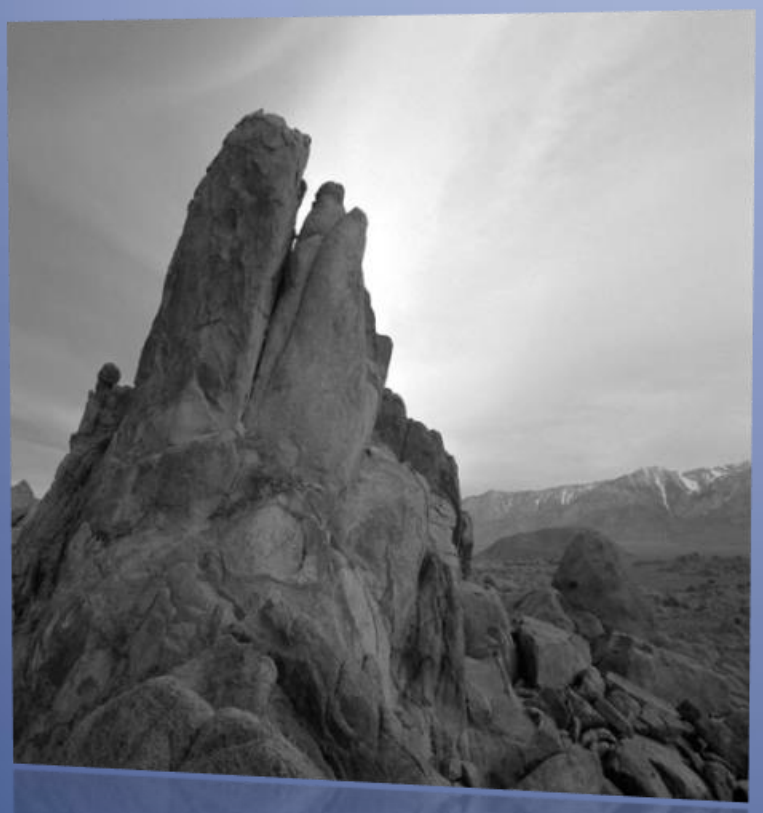

An Approach to Logic Building

Manoj Jauhari

Contents

Purpose of this document

- To help programmers in enhancing their logic building ability
- Especially aimed at programmers who are new to programming and developing logic

Focus of learning

Many beginners find it difficult to *think* of possible *programming solutions* to a given logical problem. Suggesting an "approach" can be of help to address this difficulty. This document will focus on the "*approach*" to be followed while finding possible solutions to a given logical problem.

Language independent

The logic building examples and approach mentioned in this document will be language agnostic i.e. independent of any specific programming language. The examples mentioned here will focus on *how* to solve the given problem. The learner should be able to convert them to programs in any known programming language such as Java, C, C++, C# etc.

SECTION – A : Basics and Pre-requisites

Basics of "Logic Building"

What is "Logic" or a "Logical solution"?

"Logic" or a "Logical solution" is a set of activities to be performed for solving a given problem.
Different people generally come up with different ways of solving the same problem.
Thus, any given problem usually can have a lot of possible logical solutions.

How can logic be represented?

In the programming world, the most basic and popular ways of representing logic are –

- Algorithms
- Flow charts

What is an algorithm?

An algorithm is a step-by-step representation of the possible solution for the given problem.
Algorithms do not use any specific programming language and are generally written in plain English.
In a sheet of paper, if we write down the steps required to solve the given problem, it can be referred to as the "algorithm" for solving the given problem.

For example, if we want to find whether a given number N is even or odd, the algorithm for it can be as below –
Step 1 – Get the value of N
Step 2 – Divide N by 2 and store the remainder in R
Step 3 – If the remainder R is zero, print "The number N is even"
Step 4 – Else print "The number N is odd"

In the above example note that each step is clear (unambiguous) and if these 4 steps are done in the specified order, they can find whether any given number N is even or odd.

- IMPORTANT: While writing algorithms it is desirable to reach a level of granularity such that each step of the algorithm can be converted into a single line of code or into a set of lines of code. Such an algorithm can be called a "Detailed algorithm".
- MORE IMPORTANT: Though our goal is to come up with a "Detailed Algorithm", on many occasions it is not possible to directly come up with a detailed algorithm. The best approach is to first come up with a "High Level algorithm" that should list down the steps at a very high level, without getting into details. Once we have a "High Level algorithm", we should start focusing on each of its steps, one by one, to define detailed-steps for each high-level step. This is a divide-and-conquer approach that we will talk about in our section – "Suggested APPROACH to logic building and problem solving". This approach will also be the focus while we practice solving a lot of logic problems in this learning journey.

