# **Pseudo code**

Programmers frequently utilize **pseudo code**, a **high-level language** that is simpler to read and comprehend than actual programming code, to convey **algorithms** and **program logic**.

## What is Pseudo Code?

A computer program, algorithm, or process that is intended to be simple to comprehend by humans is described in **pseudo code**, which is **plain English**. It is a high-level illustration of the program's logic and flow that can serve as a guide for developing the real code. Pseudo code can be used to explain both **complicated procedures**, like **sorting** algorithms, and **straightforward duties,** like **input validation**.

**Pseudo code** is a method of representing logic in programming that is independent of any programming language. It uses natural language rather than the **rigid syntax** of programming languages since it is intended to be easily understood by humans. Common programming features like **loops, conditionals,** and **function calls** are frequently used in pseudo-code but in a condensed form, making the program logic more understandable.

Software engineers frequently use **pseudo-code** as a design technique since it enables them to work out the specifics of a program's logic before writing any actual code. Because errors are considerably simpler to rectify in the **pseudo-code** than they are in the actual code, this can reduce errors and save time.

Although there isn't a dedicated programming language for pseudo code but C is a good choice for implementation. C is a **general-purpose**, **high-level programming language** that is frequently used to create system software. It is a structured programming language, which implies that it controls the flow of the program using a few predefined programming constructs.

C programming language simply translates the plain language description of the program logic into C syntax to construct pseudo-code. The C programming language directly corresponds to the fundamental programming features found in pseudo-code, such as **loops**, **conditionals**, and **function calls**.

**Example:**

Let's consider an example of implementing a simple program that calculates the average of three numbers using pseudo code in C:

**Pseudo code:**

1. Start
2. Input three numbers
3. Calculate the sum of the three numbers
4. Divide the sum by 3 to get the average
5. Display the average
6. End

**C code:**

1. #include <stdio.h>
2. **int** main()
3. {
4. **float** num1, num2, num3, sum, avg;
5. printf("Enter three numbers: ");
6. scanf("%f %f %f", &num1, &num2, &num3);
8. sum = num1 + num2 + num3;
9. avg = sum / 3;
11. printf("The average of the three numbers is %.2f", avg);
13. **return** 0;
14. }

**Output:**

If the user enters 10, 20, and 30, the program will output:

Enter three numbers: 10 20 30

The average of the three numbers is 20.00

**Explanation:**

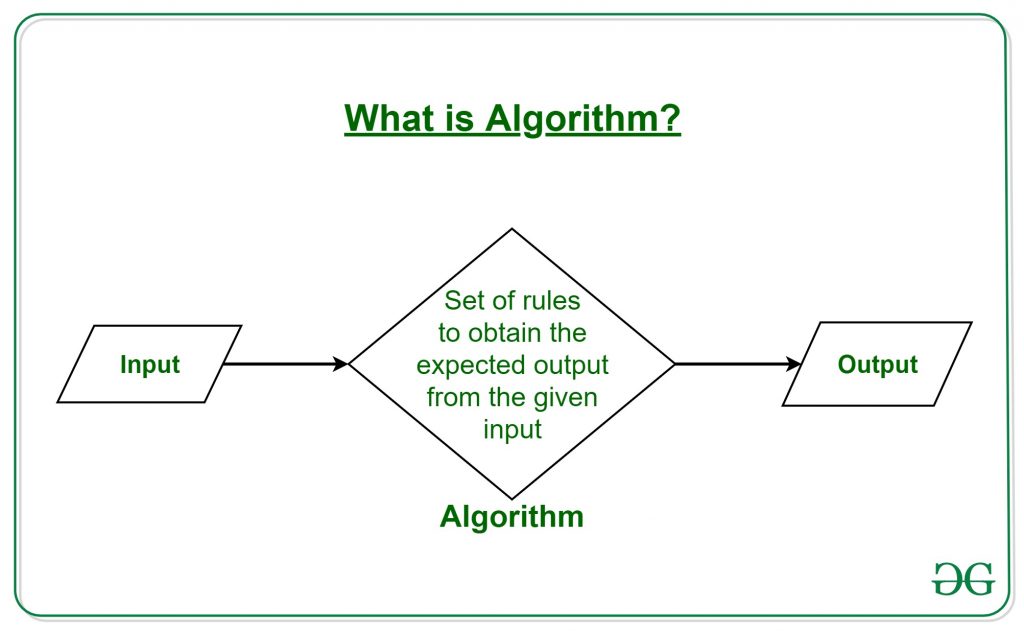
In this example, we start by declaring the variables that we will use in the program, including the **three input numbers**, the sum of the three numbers, and the average. After that, we use the **printf** and **scanf** functions to prompt the user for input and read in the three numbers. Next, we calculate the sum of the three numbers by adding them together and divide the sum by 3 to get the average. Finally, we use the **printf** function to display the average to the user.

