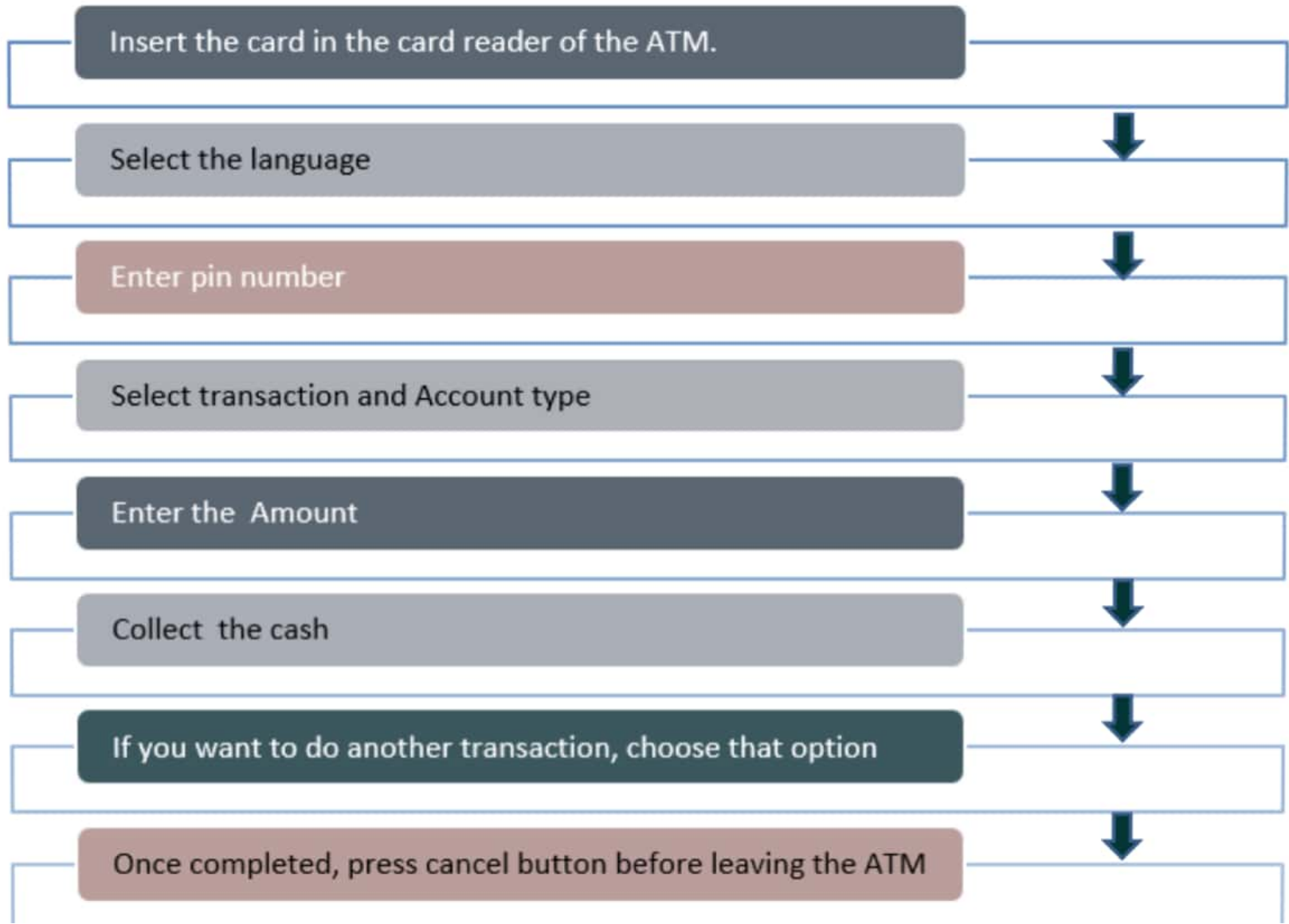




LOGIC DEVELOPMENT





- Here our problem statement is
Withdrawal of cash from ATM
- To solve it, we followed some instructions in a specific order, which is nothing but an **ALGORITHM**




In this module you will learn



- Introduction to Algorithms , flow charts and Pseudo Code
 - Selection and Iterative statements
- Introduction to Arrays
 - 1-Dimension and 2-Dimensional arrays
 - Algorithm for manipulating arrays
- Characteristics of a good program
- Difference between correctness and robustness
- Coupling and Cohesion



What is an Algorithm?

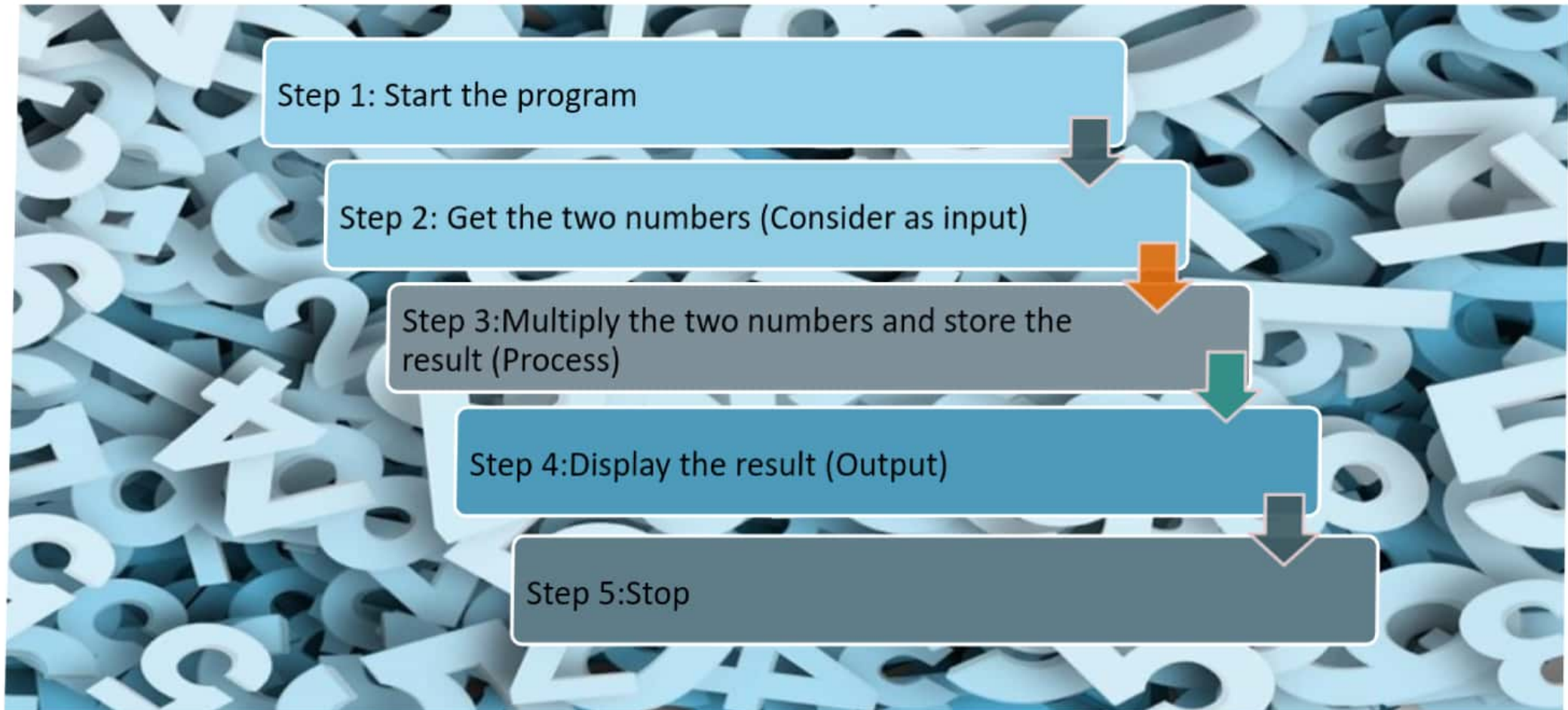


Set of instructions to be followed in problem solving operations

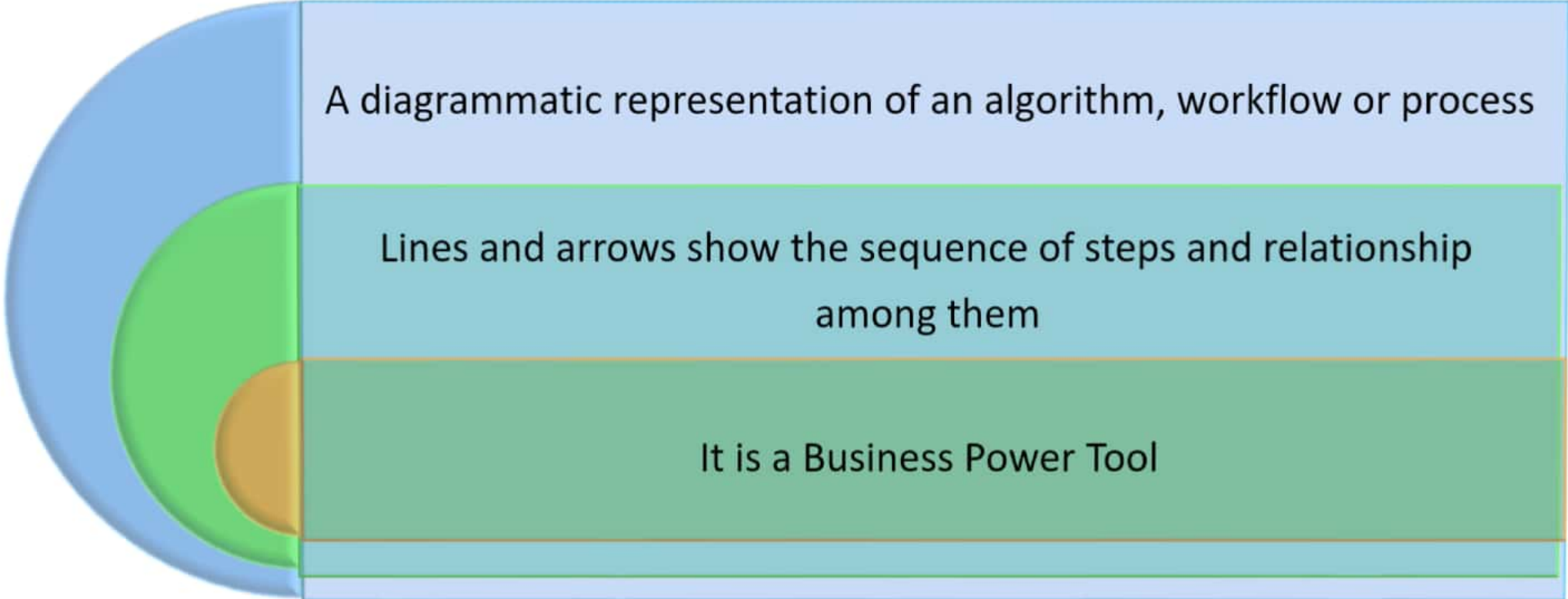
Written using simple statements in English

