RA- 1b

kdgl37g

BAI-1B

vwvjzoo

Programming Fundamentals

CS1002

Fall-2022

26th August 2022

Dr. Muhammad Farrukh Shahid

Computer Science Department NUCES-FAST Karachi

Welcome to the Course

Congratulations

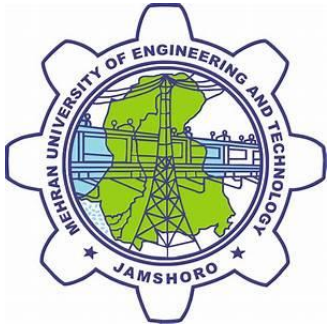
&

Mubaraks

About myself


QUEEN MARY
UNIVERSITY OF LONDON

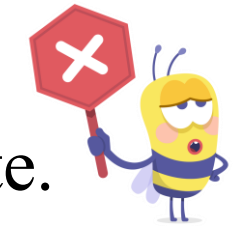

UNIVERSITÀ DEGLI STUDI
DI GENOVA


MEHRAN UNIVERSITY OF ENGINEERING AND TECHNOLOGY
JAMSHORO


COMSATS


PAF
KIET

Class Policies – Pay attention

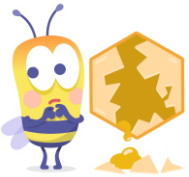


- Don’t come into the class if you are late.
- Don’t sit in my class for passing the time or you have less interest.

- Attend the class with full motivation and inspiration.

- **80 % Attendance** must be maintained

Remember you are in the class to learn something new.



Class Room Policies

Activities such as Lecture recording and taking Photos of the board are not allowed

Course Policies – Pay attention

- Make a separate register for practice the problems ***(Mandatory)***
- Assignments must be submitted with in ***due dates***.
- Late submission will be subjected to the penalty which is as follow:

After ***2*** days of deadline ***30 %*** of marks deductions
After ***4*** days of deadline ***40 %*** of marks deductions
After ***5*** days of deadline ***100 %*** of marks deductions

- Student contact hours in my office:

Tuesday ***10:00 AM - 11:00 PM and 2:00 PM to 3:00 PM***

Communication rules and Contact hours

- For your queries related to the course, send an email to the following email address with clearly mention your Class and Student ID in the SUBJECT of an email.

mfarrukh.shahid@nu.edu.pk

Or you can visit the office (in contacting hours)

Or discuss in the class (this is highly subjected to the time availability in a class)

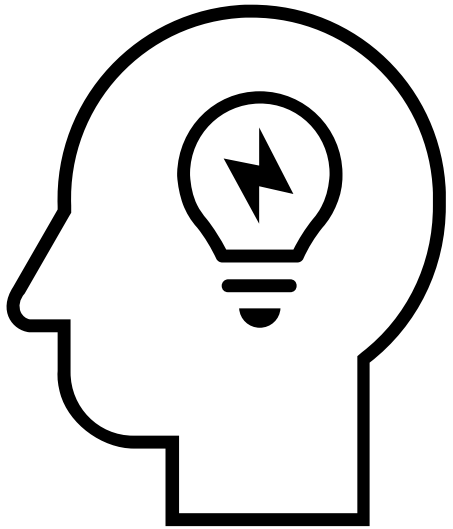
How we will proceed in a course ?

- Lectures (On board, slides (wherever necessary))
- Example problems
- Practice problems
- For practice problems – Make a separate notebook – I will check that in each class randomly.

- All you need to do is study and practice the codes !!!!!

Why do we actually need Programming Languages ?

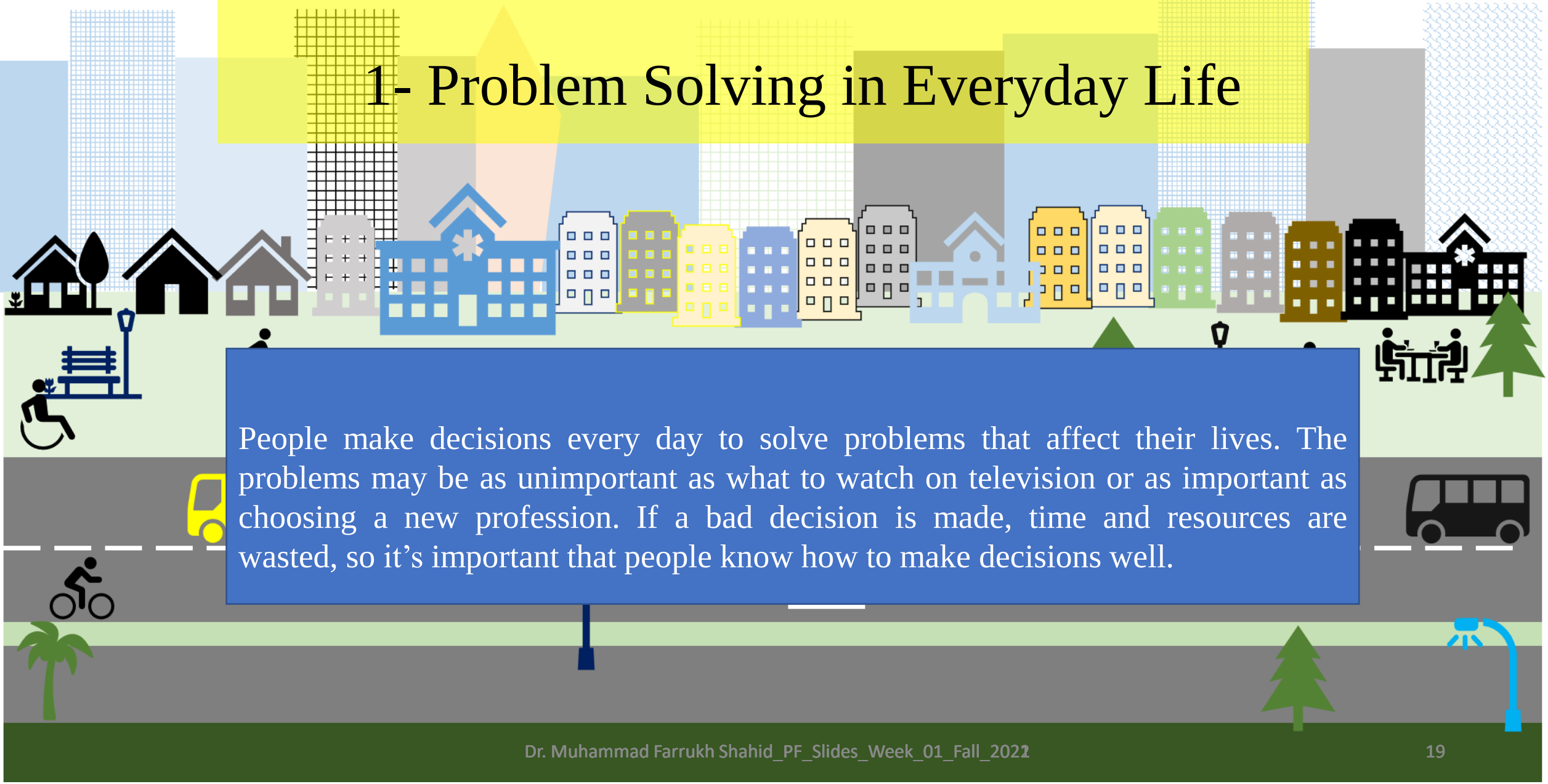
General Problem-Solving Concepts



General Problem-Solving Concepts

1. **Problem Solving in Everyday Life**
2. **Types of Problems**
3. **Problem Solving with Computers**
4. **Difficulties with Problem Solving**

1- Problem Solving in Everyday Life



People make decisions every day to solve problems that affect their lives. The problems may be as unimportant as what to watch on television or as important as choosing a new profession. If a bad decision is made, time and resources are wasted, so it's important that people know how to make decisions well.

1- Problem Solving in Everyday Life (Contd..)

Six Steps to follow

1- Identify the problem

2- Understand the problem

3- Identify alternative ways to solve the problem

4- Select the best way to solve the problem from the list of alternative solutions

5- List instructions that enable you to solve the problem using selected solution

6- Evaluate the solution

2- TYPES OF PROBLEMS

Algorithmic Solutions

- Some problems require series of actions to be performed to get a solution or desired goal

→ Cooking food, Driving a car

Heuristic Solutions

- Requires analytic reasoning, some past knowledge and experience to get a desired solution

→ Stock market, bitcoin, Self driving cars, robots, UAVs etc..

Heuristic Solutions – Key Points

The problem solver can use the six steps for both algorithmic and heuristic solutions. However, in step 6, evaluating the solution, the correctness and appropriateness of heuristic solutions are far less certain. It's easy to tell if your completed check book balance is correct and satisfactory, but it's hard to tell if you have bought the best stock.

With heuristic solutions, the problem solver will often need to follow the six steps more than once, carefully evaluating each possible solution before deciding which is best.

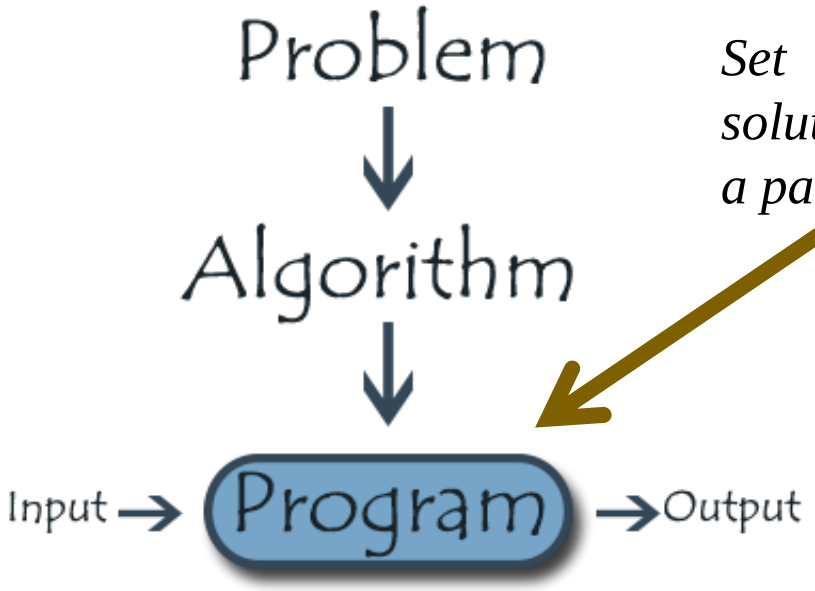
Furthermore, this same solution may not be correct and satisfactory at another time, so the problem solver may have to reevaluate and resolve the same problem later. The stock that did well in January may do poorly in June.

Most problems require a combination of the two kinds of solutions

ALGORITHM FOR GOING TO THE MARKET TO BUY SWEETS (Methai) ON YOUR ADMISSION INTO THE UNIVERSITY

ALGORITHM FOR GOING TO THE MARKET TO PURCHASE A NEW BOOK FOR YOUR PF COURSE

3- Problem Solving with Computers



Set of instructions that make up the solution after they have been coded into a particular computer language.

- *Computers are built to deal with **Algorithmic solutions**, which are often difficult for humans.*
- *People are better than computers a developing heuristic solutions- throw a ball, speak Italian, French and English*
- *AI enables computers to deal with **Heuristic type of problems** – **Modern AI and Cognitive Systems***

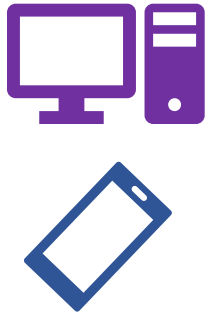
Can cognitive agents feel the emotions ?

4- Difficulties with Problem Solving

People may have difficulties in solving problems

(Weak visualization of the problem, can't make decisions etc.)

Problem solving requires practice and time synchronization.



While solving problems on/with the ***Agents***, one of the most critical jobs for the problem solver is writing the **instructions**.

What you have been introduced so far ?

Algorithm

Heuristic solution

Program

Solution

Instructions

Class Activities

1.1- Name two current problems in your life that could be solved through an algorithmic process.

Complete the six problems-solving steps to solve above problems

1.2 - Name two current problems in your life that could be solved through an heuristic process.

Complete the six problems-solving steps to solve above problems

Class Activities (contd..)

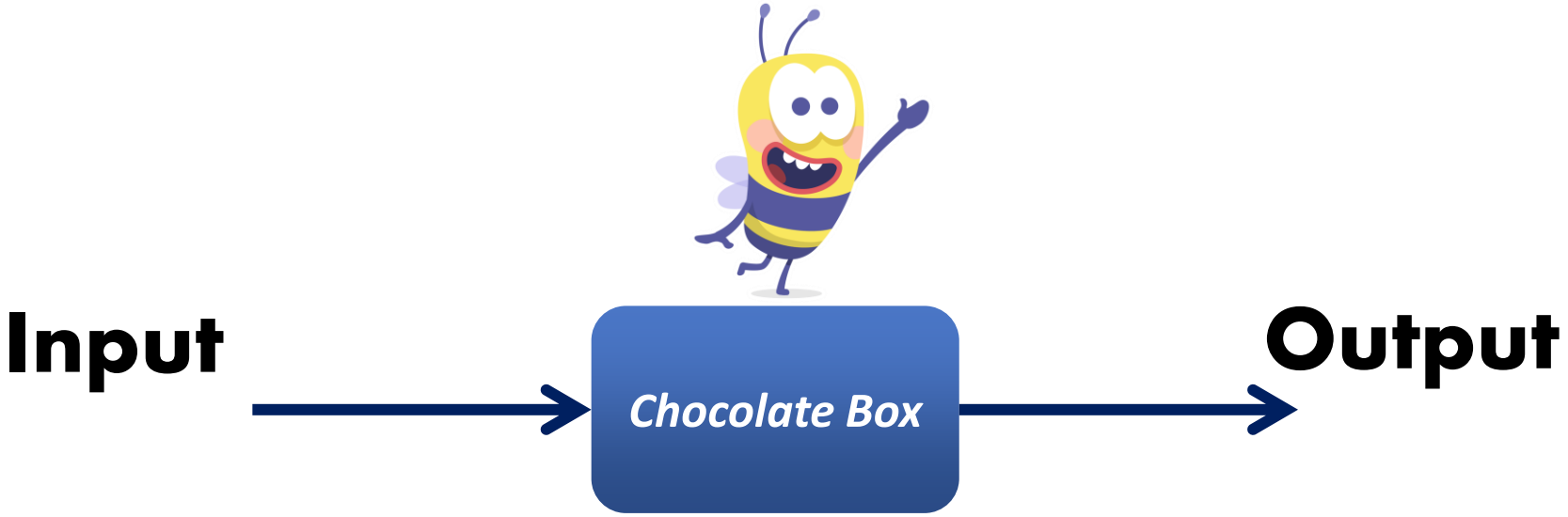
1.3 Write step-by-step instructions for the following tasks so that another person can understand very well.

a) Make a cup of coffee/ tea

b) Reset your mobile phone

c) Creating a Gmail account.

Some insights on Inputs and outputs of an agent



Planning Your Solution

Planning your Solution

The computer must be told what to do?

WHAT PROBLEM CAN BE SOLVED BY COMPUTER ?

- The solution can be produced by a set of step-by-step procedures or actions.
- This step-by-step action is called an ***algorithm***.
- The algorithm will process some inputs and produced output.
- Solving problem by computer undergoes **two phases**:
- Phase 1:
  - ***Organizing the problem or pre-programming phase*.**
- Phase 2:
  - ***Programming phase.***

Planning your Solution (contd..)

1- Pre-programming phase.

This phase requires four steps:

a) Analyzing the problem using ***Problem Analysis Chart (PAC).***

b) Developing ***Hierarchy Input Process Output (HIPO)*** chart or ***Interactivity Chart (IC)*** and ***Input Process Output (IPO).***

c) Writing an Algorithms.

d) Drawing the Program flowcharts.

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

a) Analyzing the problem.

- To organize a solution, the programmer must first understand and analyze the requirements of a problem.
- A good way to analyze a given problem is to separate it into four parts as shown in Fig.

Given Data	***Required Results***
Section 1: Data given in the problem or provided by the user.	Section 2: Requirements for the output.
Processing Required	***Solution Alternatives***
Section 3: List of processing required. Equations, expressions etc..	Section 4: List of ideas for the solution of the problem.

Fig. Problem Analysis Chart (PAC)

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

a) Analyzing the problem. (Contd…)

Example:1

Build a PAC chart to calculate the salary of an employee given the hours worked and the payrate.

The formula to be used is:

Salary = Hour works * Pay rate

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

a) Analyzing the problem.

Example: 1 SOLUTION

Build a PAC chart to calculate the salary of an employee who works by hourly basis.
The formula to be used is:

Salary = Hour works * Pay rate

Given Data	**Required Results**
Hour works, Pay rate	Salary
Processing Required	**Solution Alternatives**
*Salary = Hour works * Pay rate*	Define the hours worked and pay rate as constant Define the hours worked and pay rate as input values

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

a) Analyzing the problem. (Contd…)

Example:2

Write a Problem Analysis Chart (PAC) to find an area of a circle

where area = pi * radius * radius

Data	Processing	Output
Radius	Area = 3.14 x radius x radius	Area

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

a) Analyzing the problem. (Contd…)

Example:3

Write a Problem Analysis Chart (PAC) to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulas are:

$$Celsius = 10 \times (depth) + 20$$

$$Fahrenheit = 1.8 \times (Celsius) + 32$$

Data	Processing	Output
depth	celsius = 10 x (depth) + 20 fahrenheit = 1.8 x (celsius) + 32	Display celsius, Display fahrenheit

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

a) Analyzing the problem. (Contd…)

Example:4

Write a problem analysis chart (PAC) that asks a user to enter the distance of a trip in miles, the miles per gallon estimate for the user's car, and the average cost of a gallon of gas. Calculate and display the number of gallons of gas needed and the estimated cost of the trip.

Data	Processing	Output
distance, miles per gallon, cost per gallon	gas needed = distance / miles per gallon. estimated cost = cost per gallon x gas needed	Display gas needed Display estimated cost

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the **Hierarchy Input Process Output (HIPO) Chart** or **Interactivity Chart** (IC).

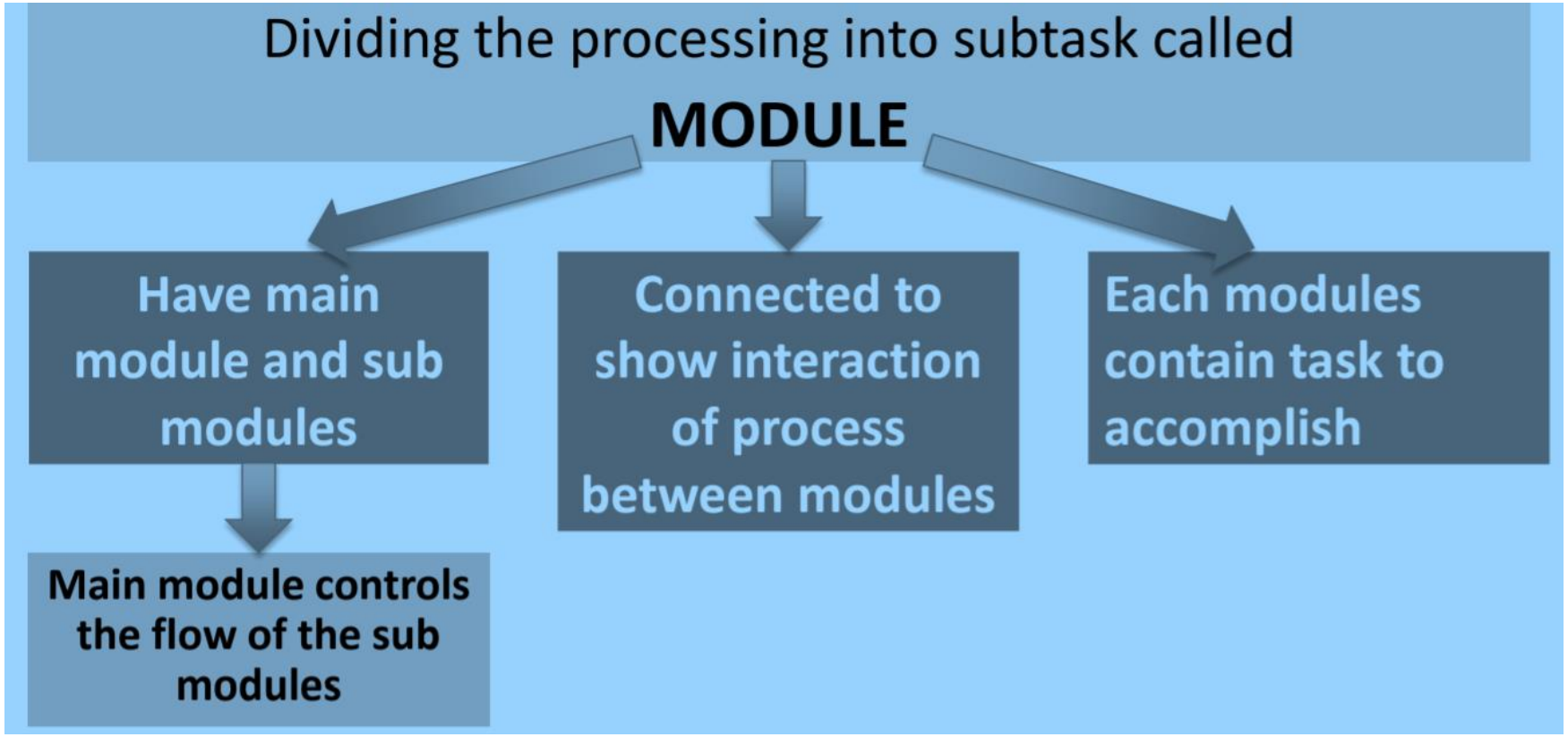
- The problem is normally big and complex.
- Thus, the processing can be divided into subtasks called ***Modules***.
- Each module accomplishes one function.
- These modules are connected to each other to show the interaction of processing between the modules.
- **Main / Control** module controls the flow all other modules.

- The ***IC*** is developed using top-down-method: top to down left to right order (also refer to order of processing).
- Modules are numbered, marked for duplication, repetition or decision.

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC) (Contd..)

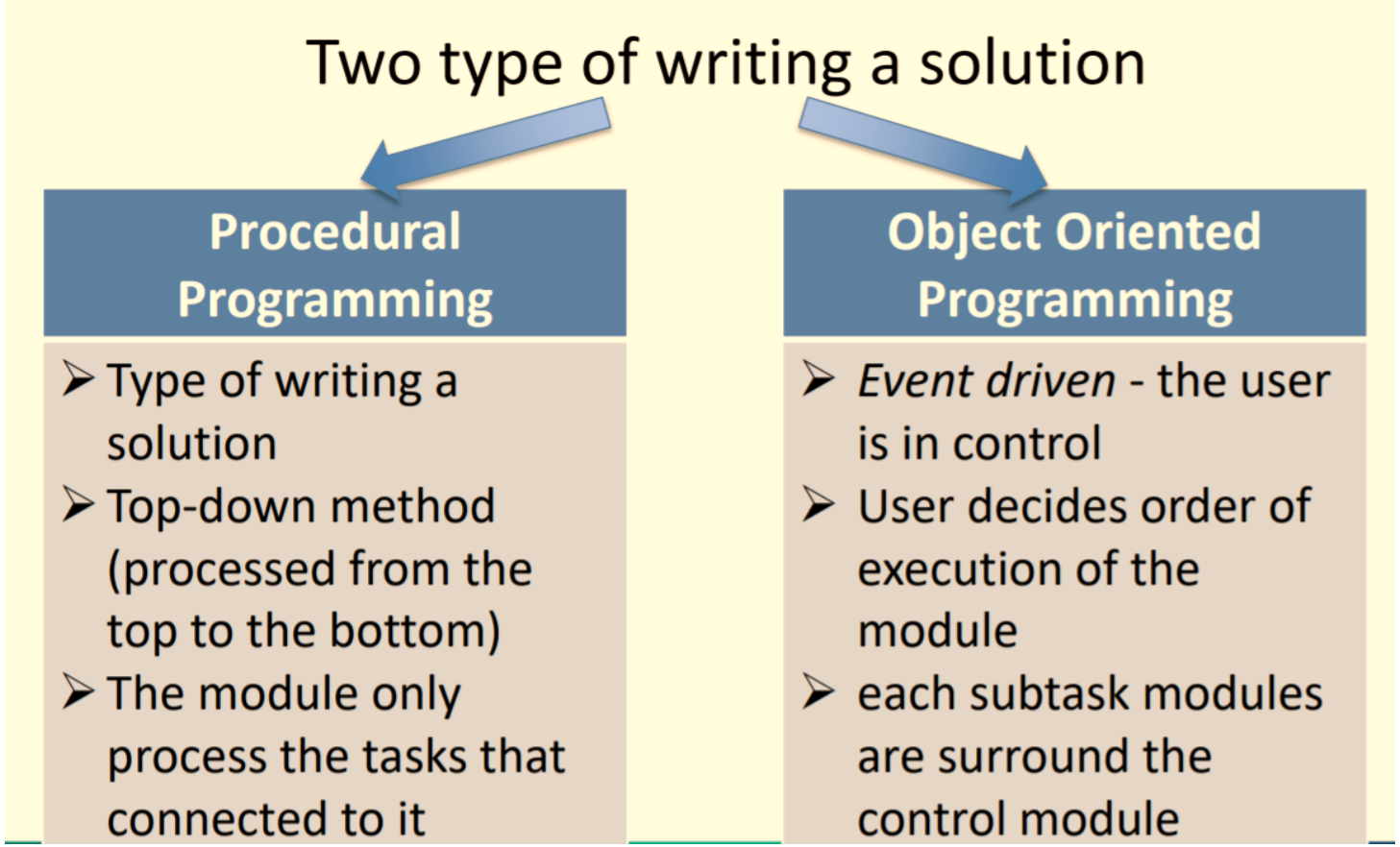


Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC) (Contd..).

Today's programmer will use two different ways of looking at a solution to a problem. Before the advent of the graphic user interface, all solutions were procedural in nature.

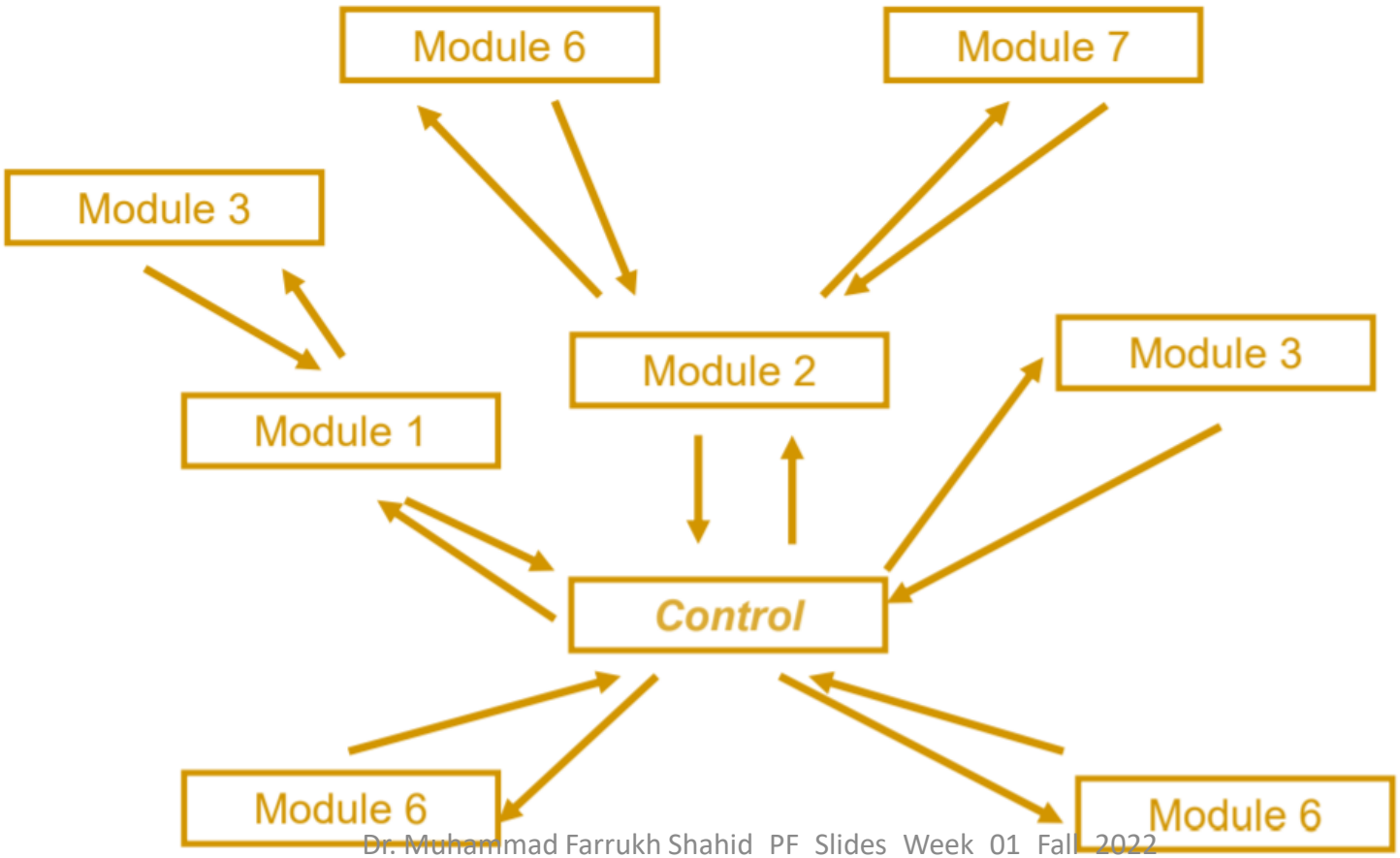


Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC) (Contd..).

