Chapter 8

Problem Solving and Program Development

Introduction

In computing, it is required to have a program that is able to produce the desired solution of a problem using the computer. To develop a program, it is necessary to design (plan) the algorithm that describes the steps of solution. Without this design, the programmer will write a bad program or fail entirely to write a program.

In this chapter, the different approaches of designing the algorithms are described. Writing programs based on an algorithm using a high-level language (HLL) will be introduced in the next chapter.

Date of lecture / / 20

Intended Learning Outcomes of the Chapter

On completing this chapter, the student will be able to:

- Solve simple problems analytically.
- Solve simple problems using an algorithm and pseudo code.
- Write a formal statement from a given problem description.
- Apply the tools for algorithm development: sequence, selection, and repetition.
- Verify an algorithm using desk checking.
- Incorporate pretty printing into an algorithm.

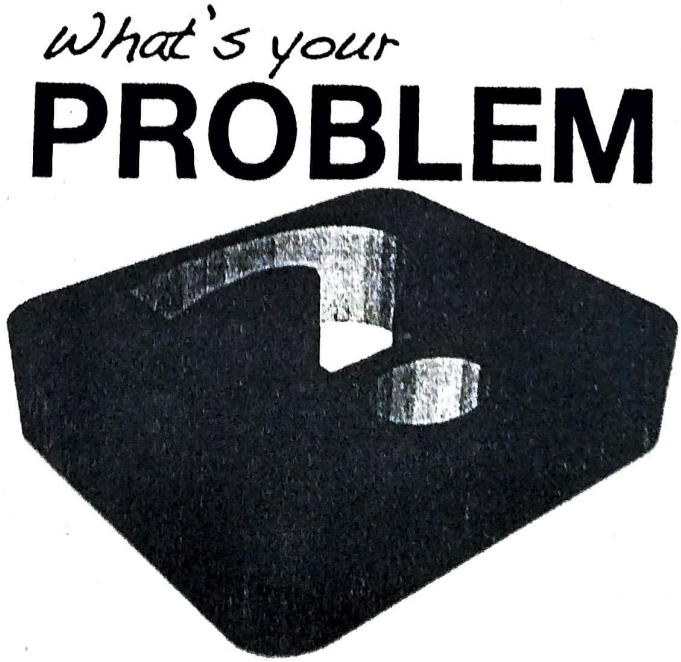
Note:

The problem

Most people are faced with a numerous of situations that demand their attention and force them to think of appropriate actions to take. Some of these scenarios might be "What should I wear today?" or "Whom do I have to call and whom should I try to avoid?" or "An accident is ahead. What deviation should I take?

Each of the above situations can be problematic. That is, you need problem-solving skills to resolve them. Most such problems are solved without much serious thought. Although some problems are indeed difficult and illogical, many problems are solvable. You may not know the best route to take to work but surely there is some route that will get you there. You may not know the solution to your physics homework but you know there is an answer. These problems are solvable but you need a systematic approach to find a reasonably good answer in a suitable amount of time. In this chapter, we will concentrate on simple kinds of problems, those that appear to have a normal solution.

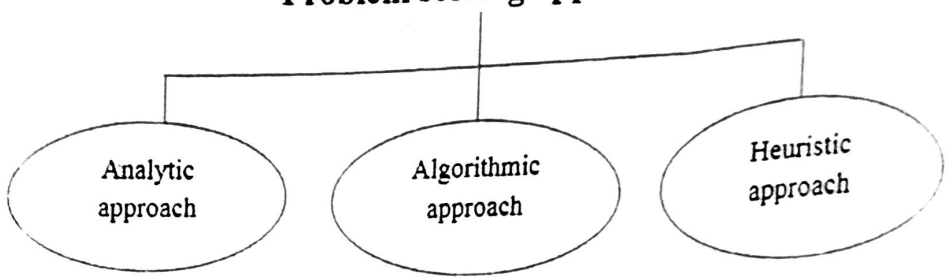
Date of lecture / / 20



Problem solving approaches

There are three well-known methods of problem solving. The first is the analytic method used in mathematics and physics. The second is the algorithmic method used by programmers for simple kinds of problems, those that appear to have a normal solution. The third is the heuristic approach which is used in the case of complex problems. Both of the second and third methods are necessary steps for computer program development.

