UMAP

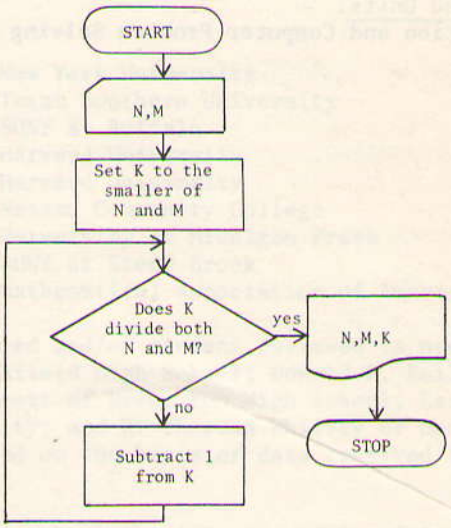
MODULES AND MONOGRAPHS IN UNDERGRADUATE MATHEMATICS AND ITS APPLICATIONS

Birkhäuser Boston Inc.
380 Green Street
Cambridge, MA 02139

MODULE 477

Computer Problem Solving

Herbert L. Dershem



Computer Science/Algorithms

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: IV 6/30/81

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 flowchart 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 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 supported, nonprofit corporation engaged in educational research in the U.S. and abroad.

This unit was field-tested and/or student reviewed in preliminary form by Harold Baker of Litchfield High School; Donald F. Bailey of Cornell College; Alex G. Bennett of Bremerton High School; Larry Sowder of Northern Illinois University; and W. Thurmon Whitley of University of New Haven and has been revised on the basis of data received from these sites.

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. SED80-07731. Recommendations expressed are those of the author and do not necessarily reflect the views of the NSF or the copyright holder.

COMPUTER PROBLEM SOLVING

by

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

TABLE OF CONTENTS

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.

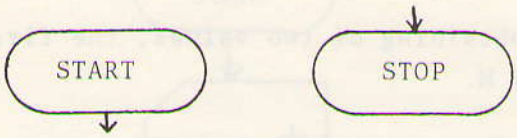
It would appear then, that a computer programming language may be an answer to our dilemma. Programming languages are, of necessity, unambiguous, and are understandable to the computer. And indeed one of our goals is to express algorithms in this form so that the computer can solve the problem. But people 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.

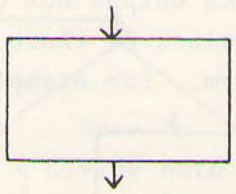
There are five different kinds of boxes that are used in representing algorithms.

1. Terminal box



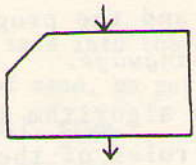
Terminal boxes are used to identify the starting point and stopping point of an algorithm. They will always contain either the word START or the word STOP.

2. Processing box

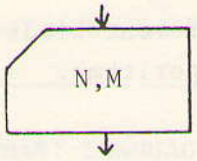


The work of the algorithm is specified in processing boxes. It contains an English statement which describes the action that is to be taken.

3. Input box

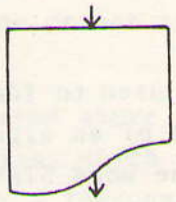


Each input box contains the names of variables whose values are to be obtained from a known source. For example, the box

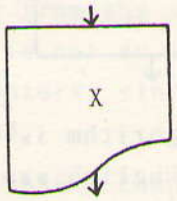


represents the obtaining of two values, the first called N and the second M.

4. Output box

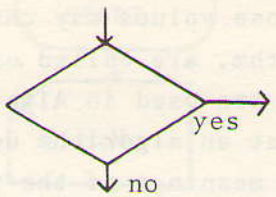


Like an input box, each output box contains the names of variables. The values of these variables are to be displayed in some form. For example, the box