Object Oriented Programming



Planning your Solution (contd..)

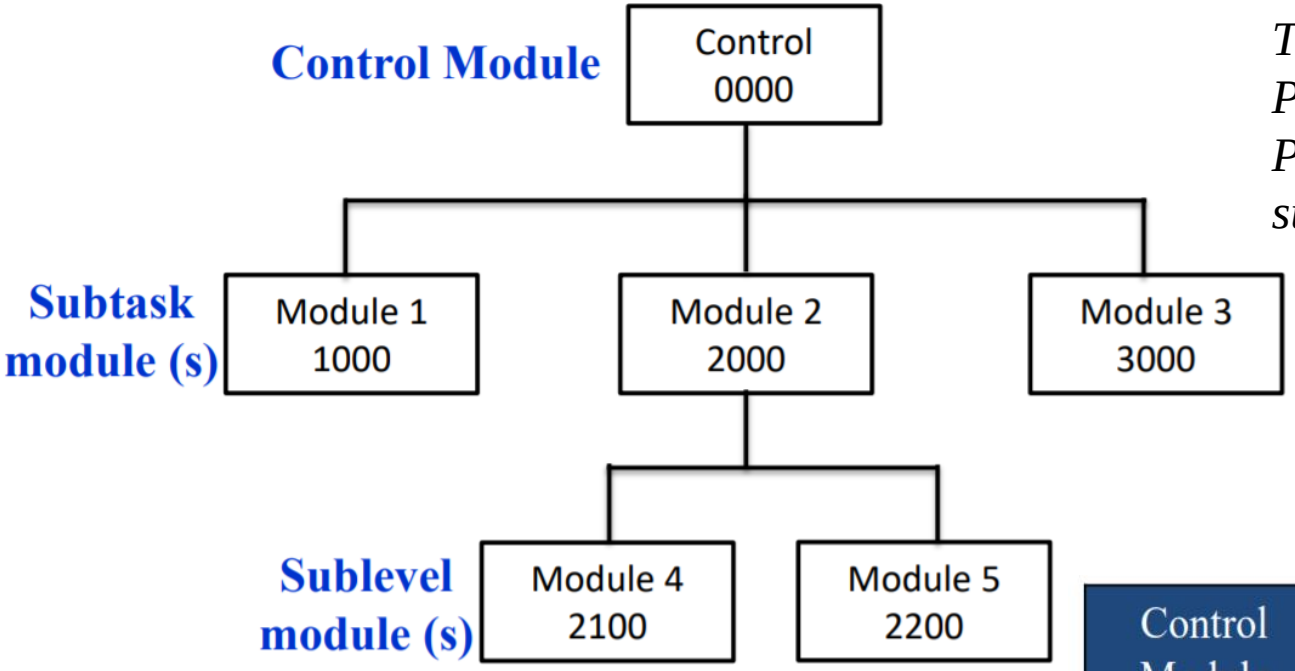


Fig. The Interactivity Chart

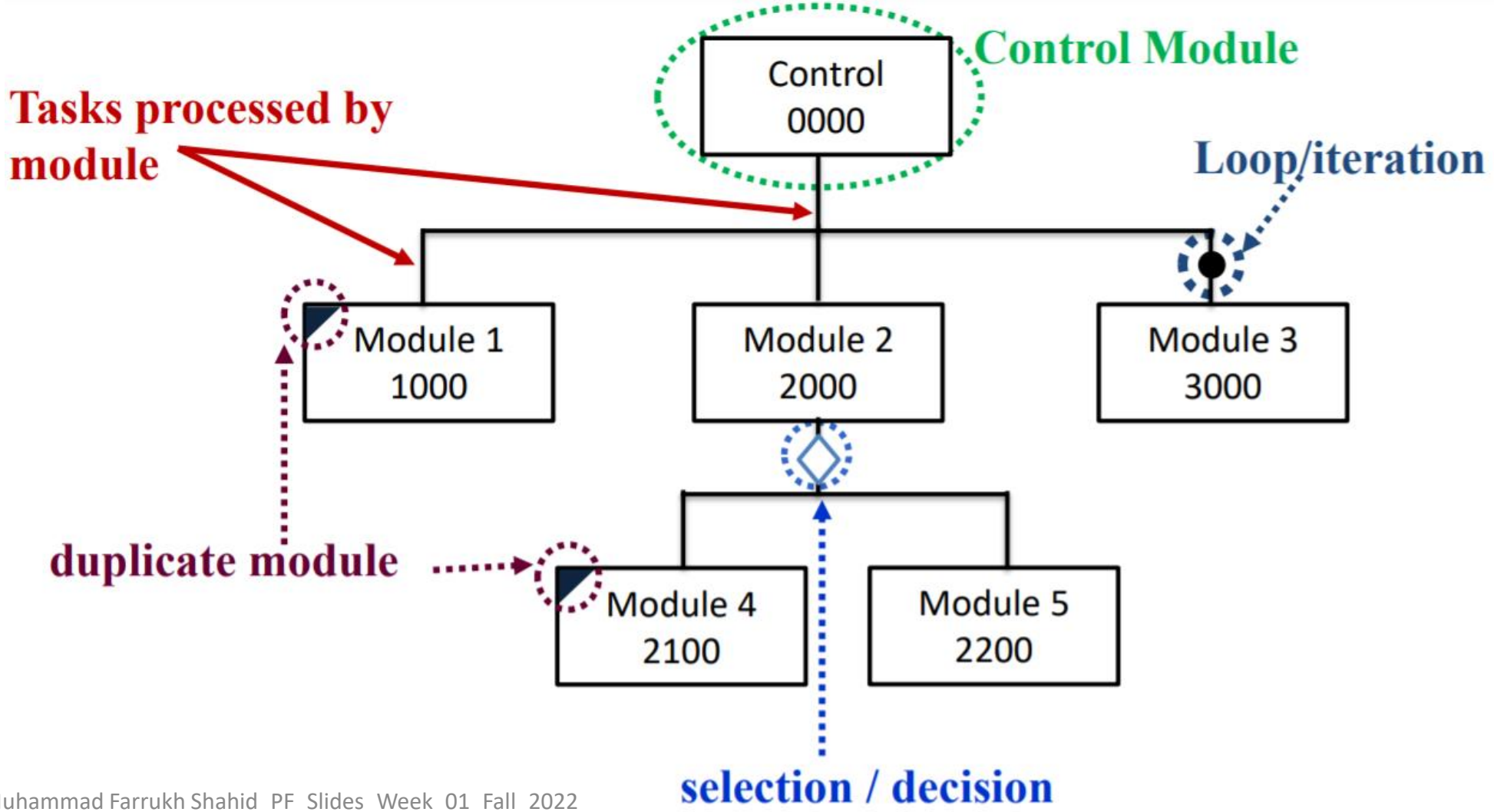
The interaction will form a hierarchy, called Hierarchy Input Process Output Chart (HIPO) or Interactivity Chart (IC). Programming which uses this approach (problem is divided into subtasks) is called ***Structured Programming.***

Control Module On top Labeled (0000)	Controls the **processing of all the data**. The chart is display in top-down method and it means that as you divide the problem into subtasks, they demonstrate the order in which processes will occur from the top to the bottom of the chart.
Subtask Module(s)	The next level of rectangles starting with the number 1000, 2000, 3000 and increases **from left to right by increments of 1000**.
Sublevel Module(s)	The next level of rectangles starting with increments of 100 as in 1100

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)



Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

- Dividing a solution into modules is not an easy job.
- Control module is required.

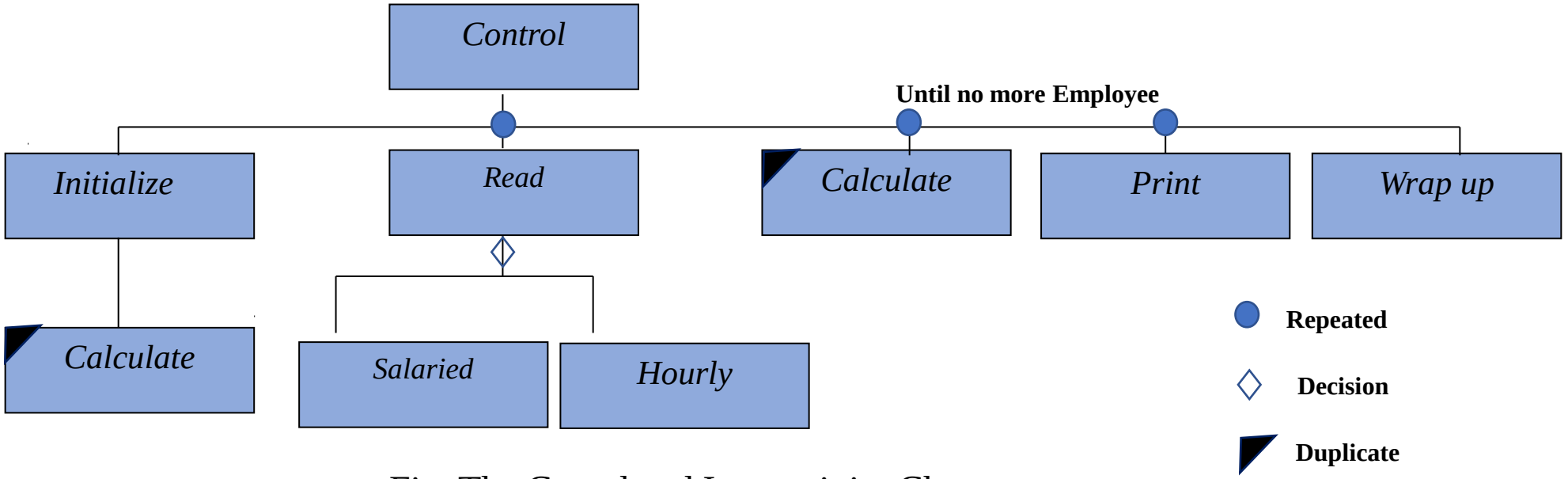


Fig. The Completed Interactivity Chart

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Example: 01 *Develop an IC for the payroll problem.*

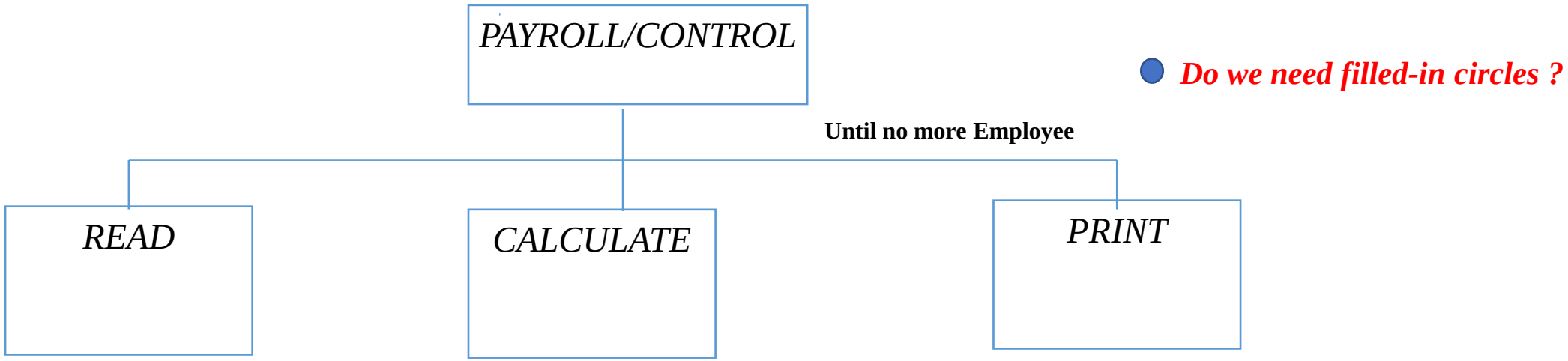


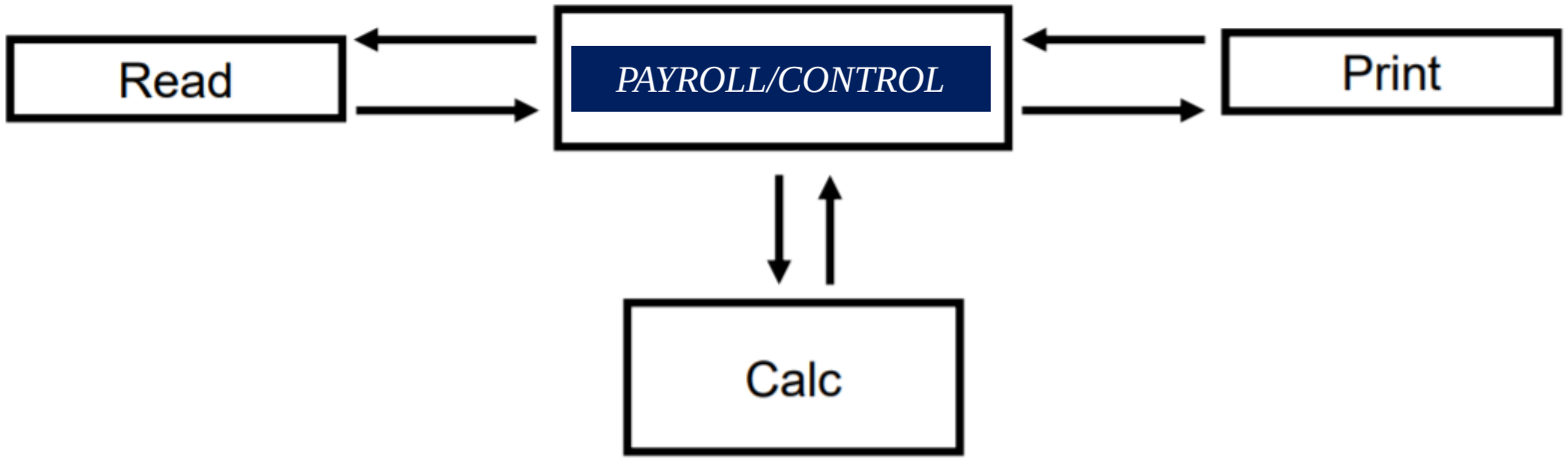
Fig. The Completed Interactivity Chart for the Payroll Problem

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Object Oriented Programming



Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Example: 02

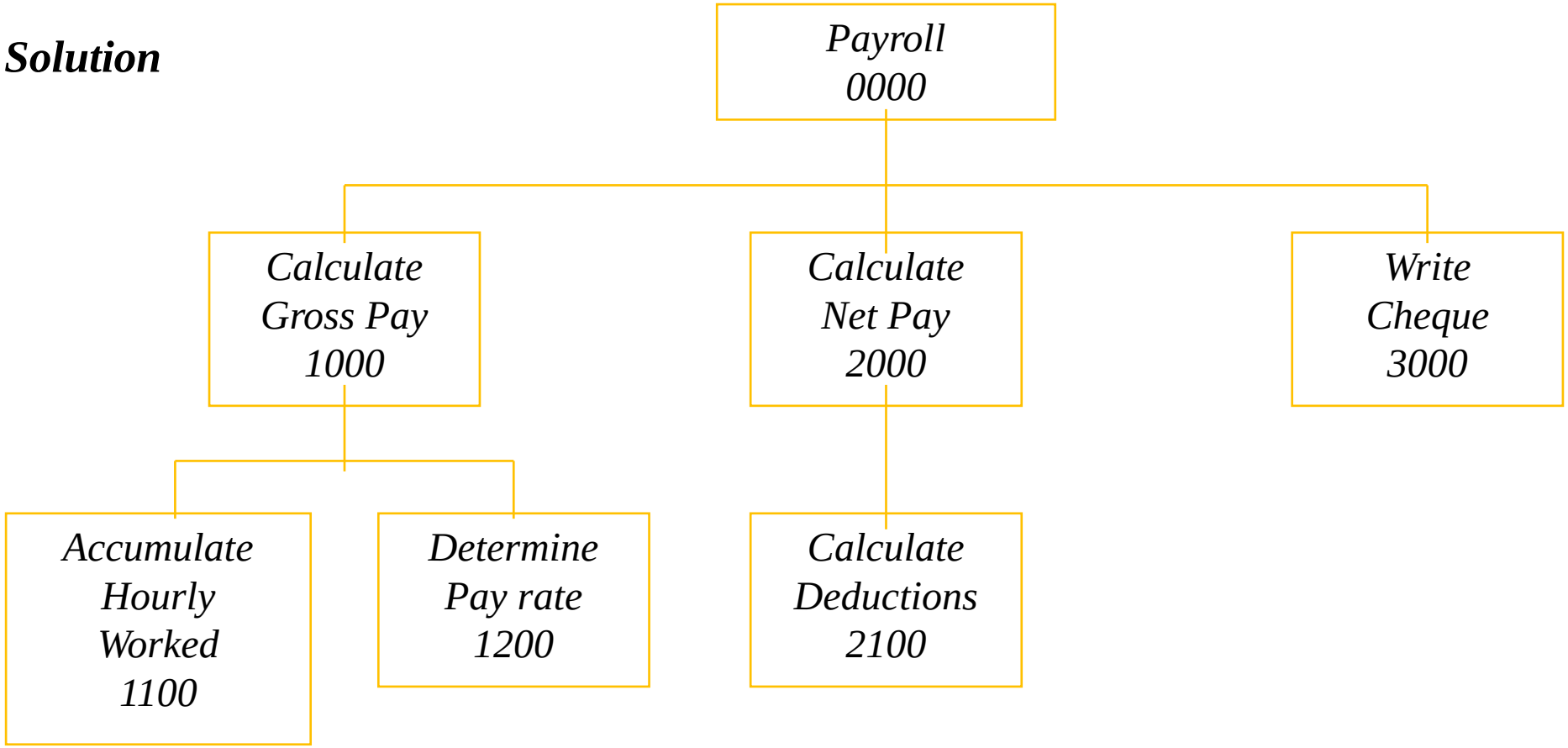
You are required to write a program to calculate both the gross pay and the net pay of every employee in your company. To determine the gross pay, you have to multiply the ***accumulated total hours*** *worked by the employee and* ***appropriate pay rate****. The program should print the cheque that shows the total net pay. The net pay is calculated by subtracting the gross pay from any deductions that may be incurred by the employee.*

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Example 2 Solution



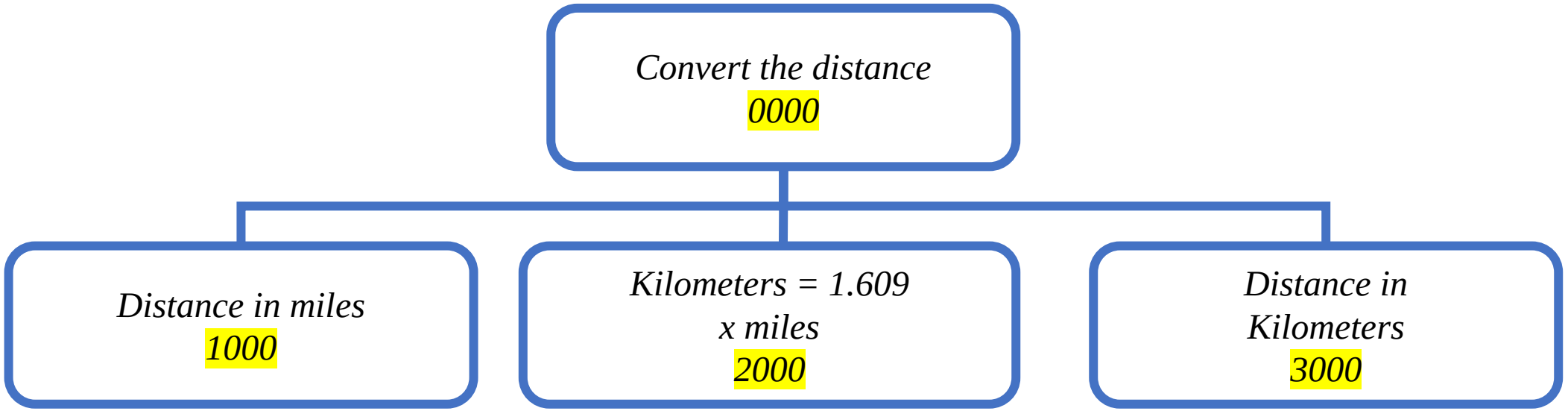
Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Example 3:

Write a Hierarchy Input Process Output (HIPO) to convert the distance in miles to kilometers where 1.609 kilometers per mile.



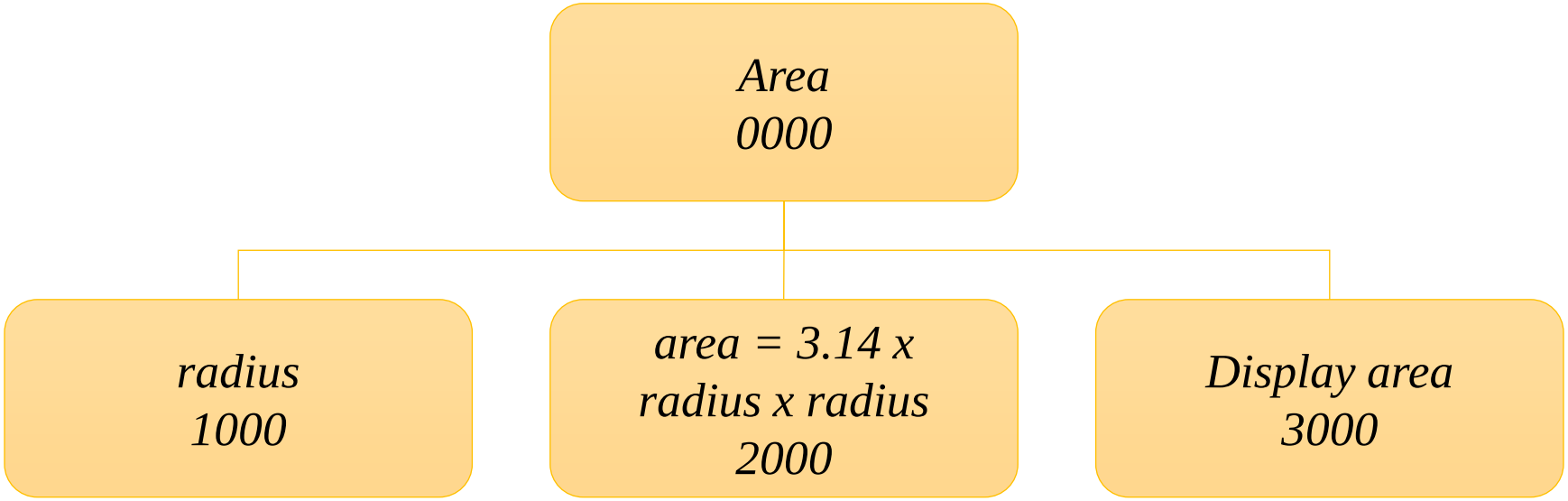
Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Example 4:

*Write a Hierarchy Input Process Output (HIPO) to find an area of a circle where area = pi * radius * radius*



Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Example 5:

Write a Hierarchy Input Process Output (HIPO) to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulas are

$$Celsius = 10 \times (depth) + 20$$

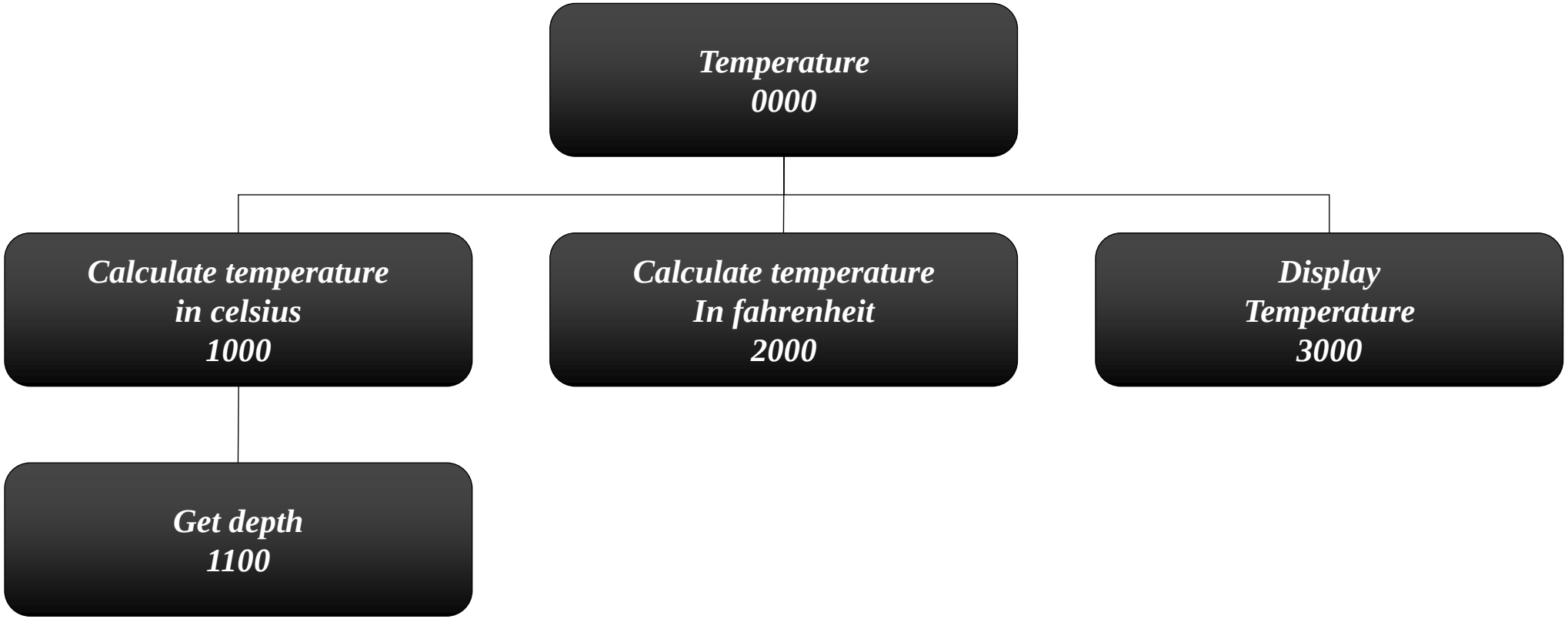
$$Fahrenheit = 1.8 \times (Celsius) + 32$$

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the Hierarchy Input Process Output (HIPO) Chart or Interactivity Chart (IC). (Contd..)

Example 5: SOLUTION



Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the ***IPO*** Chart

- The IPO (input-processing-output) chart extends and organizes the information in the PAC.
- It shows in more detail what data items are required, what processing takes place on that data and what information will be the result.
- It combines information from ***PAC*** and ***HIPO*** Chart.

