WRITING AN ALGORITHM

A programming task can be divided into two phases:

- ***Problem-solving phase -*** produce an ordered sequence of steps that describe the solution of a problem this sequence of steps is called an ***algorithm.***
- ***Implementation phase -*** implement the program in some programming language.

Steps In Problem-Solving

- First, produce a general algorithm (one can use ***pseudo-code***)
- Refine the algorithm successively to get step by step detailed ***algorithm*** that is very close to a computer language.
- ***Pseudo code*** is an artificial and informal language that helps programmers develop algorithms. Pseudo-code is very similar to everyday English.

What is an Algorithm?

An algorithm is a step-by-step process of solving any problem. **Algorithm** means "a process or set of rules to be followed in calculations or other problem-solving operations". Therefore, an Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed to get the expected results.

Features of Algorithm

The first step of every algorithm should be Start, and the last step should be Stop.
An algorithm can be written in sentence form, or it can be written using mnemonics and pseudo codes.
An algorithm is mainly written to specify the following structure of the problem-solving technique:

- Input operation
- Process or computation
- Output operation

Assignment operation in algorithm is done using these operators: **:= or ←**
e.g. To show assignment of a value say 5 to a variable a:

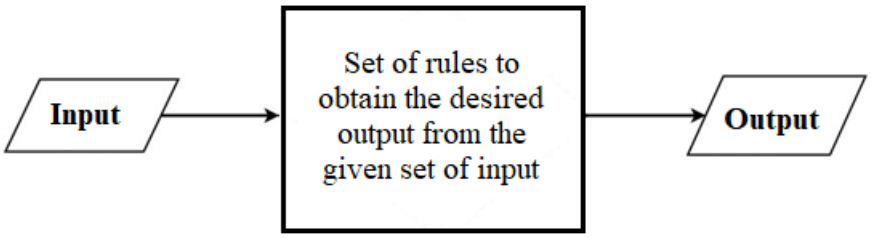
a:=5 or a ← 5

Note: Question may arise why = (assignment operator) is not used here?
The answer is, (=) operator is used to show comparison (equals to) in algorithm.

Myths in writing Algorithm

- Algorithm is dependent on the program code.
  - Algorithm does not depend upon any programming language and program code.
  - It is independent of any programming language.
- Algorithm cannot be written without writing the program code.
  - Algorithm can be written prior to write the program code as it only describes the IPO of the problem-solving technique.
- Algorithm which follows any programming language and program code is the best algorithm
  - Algorithm that is concise, clear and unambiguous, is the best algorithm. It should describe the input, process and output clearly, that's it.

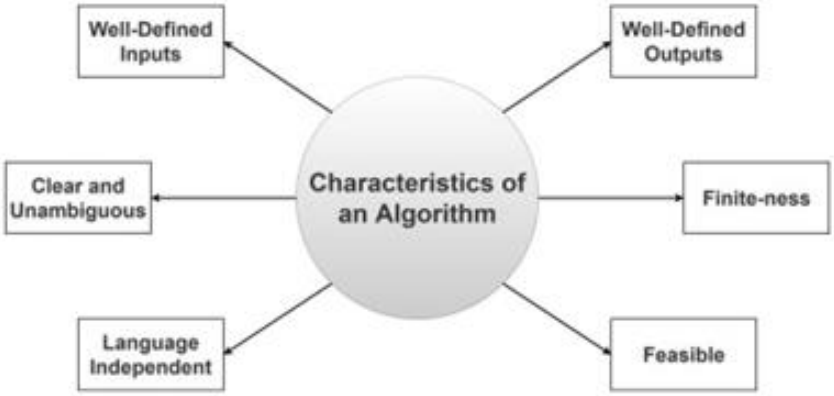
Pictorial form of an algorithm



It can be understood by taking an example of cooking a new recipe. To cook a new recipe, one reads the instructions and steps and execute them one by one, in the given sequence. The result thus obtained is the new dish cooked perfectly. Similarly, algorithms help to do a task in programming to get the expected output.