Have a definite beginning and a definite end, and a finite number of steps

Algorithm to multiply 2 numbers



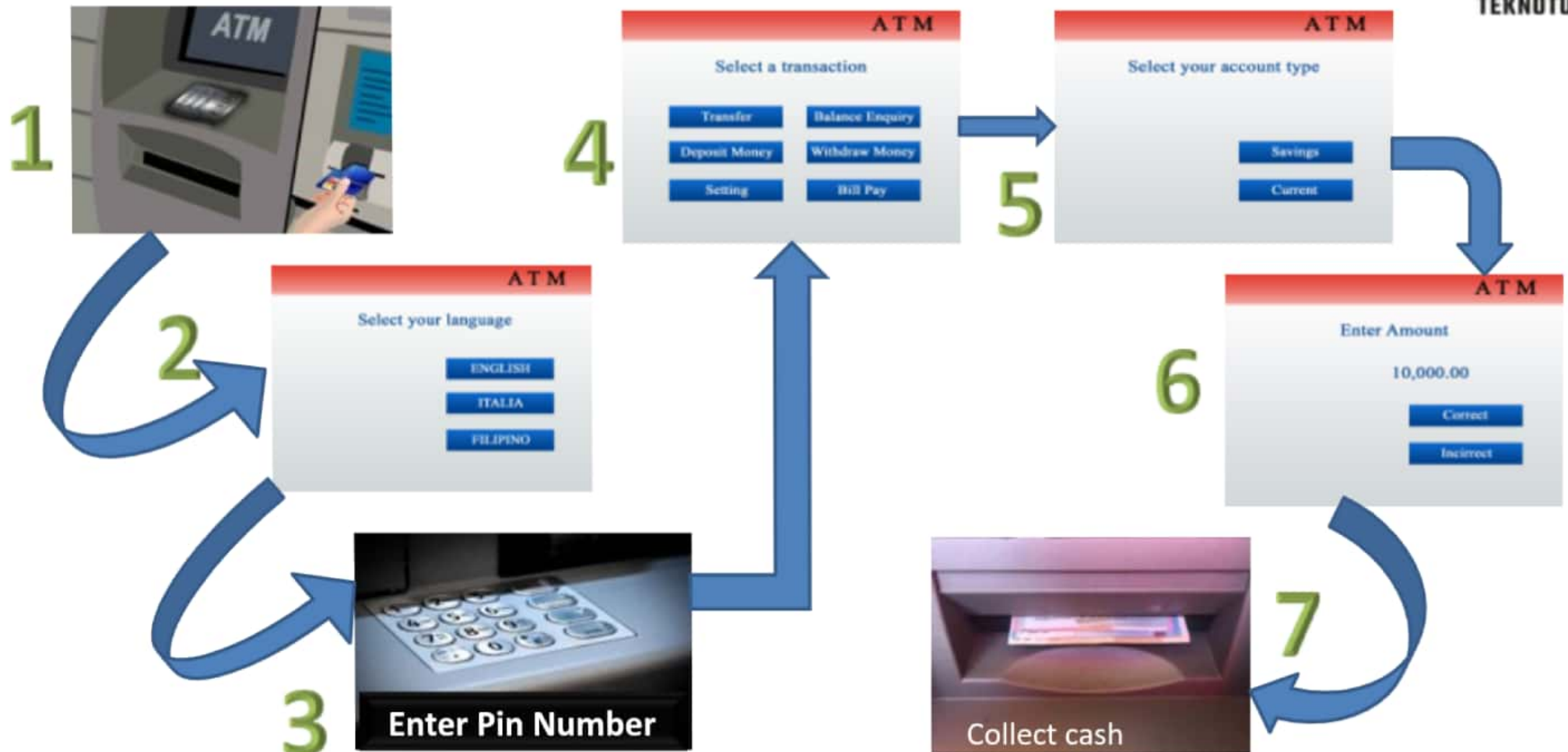
What is a Flowchart?



A diagrammatic representation of an algorithm, workflow or process

Lines and arrows show the sequence of steps and relationship among them

It is a Business Power Tool



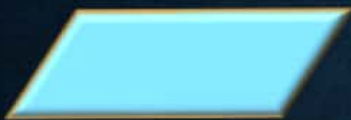
SYMBOLS IN FLOWCHART



START/END



CONNECTOR



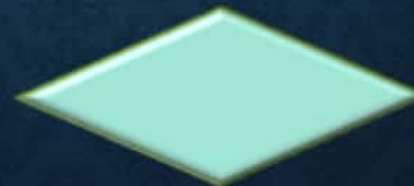
INPUT/OUTPUT



FLOW DIRECTION

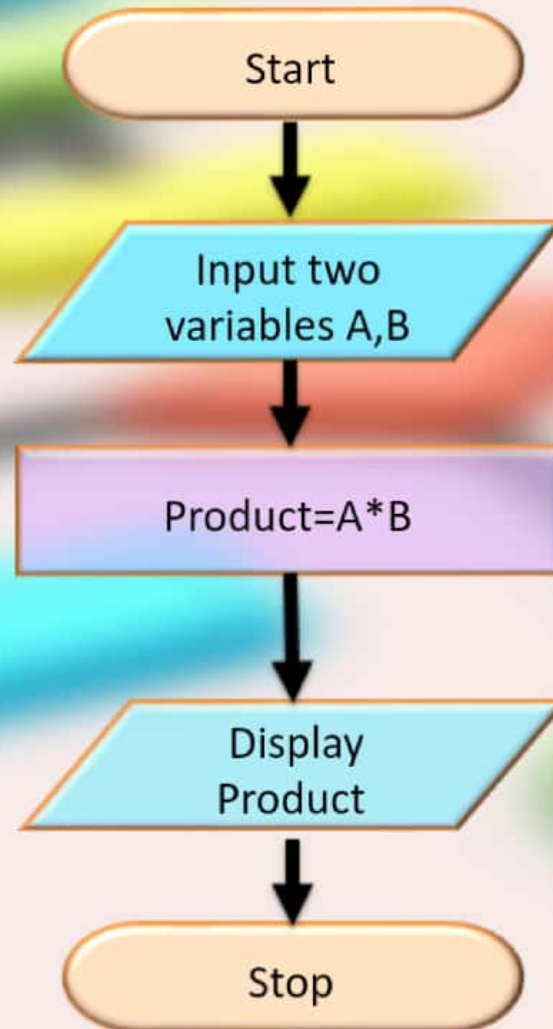


PROCESS



DECISION

Sample flowchart to multiply 2 numbers



Pseudocode:



It is an Informal way of describing algorithms

Does Not require any strict programming language syntax

It is Easily readable and modular form

Pseudocode to multiply two numbers



BEGIN

DECLARE variables number1, number2, Product

READ variables number1, number2

SET Product=number1*number2

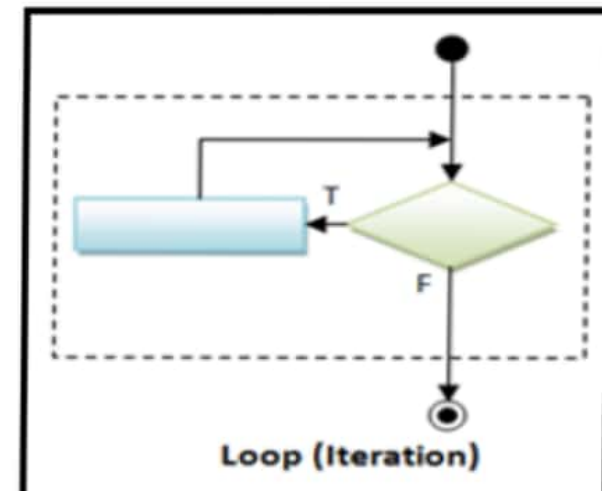
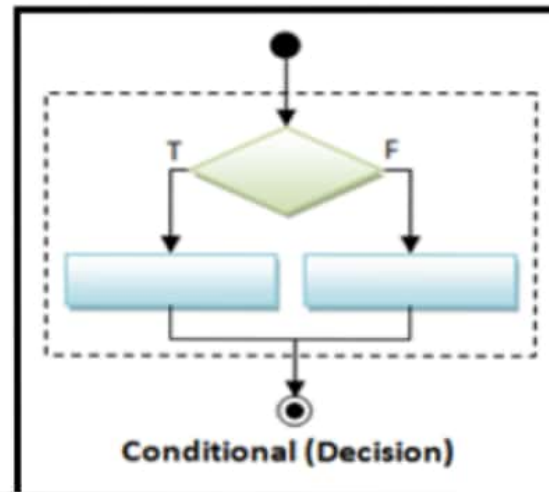
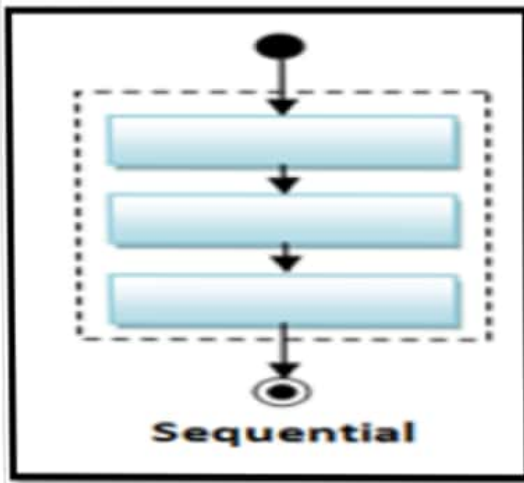
PRINT Product

END

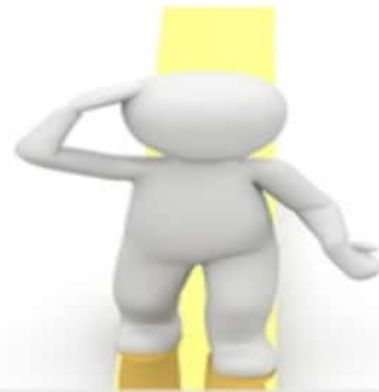
Flow of a program

Order in which the computer executes the statements in a program.

They can even be combined to deal with a given problem.



Selection or Conditional statements



It would be necessary to make a decision before arriving at a conclusion or to go on to the next step of processing in many situations.

Selection or Conditional statements

Step 1: Start the program

Step 2: Get the number

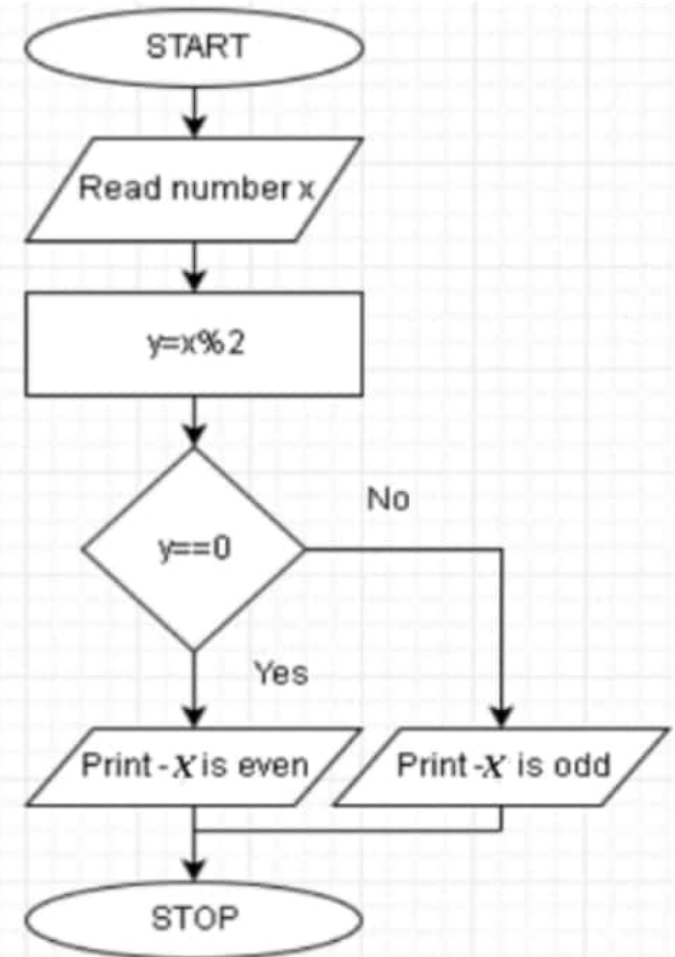
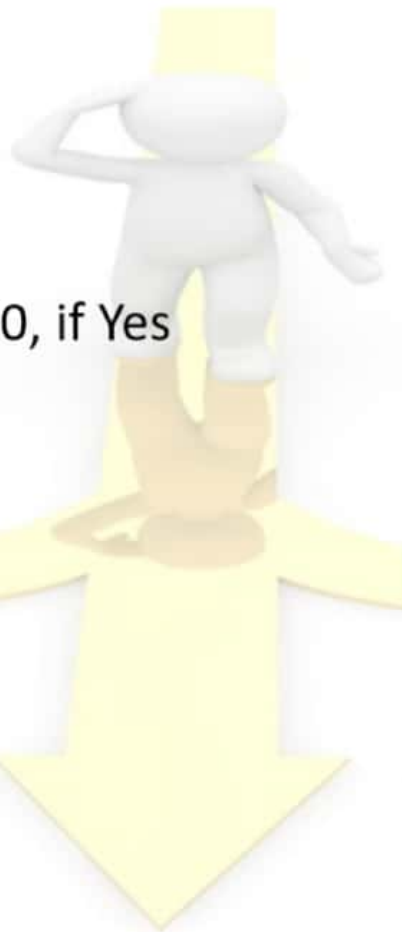
Step 3: check whether $\text{number} \% 2 == 0$, if Yes

