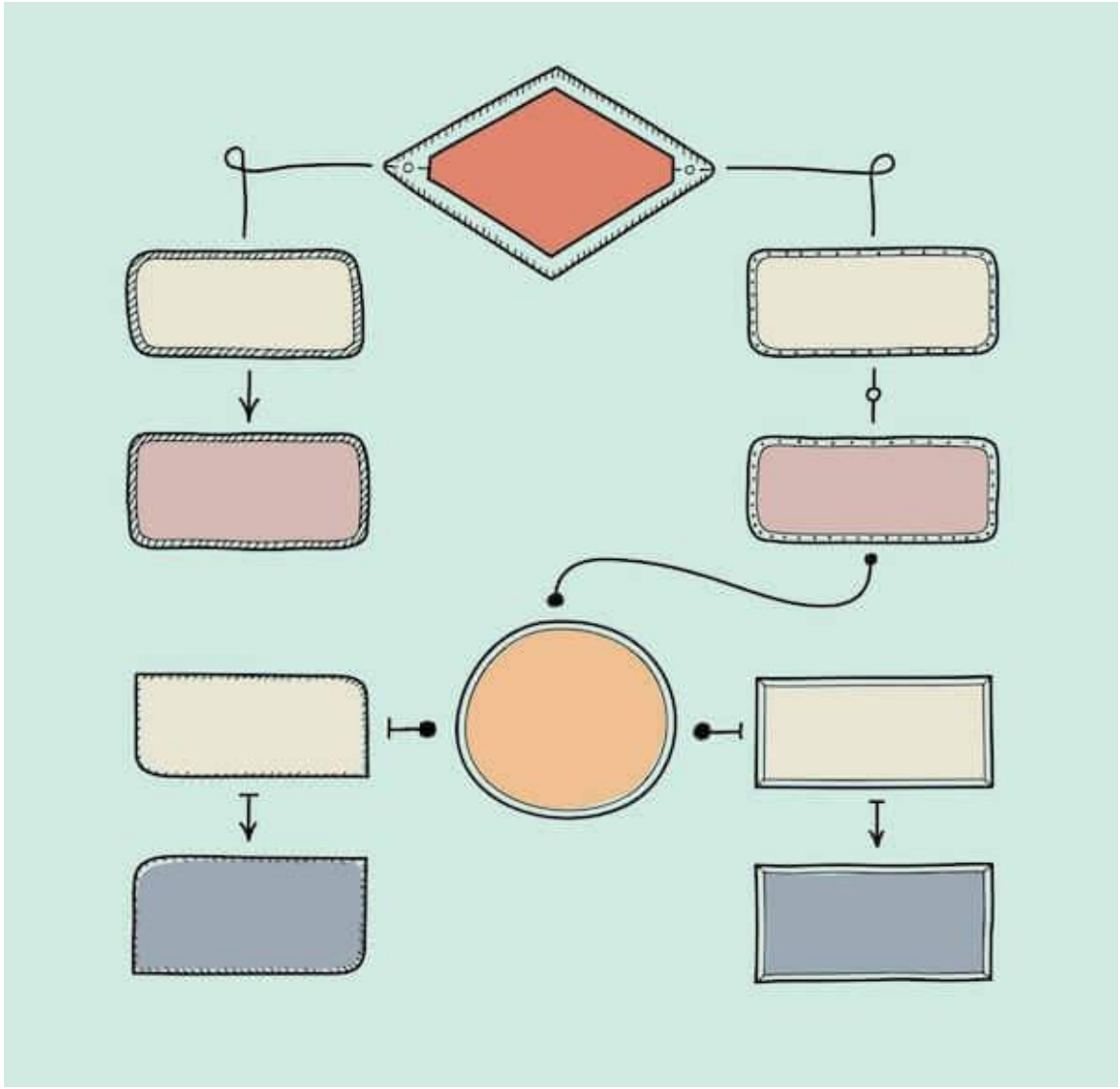
# Flowchart



# CHAPTER ONE: INTRODUCTION TO PROGRAMMING

# ALGORITHM

A set of steps that generates a finite sequence of elementary computational operations leading to the solution of a given problem is called an *algorithm*.

# FLOWCHART

a flowchart is a diagrammatic representation of the steps of an algorithm.

# SOLUTION STRATEGIES

The different ways of solving a problem are called *solution strategies*

Flowcharts may be classified into two categories:

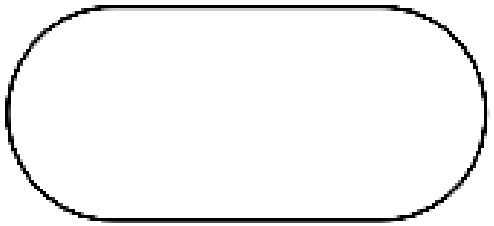
**(i)** Program Flowchart **(ii)** System Flowchart

A *program flowchart* is an extremely useful tool in program development. First, any error or omission can be more easily detected from a program flowchart than it can be from a program because a program flowchart is a pictorial representation of the logic of a program. Second, a program flowchart can be followed easily and quickly. Third, it serves as a type of documentation, which may be of great help if the need for program modification arises in future.

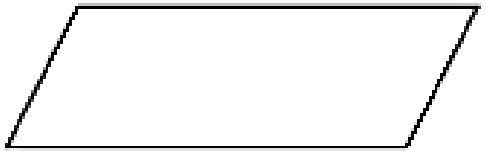
Flowcharts can be used to show the sequence of steps for doing any job. A set of simple operations involving accepting inputs, performing arithmetic operation on the inputs, and showing them to the users demonstrate the *sequence logic structure* of a program. The following flowchart shows the steps in cooking rice and then utilizing the cooked rice.

The algorithm for the flowchart about cooking rice is as follows:

**Step 1.** Take the rice to be cooked.
**Step 2.** Procure the container.
**Step 3.** Procure the water.
**Step 4.** Wash the rice in the water.
**Step 5.** Put the rice into the container.
**Step 6.** Pour water into the container.
**Step 7.** IF WATER LEVEL = 1 INCH ABOVE THE RICE
THEN GOTO STEP 8
ELSE GOTO STEP 6
ENDIF
**Step 8.** Light the burner on the stove.
**Step 9.** IF THE RICE IS BOILED
THEN GOTO STEP 12
ELSE GOTO STEP 10
ENDIF
**Step 10.** Heat the container.
**Step 11.** Go to step 9.
**Step 12.** Turn off the flame.
**Step 13.** Move the container off the stove.
**Step 14.** Distribute the cooked rice.
**Step 15.** STOP.



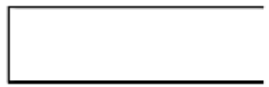
*Terminal*: used to show the beginning and end of a set of computer-related processes



*Input/Output*: used to show any input/output operation

*Computer processing*: used to show any processing performed by a computer system

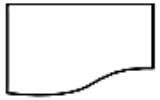
*Predefined processing*: used to indicate any process not specially defined in the flowchart



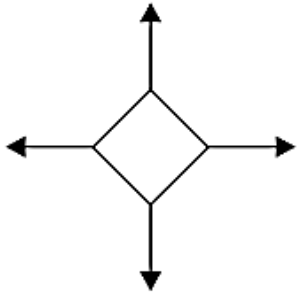
*Comment*: used to write any explanatory statement required to clarify something



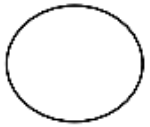
*Flow line*: used to connect the symbols



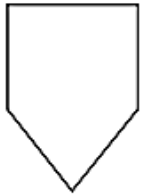
*Document Input/Output*: used when input comes from a document and output goes to a document.



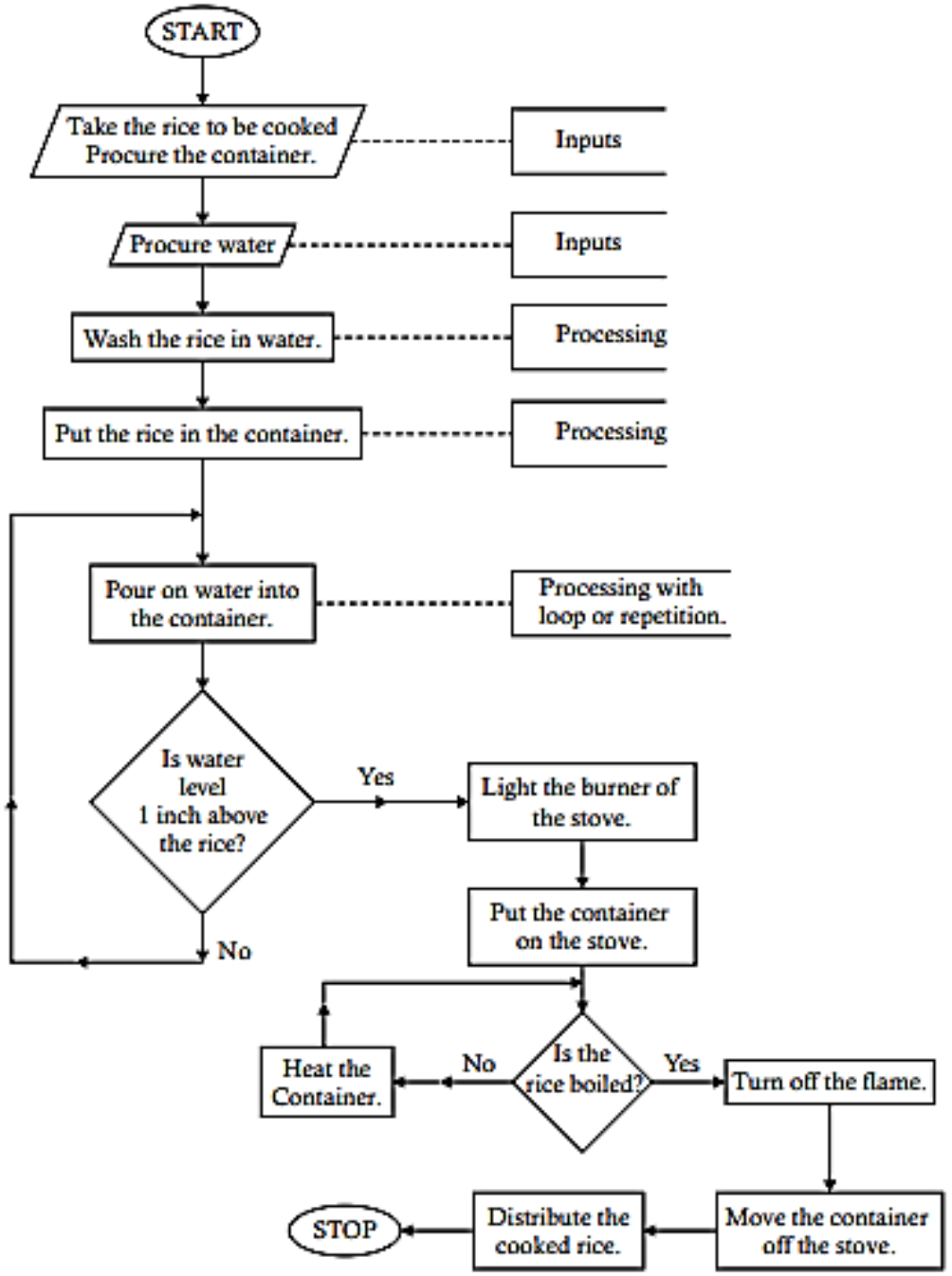
*Decision*: used to show any point in the process where a decision must be made to determine further action



*On-page connector*: used to connect parts of a flowchart continued on the same page

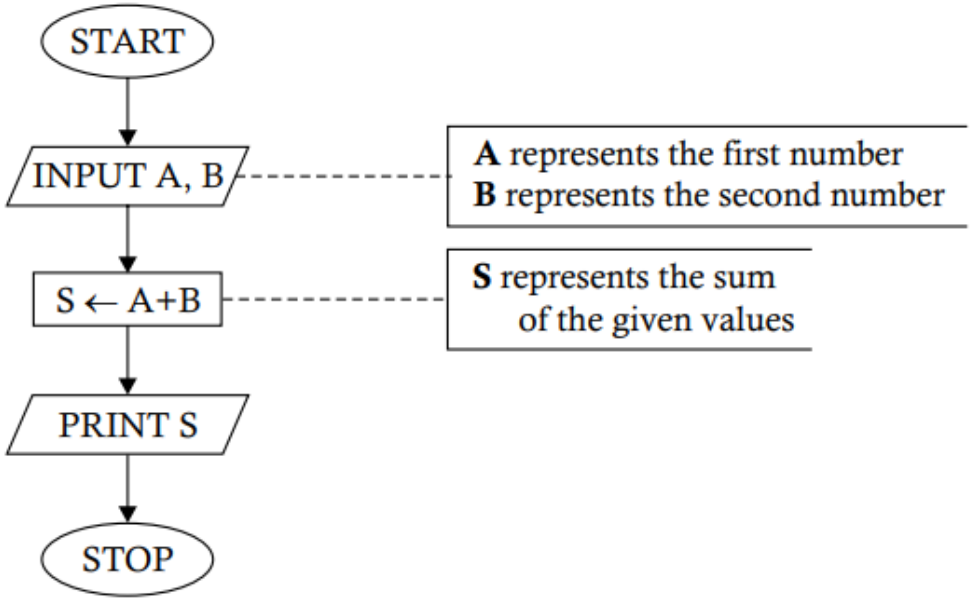


*Off-page connector*: used to connect parts of a flowchart continued to separate pages


START
Take the rice to be cooked
Procure the container.
Inputs
Procure water
Inputs
Wash the rice in water.
Processing
Put the rice in the container.
Processing
Pour on water into the container.
Processing with loop or repetition.
Is water level 1 inch above the rice?
Yes
No
Light the burner of the stove.
Put the container on the stove.
Is the rice boiled?
No
Heat the Container.
Yes
Turn off the flame.
Move the container off the stove.
Distribute the cooked rice.
STOP

# PROBLEMS

**Problem 1.1.** *Draw a flowchart to show how the sum of two numbers can be obtained.*

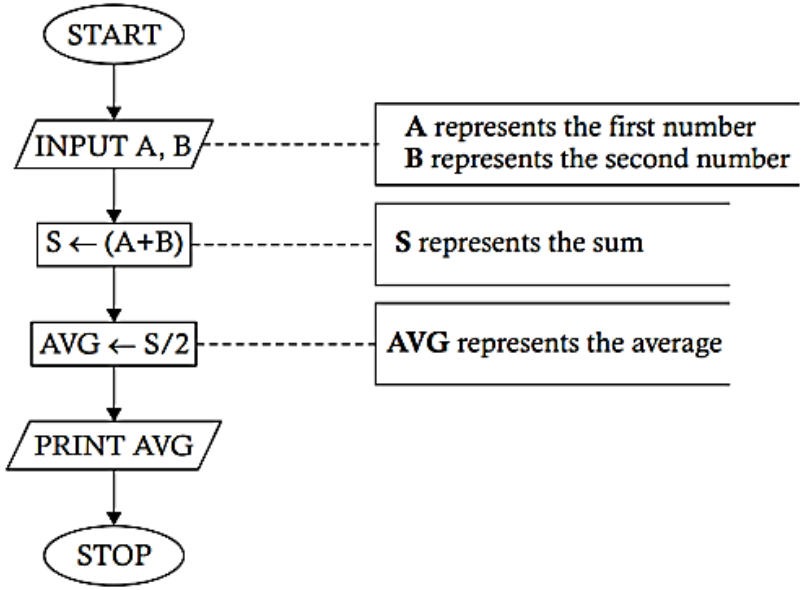



The following algorithm shows the desired procedure:

**Step 1.** INPUT TO A, B
**Step 2.** S ← A+B
(Store the sum of the values in A and B in S)
**Step 3.** PRINT S
(Show the sum obtained in Step 2)
**Step 4.** STOP

A sequence structure shows simple input, output, and process operations.