The Algorithm designed are language-independent, i.e. they are just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

Characteristics of an Algorithm



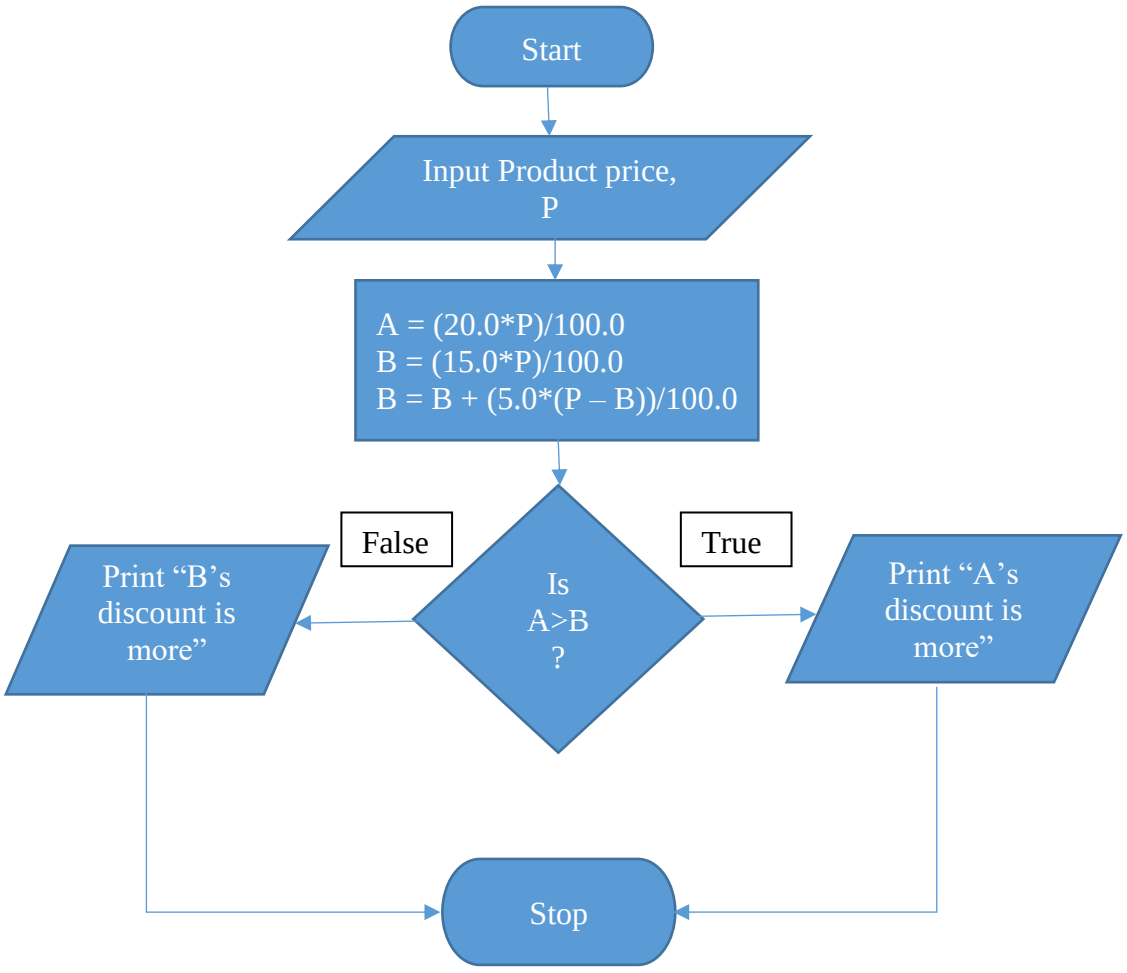
One would not follow any written instructions to cook the recipe, but only the standard one. Similarly, not all written instructions for programming is an algorithm. For some instructions to be an algorithm, it must have the following characteristics:

- **Clear and Unambiguous**: The algorithm should be clear and unambiguous. Each step should be clear in all aspects and lead to only one meaning.
- **Well-Defined Inputs**: If an algorithm says to take inputs, it should be well-defined inputs.
- **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and should also be well-defined.
- **Finite-ness:** The algorithm must be finite, i.e. it should not end up in an infinite loop or similar.
- **Feasible:** The algorithm must be simple, generic and practical, so it can be executed with the available resources. It must not contain some future technology or anything.
- **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

Write down the algorithm and draw the flowchart for the given question: Two shopkeepers are giving discounts on the same product in two different ways. 1st one is giving directly 20% and 2nd one is giving in the break of 15% and 5%. Find out whose discount is more.