Date of lecture / / 20



Note:

These methods have four common solution steps: problem, reasoning, solution, and test.

1. **Problem:** presents the situation that requires a solution.
2. **Reasoning:** implies a true comprehension of the problem (a brief idea of it is not enough).
3. **Solution:** means the process we develop to solve the problem, which may include insight and practical skills.
4. **Test:** means the checking process we use to confirm that the solution is correct.

Normally, algorithmic and heuristic approaches are applied in order to develop a computer program for the problem under consideration. In these cases, the program is written (generally in a high-level language) after designing or planning and testing the algorithm (steps 3 and 4). Because this course concentrates on simple kinds of problems, the analytic and algorithmic approaches will be considered in detail.

1. The Analytic Approach

The analytic approach is what we use to solve algebra, chemistry, and physics problems. After reading over a word or story problem several times, we try to solve it. First we isolate the given quantities, then we determine what is to be solved, then we apply formulas, then we may do mathematical operations such as factoring and dividing, and finally we get an answer. If we are hard-working, we check the answer, often by substituting it back into the original formula. Let's illustrate this method with the following simple example.

Date of lecture / / 20

➢ Example 1:

Problem:

Six identical computers are bought. The total cost is $14,627.50, which includes $340.00 for shipping and a sales tax of $724.00. Find the cost of an individual computer, excluding shipping and taxes.

Reasoning:

- The input variables:
  Number of computers, Total cost, shipping cost and the sales tax.
- The output:
  The cost of an individual computer, excluding shipping and taxes.

Processes:

To find the cost of one computer:

- Find the total net cost of the six computers = Total cost - shipping - tax.
- The cost of an individual computer, excluding shipping and taxes = total net cost of the 6 computers/6.

Note:

Solution:

Let x be the cost of one computer, and then the cost of all six computers is 6x. The net cost of the six computers is the total cost less shipping and tax. In this case

Net cost of the 6 computers = total cost of the 6 computers - shipping – tax

Substituting:

Net cost of the 6 computers = \$14, 627 .30 - \$340.00 - \$724.00

= \$13, 563 .30

So

$6x = \$13, 563 .30$

$x = \$13, 563 .30/6 = \$2260.55$

Test:

A single computer costs \$2260.55. T0 check that the answer is right, substitute the value 2260.55 for X in the original equation. That is,

6X + shipping + tax = total cost

So

6(2260.55) + 340.00 + 724.00 = 14,627.30

14,627.30 = 14,627.30

Date of lecture / / 20

Note:

2. The Algorithmic Approach

In the algorithmic approach, a finite set of steps is used to solve the problem.

- **Definition of an algorithm:**
  The algorithm is a sequence of executable instructions with the following properties:
  - No ambiguity between the instructions of the algorithm.
  - No ambiguity about which instruction is to be executed next.
  - Execution of the algorithm ends after a finite number of steps.
- **Definition of an executable instruction:**
  An executable instruction is the instruction that can be carried out.
- **Phases of Algorithmic Problem Solving:**
  The phases of the algorithmic problem solving approach are four as follows:
  - **Problem:** the situation that requires a solution.
  - **Reasoning:** it a formal statement that summarizes the problem as understood. In this phase, the following 3 components should be determined:
    - The input or inputs.
    - The output or outputs.
    - The process or processes needed to get the output(s) from the input(s).
  - **Solution (Design the algorithm):** in this phase, one should describe the algorithm. Two methods can be used to describe the algorithm:
    - The flow chart.
    - The pseudocode.
  - **Test:** it is the hand verification or desk checking.

Date of lecture / / 20

Note: .....

In the solution stage, the algorithm may contain one or more of the following three control structures:

- **Sequence:** this means that step by step procedure.
- **Selection:** this means that choices can be made where necessary.
- **Repetition:** this means that all or part of the process can be repeated.