## Pseudo-code in C/JAVA programming has the following advantages:

1. **Pseudo code** is easier to read and comprehend than actual programming code since it describes program logic in everyday language. When working on intricate algorithms or processes, this can be quite helpful.
2. **Saving time:** By working out the specifics of the program logic before creating the actual code, producing pseudo code allows you to save time in the long run. It can make the coding process simpler and help you spot any problems.
3. **Reduces errors:** Because pseudo code enables you to develop the logic of your program in a more streamlined manner, it can help to decrease errors in your code. Before writing the real code, you can use this to find probable problems, which can save time and lower the chance of introducing issues.
4. **Enhances teamwork:** Pseudo coding can be utilized as a tool for team design cooperation. Regardless of programming experience, it offers a universal language that anyone can understand.
5. **Pseudo code** is portable because it is **language-independent** and is simple to adapt to new programming languages. Therefore, it can be used as a tool for creating **cross-platform**

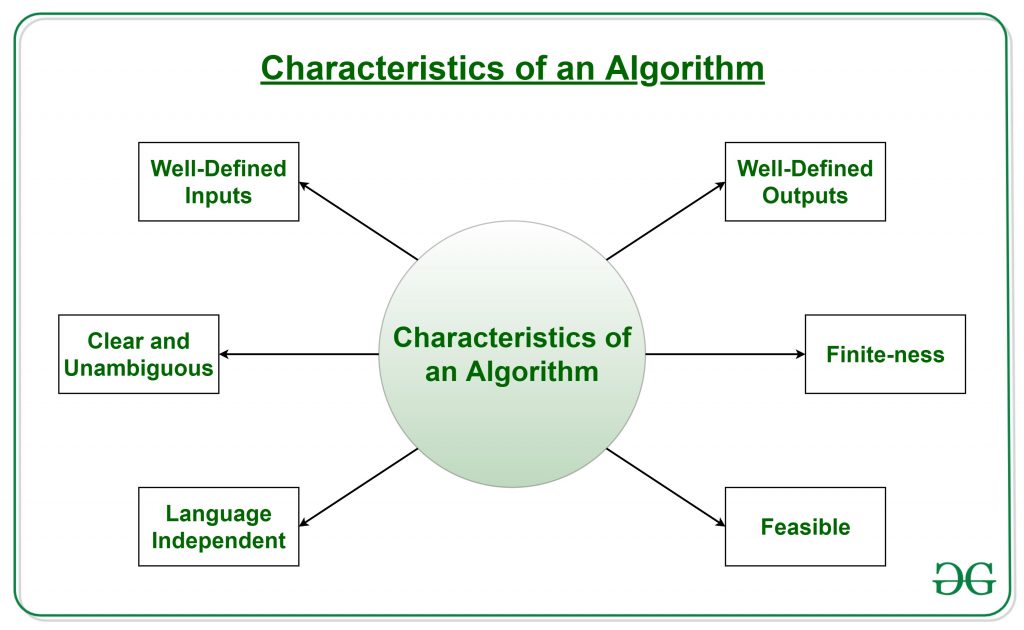
**What is Algorithm? Algorithm Basics**

The word [**Algorithm**](https://www.geeksforgeeks.org/fundamentals-of-algorithms/) means “a process or set of rules to be followed in calculations or other problem-solving operations”. Therefore Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed upon in order to get the expected results.