GOTO step 4, if No GOTO step 5

Step 4: Display EVEN

Step 5: Display ODD

Step 6: Stop



Selection or Conditional statements

```
BEGIN
```

```
Declare variables number , result.
```

```
Input the number.
```

```
result = number% 2
```

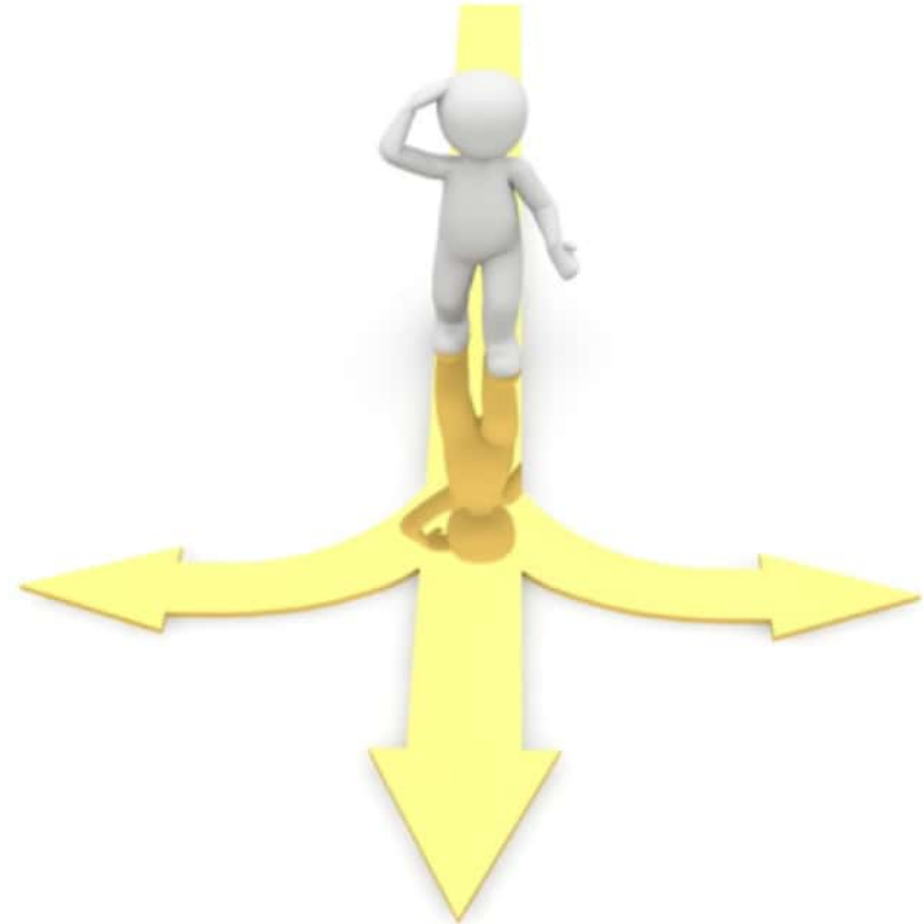
```
IF result= = 0 THEN
```

```
print number is even
```

```
ELSE print number is odd.
```

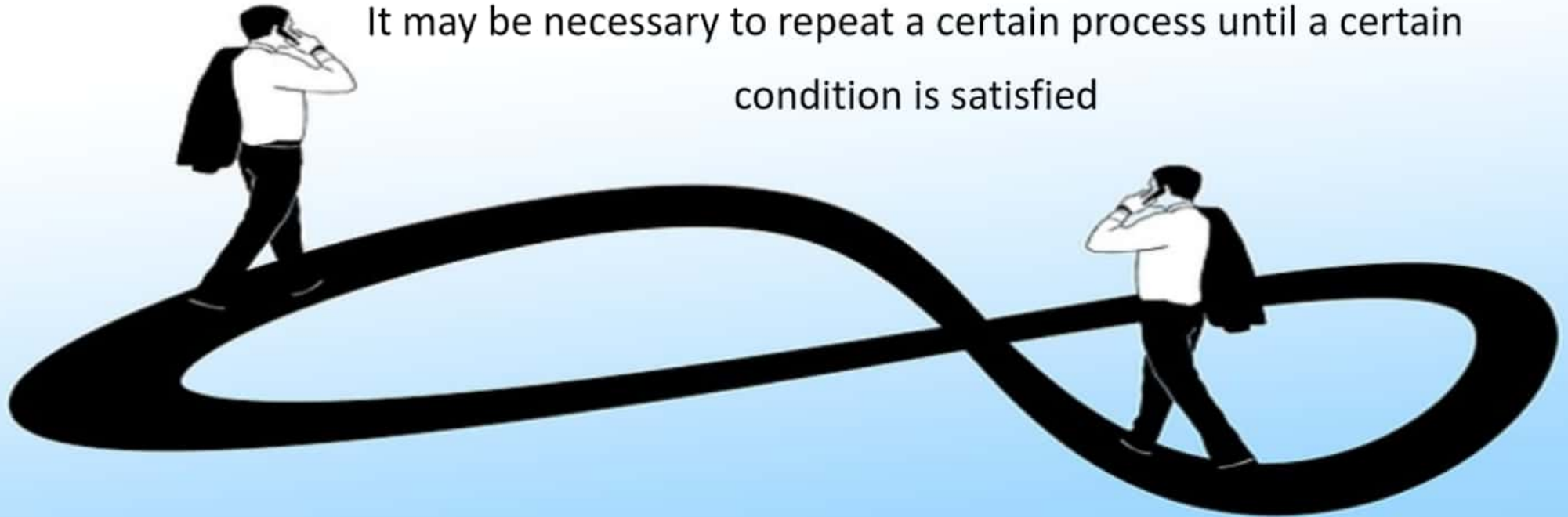
```
ENDIF
```

```
END
```



Looping or Iteration

It may be necessary to repeat a certain process until a certain condition is satisfied



Looping or Iteration

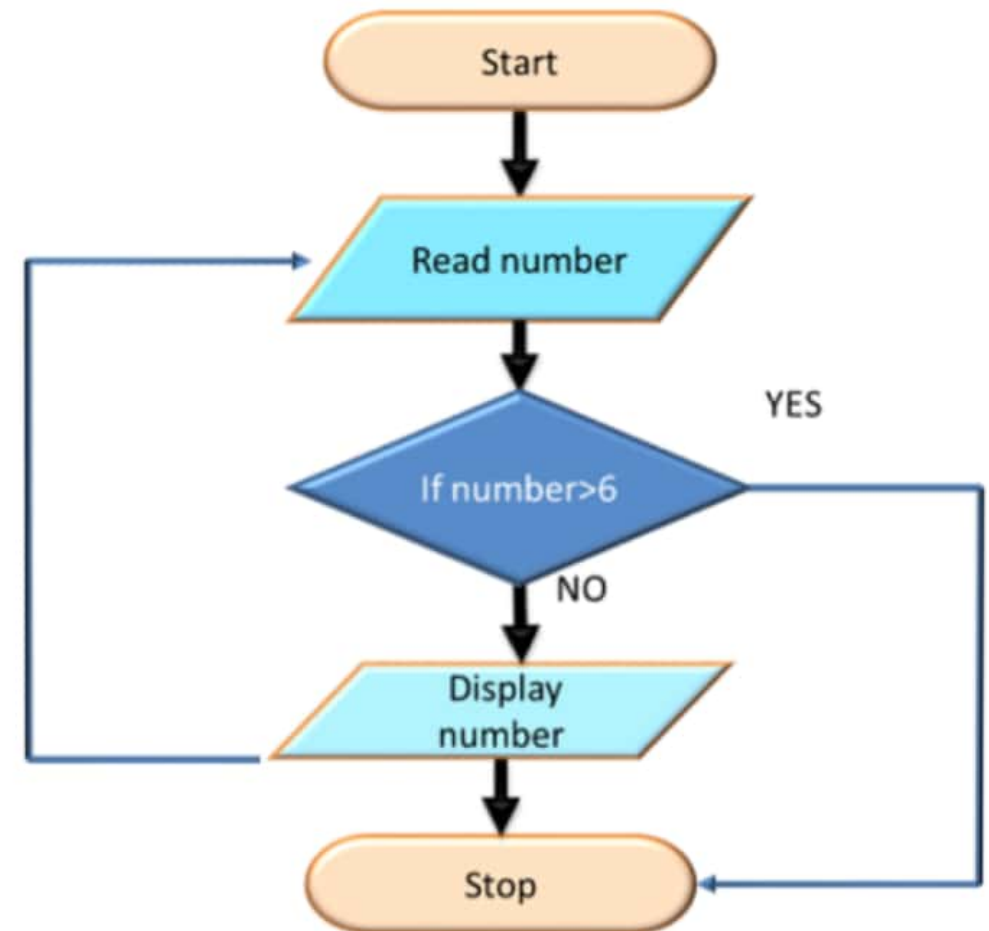
Step 1: Start the program

Step 2: Set the number

Step 3: Check Whether number > 6,
if Yes GOTO Step 5, if No GOTO Step 4

Step 4: Display number. GOTO Step 2

Step 5: Stop



Looping or Iteration

```
BEGIN  
DECLARE number.  
READ number.  
WHILE number <6  
PRINT number  
READ number  
END WHILE  
END
```



Looping structures



Initial conditions that need to be applied before the loop begins to execute

The invariant relation that must be applied after each iteration of the loop

The condition under which the iterative process must be terminated

Order the algorithm to find the modulo of any two given numbers

1. Declare 3 variables – multiplier & multiplicand and resultant_modulo

2. Read the values of multiplier and multiplicand

3. $\text{resultant_modulo} = \text{multiplier} \% \text{multiplicand}$

4. Display the resultant_modulo

Correct 

That's right! You chose the correct response.

Map the symbols n flowchart to its appropriate functionality

1. Parallelogram



1. Input/Output

2. Diamond



2. Decision making

3. Rectangle



3. Process

Correct



That's right! You chose the correct response.

Select one or more correct answers:

Expression is a combination of

- Variables
- operators
- constants
- functions
- keywords
- datatypes

Correct

That's right! You chose the correct response.

_____ is a step by step procedure to solve any problem.

- Pseudocode
- Algorithm
- Data Structure
- Flowchart

Correct

That's right! You chose the correct response.

A computer program must either use conditional statements or looping statements or sequential statements to solve a problem. All of them must not appear in the same program. State true/false.

True

False

Correct

That's right! You chose the correct response.