The following sections will explain these types of structures.

Date of lecture
/ / 20

A. Sequence structure

In this structure, the algorithm is based on the concept of ordering the instructions. In other words, when we write an algorithm we number the steps. This is both for reference and to stress that there is a specific order to the algorithm that becomes significant when the algorithm is put to use. In this type of algorithms, it is assumed that **step *n* cannot be started until step *n* – 1 is completed**. The following example is an application for the sequence structure.

Date of lecture / / 20

➢ Example 2:

Develop an algorithm that calculates and prints the area of a trapezoid with two bases of lengths base1 and base2 and height h.

Problem:

Write an algorithm to calculate and print the area of a trapezoid with two bases of lengths base1 and base2 and height h

Reasoning:

In this step we describe the following:

- Formal statement: trapezoid problem
- Input: base1, base2, height
- Output: area
- Process: area = (base1 + base2) x height / 2

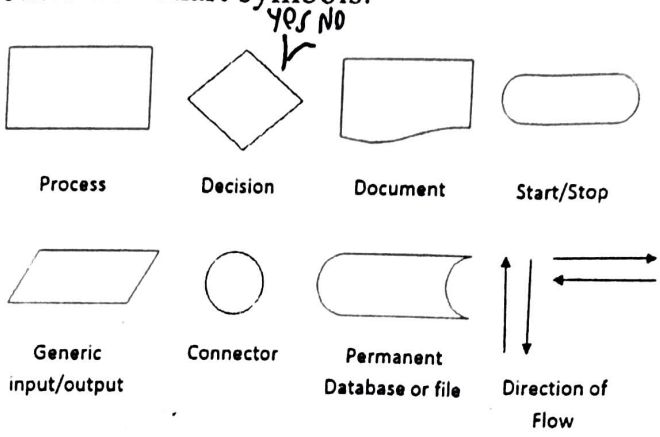
Note:

Solution (Design the algorithm):

In this step, the solution can be put in the form of a flow chart or an algorithm written in a pseudo code language.

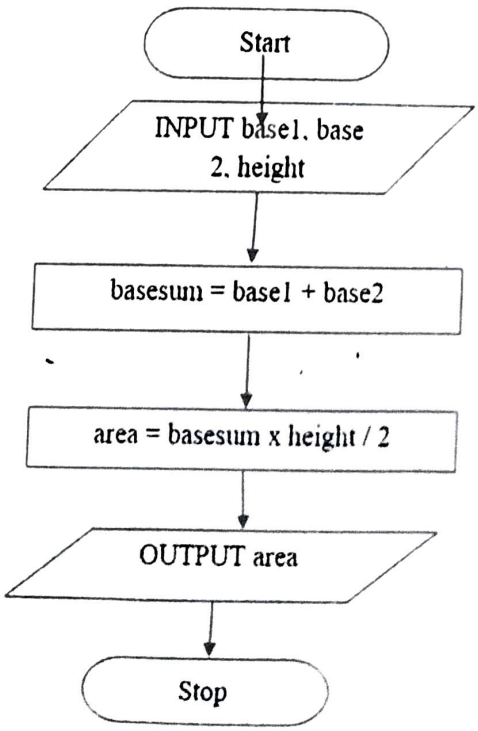
1. The Flow Chart:

In the flow chart, it is used a set of standard symbols to show the flow of the processes. The following Figure shows some of the basic flowchart symbols.



Some basic flowchart symbols

The following Figure shows the flow chart for the previous example:



Date of lecture / / 20

Note:

2. The Algorithm in pseudocode:

Step 1: INPUT base1
Step 2: INPUT base2
Step 3: INPUT height
Step 4: basesum = base1 + base2
Step 5: area = basesum x height / 2
Step 6: OUTPUT area
Step 7: STOP

Date of lecture / /20

Test:

Desk checks that this sequence of steps is an algorithm whose execution solves the intended problem. Try the reasonable values

base1 = 10

base2 = 20

Height = 15

Then

basesum = 10 + 20 = 30

And

Area = 30 x 15/2

= 225

You can see that the sequence of steps is critical because it does not make sense to compute the area of the trapezoid (step 5) until the basesum has been calculated (step 4). Similarly, the area cannot be output until it has been calculated.