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.

**What are the Characteristics of an Algorithm?**



As 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. In order for some instructions to be an algorithm, it must have the following characteristics:

* **Clear and Unambiguous**: Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must 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 it should be well-defined as well.
* **Finite-ness:** The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.
* **Feasible:** The algorithm must be simple, generic and practical, such that it can be executed upon 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.

**Advantages of Algorithms:**

* It is easy to understand.
* Algorithm is a step-wise representation of a solution to a given problem.
* In Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

**Disadvantages of Algorithms:**

* Writing an algorithm takes a long time so it is time-consuming.
* Branching and Looping statements are difficult to show in Algorithms.

**How to Design an Algorithm?**

In order to write an algorithm, following things are needed as a pre-requisite: 

1. The **problem** that is to be solved by this algorithm.
2. The **constraints** of the problem that must be considered while solving the problem.
3. The **input** to be taken to solve the problem.
4. The **output** to be expected when the problem the is solved.
5. The **solution** to this problem, in the given constraints.

Then the algorithm is written with the help of above parameters such that it solves the problem.  
**Example:** Consider the example to add three numbers and print the sum.

* **Step 1: Fulfilling the pre-requisites**   
  As discussed above, in order to write an algorithm, its pre-requisites must be fulfilled.
  1. **The problem that is to be solved by this algorithm**: Add 3 numbers and print their sum.
  2. **The constraints of the problem that must be considered while solving the problem**: The numbers must contain only digits and no other characters.
  3. **The input to be taken to solve the problem:** The three numbers to be added.
  4. **The output to be expected when the problem the is solved:** The sum of the three numbers taken as the input.
  5. **The solution to this problem, in the given constraints:** The solution consists of adding the 3 numbers. It can be done with the help of ‘+’ operator, or bit-wise, or any other method.
* **Step 2: Designing the algorithm**  
  Now let’s design the algorithm with the help of above pre-requisites:  
  **Algorithm to add 3 numbers and print their sum:**
  1. START
  2. Declare 3 integer variables num1, num2 and num3.
  3. Take the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
  4. Declare an integer variable sum to store the resultant sum of the 3 numbers.
  5. Add the 3 numbers and store the result in the variable sum.
  6. Print the value of variable sum
  7. END

1. **Priori Analysis:** “Priori” means “before”. Hence Priori analysis means checking the algorithm before its implementation. In this, the algorithm is checked when it is written in the form of theoretical steps. This Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation. This is done usually by the algorithm designer. It is in this method, that the Algorithm Complexity is determined.
2. **Posterior Analysis:** “Posterior” means “after”. Hence Posterior analysis means checking the algorithm after its implementation. In this, the algorithm is checked by implementing it in any programming language and executing it. This analysis helps to get the actual and real analysis report about correctness, space required, time consumed etc.

* **Time Factor**: Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
* **Space Factor**: Space is measured by counting the maximum memory space required by the algorithm.

1. **Space Complexity:** Space complexity of an algorithm refers to the amount of memory that this algorithm requires to execute and get the result. This can be for inputs, temporary operations, or outputs.  
   **How to calculate Space Complexity?**  
   The space complexity of an algorithm is calculated by determining following 2 components:
   * **Fixed Part:** This refers to the space that is definitely required by the algorithm. For example, input variables, output variables, program size, etc.
   * **Variable Part:** This refers to the space that can be different based on the implementation of the algorithm. For example, temporary variables, dynamic memory allocation, recursion stack space, etc.
2. **Time Complexity:** Time complexity of an algorithm refers to the amount of time that this algorithm requires to execute and get the result. This can be for normal operations, conditional if-else statements, loop statements, etc.  
   **How to calculate Time Complexity?**  
   The time complexity of an algorithm is also calculated by determining following 2 components:
   * **Constant time part:** Any instruction that is executed just once comes in this part. For example, input, output, if-else, switch, etc.
   * **Variable Time Part:** Any instruction that is executed more than once, say n times, comes in this part. For example, loops, recursion, etc.