P = Rs. 100.00
A = 20% on Rs100 = Rs. 20
B = 15% on Rs100 then 5% on the reduced amount
= 15 + (100 – 15)*5%
= 15 + 4.25 = Rs. 19.25

Step 1: Start
Step 2: Input the price of the product, P
Step 3: A := (20.0*P)/100.0 OR A ← (20.0*P)/100.0
Step 4: B := (15.0*P)/100.0 OR B ← (15.0*P)/100.0
Step 5: B := B + (5.0*(P – B))/100.0 OR B ← B + (5.0*(P – B))/100.0
Step 6: If A>B then
Print "A's discount is more"
else
Print "B's discount is more"
Step 7: Stop



Detailed algorithm for Scary Number program.
A no. is said to be a **Scary no**. if it contains 13. Eg. 113, 131, 1300, 1013…

Step 1. Start
Step 2. Input 2 nos. in L, B
Step 3. If L<1 OR B<1 then goto Step 2
Step 4. SP←0
Step 5. For A<=L to B repeat Step 6 to Step 12
Step 6. T←A : C←0
Step 7. While T>0 repeat Steps 8 to Step10
Step 8. R←T%100
Step 9. If R=13 then C←C+1
Step 10. T←T/10
Step 11. If C=1 then Print A : SP←SP+1
Step 12. A←A+1
Step 13. If SP=0 Print 0 else Print SP
Step 14. Stop

Detailed algorithm for Removal of duplicate alphabets in a given string program
Eg. INPUT: MISSISIPPI
OUTPUT: MISP

Step 1. Start
Step 2. Read a String, str
Step 3. get string length in L
Step 4. for I←0 to L-1 repeat Step 5 to Step 7
Step 5. ch←str.charAt(I)
Step 6. if ch>=48 AND ch<=57 OR ch>33 AND ch<=47 then goto step 2
Step 7. if I=L-1 AND ch!='.' then goto step 2
Step 8. declare object (ST) of StringTokenizer class and assign str
Step 9. C←ST.countTokens()
Step 10. for J←1 to C repeat Step 11 to Step 14
Step 11. W←ST.nextToken()
Step 12. R←removeDupl(W)
Step 13. Print R+(R.length()-W.length())
Step 14. J<=J+1
Step 15. Stop

Algorithm for removeDupl(W)
Step 1. Start
Step 2. L←W.length()
Step 3. R←W.charAt(0)
Step 4. For I←1 to L-1 repeat Step 5 to Step 10
Step 5. C←W.charAt(I)
Step 6. For J←0 to R.length()-1 repeat Step 7 to Step 8
Step 7. If C←R.charAt(J) then end J loop
Step 8. J←J+1
Step 9. If J loop completes the iteration, then R←R+C
Step 10. I←I+1
Step 11. Return R

Algorithm for the Fascinating Numbers Program:

To print all the Fascinating nos. b/w a range (from M – N)

A no. when concatenated with its double and triple, a new no. is formed where all the digits from 1 to 9 are present exactly once.

Step 1. Start

Step 2. Define a class named "**Fascinating**" with the following data members:

Step 2.1. int m, n (for storing lower and upper limits)

Step 2.2. Constructor Fascinating(a, b):

Step 2.3. Initialize m := a and n := b

Step 3. Define a method named "**formFascinate**(a)" in the "Fascinating" class:

Step 3.1 Take an integer 'a' as input

Step 3.2 Calculate b := a * 2 and c := a * 3

Step 3.3 Convert a, b, and c to a string and concatenate them: s := a + "" + b + "" + c

Step 3.4 Convert the concatenated string 's' to a long integer:

num := Long.parseLong(s)

Step 3.5 Return 'num'

Step 4. Define a method named "**checkFascinating**(a)" in the "Fascinating" class:

Step 4.1 Take a long integer 'a' as input

Step 4.2 Initialize a variable 'temp' as 'a' and a variable 'd' as 0

Step 4.3 Loop through digits from 1 to 9:

Step 4.4 Initialize a variable 'c' as 0

Step 4.5 While 'temp' is greater than 0:

- Extract the last digit 'r' from 'temp'
- Reduce 'temp' by dividing it by 10
- If 'r' is equal to the current digit 'i', increment 'c'

Step 4.6 If 'c' is equal to 1, increment 'd'

Step 4.7 If 'd' is equal to 9, return true; otherwise, return false

Step 5. Define a method named "**display**()" in the "Fascinating" class:

Step 5.1 Initialize a variable 'c' as 0

Step 5.2 Print "THE FASCINATING NUMBERS ARE:"

Step 5.3 Loop through numbers from 'm' to 'n':

- Calculate 'j' using the "formFascinate" method for the current number 'i'
- If the "checkFascinating" method returns true for 'j':
  - Increment 'c'
  - Print the current number 'i'

Step 5.4 Print "FREQUENCY OF FASCIATING NUMBERS IS: " followed by the value of 'c'

Step 6. Define the "main()" method:

Step 6.1 Create a Scanner object 'sc' for input

Step 6.2 Prompt the user to enter the lower limit 'x'

Step 6.3 Read and store 'x' using 'nextInt()'

Step 6.4 Prompt the user to enter the upper limit 'y'

Step 6.5 Read and store 'y' using 'nextInt()'

Step 6.6 Create an object 'obj' of the "Fascinating" class with 'x' and 'y' as parameters

Step 6.7 Call the "display()" method on the 'obj' object

Step 7. End

Algorithm for Stack program having Push, Pop and Display operations.