Date of lecture

Note that we capitalize the words *INPUT*, *OUTPUT*, and *STOP* as part of an algorithm language that is sometimes called pseudocode because of its resemblance to high-level programming languages. Also, note that the words *base1*, *base2*, *height*, *basesum*, and *area* are in lowercase. These words are not part of the pseudocode but are arbitrarily chosen to represent the data manipulated by the algorithm.

Note

Mathematically, *base1*, *base2*, etc. represent the names of **variables** because their values can change depending on the actions taken in response to the algorithm. In programming they are called **identifiers** or **variable identifiers** because they serve to identify, by name, the memory locations in the computer where the corresponding data are stored. Although the names chosen are arbitrary, as a matter of style it is better to choose names that are **mnemonic**-that is, suggest what the name means.

B. Selection

Selection is the choice of alternate paths (branches) depending on a possibility that may arise in the logical flow of the algorithm. The ability (and necessity) to allow for possibilities is also the biggest source of problems with algorithms. When writing an algorithm, you must think ahead to all possible conditions that might occur and include steps for appropriate action. This is hard to do and explains why computer software sometimes has hidden problems called **bugs**.

Date of lecture / / 20

➢ Example 3:

Problem:

Write an algorithm that accepts a number representing either a Fahrenheit or a Celsius temperature scale and converts it to the other scale.

Reasoning:

To solve this you need to know the formulas for converting from Fahrenheit to Celsius and from Celsius to Fahrenheit.

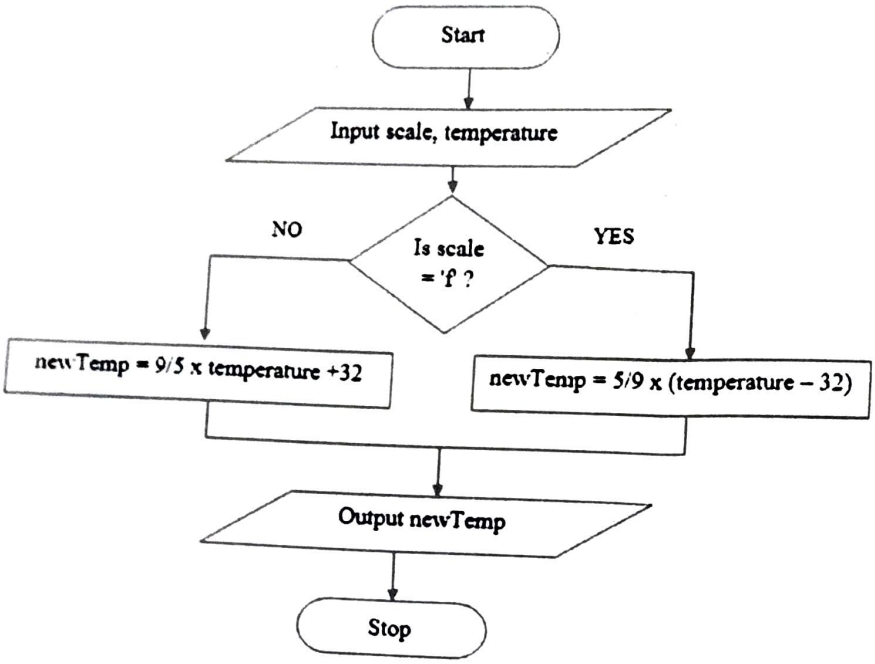
Formal statement:	temperature conversion problem
Input:	scale (Fahrenheit or Celsius), temperature
Output:	converted temperature
Process:	CelsTemp = 5/9 (FahrenTemp – 32)
	FahrenTemp = 9/5 CelsTemp +32

Note

Solution (Design the algorithm):

The algorithm must make a selection based on the user's input values and take an appropriate action. The action is dependent on the value of *scale*, which is unknown at the time the algorithm is being written. Therefore, one scale or the other must be expected and processed.