Introduction to Arrays

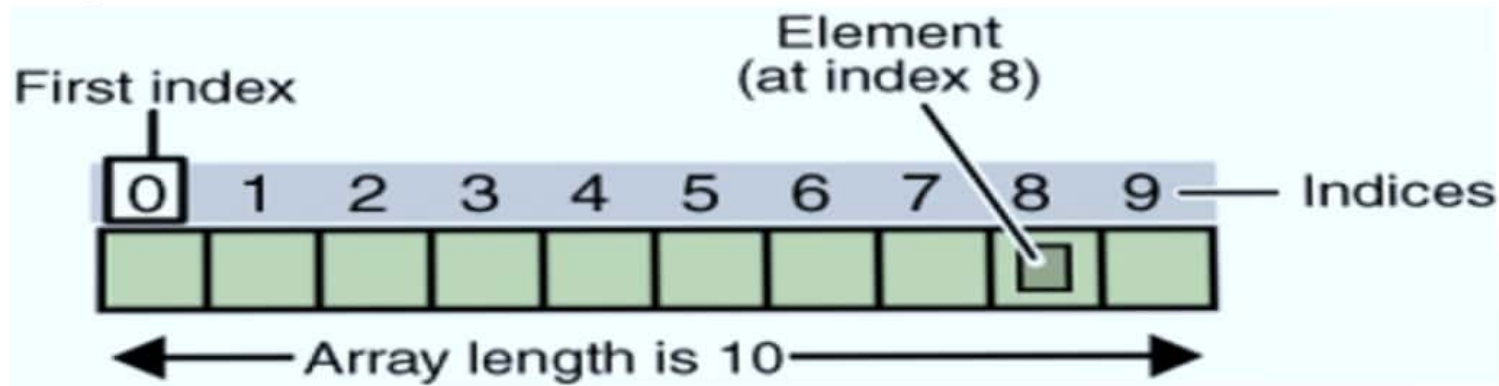


An array is a variable name which is used to store large amount of data

An array can be considered as a container with equally spaced slots, where the data could be stored

Data would be stored continuously, and the type of data would be the same

1-D Array



- All locations in an array are numbered from 0..n-1, if n is the total number of elements.
To access the location 1, we would use the notation mark[0]
- To store a value in the 1st position, the assignment would be marks[0]=93
- marks[4]=95 or
- marks[3]=mark[4] //Expression within [] should always resolve to an integer

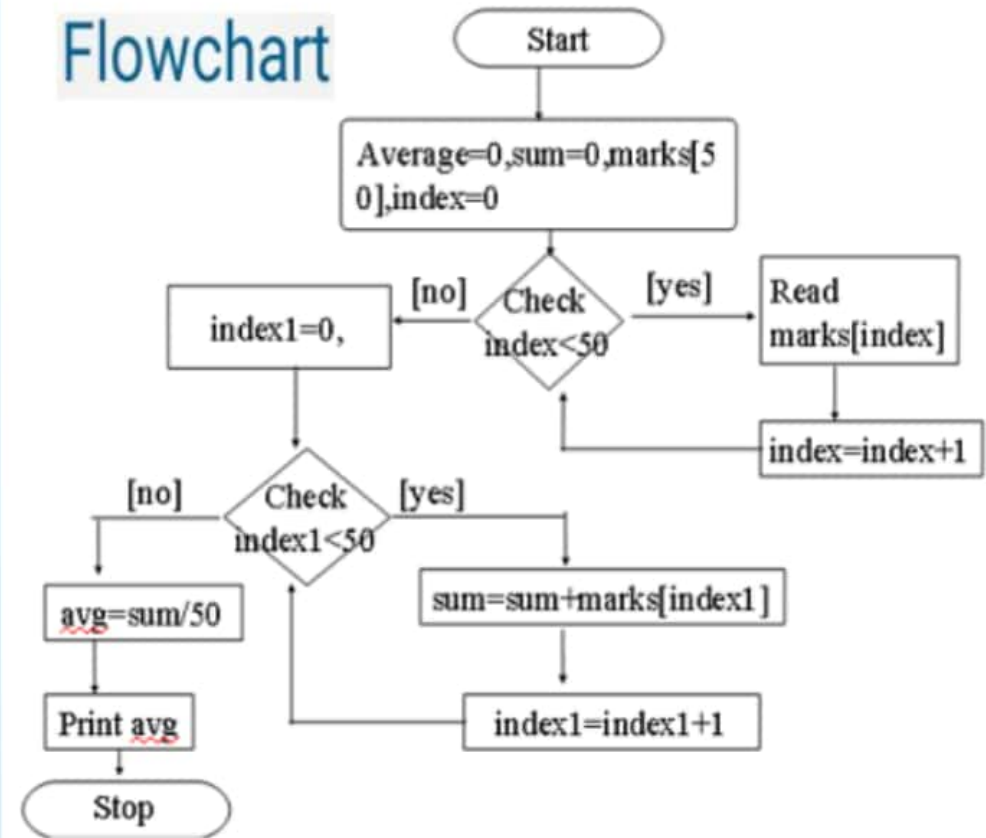
1-D Array

To read the marks of 50 students and print the average. Assumption: marks is an array of 50 marks

ALGORITHM

- Step 1. index=0
- Step 2. if index==50 then goto 6
- Step 3. read marks[index]
- Step 4. index=index +1
- Step 5. goto 2
- Step 6. index=0, avg=0, sum=0
- Step 7. if index==50 then goto 11
- Step 8. sum =sum + mark[index]
- Step 9. index=index+1
- Step 10. goto 7
- Step 11. avg=sum/50
- Step 12. print avg.
- Step 13. stop

Flowchart



2-D Array



The two-dimensional array is treated like a matrix.
The values are arranged in rows and columns.

	0	1	2	3
0	45	22	-5	11
1	-1	45	89	23
2	0	34	90	76

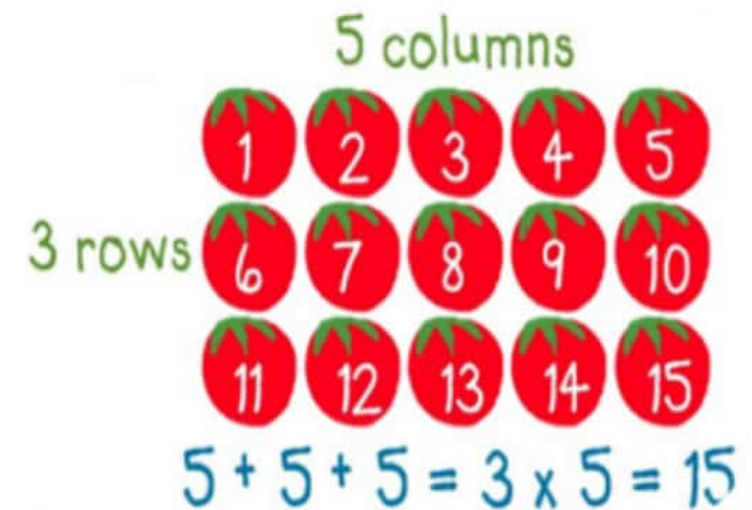
An array which has 3 rows and 4 columns

2-D Array



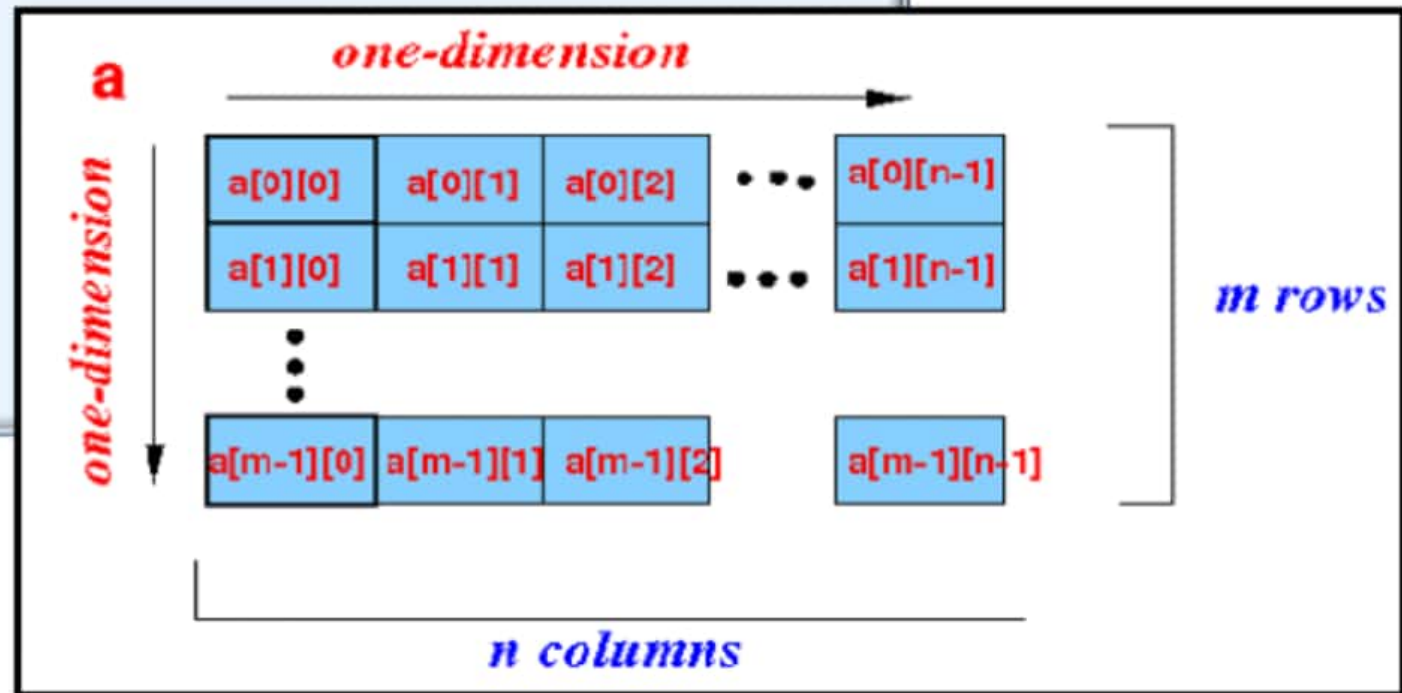
Pseudo code for reading the marks of 3 students for
5 subjects and printing the average of each student

```
BEGIN  
DECLARE student_marks[3][5]  
SET average TO 0, sum TO 0  
FOR student_index=0 TO 3  
FOR mark_index=0 TO 5  
INPUT mark  
SET student_marks[student_index][mark_index] TO mark  
NEXT mark_index  
NEXT student_index
```

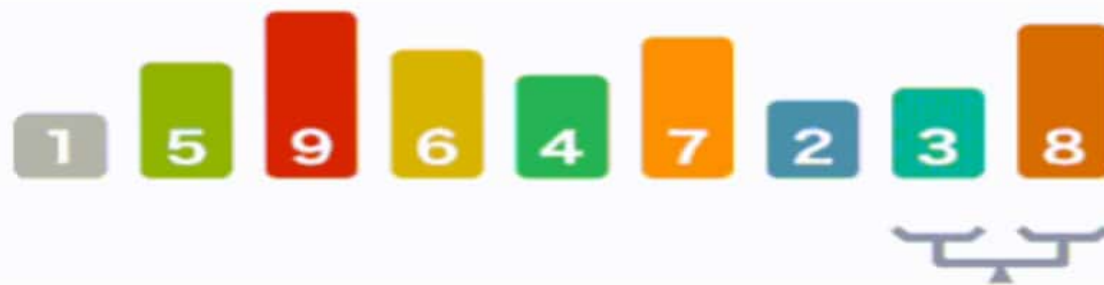
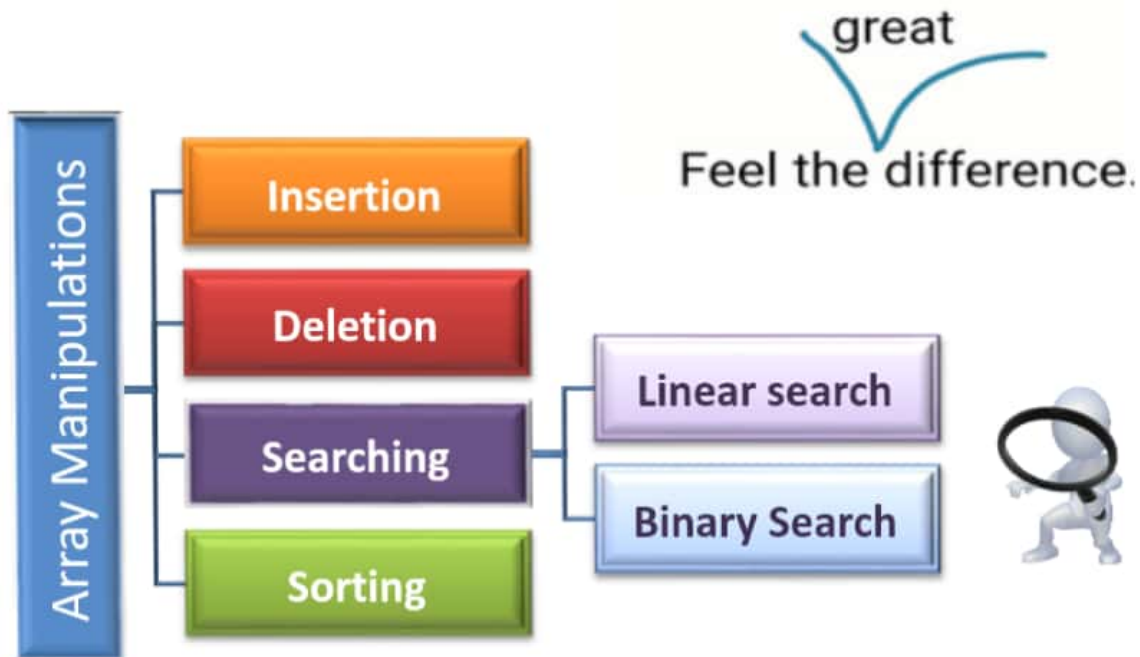


