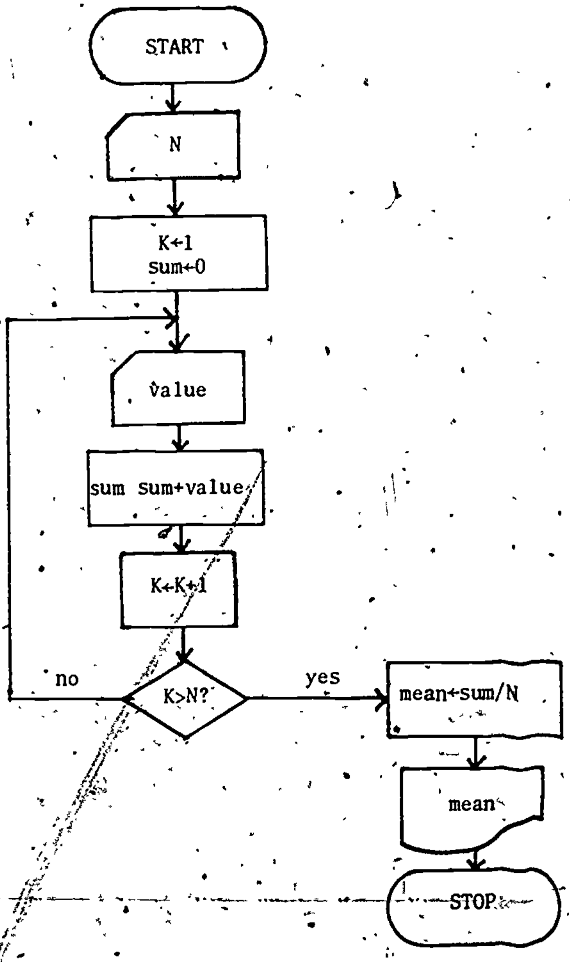


6. Precise formulation: Read a value N, followed by N real numbers and calculate the mean of the N real numbers.
Input: N and N real values.
Output: The mean of the N real values.

Algorithm:

Variables

Name	Description
N	The number of values read.
K	The counter from 1 to N.
sum	The sum of the values read.
value	The values whose mean is computed.
mean	The mean.



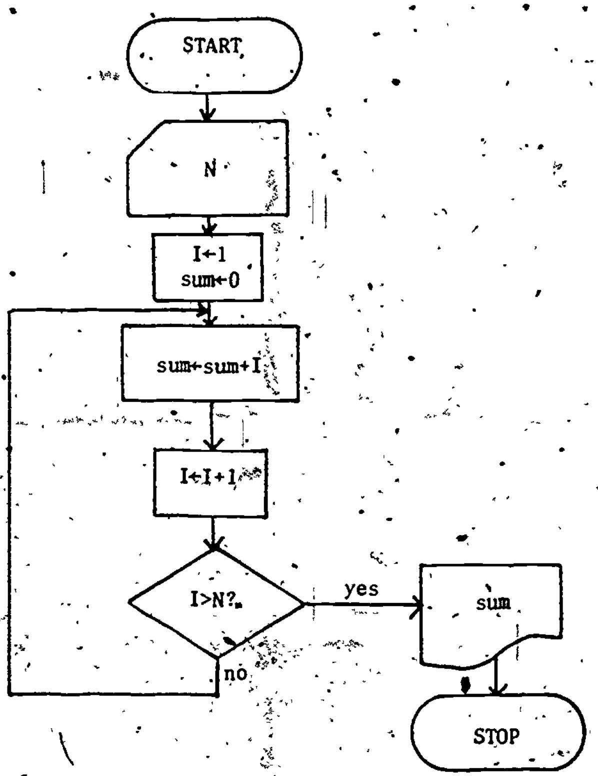
7. Precise formulation: Input a positive integer and compute the sum of all positive integers less than or equal to the integer input.

Input: A positive integer.

Output: The sum of all positive integers less than or equal to the integer input.

Algorithm: Variables

Name	Description
N	The number of integers summed.
I	A counter from 1 to N.
sum	The sum of the integers.



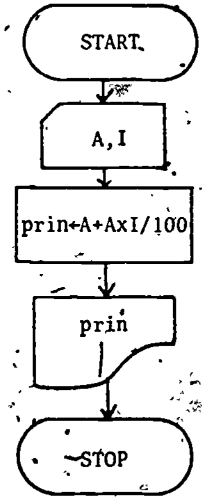
8. Precise formulation: Given a principal amount A in dollars and an interest rate I in percent per time period, compute the principal at the end of one time period.

Input: Principal amount A and interest rate I.

Output: Principal after one time period.

Algorithm: Variables

Name	Description
A	The amount of beginning principal.
I	The interest rate in percent.
prin	The amount of principal at the end of the time period.



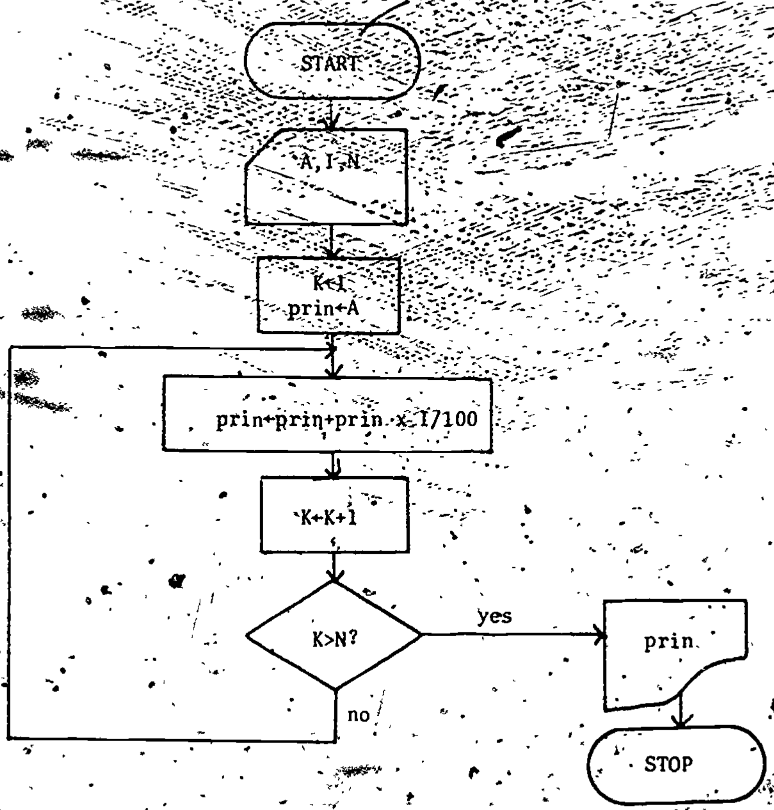
9. Precise formulation: Given a principal amount A in dollars, an interest rate I in percent per time period, and a number of time periods N, compute the principal at the end of N time periods.

Input: Principal amount A, interest rate I, and number of time periods N.

Output: Principal after N time periods.

Algorithm Variables

Name	Description
A	The amount of beginning principal
I	The interest rate in percent
N	The number of time periods
K	A counter from 1 to N
prin	The principal after K time periods



7. MODEL EXAM

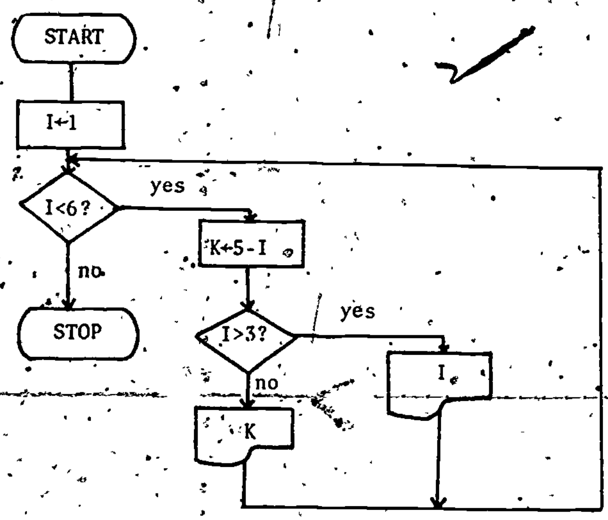
1. Determine which of the following are algorithms and which are nonalgorithms. For those that are nonalgorithms, explain why.

