Computational Thinking

23Z104

Syllabus

Introduction: Computational thinking - Logical thinking - Flow charts - Algorithmic thinking - Characteristics of algorithms – Pseudo code - Example problems

Problem Solving and Decomposition: Defining the problem - Devising a solution - Decomposition - Other effective strategies - Patterns - Example problems

Abstraction and Modeling: Generalization - Abstraction - Modeling – Examples

Iterative Logic: Iterator - Variable - Filtering - Dynamic filtering - Example problems

Case Studies: Text processing - Pattern search - Linear search - Sorting

Books

TEXT BOOKS:

- Karl Beecher, "Computational Thinking: A Beginner's Guide to Problem Solving and programming", 1st Edition, BCS Learning & Development Limited, 2017.
- G Venkatesh and Madhavan Mukund, "Computational Thinking: A Primer for Programmers and Data Scientists", 1st Edition, Notion Press, 2021.

REFERENCES:

- R.G.Dromey, "How to Solve it by Computer", Pearson Education, Second Edition, 2008.
- Peter j Denning , Matti Tedre, "Computational Thinking", The MIT Press, 2019
- Anany Levitin, "Introduction to the Design and Analysis of Algorithms", Third Edition, Pearson Education, 2017
- Peter William Mcowan, Paul Curzon, "Power of Computational Thinking, The: Games, Magic And Puzzles To Help You Become A Computational Thinker ", World Scientific Europe Ltd, 2017

COURSE OUTCOMES

Upon completion of this course, the students will be able to

CO1: Understand and apply computational thinking for algorithm development

CO2: Devise a solution by using effective problem solving techniques like decomposition and pattern identification

CO3:Understand the concept of generalization, Abstraction and modeling the solution

CO4: Apply iterative logic for solving problems

CO5: Solve real world problems using computational resources efficiently

What is computational thinking?

- Computational thinking is the thought processes involved in
  - –formulating a problem and
  - –expressing its solution(s) in such a way that a computer— human or machine—can effectively carry out. (Wing 2014)

Core concepts of CT

1) Logical thinking
2) Algorithmic thinking
3) Decomposition
4) Pattern recognition and generalization
5) Abstraction
6) Modelling
7) Evaluation

Algorithm

- A finite set of steps that must be followed to solve any problem is called an **algorithm**.
- Algorithm is generally developed before the actual coding is done.
- It is written using English like language so that it is easily understandable even by non-programmers.
- Sometimes algorithms are written using **pseudocodes**, i.e. a language similar to the programming language to be used.

Advantages of Algorithm

- Promotes effective communication between team members
- Enables analysis of problem at hand
- Acts as blueprint for coding
- Assists in debugging
- Becomes part of software documentation for future reference during maintenance phase

characteristics of a good and correct algorithm

- It should terminate after a finite time.
- It should produce at least one output.
- It should take zero or more input.
- It should be deterministic means giving the same output for the same input case.
- Every step in the algorithm must be effective i.e. every step should do some work/single task.

Algorithm - Example

- Algorithm for going to the market to purchase a pen.
  1. Get dressed
  2. Check your wallet for money
  3. If there is no money in wallet , fill it.
  4. Go to shop
  5. Ask for your favourite brand
  6. If pen not available , go to step 10
  7. Pay for the pen
  8. Get the pen and keep safely
  9. Go back home
  10. Ask for any other brand
  11. Go to step 6

Example 2

- An algorithm to check whether a number is positive or negative.

Example 2

- An algorithm to check whether a number is positive or negative.

Steps:

1. Print, ”Enter any number”
2. Read num
3. If(num==0) print “Entered number is 0”
4. If(num>0) print “Entered number is a positive number”
5. If(num<0) print “Entered number is a negative number”

Flowchart

Computer algorithms could be represented using **Flowchart** (also known as Flow Diagram) and **Pseudocode**.

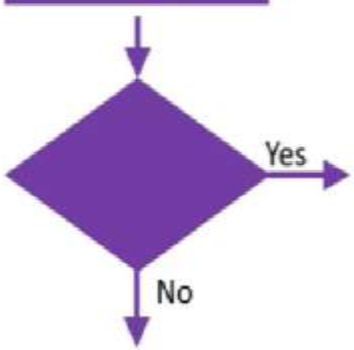
- Flowchart is a graphical or **visual representation** of an algorithm.
- Flowchart uses **formally agreed** standard **symbols** connected by lines with arrows to show the flow of the algorithm.
- Flowchart is **easier** to read and **understand**.
- Since flowcharts use formally agreed symbols these are **standardized**.

Flowchart-Symbols

Indicates **starting** point or **ending** point of the flowchart or algorithm.

Indicates **input** to or **output** from the process within the algorithm.

Indicates the **process** or operation that is performed within the algorithm.



Indicates the **selection** or **decision** within the algorithm.

Indicates the **logical flow** of the algorithm.