1. The algorithm as a flow chart:



2. The Algorithm in pseudocode:

Step 1: INPUT scale
Step 2: INPUT temperature
Step 3: IF scale = 'f' THEN
newTemp = 5/9 x (temperature – 32)
ELSE
newTemp = 9/5 x temperature +32
Step 4: OUTPUT newTemp
Step 5: STOP

Test:

You can easily check this algorithm by trying various values for temperatures such as 212 Fahrenheit and 100 Celsius, which are equivalent.

Step 3 of the previous example introduces the new pseudocode IF... **THEN...ELSE,** which is a **selection structure.** This can be read as "*IF* it is true that the scale equals f (Fahrenheit), THEN convert the input temperature to Celsius; otherwise (ELSE) convert it to Fahrenheit."

Notice that we have introduced **pretty printing** to show an indentation in the algorithm. In step 3 of the algorithm of the previous example, statements that are affected by the *IF* statement are indented from the **structure heading** to show that they form the **body** of the selection structure. Through pretty printing, it should be visually apparent that step 4 of the example not affected by the selection and is to be performed after the structure has terminated regardless of which branch was taken at step 3.

Date of lecture

➢ Example 4:

Problem:

Assume that a salesperson is paid a commission based on the number of sales made during the week. The salesperson is paid a commission of $8 per sale for fewer that the established quota of 15 sales, $12 per sale if the quota is reached, and $16 per sale if the quota is exceeded. Write an algorithm to find the salesperson commission.

Date of lecture

Reasoning:

The problem is to compute a commission based on a pay rate and number of sales made. The pay rate is specified in three tiers based on the sales and the preset quota.

Formal statement:	sales commission problem
Input:	number of sales
Output:	commission
Process:	commission = rate x number of sales
	Where rate is determined as
	$8/sale for fewer than 15 sales
	$12/sale for exactly 15 sales
	$16/sale for more than 15 sales

Note:

Solution:

This problem requires multiple selection based on the input value of *sales*, which is unknown at the time the algorithm is being written.

1. The Algorithm in Flow Chart:

Homework: Draw the flow chart for the algorithm that solves this problem.

Date of lecture / / 20

2. The Algorithm in pseudocode:

```
Step 1:  INPUT sales
Step2 : INPUT quota
Step 3:  IF sales < quota THEN
             Rate = 8
         ELSE IF sales = quota THEN
             Rate = 12
         ELSE
             Rate = 16
Step 4:  commission = rate x sales
Step 5:  OUTPUT commission
Step 6:  STOP
```

Note: .....

Test:

To check this algorithm you must verify that it can handle all the specified cases. Do this by trying values that are below the quota, above the quota, and at the quota.

Step 2 of the previous example introduces **multiple selections** and can be read as "*IF* the sales are less than the quota, THEN the rate = 8; otherwise, *IF* the sales equal the quota, THEN the rate = 12; otherwise by default the rate = 16." This structure can be extended indefinitely by repeated use of ELSE IF.

Date of lecture / / 20

Notice that in this example, the pay rate must be determined before the commission can be computed (in sequence). Also notice that the last ELSE of the multiple IF becomes the default or catchall case that is considered if none of the other possibilities is true. Finally, notice that for any given run, only one path (one value for rate) can be chosen. This structure is sometimes called **mutual exclusion** because choosing one path excludes all the remaining ones.

Note:

Although, input errors are seldom deliberate, one of the rules of program development is that all input should be tested to ensure that it is reasonable within the boundaries of the problem. This is important to avoid the possible data entry errors (consider an input of negative number in the previous example). The algorithm will produce an output which is negative number (illogical output). Here it seems reasonable that the number of sales must be at least 0 unless returns are included in the commission scheme. What should the upper bound be? 100? 1,000? There is no easy answer because whatever number you choose is arbitrary. More information would be needed about this particular problem to set an appropriate upper bound.