(contd...)

```
FOR student_index= 0 TO 3
FOR mark_index=0 TO 5
SET sum TO sum+ student_marks[student_index][mark_index]
NEXT mark_index
SET average TO sum/3
PRINT average
NEXT student_index
END
```



Manipulating Arrays

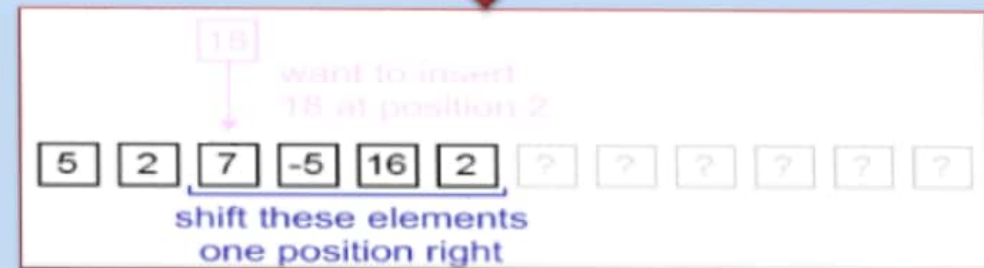
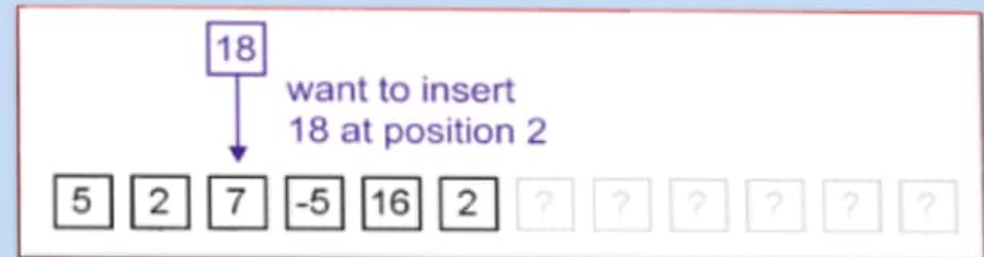


Insertion Algorithm

INSERT (A, N, K, ITEM)

Here A is a linear array with N elements and K is a positive integer such that $K \leq N$. This algorithm inserts an element ITEM into the Kth position in A.

1. [Initialize counter] Set $J = N$.
 2. Repeat steps 3 and 4 while $J \geq K$.
 3. [Move Jth element downward] Set $A[J + 1] = A[J]$
 4. [Decrease counter] Set $J = J - 1$
- [End of step 2 loop]
5. [Insert element] Set $A[K] = \text{ITEM}$
 6. [Reset N] Set $N = N + 1$
 7. Exit



Deletion Algorithm

DELETE (A, N, K, ITEM)

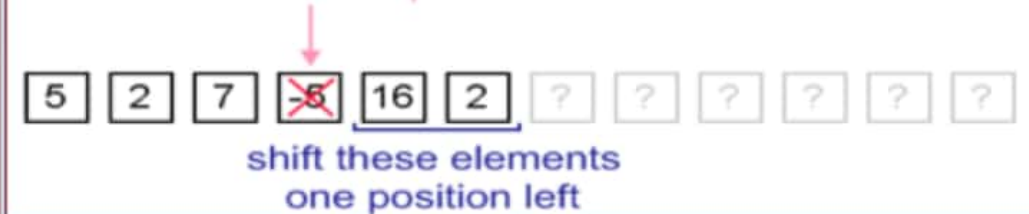
Here A is a linear array with N elements and K is a positive integer such that $K \leq N$. This algorithm deletes the Kth element from A.

1. Set ITEM = A[K]
2. Repeat from J = K to N - 1
 [Move J + 1st element upward] Set A[J] = A[J + 1]
 [End of loop]
3. [Reset the number N of elements in A] Set N = N - 1
4. Exit

want to remove
element at the position 3



want to remove
element at the position 3



element has been
successfully removed



Linear Search Algorithm

If an element is to be retrieved from a specific location in an array, the array has to be traversed from the first position until the element is found.

Step 1 : flag = notfound , l=0
Step 2 : If $l > n - 1$ and flag = found
 Go to 5
Step 3 : If $array[l] = element$
 flag = found
Step 4 : $l = l + 1$
Step 5 : if (flag == found)
 Print element found at position $l+1$
 Else
 Print element not found
Step 6 : Stop

Key	List
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8

Binary Search Algorithm



Searching technique for an element from a sorted array

Divide the array in two parts by finding the middle value in iteration

If the value of the search element is lesser than the item in the middle of this interval, then split the left interval in half. Otherwise split the right interval in half.

Search for the element in the new interval and continue doing this until the search value is found or until the interval is not found to contain the search value.

ALGORITHM

1. Initialize low =0, high = n-1
2. While low <= high
3. $mid = \lfloor (low+high)/2 \rfloor$
4. If $a[mid] == Item$
5. Set Pos = mid
6. Break and Jump to step 10
7. Else if $Item < a[mid]$
8. high = mid -1
9. Else low =mid +1
10. If Pos < 0
11. Print "Element is not found"
12. Else Print Pos

Binary Search

SEARCH FOR THE ELEMENT 76

↓ 47 IS THE MIDDLE ELEMENT $47 < 76$

2	9	11	15	28	33	40	47	51	64	76	77	82	85	94
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----

SEARCH FOR 76 IN THE RIGHT SUB ARRAY
77 IS THE MIDDLE ELEMENT $77 > 76$

2	9	11	15	28	33	40	47	51	64	76	77	82	85	94
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----

SEARCH FOR 76 IN THE LEFT SUB ARRAY
64 IS THE MIDDLE ELEMENT $64 < 76$

2	9	11	15	28	33	40	47	51	64	76	77	82	85	94
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----

SEARCH FOR 76 IN THE RIGHT SUB ARRAY
SEARCH ELEMENT 76 IS FOUND

2	9	11	15	28	33	40	47	51	64	76	77	82	85	94
---	---	----	----	----	----	----	----	----	----	----	----	----	----	----

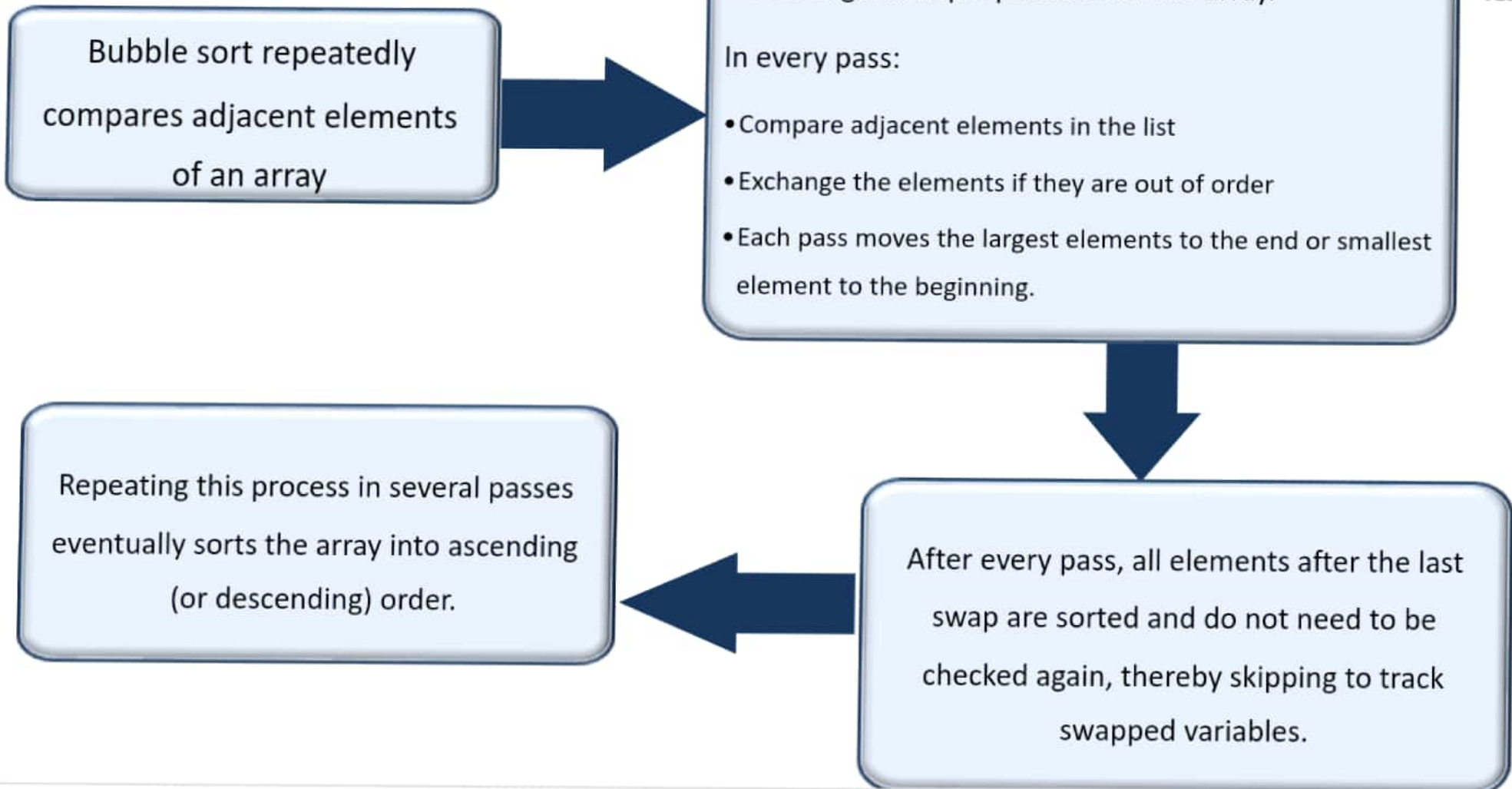
Sorting Algorithms

Sorting is the process of placing elements from a collection in some kind of order

- Ordering of data allows for easy and faster access of data
- Most common and efficient algorithms are:
 - Bubble Sort
 - Insertion Sort
 - Selection Sort
 - Exchange Sort