What is a flowchart?

A flowchart is a pictorial representation of the flow of a programming solution.
Flow-charts are mainly useful when a solution involves handling various conditions, and different actions need to be taken to handle the different conditions.

Suggested reading: Read the document "Problem Solving by Computer" (Problem Solving by Computer.pdf) available in the "Logic Building" link in PBLApp, to understand the basics of Algorithms and Flowcharts.

Pre-requisites: Conditional statements & Looping constructs

This section expects that the learner is already aware of the below fundamentals of programming languages –

- Conditional statements - (if..else) (switch..case)
- Looping constructs - (for loop) (while loop)

The syntax (grammar) of conditional statements and looping constructs is almost similar in most programming languages. Below we are mentioning a general syntax for these two which we will follow while suggesting the logical solutions in this document.

Conditional statements:

if	If..else	If .. else if .. else if .. else	switch .. case

`if (condition)`
`{`
`  Statement;`
`  Statement;`
`  …`
`  Statement;`
`}`
	`if (condition)`
	`{`
	`  Statement;`
	`  Statement;`
	`  …`
	`  Statement;`
	`}`
	`else`
	`{`
	`  Statement;`
	`  Statement;`
	`  …`
	`  Statement;`
	`}`
		`if (condition 1)`
		`{`
		`  Statement;`
		`  Statement;`
		`  …`
		`  Statement;`
		`}`
		`else if(condition 2)`
		`{`
		`  Statement;`
		`  Statement;`
		`  …`
		`  Statement;`
		`}`
		`else if(condition 3)`
		`{`
		`  Statement;`
		`  Statement;`
		`  …`
		`  Statement;`
		`}`
		`else`
		`{`
		`  Statement;`
		`  Statement;`
		`  …`
		`  Statement;`
		`}`
			`Switch (variable)`
			`{`
			`  case value1:`
			`  Statement;`
			`  Statement;`
			`  …`
			`  Statement;`
			`  break;`
			`  case value2:`
			`  Statement;`
			`  Statement;`
			`  …`
			`  Statement;`
			`  break;`
			`  case value3:`
			`  Statement;`
			`  Statement;`
			`  …`
			`  Statement;`
			`  break;`
			`  ...`
			`  ...`
			`  default:`
			`  Statement;`
			`  Statement;`
			`  …`
			`  Statement;`
			`}`

Looping constructs:

For	while

`for (initialization ; check condition; incr or decrement)`
`{`
`  Statement;`
`  Statement;`
`  Statement;`
`  ...`
`  Statement;`
`}`
	`while(condition)`
	`{`
	`  Statement;`
	`  Statement;`
	`  Statement;`
	`  ...`
	`  Statement;`
	`}`

Looping flow – **Step1:** First the "initialization" part is executed. **Step2:** Then the "check condition" part is evaluated. **Step3:** If the 'condition' evaluates to true, the control flows into the block and all 'Statements' are executed. **Step4:** The flow then proceeds to the "increment or decrement" part and executes the expressions mentioned there. **Step5:** Then the "check condition" part is again evaluated. Till the 'condition' remains true, Steps 3 to 5 are continuously executed. When the 'condition' evaluates to false, the loop ends, and the control reaches to the statement after the loop.	**Looping flow –** While the condition evaluates to true, the statements in the looping block are continuously executed.

Pre-requisite programs for "conditional statements" and "looping constructs"

IMPORTANT POINTS:

Note 1: The learners are expected to properly understand the usage of "conditional statements" and "looping constructs", before they proceed with reading the logical questions in this document. Clear understanding of these concepts will help in easily understanding the logic building approach.

Note 2: Teaching "conditional statements" and "looping constructs" is not within the scope of this document. The learners should clarify their doubts regarding these with their faculty and mentor. The learners should also practice "conditional statements" and "looping constructs" by trying out many programs.