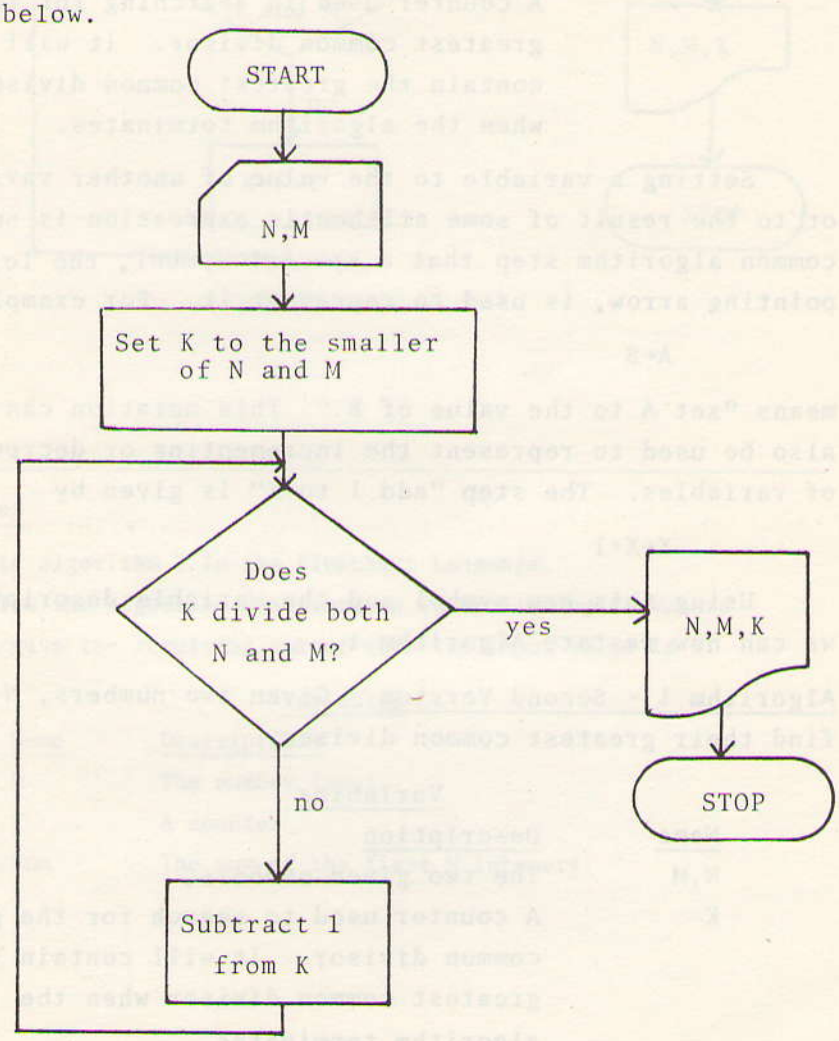
indicates that the current value of X is to be displayed.

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←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

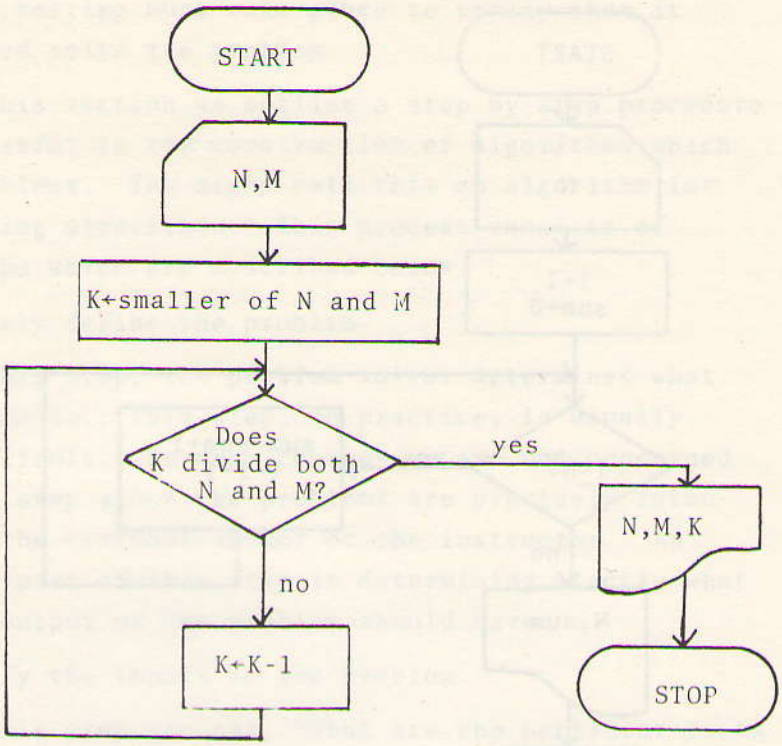
X←X+1

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.

