

ALGORITHMS

The term **algorithm** originally referred to any computation performed via a set of rules applied to numbers written in decimal form. The word is derived from the phonetic pronunciation of the last name of *Abu Ja'far Mohammed ibn Musa al-Khowarizmi*, who was an Arabic mathematician who invented a set of rules for performing the four basic arithmetic operations (addition, subtraction, multiplication and division) on decimal numbers.

An algorithm is a representation of a solution to a problem. If a problem can be defined as a difference between a desired situation and the current situation in which one is, then a problem solution is a procedure, or method, for transforming the current situation to the desired one. We solve many such trivial problems every day without even thinking about it, for example making breakfast, travelling to the workplace etc. But the solution to such problems requires little intellectual effort and is relatively unimportant. However, the solution of a more interesting problem of more importance usually involves stating the problem in an understandable form and communicating the solution to others. In the case where a computer is part of the means of solving the problem, a procedure, explicitly stating the steps leading to the solution, must be transmitted to the computer. This concept of problem solution and communication makes the study of algorithms important to computer science.

Throughout history, man has thought of ever more elegant ways of reducing the amount of labour needed to do things. A computer has immense potential for saving time/energy, as most (computational) tasks that are repetitive or can be generalised can be done by a computer. For a computer to perform a desired task, a method for carrying out some sequence of events, resulting in accomplishing the task, must somehow be described to the computer. The algorithm can be described on many levels because the algorithm is just the procedure of steps to take and get the result. The language used to describe an algorithm to other people will be quite different from that which is used by the computer; however the actual algorithm will in essence be the same. An example of an algorithm people use would be a recipe to make a cake.

"4 extra-large eggs, beaten

1&1/2 C. stock

1/2 teaspoon salt

1 scallion, minced

1 C. small shrimp or lobster flakes

1 t. soy sauce 1 Tablespoon oil

1. Mix all the ingredients, except the oil, in a deep bowl.
2. Put 1" water in wide pot, then place deep bowl of batter inside.
3. Cover pot tightly and steam 15 min.
4. Heat oil very hot and pour over custard.
5. Steam 5 more min. Serves 4 people"

This breaks down 'Making Chinese egg custard' into smaller steps. To make the product one still needs to know how to execute each of the steps in the procedure and understand all of the terms.

Definition:

*A **procedure** is a finite sequence of well-defined instructions, each of which can be mechanically carried out in a finite amount of time.*

The procedure must break up the problem solution into parts that the recipient party can understand and execute. In the case of a computer, the problem solution is usually in the form of a program that encompasses the algorithm and explains to the computer a clearly defined procedure for achieving the solution. The procedure must consist of smaller steps each of which the computers understand. There may be no ambiguities in the translation of the procedure into the necessary action to be taken. A program is then just a specific realization of an algorithm, which may be executed on a physical device.

A computer is essentially a physical device designed to carry out a collection of primitive actions. A procedure is a sequence of instructions written in terms of which evoke a proper operation. To make effective use of an algorithm on a computer one must not only find and understand a solution to the problem but also convey the algorithm to the computer, giving the correct sequence of understood commands that represent the same algorithm.

Definition:

*An **algorithm** is procedure consisting of a finite set of unambiguous rules (instructions) which specify a finite sequence of operations that provides the solution to a problem, or to a specific class of problems for any allowable set of input quantities (if there are inputs). In other word, an **algorithm** is a step-by-step procedure to solve a given problem*

Alternatively, we can define an algorithm as a set or list of instructions for carrying out some process step by step. A recipe in a cookbook is an excellent example of an algorithm. The recipe includes the requirements for the cooking or ingredients and the method of cooking them until you end up with a nice cooked dish.

In the same way, algorithms executed by a computer can combine millions of elementary steps, such as additions and subtractions, into a complicated mathematical calculation. Also by means of algorithms, a computer can control a manufacturing process or coordinate the reservations of an airline as they are received from the ticket offices all over the country. Algorithms for such large-scale processes are, of course, very complex, but they are built up from pieces.

One of the obstacles to overcome in using a computer to solve your problems is that of translating the idea of the algorithm to computer code (program). People cannot normally understand the actual machine code that the computer needs to run a program, so programs are written in a programming language such as C or Pascal, which is then converted into machine code for the computer to run.

In the problem-solving phase of computer programming, you will be designing algorithms. This means that you will have to be conscious of the strategies you use to solve problems in order to apply them to programming problems. These algorithms can be designed though the use of **flowcharts** or **pseudocode**.

1.1 Flowcharts

Flowcharting is a tool developed in the computer industry, for showing the steps involved in a process. A flowchart is a diagram made up of *boxes, diamonds* and other shapes, *connected by arrows* - each shape represents a step in the process, and the arrows show the order in which they occur. Flowcharting combines symbols and flowlines, to show figuratively the operation of an algorithm.

In computing, there are dozens of different symbols used in flowcharting (there are even national and international flowcharting symbol standards). In business process analysis, a couple of symbols are sufficient. A box with text inside indicates a step in the process, while a diamond with text represents a decision point. See the figure for an example.






If the flowchart is too messy to draw, try starting again, but leaving out all of the decision points and concentrating on the simplest possible course. Then the session can go back and add the decision points later. It may also be useful to start by drawing a high-level flowchart for the whole organisation, with each box being a complete process that has to be filled out later.

From this common understanding can come a number of things - process improvement ideas will often arise spontaneously during a flowcharting session. And after the session, the facilitator can also draw up a written procedure - a flowcharting session is a good way of documenting a process.

Process improvement starts with an understanding of the process, and flowcharting is the first step towards process understanding.