**Problem 1.2.** *Construct a flowchart to show the procedure for obtaining the average of two given numbers.*




**Task Analysis.** From the concept of determining the average of two given numbers, we know that the given numbers must be added together to obtain the sum first; the sum is then divided by 2 to obtain the average. The flowchart for Problem 1.2 illustrates this idea.

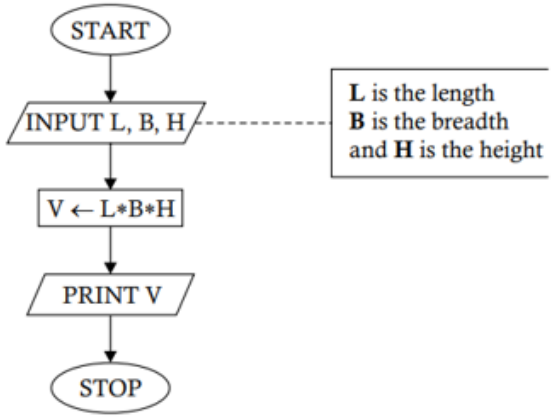
The algorithm corresponding to Problem 1.2 is shown below:

**Step 1.** INPUT TO A, B
**Step 2.** S ← A + B
(Store the sum of the values in A and B and store in S)
**Step 3.** AVG ← S/2
(Compute the average)
**Step 4.** PRINT AVG (Show the average)
**Step 5.** STOP

**Problem 1.3.** *Construct a flowchart to show how to obtain the volume of a rectangular box.*

**Task Analysis.** We know that the formula to determine the volume of a rectangular box is Volume = Length × Breadth × Height. To determine the volume of a rectangular box, we need to know the length, breadth, and height of the box. When these values are multiplied together, the product represents the desired volume.



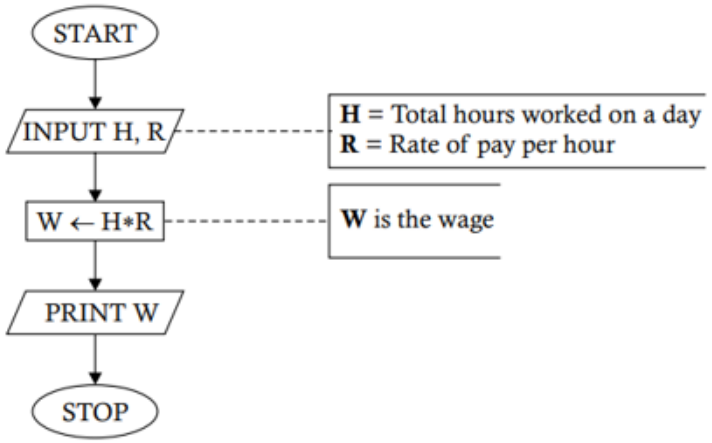



The algorithm for the solution of Problem 1.3 is given below:

**Step 1.** INPUT TO L, B, H
**Step 2.** COMPUTE V ← L*B*H
**Step 3.** PRINT V
**Step 4.** STOP

**Problem 1.4.** *Construct a flowchart to show how to obtain the daily wage of a worker on the basis of the hours worked during the day.*

**Task Analysis.** The daily wage depends on two factors: the hours worked and hourly rate of pay. When the hours worked is multiplied by the rate of pay, the product represents the wage of the worker.






The algorithm for the solution of Problem 1.4. is given below:

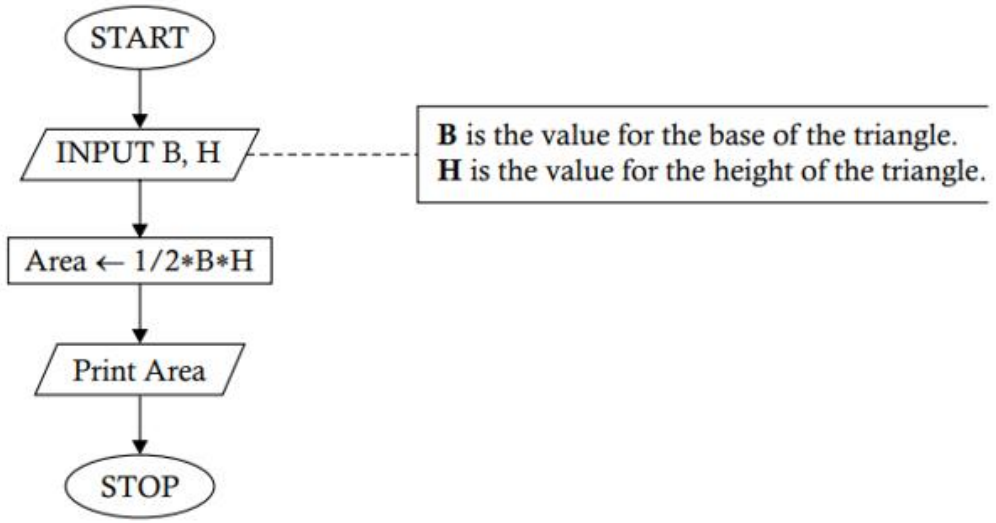
**Step 1.** INPUT TO H, R
**Step 2.** COMPUTE W ← H*R
(Store the product of the values in H and R in W)
**Step 3.** PRINT W
**Step 4.** STOP

**Problem 1.5.** *Construct a flowchart to show how to obtain the area of a triangle on the basis of the base and height.*

**Task Analysis.** We know that the formula to find out the area of a triangle is

$$\text{Area} = \frac{1}{2} \times \text{base} \times \text{height}$$

The inputs required to obtain the area of a triangle are its base and height. We can then put the values in the above formula to obtain the area.




The algorithm corresponding to the above procedure is given below:

**Step 1.** INPUT TO B, H
(B is for the base and H is for the height of the triangle)

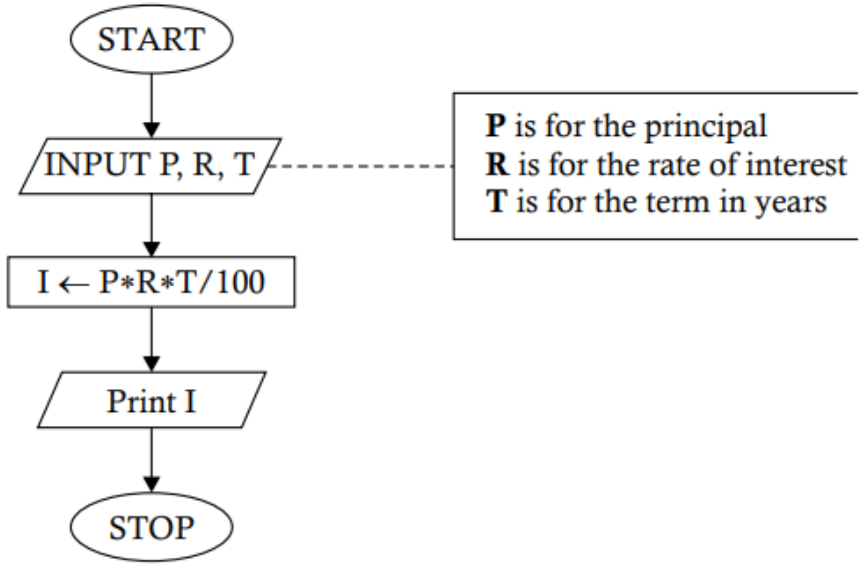
**Step 2.** COMPUTE AREA $\leftarrow \frac{1}{2}$*B*H

**Step 3.** PRINT AREA

**Step 4.** STOP

**Problem 1.6.** *Develop a flowchart to show the steps in finding the simple interest on a given amount at a given rate of interest.*

**Task Analysis.** We know that if P is the principal, R is the rate of interest, and T is the term in years, then the simple interest I is given by the formula $I = \frac{P*R*T}{100}$. To determine the simple interest on a given amount, we need the principal amount (P), the rate of interest (R), and the term in years (T). By putting the values in the formula above, we get the desired simple interest.



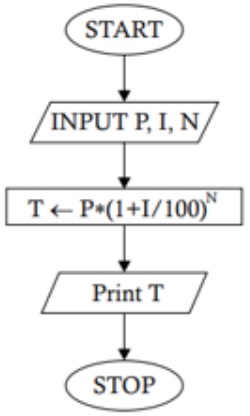

The algorithm corresponding to the above logic is given below:

**Step 1.** INPUT TO P, R, T
**Step 2.** COMPUTE I ← P*R*T / 100
**Step 3.** PRINT I
**Step 4.** STOP

**Problem 1.7.** *If P amount of money is invested for N years at an annual rate of interest I, the money grows to an amount T, where T is given by* $T = P(1 + I/100)^N$. *Draw a flowchart to show how T is determined.*

**Task Analysis.** The solution to this problem is very simple, and it is similar to the preceding one. The inputs required are the values for P, I, and N. The output T can then be obtained by putting the values in the formula.






The algorithm corresponding to Problem 1.7 is given below:

**Step 1.** INPUT TO P, I, N

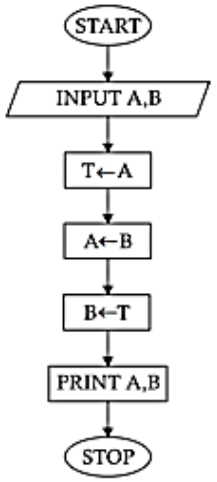
**Step 2.** COMPUTE $T \leftarrow P * \left(1 + \frac{I}{100}\right)^N$

**Step 3.** PRINT T

**Step 4.** STOP

**Problem 1.11.** *Draw a flowchart to show how to interchange the values of two variables.*

**Task Analysis.** The task of interchanging the values of two variables implies that the values contained by the variables are to be exchanged *i.e.*, the data value contained by the first variable should be contained by the second variable and that by the second variable should be contained by the first variable. If A and B are two variables, and if the values contained by them are 10 and 20 respectively, the problem is to make the contents of A and B, 20 and 10, respectively. This can be done simply with the help of a third variable used as an intermediate variable. The third variable holds the value of either A or B,

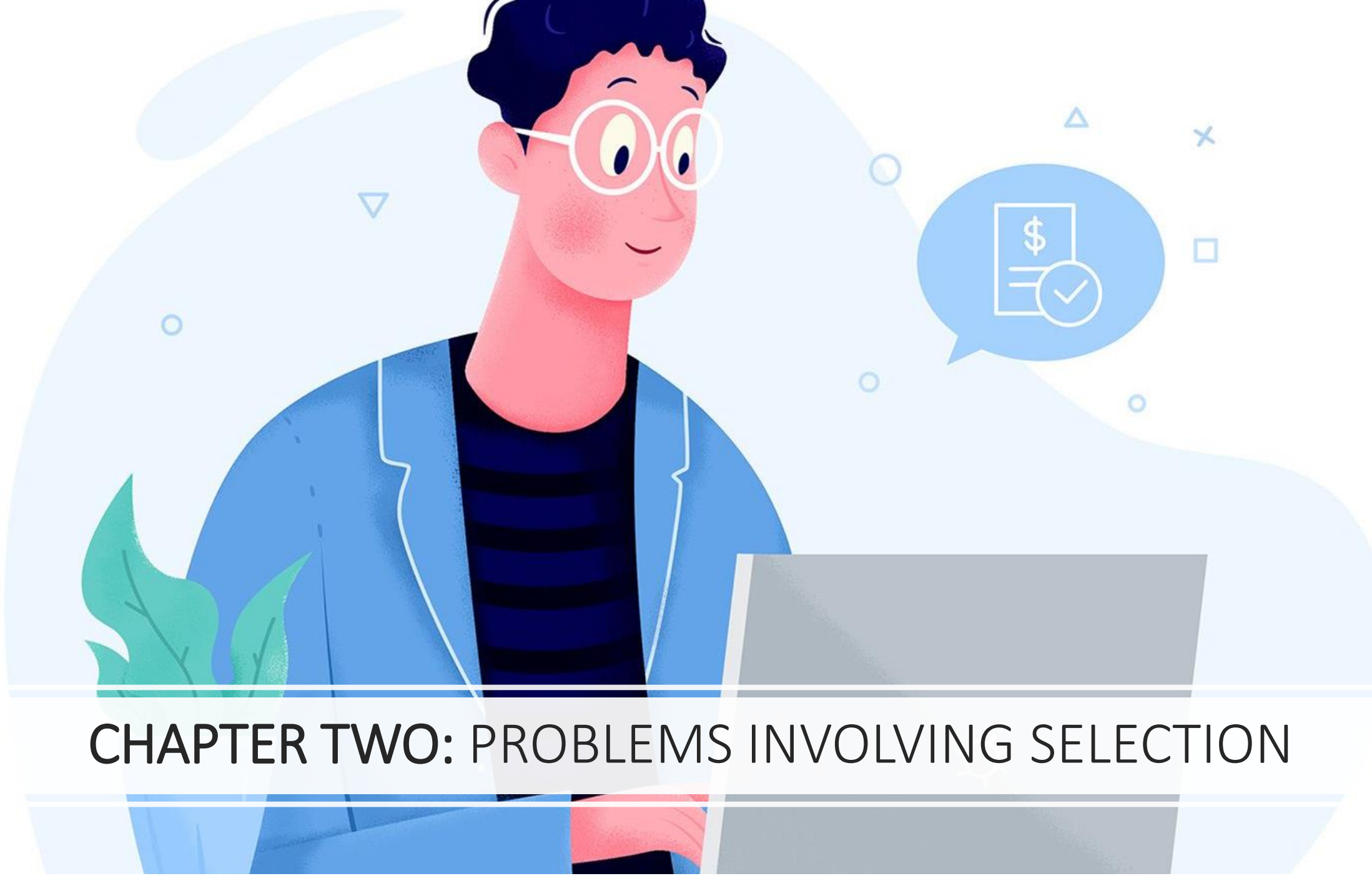





so that if the value of one variable is assigned to the other, the assignee's value is not lost forever but is available in the intermediate variable. Hence, it can then be assigned to the other variable.