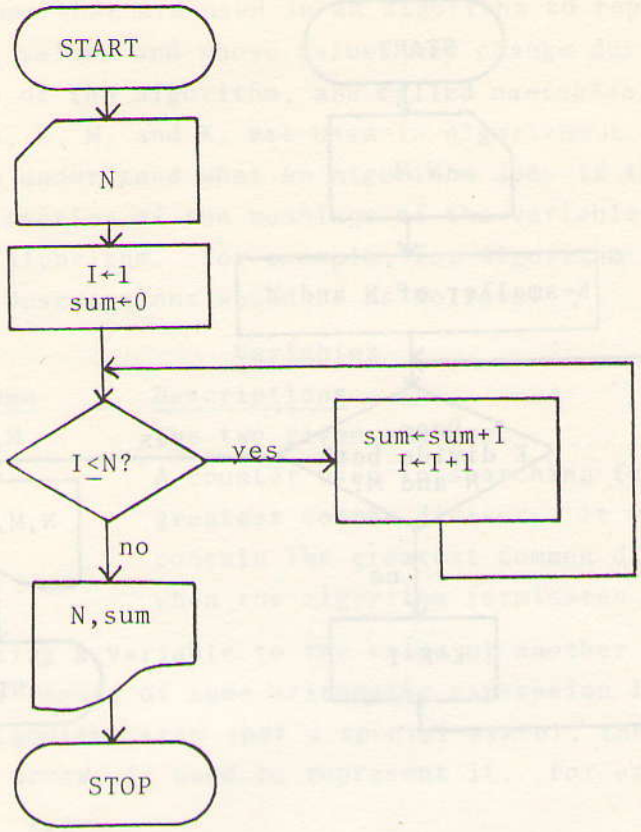


Exercises

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 originate 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:

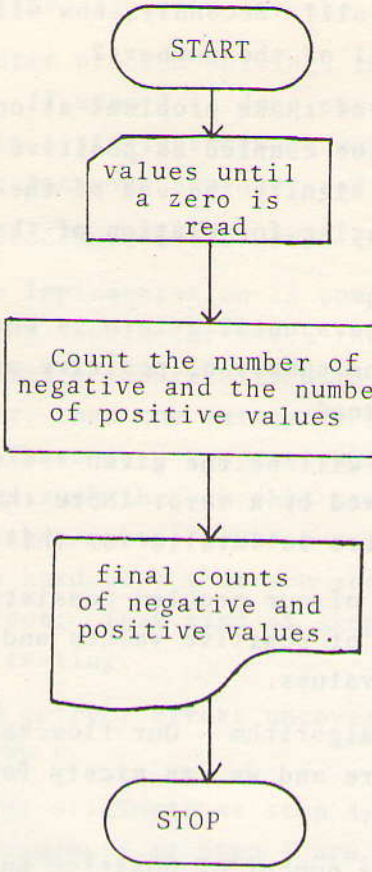
> Read a set of numbers until a zero is encountered and count the number of positive and negative numbers read.

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:

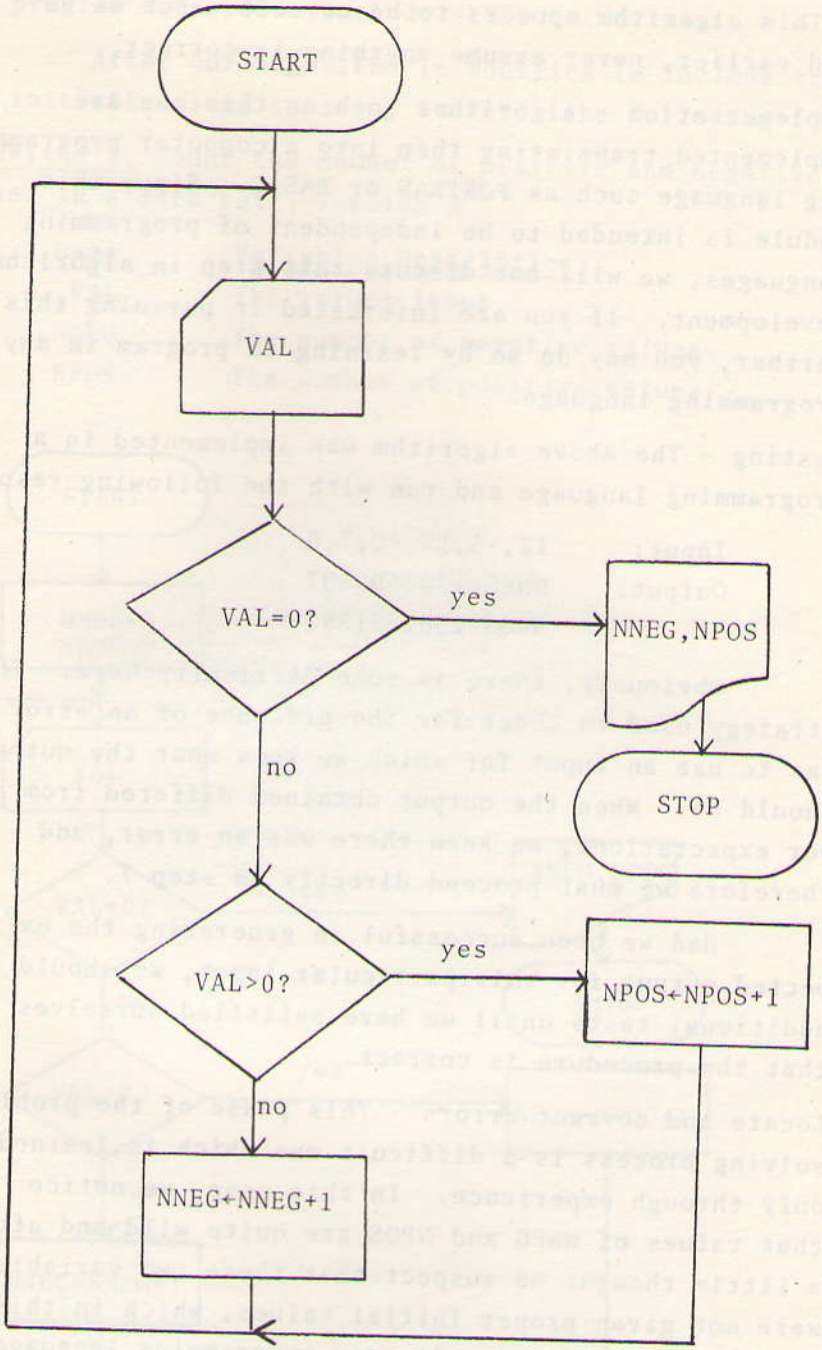
Algorithm 2. Count the number of positive and negative values in a data set - Version I.



