

Problem Solving by Computer – Logical Thinking

Pranam K Abhyuday

Associate Consultant, Talent Transformation

Wipro Technologies

Introduction

This tech capsule “**Problem Solving by Computer**” will focus on **how to solve computation problems**.

In order to solve computational problems we will focus a lot of how to build logic. As we go ahead and understand logic building (basics) we will look at effective logic building (to come up with logic which is small, consumes less time, etc.) and how to write good logic quickly.

Examples of Computational problems

- Adding numbers by computer
- Check if a given string is a palindrome or not
- Find the length of a given string
- And many more

What is LOGIC?

Logic is a collection of well-defined activities to be performed (in our case a set of instructions to be executed by the computer) in order to solve the problem.

Say for example we need to add two numbers.

So our logic will be to -

- get the 2 numbers to be added from the user
- add them using the addition operator (+)
- save the result
- print the result

These steps of activities which when performed would give us the solution to our problem can be called as **LOGIC**.

Logic must be represented in a form which can be understood by humans. The different ways to do so are by writing a flowchart or a pseudo code (discussed later).

Once represented as an algorithm the same logic must be translated in a form which can be understood by the computer. We call this form as a Program. Programs are written using any of the programming languages like Java or .net or C or C++ (there are many more and you can choose the one you are comfortable with).

What is Logic Building or how do we build logic to solve problems?

Logic building can be looked at a series of steps starting from understanding the problem statement, followed by thinking of ways/ideas to solve them and eventually represent them in the form of a solution which can be understood.

Logic building for programming problems improves with practice and by trying to solve the same problem in different ways.

What are Algorithms and Programs?

A typical programming task can be divided into 2 phases:

1) **Algorithm Building:**

Once we have understood a given problem and have a fair idea of how to solve it, we represent the solution in the form of steps or activity which will comprise of our logic. Each step does something constructive and all the steps put together represent the solution.

These steps or activities are called as an Algorithm. Algorithms are written in English language or diagrammatically represented which is **unambiguous** and easily **understandable** by humans.

Algorithms can be classified into two forms –

a) **Pseudo code:**

Pseudo codes are written in simple English language and represented as steps, step1 to stepN.

Example: Write an algorithm (using pseudo code representation) to determine if a student has passed the English Language exam or not. The students mark must be entered by the user. The pass mark for the exam is greater than 35.

Solution:

STEP1: Read student **MARK** from user

STEP2: if **MARK** is greater than 35 then

Print "PASS"




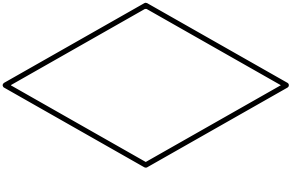

else

Print "FAIL"

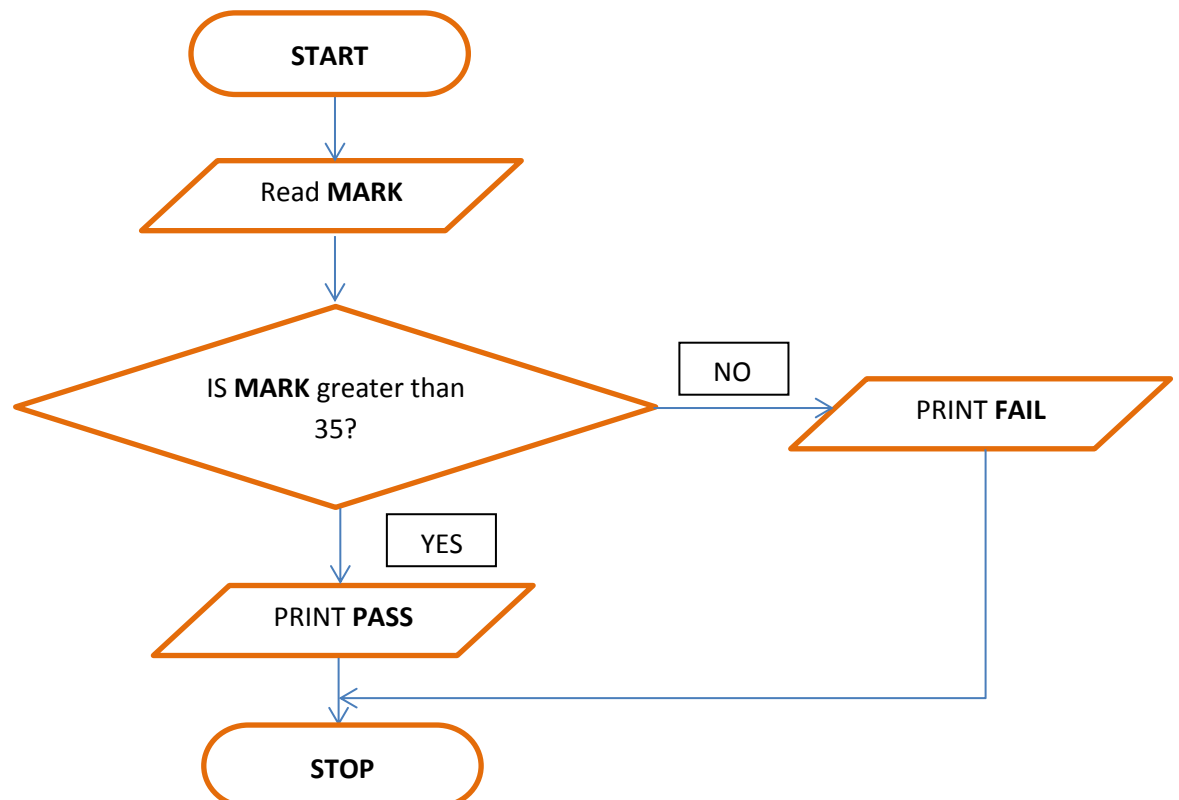
These 2 steps are our logic to check if a student has passed the exam or not and they are represented using a pseudo code.