===============================================================

The algorithm and flowchart are two types of tools to explain the process of a program. In this page, we discuss the differences between an algorithm and a flowchart and how to create a flowchart to illustrate the algorithm visually.

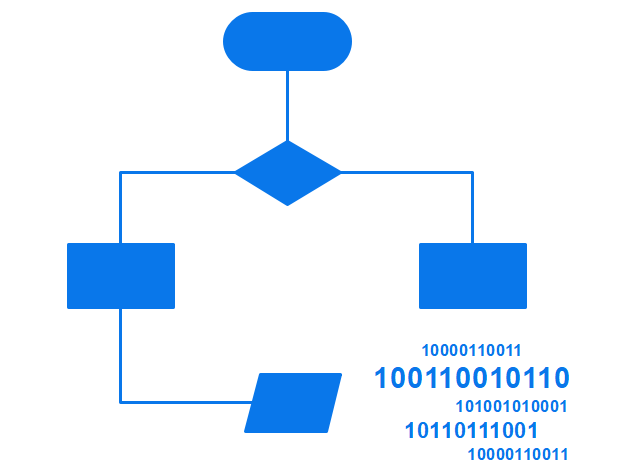
Algorithms and flowcharts are two different tools that are helpful for creating new programs, especially in computer programming. An algorithm is a step-by-step analysis of the process, while a flowchart explains the steps of a program in a graphical way.

* [Part 1: Definition of Algorithm](https://www.edrawsoft.com/explain-algorithm-flowchart.html#algorithm)
* [Part 2: Definition of Flowchart](https://www.edrawsoft.com/explain-algorithm-flowchart.html#flowchart)
* [Part 3: Difference between Algorithm and Flowchart](https://www.edrawsoft.com/explain-algorithm-flowchart.html#difference)
* [Part 4: Types of Algorithm](https://www.edrawsoft.com/explain-algorithm-flowchart.html#types)
* [Part 5: Use Flowcharts to Represent Algorithms](https://www.edrawsoft.com/explain-algorithm-flowchart.html#represent)
* [Part 6: Conclusion](https://www.edrawsoft.com/explain-algorithm-flowchart.html#conclusion)

**Part 1: Definition of Algorithm**

Writing a logical step-by-step method to solve the problem is called the [**algorithm**](https://www.edrawsoft.com/algorithm-definition.html). In other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step in the process.

An algorithm includes calculations, reasoning, and data processing. Algorithms can be presented by natural languages, pseudocode, and flowcharts, etc.



**Part 2: Definition of Flowchart**

A [**flowchart**](https://www.edrawsoft.com/flowchart-definition.html) is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of using a flowchart is to analyze different methods. Several standard symbols are applied in a flowchart:

|  |  |
| --- | --- |
| Terminal Box - Start / End | Terminal Box Figure |
| Input / Output | Input / Output Figure |
| Process / Instruction | Process Figure |
| Decision | Decision Figure |
| Connector / Arrow | Connector Figure |

The symbols above represent different parts of a flowchart. The process in a flowchart can be expressed through boxes and arrows with different sizes and colors. In a flowchart, we can easily highlight certain elements and the relationships between each part.

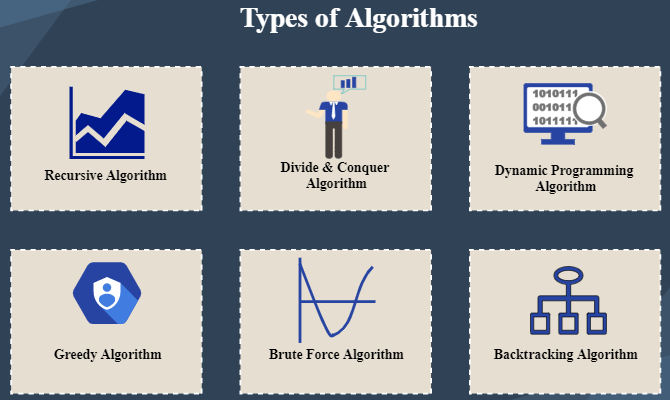
**Part 3: Difference between Algorithm and Flowchart**