Step 1. Initialise the Stack
    Step 1.1. Create an array to represent the stack.
    Step 1.2. Set a variable top to -1 to indicate that the stack is empty.
Step 2. Push Operation
    Step 2.1. Check if the stack is full, display “STACK is FULL” : return
    Step 2.2. Increment the top variable.
    Step 2.3. Add the new element at the position indicated by top.
Step 3. Pop Operation
    Step 3.1. Check if the stack is empty.
    Step 3.2. If empty, print “STACK is EMPTY” and return -999.
    Step 3.4. Retrieve the element at the position indicated by top.
    Step 3.5. Decrement the top variable.
    Step 3.6. Return the retrieved element.
Step 4. Display Operation
    Step 4.1. Check if the stack is empty.
    Step 4.2. If empty, print an appropriate message.
    Step 4.3. Iterate from the top of the stack to the bottom and print each element.
Step 5. Menu-Driven Main Method
    Step 5.1. Display a menu with options for push, pop, display, and exit.
    Step 5.2. Use a loop to repeatedly display the menu and perform the selected operation until the user chooses to exit.

Algorithm for Queue program having Push, Pop and Display operations.

Step 1. Initialise the Queue
    Step 1.1. Create an array to represent the queue.
    Step 1.2. Set a variables front & rear with 0 to indicate that the queue is empty.
Step 2. Push Operation
    Step 2.1. Check if the queue is full, then display “QUEUE is FULL” : return
    Step 2.2. Add the new element at the position indicated by rear.
    Step 2.3. Increment the rear variable by 1.
Step 3. Pop Operation
    Step 3.1. Check if the queue is empty.
    Step 3.2. If empty, print “QUEUE is EMPTY” : set front & rear to 0 and return -999.
    Step 3.4. Retrieve the element at the position indicated by front.
    Step 3.5. Increment the front variable by 1.
    Step 3.6. Return the retrieved element.
Step 4. Display Operation
    Step 4.1. Check if the stack is empty.
    Step 4.2. If empty, print an appropriate message.
    Step 4.3. Iterate from the front to the rear and print each element.
Step 5. Menu-Driven Main Method
    Step 5.1. Display a menu with options for push, pop, display, and exit.
    Step 5.2. Use a loop to repeatedly display the menu and perform the selected operation until the user chooses to exit.

Algorithm for Circular Queue Implementation

Step 1. Start

Step 2. Initialize Circular Queue

Step 2.1 Input: n (capacity of the queue)
Step 2.2 Create a CircularQueue object with:
Step 2.2.1 capacity := |n|
Step 2.2.2 F := 0 (front pointer)
Step 2.2.3 R := 0 (rear pointer)
Step 2.2.4 CQue = new array of size capacity

Step 3. Push an element at the rear

Step 3.1 Input: num (element to be added)
Step 3.2 Check if (R + 1) % capacity = F (if queue is full):
Step 3.2 If True, print "Queue is full now" and exit the method
Step 3.4 Otherwise:
Step 3.4.1 Assign CQue[R] := num
Step 3.4.2 Update R = (R + 1) % capacity
Step 3.4.3 Print "FRONT: F & REAR: R"

Step 4. Remove an element from the front

Step 4.1 Check if F = R (if queue is empty):
Step 4.2 If True, print "Queue has been emptied", return -999 and exit the method
Step 4.3 Otherwise:
Step 4.3.1 Assign n := CQue[F]
Step 4.3.2 Update F := (F + 1) % capacity
Step 4.3.3 Print "FRONT: F & REAR: R"
Step 4.3.4 Return n

Step 5. Display elements of the queue

Step 5.1 Check if R = F (if queue is empty):
Step 5.2 If True, print "Queue is already empty" and exit the method
Step 5.3 Otherwise:
Step 5.3.1 Initialize i := F
Step 5.3.2 While i != R:
Step 5.3.3.1 Print CQue[i]
Step 5.3.3.2 Update i := (i + 1) % capacity
Step 5.3.4 Print a newline character

Step 6. End