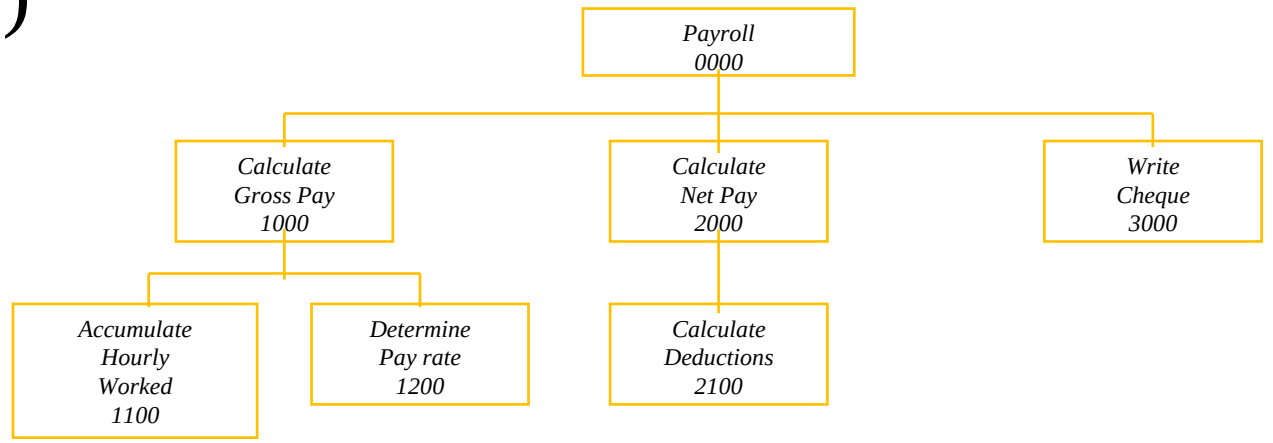
Input	***Processing***	***Module***	***Output***
All input data (from Section 1 of the PAC)	*All processing in steps (Section 3 and 4 of the PAC)*	*Module reference from IC*	*All output requirements (Section 1and 2 of the PAC)*

Fig. The IPO chart

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the ***IPO*** Chart (Contd..)



Example:01 **The IPO for the Payroll problem**

Input	*Processing*	*Module*	*Output*
Hours Worked -Pay Rate , deduction	*1. -Enter Hourly Worked* *2. -Enter Pay Rate* *3. -Calculate Gross Pay* *4. -Enter Deduction* *5. Calculate Net Pay* *6. -Print / Write Cheque* *7. -End*	*Read* *Read* *Calculate* *Calculate* *Read* *Calculate* *Print* *Payroll/* *Control*	*Net pay*

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

b) Developing the ***IPO*** Chart (Contd..)

Example:02.

Develop an Input Process Output (IPO) to convert the distance in miles to kilometers where 1.609 kilometers per mile

Input	*Processing*	*Module*	*Output*

Example:02 SOLUTION

Input	*Processing*	*Module*	*Output*
-Distance in Miles	*-Enter distance* *-Kilometers = 1.609 x distance* *-Display kilometers*	*1000* *2000* *3000*	*-Distance in kilometers*

Some more examples !

1. Write an Input Process Output (IPO) to find an area of a circle
where area = pi * radius * radius

2. Write an Input Process Output (IPO) to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulas are

Celsius = 10 x (depth) + 20

Fahrenheit = 1.8 x (Celsius) + 32

3. Write an Input Process Output (IPO) that asks a user to enter the distance of a trip in miles, the miles per gallon estimate for the user's car, and the average cost of a gallon of gas. Calculate and display the number of gallons of gas needed and the estimated cost of the trip.

ALGORITHMS AND FLOW CHARTS

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

c) – Writing an algorithm

- After developing IC and IPO the next step for a programmer is to develop sets of instructions for the computer, called Algorithms.
- **The algorithm makes use of IC and IPO to give a logical step-by-step solution**.

Control Module	Name of module (List of parameters)
1. Instruction	1.Instruction
2. Instruction	2. Instruction
3. ..	3..
4..	4..
- End	-- exit

Planning your Solution (contd..)

1- Pre-programming phase.(Contd..)

d) - Drawing the Program flowcharts

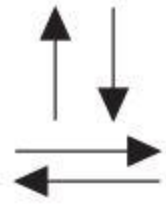
- Flowchart is the graphic representations of the individual steps or actions to implement a particular module.
- The flowchart can be likened to the blueprint of a building. An architect draws a blueprint before beginning construction on a building, so the programmer draws a flowchart before writing a program.
- *Flowchart is independent of any programming language.*

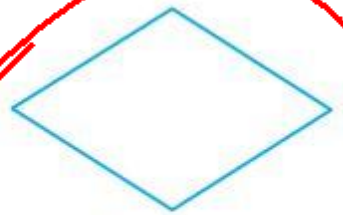
Planning your Solution (contd..)

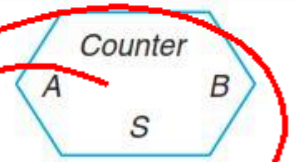
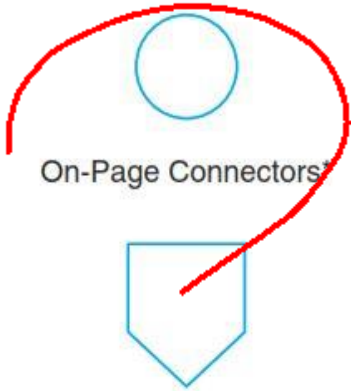
1- Pre-programming phase.(Contd..)

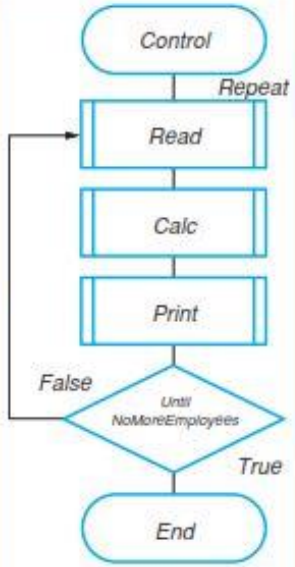
d) - Drawing the Program flowcharts (Contd..)

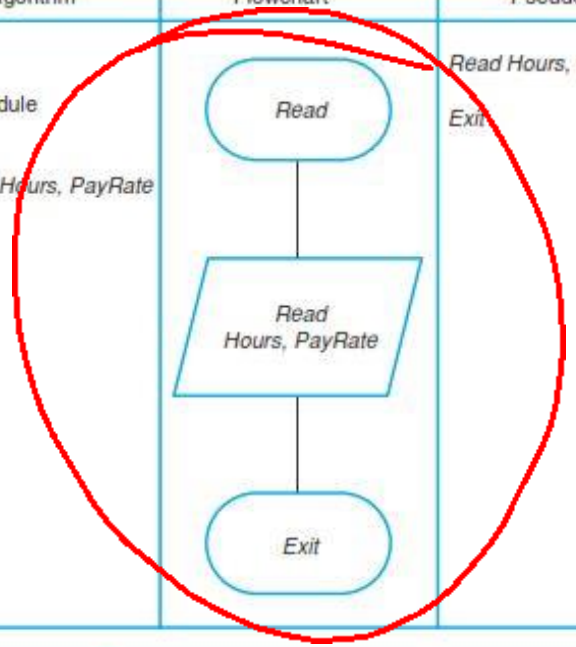
- Flowchart is the logical design of a program.
- It is the basis from which the actual program code is developed.
- Flowchart serves as documentation for computer program.
- The flowchart must be drawn according to definite rules and utilizes standard symbols adopted internationally.
- The **International Organization for Standardization (IOS)** was the symbols shown below (You can draw the symbols using ready-made flowcharting template)

Flowchart Symbol	Explanation
 Flowlines	Flowlines are indicated by straight lines with optional arrows to show the direction of data flow. The arrowhead is necessary when the flow direction might be in doubt. Flowlines are used to connect blocks by exiting from one and entering another.
 Start  End/Stop/Exit	Flattened ellipses indicate the start and the end of a module. An ellipse uses the name of the module at the start. The end is indicated by the word *end* or *stop* for the top or *Control* module and the word *exit* for all other modules. A start has no flowlines entering it and only one exiting it; an end or exit has one flowline entering it but none exiting it.
 Processing	The rectangle indicates a processing block, for such things as calculations, opening and closing files, and so forth. A processing block has one entrance and one exit.

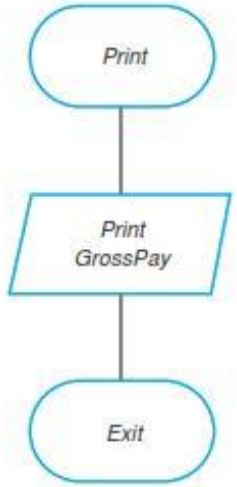
 I/O	The parallelogram indicates input to and output from the computer memory. An input/output (I/O) block has one entrance and only one exit.
 Decision	The diamond indicates a decision. It has one entrance and two and only two exits from the block. One exit is the action when the resultant is *True* and the other exit is the action when the resultant is *False*.

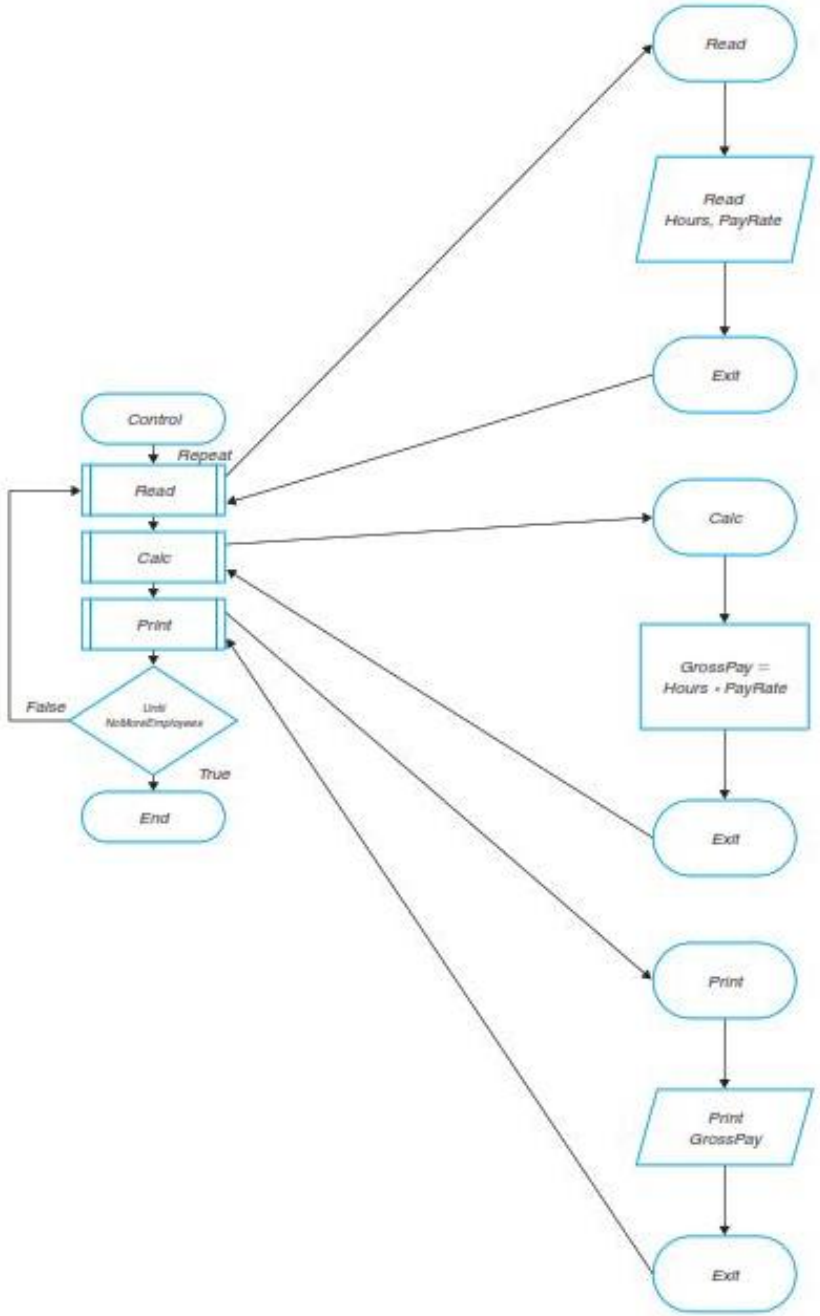
Flowchart Symbol	Explanation
 Process Module	Rectangles with lines down each side indicate the process of modules.They have one entrance and only one exit.
 Automatic-Counter Loop	The polygon indicates a loop with a counter. The counter starts with *A* (the beginning value) and is incremented by *S* (the incrementor value) until the counter is greater than *B* (the ending value). *Counter* is a variable. *A*, *B*, and *S* may be constants, variables, or expressions.
 On-Page Connectors* Off-Page Connectors*	Flowchart sections can be connected with two different symbols. The circle connects sections on the same page, and the home base plate connects flowcharts from page to page. Inside these two symbols the programmer writes letters or numbers. The on-page connector uses letters inside the circle to indicate where the adjoining connector is located. An *A* connects to an *A*, a *B* to a *B*, etc. The off-page connectors use the page number where the next part or the previous part of the flowchart is located. This allows the reader to easily follow the flowchart. On- and off-page connectors will have either an entrance or an exit.

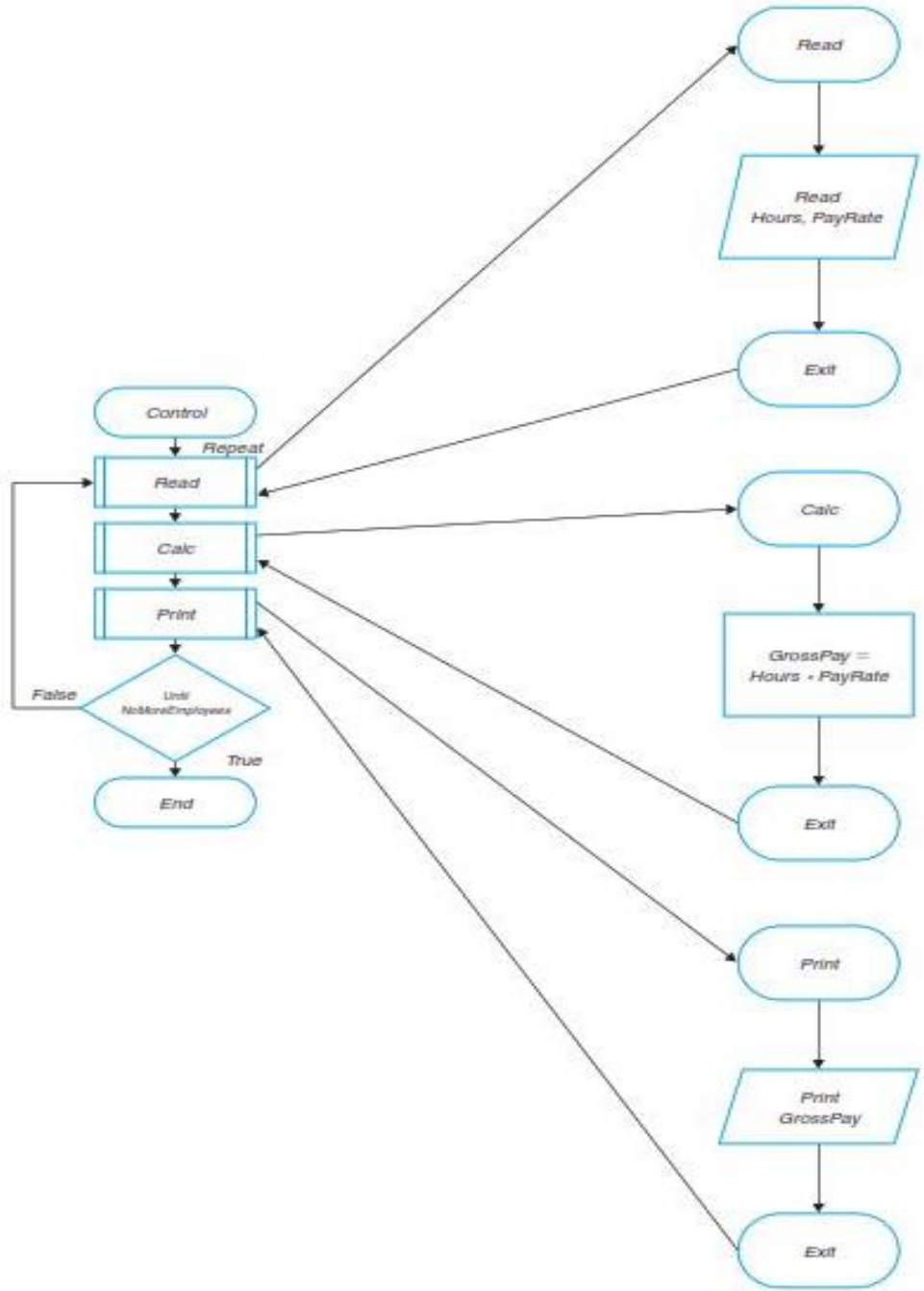
Algorithm	Flowchart	Pseudocode
Control Module 1. *Repeat* *Process Read* *Process Calc* *Process Print* *Until* *NoMoreEmployees* 2. *End*		*Repeat* *Process Read* *Process Calc* *Process Print* *Until* *NoMoreEmployees* *End*

Algorithm	Flowchart	Pseudocode
Read Module 1. *Read Hours, PayRate* 2. *Exit*		*Read Hours, PayRate* *Exit*

Algorithm	Flowchart	Pseudocode
Calc Module 1. *GrossPay = HoursWorked * PayRate* 2. *Exit*		*GrossPay = Hours * PayRate* *Exit*

Algorithm	Flowchart	Pseudocode
Print Module 1. *Print Pay* 2. *Exit*		*Print Pay* *Exit*




Read
Read
Hours, PayRate
Exit
Control
Repeat
Read
Calc
Print
Until
NoMoreEmployees
False
True
End
Calc
GrossPay =
Hours * PayRate
Exit
Print
Print
GrossPay
Exit

RULES FOR DRAWING FLOWCHART

1. You should write the instructions inside the blocks.

2. If there is something you need to remember, you can write a note beside a block. Test values of variables also can be placed beside flowchart blocks. This makes the flowchart an annotated flowchart.

3. A flowchart always starts at the top of the page and flows to the bottom. If you need more than one page for a flowchart, start another column on the same page or go on to another page. On- and off-page connectors are used to connect parts of the flowchart. A flowchart should not flow up, or sideways, or all over the page.

4. Use a computer program, or a template and a straightedge, to draw the flowchart.

When you use a computer program or a template, the symbols are the same size and your flowchart will be neater and easier to read.

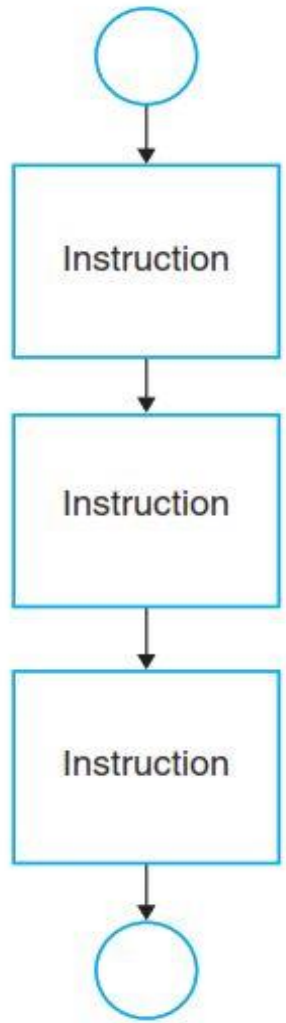
5.Make the blocks big enough to write instructions so they can be easily read.

6. Put the module number and name from the interactivity chart in the upper

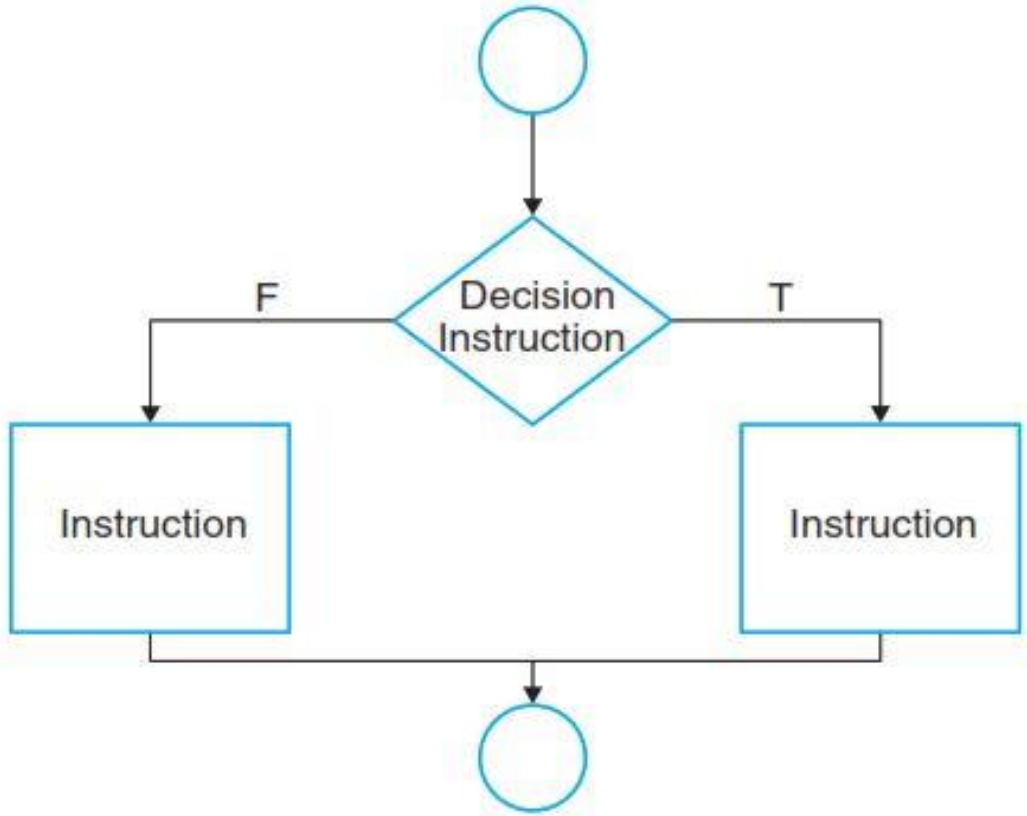
right-hand corner of the page for quick reference to the correct module.

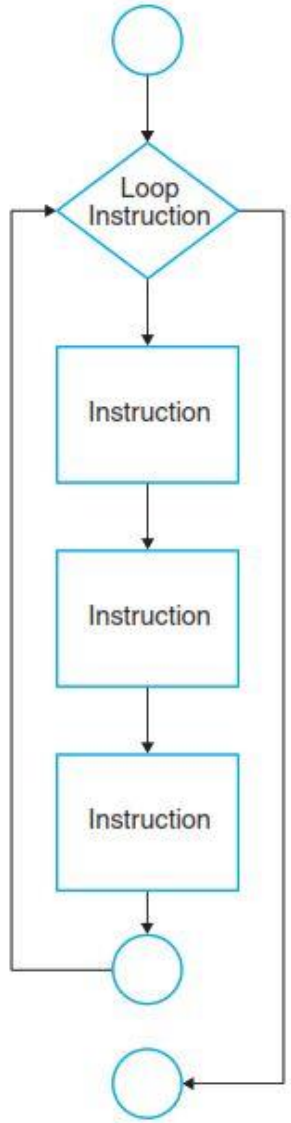
7. Have plenty of paper on hand since the final copy of the flowchart normally will not be the first draft.

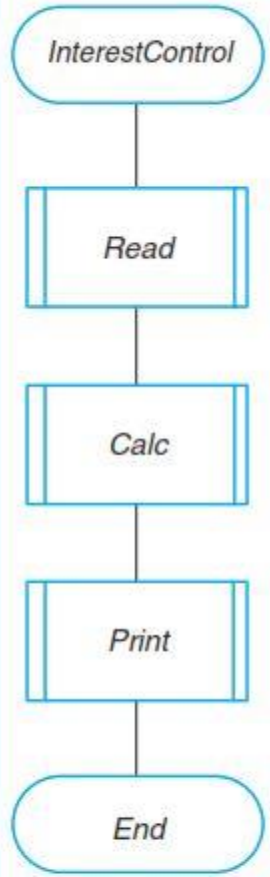
8. Use a pencil with a large eraser.

Algorithm	Flowchart	Pseudocode
⋮ 5. Instruction 6. Instruction 7. Instruction 8. ⋮		⋮ Instruction Instruction Instruction ⋮

Algorithm	Flowchart	Pseudocode

⋮
 	
5. If <decision>
then
Instruction
else
Instruction
6. ⋮
		⋮
		
		If <decision>
		then
		Instruction
		else
		Instruction
		Endif
		⋮

Algorithm	Flowchart	Pseudocode
⋮		⋮
5. Loop		Loop
Instruction Instruction Instruction		Instruction Instruction Instruction
Until <logical expression>		Until <logical expression>
6. ⋮		⋮

Algorithm	Flowchart	Pseudocode	Annotation	Test
InterestControl 1. *Process Read (*Principal, *Interest *Years, *Time)* 2. *Process Calc (Principal, Interest, Years, Time, *Amount)* 3. *Process Print (Principal, Interest, Years, Time, Amount)* 4. *End* * Specifies call-by-reference parameters.		*Process Read (*Principal, *Interest *Years, *Time)* *Process Calc (Principal, Interest, Years, Time, *Amount)* *Process Print (Principal, Interest, Years, Time, Amount)* *End*	Enters all data from keyboard Calculates amount Prints data and amount	1. Start 2. Transfer to *Read* 3. Transfer to *Calc* 4. Transfer to *Print*

Internal Documentation	External Documentation
1. Remark at top: Calculates principal and interest given, beginning principal, interest rate, number of years, and compounded time interval. 2. Include annotations	1. Same as 1 in Internal Documentation

SUMMARY

1.The problem analysis chart—helps you define and understand the problem, develop ideas for the solution, and select the best solution.

2. The interactivity chart—breaks the solution to the problem into parts.

3. The IPO chart—helps define the input, the output, and the processing steps.

4. The coupling diagram and the data dictionary designate the data flow between modules. The data dictionary records information on the items.

5. The algorithms define the steps of the solution.

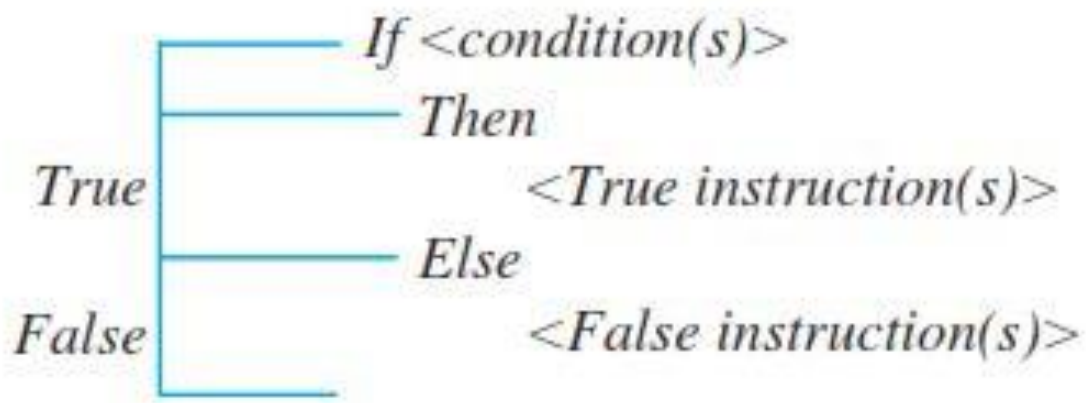
6. The flowcharts are a graphic form of the algorithms.