(a) How to shovel snow off a driveway.
   1. Get a shovel.
   2. Remove a shovelfull of snow from the driveway.
   3. If there is still snow on the driveway, go to step 2.
   4. Stop.

(b) How to buy a new car.
   1. Find the car you want.
   2. Find out its price.
   3. Ask your father for the money.
   4. Buy the car.

(c) How to find a job.
   1. Determine which jobs are open.
   2. Interview for the jobs you want.
   3. Decide which jobs you would like.
   4. Get the necessary training.
   5. Write a resume.

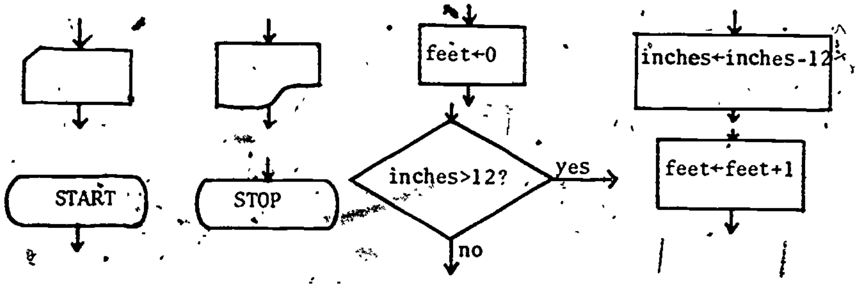
2. What output will the following algorithm generate?



3. Design an algorithm for solving the following problem: Convert a length from inches to feet and inches.

(a) Define the problem more precisely.

(b) The following are the boxes needed for an algorithm to solve this problem. Fill in the two empty boxes.



(c) Arrange the boxes above into a flowchart language algorithm to solve the stated problem. Use each box exactly once.

8. SOLUTIONS TO MODEL EXAM

1. (a) Algorithm
   (b) This does not solve the problem. In step 3, you do not receive the money by asking.
   (c) The steps are out of sequence.

2. 4
   3
   2
   4
   5

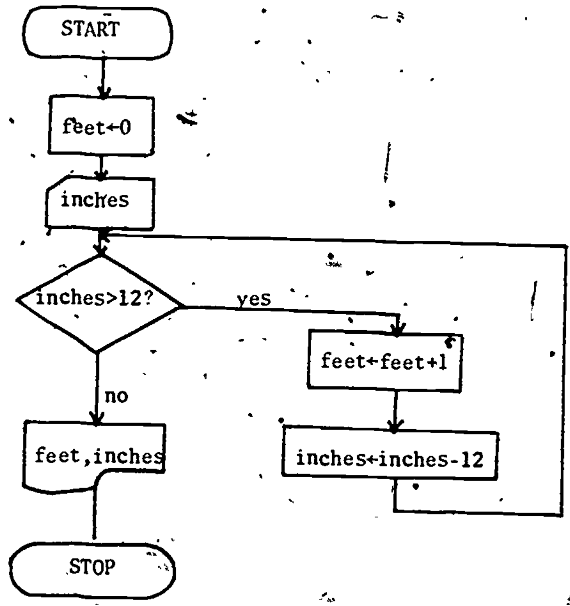
3. (a) The problem could more precisely say:

   Convert a length from inches to feet and inches where the number of feet is integer and as large as possible.

   (b) Input box should contain "inches".

   Output box should contain "feet, inches".

(c)



STUDENT FORM 1

Request for Help

Return to:
EDC/UMAP
55 Chapel St.
Newton, MA 02160

Student: If you have trouble with a specific part of this unit, please fill out this form and take it to your instructor for assistance. The information you give will help the author to revise the unit.

Your Name ____________________ Unit No. ______

Page ______ ○ Upper ○ Middle ○ Lower	OR	Section ______ Paragraph ______	OR	Model Exam Problem No. ______ Text Problem No. ______

Description of Difficulty: (Please be specific)

Instructor: Please indicate your resolution of the difficulty in this box.

○ Corrected errors in materials. List corrections here:

○ Gave student better explanation, example, or procedure than in unit. Give brief outline of your addition here:

○ Assisted student in acquiring general learning and problem-solving skills (not using examples from this unit.)

Instructor's Signature ____________________

Please use reverse if necessary.

STUDENT FORM 2

Unit Questionnaire

Return to:
EDC/UMAP
55 Chapel St.
Newton, MA 02160

Name__________ Unit No.__________ Date__________

Institution__________ Course No.__________

Check the choice for each question that comes closest to your personal opinion.

1. How useful was the amount of detail in the unit?

____Not enough detail to understand the unit
____Unit would have been clearer with more detail
____Appropriate amount of detail
____Unit was occasionally too detailed, but this was not distracting
____Too much detail; I was often distracted

2. How helpful were the problem answers?

____Sample solutions were too brief; I could not do the intermediate steps
____Sufficient information was given to solve the problems
____Sample solutions were too detailed; I didn't need them

3. Except for fulfilling the prerequisites, how much did you use other sources (for example, instructor, friends, or other books) in order to understand the unit?

____A Lot ____Somewhat ____A Little ____Not at all

4. How long was this unit in comparison to the amount of time you generally spend on a lesson (lecture and homework assignment) in a typical math or science course?

____Much Longer ____Somewhat Longer ____About the Same ____Somewhat Shorter ____Much Shorter

5. Were any of the following parts of the unit confusing or distracting? (Check as many as apply.)

____Prerequisites
____Statement of skills and concepts (objectives)
____Paragraph headings
____Examples
____Special Assistance Supplement (if present)
____Other, please explain__________

6. Were any of the following parts of the unit particularly helpful? (Check as many as apply.)

____Prerequisites
____Statement of skills and concepts (objectives)
____Examples
____Problems
____Paragraph headings
____Table of Contents
____Special Assistance Supplement (if present)
____Other, please explain__________

Please describe anything in the unit that you did not particularly like.

Please describe anything that you found particularly helpful. (Please use the back of this sheet if you need more space.)

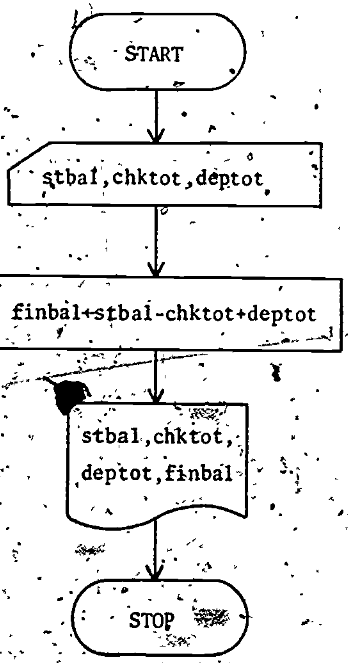
umap

UNIT 478

MODULES AND MONOGRAPHS IN UNDERGRADUATE MATHEMATICS AND ITS APPLICATIONS PROJECT

ITERATION AND COMPUTER PROBLEM SOLVING

by Herbert L. Dershem



COMPUTER SCIENCE/ALGORITHMS

edc/umap/55chapel st./newton,mass.02160

ITERATION AND COMPUTER PROBLEM SOLVING

by

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

TABLE OF CONTENTS

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: 9/30/80

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 which may be used to supplement existing courses and from which complete courses may eventually be built.

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

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. SED76-19615 A02. Recommendations expressed are those of the author and do not necessarily reflect the views of the NSF or the copyright holder.