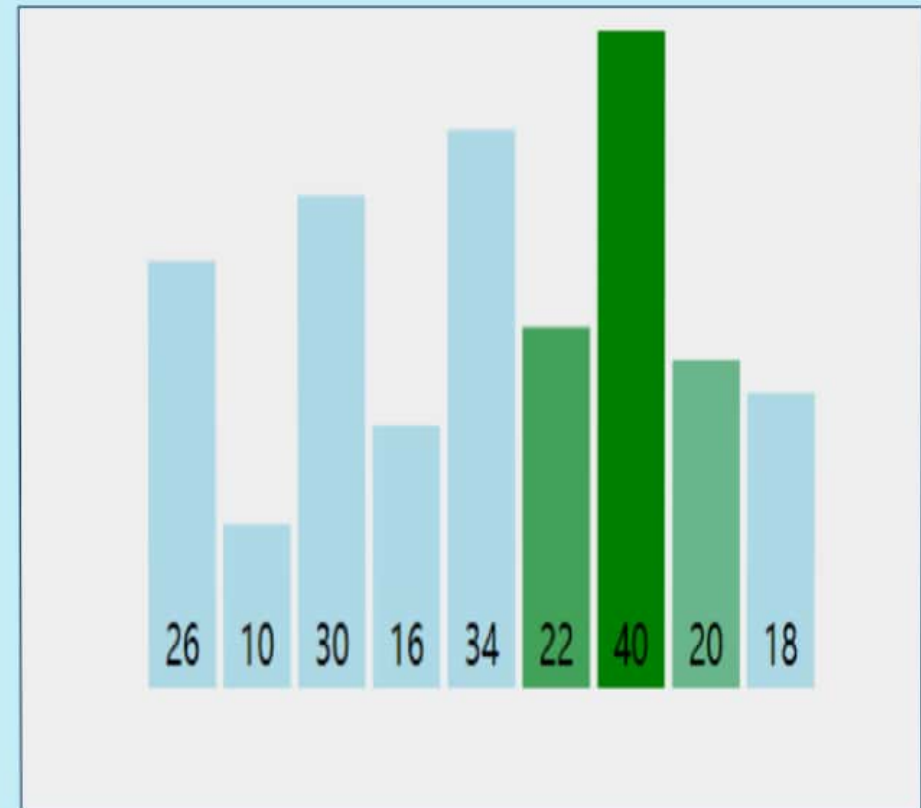
Bubble Sort - Activities



Bubble Sort - Algorithm

Let A be a linear array of n numbers. Swap is a temporary variable for swapping (or interchange) the position of the numbers

1. Input n numbers of an array A
2. Initialize $i = 0$ and repeat through step 4 if $(i < n)$
3. Initialize $j = 0$ and repeat through step 4 if $(j < n - i - 1)$
4. If $(A[j] > A[j + 1])$
 - (a) $\text{Swap} = A[j]$
 - (b) $A[j] = A[j + 1]$
 - (c) $A[j + 1] = \text{Swap}$
5. Display the sorted numbers of array A
6. Exit.



When we write `num[100] = 99`.

How many elements can be stored inside the array variable `num`?

- The statement gives no clue about the number of elements can be stored
- Infinite number of elements
- 99
- 100

Correct

That's right! You chose the correct response.

Negative elements can be placed inside the array. State true / false

-  True
- False

Correct

That's right! You chose the correct response.

Which of the following are False with respect to the manipulation of arrays?

- It is possible to sort the elements of an array
- It is possible to increase the size of the array
- An array can store heterogeneous data
- An array can store homogenous data.

Correct

That's right! You chose the correct response.

Rearrange the algorithm to obtain and display a name, in correct order:

1. DECLARE names[20]

2. INPUT name

3. PRINT name

Correct 

That's right! You chose the correct response.

Map the scenario to its appropriate array type

Matrix multiplication **2D ARRAY**

To create a list of all prime numbers below 100 **1D ARRAY**

To store 5 marks of 3 students **2D ARRAY**

Correct

That's right! You chose the correct response.

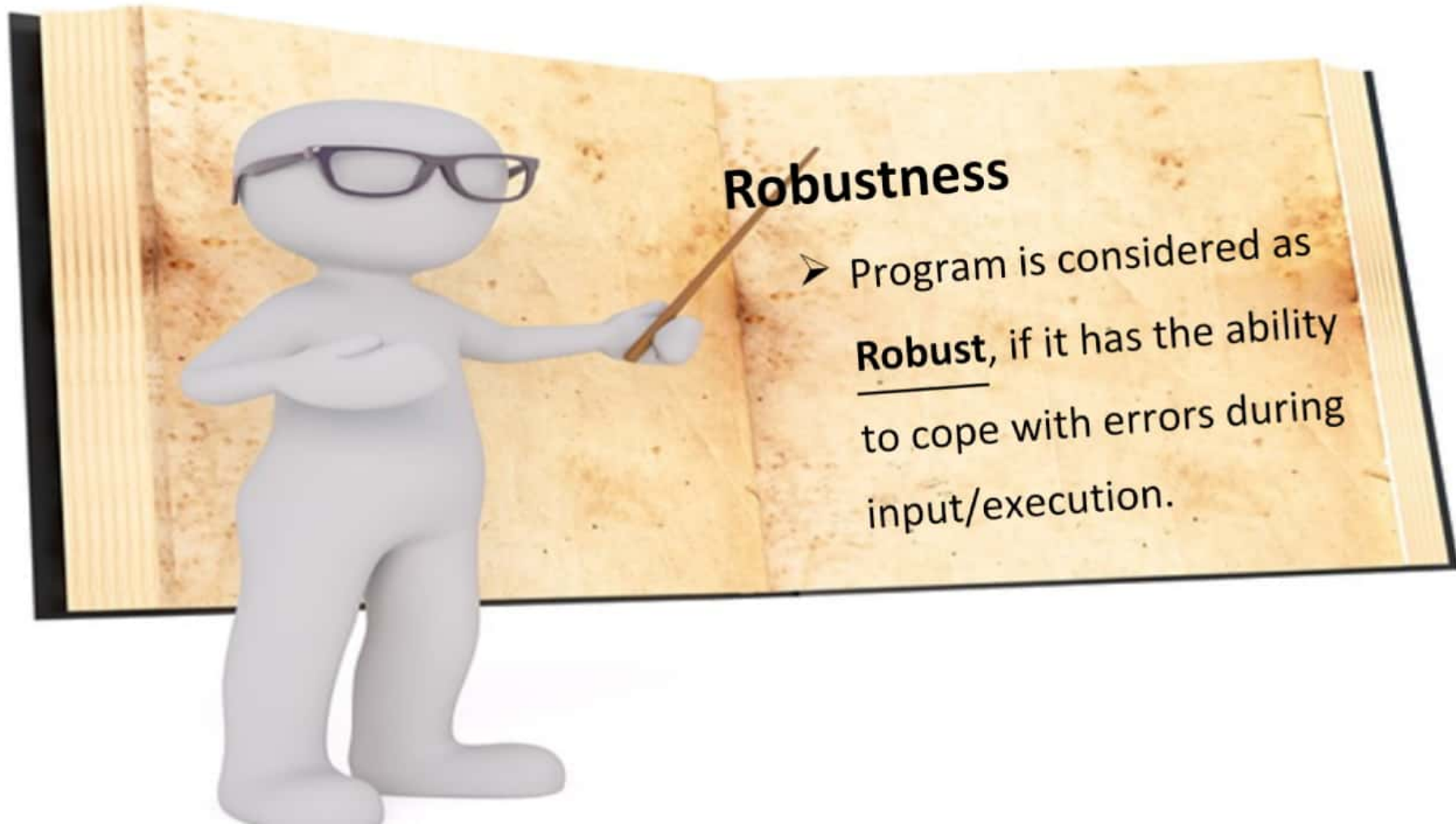
GOOD PROGRAMMING PRACTICES



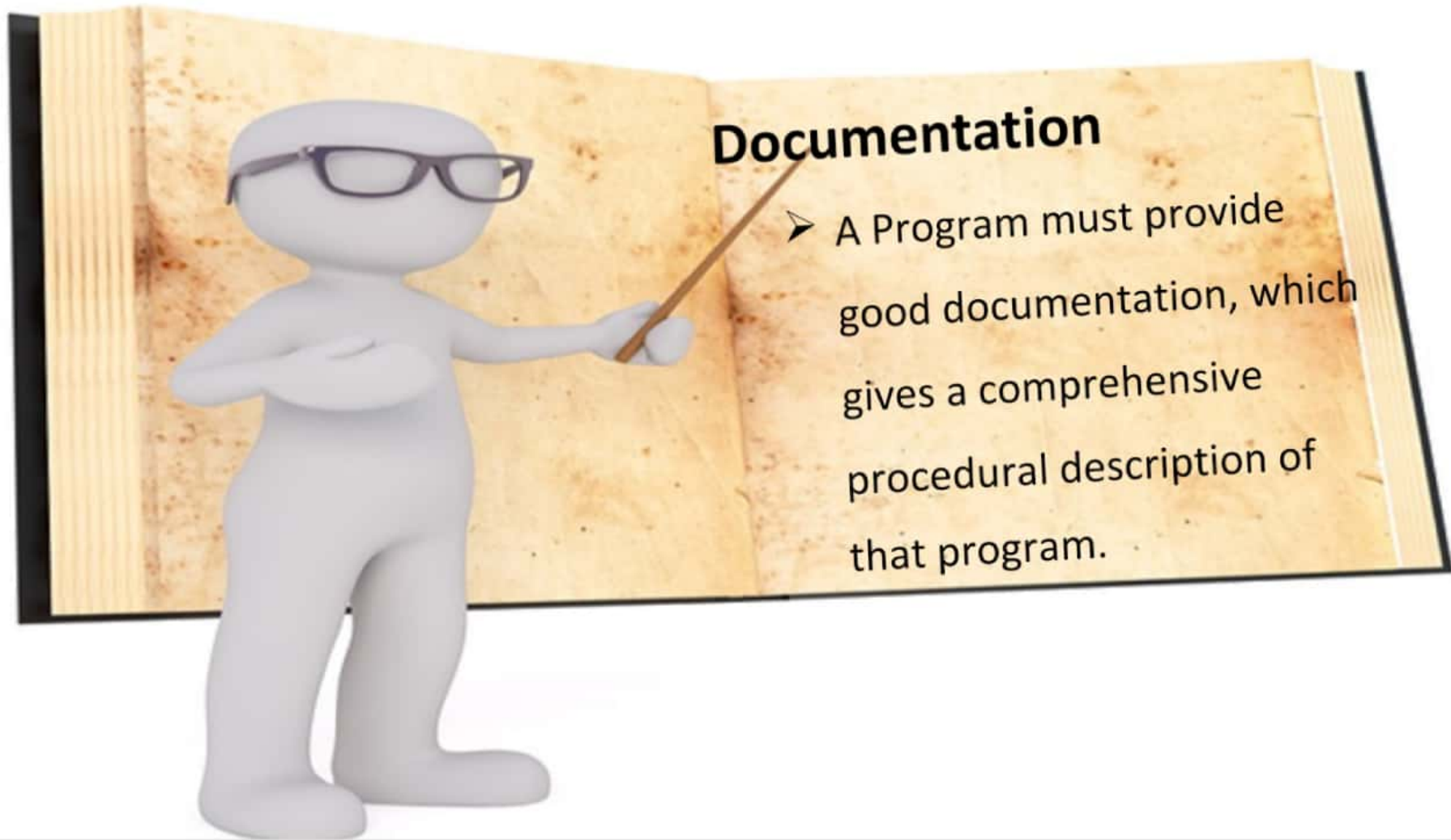
- ✓ Meaningful variable names that help readability of our code
- ✓ Liberal use of comments that helps the user to understand what a program does and why
- ✓ Avoid obvious comments if the code is already readable
- ✓ Keep your code simple. Simpler the code, lesser the bugs
- ✓ Use indentation for better understanding



CHARACTERISTICS OF A GOOD PROGRAM



CHARACTERISTICS OF A GOOD PROGRAM



Correctness Vs. robustness



Here, your program is correct, but not robust. It is not handling unexpected situations. You need to modify your program to support robustness.

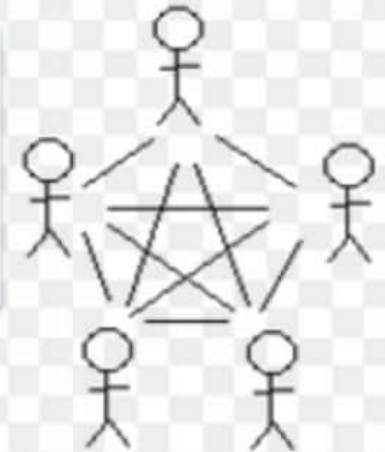
If your program supports robustness, then it should accept only numbers. Otherwise the output should be: "Only Numbers are allowed".