If you compare a flowchart to a movie, then an algorithm is the story of that movie. In other words, **an algorithm is the core of a flowchart.** Actually, in the field of computer programming, there are many differences between algorithm and flowchart regarding various aspects, such as the accuracy, the way they display, and the way people feel about them. Below is a table illustrating the differences between them in detail.

|  |  |
| --- | --- |
| Algorithm | Flowchart |
| It is a procedure for solving problems. | It is a graphic representation of a process. |
| The process is shown in step-by-step instruction. | The process is shown in block-by-block information diagram. |
| It is complex and difficult to understand. | It is intuitive and easy to understand. |
| It is convenient to debug errors. | It is hard to debug errors. |
| The solution is showcased in natural language. | The solution is showcased in pictorial format. |
| It is somewhat easier to solve complex problem. | It is hard to solve complex problem. |
| It costs more time to create an algorithm. | It costs less time to create a flowchart. |

**Part 4: Types of Algorithm**

It is not surprising that algorithms are widely used in computer programming. However, it can be applied to solving mathematical problems and even in everyday life. Here comes a question: how many types of algorithms? According to Dr. Christoph Koutschan, a computer scientist working at the Research Institute for Symbolic Computation (RISC) in Austria, he has surveyed voting for the important types of algorithms. As a result, he has listed 32 crucial algorithms in computer science. Despite the complexity of algorithms, we can generally divide algorithms into six fundamental types based on their function.

[](https://www.edrawmax.com/templates/1011834/)

**1. Recursive Algorithm**

It refers to a way to solve problems by repeatedly breaking down the problem into sub-problems of the same kind. The classic example of using a recursive algorithm to solve problems is the Tower of Hanoi.

**2. Divide and Conquer Algorithm**

Traditionally, the divide and conquer algorithm consists of two parts: 1. breaking down a problem into some smaller independent sub-problems of the same type; 2. finding the final solution of the original issues after solving these more minor problems separately.

The key points of the divide and conquer algorithm are:

* If you can find the repeated sub-problems and the loop substructure of the original problem, you may quickly turn the original problem into a small, simple issue.
* Try to break down the whole solution into various steps (different steps need different solutions) to make the process easier.
* Are sub-problems easy to solve? If not, the original problem may cost lots of time.

**3. Dynamic Programming Algorithm**

Developed by Richard Bellman in the 1950s, the dynamic programming algorithm is generally used for optimization problems. In this type of algorithm, past results are collected for future use. Like the divide and conquer algorithm, a dynamic programming algorithm simplifies a complex problem by breaking it down into some simple sub-problems. However, the most significant difference between them is that the latter requires overlapping sub-problems, while the former doesn’t need to.

**4. Greedy Algorithm**

This is another way of solving optimization problems – greedy algorithm. It refers to always finding the best solution in every step instead of considering the overall optimality. That is to say, what he has done is just at a local optimum. Due to the limitations of the greedy algorithm, it has to be noted that the key to choosing a greedy algorithm is whether to consider any consequences in the future.

**5. Brute Force Algorithm**

The brute force algorithm is a simple and straightforward solution to the problem, generally based on the description of the problem and the definition of the concept involved. You can also use "just do it!" to describe the strategy of brute force. In short, a brute force algorithm is considered as one of the simplest algorithms, which iterates all possibilities and ends up with a satisfactory solution.

**6. Backtracking Algorithm**

Based on a depth-first recursive search, the backtracking algorithm focusing on finding the solution to the problem during the enumeration-like searching process. When it cannot satisfy the condition, it will return “backtracking” and tries another path. It is suitable for solving large and complicated problems, which gains the reputation of the “general solution method.” One of the most famous backtracking algorithm example it the eight queens puzzle.

**Part 5: Use Flowcharts to Represent Algorithms**

Now that we have learned the definitions of algorithm and flowchart, how can we use a flowchart to represent an algorithm? To create an algorithm flowchart, we need to use a handy diagramming tool like EdrawMax to finish the work.