C. Repetition

The third structure used in the development of an algorithm is called **repetition,** or **looping,** which provides for repeated execution of part of the algorithm. An obvious deficiency in example 4 is its inability to handle more than one salesperson. You could replicate steps 1 through 4 for each salesperson. Then you have the accounting job of writing these four steps over and over, especially for a large sales staff, and the problem of hiring and firing. There may be 20 salesperson at one time and 50 at another. It seems unreasonable to have to expand and shrink an algorithm depending on the business cycle! One way to deal with this problem is shown next.

Date of lecture / / 20

➢ Example 5:

Problem:

Reconsider the problem in example 4 but allow for an entire sales staff.

Reasoning:

This is essentially the same problem as example 4 with the addition of a **looping structure**. The looping structure processes the data for each salesperson. The input is adjusted to include the number of salespeople to be paid.

Date of lecture

Formal statement: sales commission problem (2)

Input: number of salespeople
Number of sales for each salesperson

Output: commission for each salesperson

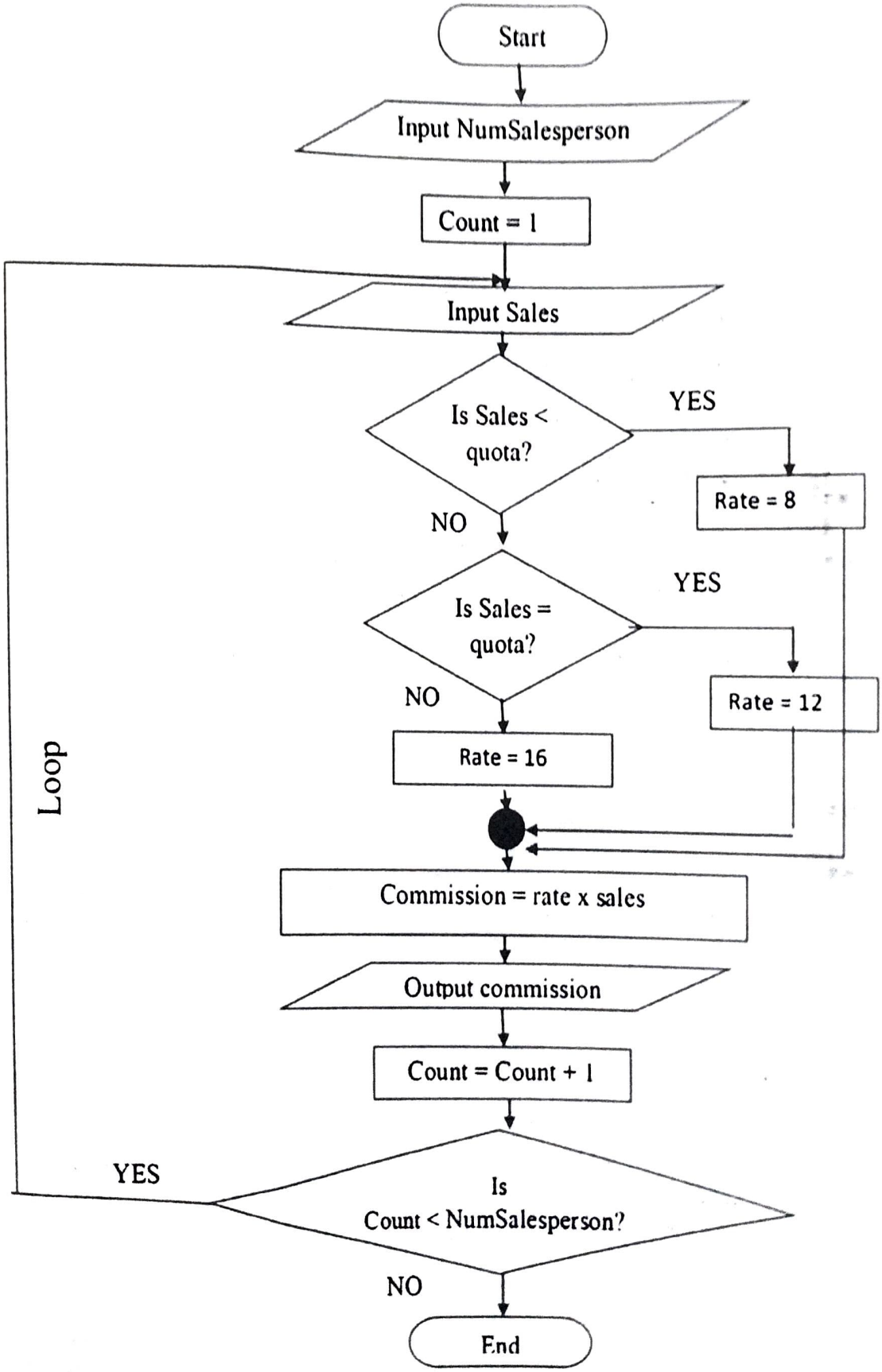
Process: commission = rate x number of sales
Where rate is determined as
$ 8/sale for < than 15 sales
$12/sale for = 15 sales
$16/sale for > than 15 sales