Mandatory (pre-requisite) programs to be done by the learner before proceeding further -

The below simple programs **must** be done by the learners to demonstrate that they have understood the basic usage of "conditional statements" and "looping constructs". If the learners are unable to do the below programs they are strongly encouraged to discuss with their faculty (mentor) and understand how to use "conditional statements" and "looping constructs".

Programs to be done by learner's to demonstrate their readiness with "Conditional statements"

- Write a program to accept a number N and print whether the number is EVEN or ODD.
- Write a program to accept two numbers and print whether their sum is EVEN or ODD

Programs to be done by learner's to demonstrate their readiness with "Looping constructs"

- Write a program to print all numbers from 1 to 100 i.e. 1 2 3 4 5 6 7 . . . 98 99 100
- Write a program to print alternate numbers starting from 1 to 99 i.e. 1 3 5 7 9 11 13 . . . 95 97 99
- Write a program to print alternate numbers starting from 0 to 100 i.e. 0 2 4 6 8 10 12 . . . 96 98 100
- Write a program to print all numbers backwards from 100 to 0 i.e. 100 99 98 97 96 . . . 4 3 2 1 0
- Write a program to print numbers backwards from 100 to 1 by skipping 2 numbers i.e. 100 97 94 91 88 85 82 79. . . 22 19 16 13 10 7 4 1

SECTION – B : Logic building Approach

Suggested APPROACH to logic building and problem solving

For beginners, I strongly recommend the below approach while trying to think of a possible solution for a given problem.

1. **Use Pen and Paper:**
   - Do not try to directly write the program on the computer
   - First try to work out the solution on a paper
2. **Do not immediately deep-dive. Initially, Think of the solution at a very HIGH LEVEL:**
   - Do not start thinking of internal and minute details for solving the given problem
   - First try to think of the possible solution at a very HIGH LEVEL
   - Once you have a set of HIGH LEVEL steps, write them down on a piece of paper. This becomes your HIGH LEVEL algorithm.
3. **Focus separately on each step of the HIGH LEVEL algorithm:**
   - Once the set of steps representing your HIGH LEVEL algorithm is ready, start focusing on each step one-by-one
   - For each step, think of the possible sub-steps to solve the step
   - Write down the sub-steps for each step
   - This way, you will start focusing on the sub-steps required for each step, and what you would get is a slightly DETAILED algorithm

In fact, this is a **DIVIDE AND CONQUER approach**.
Instead of solving the entire given problem in one shot, we are first identifying different parts of the solution, and then focusing on how to solve each part (step), one at a time.

Note: The above approach requires time and practice. The learners (beginners) are recommended to start following this approach by practicing the problems mentioned in the following pages/sections.

Logical Problems involving loops

Problem1: Print the below shape on a console window [10 rows right-angled left justified numbers]

```
1
12
123
1234
12345
123456
1234567
12345678
123456789
12345678910
```

Constraints: In a console window, the cursor cannot be placed on any specific pixel at will. That is, the cursor moves row-by-row. For example, If the cursor is at row 5, it is not possible to again move it back to row 4 or earlier. So it is important for us to think of a possible solution where we keep printing the pattern row-by-row.

Instruction: [15 minutes window] – First think of a possible solution by yourself. Before looking at the below suggested solution, please spend at least 15 minutes and try to come up with a possible algorithm to print the above shape.

Suggested approach to find a possible solution:

1. **Use Pen and Paper**

2. **Do not immediately deep-dive. Initially, Think of the solution at a very HIGH LEVEL:**
   - If we carefully look at the above pattern, at a high-level (from a birds-view) we will realize that –
     - There are 10 rows
     - In each row -
       - a set of numbers are printed
       - and then the cursor comes to the next row
   - So, our **HIGH LEVEL programming solution** for this problem can be as below –
     - **Step1** - Write a loop that repeats 10 times
     - In each iteration of the loop
       - **Step2** - Print numbers
       - **Step3** - Print newline character

3. **Focus separately on each step of the HIGH LEVEL algorithm**
   - Till now, we have come up with a HIGH LEVEL programming solution consisting of 3 steps
   - Now, let us focus on each Step one by one

- ```
  // Step1 – Write a loop that repeats 10 times
  for (int row = 1; row <= 10; row++)
  {
  }
  ```