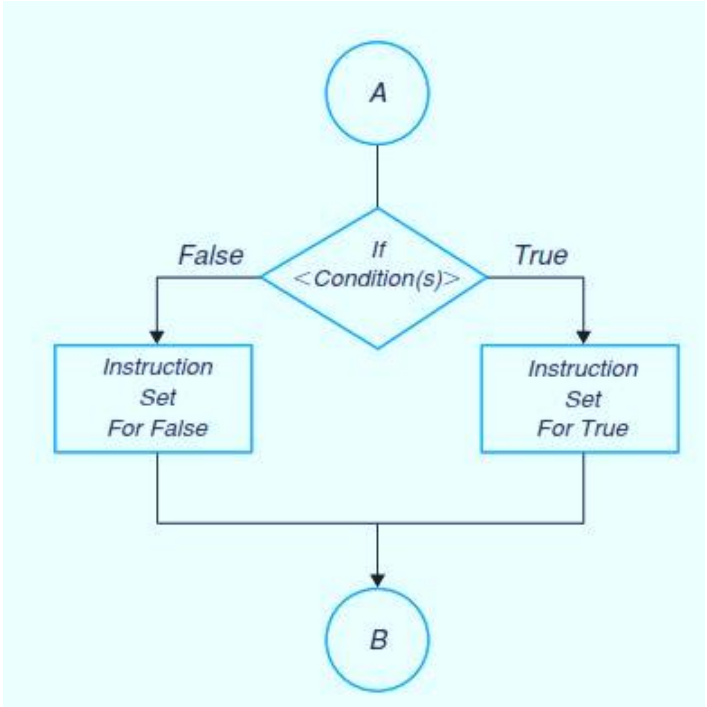
7. The pseudocode presents a generic language representation of the algorithm.

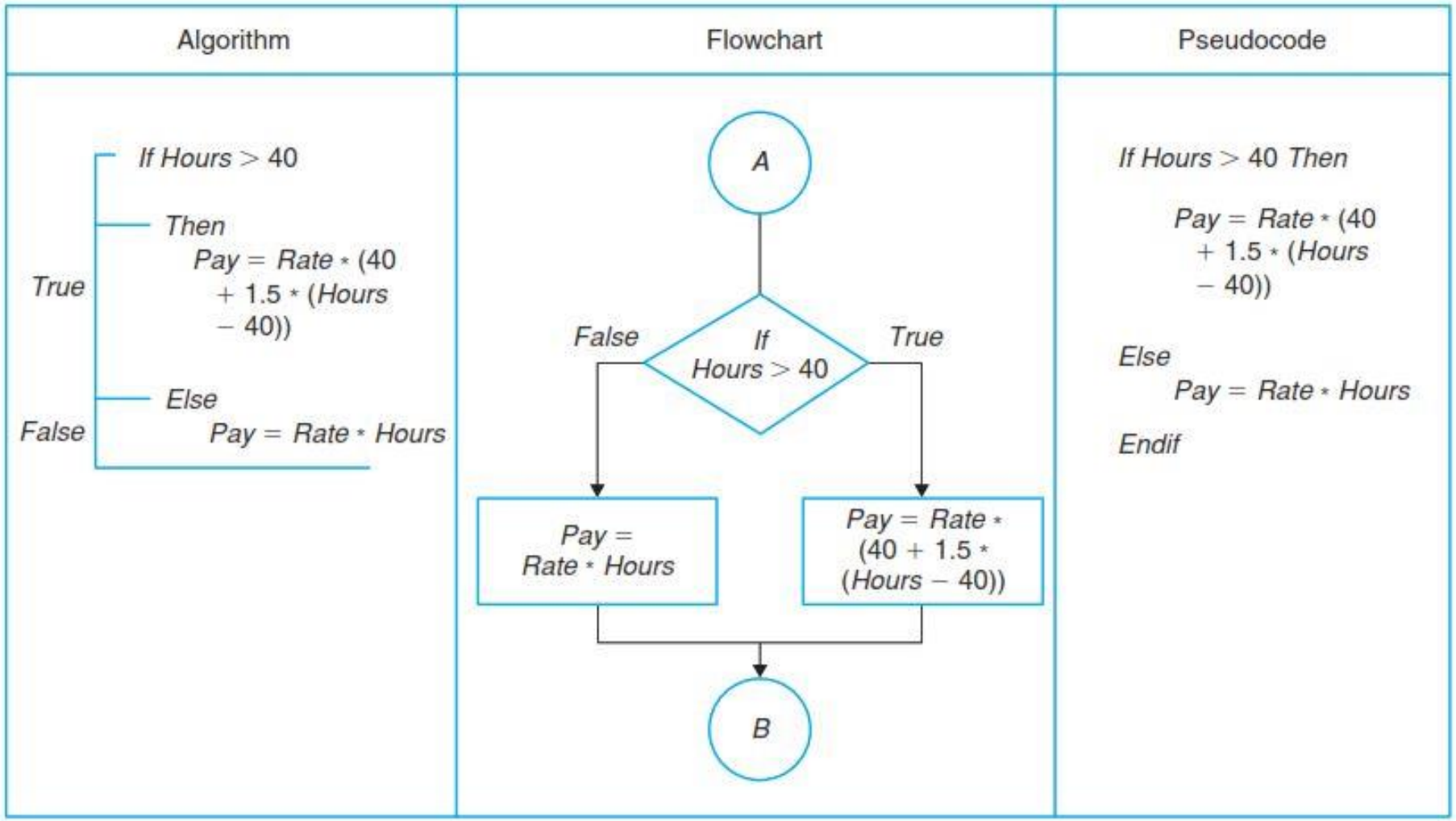
Decision Logical Structures

The decision logic structure uses the **If/Then/Else** instruction. It tells the computer that If a condition is **true**, Then execute a set of instructions, or Else execute another set of instructions.

The Else part is optional, as there is not always a set of instructions if the conditions are false.






Algorithm
Flowchart
Pseudocode
If Hours > 40
Then
Pay = Rate * (40 + 1.5 * (Hours − 40))
True
Else
Pay = Rate * Hours
False
A
If Hours > 40
False
True
Pay = Rate * Hours
Pay = Rate * (40 + 1.5 * (Hours − 40))
B
If Hours > 40 Then
Pay = Rate * (40 + 1.5 * (Hours − 40))
Else
Pay = Rate * Hours
Endif

Multiple IF/ Else Statements

Straight-through logic means that all of the decisions are processed sequentially, one after the other. There is no Else part of the instructions; the False branch always goes to the next decision, and the True branch goes to the next decision after the instructions for the True branch have been processed.

Positive logic allows the flow of the processing to continue through the module instead of processing succeeding decisions, once the resultant of a decision is True. Whenever the resultant is False (the Else part of the decision), another decision in the sequence is processed until the resultant is True, or there are no more decisions to process. At that time, the False branch processes the remaining instructions.

Negative logic is similar to positive logic except that the flow of the processing continues through the module when the resultant of a decision is False. Whenever the resultant is True, another decision is processed until the resultant is False, or there are no more decisions to process.

Some series of decisions will require a combination of two or more of these logic types.

Planning your Solution (contd..)

d) - Drawing the Program flowcharts (Contd..)

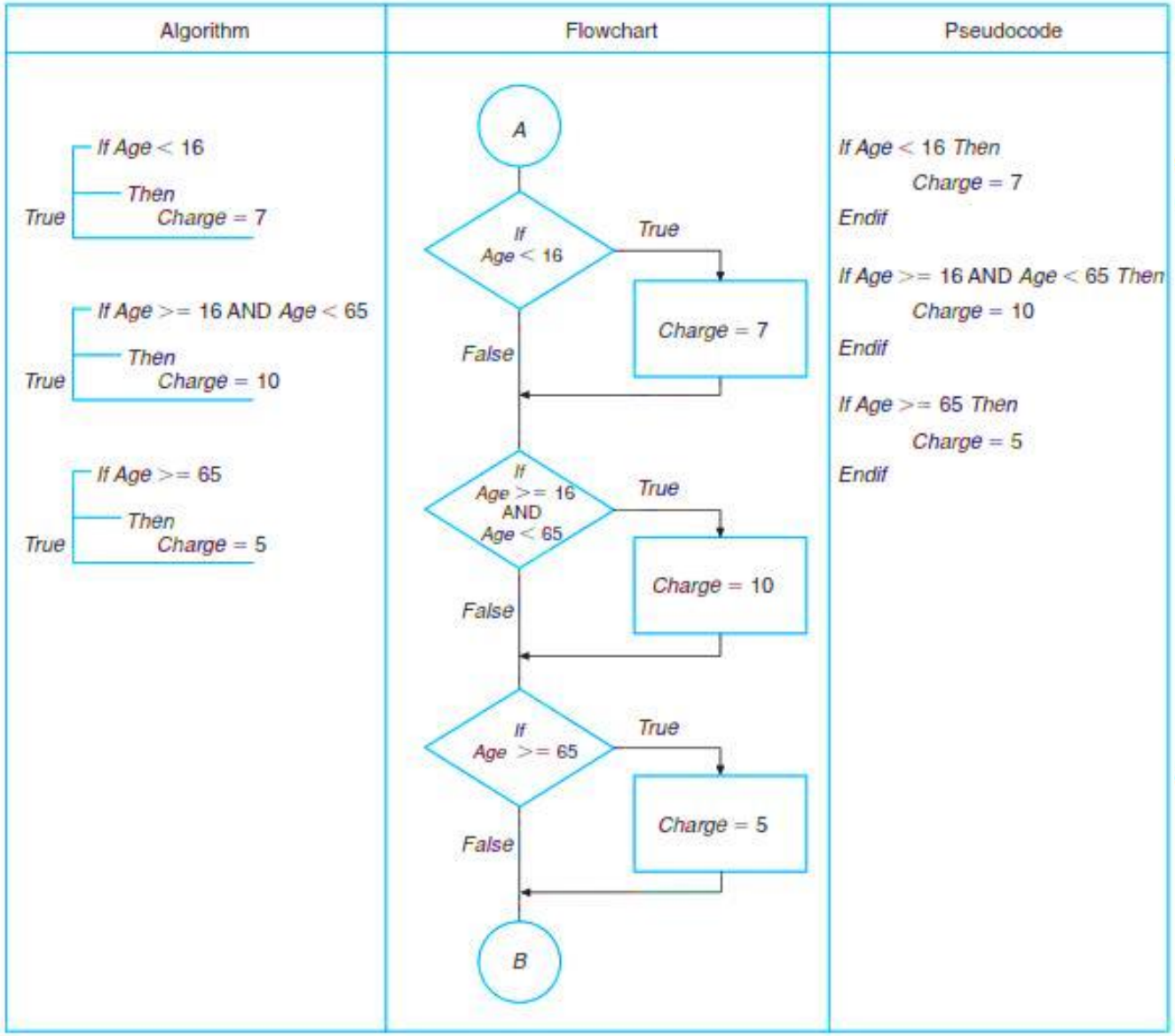
Example: Sale Problem

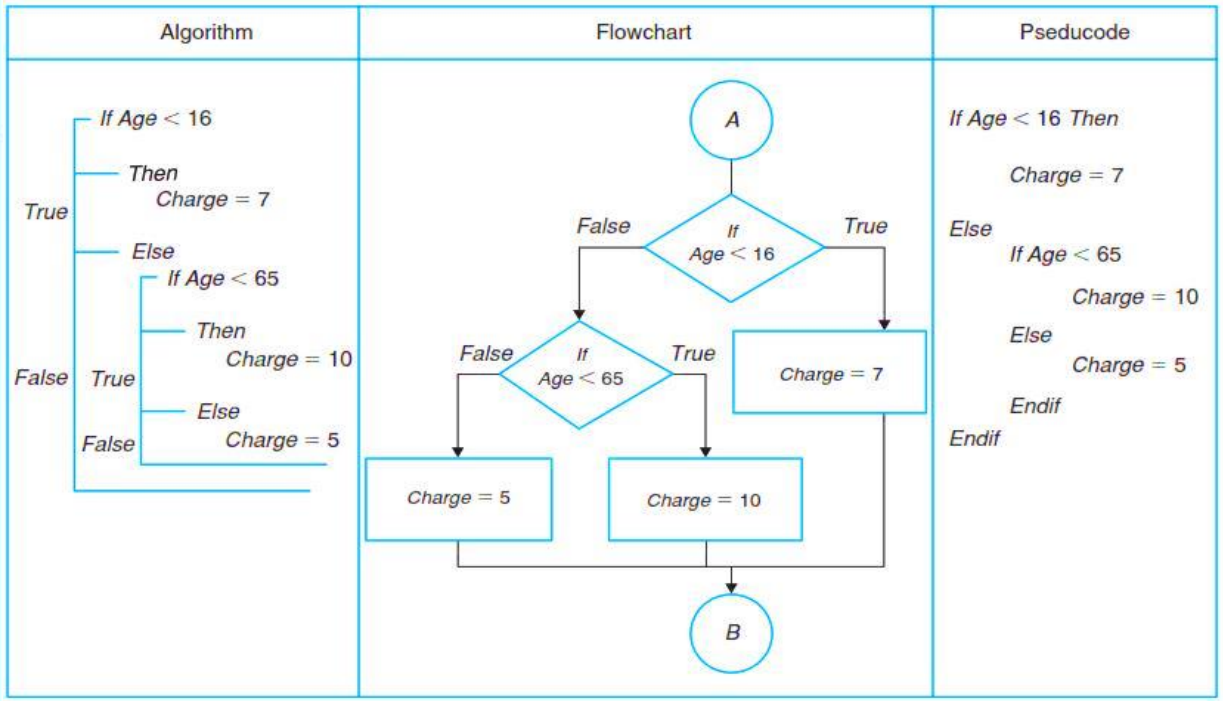
Draw a flowchart for a problem that to read two numbers. The first number represents the unit price of a product and the second number represents the quantity of the product sold. Calculate and print the total sale.

Solution: Stepwise Analysis of the Sale Problem

- Start of processing
- Read the unit price
- Read the quantity
- Calculate total sale
- Print total sale

Straight-through Logic





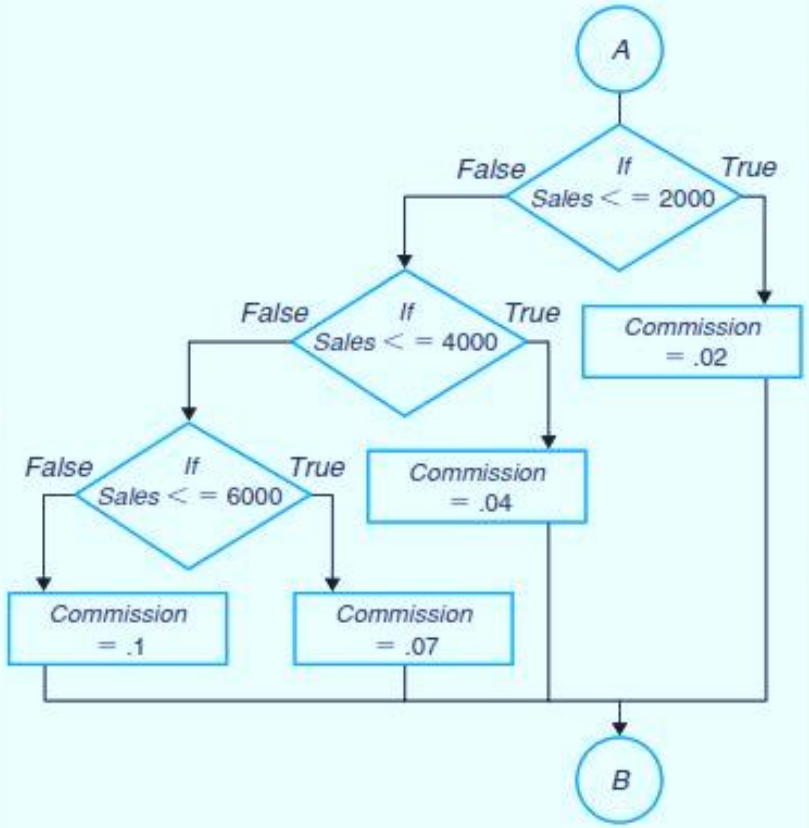
Positive Logic

Positive Logic

Algorithm

If Sales < = 2000
Then
Commission = .02
True
Else
If Sales < = 4000
Then
Commission = .04
True
Else
If Sales < = 6000
Then
Commission = .07
True
False
Else
Commission = .1
False
False

Flowchart



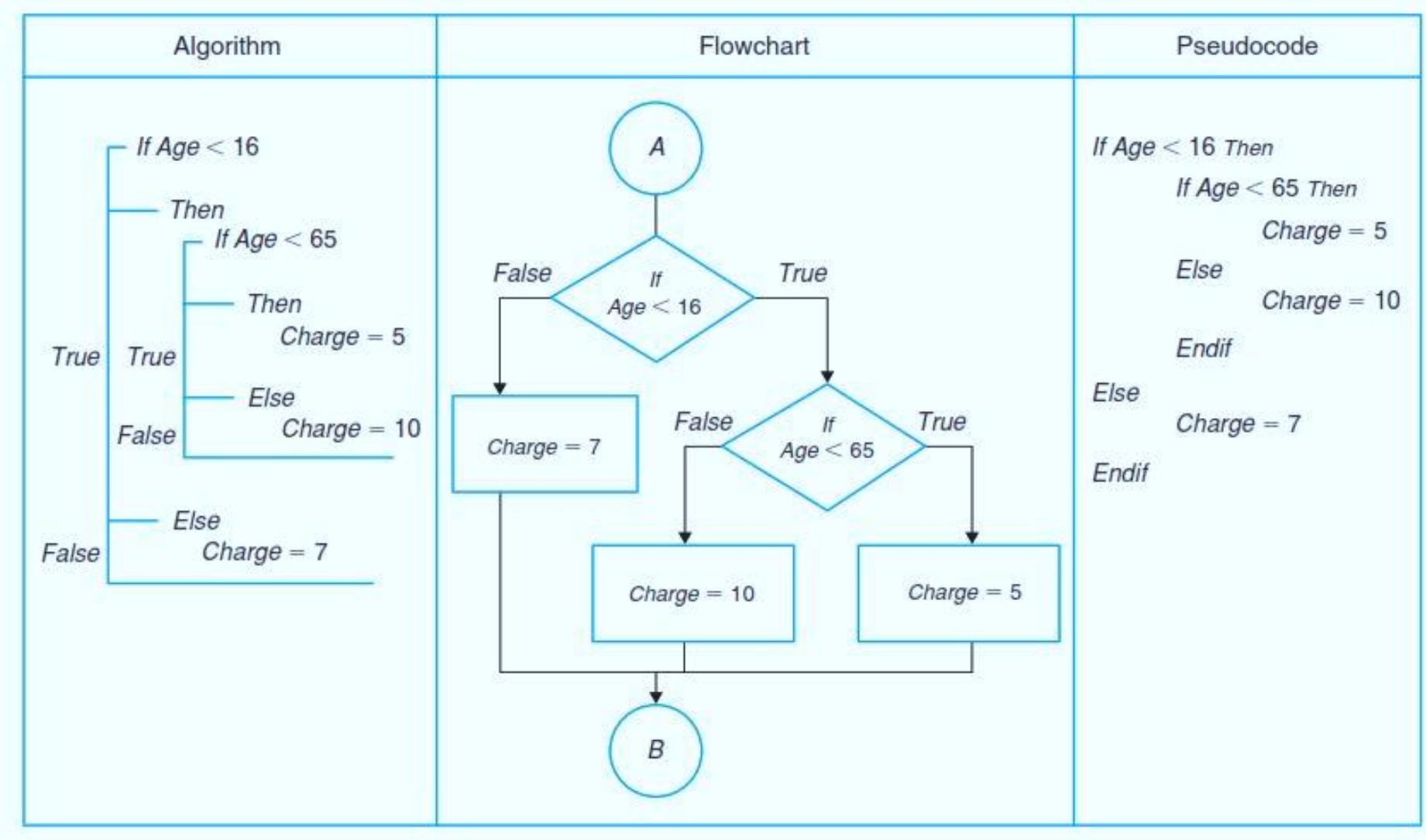
Pseudocode

```
If Sales < = 2000 Then
        Commission = .02
Else
        If Sales <4000 Then
                Commission = .04
        Else
                If Sales <6000 Then
                        Commission = .07
                Else
                        Commission = .1

                Endif
        Endif
Endif
```

Test

1. Test For
   Sales = 1500
   Is Sales < = 2000?
   True
   Commission = .02
2. Test For
   Sales = 3500
   Is Sales < = 2000?
   False
   Is Sales < = 4000?
   True
   Commission = .04
3. Test For
   Sales = 5500
   Is Sales < = 2000?
   False
   Is Sales < = 4000?
   False
   Is Sales < = 6000?
   True
   Commission = .07
4. Test For
   Sales = 7500
   Is Sales < = 2000?
   False
   Is Sales < = 4000?
   False
   Is Sales < = 6000?
   False
   Commission = .1

?? Logic


Algorithm
If Age < 16
Then
If Age < 65
Then
Charge = 5
True
True
Else
Charge = 10
False
Else
False
Charge = 7
Flowchart
A
False
If
Age < 16
True
Charge = 7
False
If
Age < 65
True
Charge = 10
Charge = 5
B
Pseudocode
If Age < 16 Then
If Age < 65 Then
Charge = 5
Else
Charge = 10
Endif
Else
Charge = 7
Endif

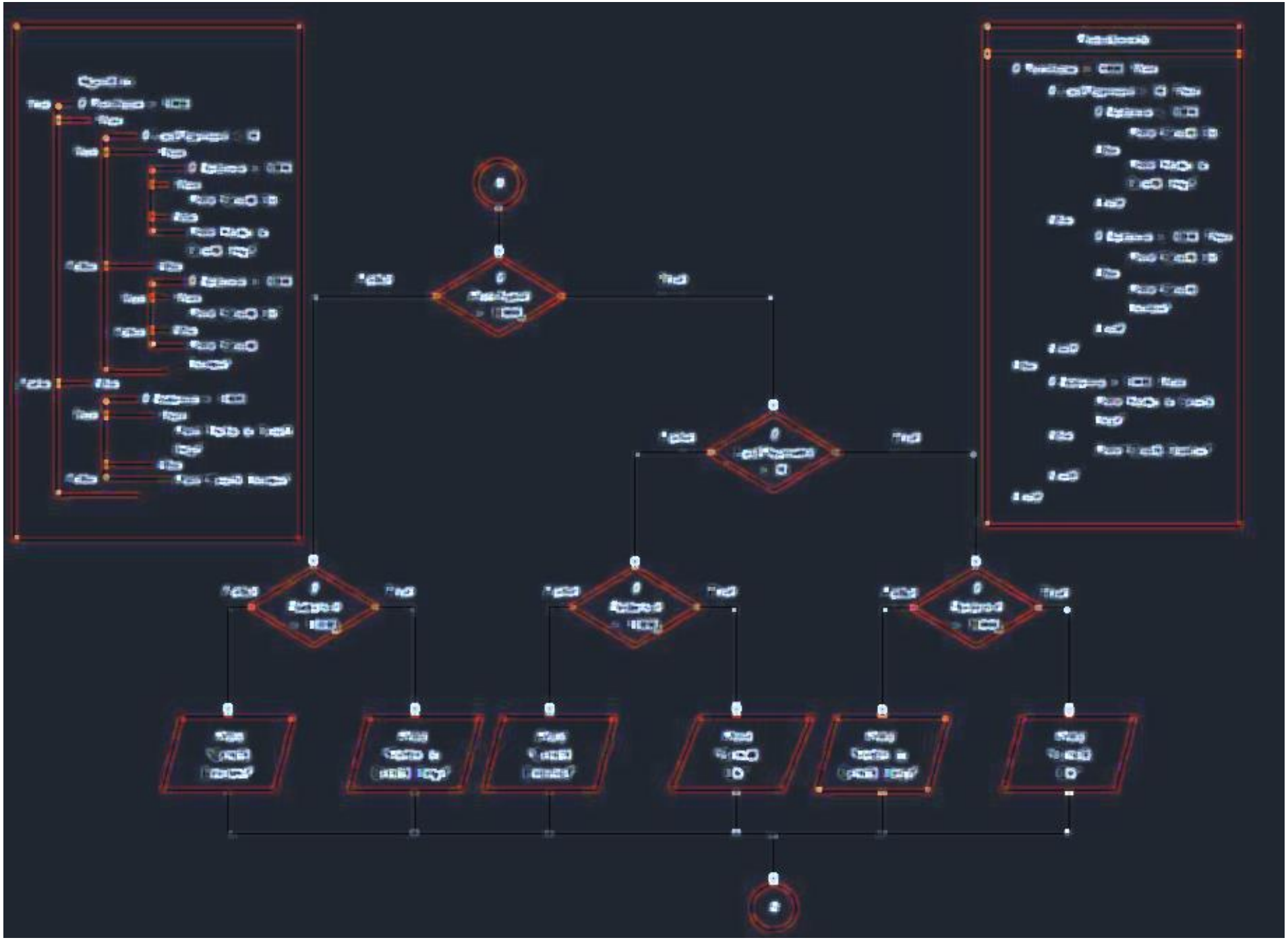
A decision table for a store policy for charging a purchase. There are three conditions:

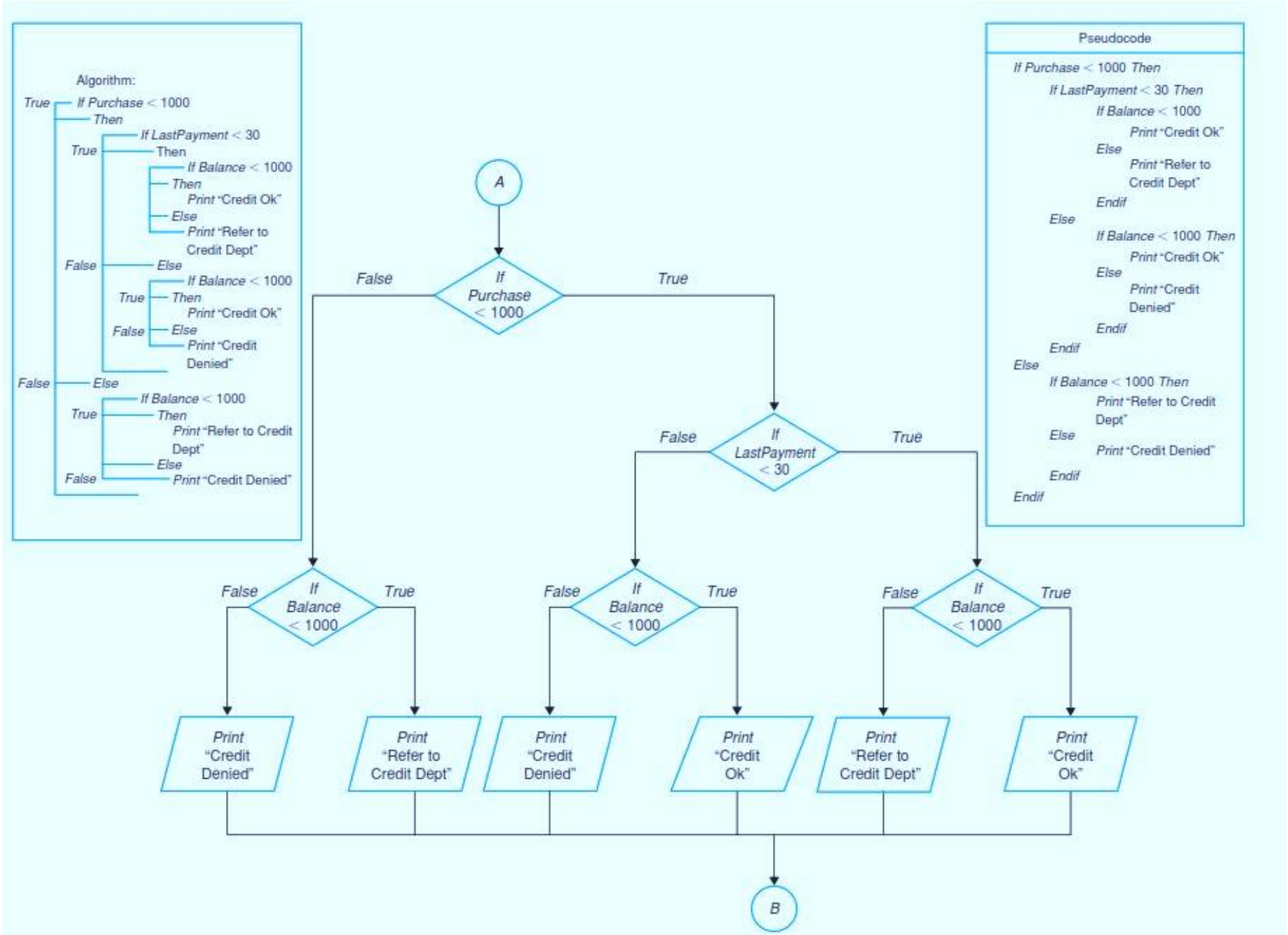
1. The purchase is less than $100.
2. The last payment to the account was made in the last 30 days.
3. The balance of the account is less than $1,000.