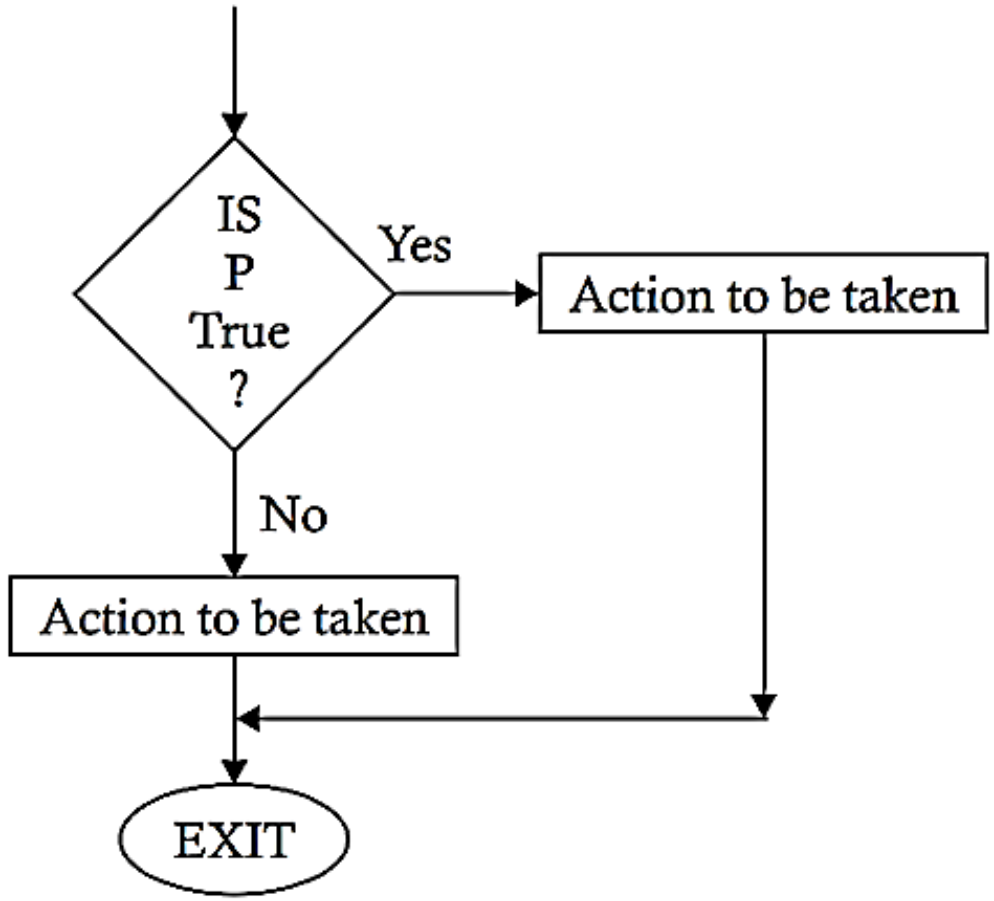
The algorithm of the problem is as follows:

**Step 1.** ACCEPT A,B
**Step 2.** T ← A (Assign value in A to T)
**Step 3.** A ← B (Assign value in B to A)
**Step 4.** B ← T (Assign value in T to B)
**Step 5.** PRINT A,B
**Step 6.** STOP



# CHAPTER TWO: PROBLEMS INVOLVING SELECTION

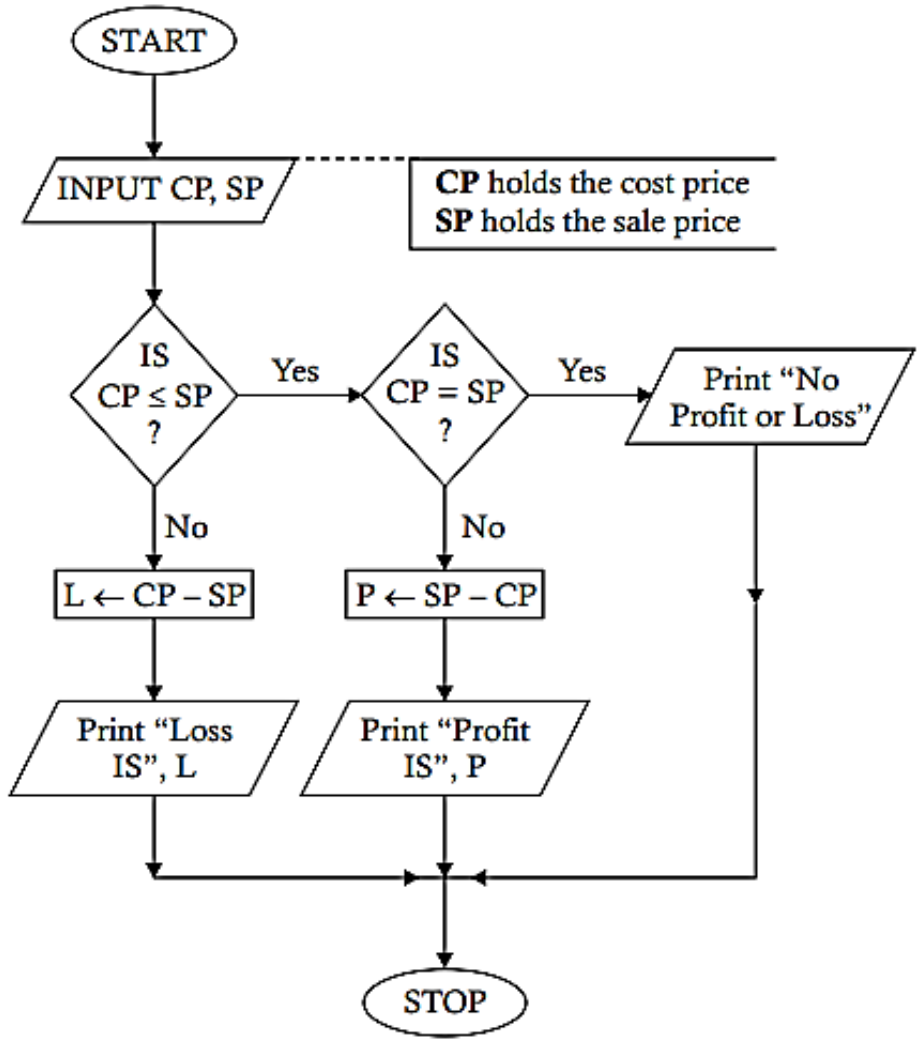
This chapter deals with problems involving decision-making. This process of decision-making is implemented through a logic structure called *selection*. Here a *predicate*, also called a *condition*, is tested to see if it is true or false. If it is true, a course of action is specified for it; if it is found to be false, an alternative course of action is expressed. We can express this process using flowchart notation.



# PROBLEMS

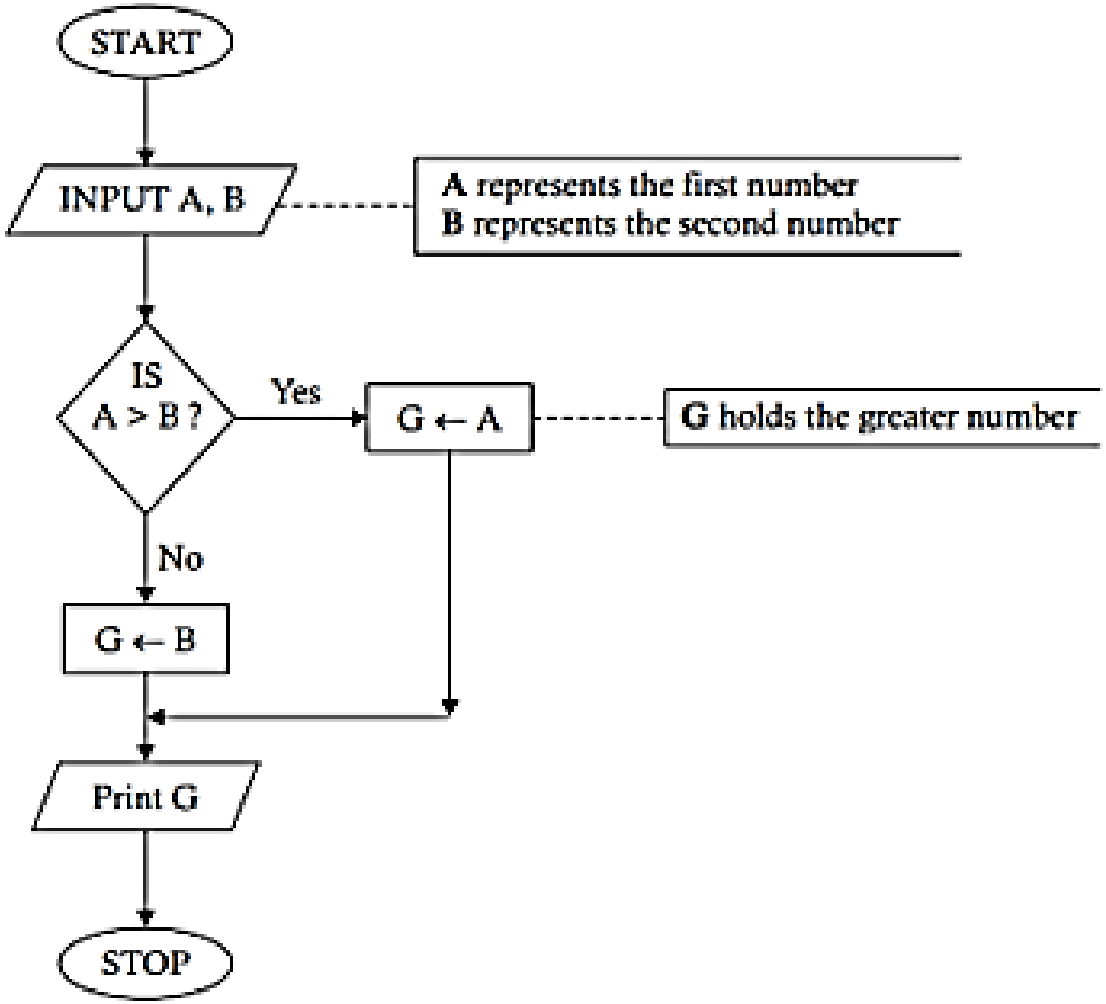
**Problem 2.1.** *Develop a flowchart to show how the profit or loss for a sale can be obtained.*

**Task Analysis.** The profit or loss for a sale can be obtained if the cost price and sale price are known. However, there is a need to make a decision here. If the cost price is more than the sale price, then it indicates a loss in the process; otherwise, there will be either zero profit (no profit or a loss) or some profit.



**Problem 2.2.** *Construct a procedure to show how to determine the greater of two given numbers.*

**Task Analysis.** We must determine the larger of two numbers. The task is to compare the given numbers to find the greater of them.