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

Algorithm 3. 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.


START
VAL
VAL=0?
yes
NNEG,NPOS
STOP
no
VAL>0?
yes
NPOS←NPOS+1
no
NNEG←NNEG+1

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.

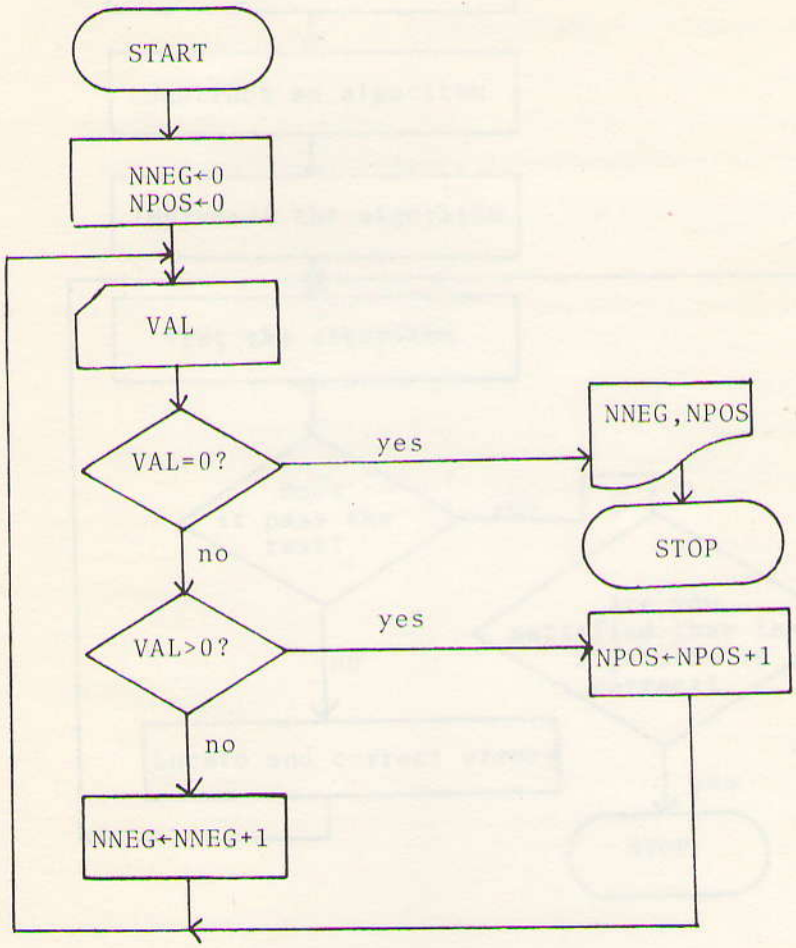
7. Locate and correct errors - This phase of the problem solving process is a difficult one which is learned only through experience. In this case, we notice that values of NNEG and NPOS are quite wild and after a little thought we suspect that these two variables were not given proper initial values, 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.