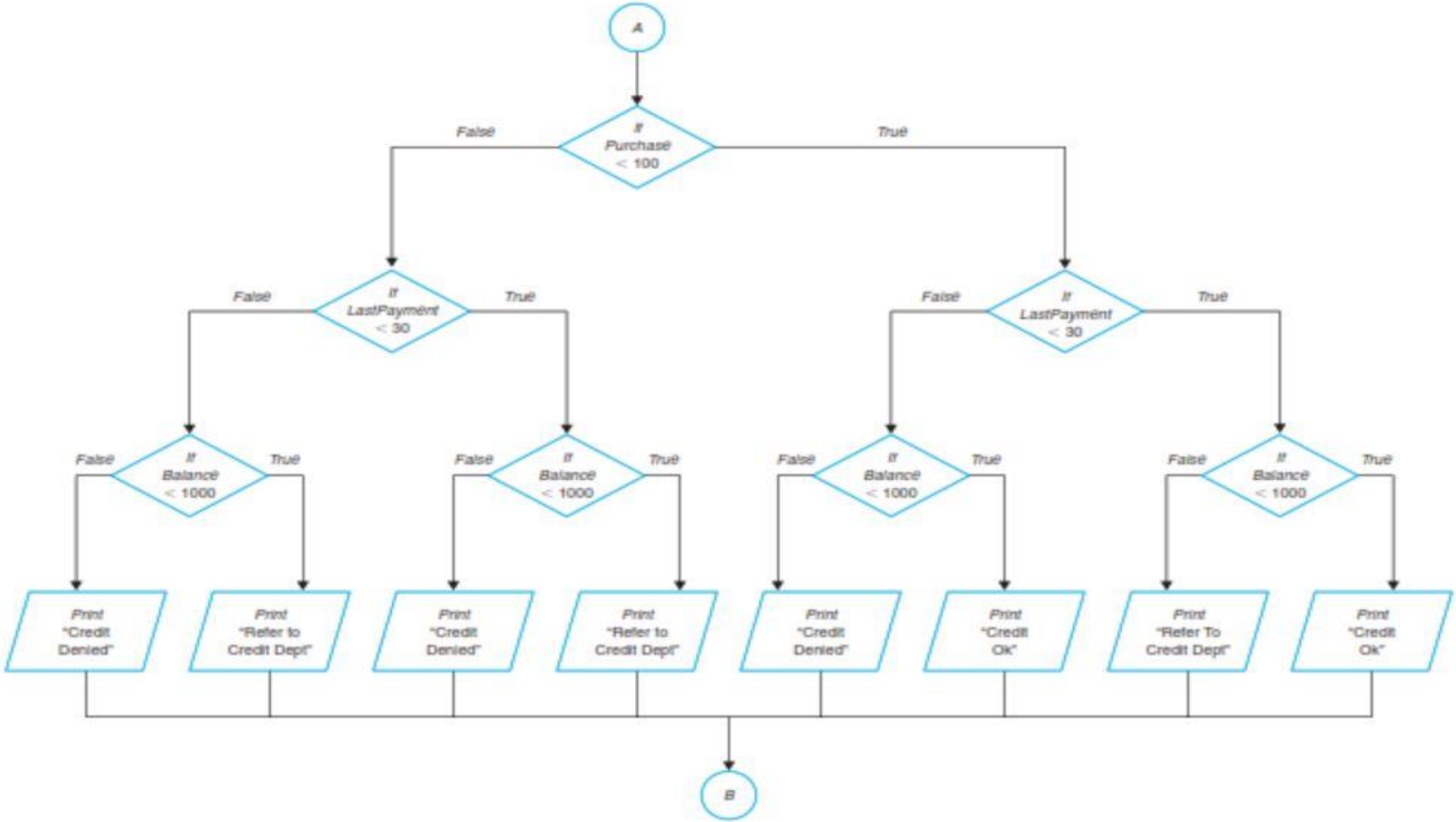
List of Conditions	1	*Purchase < $100*	*True*	*True*	*True*	*True*	*False*	*False*	*False*	*False*
	2	*LastPaymentWithin30Days*	*True*	*True*	*False*	*False*	*True*	*True*	*False*	*False*
	3	*Balance < $1000*	*True*	*False*	*True*	*False*	*True*	*False*	*True*	*False*
List of Actions	1	*CreditOk*	X		X					
	2	*ReferToCreditDepartment*		X			X		X	
	3	*CreditDenied*				X		X		X

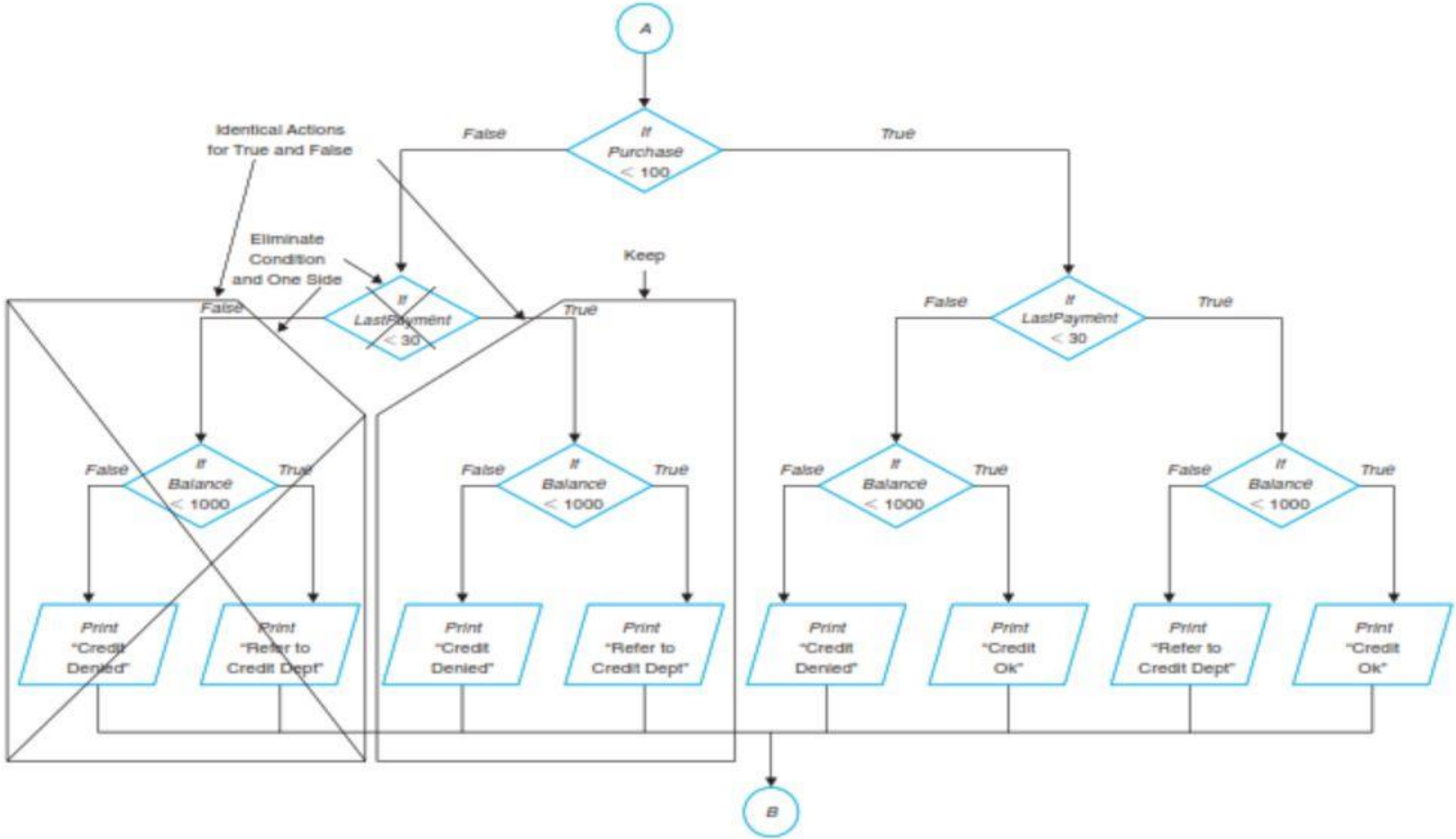
Each of these three conditions could be True or False depending on the customer. The following actions could be taken:

1. Credit is okay, and the customer can charge the item.
2. Refer the customer to the credit department.
3. Credit is denied and the customer cannot charge the item.




Algorithm:
True If Purchase < 1000
Then
If LastPayment < 30
True Then
If Balance < 1000
Then
Print "Credit Ok"
Else
Print "Refer to Credit Dept"
False Else
If Balance < 1000
True Then
Print "Credit Ok"
False Else
Print "Credit Denied"
False Else
If Balance < 1000
True Then
Print "Refer to Credit Dept"
Else
False Print "Credit Denied"
Pseudocode
If Purchase < 1000 Then
If LastPayment < 30 Then
If Balance < 1000
Print "Credit Ok"
Else
Print "Refer to Credit Dept"
Endif
Else
If Balance < 1000 Then
Print "Credit Ok"
Else
Print "Credit Denied"
Endif
Endif
Else
If Balance < 1000 Then
Print "Refer to Credit Dept"
Else
Print "Credit Denied"
Endif
Endif
A
If Purchase < 1000
False
True
If LastPayment < 30
False
True
If Balance < 1000
False
True
If Balance < 1000
False
True
If Balance < 1000
False
True
Print "Credit Denied"
Print "Refer to Credit Dept"
Print "Credit Denied"
Print "Credit Ok"
Print "Refer to Credit Dept"
Print "Credit Ok"
B


A
If Purchase < 100
False
True
If LastPayment < 30
False
True
If LastPayment < 30
False
True
If Balance < 1000
False
True
If Balance < 1000
False
True
If Balance < 1000
False
True
If Balance < 1000
False
True
Print "Credit Denied"
Print "Refer to Credit Dept"
Print "Credit Denied"
Print "Refer to Credit Dept"
Print "Credit Denied"
Print "Credit Ok"
Print "Refer To Credit Dept"
Print "Credit Ok"
B


A
If Purchase < 100
False
True
Identical Actions for True and False
Eliminate Condition and One Side
Keep
If LastPayment < 30
False
True
If Balance < 1000
False
True
Print "Credit Denied"
Print "Refer to Credit Dept"
Print "Credit Ok"
B

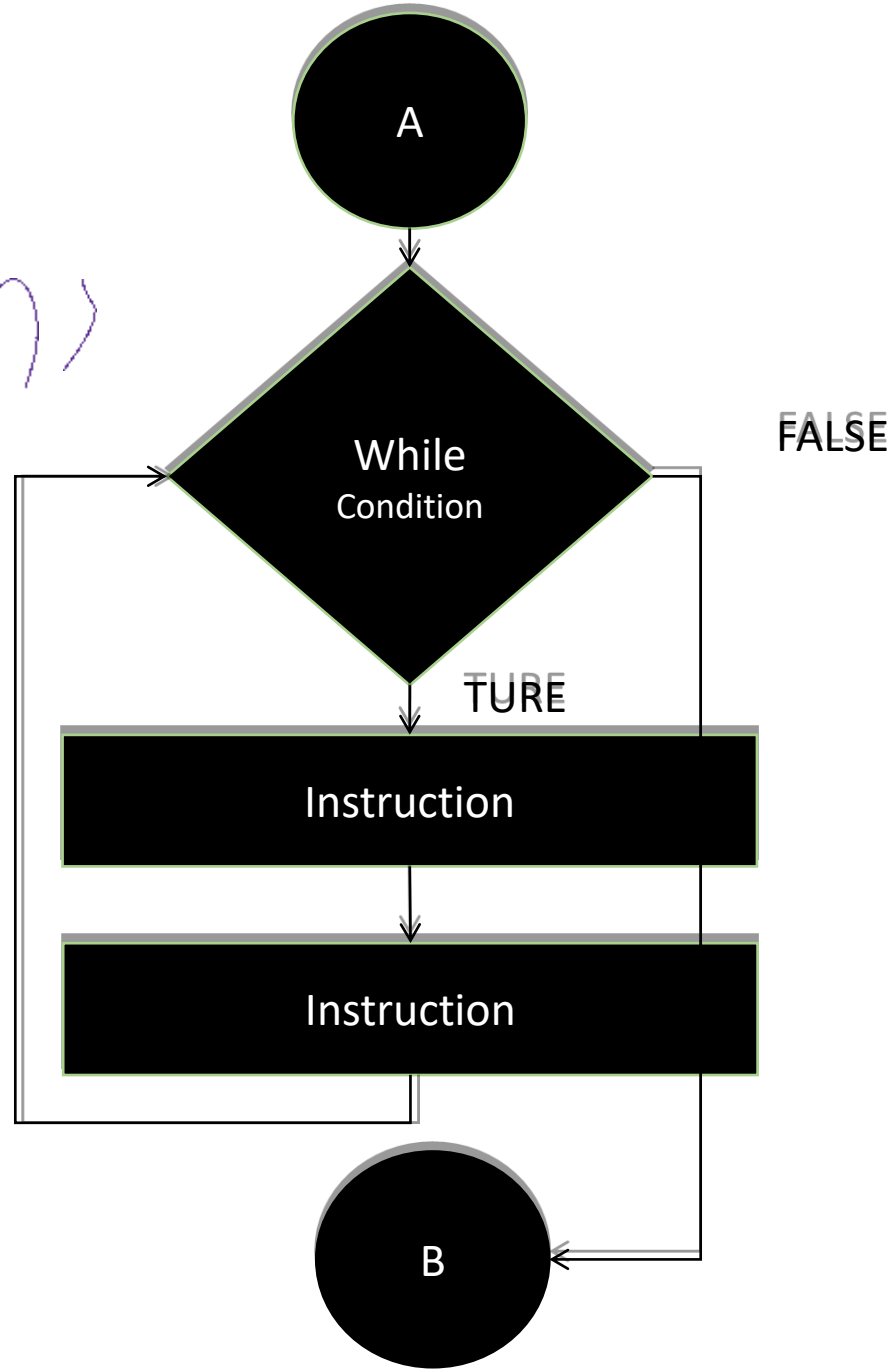
Logical Structures

Logical structures

- Sequential
- Decision
- Loop

WHILE / WHILE END

While <condition>
[Instruction
:
While End



Example Problem:-

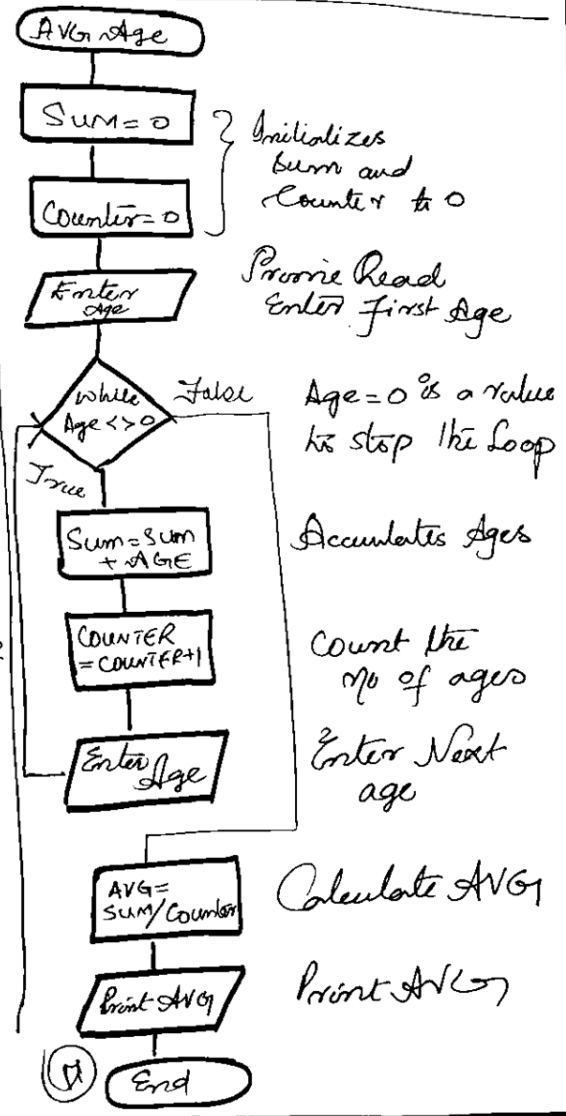
Develop the algorithm and the flow chart to find the average age of all the students in a class.

Algorithm

Average Age

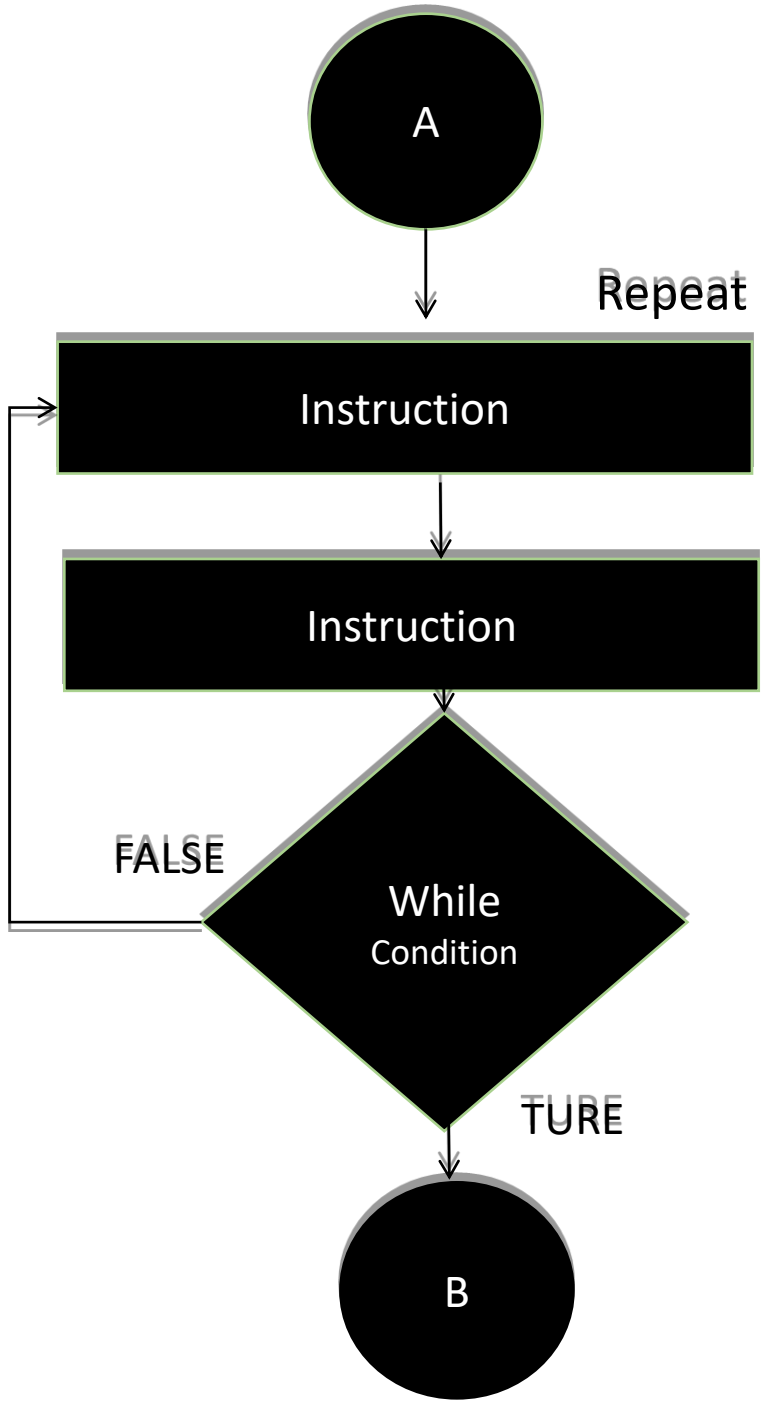
1 SUM = 0
2 COUNTER = 0
3 ENTER AGE
4 WHILE AGE <> 0
  SUM = SUM + AGE
  COUNTER = COUNTER + 1
  ENTER AGE
  WHILE END
5 AVERAGE = SUM / COUNTER
6 PRINT AVERAGE
7 END

Flow chart



Repeat Until

Repeat
[Instruction
Instruction
Until (cond)]



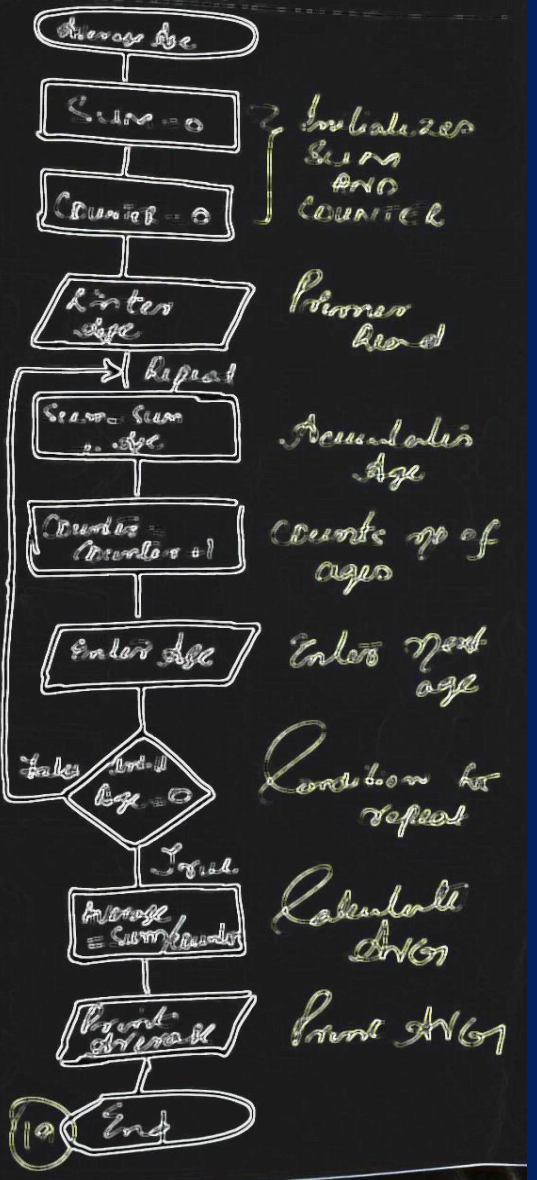
Problem: Average of a class using Repeat Until

Algorithm

AVERAGE AGE

1 SUM = 0
2 COUNTER = 0
3 ENTER AGE
4 Repeat
  SUM = SUM + Age
  Counter = Counter + 1
  Enter Age
  Until Age = 0
5 Average = Sum / Counter
6 Print Average
7 End

Flowchart



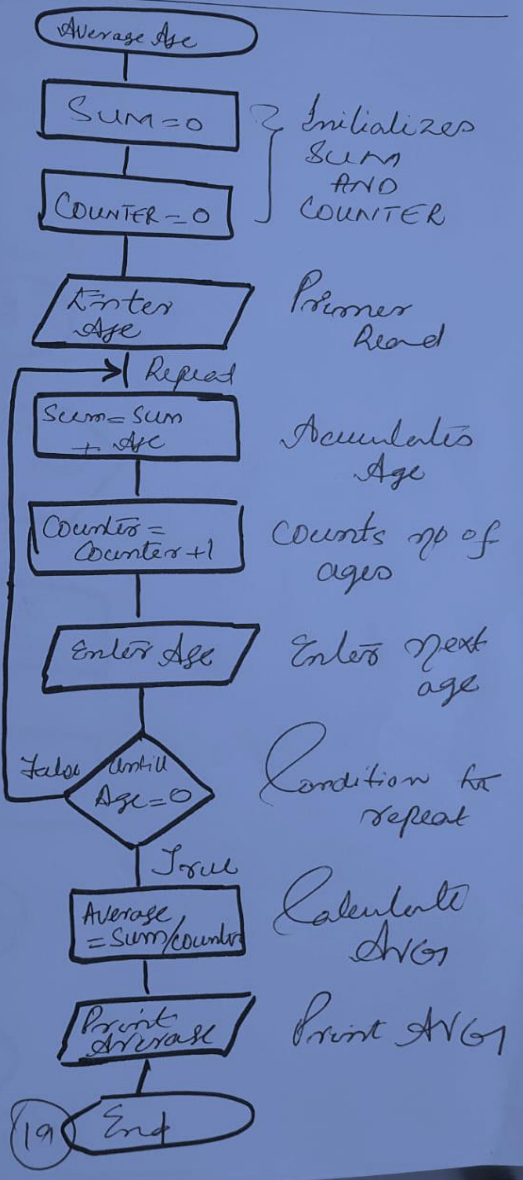
Problem: Average of a class using Repeat Until