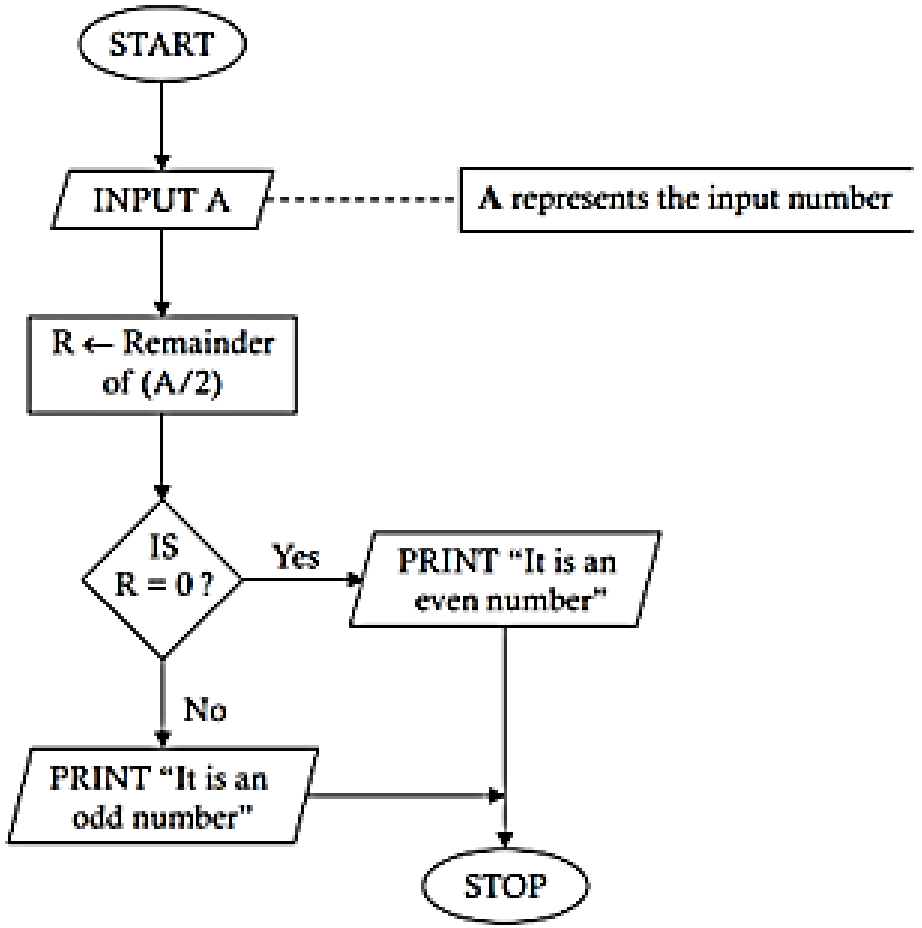



The algorithm corresponding to Problem 2.2 is given below:

**Step 1.** INPUT TO A, B
**Step 2.** IF A > B
THEN G ← A
ELSE
G ← B
END-IF
**Step 3.** PRINT G
**Step 4.** STOP

**Problem 2.3.** *Construct a flowchart to determine whether a given numb is even or odd.*

**Task Analysis.** We know that a number is an even number if it is complete divisible by 2. This means that if we perform integer division upon the giv number, then the remainder of the division will be zero. To construct tl flowchart, we accept a number as input, obtain the remainder of the integ division by taking it as the divisor, and then check whether the remainder zero. If it is zero, then our conclusion will be that the number is an even nur ber; otherwise, it will be an odd number.

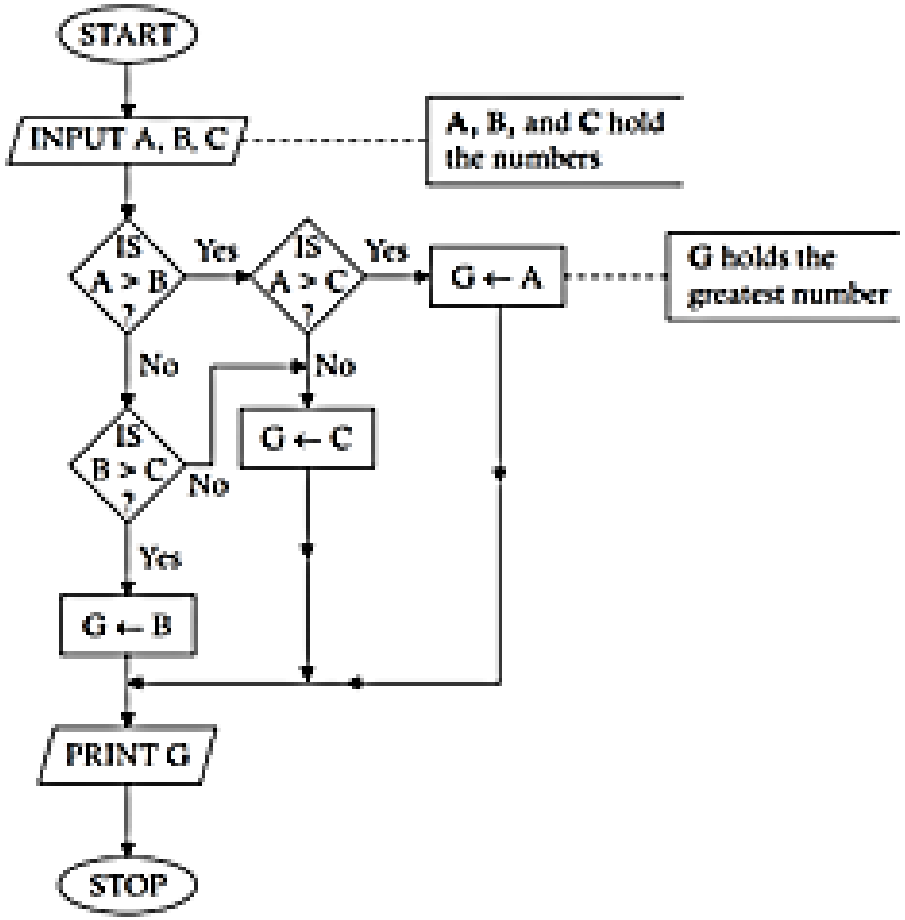



The algorithm corresponding to Problem 2.3 is shown below:

**Step 1.** INPUT TO A
**Step 2.** COMPUTE R ← Remainder of (A/2)
**Step 3.** IF R = 0
THEN PRINT "It is an even number."
ELSE
PRINT "It is an odd number."
END-IF
**Step 4.** STOP

**Problem 2.10.** *Construct a flowchart to show how the greatest of the three given numbers can be obtained.*

**Task Analysis.** This problem is similar to the problem for finding the greater of two given numbers. The only difference is that two successive comparisons are needed because three numbers cannot be compared at a time.




The following algorithm shows the procedure to follow for Problem 2.10:

```
Step 1.  INPUT TO A, B, C
         (Accept three numbers for A, B, and C)
Step 2.  IF A > B
            THEN IF A > C
                    THEN G ← A
                       (G holds the desired number)
                 ELSE
                       G ← C
                 END-IF
         ELSE
            IF B > C
               THEN G ← B
            ELSE
                 G ← C
            END-IF
```



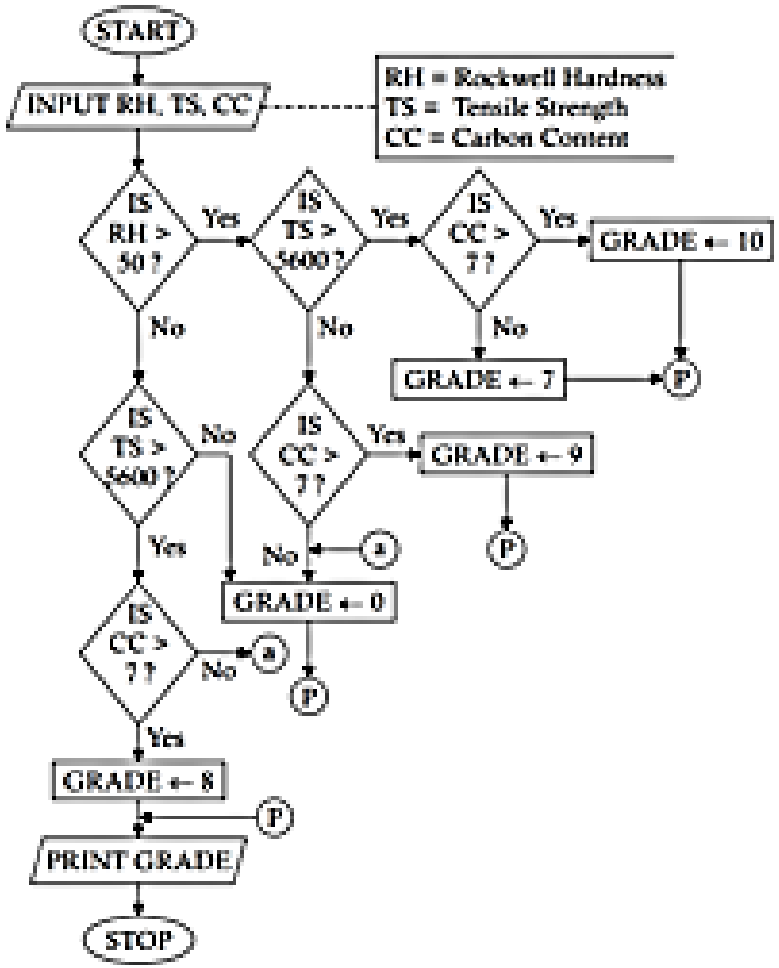
```
            END-IF
Step 3.  PRINT "THE GREATEST OF THE GIVEN NUMBERS IS", G
Step 4.  STOP
```

**Problem 2.9.** *A certain steel is graded according to the following conditions:*

(i) *Rockwell hardness > 50*
(ii) *Carbon content > 0.7*
(iii) *Tensile strength > 5600 kg/cm*$^2$

*The steel is graded as follows:*

a. *Grade 10, if all the conditions are satisfied*
b. *Grade 9, if conditions (i) and (ii) are satisfied*
c. *Grade 8, if conditions (ii) and (iii) are satisfied*
d. *Grade 7, if conditions (i) and (iii) are satisfied*
e. *Grade 0, otherwise*




**Task Analysis.** We must determine the grade of the steel on the basis of the values of three characteristics, namely, the Rockwell hardness, carbon content, and tensile strength. The values of these three features are the input.

The algorithm corresponding to Problem 2.9 is given below:

**Step 1.** INPUT TO RH, TS, CC
**Step 2.** IF RH > 50
THEN IF TS > 5600
THEN IF CC > 0.7
THEN GRADE ← 10
ELSE
GRADE ← 7
END-IF
ELSE
IF CC > 0.7
THEN GRADE ← 9
ELSE
GRADE ← 0
END-IF
END-IF
ELSE
IF TS > 5600
THEN IF CC > 0.7
THEN GRADE ← 8
ELSE
GRADE ← 0
END-IF
ELSE
GRADE ← 0
END-IF
**Step 3.** PRINT GRADE
**Step 4.** STOP

# CHAPTER THREE: PROBLEMS INVOLVING LOOPING

# Iteration

The term *iteration* means repetition. Sometimes, a procedure should be executed repeatedly. All procedures should be built so that they can be repeated as many times as needed. We should not develop procedures to execute only once.

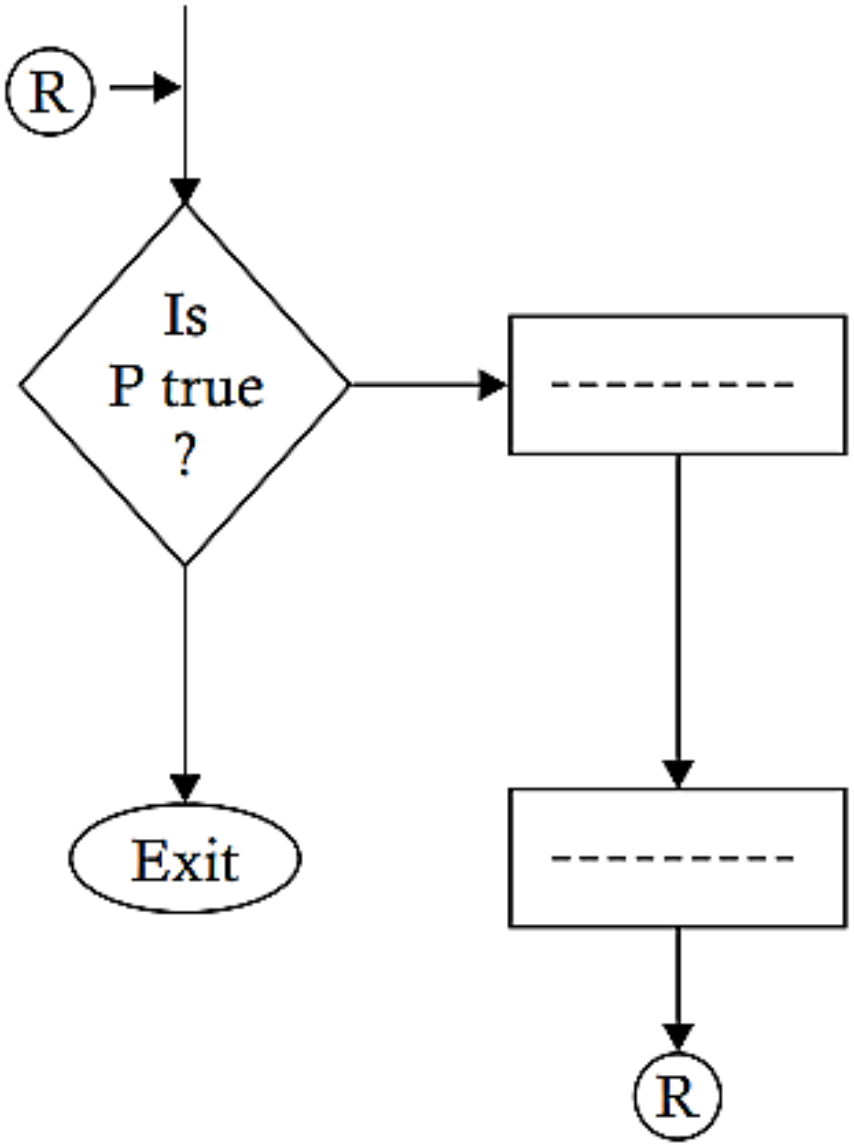
# Looping

An iterative logic structure is also known as a *loop*. *Looping* means repeating a set of operations to obtain a result repeatedly.

# Pre-test Iteration

post-test iteration. In case of a *pre-test iteration*, a predicate is tested to decide whether a set of operations is to be performed or not. If the condition implied by the predicate is true, then the desired operations are performed. If it is false, then the iteration is terminated. This is shown in the following diagram.