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

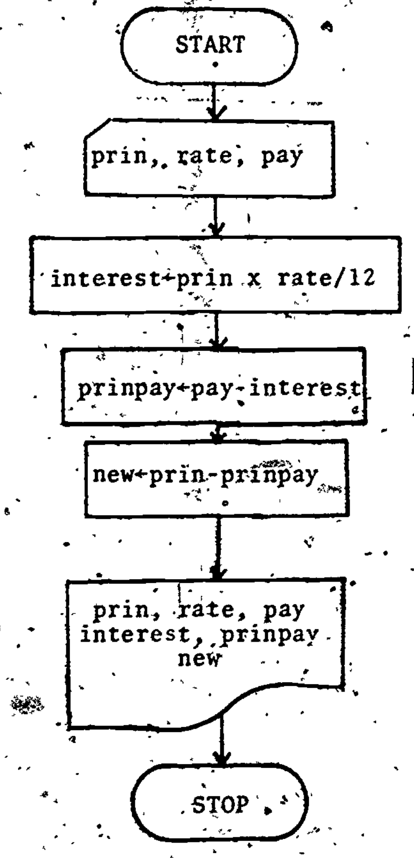
borrowed principal, a yearly interest rate, and a monthly payment, determine the new principal after one monthly payment has been made and determine the interest for one month.

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:

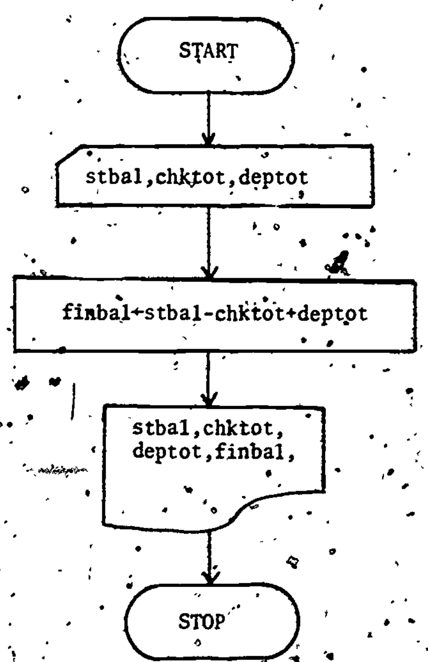
1. Choose sample values for prin, rate, and pay, and follow through algorithm 4 as a computer would.

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.

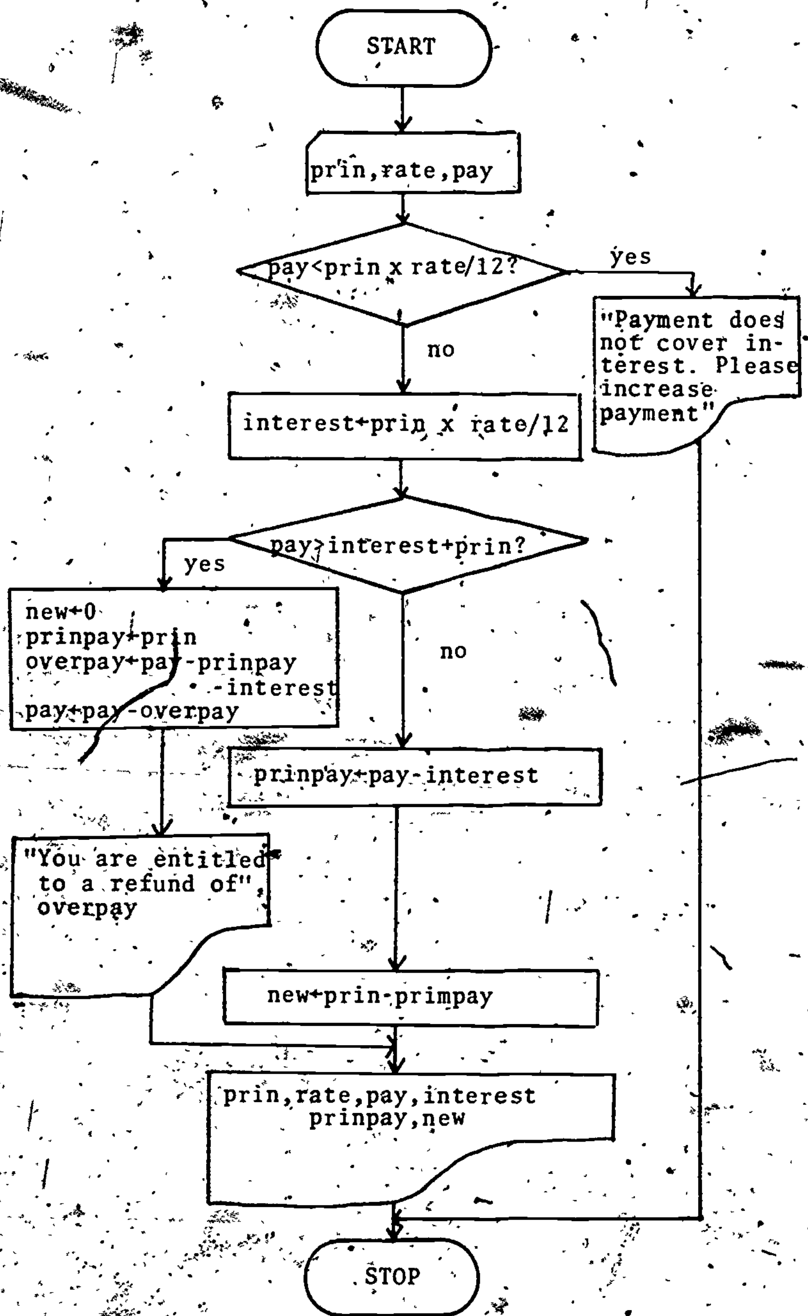
7 These modifications to Algorithm 1 are indicated in Algorithm 2. The decision box included immediately after the input box tests for a payment too small to cover the interest. When this is the case, we provide a message to the user and then immediately halt the algorithm. This action represents a policy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



Exercises:

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

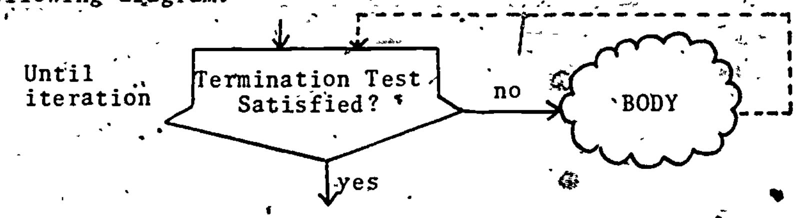
	Prin	rate	pay
Test 1	20000.	.12	250.
Test 2	20000.	.12	150.
Test 3	200.	.12	400.

---

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.

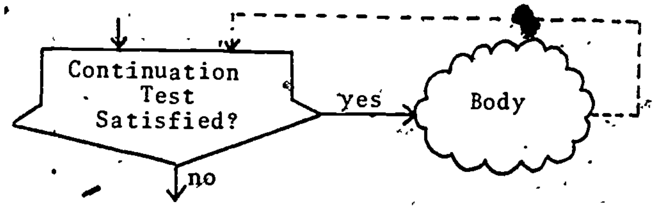
We shall discuss two forms of iteration here, and a third form in the next section. The first form will be called the *until iteration*. The iteration box which we use to describe this construction is indicated in the following diagram.



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.

While Iteration



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.

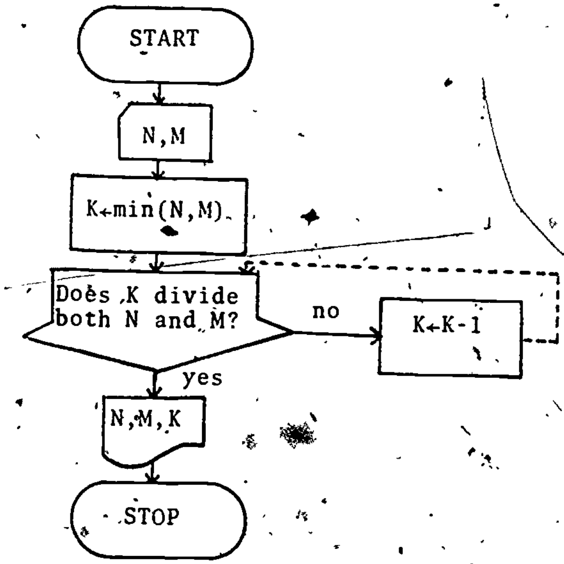
As an example of the use of the iterations, consider the algorithm for finding the greatest common divisor. We have two ways of expressing this algorithm using the iteration boxes just introduced.