Coupling and Cohesion

- Modules in themselves are not “good” – We must design them to have good properties
- Properties of good Design
 - Component independence
 - Fault prevention and fault tolerance
 - Design for change

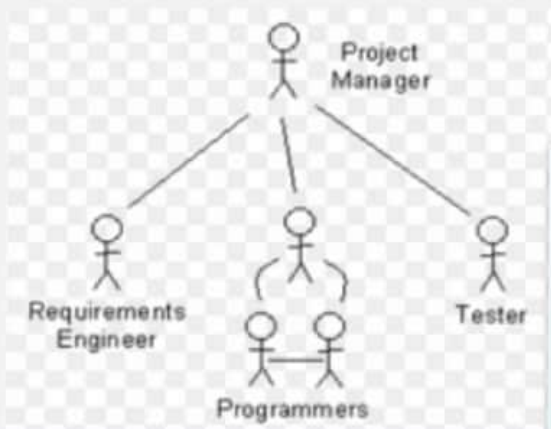
With poor module design:

- Hard to understand
- Hard to locate faults
- Difficult to extend or enhance



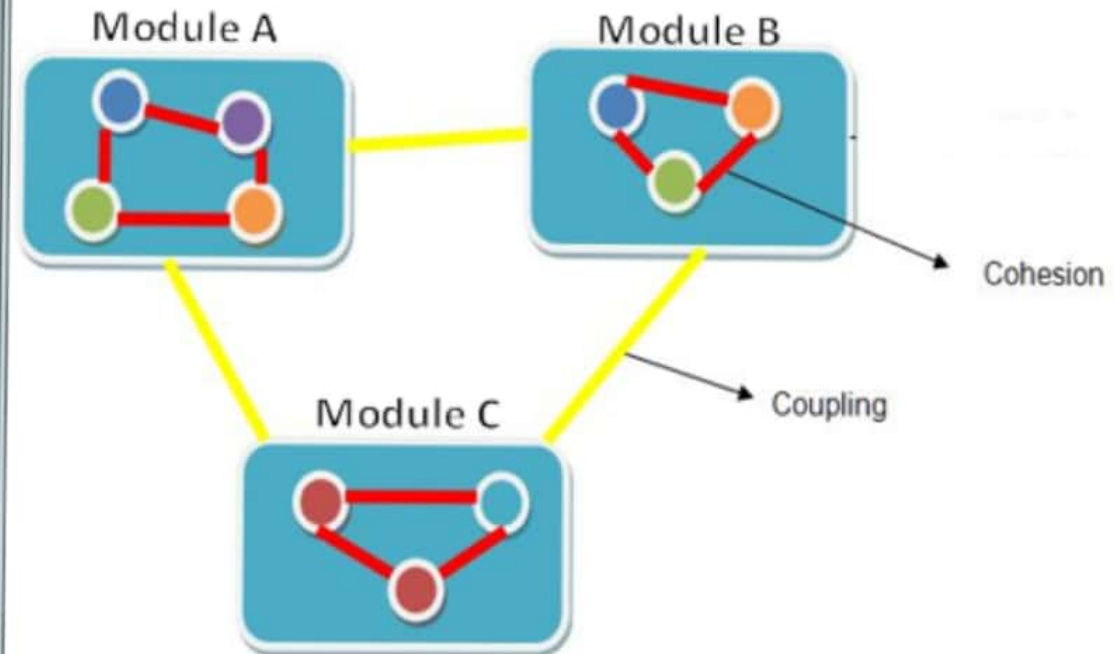
With good module design:

- Maximal relationships within modules (cohesion)
- Minimal relationships between modules (coupling)
- This is the main contribution of structured design



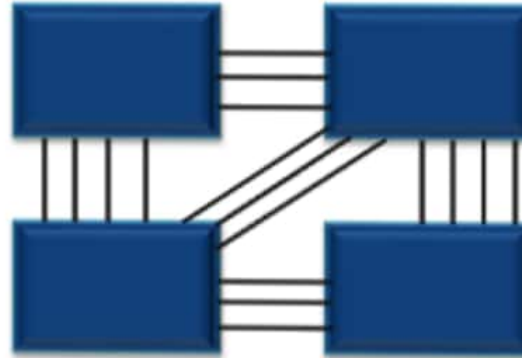
Coupling and Cohesion

- The goal of design is to divide the system into modules and assign responsibilities among the components so that they have
 - High cohesion within the modules
 - Loose coupling between modules
- The principle of coupling and cohesion are the most important design principles



Coupling – Degree of interaction between two modules

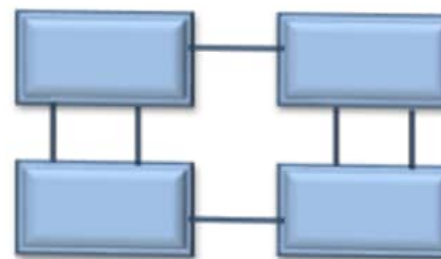
Two modules are tightly coupled when they depend a great deal on each other



Tightly coupled - many dependencies

Uncoupled modules have no interconnections at all; they are completely unrelated

Loosely coupled modules have some dependence, but their interconnections are weak

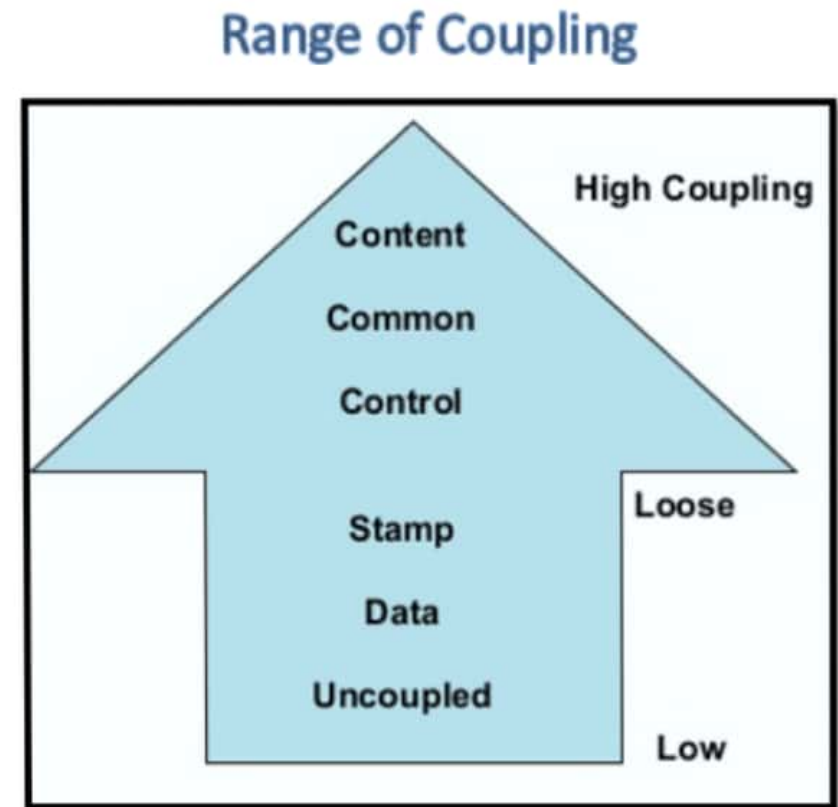
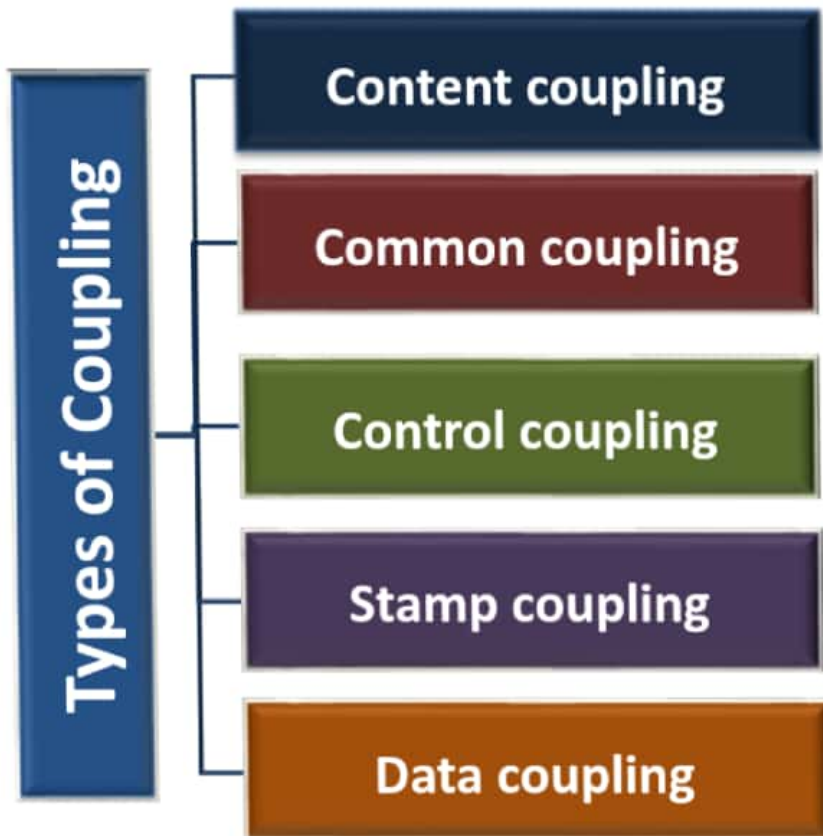


Loosely coupled - some dependencies



Uncoupled - no dependencies

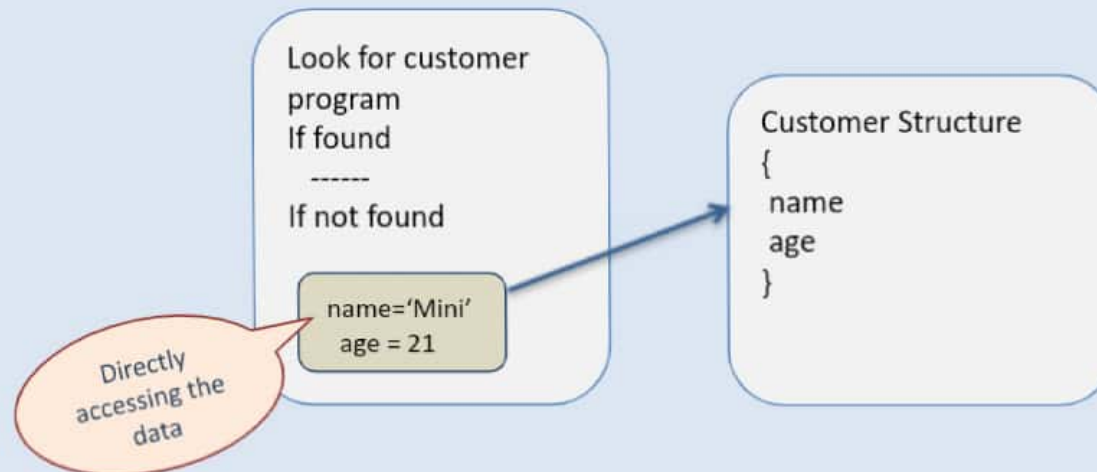
Types of coupling



Types of coupling

Content coupling

- Occurs when one component modifies an internal data item in another component, or when one component branches into the middle of another component