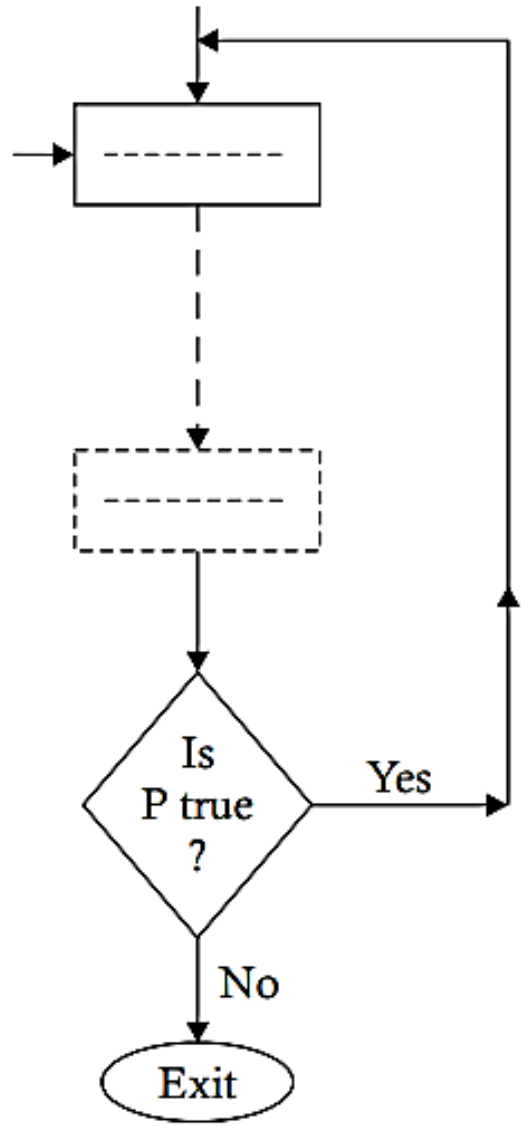
# Pre-test Iteration



# Post-test Iteration

For a post-test iteration, the predicate is tested after performing a set of operations once to decide whether to repeat the set of operations or to terminate the repetition. If the condition happens to be true, then the set of operations is repeated; otherwise, it is not repeated. The diagrammatic structure of this logic is as follows.

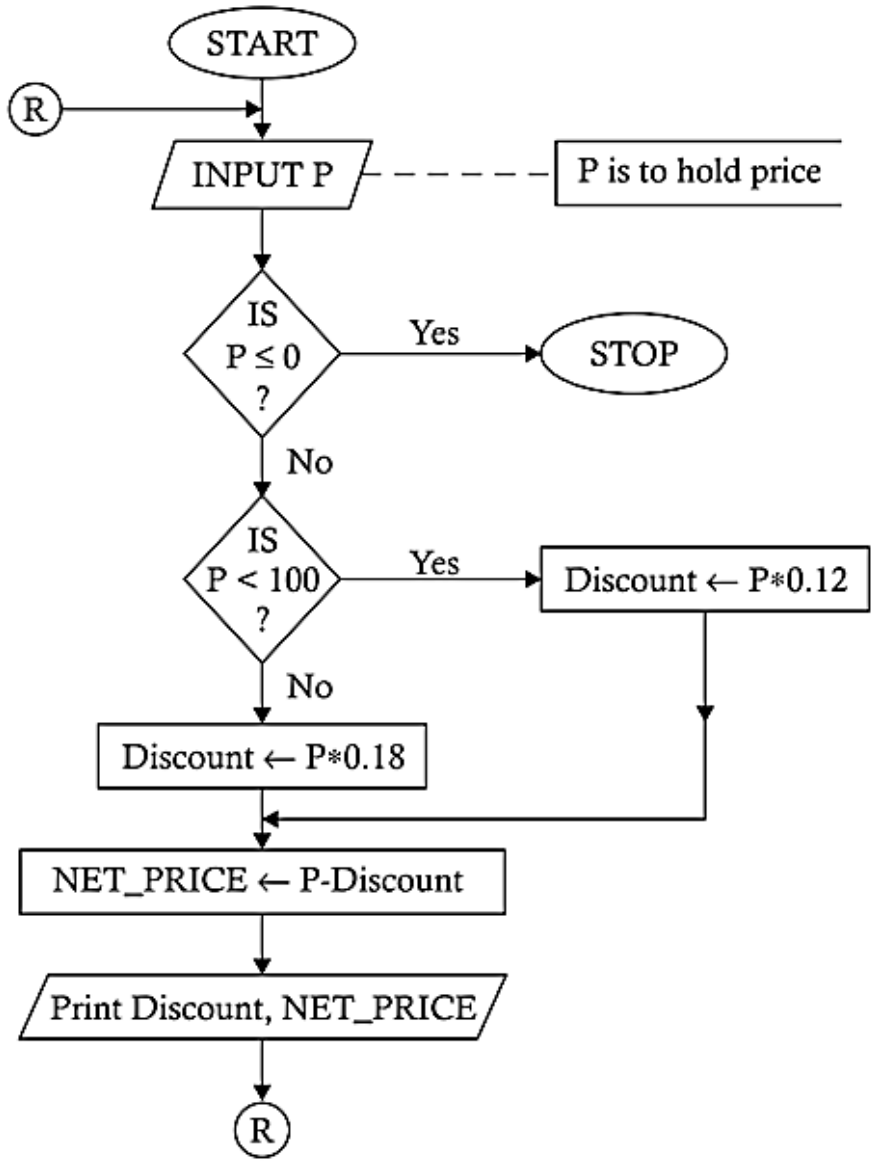
# Post-test Iteration



# Example

Let us assume that the repetitive task of calculating the discounts and net prices is terminated when we provide negative or zero as the price for the input

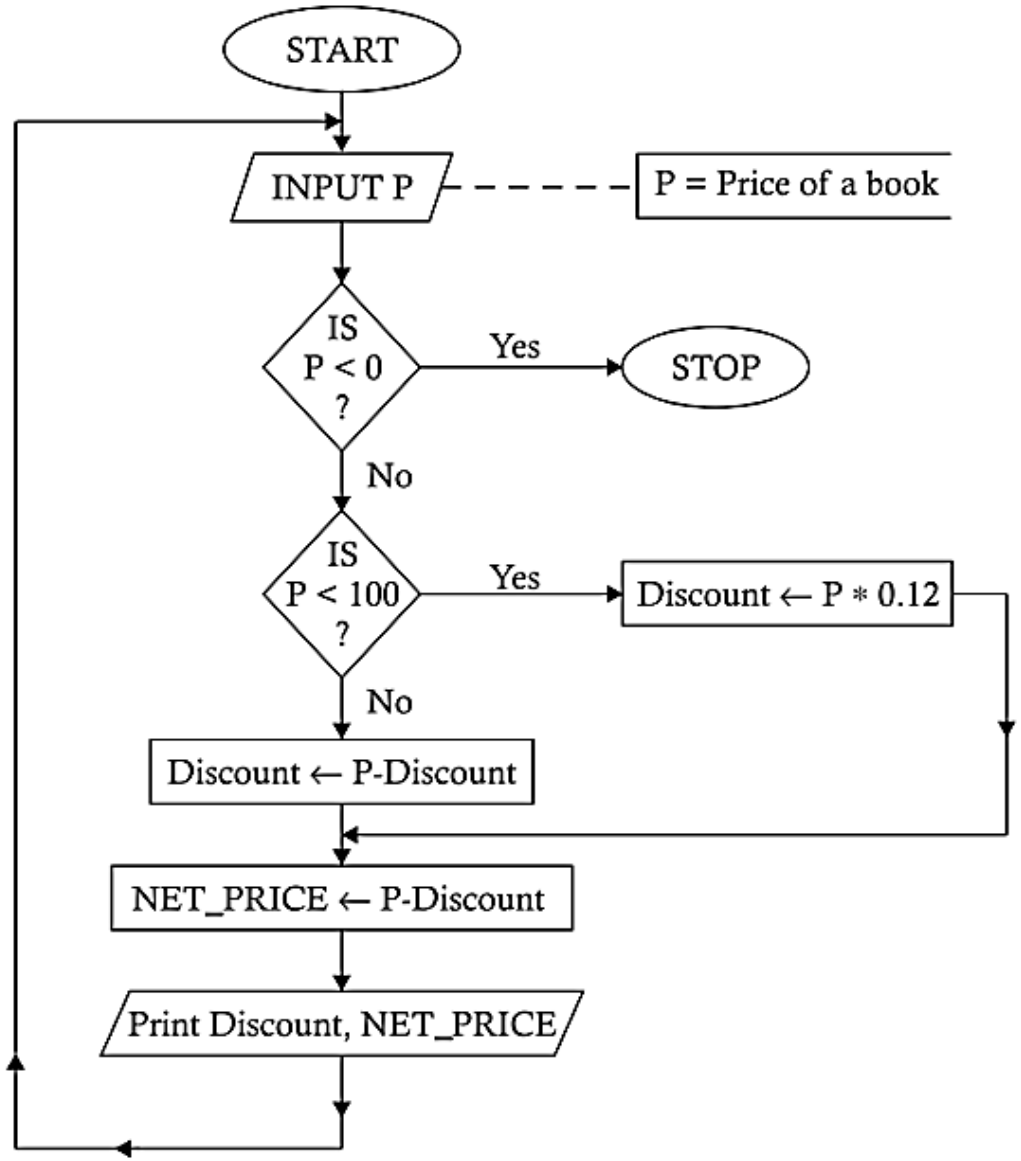
**Step 1.** REPEAT STEPS 2 THROUGH 6 (Start Loop)
**Step 2.** INPUT TO P
**Step 3.** IF P ≤ 0 THEN EXIT (Stop Repetition, *i.e.*, transfer the control to STOP).




**Step 4.** If P < 100
THEN COMPUTE D ← P*0.12
ELSE COMPUTE D ← P*0.18
END-IF
**Step 5.** COMPUTE NET_PRICE ← P – D
**Step 6.** PRINT D, NET_PRICE (End of loop)
**Step 7.** STOP

Note that the out-connector (R) shows the end point of the loop and the in-connector. (R)→ shows the start point of the loop. The operations start-

However, we prefer the first flowchart to the following one, because if the flowchart cannot be accommodated on a single page (or in a continuous structure on a single page), it would be difficult or impossible difficult to connect the start point and the end point.



# PROBLEMS

**Problem 3.1.** *The salesmen of a sales firm are given a commission on sales achieved, using the following rules:*

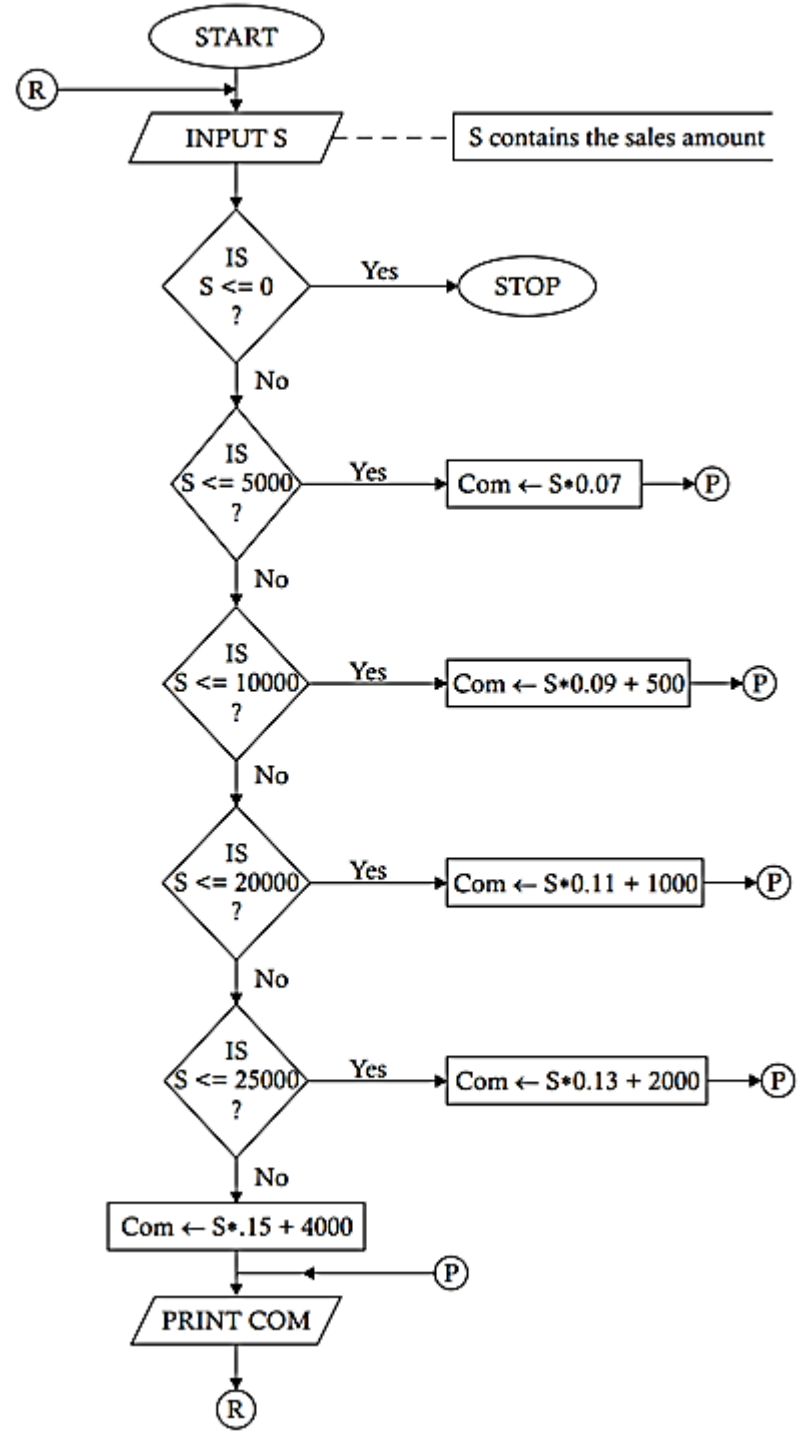
| *Sales* | *Rate of commission* |
|---|---|
| *<= 5,000* | *7% of sales* |
| *> 5,000 but <= 10,000* | *9% of sales + $500* |
| *> 10,000 but <= 20,000* | *11% of sales + $1,000* |
| *> 20,000 but <= 25,000* | *13% of sales + $2,000* |
| *> 25,000* | *15% of sales + $4,000* |

*Devise a procedure to calculate the commission of the salesmen.*

**Task Analysis.** The output required is the commission earned by a salesman. The only input required is the amount of the sale. A number of decision-making steps are involved, and the process is likely to be repeated a number of times. Let us assume that the process can be terminated when the amount of the sale is zero or negative. The procedure is illustrated in the following flowchart.