Algorithm

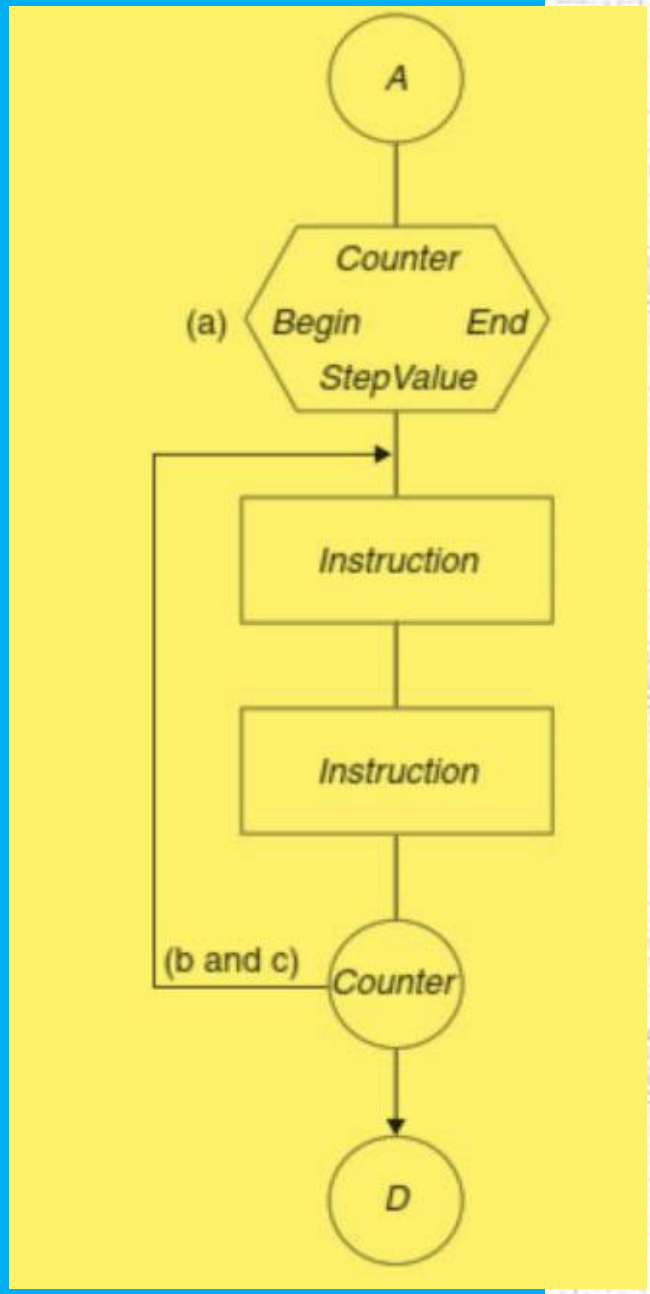
AVERAGE AGE

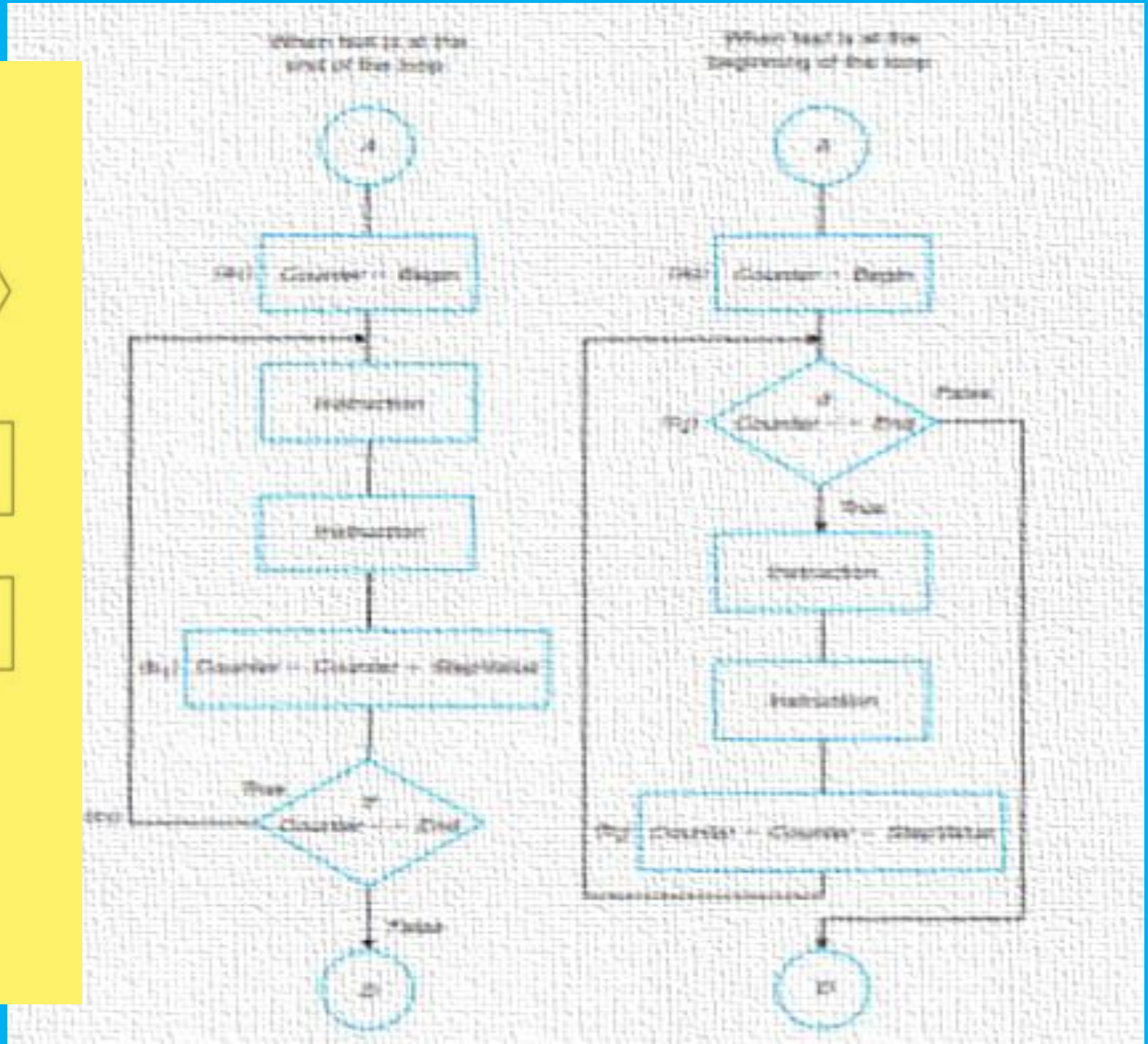
1 SUM = 0
2 COUNTER = 0
3 ENTER AGE
4 Repeat
  SUM = SUM + Age
  Counter = Counter + 1
  Enter Age
  Until Age = 0
5 Average = Sum / Counter
6 Print Average
7 End

Flowchart

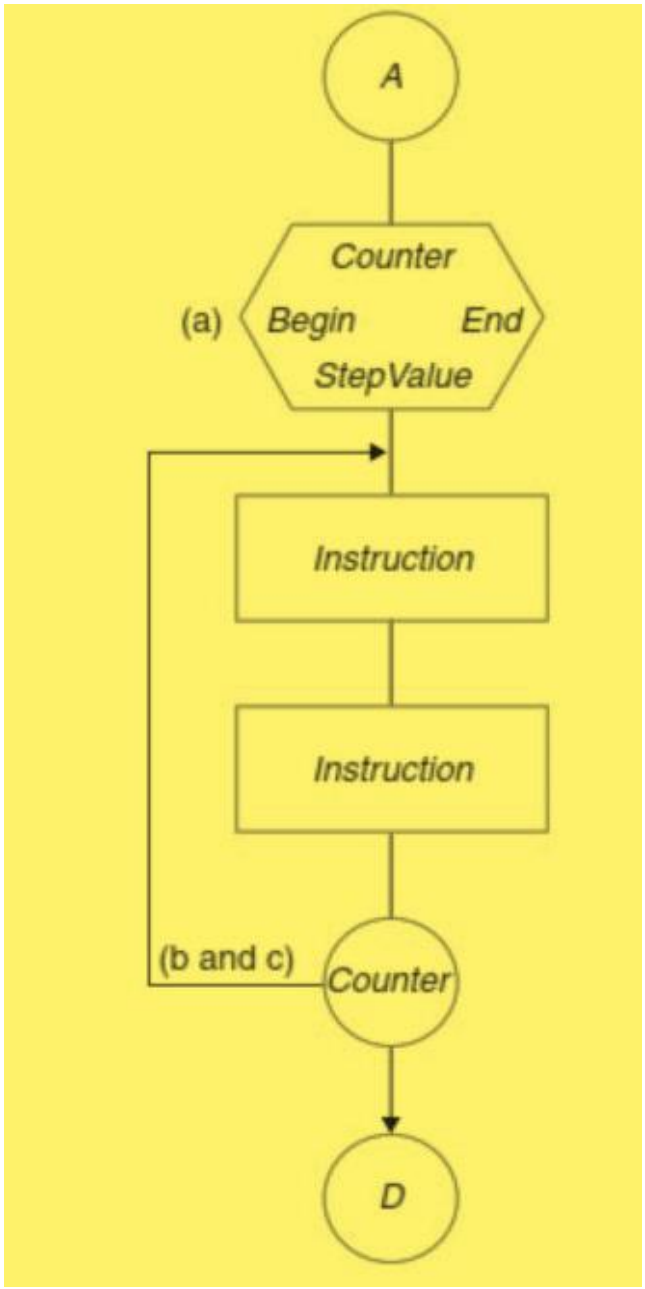


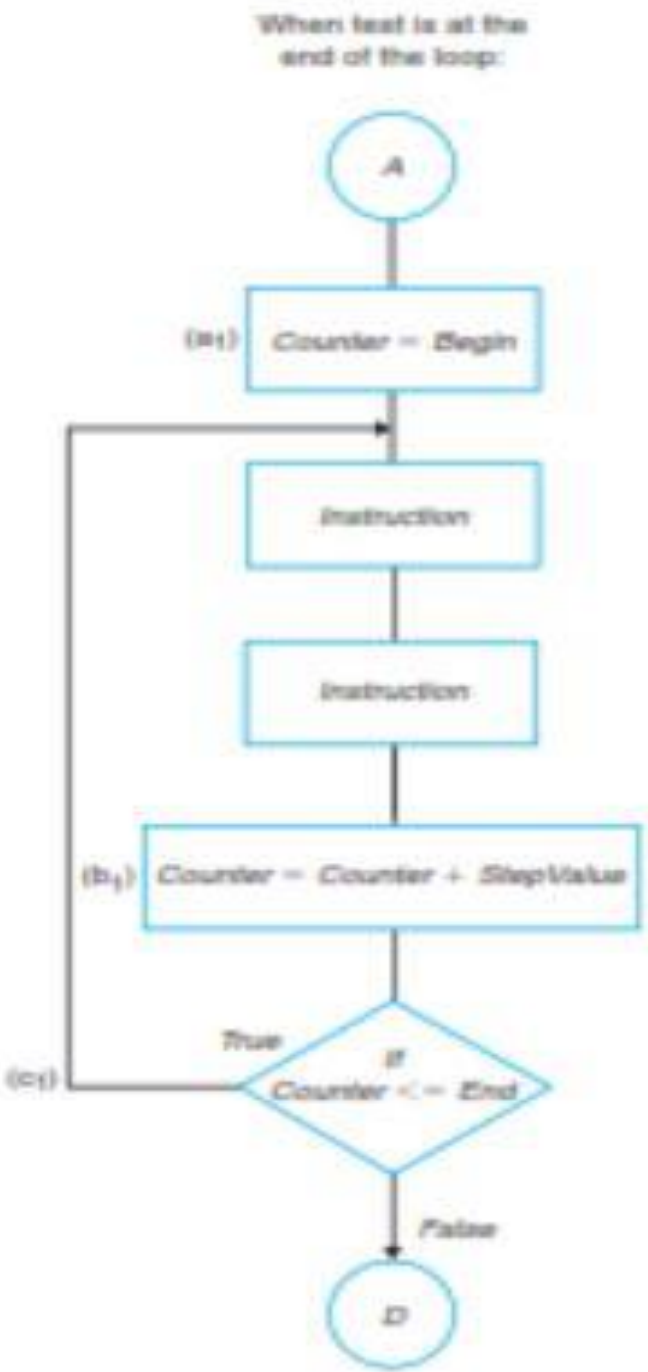
LOOP

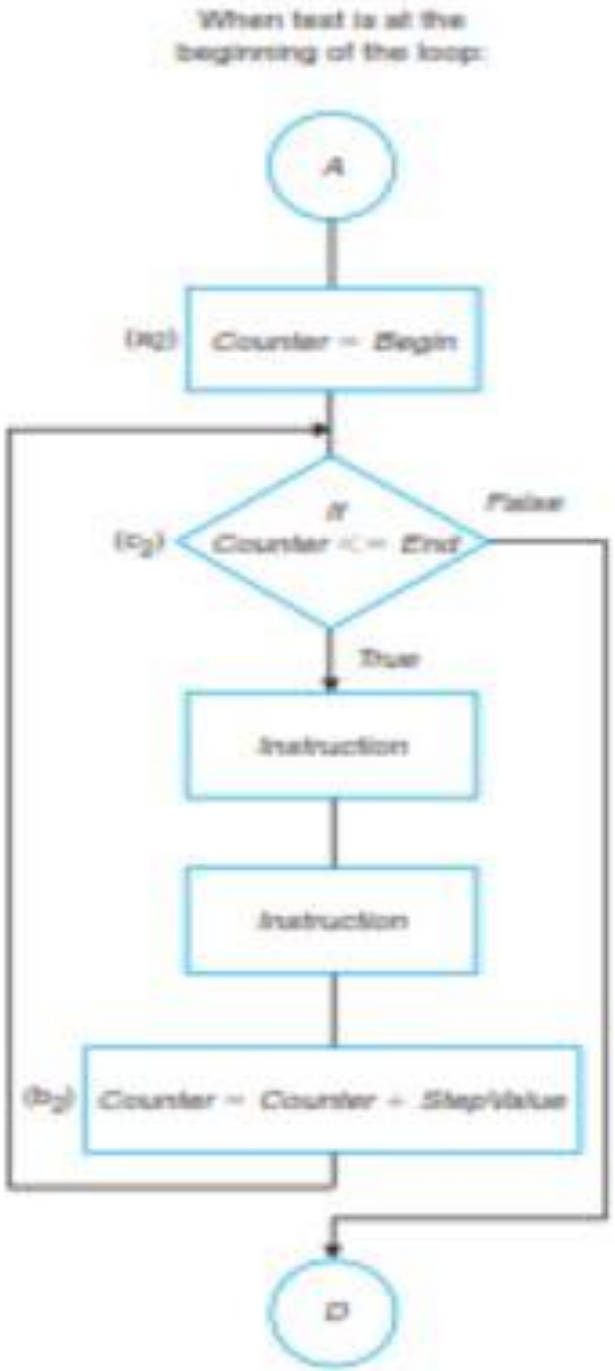

A
Counter
(a)
Begin
End
StepValue
Instruction
Instruction
(b and c)
Counter
D



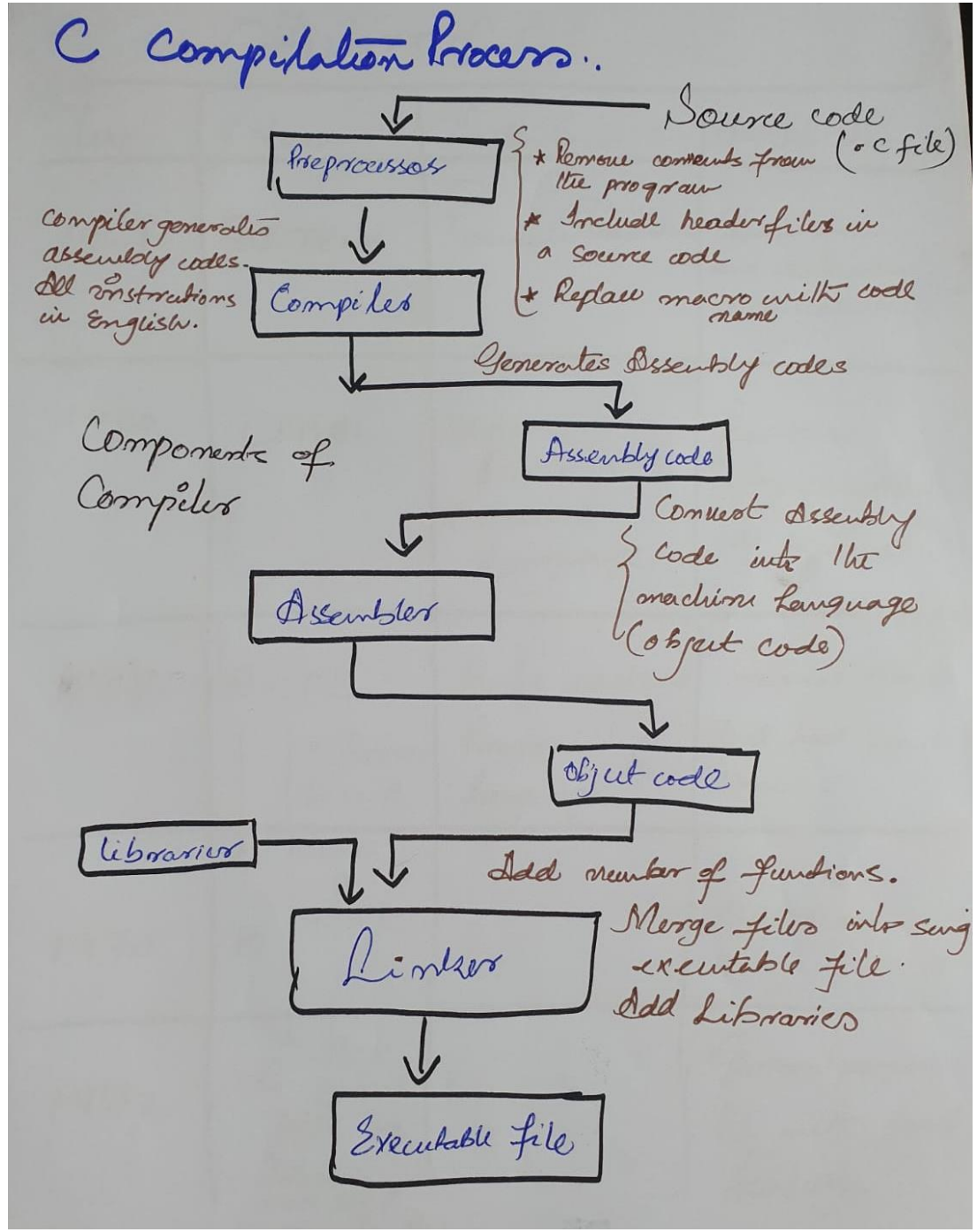
LOOP


A
Counter
(a) Begin End
StepValue
Instruction
Instruction
(b and c)
Counter
D


When test is at the end of the loop:
A
(a1) Counter = Begin
Instruction
Instruction
(b1) Counter = Counter + StepValue
True
(c1) If Counter <= End
False
D


When test is at the beginning of the loop:
A
(a2) Counter = Begin
(c2) If Counter <= End
False
True
Instruction
Instruction
(b2) Counter = Counter + StepValue
D

C Compilation Process..



History of C

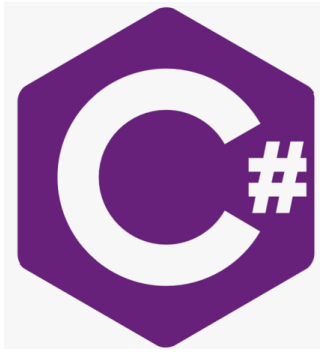
Year	Name	Full-Form	Applications
1957	FORTRAN	Formula Translation	Scientific and Mathematical application
1960	COBOL	Common Business Oriented Language	Business Applications or software
1967	BCPL	Basic combined Programming Language	General purpose but had some issues
1970	B	some issue in BCPL	Still issues
1972	C	Then Dennis took the initiative and solve issues in B.	General purpose PL with good features

M. Thompson solved

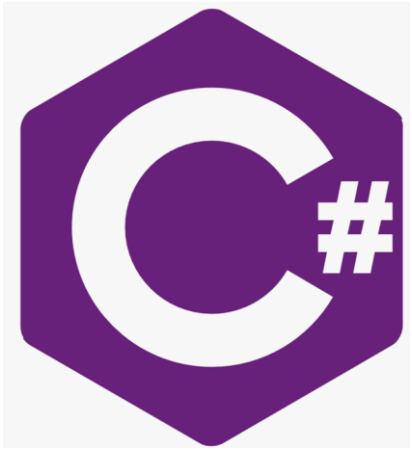
WINDOWS
1.3 Billion Active User

ANDROID

3 Billion Active User Monthly



IOS
1 Billion Active User



GOOGLE

3.5 Billion Requests Per Day

SIMPLE C PROGRAM

```
#include <stdio.h>
int main()
{
    Printf("Welcome to C!\n");
return 0;
}
```

Escape Sequence

Escap Sequence	Descriptions
\n	New Line.
\t	Horizontal tab.
\a	Alert.
\\	Black slash.
\"	Double quote.

The First C Program

```
# include <stdio.h>

int main ()        -> Main function always returns an integer value.
{                  Function - { container for a set of instructions.
                              { There can be multiple functions

    int s, h, p;   /* Initialize */
                   comments
    h = 5;
    p = 12;

    s = h * p;     /* Formula */

    Printf("%d", s);
                   -> Format Specifier.

    return 0;      -> We are returning 0 here. Mea
                      0 indicates Success.

}
```

PROGRAM To Calculat
Salary

PROGRAM ADDING TWO NUMBERS

```
#include <stdio.h>
int main ()
{
 int integer 1;
 int integer 2;
printf("Enter First integer \n");
scanf ("%d", &integer 1);
Printf("Enter second integer \n");
Scanf("%d", &integer 2);
 int sum;
 Sum = integer 1 + integer 2;
 Printf("sum is %d \n", Sum);
return 0;
}
```

Understanding Scanf Function

F stands for formatted.
Scanf reads from the Standard i/p.

Syntax

```
Scanf(" ", &);
```

& — ampersand — Address operator
It is followed by variable name

```
Scanf("%d", &integer 1);
```

& with variable name informs scanf about the Location (Address) in memory at which the variable integer 1 is stored.

Arithmetic in C

C Operation	Arithmetic Operator	Algebraic Expression	C Expression
Addition	+	$f + 7$	f + 7
Subtraction	-	$p - c$	p - c
Multiplication	*	bm	b * m
Division	/	x / y or $\frac{x}{y}$	x / y
Remainder	%	$r \bmod s$	r % s

Sample Algebraic and C Expressions

Algebra: $m = \frac{a+b+c+d}{5}$

C : `(a+b+c) /5;`

Algebra: $y = mx + b$

C : `y = m * x + b;`

Algebra: $z = pr \bmod q + w/x - y$

C : `z = p * r % q + w/x - y;`

6 1 2 4 3 5

Solve

$$y = 2*5*5 + 3*5 + 7;$$

Decision Making – Relation operators :

Algebraic	C relational operator	Example
$>$	$>$	$x > y$
$<$	$<$	$x < y$
$\geq$	$>=$	$x >= y$
$\leq$	$<=$	$x <= y$
$=$	$==$	$x == y$
$\neq$	$!=$	$x != y$

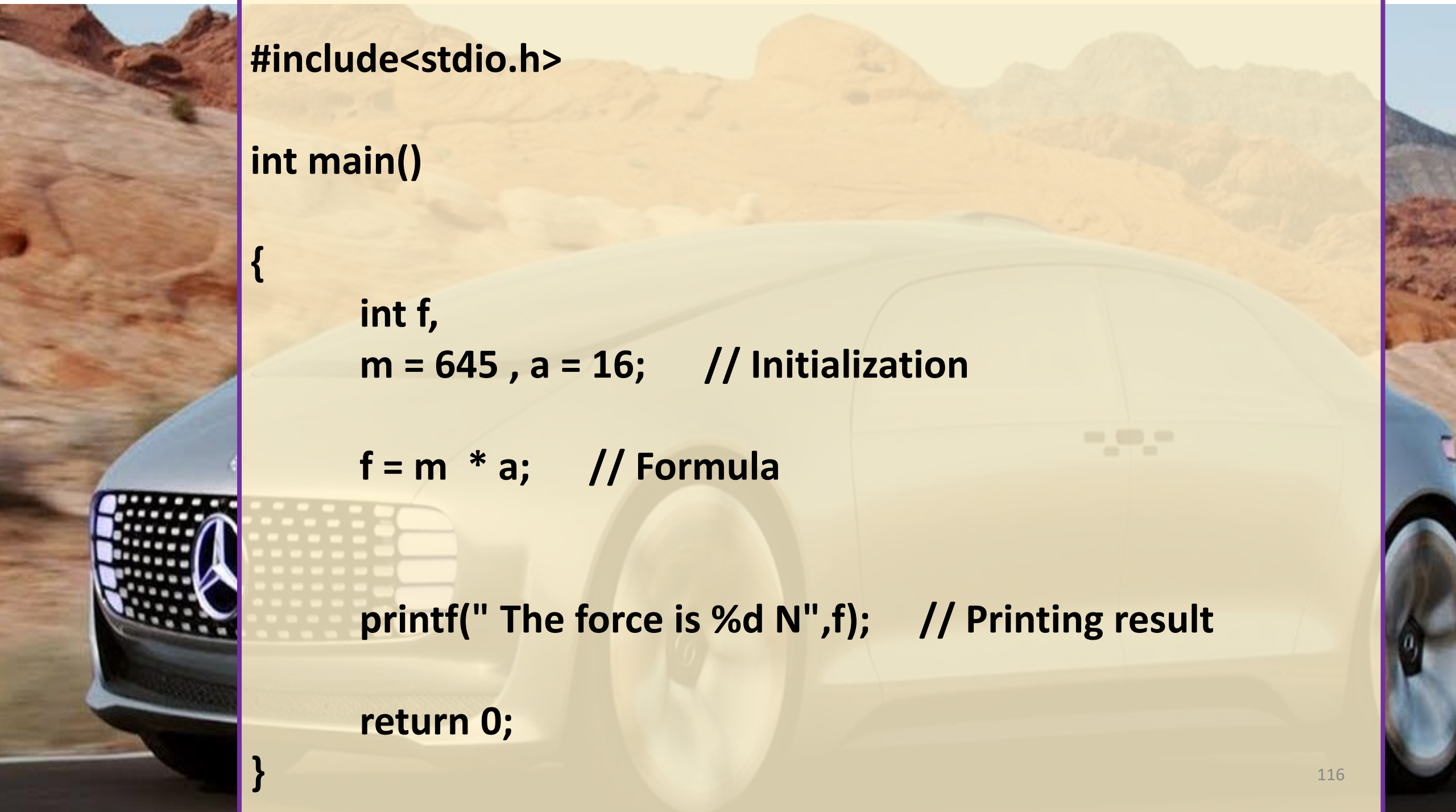
Key Words:-

auto	for	double
break	while	case
case	static	typedef
char	int	union
const	float	volatile
do	enum	unsigned
else	extern	
if	long	

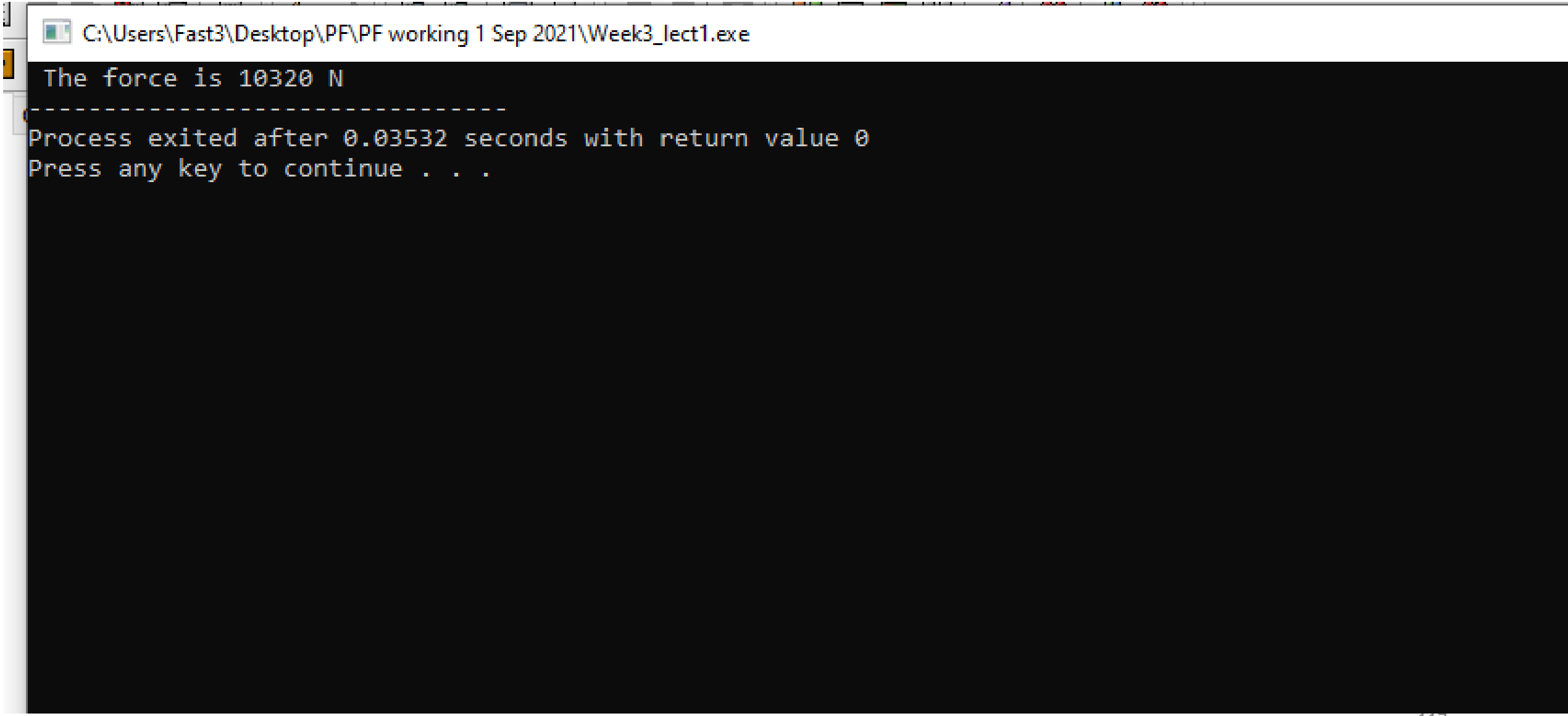
Keywords are the words whose meaning has already been explained to the C compiler. There 32 keywords in C.

WEEK 3 Lecture Slides

A Self-driving (SD) car is moving on a straight road with an acceleration of 16 m/s^2. The mass of a SA car is 645 kg. Develop a C-code that determines what force is required to accelerate SD car ?