Note:

Solution:

A loop is used to repeatedly read the data for each salesperson and compute the associated commission. The word *LOOP* here means that the following sequence of steps (2a to 2d in this case) is to be repeated for a specific number of iterations (numSalespeople in this case).

1. The Algorithm in Flow Chart:



Date of lecture / / 20

Note:

2. The Algorithm in pseudocode:

```
Step 1: INPUT numSalespeople
Step 2: LOOP numSalespeople times
INPUT sales
IF sales < quota THEN
  Rate = 8
            ELSE IF sales = quota THEN
              Rate = 12
            ELSE
              Rate = 16
Commission = rate x sales
OUTPUT commission
Step 3: STOP
```

Test:

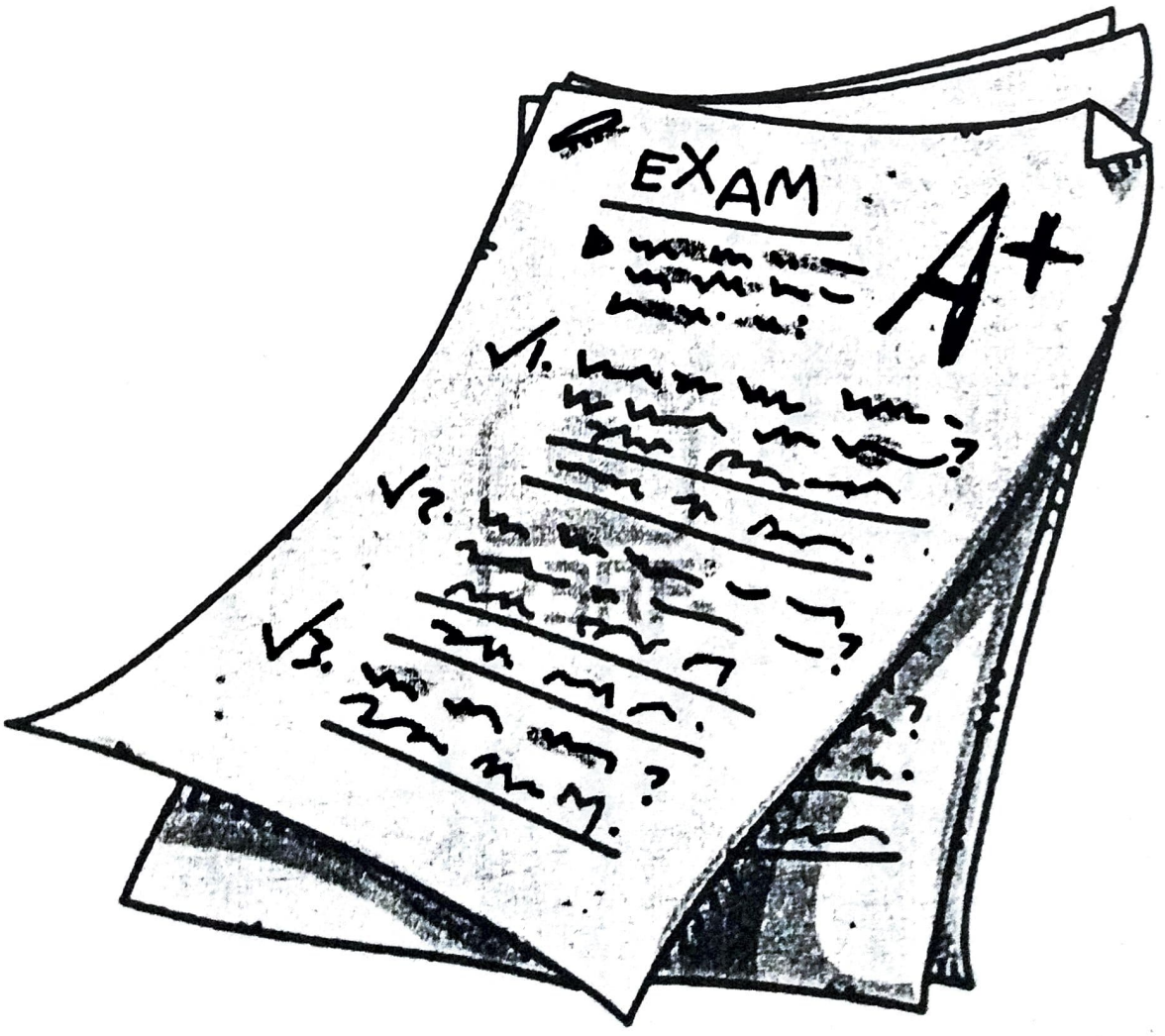
You can test this algorithm by desk checking; that is, by trying different values for sales (above, below, and at the quota) and calculating the results.