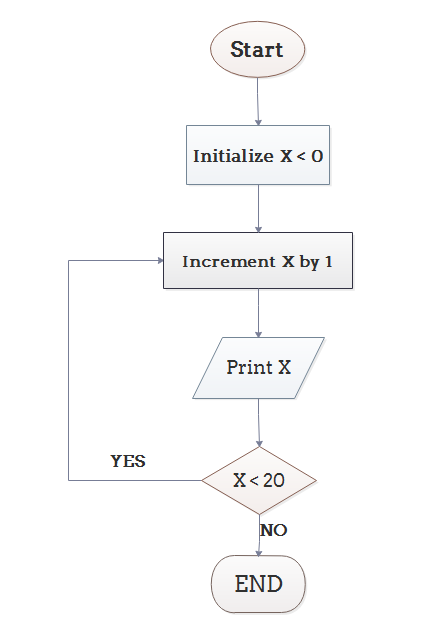
Algorithms are mainly used for mathematical and computer programs, whilst flowcharts can be used to describe all sorts of processes: business, educational, personal, and algorithms. So flowcharts are often used as a program planning tool to organize the program's step-by-step process visually. Here are some examples:

**Example 1: Print 1 to 20:**

**Algorithm:**

* Step 1: Initialize X as 0,
* Step 2: Increment X by 1,
* Step 3: Print X,
* Step 4: If X is less than 20 then go back to step 2.

**Flowchart:**

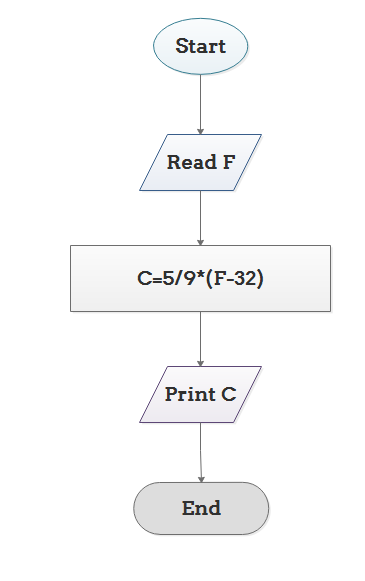


**Example 2: Convert Temperature from Fahrenheit (℉) to Celsius (℃)**

**Algorithm:**

* Step 1: Read temperature in Fahrenheit,
* Step 2: Calculate temperature with formula C=5/9\*(F-32),
* Step 3: Print C.

**Flowchart:**

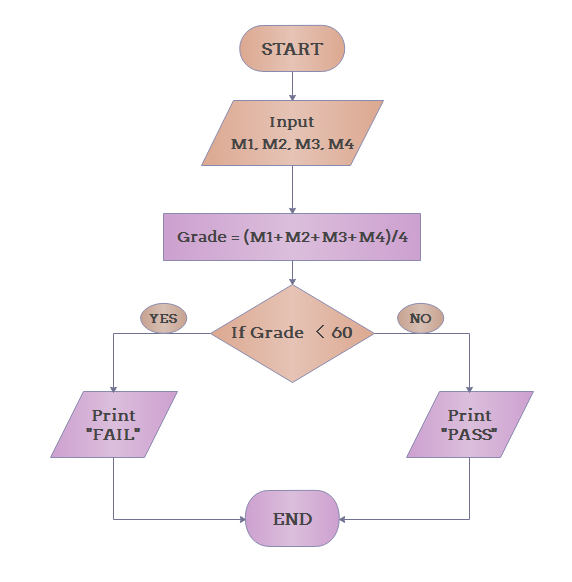


**Example 3: Determine Whether A Student Passed the Exam or Not:**

**Algorithm:**

* Step 1: Input grades of 4 courses M1, M2, M3 and M4,
* Step 2: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"
* Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".

**Flowchart:**



**Part 6: Conclusion**

From the above, we can come to the conclusion that a flowchart is a pictorial representation of an algorithm, an algorithm can be expressed and analyzed through a flowchart. An algorithm shows you every step of reaching the final solution, while a flowchart shows you how to carry out the process by connecting each step. An algorithm uses mainly words to describe the steps while you can create a flowchart with flowchart symbols to make the process more logical.,