INTRODUCTION :

Hello friend !!!

Welcome to my world !!!

DO ASK ME ANY QUERRY

DO ASK ME FOR ANY SOFTWARE and lot of more u can share with me any time, ,,,,,

ALSO TELL UR FRIENDS ABOUT ME !!!!!!

Regards

Aniruddha Rathi

Call / SMS me any time from any where at (+91) 9423575556

email id's

aniruddha.rathi@gmail.com

aniruddha.rathi@rediffmail.com

aniruddha.rathi@hotmail.com and

cacaca333@gmail.com

also visit my blogs at

http://aniruddharathi.bravehost.com/about_me.html

http://caaniruddharathi.blogspot.com/

and

http://aniruddharathi.blogspot.com/

waiting 4ur replies and comments …

thanx

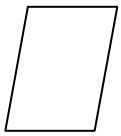
dont 4gt to tel abt me to all of ur frnds ....

GO TO NEXT PAGE FOR THE BASIC INFORMATION ABOUT “FLOW CHARTS “

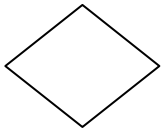
A Brief Synopsis of Flowcharts

Flowcharts are pictorial representations of processes or events. For example, there might be a flowchart representing the process of loading a bus or making coffee. Flowcharts are useful when one needs to sort through or understand a complex process. As programming requires understanding of problems at the micro level, flowcharts help in logically dissecting the problem and arranging it in sequential connections. Expert programmers typically do not need flowcharts for simple problems. However, complex tasks, especially ones that require teamwork, absolutely entail use of flowcharts.

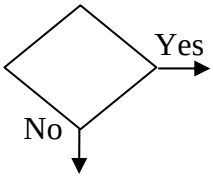
Every problem can be thought of as a process. A process has several separate constituent parts, such as inputting data into a process, evaluating a situation or seeking clarification (decision making) and end of process. There can be many more sub-groups, but for now, these are enough. When drawing a flowchart, we use different shapes to symbolize the various features. Process flow is given by the direction of the arrow i.e., a process proceeds only in the direction of the arrow. A given feature may have several entry points and generally, just one exit point. Find below the different shapes used to draw a flowchart[†].

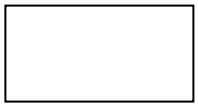


Enter data

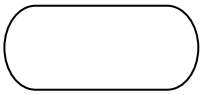


Decision. Yes or no or pick an alternative. Generally these are accompanied by the responses and the path to be followed for a given response (shown below).

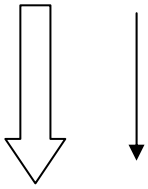




A process activity. For example, a computation with the data input. There can be several activities in a process.



End of the flowchart/process.



Direction of flow

Note that flowcharts for a given problem can vary from very general to very specific. For example, a flowchart for traveling from Fairbanks to Seattle can be a two step process:

- buy a plane ticket
- fly to Seattle.

However, a more detailed flowchart may look something like this:

- Make a decision: Fly or drive?
  - Decided to fly
- Set itinerary
- Search for the cheapest (best) ticket
- Pack clothes etc
- Get a ride to FAI
- Hop on board (hopefully, you didn't get bumped :-)
- Get off the plane