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

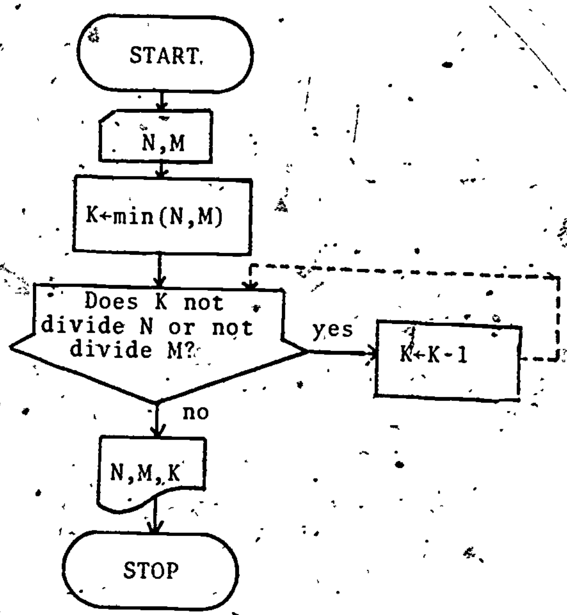
Variables

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



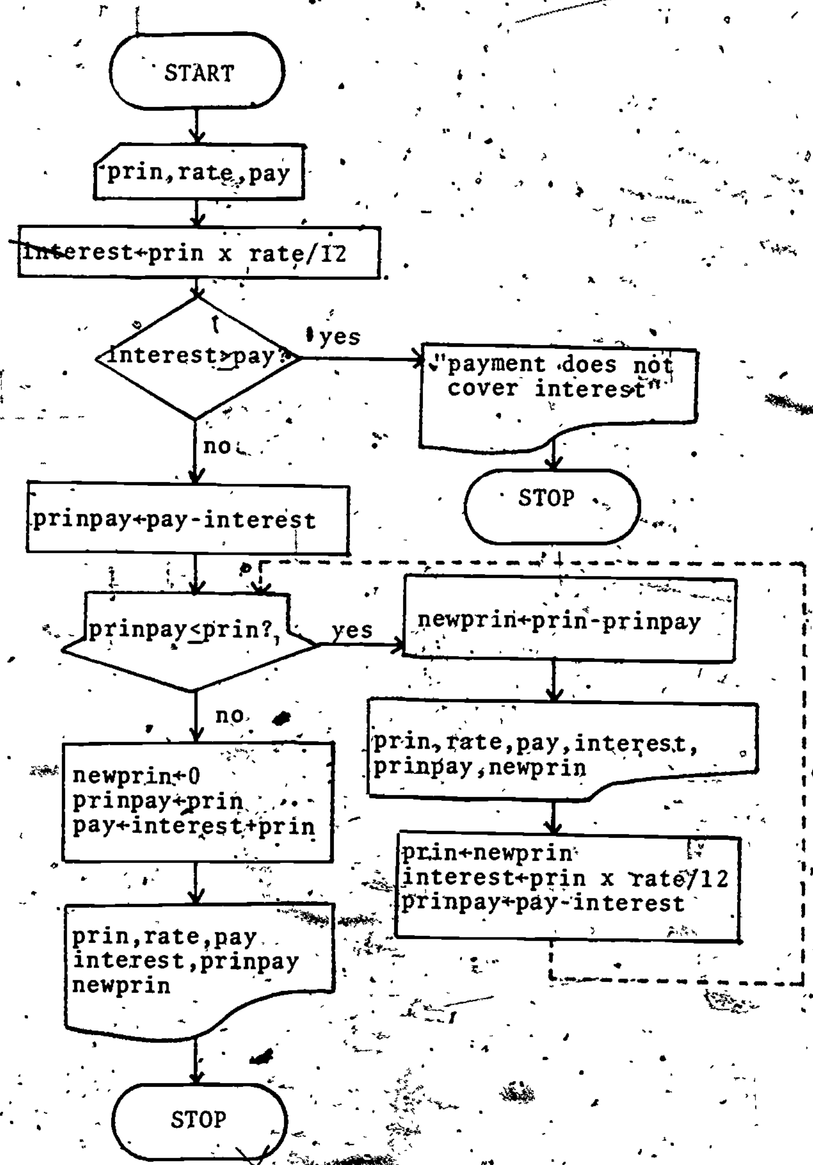
Algorithms 3a and 3b are logically equivalent in that they accomplish the same task. Note that the condition in the iteration box of Algorithm 3b is the logical negation of the one in the iteration box of Algorithm 3a. That is, you terminate precisely when you do not continue.

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

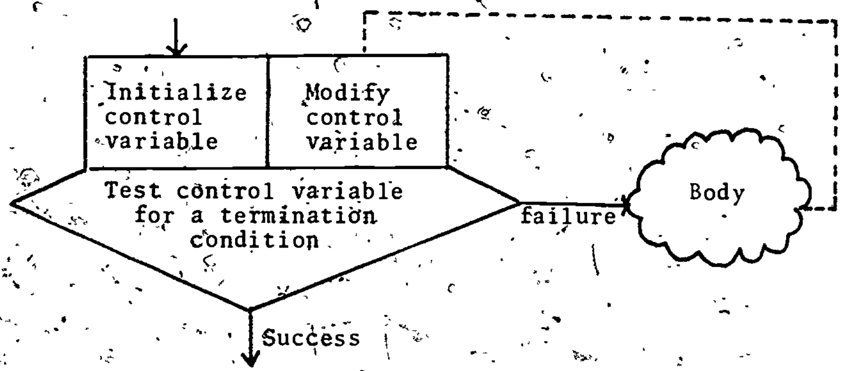
Name	Description
prin	Principal amount.
rate	Yearly rate of interest expressed as a decimal.
pay	Amount of the monthly payment.
interest	Part of the monthly payment which goes toward interest.
prinpay	Part of the monthly payment which goes toward principal.
newprin	New principal after a payment has been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



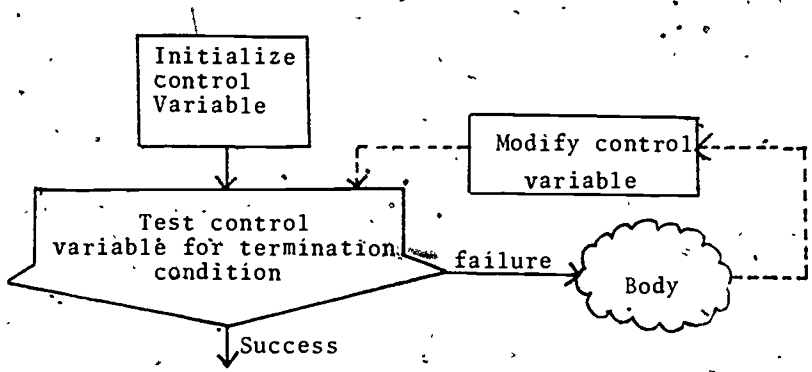
Much of this algorithm looks familiar from Algorithm 2. After the results for one month are printed in an iteration, that month's new principal is then set to be the next month's beginning principal. The iteration finally terminates when the amount paid on the principal exceeds the principal balance. 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.