The algorithm corresponding to Problem 3.1 is given below:

```
Step 1.  REPEAT STEPS 2 THROUGH 5
Step 2.  INPUT TO S
         (Accept sales amount in S)
Step 3.  If S <= 0
           THEN EXIT
         END-IF
Step 4.  IF S <= 5000
           THEN COMPUTE COM ← S * .07
         ELSE
           IF S <= 10000
             THEN COMPUTE COM ← S * .09 + 500
           ELSE
             IF S <= 20000
               THEN COMPUTE COM ← S * 0.11 + 1000
             ELSE
               IF S <= 25000
                 THEN COMPUTE COM ← S * 0.13 + 2000
               ELSE
                 COMPUTE COM ← S * 0.15 + 4000
               END-IF
             END-IF
           END-IF
         END-IF
Step 5.  PRINT COM
Step 6.  STOP
```



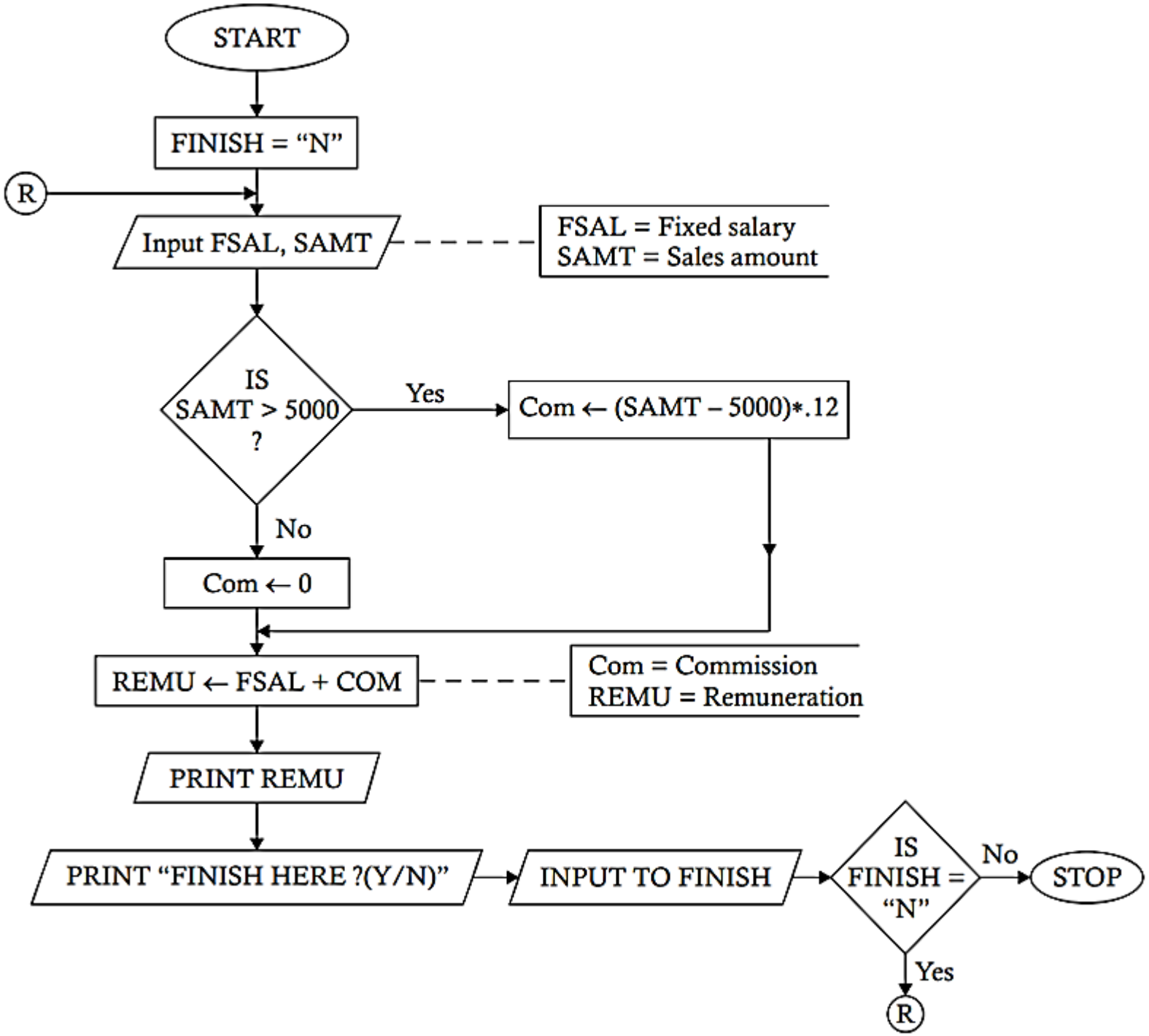
**Problem 3.2.** *A sales organization offers a fixed salary and a percentage of sales as a commission to determine the monthly remuneration of an employee under the following conditions.*

*If the sales amount of an employee exceeds $5,000, then the commission is 12% of the sales that exceed $5,000; otherwise, it is nil. Draw a flowchart to show how the remuneration of an employee is decided.*

**Task Analysis.** The remuneration of an employee consists of two parts: a fixed salary part and a commission part that depends on the sales amount. We use the fixed salary part and the sales amount as input to determine the commission and hence, the remuneration.

The algorithm corresponding to Problem 3.2 is given below:

**Step 1.** FINISH ← "N"
**Step 2.** REPEAT STEPS 3 THROUGH 9 WHILE FINISH = "N"
**Step 3.** INPUT TO FSAL, SAMT
**Step 4.** IF SAMT > 5000
THEN COMPUTE COM ← (SAMT – 5000) * .12
ELSE
COM ← 0
END-IF
**Step 5.** COMPUTE REMU ← FSAL + COM
**Step 6.** PRINT "REMUNERATION IS", REMU
**Step 7.** PRINT "FINISH (Y/N)?"
**Step 8.** INPUT TO FINISH
**Step 9.** IF FINISH = "Y"
THEN EXIT
END-IF
**Step 10.** STOP


START
FINISH = "N"
R
Input FSAL, SAMT
FSAL = Fixed salary
SAMT = Sales amount
IS
SAMT > 5000
?
Yes
Com ← (SAMT – 5000)*.12
No
Com ← 0
REMU ← FSAL + COM
Com = Commission
REMU = Remuneration
PRINT REMU
PRINT "FINISH HERE ?(Y/N)"
INPUT TO FINISH
IS
FINISH =
"N"
No
STOP
Yes
R

**Problem 3.3.** *A labor contractor pays the workers at the end of each week according to the rules given below:*

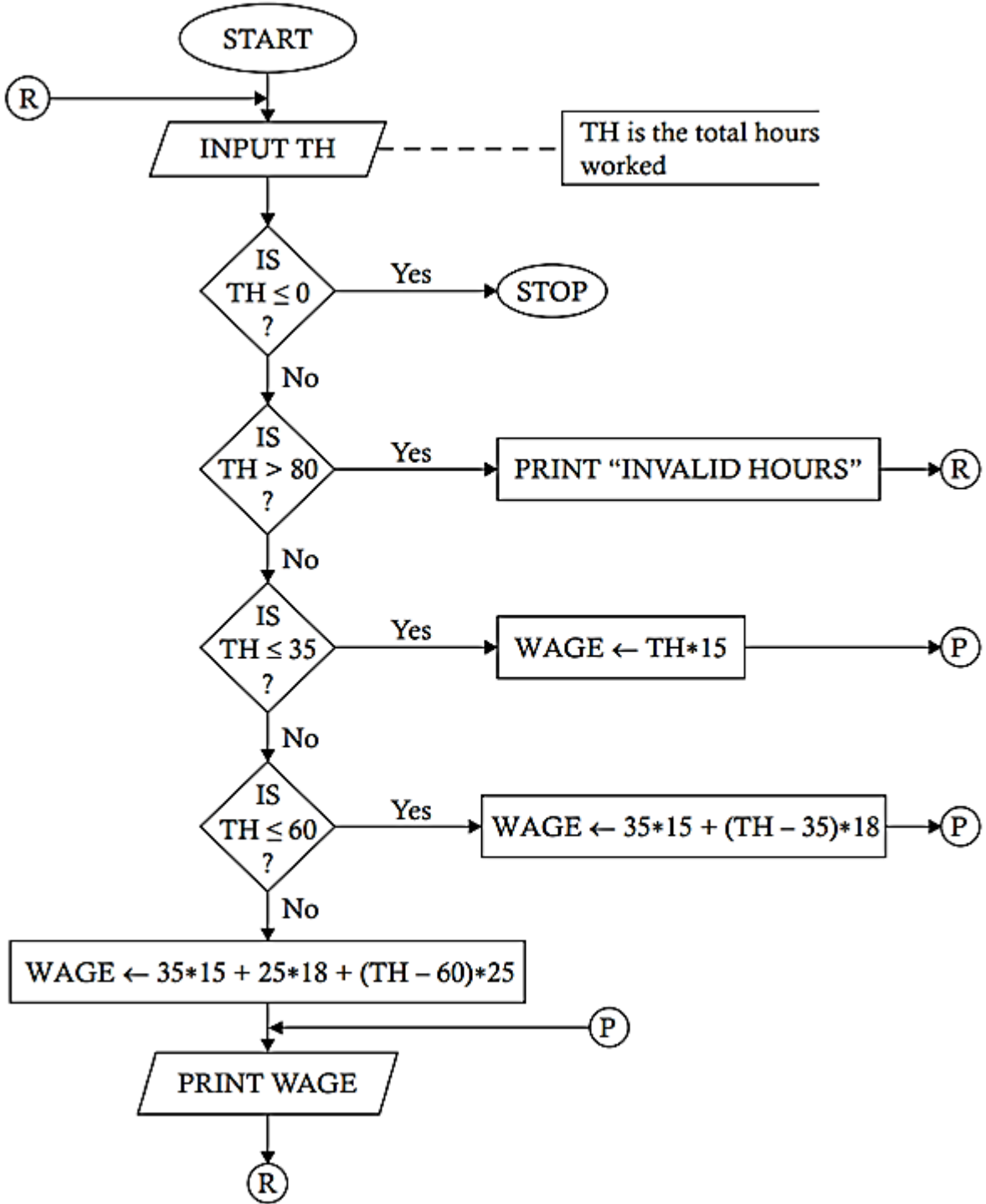
*For the first 35 hours of work, the rate of pay is $15 per hour; for the next 25 hours, the rate of pay is $18 per hour; for the rest, the rate of pay is $26 per hour. No worker is allowed to work for more than 80 hours in a week. Develop a flowchart to show how the wages of the workers can be calculated on the basis of valid inputs.*

**Task Analysis.** The input required is the total number of hours worked. The rates of payment depend on the different numbers of hours worked. The total hours worked may be considered valid if the number lies in the range of 0 through 80. Our procedure for evaluating the wage consists of the (*i*) validation of the hours worked, (*ii*) identifying the category to which the hours worked pertain, and then (*iii*) applying different rates to calculate the wage. The procedure is shown within a loop, and it is terminated when zero or a negative value is given as the input against hours worked.

```
Step 1.  REPEAT STEPS 2 THROUGH 6
Step 2.  INPUT TO TH
Step 3.  IF TH <= 0
           THEN EXIT
         END-IF

Step 4.  IF TH > 80
           THEN PRINT "INVALID HOURS"
           CONTINUE
         END-IF
Step 5.  IF TH <= 35
           THEN COMPUTE WAGE ← TH*15
         ELSE
           IF TH <= 60
             THEN COMPUTE WAGE ← 35*15 + (TH–35)*18

           ELSE
             COMPUTE WAGE ← 35*15 + 25*18 + (TH–60)*25
           END-IF
         END-IF
Step 6.  PRINT "WAGE IS", WAGE
Step 7.  STOP
```

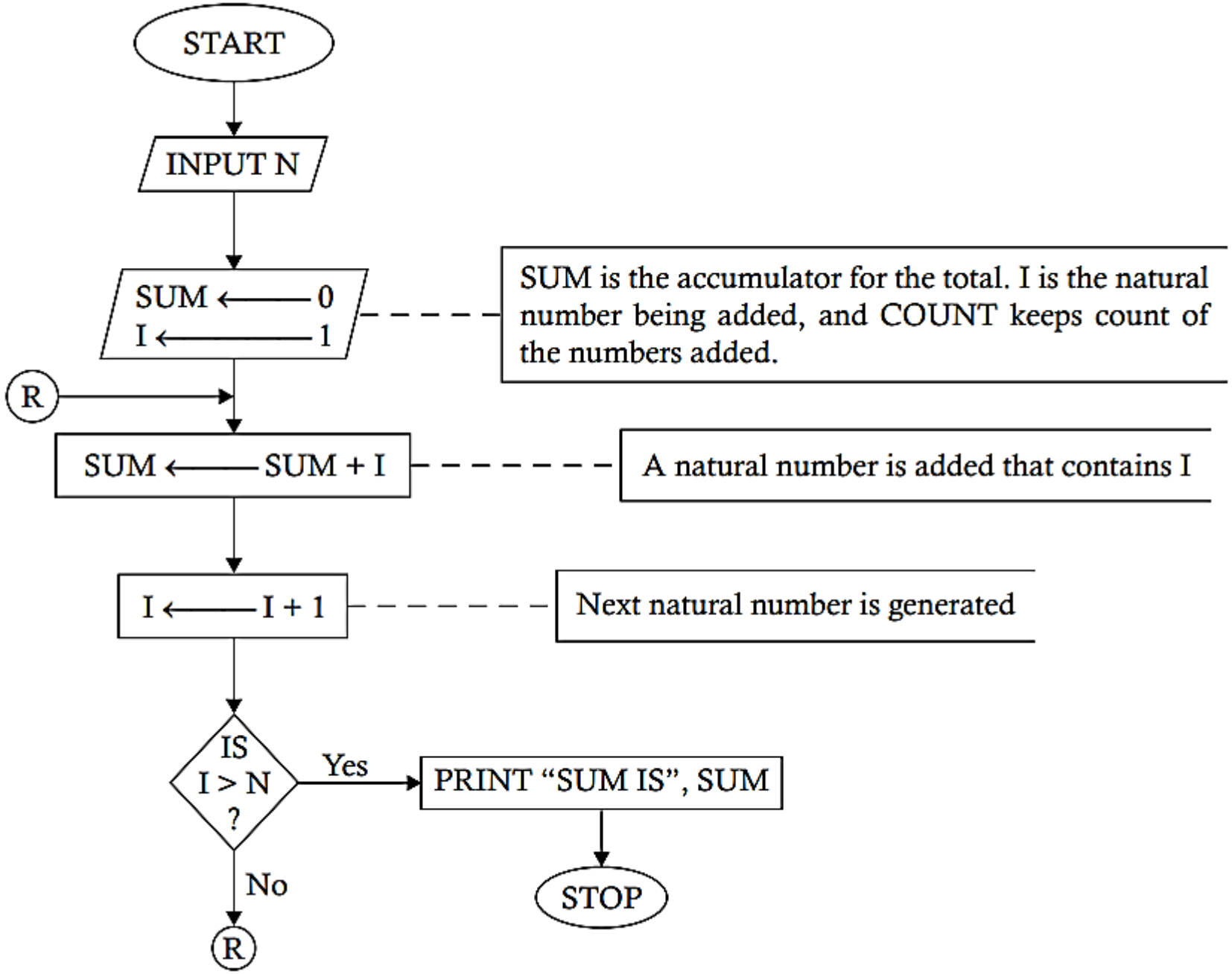

START
R
INPUT TH
TH is the total hours worked
IS TH ≤ 0 ?
Yes
STOP
No
IS TH > 80 ?
Yes
PRINT "INVALID HOURS"
R
No
IS TH ≤ 35 ?
Yes
WAGE ← TH*15
P
No
IS TH ≤ 60 ?
Yes
WAGE ← 35*15 + (TH – 35)*18
P
No
WAGE ← 35*15 + 25*18 + (TH – 60)*25
P
PRINT WAGE
R