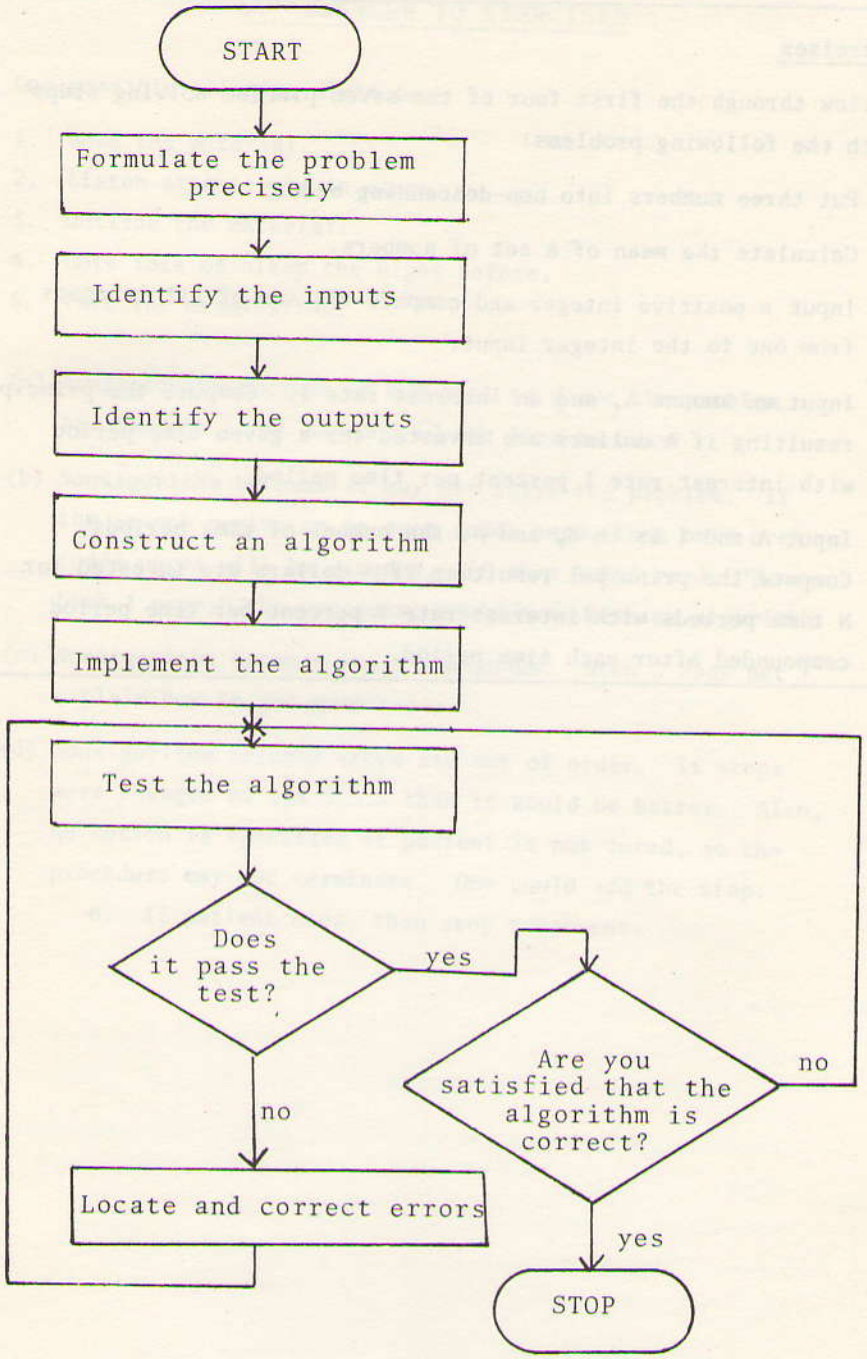
Algorithm 4. Count the number of positive and negative values in a data set - Version 3.

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.


START
Formulate the problem precisely
Identify the inputs
Identify the outputs
Construct an algorithm
Implement the algorithm
Test the algorithm
Does it pass the test?
yes
no
Are you satisfied that the algorithm is correct?
no
yes
Locate and correct errors
STOP

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 a positive 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 8, 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?