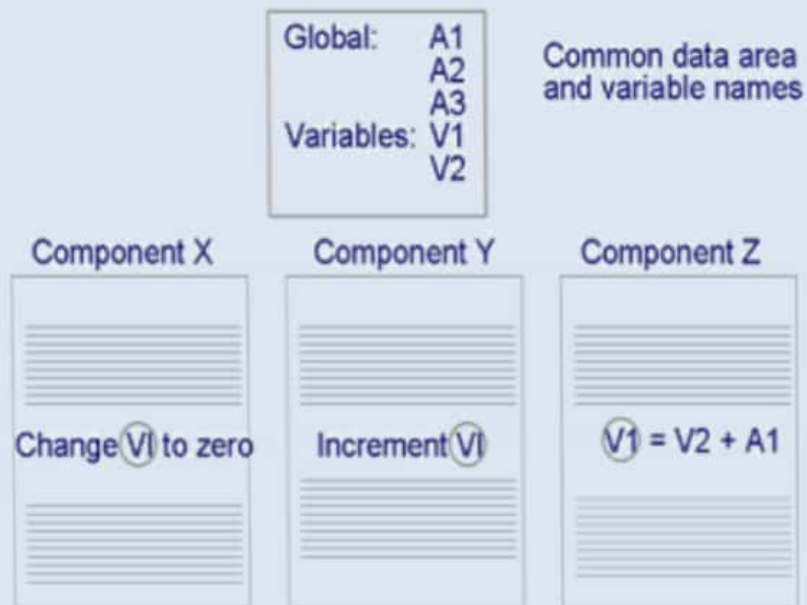


- To reduce content coupling
 - Hide the data so that it can be accessed only by calling the method that can access or modify the data

Types of Coupling

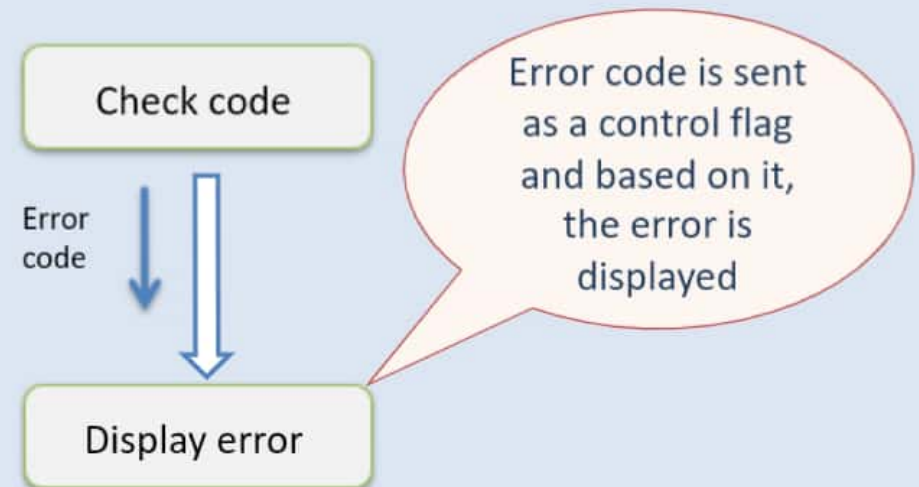
Common Coupling

- Two modules have write access to the same global data



Control coupling

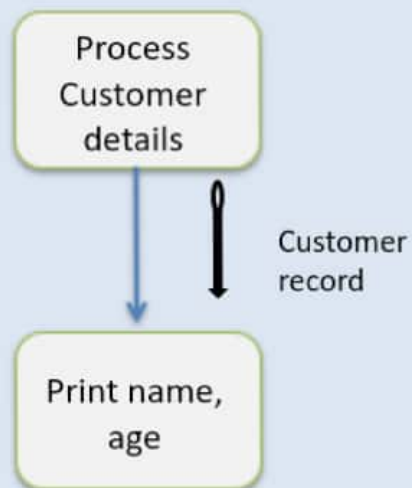
- One module passes an element of control to the other
- It is impossible for the controlled module to function without some direction from the controlling module



Types of Coupling

Stamp Coupling

- Data structure is passed as parameter, but the called module operates on only some of individual components



Data Coupling

- Every argument is either a simple argument or a data structure in which all elements are used by the called module



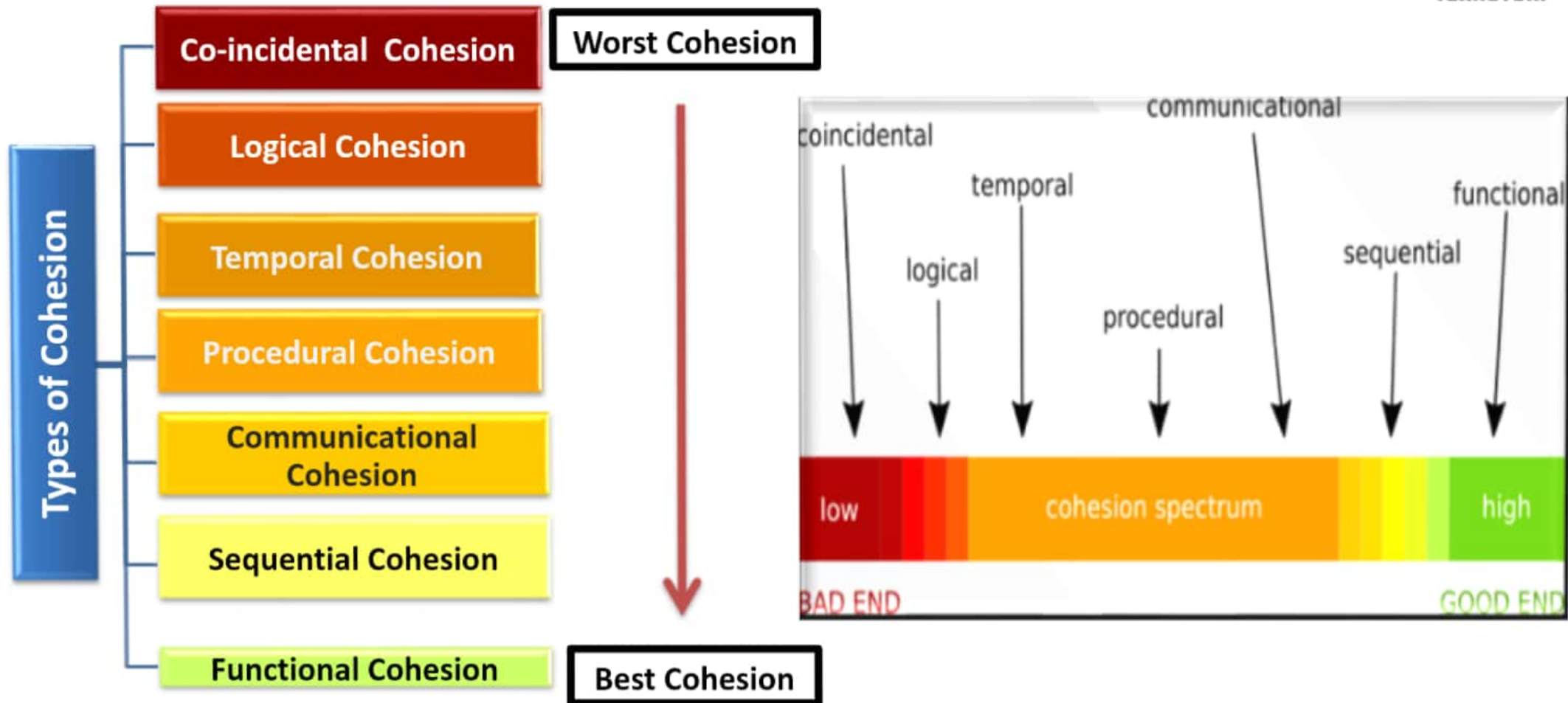
Cohesion

Cohesion refers to the dependence within and among a module's internal elements (e.g., data, functions, internal modules)

Greater the cohesion, the better is the program design



Types of Cohesion



Types of Cohesion

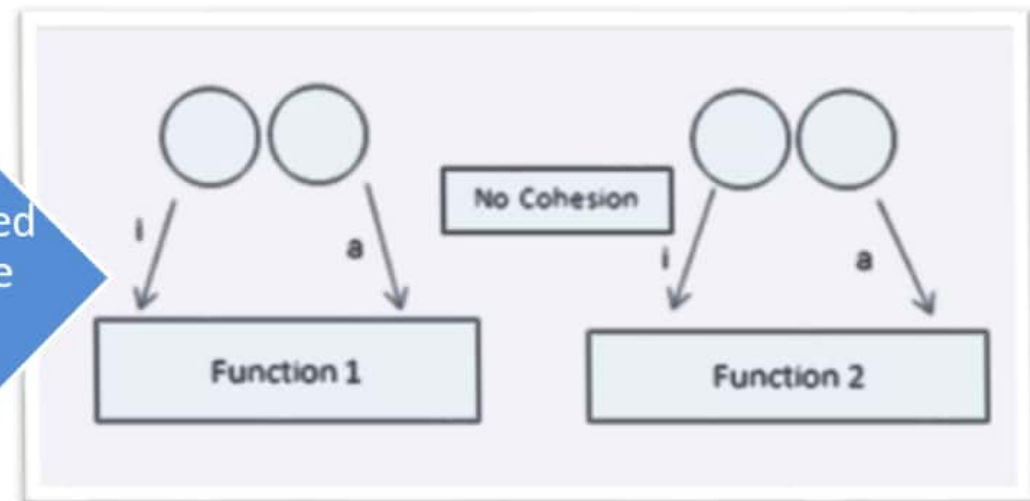
Co-incidental Cohesion

Module performs multiple, completely unrelated actions

Module 1

- Process customer details
- Calculate total sales
- Perform transaction deletion
- Read sales details

Has unrelated tasks inside module



Types of Cohesion

Logical Cohesion

Here, elements perform similar tasks and the activities to be executed are chosen from outside the module.

Operations are related, but the functions are significantly different.

Module

```
Module(data[] , type)
  if type is bar
    display as bar chart
  else if type is pie-chart
    display as pie-chart
  else if type is graph
    display as graph
  .....
```

Task is the same which is "display", but the 'how to be displayed' is decided by the calling function.

Temporal Cohesion

Module's data and functions are related because they are used at the same time in an execution. Elements are grouped by when they are processed.

Process Error Module

```
processError()
  Release the database connection
  Open error file
  Write error log
  Send error message to user
```

All the activities of the module are performed when an error occurs in the software.

Types of Cohesion

Procedural Cohesion

Similar to temporal, and functions pertain to some related action or purpose

Module

processCustomer()
create customer id
add new record to customer file
display customer id to user

All the activities of the module related to creating a customer record

Communicational Cohesion

Module which has activities executed sequentially and work on same data

Module

processCustomer(customer id)
get customer details
get customer purchaser details
display customer id, customer name, purchase items

All the activities of the module act on the customer id

Types of Cohesion



Sequential Cohesion

Elements are involved in activities such that output data from one activity becomes input data to the next activity

CalculateGrade Module

```
CalculateGrade(marks) {  
  calculateSum  
  ↓      sum  
  calculateAverage  
  ↓      average  
  calculateGrade  
}
```

The output of calculateSum is given as input to calculateAverage

Functional Cohesion

Functionally cohesive module performs exactly one action.
Highly recommended Cohesion

Example: `convertCelciusToFarenheit`

Advantages

- More reusable
- Easier corrective maintenance
 - Fault isolation
 - Reduced regression faults
- Easier to extend product

Choose the correct answer in each drop-down list:

The goal of design must have

coupling between modules

High

cohesion within the modules and

Loose

Correct

That's right! You chose the correct response.

Choose the correct answer in each drop-down list:

Degree of interaction between 2 modules is called and the Degree of interaction within the elements of a module is called

Correct 

That's right! You chose the correct response.

When two modules have write access to the same global data, it is called Content coupling.
State true/false.

True

False

Correct

That's right! You chose the correct response.

What kind of coupling is it when one module passes an element of control to the other?

- Control Coupling
- Content coupling
- Stamp coupling
- Common coupling

Correct

That's right! You chose the correct response.

Which of the following are the ways in which modules can be dependent on each other?

- The amount of data passed from one module to another
- The amount of control that one module has over the other
- The references made from one module to another
- None of the options

Correct

That's right! You chose the correct response.

Summary

- How to write Algorithms , flow charts and Pseudo Code
 - Selection and Iterative statements
- Introduction to Arrays
 - 1-Dimension and 2-Dimensional arrays
 - Algorithm for manipulating arrays
- Characteristics of a good program
- Difference between correctness and robustness
- Coupling and Cohesion





THANK YOU