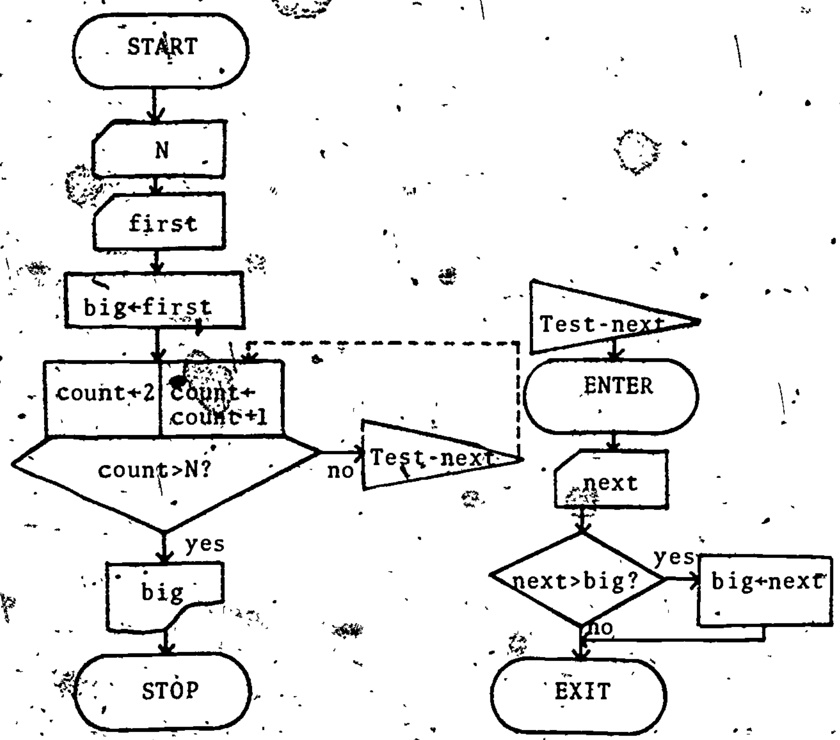


The reason we give the variable-controlled iteration its own form is that it is commonly used and is directly implemented in most programming languages.

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.

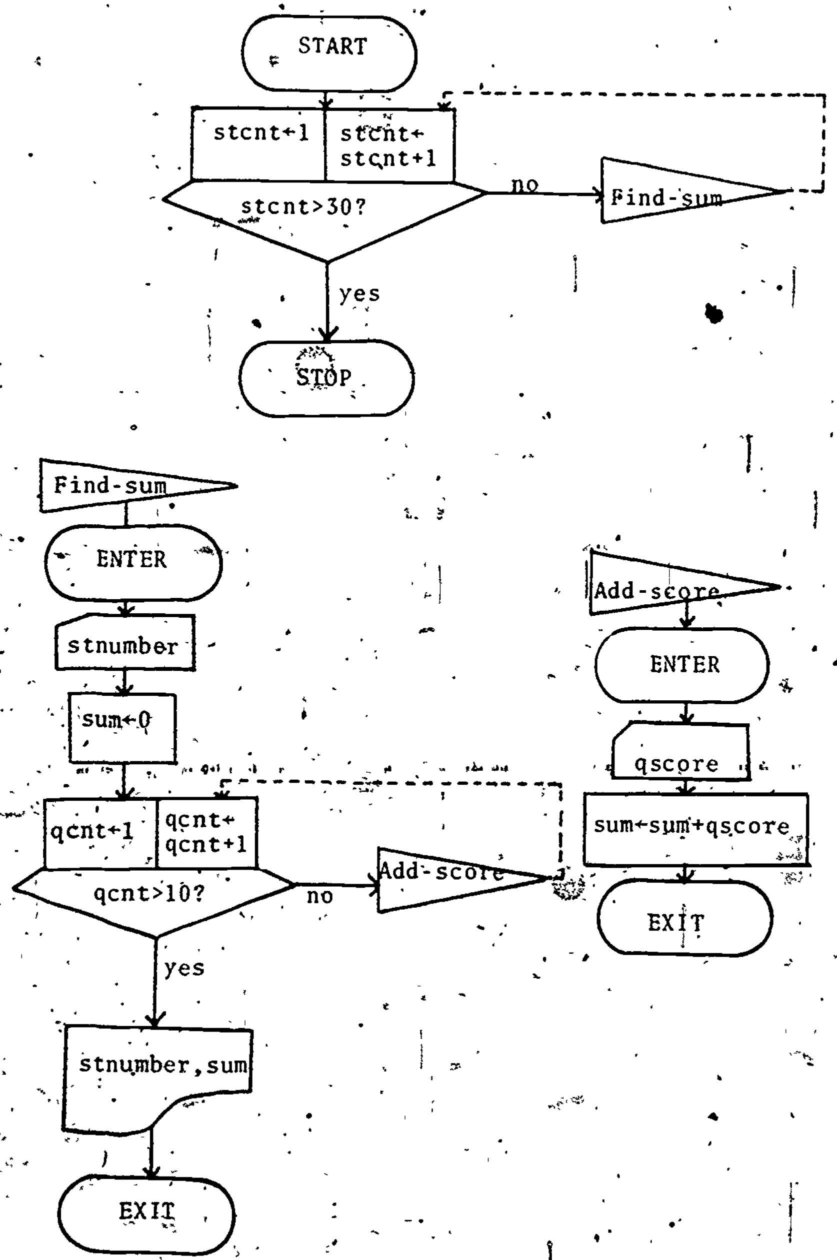
We now consider another example. Suppose there are 30 students in a class and each student has a student number and 10 quiz scores. The algorithm is to determine the sum of the quiz scores for each student. The algorithm for solving this problem is presented as Algorithm 6.

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.



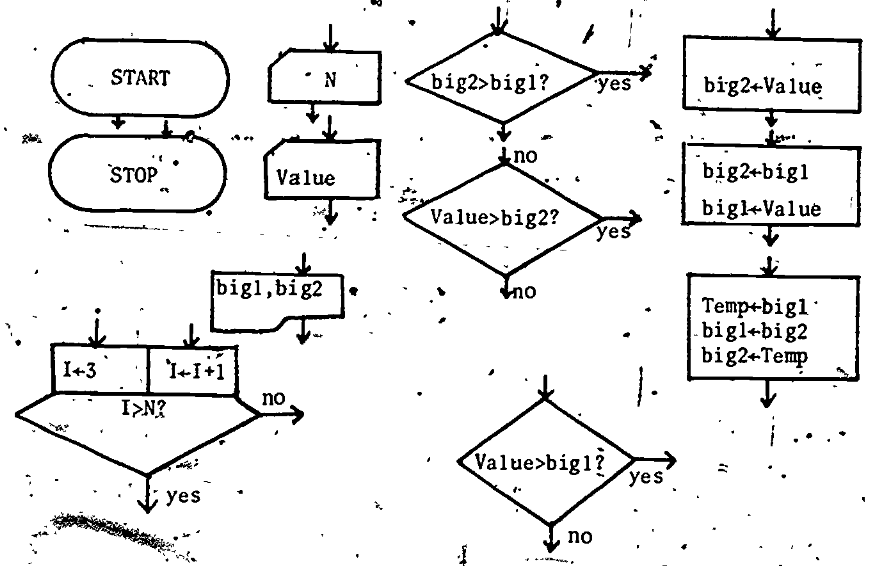
Exercises.

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

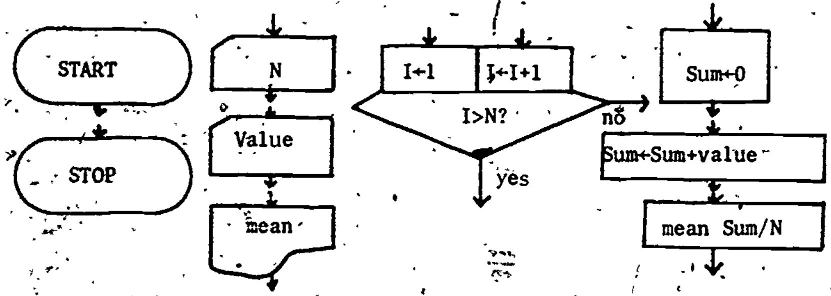
Name	Description
N	The number of values in the set.
Value	The value read.
big1	The biggest value read so far.
big2	The next-to-biggest value read so far.
I	A counter used to count the number of values read.