Explanation: In 'Step1' our focus is only on creating a loop that will repeat 10 times. We can achieve this using a 'for' loop as shown above.
Note that it is good to use suggestive variable names in algorithms (as well as in programs) because it helps to read and understand the logic. For e.g. In the above case, we have named the loop iteration variable as 'row', to represent that in each iteration of the loop we are moving to a different row of the given pattern.

- ```
  // In each iteration of the loop
  // Step2 – Print numbers

  for (int row = 1; row <= 10; row++)
  {
        // Step2 - Print numbers
        for (int num = 1; num <= row; num++)
        {
              print(num);
        }
  }
  ```

Explanation: In 'Step2' our focus is only to 'Print numbers' in each row. How do we achieve that? Each row is supposed to print a different set of numbers. Let us look carefully at the numbers in various rows.
In row **5**, the numbers to be printed are 12345 i.e. **1** to **5**
In row **8**, the numbers to be printed are 12345678 i.e. **1** to **8**
In row **3**, the numbers to be printed are 123 i.e. **1** to **3**
In row **10**, the numbers to be printed are 12345678910 i.e. **1** to **10**
So, we can generalize that –
In row **R**, the numbers to be printed are **1** to **R**
Therefore, we need to write a loop that will print the numbers from **1** to the **row number**.

```
for (int num = 1; num <= row; num++)
{
     print(num);
}
```

- ```
  // In each iteration of the loop
  // Step3 – Print newline character

  // Step1 – Write a loop that repeats 10 times
  for (int row = 1; row <= 10; row++)
  {
        // Step2 - Print numbers
        for (int num = 1; num <= row; num++)
  ```
  ```

```
        {
              print(num);
        }

        // Step3 - Print newline character
        print("\n");
}
```

Explanation: In 'Step3' our focus is only to 'Print a newline character' immediately after the numbers have been printed using the previous step.
Therefore, we just need to write a print statement with a newline character '\n'.

```
print("\n");
```

Summary: Above, we have demonstrated how to use the **DIVIDE AND CONQUER approach** while trying to find a possible programming solution for a given problem. The approach is to first come up with a **High-Level algorithm**, and then expand each step of the High-Level algorithm to come up with a **Detailed algorithm**.

High-Level Algorithm:

- **Step1** - Write a loop that repeats 10 times
- In each iteration of the loop
  - **Step2** - Print numbers
  - **Step3** - Print newline character

Detailed Algorithm:

```
// Step1 - Write a loop that repeats 10 times
for (int row = 1; row <= 10; row++)
{
      // In each iteration of the loop
      // Step2 - Print numbers
      for (int num = 1; num <= row; num++)
      {
            print(num);
      }

      // Step3 - Print newline character
      print("\n");
}
```

Problem2: Print the below shape on a console window [10 rows right-angled right-justified stars]

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

Suggested approach to find a possible solution:

1. **Use Pen and Paper**

2. **Do not immediately deep-dive. Initially, Think of the solution at a very HIGH LEVEL:**
   - If we carefully look at the above pattern, at a high-level (from a birds-view) we will realize that –
     - There are 10 rows
     - In each row -
       - A set of spaces are printed
       - a set of stars (asterisks) are printed
       - and then the cursor comes to the next row
   - So, our **HIGH LEVEL programming solution** for this problem can be as below –
     - **Step1** - Write a loop that repeats 10 times
     - In each iteration of the loop
       - **Step2** - Print spaces
       - **Step3** - Print stars
       - **Step4** - Print newline character

3. **Focus separately on each step of the HIGH LEVEL algorithm**
   - Till now, we have come up with a HIGH LEVEL programming solution consisting of **4** steps
   - Now, let us focus on each Step one by one

   - ```
     // Step1 – Write a loop that repeats 10 times
     for (int row = 1; row <= 10; row++)
     {
     }
     ```

     Explanation: This is same as the step in the previous example.

   - ```
     // In each iteration of the loop
     // Step2 – Print spaces

     for (int row = 1; row <= 10; row++)
     ```

```
{
        // Step2 - Print spaces
        for (int space = 1; space <= 10-row; space++)
        {
                print(" ");
        }
}
```

Explanation: In 'Step2' our focus is only to 'Print spaces' in each row. How do we achieve that? Each row is supposed to print a different set of spaces. Let us look carefully at the spaces in various rows.
In row **1**, we need to print **9** spaces
In row **2**, we need to print **8** spaces
In row **3**, we need to print **7** spaces
In row **4**, we need to print **6** spaces
. . . and so on . . .
In row **9**, we need to print **1** space
In row **10**, we need to print **0** spaces

So, we can generalize that –
In row **R**, we need to print **(10-R)** spaces

Therefore, we need to write a loop that will iterate from **1** to **10-R** and will print a space character in each iteration.