**Problem 3.6.** *Devise a procedure to find the sum of first n natural numbers, where n is any given integer, without using a formula.*

**Task Analysis.** Natural numbers are those numbers that are obtained through sequential counting. The starting number here for the summation process is 1, the next number is 2 and so on, until we reach *n*—the number of natural numbers to be summed. The numbers to be added are known as inputs and

The algorithm corresponding to Problem 3.6 is shown below:

**Step 1.** INPUT "ENTER NUMBER OF TERMS TO ADD" TO N
**Step 2.** SUM ← 0 [INITIALIZATION]
**Step 3.** I ← 1 [INITIALIZATION]
**Step 4.** REPEAT STEPS 5 THROUGH 6 WHILE I <= N.
**Step 5.** COMPUTE SUM ← SUM + I
**Step 6.** COMPUTE I ← I + 1
**Step 7.** PRINT "THE SUM IS", SUM
**Step 8.** STOP


START
INPUT N
SUM ← 0
I ← 1
SUM is the accumulator for the total. I is the natural number being added, and COUNT keeps count of the numbers added.
R
SUM ← SUM + I
A natural number is added that contains I
I ← I + 1
Next natural number is generated
IS I > N ?
Yes
PRINT "SUM IS", SUM
STOP
No
R

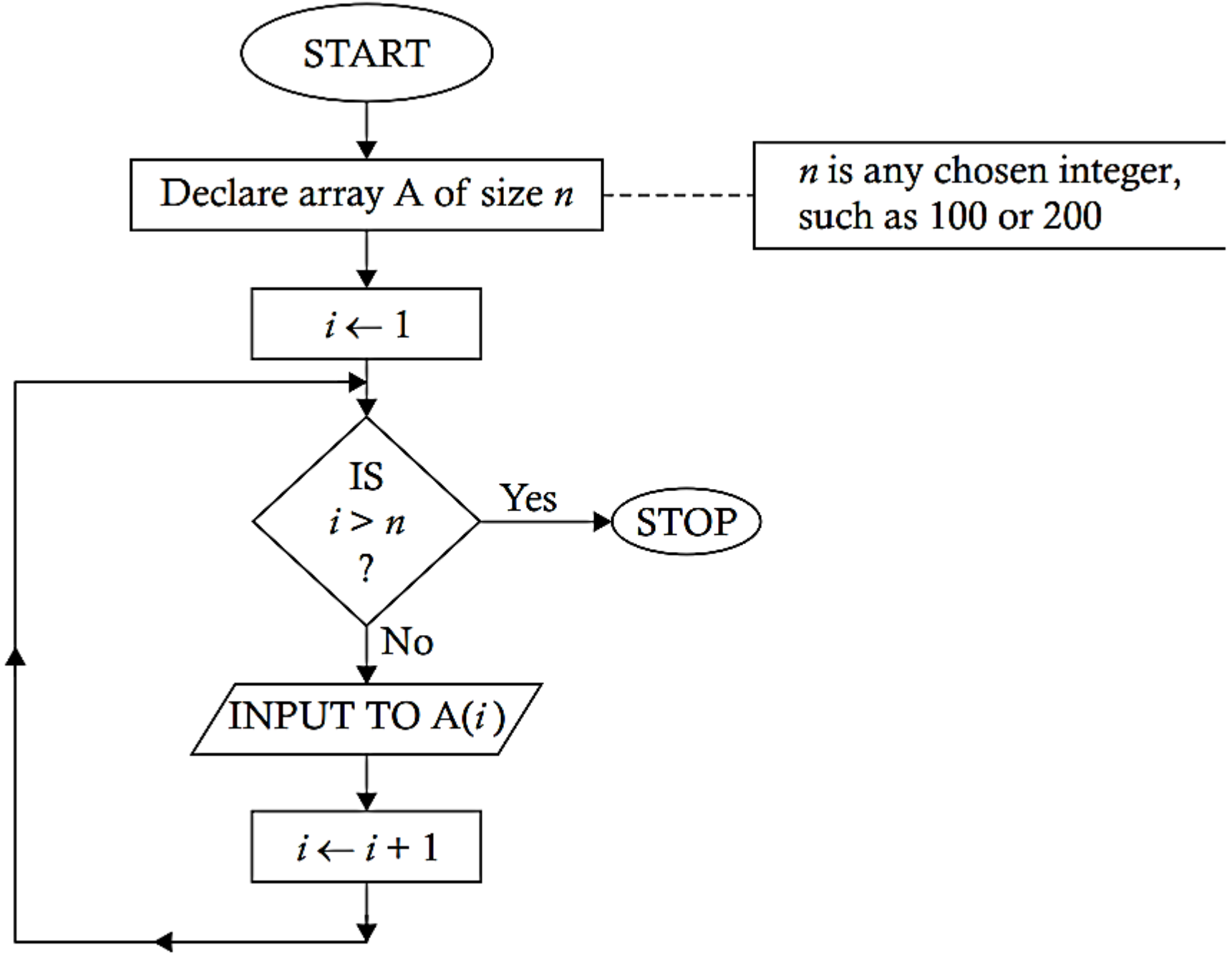

# CHAPTER FOUR: PROBLEMS INVOLVING ARRAYS

# Array

array is a finite collection of homogeneous data values usually stored in consecutive memory locations with a common name. The term *finite* implies that the number of data values of an array must be limited by its size. The term *homogeneous* means "having the same nature or characteristic." The term *usually* implies that arrays are almost always implemented by using contiguous locations of the computer's main memory in a linearly ordered fashion, but not always. The common name assigned to a set of adjacent memory locations to hold the data of a particular type is called the *name* of the array. The different data values of an array are mentioned by using the name of the array along with a subscript within brackets, such as A[1], A(1), and A[2],or in general, A[*i*], where *i* must be an integer. The value of *i* is the location. The subscript is also called an *index*. This is why an array element such as A[*i*] is also called an indexed or subscripted variable.

**Problem 4.1.** *The goal here is to show you how to construct an array. The following algorithm will clarify the steps:*

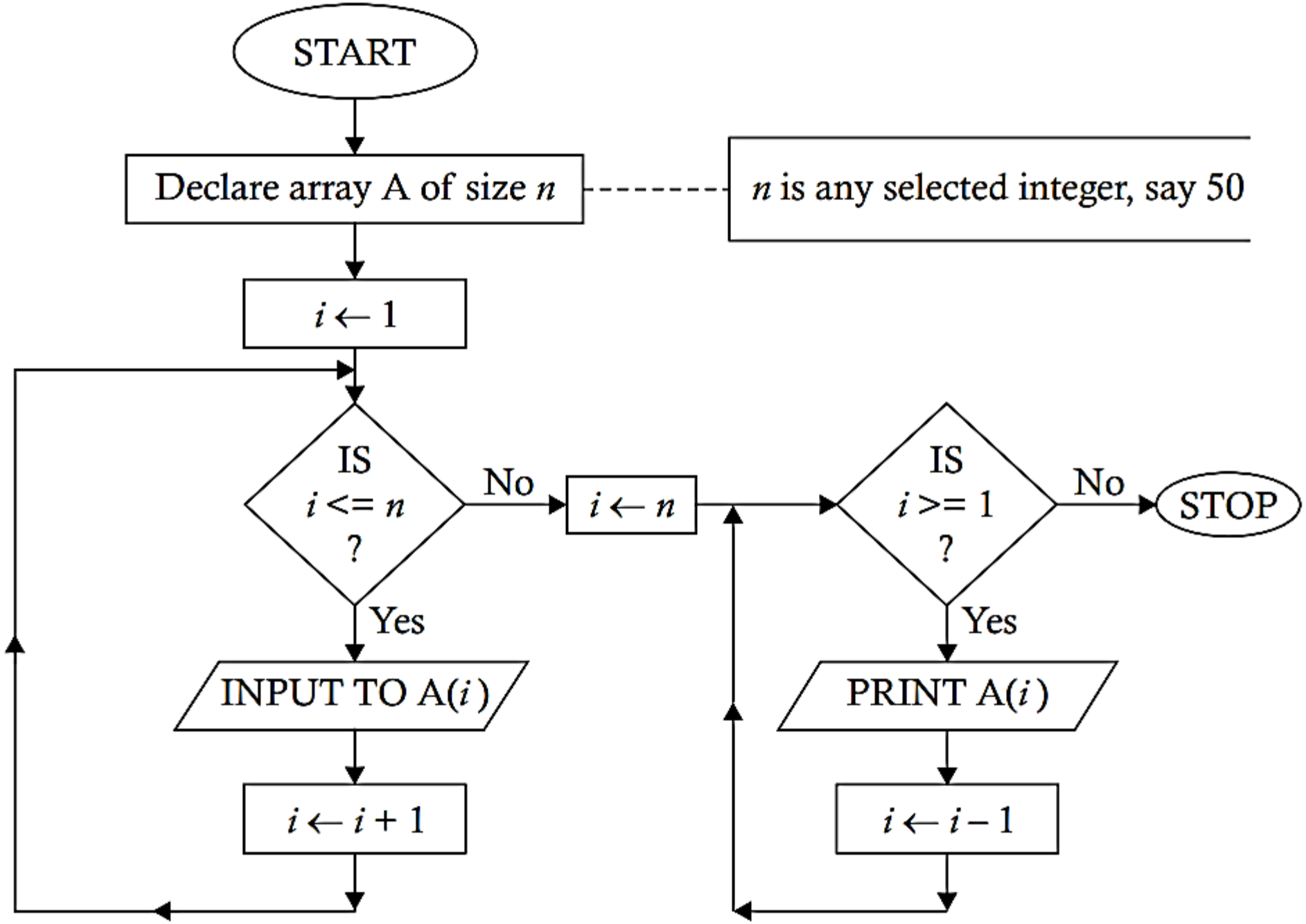
1. Decide the size of the array to be formed, say $n$.
2. Declare an array of size $n$ with some desired name, say A.
3. Initialize the variable or location that will be used as a subscript, say $i$, with a statement like $i \leftarrow 1$.
4. Repeat Steps 5 and 6 until $i > n$.
5. Accept the data value for the array element A($i$).
6. Increment the value of the subscript: $i \leftarrow i + 1$.
7. Stop.