The algorithm in Example 5 assumes that the number of salespersons currently on the payroll is known. The loop construction is only appropriate if the number of iterations can be predetermined as it is here. For many situations, however, this is not possible and a more flexible looping structure is needed.

Date of lecture

Note:

Exercises



1- Design a flow chart and pseudo code that receive 2 integers and calculate / display their Sum and Average.

2- Design a flow chart and pseudo code that receive 2 integers from the user and Calculate / display their Difference, Product and quotient.

3- Design Flow chart and pseudo code that will prompt for the price of a computer and tax rate, then calculate the tax and new price, then display them to the screen.

4- Design Flow chart and pseudo code that will prompt for length and width of a rectangle and radius of a circle, receive them, then calculate the area of rectangle and area of circle and its circumference and display them to the screen.

5- Design a flow chart and pseudo code that will receive 3 numbers then find the maximum and display it to screen.

6- Design a flow chart and pseudo code that will prompt for 2 numbers and accept them then sort them into ascending order, then display them to the screen.

7- Design a flow chart and pseudo code that will ask for student score in a subject, and receive it then match the score to a letter(s) grade, then display grade to the screen. the letter grade is to be calculated as follows:

Exam Score	Assigned Grade
85 and above	Ex
75-84	VG
65-74	G
50-64	Pass
Below 50	Fail

8- Design a flow chart and pseudo code that accepts pairs of integers from the user and Calculate / display their Sum, Difference, Product and quotient. The program will terminate after 5 entries.

9- Design a flow chart and pseudo code that will ask for 10 temperatures expressed in degree Fahrenheit. Then it will accept each Fahrenheit temperature, then convert it into Celsius, display the converted temperature to the screen.

$$Celsius = (Fahrenheit - 32) * 5 / 9$$

10- Design a flow chart and pseudo code that will accept pairs of numbers , calculate the sum, if the sum is +ve then display a message beside it says "the sum is positive" and if the sum is –ve then display a message beside it says "the sum is negative". The program will terminate after 10 entries is entered.

11- Design a flow chart and pseudo code that will calculate the factorial of a number and display the result.