```
for (int space = 1; space <= 10-row; space++)
{
      print(" ");
}
```

- // In each iteration of the loop
  // Step3 – Print stars

```
for (int row = 1; row <= 10; row++)
{
        // Step2 - Print spaces
        for (int space = 1; space <= 10-row; space++)
        {
                print(" ");
        }

        // Step3 - Print stars
        for (int star = 1; star <= row; star++)
        {
                print("*");
        }

}
```

Explanation: In 'Step3' our focus is only to 'Print stars' in each row. How do we achieve that? Each row is supposed to print a different set of stars. Let us look carefully at the stars in various rows.
In row **1**, we need to print **1** stars
In row **2**, we need to print **2** stars
In row **3**, we need to print **3** stars
In row **4**, we need to print **4** stars
. . . and so on . . .
In row **10**, we need to print 1**0** stars

So, we can generalize that –
In row **R**, we need to print **R** stars

Therefore, we need to write a loop that will iterate from **1** to **R** and will print a star character in each iteration.

```
for (int star = 1; star <= row; star++)
{
      print("*");
}
```

- // In each iteration of the loop
  // Step4 – Print newline character

```
for (int row = 1; row <= 10; row++)
{
      // Step2 - Print spaces
      for (int space = 1; space <= 10-row; space++)
      {
            print(" ");
      }

      // Step3 - Print stars
      for (int star = 1; star <= row; star++)
      {
            print("*");
      }

      // Step3 - Print newline character
      print("\n");
}
```

Explanation: In 'Step3' our focus is only to 'Print a newline character' immediately after the spaces and stars have been printed using the previous steps.
Therefore, we just need to write a print statement with a newline character '\n'.

```
print("\n");
```

Summary: Above, we have demonstrated how to use the **DIVIDE AND CONQUER approach** while trying to find a possible programming solution for a given problem. The approach is to first come up with a **High-Level algorithm**, and then expand each step of the High-Level algorithm to come up with a **Detailed algorithm**.

High-Level Algorithm:

- **Step1** - Write a loop that repeats 10 times
- In each iteration of the loop
  - **Step2** - Print spaces
  - **Step3** - Print stars
  - **Step4** - Print newline character

Detailed Algorithm:

```
// Step1 – Write a loop that repeats 10 times
for (int row = 1; row <= 10; row++)
{
      // Step2 - Print spaces
      for (int space = 1; space <= 10-row; space++)
      {
            print(" ");
      }

      // Step3 - Print stars
      for (int star = 1; star <= row; star++)
      {
            print("*");
      }

      // Step4 - Print newline character
      print("\n");
}
```

Problem to be solved by the Learner based on the given High-Level algorithm

Problem3: Print the below shape on a console window [10 rows triangle of incr-decr numbers]

```
         1
        121
       12321
      1234321
     123454321
    12345654321
   1234567654321
  123456787654321
 12345678987654321
1234567891087654321
```

High-Level Algorithm:

- **Step1** - Write a loop that repeats 10 times
- In each iteration of the loop
  - **Step2** - Print spaces
  - **Step3** - Print numbers in increasing order
  - **Step4** - Print numbers in decreasing order
  - **Step5** - Print newline character

Detailed Algorithm: [To be completed by the learner]

The High-Level algorithm is already mentioned above.
Please follow the approach suggested in the previous two problems and come up with a detailed algorithm for this problem.

Further PRACTICE problems to be completed by the Learner

Practice 1: Print the below shape on a console window [10 rows triangle incr-nums decr-alphabet]

```
         1
        12A
       123BA
      1234CBA
     12345DCBA
    123456EDCBA
   1234567FEDCBA
  12345678GFEDCBA
 123456789HGFEDCBA
12345678910IHGFEDCBA
```

Practice 2: Print the below shape on a console window [10 rows rhombus inc-decr numbers]

```
         1
        121
       12321
      1234321
     123454321
    12345654321
   1234567654321
  123456787654321
 12345678987654321
1234567891987654321
 12345678987654321
  123456787654321
   1234567654321
    12345654321
     123454321
      12321
       121
        1
```