START
Declare array A of size n
n is any chosen integer, such as 100 or 200
i ← 1
IS i > n ?
Yes
STOP
No
INPUT TO A(i)
i ← i + 1

# PROBLEMS

**Problem 4.2.** *Let us define the objective of our array creation. We wish to view the stored data values in the reverse sequence of inputs, i.e., we want to see the last input value first and the first input value last and others in that sequence. With this purpose in mind, we re-write the above algorithm as follows:*

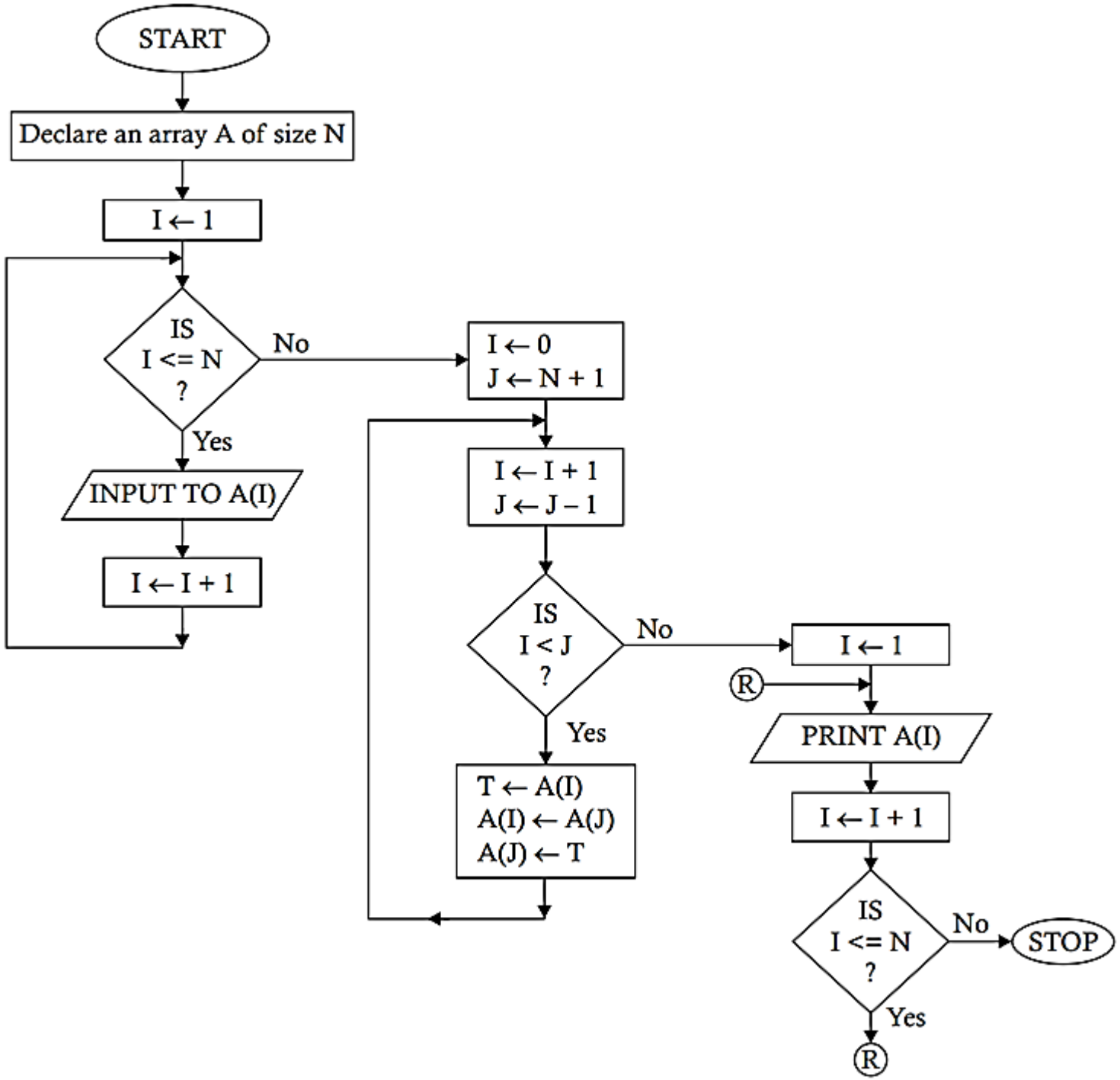
1. Decide the size of the array to be formed ($n$).
2. Declare an array of size $n$ with some desired name (A).
3. Initialize the variable or location that will be used as a subscript, say $i$, with a statement like $i \leftarrow 1$.
4. Repeat Steps 5 and 6 while $i <= n$.
5. Accept the data value for the array element A($i$).
6. Increment the value of the subscript: $i \leftarrow i + 1$.
7. Set $i \leftarrow n$.
8. Repeat Steps 9 and 10 while $i > = 1$
9. Display A($i$)
10. Set $i \leftarrow i - 1$.
11. Stop


START
Declare array A of size n
n is any selected integer, say 50
i ← 1
IS i <= n ?
No
Yes
i ← n
IS i >= 1 ?
No
STOP
Yes
INPUT TO A(i)
PRINT A(i)
i ← i + 1
i ← i – 1

**Problem 4.3.** *Construct a flowchart to show how to rearrange the elements in an array so that they appear in reverse order.*

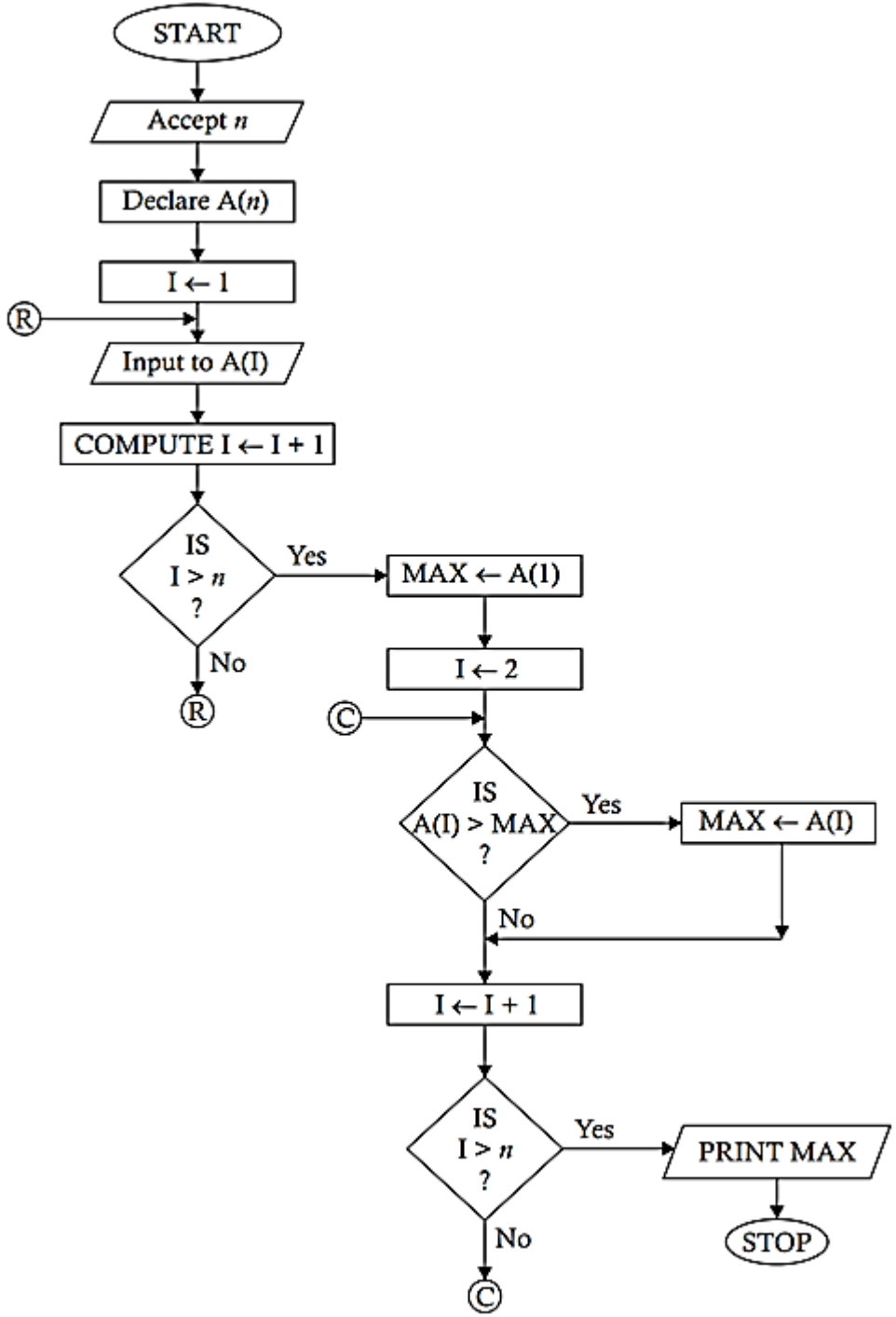
The algorithm corresponding to Problem 4.3 is given below:

**Step 1.** Declare the array A(1 ... N) of N elements to be reversed.
**Step 2.** Repeat Step 3 for I = 1 to N.
**Step 3.** Accept a data value at the $I^{th}$ location.
**Step 4.** Set I = 0, J = N + 1
**Step 5.** Repeat steps 6 through 7 until I >= J
**Step 6.** I ← I + 1, J = J – 1
**Step 7.** T ← A (I)
A(I) ← A(J)
A(J) ← T
**Step 8.** Repeat step 9 for I = 1 to N
**Step 9.** PRINT A(I)
**Step 10.** STOP


START
Declare an array A of size N
I ← 1
IS
I <= N
?
No
Yes
INPUT TO A(I)
I ← I + 1
I ← 0
J ← N + 1
I ← I + 1
J ← J – 1
IS
I < J
?
No
Yes
T ← A(I)
A(I) ← A(J)
A(J) ← T
I ← 1
R
PRINT A(I)
I ← I + 1
IS
I <= N
?
No
STOP
Yes
R

**Problem 4.4.** *Construct a flowchart to show how to determine the maximum number in a set of n numbers.*

**Step 1.** Accept the size of the set, $n$.
**Step 2.** Declare an array A(1 ... $n$) of $n$ elements where $n \geq 1$.
**Step 3.** Repeat step 4 for I = 1, 2, ..., $n$.
**Step 4.** Accept a number to A(I).
**Step 5.** [Set temporary maximum MAX to first array element] MAX ←A(1).
**Step 6.** Repeat step 7 for I = 2, 3, ..., $n$.
**Step 7.** If A(I) > MAX,
    Then MAX ← A(I)
    End-if
**Step 8.** Print MAX as the maximum for the array of $n$ elements.
**Step 9.** Stop.

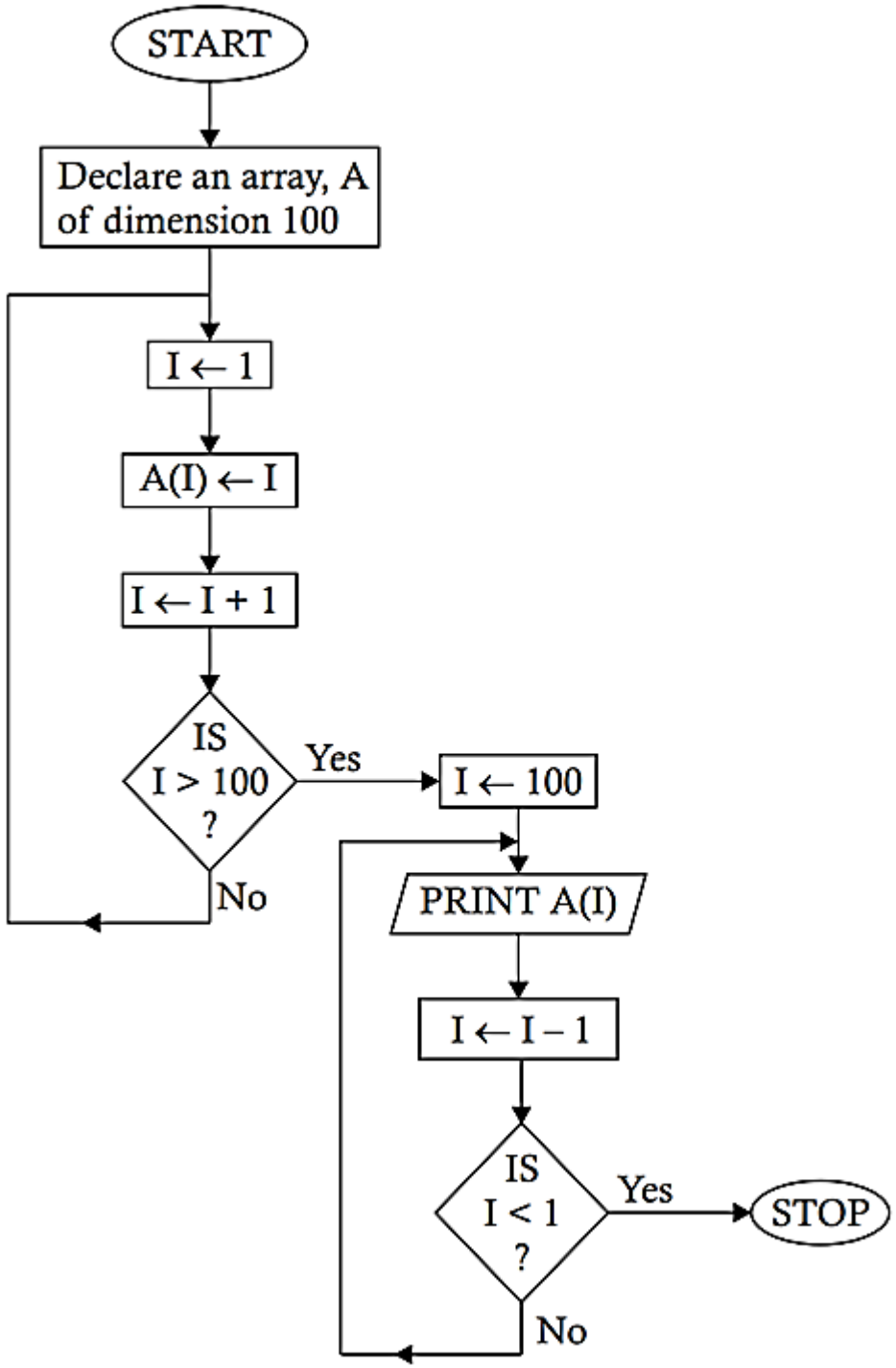

START
Accept n
Declare A(n)
I ← 1
R
Input to A(I)
COMPUTE I ← I + 1
IS I > n ?
Yes
MAX ← A(I)
No
R
I ← 2
C
IS A(I) > MAX ?
Yes
MAX ← A(I)
No
I ← I + 1
IS I > n ?
Yes
PRINT MAX
STOP
No
C

**Problem 4.5.** *Construct a flowchart to show how to store the first 100 natural numbers in an array and then show them in the reverse sequence.*

The algorithm corresponding to Problem 4.5. is given below:

**Step 1.** Declare an array of size 100 with any chosen name, say A.
**Step 2.** Initialize the variable that will be used as a subscript, say I, with a statement like I ← 1
**Step 3.** REPEAT steps 5 and 6 UNTIL I > 100.
**Step 4.** A(I) ← I
**Step 5.** I ← I + 1
**Step 6.** Initialize I again with the starting print location, I ← 100
**Step 7.** REPEAT STEPS 9 & 10 UNTIL I < 1.

**Step 8.** PRINT A(I).
**Step 9.** I ← I – 1
**Step 10.** END.


START
Declare an array, A of dimension 100
I ← 1
A(I) ← I
I ← I + 1
IS I > 100 ?
Yes
No
I ← 100
PRINT A(I)
I ← I – 1
IS I < 1 ?
Yes
No
STOP

# Thank You So Much!