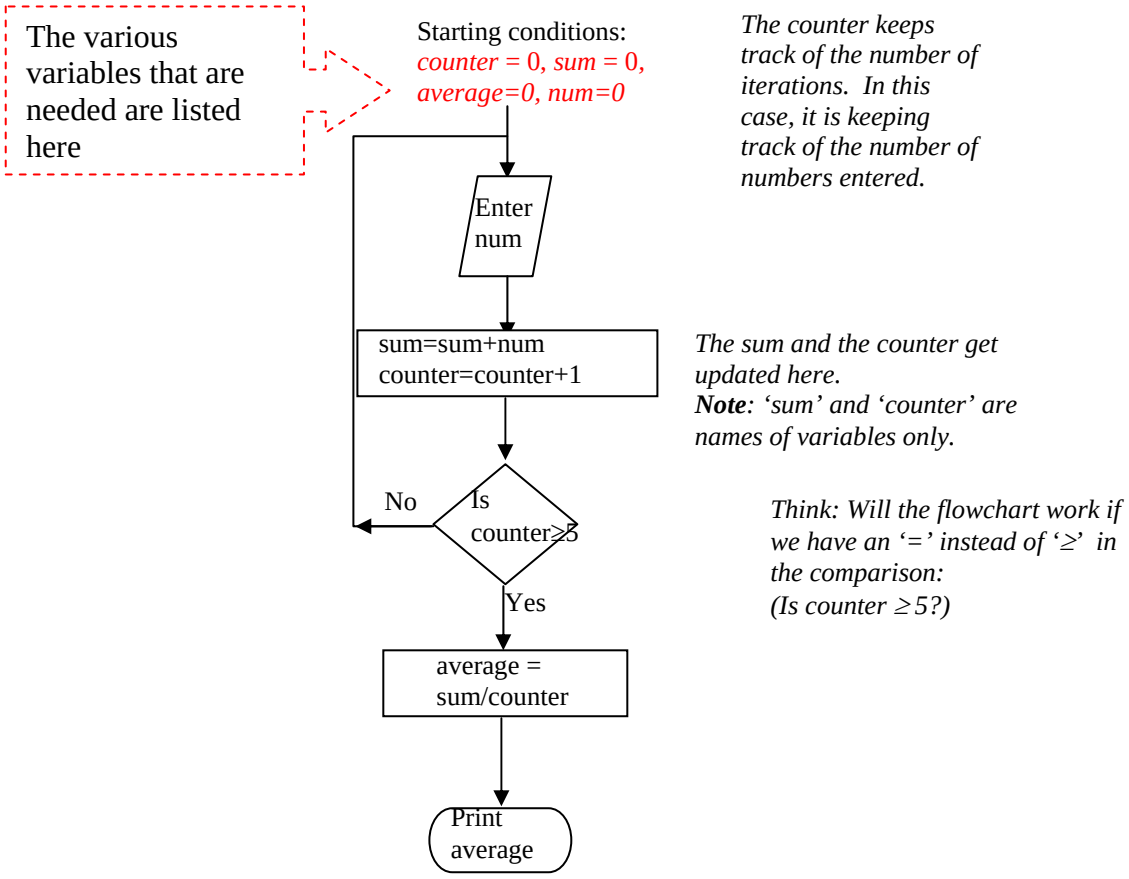
Which one is more useful? Depends on the problem and the person whose problem it is. When the boss orders you to go to Seattle, his flowchart is a one step process: John Doe will go to Seattle. Your flowchart, on the other hand, will be more like the second one. Why? Because as you actually execute the task, you *have* to go through all the steps. You can not, for example, say you will board the plane without going to the airport. This brings us to flowcharts for this class. What level of detail would we want? As the programmers' job is to convert into computer code a given process, they cannot skip a step. Therefore, your flowcharts will include all details required to implement a task.

In this class, we look at flowcharts with programming in mind. Therefore, the flowchart has to be designed not only with the problem in mind, but also with an awareness of programming/computing issues.

Some examples of flowcharts follow. Note the use of **variables** to represent the different items of a process. Variables are needed for the computer to store information. To add five numbers, one must first remember the numbers[1]. Therefore, a variable is needed to store the number when it is entered (this variable may be reused to enter other numbers as well). Also, as a computer does not explicitly know to count, any program that involves counting will need a variable that stores the count i.e. how many numbers have I dealt with so far? This is akin to the fingers that kids use to count. Finally, one may store the sum in a distinct variable. Similarly, if the average of 5 numbers is being computed, the various items that need to be tracked are: i) the entered numbers ii) the number of numbers iii) the sum of numbers and iv) the answer or the average. We could use variables called *num*, *counter*, *sum* and *average*, respectively, to represent the mentioned items. The examples below will make this more clear.

Example Flowcharts

Example 1. The following flowchart computes the average of 5 numbers.