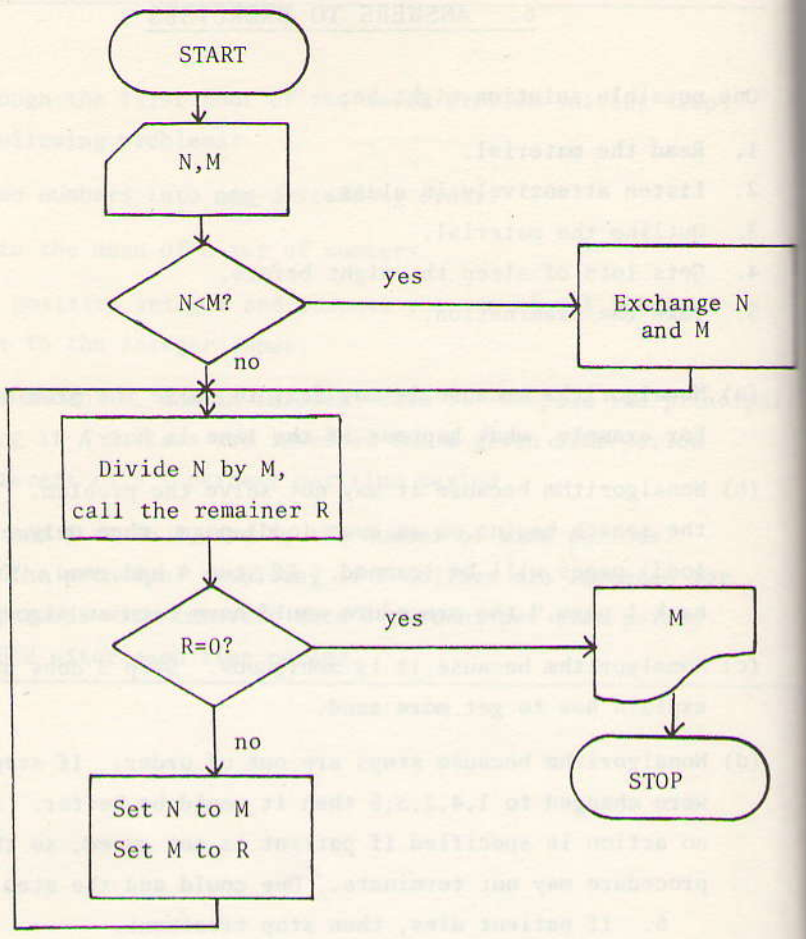
   (b) Nonalgorithm because it may not solve the problem. If the search begins on an even (odd) page, then only even (odd) pages will be scanned. If step 4 had read, "Turn back 1 page," the procedure would have been an algorithm.

   (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

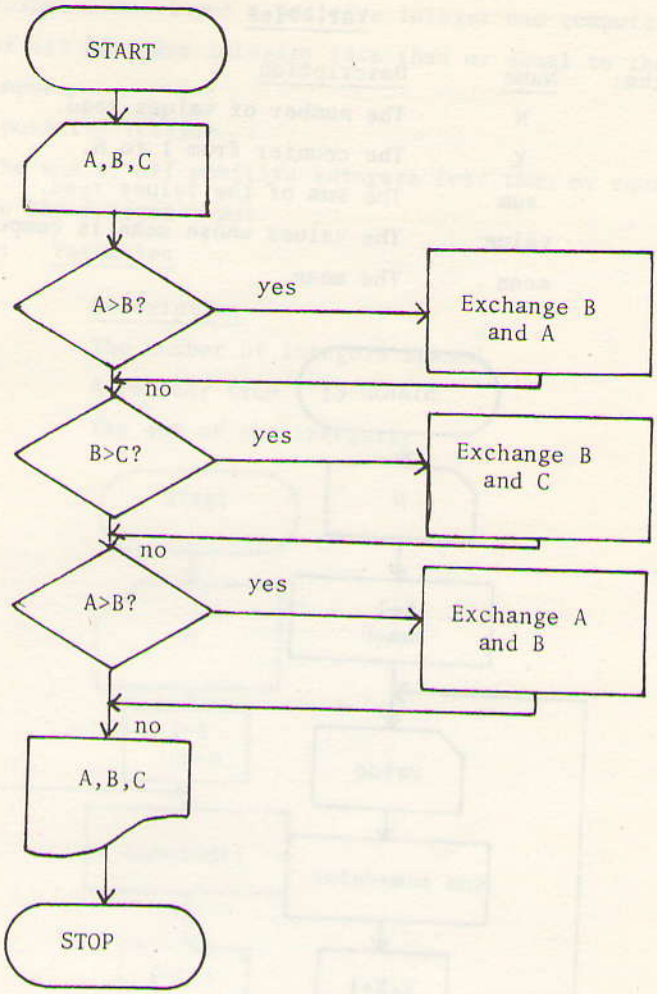
5. Precise formulation: Put three real numbers into non-descending order.

   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.