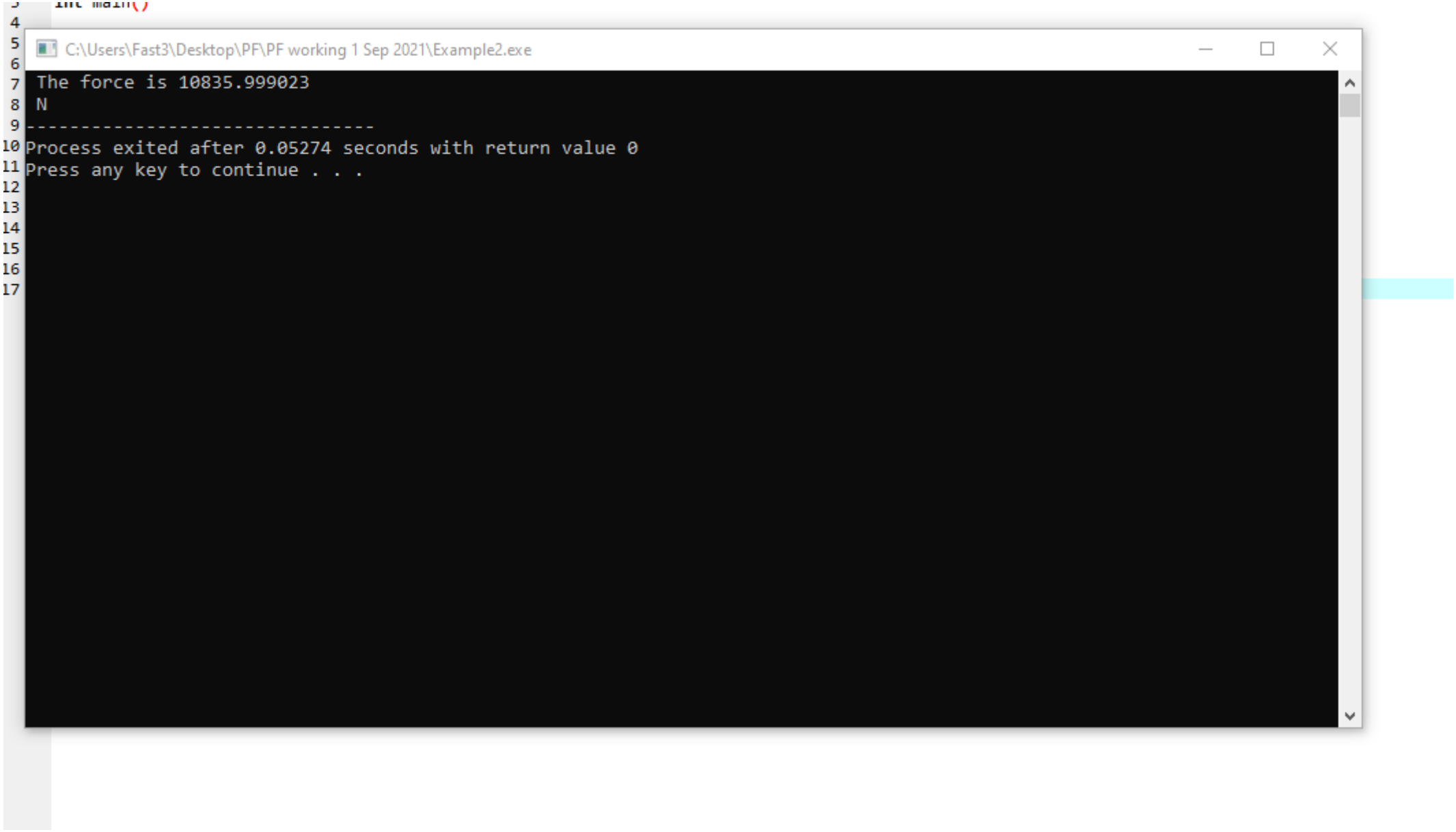
```
#include<stdio.h>

int main()

{
        int f,
        m = 645 , a = 16;      // Initialization

        f = m  * a;      // Formula


        printf(" The force is %d N",f);     // Printing result

        return 0;
}
```


C:\Users\Fast3\Desktop\PF\PF working 1 Sep 2021\Week3_lect1.exe
The force is 10320 N

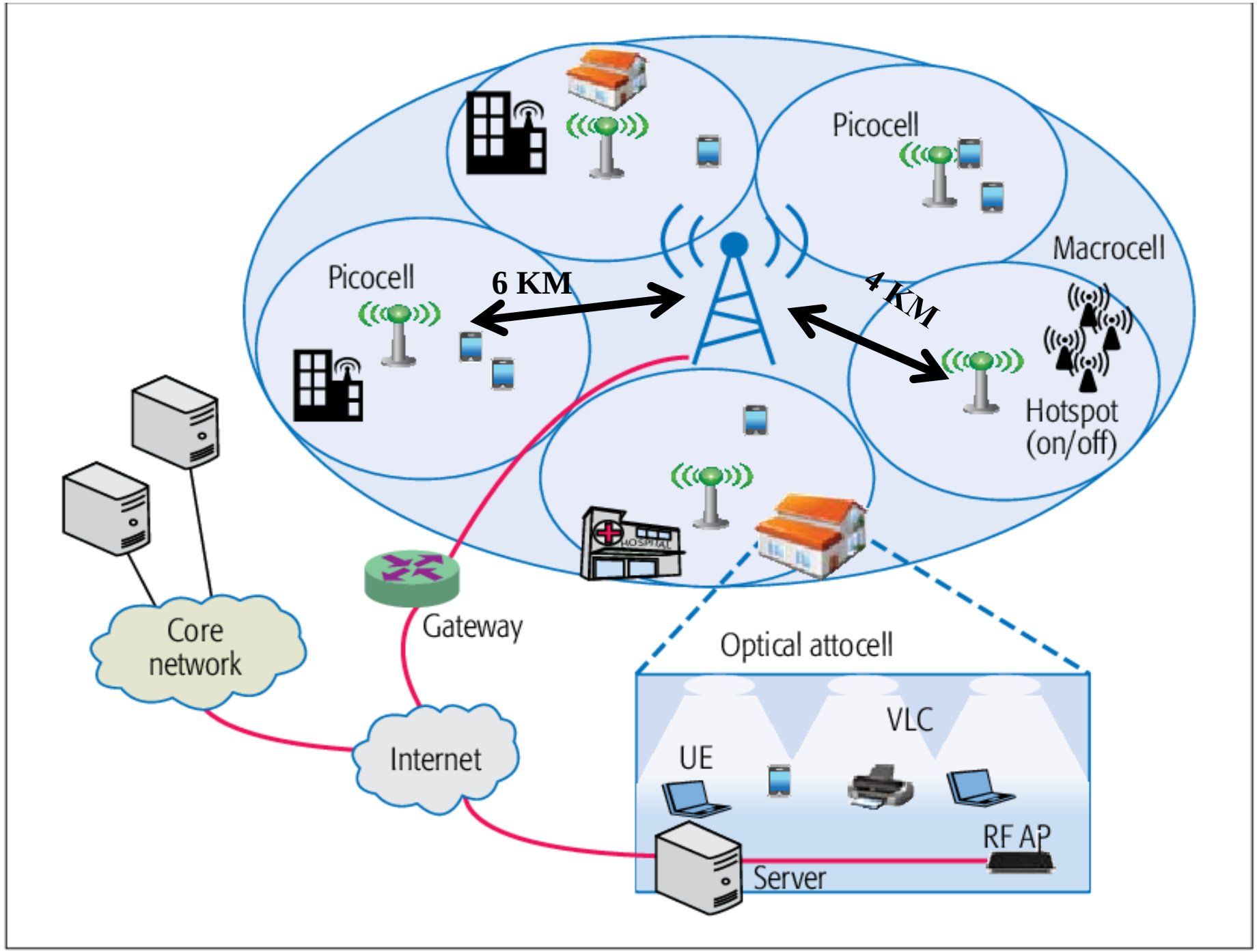
Process exited after 0.03532 seconds with return value 0
Press any key to continue . . .

A Self-driving car is moving on a straight road with an acceleration of 16.8 m/s^2. The mass of a SA car is 645 kg. Develop a C-code that determines what force is required to accelerate SD car ?

```
#include<stdio.h>

int main()

{
        int  m = 645;

        float f, a = 16.8;  // Initialization


        f = m  * a;   // Formula


        printf(" The force is %f \n N",f);    // Printing
result

        return 0;
}
```


int main()
C:\Users\Fast3\Desktop\PF\PF working 1 Sep 2021\Example2.exe
The force is 10835.999023
N

Process exited after 0.05274 seconds with return value 0
Press any key to continue . . .


Picocell
Picocell
6 KM
4 KM
Macrocell
Hotspot
(on/off)
HOSPITAL
Core
network
Gateway
Internet
Optical attocell
VLC
UE
RF AP
Server

Steps in Learning English Language

Alphabets → Word → Sentences → Paragraph

Steps in Learning C Language

Alphabets
Digits
Special Symbols → Constants
Variables
Keywords → Instructions → Program

The C character Set

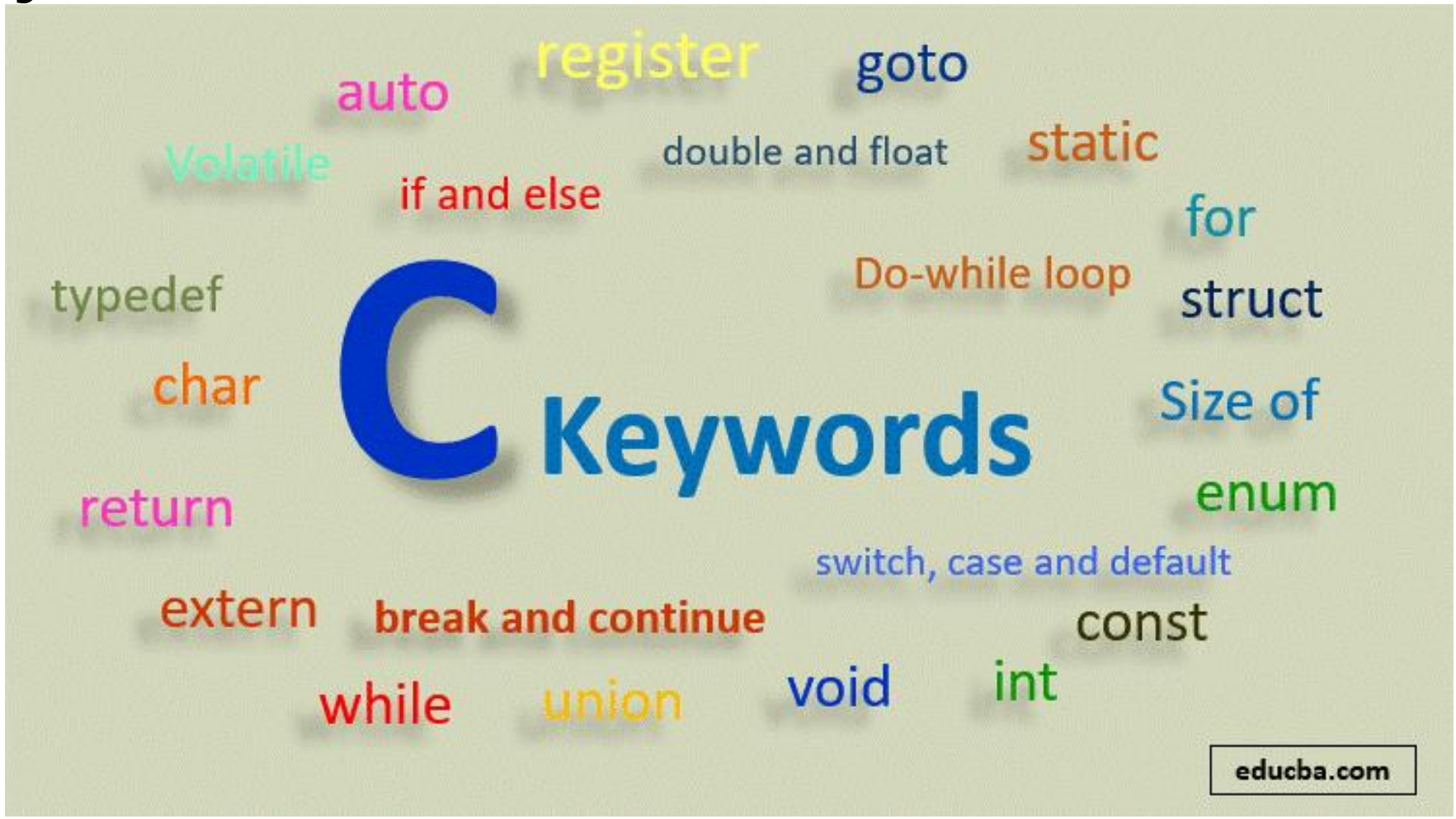
Alphabets	A,B,C……..,Z a,b,c…..z
Digits	0,1,2,3,4,5,6,7,8,9
Special Symbols	@,¬><~#&^%$££”!_- [],{} ()

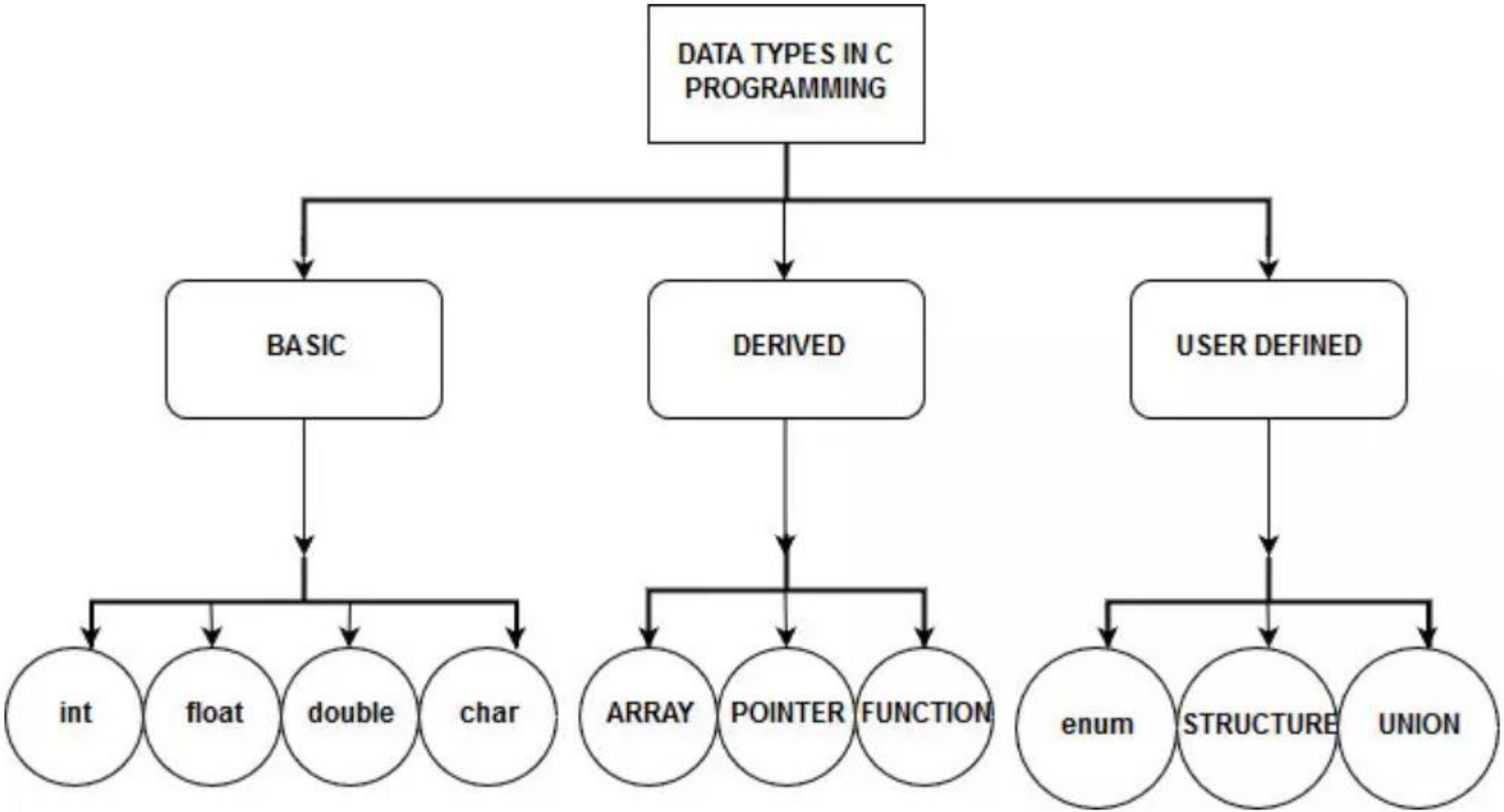
Constant, Variables and Key Words

- Constant: It is an entity that doesn't change.
- Variable: It is an entity that may change.
- Keyword is a word that carries special meaning.

In programming language, constants are also called as ***Literals*** and variables are also called as ***Identifiers***.

C keywords


register
goto
auto
static
double and float
Volatile
if and else
for
Do-while loop
typedef
struct
C
Keywords
char
Size of
enum
return
switch, case and default
extern
break and continue
const
while
union
void
int
educba.com


DATA TYPES IN C PROGRAMMING
BASIC
DERIVED
USER DEFINED
int
float
double
char
ARRAY
POINTER
FUNCTION
enum
STRUCTURE
UNION

Type	Bits	Minimal Range
char	8	–127 to 127
unsigned char	8	0 to 255
signed char	8	–127 to 127
int	16 or 32	–32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	Same as **int**
short int	16	–32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	Same as **short int**
long int	32	–2,147,483,647 to 2,147,483,647
long long int	64	$-(2^{63}-1)$ to $2^{63}-1$ (Added by C99)
signed long int	32	Same as **long int**
unsigned long int	32	0 to 4,294,967,295
unsigned long long int	64	$2^{64}-1$ (Added by C99)
float	32	1E–37 to 1E+37 with six digits of precision
double	64	1E–37 to 1E+37 with ten digits of precision
long double	80	1E–37 to 1E+37 with ten digits of precision

Range of data type depend upon the compiler

Example: for VS it is ***-2147483648 to + 214748364***

whereas, for compiler like Turbo

C or Turbo C++ the range is ***-32768 to +32767***

Types of Constants

- Primary Constant
  - Integer
  - Float
  - Character Constant
- Secondary Constant
  - Array
  - Pointers
  - Structure
  - Union and Enum etc

Rules for constructing Integer constants

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can either be positive or negative.
- No commas or blanks within an integer constant.
- The range for integer constant is

-2147483648 to +2147 2147483647

Example: 426 , 782, -6000, 500

Rules for constructing Real constants

- Real constant are also called as Floating point constant.
- It can be written in two formats: **Fractional** and **Exponential**.

For Fractional format:

- A real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.
- Default sign is positive.
- No commas or blanks within the real constant.

Rules for constructing Real constants

For Exponential format:

It is used when the value of the constant is either very large or very small.

It has two parts mantissa and exponent. (0.000342 as 3.42e-4).

Rules:

- The mantissa part and the exponential part should be separated by the letter E.
- The mantissa part may have positive or negative sign.
- Default sign of mantissa is positive.
- The exponent must have at least one digit which must be a positive or negative integer. Default is positive.

Range in exponential form is

-3.4e38 to 3.4e38

Example: +3.2e-5, 4.1e8, -0.3e-3

Rules for constructing Character constants

- A character constant is a single alphabet, a single digit or a single special symbol enclosed with in the single inverted commas.
- Both the inverted commas should point to the left.

’A’ valid

‘A’ not valid

’I’

’5’

’=’

Variables

A variable is a space in memory that plays the same role many times but may contain a different value each time.

Defining Variables name

- A variable can hold only the same type constant. Example integer can hold integer constant. A real variable can hold real constant.

Rules for constructing Variable Names

- A variable name is any combination of 1 to 31 alphabets or alphabet with digits or alphabet with digits and underscores.
- Some compiler allow variable names whose length could be up to 247 characters. (Conventional length 31 characters).
- The first character in the variable name must be an alphabet or underscore (_).
- No commas or blanks are allowed within a variable name.
- No special symbol other than underscore can be used in a variable.

si_int

t_e_7

How is C able to differentiate between variable types ?

```
int si, m_hi;
float temp;
char code;
```

Types of Instructions in C

A Type Declaration instruction

Use to declare the type variables used in a C program

B Arithmetic Instruction

Use to perform arithmetic operations on constant and variables

C Control Instruction

Control the sequence of execution of various statements in a C program.

A Type Declaration Instruction

- It is used to declare the type of variable used in a program.
- Declaration must be done at the beginning of main function.

ex: int bas;

float rs, salary;

char name, code;

A Type Declaration Instruction (contd..)

a) While declaring the type of variable we can also initialize it as :

```
int i =10; j =25;

 float a = 1.7, b = 1.99 +2.4*1.8;
```

```
#include<stdio.h>
int main()
{
        float b = 1.99 + 2.4 *1.44;

        printf("%f",b);
}
```

```
5.446000
--------------------------------
Process exited after 0.06468 seconds with return value 8
Press any key to continue . . .
```

What if we need 5.44 as an output ?

A Type Declaration Instruction (contd..)

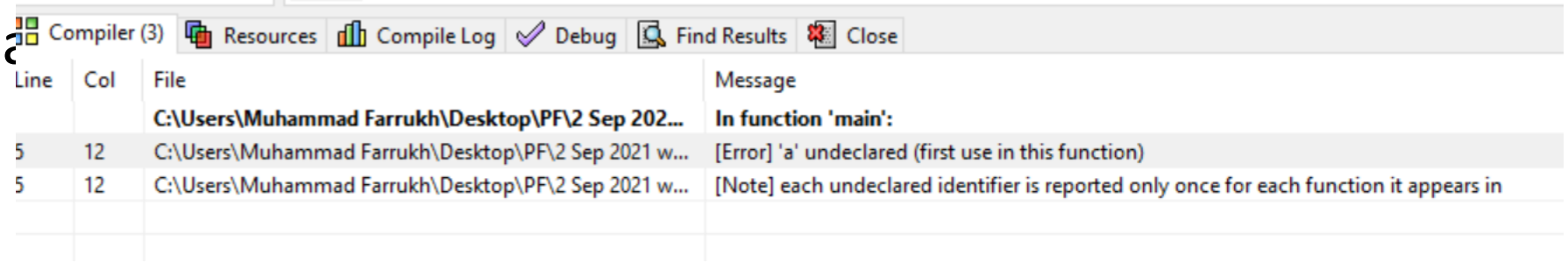
b) The order in which we define the variable is sometimes important sometimes not.

```
int i = 10, j = 25;
```

```
int j = 25, i = 10;
```

SAME

```
float a = 1.5, b = a
float b = a + 3.1,
```



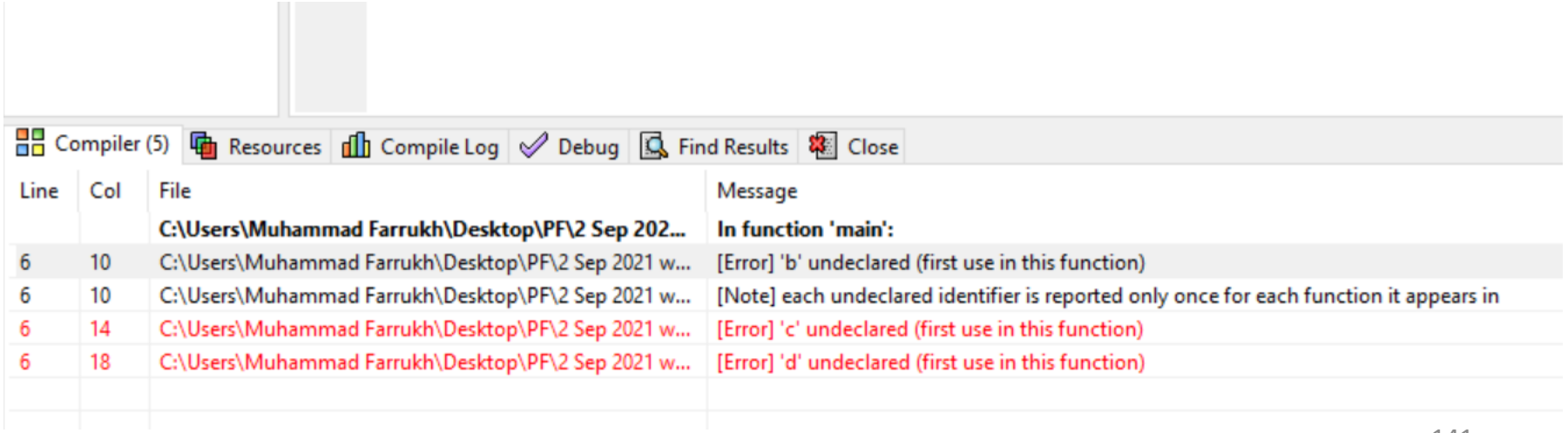
A Type Declaration Instruction (contd..)

C) The following statements would work:

```
int a,b,c,d;
a=b=c=10;
```

How about ?

```
int a = b = c =
```



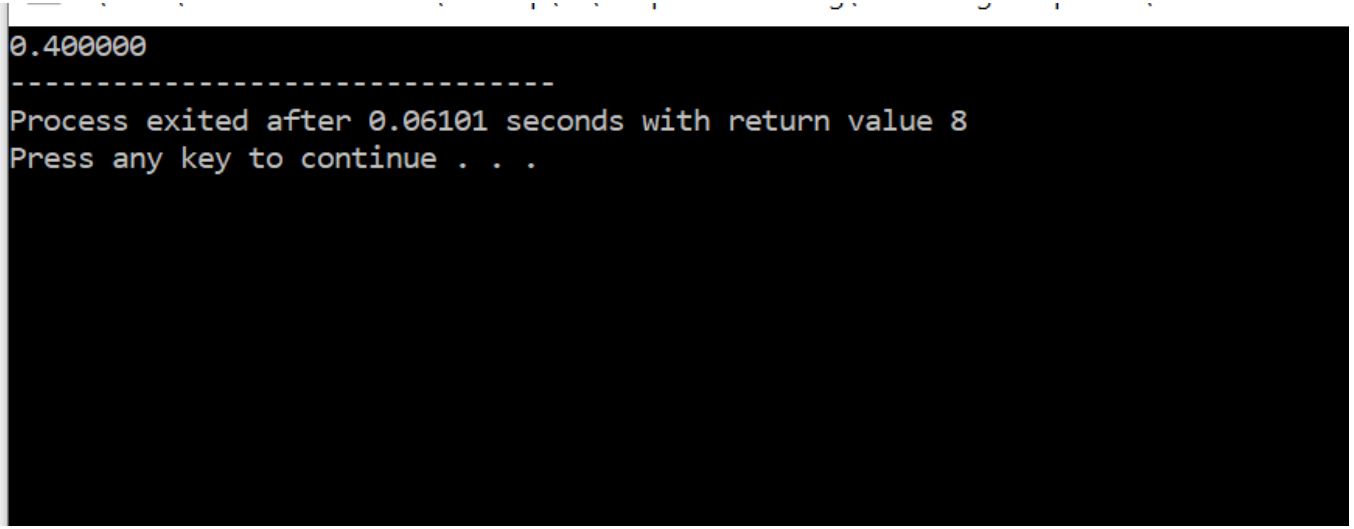
B) Arithmetic Instruction

- Arithmetic instruction consists of a variable name on the left hand side of = and variable names and constants on the right hand side of =.

- The variables and constants appearing on the right hand side of = are connected by arithmetic operation like + - * and /

B) Arithmetic Instruction (contd..)

```
#include<stdio.h>
int main()
{

        int time, u;

        float s, acc;

        time = 2;
        u = 0;

        acc = 0.2;

        s = u*time + 0.5*(acc*time*time);

        printf("%f",s);
return 0;
}
```



The variables and constants together are called operands

B) Arithmetic Instruction (contd..)