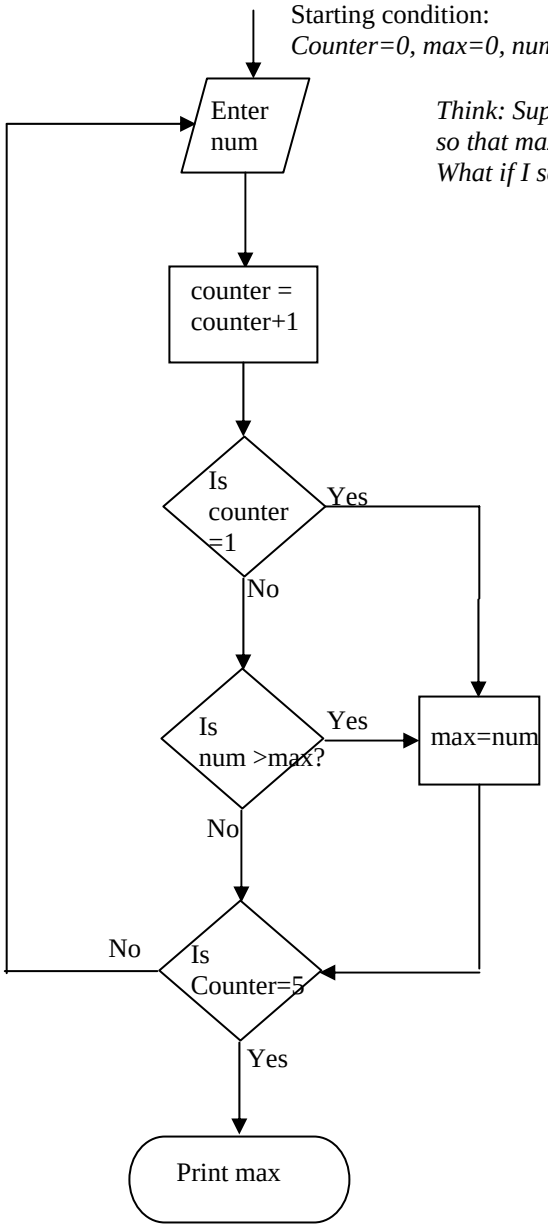
In the beginning, the number of numbers entered is zero, and therefore, so is their sum. The first step is to enter a number, say x. Thus, the variable 'num' has a value of 'x' (note how the initial value of 'num', zero, was *replaced* by the entered value). Next, we *update* the sum, which is now, *old value of sum + x*, i.e., 0+x = x. The counter, which is simply a variable (i.e., 'counter' is a name), is incremented to 1. We next check if 5 numbers have been entered yet, and since it hasn't been, we go back up to enter the next number, say y. The sum is updated to, *old value of sum + y*, i.e., x+y, while the counter is incremented to 2. For the next number, z, the sum is x+y+z, while the counter is 3. This continues till 5 numbers have been entered, at which point the counter becomes 5. Therefore, it follows a different path so that the average is computed, and printed, to end the flow. See the last page of this handout for a simple example that may help you understand the use of variables.

Assuming that the numbers that are entered are 1,-3,4,10 and 6, the various variables change with the progress of the process as shown below.

	Start Conditions	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
Num	0	1	-3	4	10	6
Counter	0	1	2	3	4	5
Sum	0	1	-2	2	12	18
Average	0	0	0	0	0	3.6

Note: The underlined variables are those that got REPLACED or OVERWRITTEN by a new value without regards to their old value. The non-underlined variables, however, do not get replaced without regards to their old value; their new and changed value depends on their previous value.

Example 2. The following flowchart determines the largest of 5 numbers.

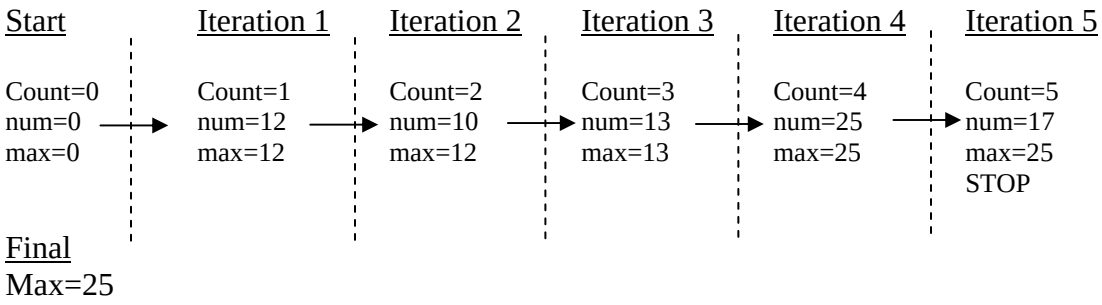


Think: Suppose I alter the starting condition so that max=999. Will the flowchart work? What if I set the starting counter to 999??