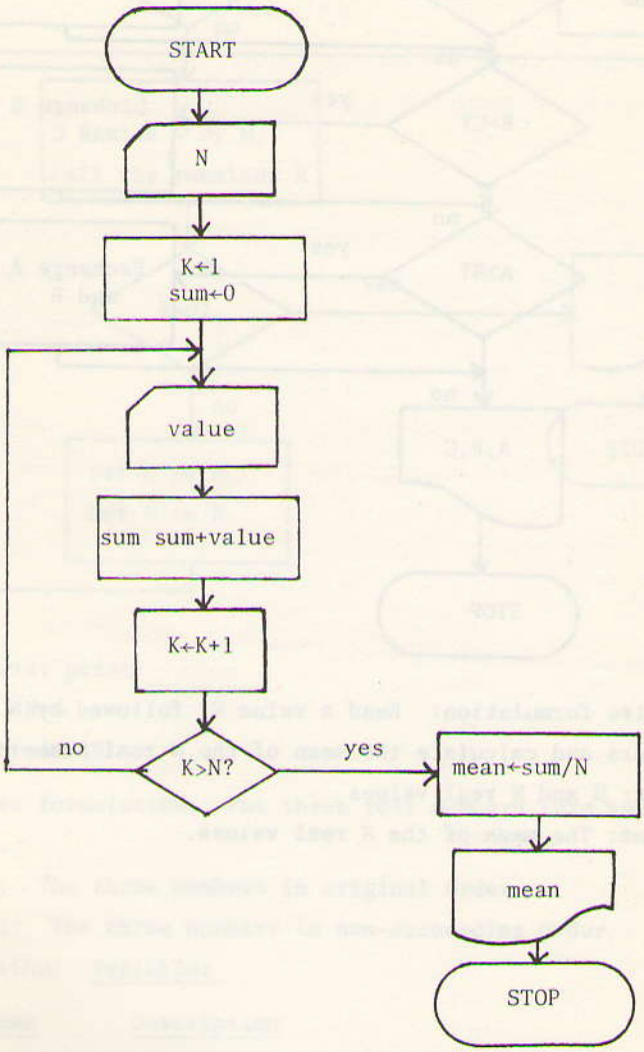


6. Precise formulation: Read a value N, followed by N real numbers and calculate the mean of the N real numbers.
   Input: N and N real values.
   Output: The mean of the N real values.

Variables

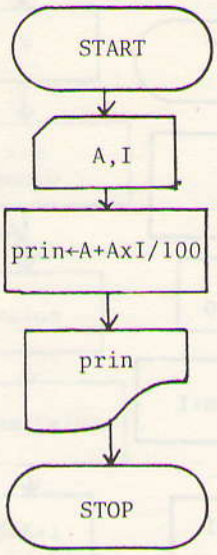
Algorithm:

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.



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

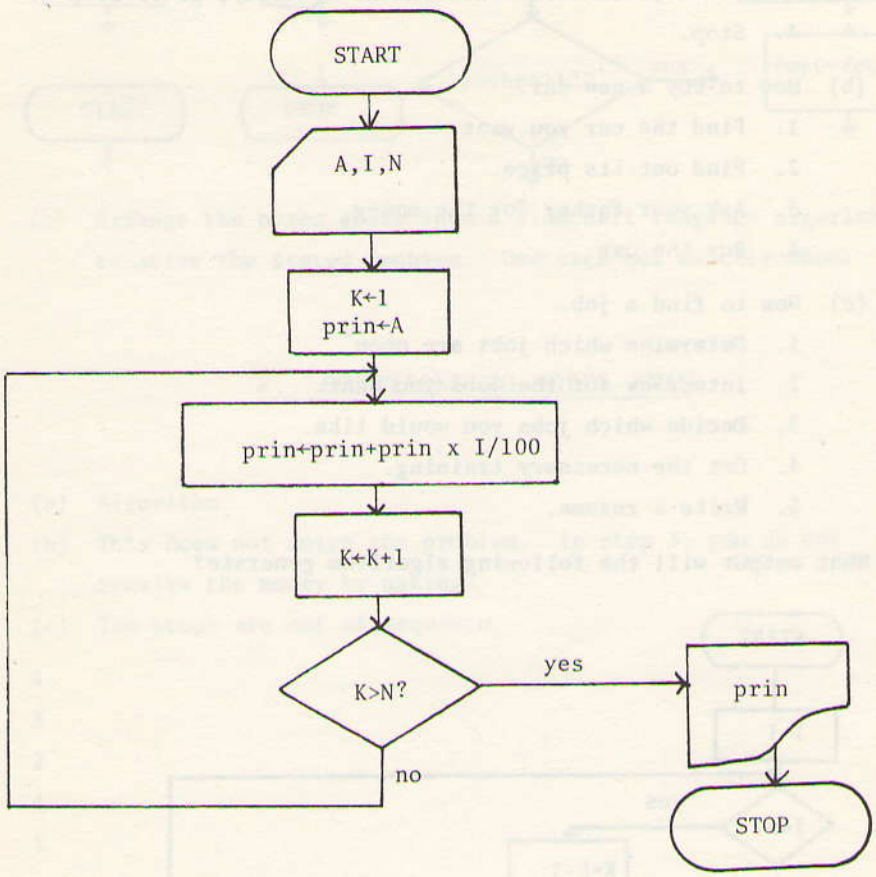
Name	Description
A	The amount of beginning principal.
I	The interest rate in percent.
prin	The amount of principal at the end of the time period.