- Three types of Arithmetic statement:

Integer mode arithmetic statement – all operands are either integer variable or integer constants.

example : int j, mass;

```
j = j + 1;
force = mass * 64;
```

- Real mode arithmetic statement- all operands are either real constants or real variables

example: float a, m, o;

```
m = 2.2;
o =  3.4;
a = m + o -3.16;
```

- Mixed mode arithmetic statement - some operands are integers and some operands are real.

example:

```
int a,b,c,d;
 float avg, si, p, an, ri;
 avg = (a +b+c+d)/4;
 si = p *an*ri / 100.0;
```

Points to be noted:

z = k * I	legal
k*I = z	Not legal
a = c.d.b(xy)	Arithmetic statement
a = c*d*b*(x*y)	C statement

Integer and Float Conversions

- An arithmetic operation between an **integer** and **integer** always yields an **integer** result.

- An operation between a **real** and **real** always yields a **real** result.

- An operation between an **integer** and **real** always yields a **real** result. In this operation the integer is first promoted to a real and then operation is performed. Hence result is **real**.

Operation	Result
5 / 2	2
2 / 5	0
5.0 / 2	2.5
2.0 / 5	0.4
5 / 2.0	2.5
2.0 / 5.0	0.4

Promotion and Demotion

```
int i ;
float b;

i = 3.5;
b = 30;
```

```
3 30.000000
--------------------------------
Process exited after 0.06017 seconds with return value 11
Press any key to continue . . .
```

Consequences ???

```
#include<stdio.h>
int main()
{


        int k ;    // integer

        float a;   // real

        k = 2/9;

        a = 2/9;


        printf("%d\n",k);
        printf("%f\n",a);
return 0;
}
```

Demonstrate on Dev C++

k is an integer

a is a real variable

k = 2 / 9	0	a = 2 / 9	0.0
k = 2 / 9	0	a = 2 / 9	0.0
k = 2 / 9.0	0	a = 2 / 9.0	0.22222
k = 2.0 / 9.0	0	a = 2.0/ 9.0	0.22222
k = 9 / 2	4	a = 9/2	4.0
k = 9.0 / 2	4	a = 9.0/2	4.5

Hierarchy of Operation

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

Hierarchy of Operation

Example 1: Determine the hierarchy of operations and evaluate the following expression:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

Stepwise evaluation of this expression is shown below:

i is an integer

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	operation: /
i = 1 + 1+ 8 - 2 + 5 / 8	operation: /
i = 1 + 1 + 8 - 2 + 0	operation: /
i = 2 + 8 - 2 + 0	operation: +
i = 10 - 2 + 0	operation: +
i = 8 + 0	operation: -
i = 8	operation: +

Example 2: Determine the hierarchy of operations and evaluate the following expression:

k = 3 / 2 * 4 + 3 / 8 + 3

k is a float

Stepwise evaluation of this expression is shown below:

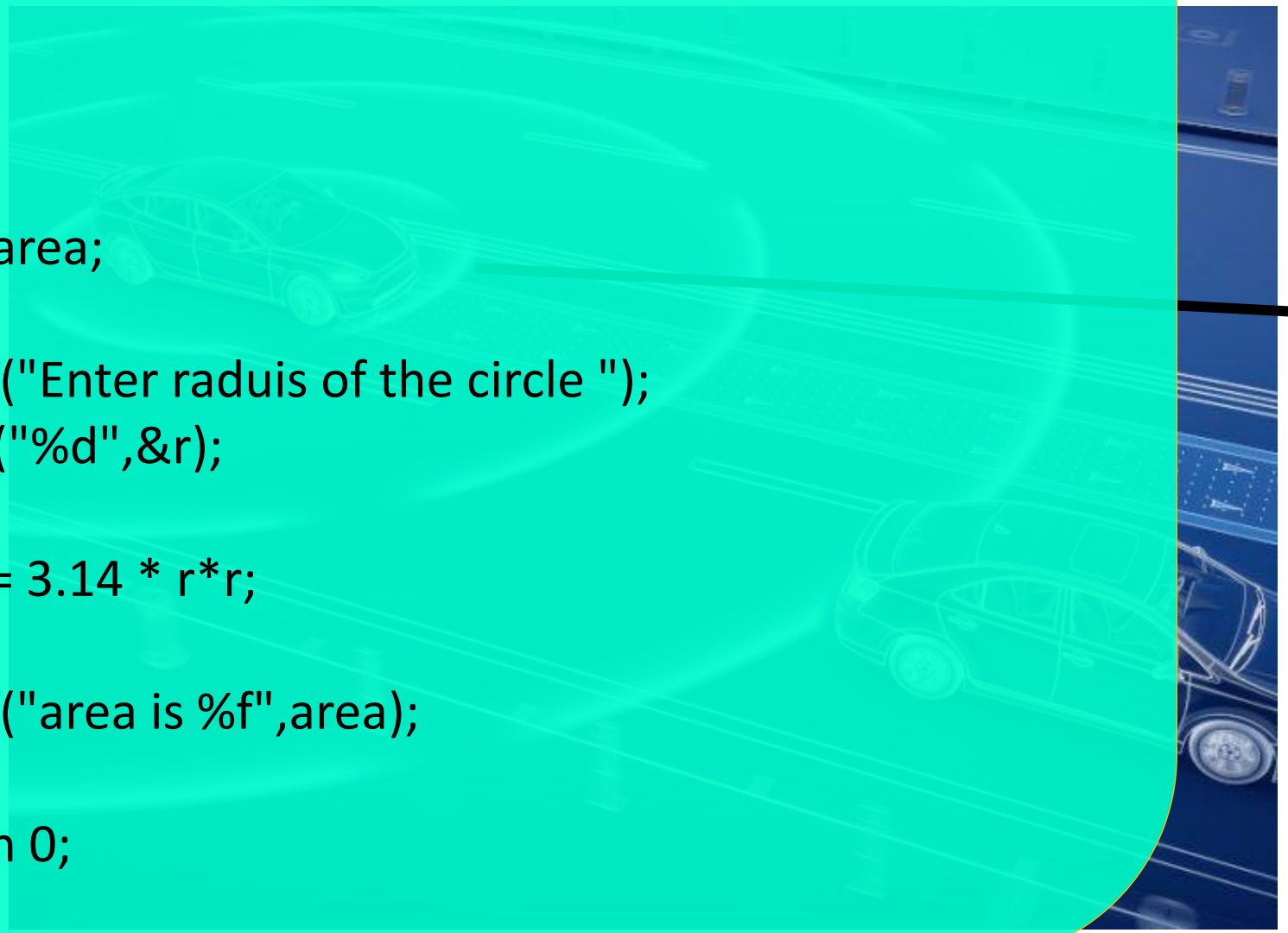
k = 3 / 2 * 4 + 3 / 8 + 3	
k = 1 * 4 + 3 / 8 + 3	operation: /
k = 4 + 3 / 8 + 3	operation: *
k = 4 + 0 + 3	operation: /
k = 4 + 3	operation: +
k = 7	operation: +

Note that 3 / 8 gives zero, again for the same reason mentioned in the previous example.

Algebric Expression	C Expression
$a \times b - c \times d$	a * b – c * d
$(m + n)\ (a + b)$	(m + n) * (a + b)
$3x2 + 2x + 5$	3 * x * x + 2 * x + 5
$\frac{a + b + c}{d + e}$	(a + b + c) / (d + e)
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)}\right]$	2 * b * y / (d + 1) – x / 3 * (z + y)

A SA is moving on a road with activated built in sensors. The circle shows the range of the different sensors and it moves on. Determine the area of first circle given the radius as a variable input specified by the user.

```
#include<stdio.h>

int main()
{
        int r;

        float area;

        printf("Enter raduis of the circle ");
        scanf("%d",&r);

        area = 3.14 * r*r;

        printf("area is %f",area);

        return 0;
}
```



Area of the first circle

Input and Output Functions

scanf()

printf()

Format Specifiers

Data Types	Format Specifier
int	%d
short	%d
long	%ld
char	%c
float	%f

Escape Sequences

Esc. Seq.	Purpose	Esc. Seq.	Purpose
\n	New line	\t	Tab
\b	Backspace	\r	Carriage return
\f	Form feed	\a	Alert
\'	Single quote	\"	Double quote
\\	Backslash		

scanf()

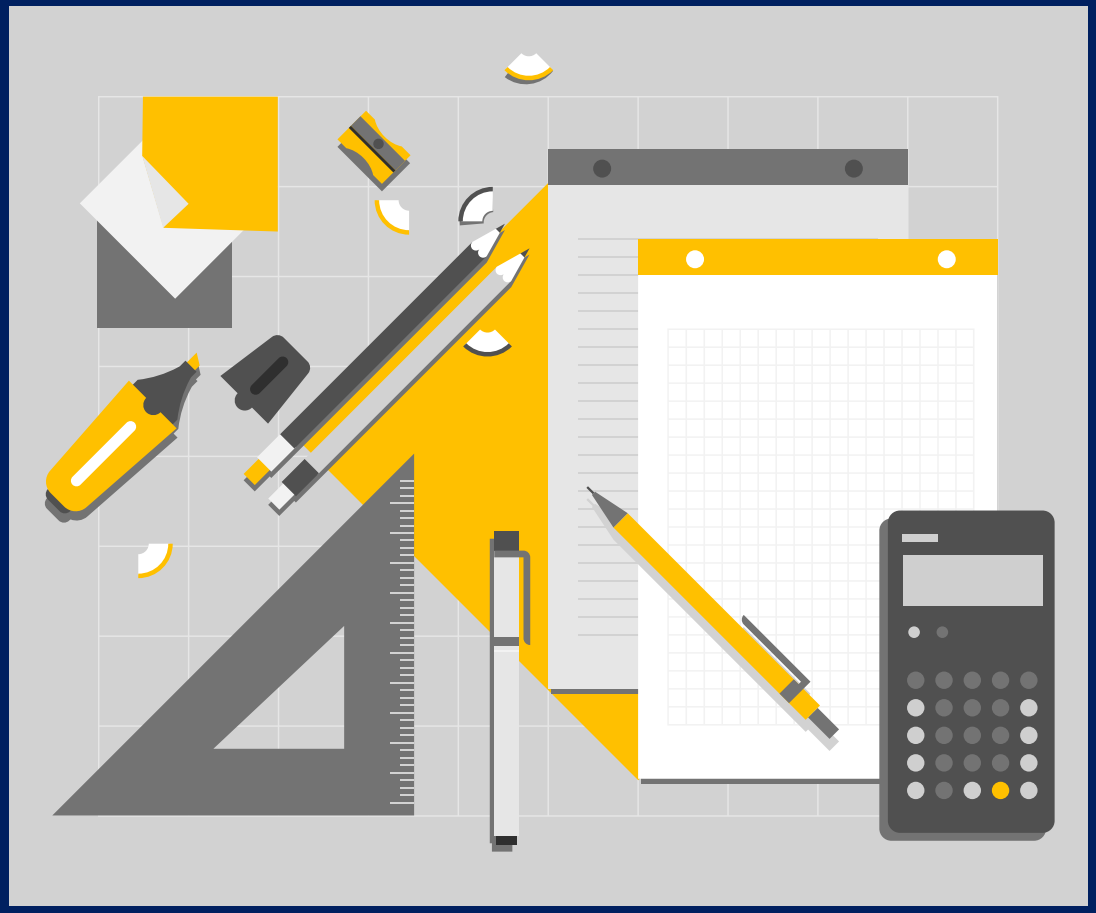
scanf(“%yyy”, &variablename);

printf()

printf(“%www”, variablename);

Convert the following algebraic expressions into equivalent C statements

- $Z = \frac{(x+3)\, x^3}{(y-4)(y+5)}$
- R= $\frac{2v+6.22\ (c+d)}{(y-4)(y+5)}$
- X= $\frac{12x^3}{4x} + \frac{8x^2}{14x} + \frac{x}{8x}$



Forms of if

The if statement can take any of the following forms:

(a) if (condition)
do this ;

(b) if (condition)
{
do this ;
and this ;
}

(c) if (condition)
do this ;
else
do this ;

(d) if (condition)
{
do this ;
and this ;
}
else
{
do this ;
and this ;
}

Forms of if

```
(e) if ( condition )
 do this ;
else
{
 if ( condition )

do this ;
 else
 {

do this ;

and this ;
 }
}
```

```
(f) if ( condition )
{
 if ( condition )


do this ;
 else
 {


do this ;


and this ;
 }
}
else
 do this ;
```

```
if ( 3 + 2 % 5 )

printf ( "This works" ) ;

if ( a = 10 )

printf ( "Even this works" )  ;

if ( -5 )

printf ( "Surprisingly even this works" ) ;
```

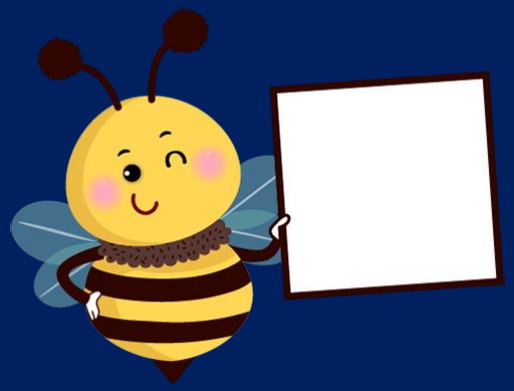
Note that in C a non-zero value is considered to be true, whereas a 0 is considered to be false.

In place of -5 even if a float like 3.14 were used it would be considered to be true. So the issue is not whether the number is integer or float, or whether it is positive or negative. Issue is whether it is zero or non-zero.

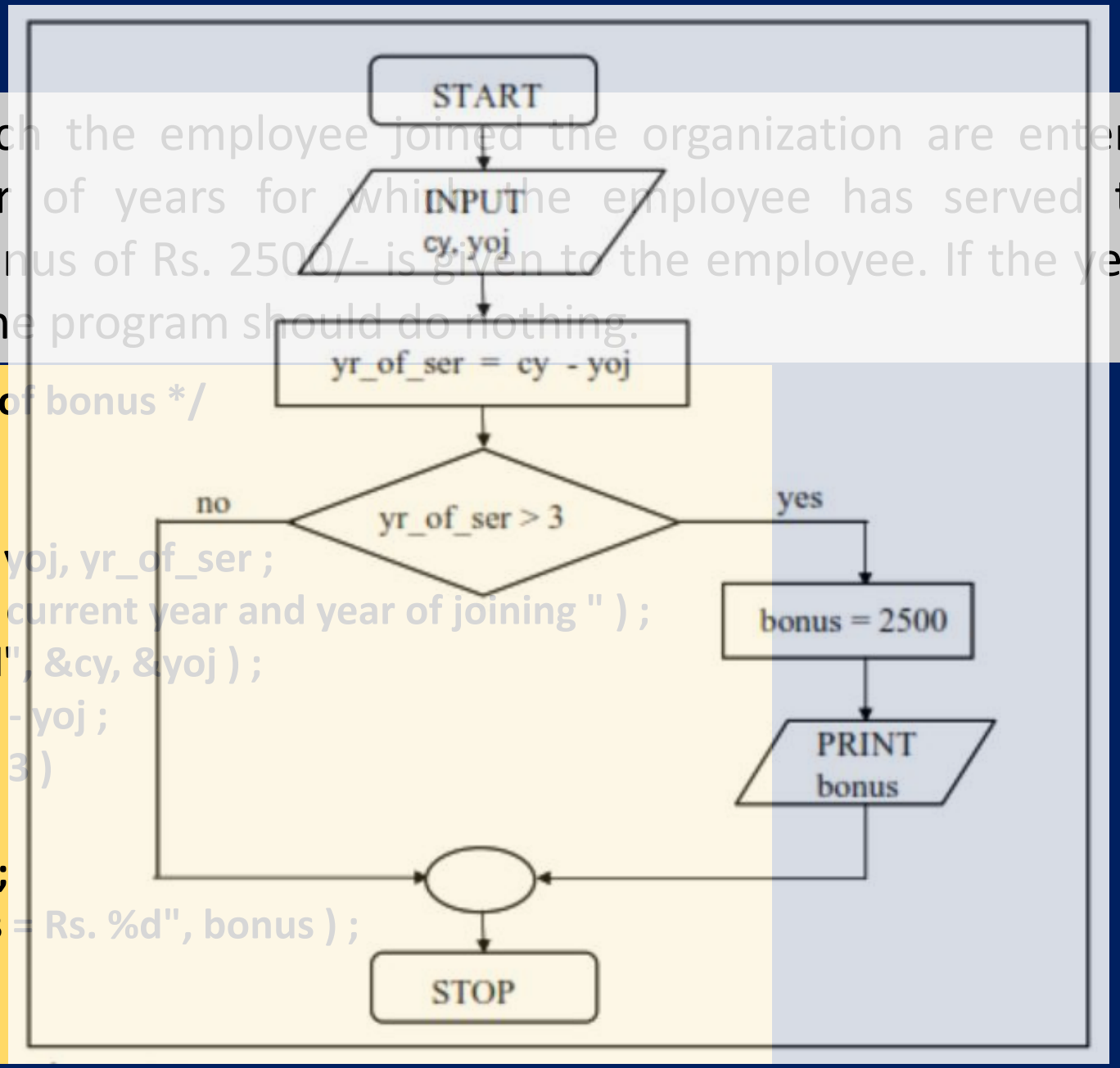
The current year and the year in which the employee joined the organization are entered through the keyboard. If the number of years for which the employee has served the organization is greater than 3 then a bonus of Rs. 2500/- is given to the employee. If the years of service are not greater than 3, then the program should do nothing.

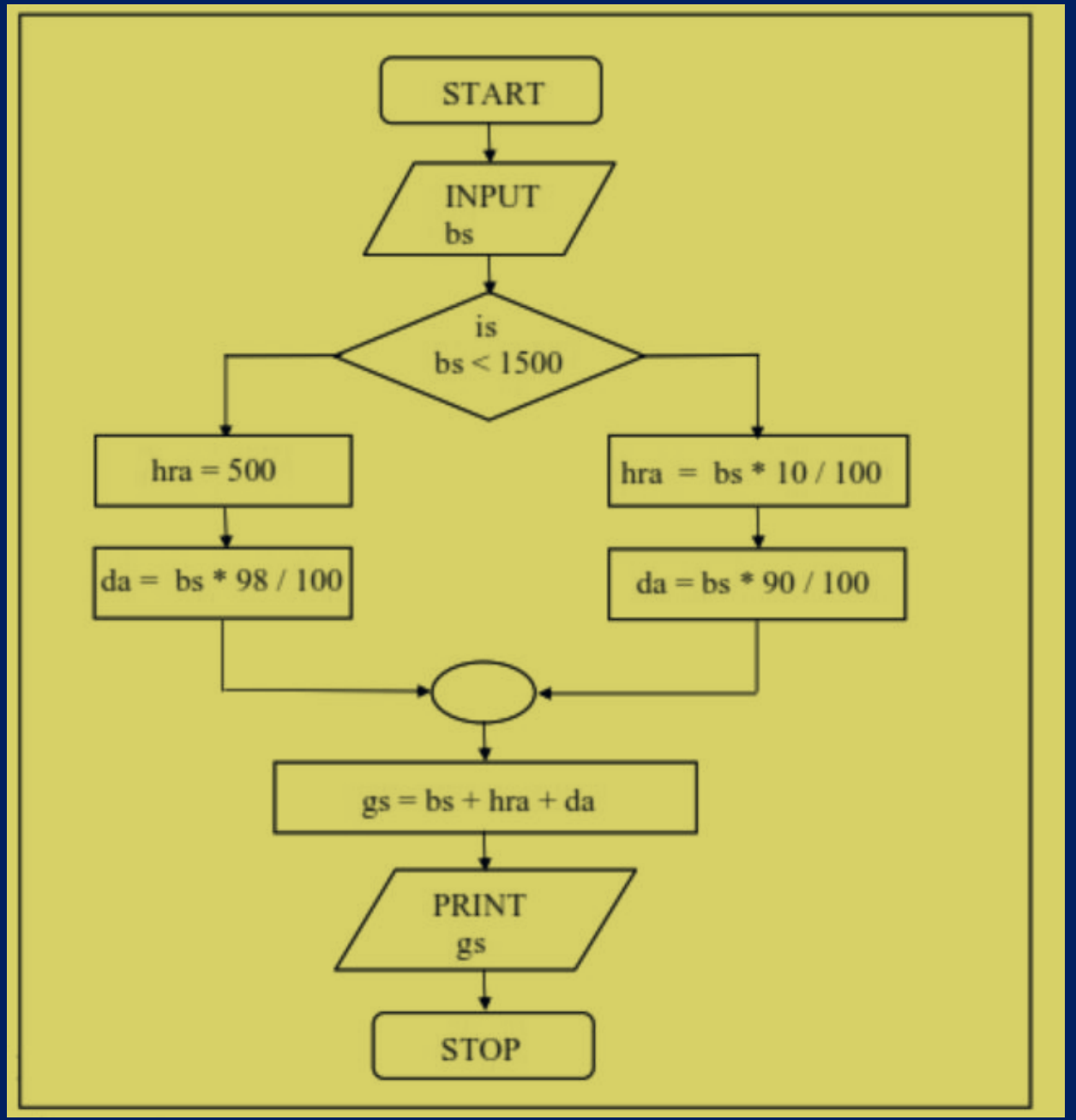
```
/* Calculation of bonus */
main( )
{
int   bonus, cy, yoj, yr_of_ser ;
 printf ( "Enter current year and year of joining " ) ;
scanf ( "%d %d", &cy, &yoj ) ;
 yr_of_ser = cy - yoj ;
if ( yr_of_ser > 3 )
 {
 bonus = 2500 ;
printf ( "Bonus = Rs. %d", bonus ) ;
 }
}
```

The current year and the year in which the employee joined the organization are entered through the keyboard. If the number of years for which the employee has served the organization is greater than 3 then a bonus of Rs. 2500/- is given to the employee. If the years of service are not greater than 3, then the program should do nothing.



```
/* Calculation of bonus */
main( )
{
int   bonus, cy, yoj, yr_of_ser ;
 printf ( "Enter current year and year of joining " ) ;
scanf ( "%d %d", &cy, &yoj ) ;
 yr_of_ser = cy - yoj ;
if ( yr_of_ser > 3 )
 {
 bonus = 2500 ;
printf ( "Bonus = Rs. %d", bonus ) ;
 }
}
```




START
INPUT
bs
is
bs < 1500
hra = 500
da = bs * 98 / 100
hra = bs * 10 / 100
da = bs * 90 / 100
gs = bs + hra + da
PRINT
gs
STOP

Codes !

```
#include<stdio.h>
#include<math.h>
int main()
{
printf("Enter x and y coordinates \n");
float x,y;
scanf("%f %f",&x,&y);
printf("Polar coordinates are (%f,%f).\n",sqrt(pow(x,2.0)+pow(y,2.0)),atan(y/x));
return 0;
}
```

```
#include<stdio.h>
int main()
{

 printf("Enter number\n");

 int number;

 scanf("%d",&number);

  if(number%2==0)

 {
 printf("Number is Even");
 }
 else
 {
 printf("Number is Odd");
 }
 return 0;
}
```

```
#include<stdio.h>
int main()
{
int num;
printf("Enter an integer : ");
scanf("%d", &num);
if(num%2 != 0)    //odd
printf("\n\nThe number is odd");
Else                    //even
printf("\n\nThe number is even.");

return 0;
}
```

```
#include<stdio.h>
int main()
{
int angle1,angle2,angle3;
printf("Enter the angles of Triangle in any order in
Degrees.\n");
scanf("%d %d %d",&angle1,&angle2,&angle3);
if(angle1+angle2+angle3==180)
{
printf("Triangle is Valid.");
}
else
{
printf("Triangle is Invalid.");
}
return 0;
}
```

```
#include <stdio.h>
int main()
{
int l,b;
printf("Enter length and breadth of rectangle\n");
scanf("%d %d",&l,&b);
if((l*b)>(2*(l+b)))
{
   printf("Area is Greater than perimeter\n");
}
else
{
   printf("Area is not Greater than perimeter\n");
}
return 0;
}
```

```
#include <stdio.h>
int main()
{
int l,b;
printf("Enter length and breadth of rectangle\n");
scanf("%d %d",&l,&b);
if((l*b)>(2*(l+b)))
{
   printf("Area is Greater than perimeter\n");
}
else
{
   printf("Area is not Greater than perimeter\n");
}
return 0;
}
```

```
#include <stdio.h>
int main()
{
 float x1,y1,x2,y2,x3,y3;
printf("Enter coordinates (x1,y1)\n");
scanf("%f %f",&x1,&y1);
printf("Enter coordinates (x2,y2)\n");
scanf("%f %f",&x2,&y2);
printf("Enter coordinates (x3,y3)\n");
scanf("%f %f",&x3,&y3);
if((y2-y1)/(x2-x1)==(y3-y1)/(x3-x1))
{
   printf("Point lies on straight line");
}
else
{
   printf("Points don't lie on straight line");
}
return 0;
}
```

```
#include<stdio.h>
#include<math.h.>
int main()
{
   float cen_x,cen_y,rad,x,y,d;
   printf("Enter Center x,y coodinate and radius\n");
   scanf("%f %f %f",&cen_x,&cen_y,&rad);
   printf("Enter coordinates of point\n");
   scanf("%f %f",&x,&y);
   d=sqrt(pow((cen_x-x),2.0)+pow((cen_y-y),2.0));
   if(d<rad)
    {
      printf("Point is inside circle\n");
    }
   else if(d==rad)
 {
      printf("Point is on the circle\n");
 }
 else if(d>rad)
 {
       printf("Point is outside the circle\n");
 }
 return 0;
}
```

```
#include<stdio.h>
int main()
{

        int n, d5,d4,d3,d2,d1;

        long int revnum;

        printf("Enter a number");

        scanf("%d",&n);

        d5 = n%10;

        n = n/10;

        d4 = n%10;

n = n/10;

        d3 = n%10;

        n = n/10;

        d2 = n%10;

        n = n/10;

        d1 = n%10;

        n = n/10;

        revnum = d5*10000+d4*1000+d3*100+d2*10+d1;

        printf("%d\n", d5);
        printf("%d\n", d1);
        printf("%ld\n", revnum);

        return 0;

}
```

Logical Operators

C allows usage of three logical operators, namely, &&, || and !
These are to be read as ‘AND’ ‘OR’ and ‘NOT’ respectively.

The marks obtained by a student in 5 different subjects are input through the keyboard. The student gets a division as per the following rules:

Percentage above or equal to 60 - First division
Percentage between 50 and 59 - Second division
Percentage between 40 and 49 - Third division
Percentage less than 40 - Fail

Write a program to calculate the division obtained by the student.

1st Method

```
main( )

{

 int   m1, m2, m3, m4, m5, per ;

printf ( "Enter marks in five subjects " ) ;

 scanf ( "%d %d %d %d %d", &m1, &m2, &m3, &m4,
&m5 ) ;

 per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
```

```
if ( per >= 60 )
printf ( "First division ") ;
 else
 {
if ( per >= 50 )

printf ( "Second division" ) ;
    else
    {

if ( per >= 40 )

printf ( "Third division" ) ;
      else
    printf ( "Fail" ) ;
    }

 }
}
```

```
main( )
{
 int  m1, m2, m3, m4, m5, per ;


printf ( "Enter marks in five subjects " ) ;
 scanf ( "%d %d %d %d %d", &m1, &m2, &m3, &m4,
&m5 ) ;
 per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
```

```
if ( per >= 60 )

printf ( "First division" ) ;

if ( ( per >= 50 ) && ( per < 60 ) )

printf ( "Second division" ) ;

if ( ( per >= 40 ) && ( per < 50 ) )

printf ( "Third division" ) ;

if ( per < 40 )

printf ( "Fail" ) ;
}
```

A company insures its drivers in the following cases:

- **If the driver is married.**
- **If the driver is unmarried, male & above 30 years of age.**
- **If the driver is unmarried, female & above 25 years of age.**

In all other cases the driver is not insured. If the marital status, gender and age of the driver are the inputs, write a program to determine whether the driver is to be insured or not.

```
main( )
{

char   gen , ms ;
int   age ;
printf ( "Enter age, gen, marital status " ) ;

scanf ( "%d %c %c“, &age, &gen, &ms ) ;

if ( ( ms == ‘m') || ( ms == 'U' && gen == 'M' && age > 30 ) ||

 ( ms == 'U' && gen == 'F' && age > 25 ) )

printf ( "Driver is insured" ) ;
 else
printf ( "Driver is not insured" ) ;
}
```