In the beginning, there is no maximum number. Therefore, the first number is the default *max*. Each number is then tested against *max*. Whenever a number exceeds *max*, *max* gets over written.

This problem is slightly more complex than the earlier one. At the outset, we have a dummy *maximum* number, whose value is 0. When the first number is entered, we update the counter. Next, we check if the entered number is the first one. If it is, the *maximum* is set to that number. Thus, it does not matter what the *maximum* is first set to; it always gets overwritten as soon as the first number is entered. (The counter gets overwritten too. Then, is the counter a dummy too? What if the counter was set to 200? Will the flowchart work?) Then we check if the counter is 5. In this case, it isn't, and therefore, the next number is entered. In the second iteration, the entered number is not the first, and therefore, continues on to be compared with *maximum*. If it is greater than *maximum*, the *maximum* gets overwritten by it, else, the *maximum* remains unchanged. This continues till the 5^{th} number is entered, when the *maximum* is printed.

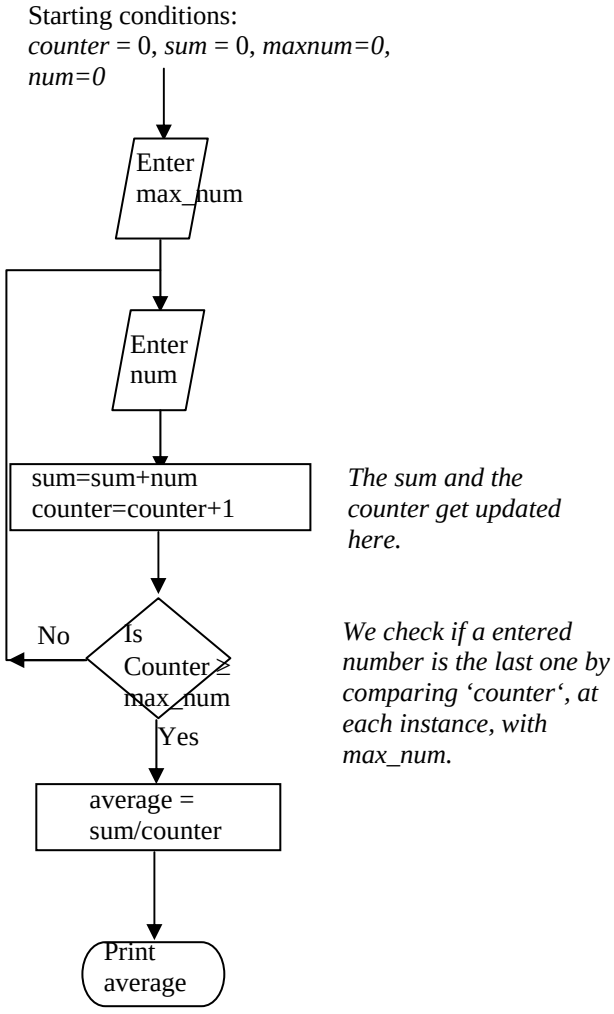
Suppose we enter the numbers 12, 10, 13, 25, 17. The changes in the process variables with iterations is given in the next page.



Final
Max=25

Example 3. We will modify Example 1 to make it more robust. Instead of computing the average of 5 numbers, we give the user the option of entering as many numbers as they want.

Note: We use the variable *max_num* to represent the maximum number of numbers the user will enter. Asking the user right at the beginning how many numbers he/she will enter makes the task very simple.



On re-using variables

For some of you, it may be confusing how a single variable ‘num’ is re-used repeatedly to enter several numbers and how, in conjunction with ‘sum’, we are able to add 5, 10 or even a million numbers. The following example may help you understand the concept.

During Christmas, we often see people from the Salvation Army standing outside grocery stores for donations. The person usually has a cup where you drop your change i.e. the person receives your donation in the cup. Once the donation is received, the person dumps the coins you donated, into a box. This cycle is repeated with every donation. The cup only holds one donation at a given time, while the box holds the sum total of all donations. This is similar to the “num” and “sum” variables. As soon as a donation is made and the contents transferred to the box (or to the sum variable), the person need not remember the previous donation(s) to know the sum total of the donations. All that he/she needs to do is to count the amount in the box (or print the “sum” variable to see its value).