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

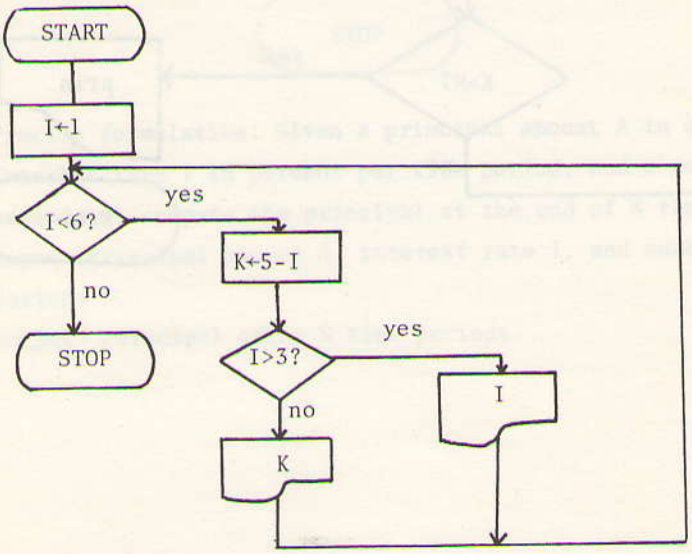
1. Determine which of the following are algorithms and which are nonalgorithms. For those that are nonalgorithms, explain why.

   (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.

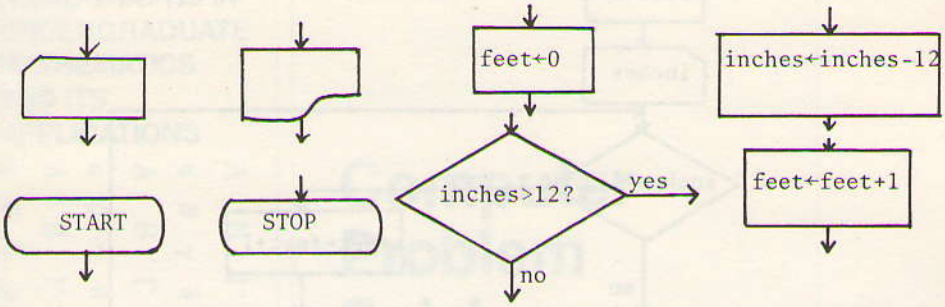
2. What output will the following algorithm generate?



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

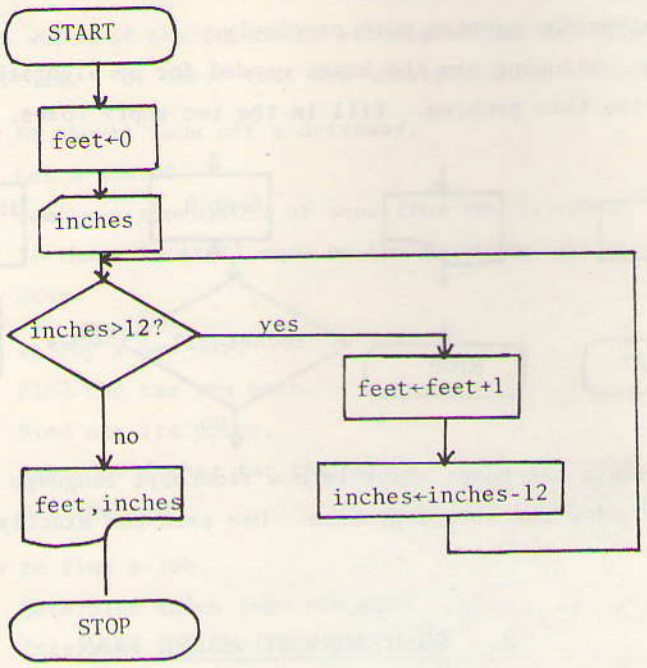
3. (a) The problem could more precisely say:

   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)


START
feet←0
inches
inches>12?
yes
feet←feet+1
no
feet,inches
inches←inches-12
STOP