b) **Flowcharts:**

Flowcharts are diagrammatic representation of an algorithm. There is a set of pre-defined symbols which must be used to draw a flowchart.

| Name | Symbol | Use in Flowchart |
|---------------|--|---|
| Oval |  | Denotes the beginning or end of the program |
| Parallelogram |  | Denotes an input or output operation |
| Rectangle |  | Denotes a process to be carried out – Add, Subtract |
| Diamond |  | Denotes a decision (or branch) to be made . The program should continue along one of two routes . (e.g. IF/THEN/ELSE) |
| Flow line |  | Denotes the direction of logic flow in the program |

Let us rewrite the logic to check if a student has passed the English Language exam or not, now using a flowchart



2) Program Building:

The second step to a programming task is to write the PROGRAM itself. Program is a set of well-defined instructions written using a programming language (may be C or C++ or Java or any other) which the computer can understand.

Once we have the logic represented in the form of an algorithm (either using pseudo code or flowchart) we rewrite the same using a programming language. Here it is very important for the programmer to have in depth knowledge of the programming language.

Types of Computation Problems

There are many computation problems in reality; however in this tech capsule we will classify them into 6 categories (as mentioned below) and our learning will focus on these itself.

- 1) Number Based problems
- 2) Loop based problems
- 3) Array (collection of data) based problems
- 4) String Based problems
- 5) Sorting and Searching related problems

In our subsequent discussions we will look at each Computational Problem in detail and understand them by solving a lot of problems by ourselves.

Algorithm Dry Run Table

Once we have understood a given problem statement and represented the required solution logic using an Algorithm, we check for its correctness. We build an **Algorithm Dry Run Table** to check correctness of an algorithm.

The template for the Algorithm Dry Run Table is shown below. This table must be used as a reference to create the table required for any algorithm.

| Algorithm Step | Variable1 | Variable2 | Variable3 | Variable4 | Test condition (PASS / FAIL) | Output |
|----------------|-----------|-----------|-----------|-----------|---------------------------------|--------|
| STEP1 | | | | | | |
| STEP2 | | | | | | |
| STEP3 | | | | | | |
| ... | | | | | | |
| ... | | | | | | |
| ... | | | | | | |
| STEPN | | | | | | |

The above table must be rewritten for different iterations to check if the algorithm works correctly for different input values.

USAGE:

- 1) For each step, the variable content is updated based on computations performed
- 2) **Test condition** is updated for an **if(condition)** if true as **PASS** otherwise as **FAIL**. It is also updated while evaluating condition within an iterative loop.
- 3) **Output** is updated with message to be displayed at different stages of the algorithm or at the end of the algorithm.
- 4) In case of iterative loops and recursive algorithms, the same **STEP** must be rewritten in the “**Algorithm Step**” column and computed again for the number of times required.

Example: Write an algorithm (using pseudo code representation) to determine if a student has passed the English Language exam or not. The students mark must be entered by the user. The pass mark for the exam is greater than 35.

Solution: The algorithm is as shown below

```
STEP1: Read student MARK from user
STEP2: if MARK is greater than 35 then
        Print "PASS"
      else
        Print "FAIL"
```

Now we construct the Algorithm Dry Run Table to check the correctness of the above algorithm. For 2 different values of MARK entered by the user we create separate tables and check.

Iteration 1:

If value of MARK is 40

| Algorithm Step | Variable1 - MARK | Variable2 | Variable3 | Variable4 | Test condition if MARK ... (PASS / FAIL) | Output |
|----------------|------------------|-----------|-----------|-----------|--|--------|
| STEP1 | 40 | NA | NA | NA | NA | NA |
| STEP2 | 40 | NA | NA | NA | PASS | PASS |

Iteration 2:

If value of MARK is 15

| Algorithm Step | Variable1 - MARK | Variable2 | Variable3 | Variable4 | Test condition if MARK ... (PASS / FAIL) | Output |
|----------------|------------------|-----------|-----------|-----------|--|--------|
| STEP1 | 15 | NA | NA | NA | NA | NA |
| STEP2 | 15 | NA | NA | NA | FAIL | FAIL |