Algorithm-1: Add 2 numbers

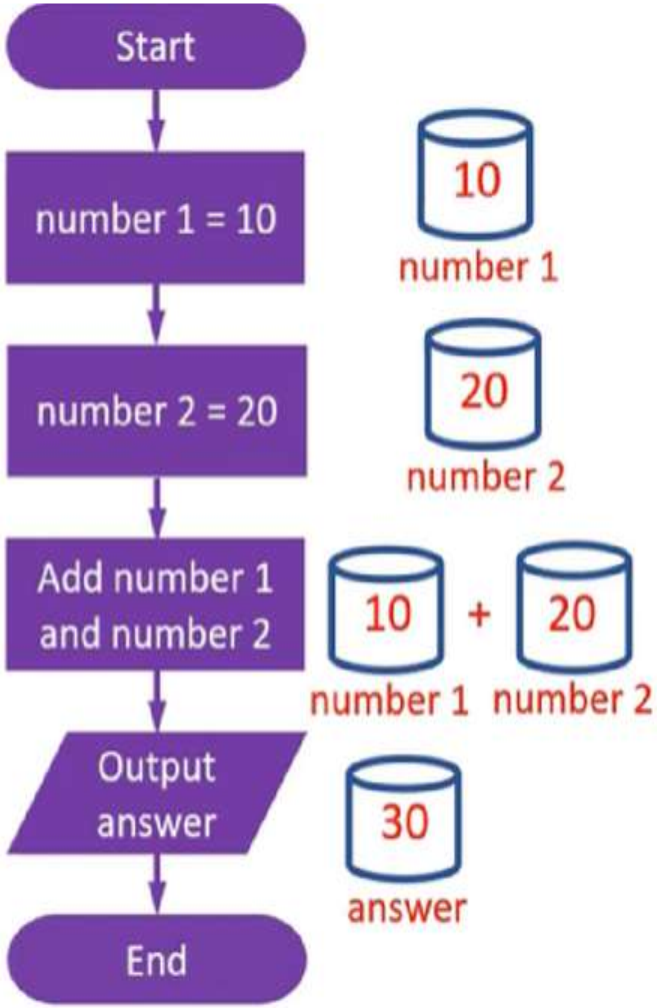
number 1 = 10

number 2 = 20

Add number 1 and number 2

Output the answer

Flowchart -1: Add 2 numbers



Variable is a container where you store a value that could be changed during the course of an algorithm or program.

Variable is given a name, also known as identifier.

Problem 2

Calculate difference in 2 nos.

Algorithm-2: Calculate difference in 2 nos.

Input number 1

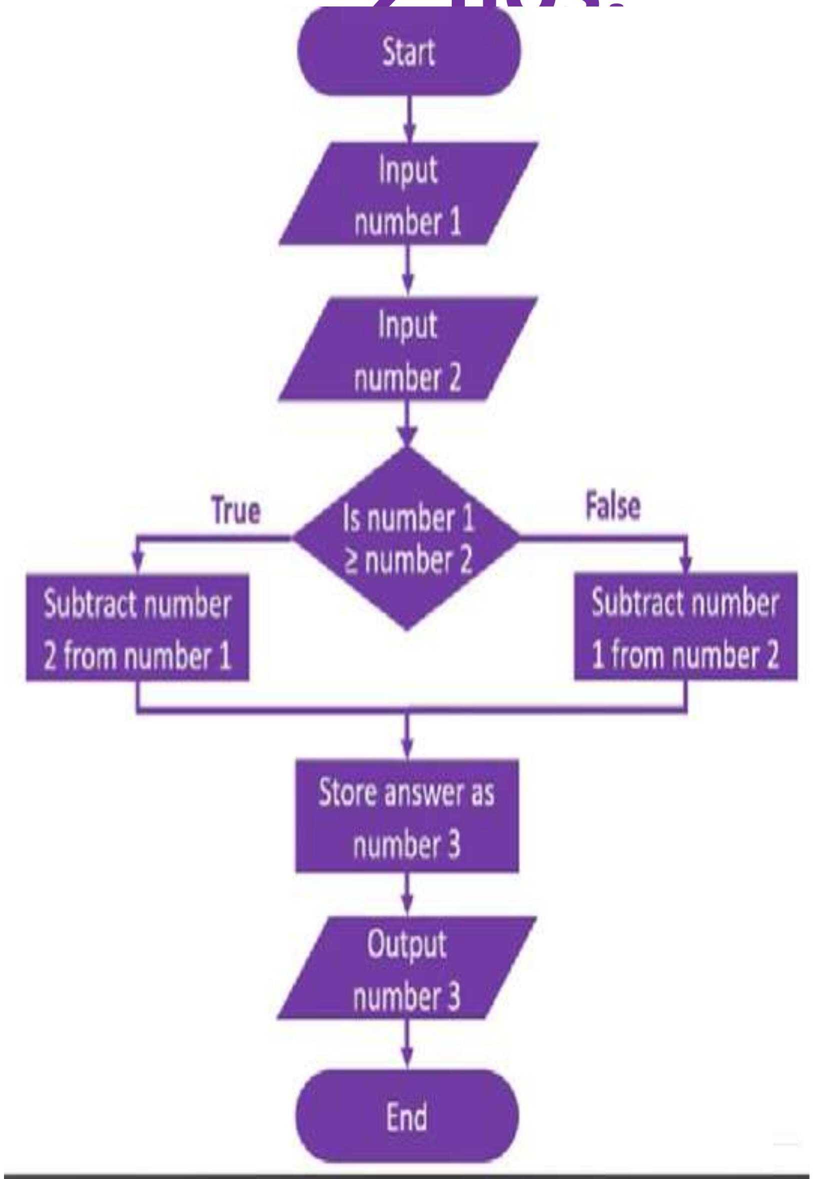
Input number 2

If number 1 is equal to or bigger than number 2
then subtract number 2 from number 1,
Otherwise subtract number 1 from number 2

Store the answer as number 3

Output number 3

Flowchart -2: Calculate difference in 2 nos.



Algorithm-3: Calculate Cashback

Ask use to enter purchase amount

If the purchase amount is > 500
then calculate the cashback
else cashback will be 0

if cashback calculated is > 100
then change the cashback to 100
else make no change to cashback

Display the cashback

Cashback is 5% of purchase amount

Flowchart -3: Calculate Cashback