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

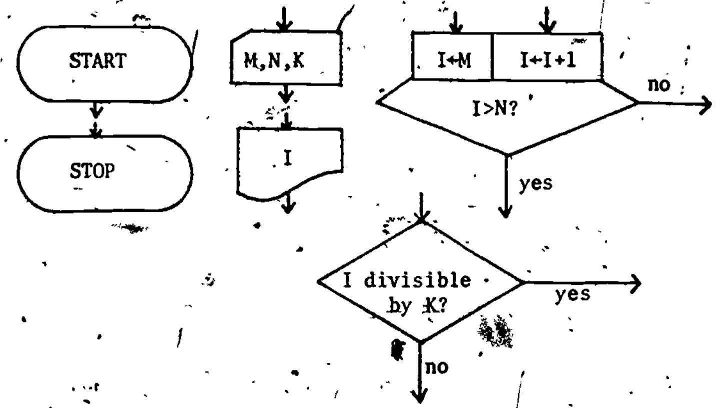
Name	Description
N	The number of values in the set.
Value	The value read.
Mean	The mean of the values.
I	A counter used to count the number of values read.
Sum	The sum of the values read.



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

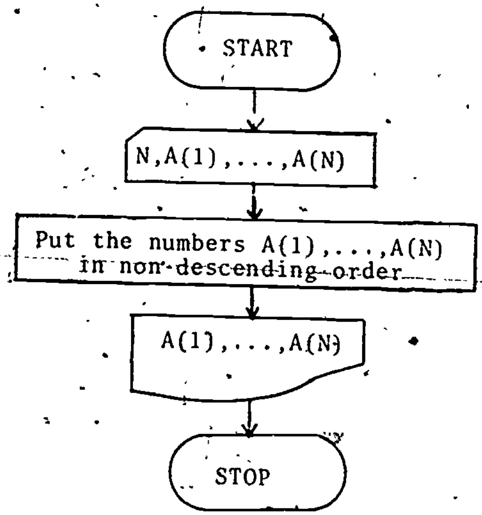
The top-down approach for designing algorithms is a technique that allows the designer to handle a complicated algorithm in a simple way. The basic approach is to design the algorithm using powerful boxes, then breaking those boxes down into flowcharts with less powerful boxes and continuing that process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

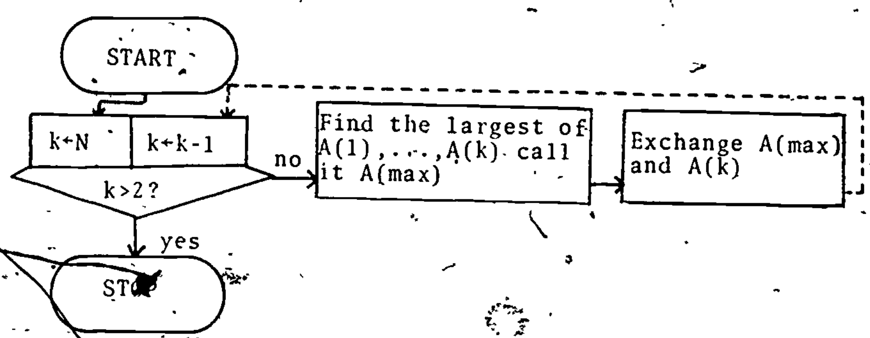


This is obviously not the final solution to our problem since we still have not described the process in enough detail for the computer to follow. The next step is to break the middle box itself down into an algorithm. This is the beginning of iterative refinement.

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).



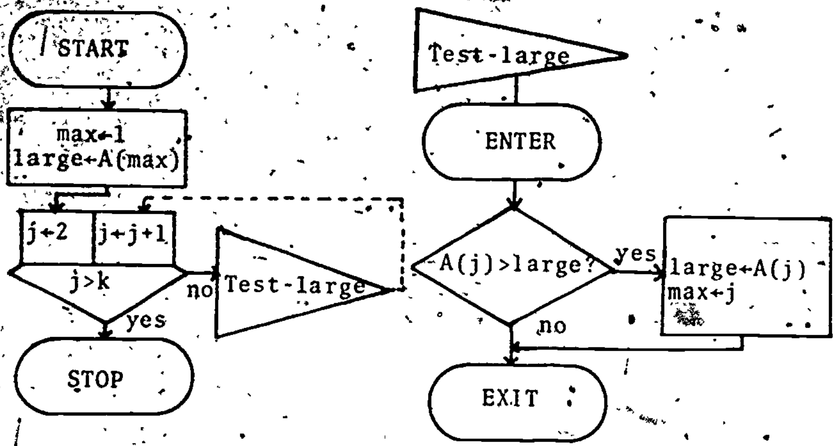
This procedure, for k=N down to k=2, is to find the largest of the first k values and place it at the kth position. The first time, with k=N, we find the largest in the entire set and put it at the bottom. The next pass through the iteration, with k=N-1, we ignore A(N) since it is already correct, and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

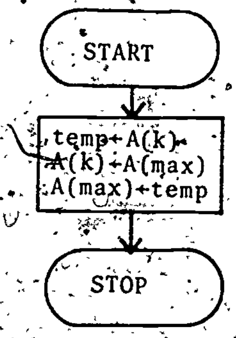


Finally, we expand the "Exchange..." box from Algorithm 9. This requires three data movements and one temporary location. It is given by the following algorithm.

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

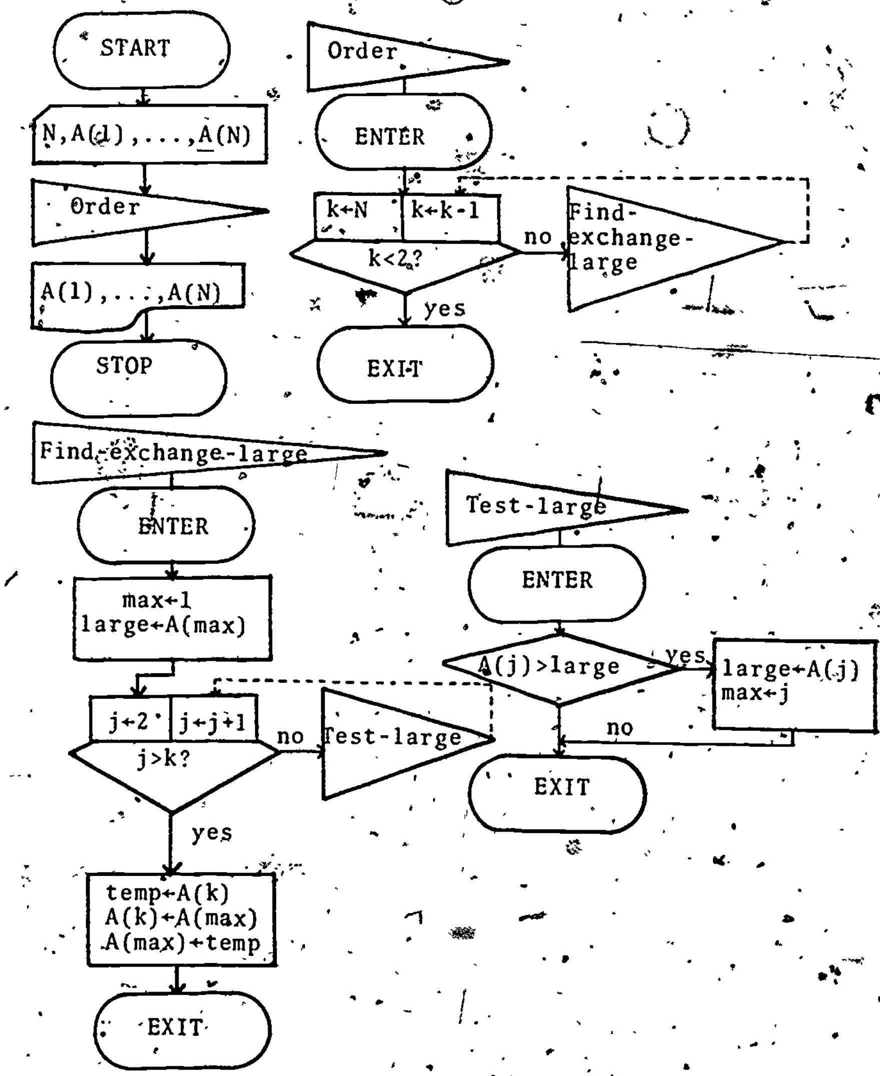
Variables

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:

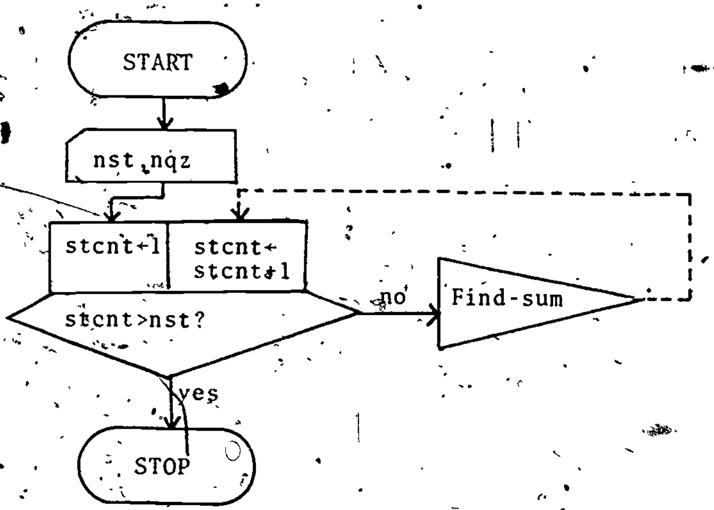
8. Using the set of numbers 5,3,9,6,2,7 follow through Algorithm 11 as a computer would.

---

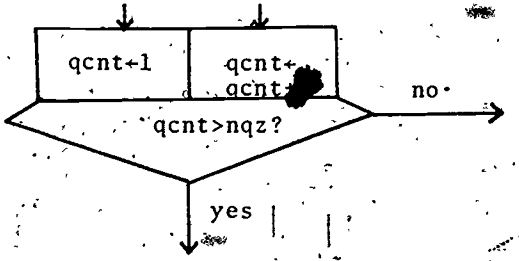
7. SOLUTIONS TO EXERCISES

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

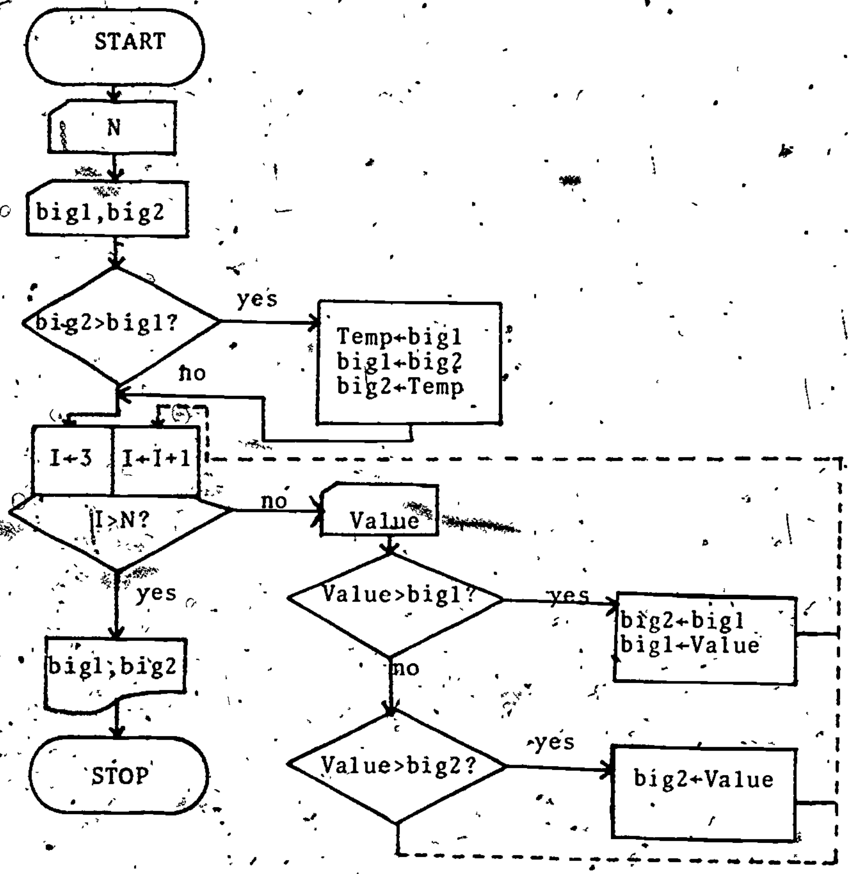
4. Change first flowchart to:



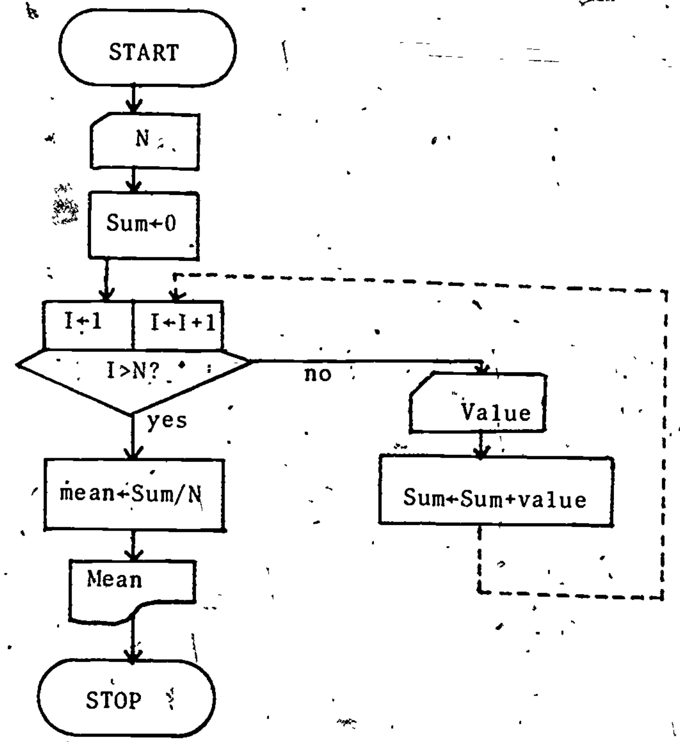
Change iteration box in Find-sum to



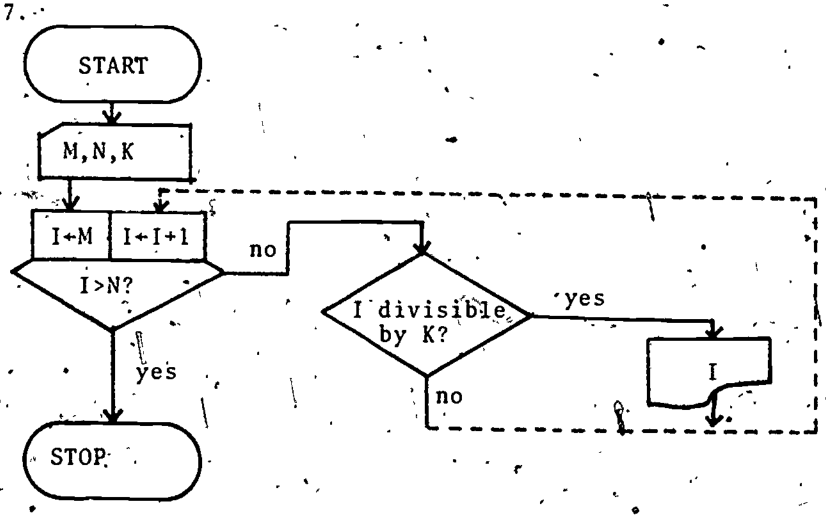
5.


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

6.

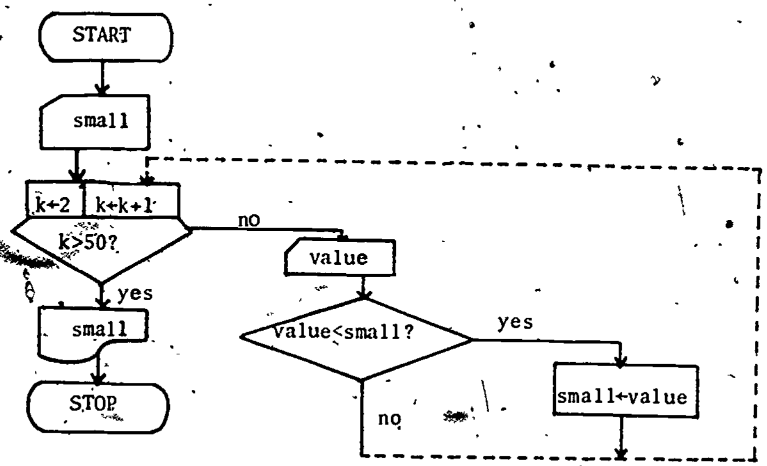

START
N
Sum←0
I←1
I←I+1
I>N?
no
Value
Sum←Sum+value
yes
mean←Sum/N
Mean
STOP

7.

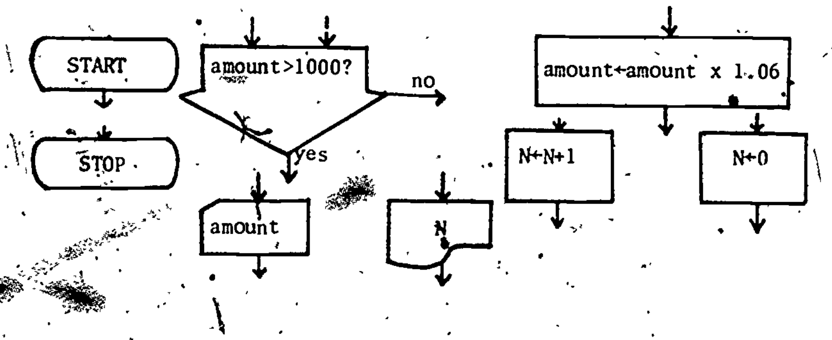

START
M,N,K
I←M
I←I+1
I>N?
no
I divisible by K?
yes
I
no
yes
STOP

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.



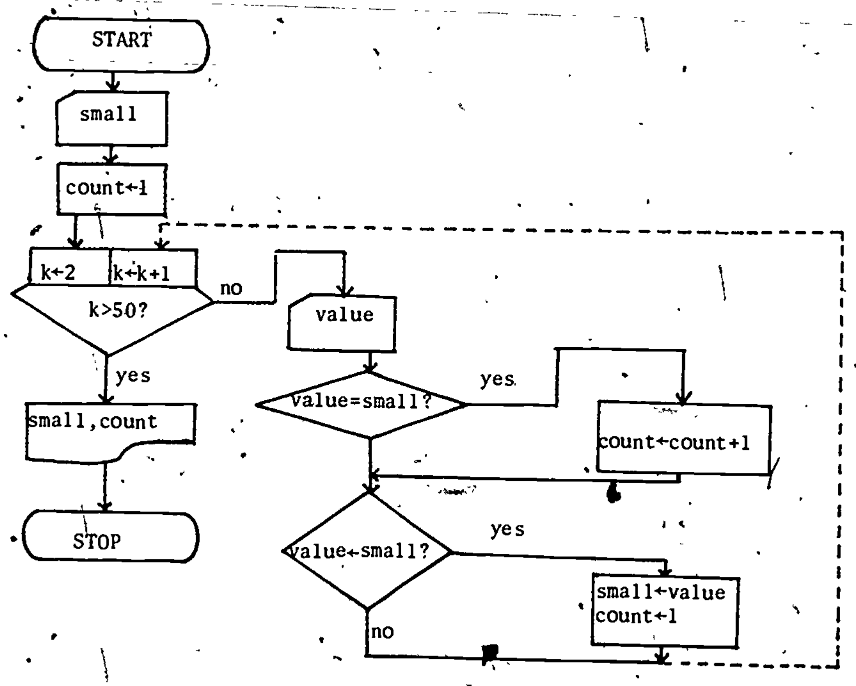
2. Write an algorithm which reads an amount of money and determines how many years it would take for that amount to grow to over $1000 if it accumulates interest at the rate of 6% compounded annually. Use each of the boxes below exactly once.



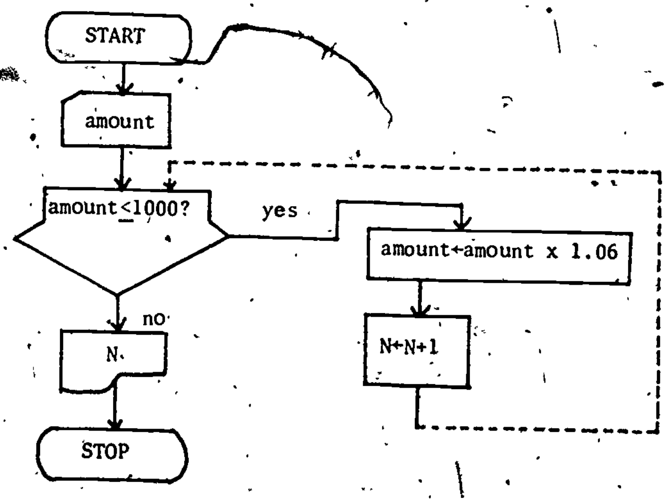
3. Explain in your own words the top-down approach to algorithm design. Discuss why it is important.

9. SOLUTIONS TO MODEL EXAM

1.



2.



STUDENT FORM 1

Request for Help

Return to:
EDC/UMAP
55 Chapel St.
Newton, MA 02160

Student: If you have trouble with a specific part of this unit, please fill out this form and take it to your instructor for assistance. The information you give will help the author to revise the unit.

Your Name________________________ Unit No.________

Page______ ○ Upper ○ Middle ○ Lower	OR	Section________ Paragraph________	OR	Model Exam Problem No.______ Text Problem No.______

Description of Difficulty: (Please be specific)

Instructor: Please indicate your resolution of the difficulty in this box.

○ Corrected errors in materials. List corrections here:

○ Gave student better explanation, example, or procedure than in unit. Give brief outline of your addition here:

○ Assisted student in acquiring general learning and problem-solving skills (not using examples from this unit.)

74

Instructor's Signature________________________

Please use reverse if necessary.

Return to:
EDC/UMAP
55 Chapel St.
Newton, MA 02160

STUDENT FORM 2

Unit Questionnaire

Name____________________ Unit No.__________ Date__________

Institution____________________ Course No.____________________

Check the choice for each question that comes closest to your personal opinion.

1. How useful was the amount of detail in the unit?

____Not enough detail to understand the unit
____Unit would have been clearer with more detail
____Appropriate amount of detail
____Unit was occasionally too detailed, but this was not distracting
____Too much detail; I was often distracted

2. How helpful were the problem answers?

____Sample solutions were too brief; I could not do the intermediate steps
____Sufficient information was given to solve the problems
____Sample solutions were too detailed; I didn't need them

3. Except for fulfilling the prerequisites, how much did you use other sources (for example, instructor, friends, or other books) in order to understand the unit?

____A Lot ____Somewhat ____A Little ____Not at all

4. How long was this unit in comparison to the amount of time you generally spend on a lesson (lecture and homework assignment) in a typical math or science course?

Much	Somewhat	About	Somewhat	Much
____Longer	____Longer	____the Same	____Shorter	____Shorter

5. Were any of the following parts of the unit confusing or distracting? (Check as many as apply.)

____Prerequisites
____Statement of skills and concepts (objectives)
____Paragraph headings
____Examples
____Special Assistance Supplement (if present)
____Other, please explain____________________

6. Were any of the following parts of the unit particularly helpful? (Check as many as apply.)
____Prerequisites
____Statement of skills and concepts (objectives)
____Examples
____Problems
____Paragraph headings
____Table of Contents
____Special Assistance Supplement (if present)
____Other, please explain____________________

Please describe anything in the unit that you did not particularly like.

Please describe anything that you found particularly helpful. (Please use the back of this sheet if you need more space.)