Flowcharting Symbols

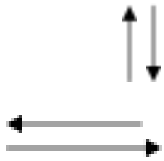
There are 6 basic symbols commonly used in flowcharting of assembly language programs: Terminal, Process, input/output, Decision, Connector and Predefined Process. This is not a complete list of all the possible flowcharting symbols; it is the ones used most often in the structure of Assembly language programming.

Symbol	Name	Function
	process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	predefined process	Used to invoke a subroutine or an interrupt program.



terminal

Indicates the starting or ending of the program, process, or interrupt program.



Flow Lines

Shows direction of flow.

General Rules for flowcharting

- 1) All boxes of the flowchart are connected with Arrows. (Not lines)
- 2) Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.
- 3) The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
- 4) Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.
- 5) Connectors are used to connect breaks in the flowchart. Examples are:
 - ✓ From one page to another page.
 - ✓ From the bottom of the page to the top of the same page.
 - ✓ An upward flow of more than 3 symbols
- 6) Subroutines and Interrupt programs have their own and independent flowcharts.
- 7) All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.
- 8) All flowcharts end with a terminal or a contentious loop.

Flowcharting uses symbols that have been in use for a number of years to represent the type of operations and/or processes being performed. The standardized format provides a common method for people to visualize problems together in the same manner. The use of standardized symbols makes the flow charts easier to interpret; however, standardizing symbols is not as important as the sequence of activities that make up the process.

Flowcharting Tips

- Chart the process the way it is really occurring. Do not document the way a written process or a manager thinks the process happens.
- People typically modify existing processes to enable a more efficient process. If the desired or theoretical process is charted, problems with the existing process will not be recognised and no improvements can be made.

Note all circumstances actually dealt with.

- Test the flow chart by trying to follow the chart to perform the process charted. If there is a problem performing the operation as charted, note any differences and modify the chart to correct. A better approach would be to have someone

unfamiliar with the process try to follow the flow chart and note questions or problems found.

- Include mental steps in the process such as decisions. These steps are sometimes left out because of familiarity with the process, however, represent sources of problems due to a possible lack of information used to make the decision can be inadequate or incorrect if performed by a different person.

Examples of Algorithms and Flowcharts

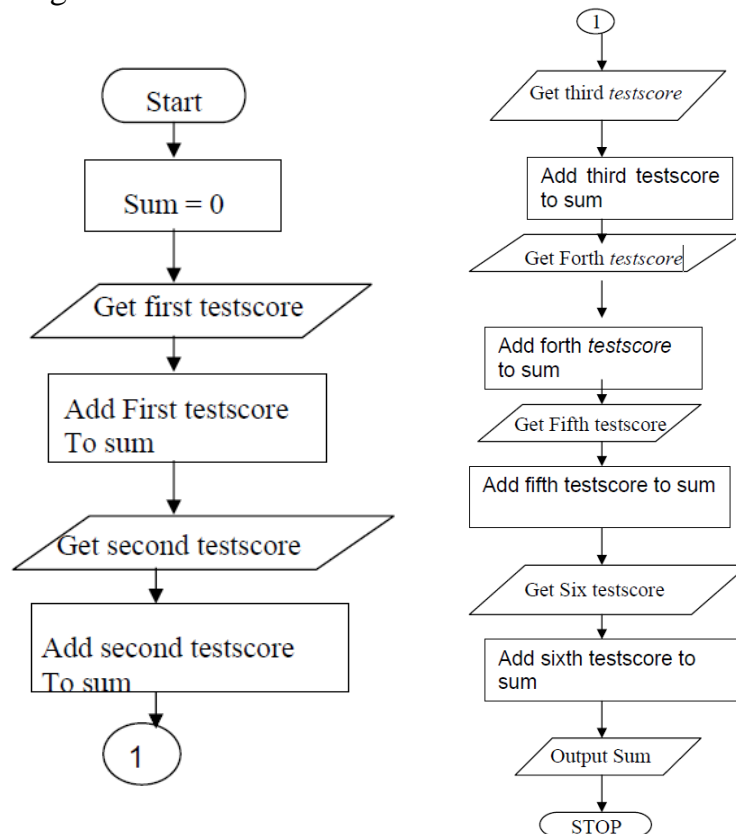
Example 1. Design an algorithm and the corresponding flowchart for adding the test scores as given below:

26, 49, 98, 87, 62, 75

a) Algorithm

- | | |
|-------------------------------|-----------------------------|
| 1. Start | 9. Get the Forth testscore |
| 2. Sum = 0 | 10. Add to sum |
| 3. Get the first testscore | 11. Get the fifth testscore |
| 4. Add first testscore to sum | 12. Add to sum |
| 5. Get the second testscore | 13. Get the sixth testscore |
| 6. Add to sum | 14. Add to sum |
| 7. Get the third testscore | 15. Output the sum |
| 8. Add to sum | 16. Stop |

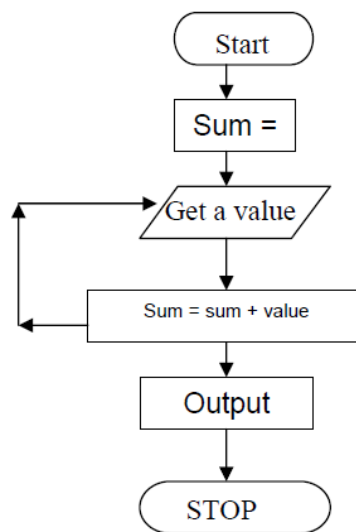
b) The corresponding flowchart is as follows:



The algorithm and the flowchart above illustrate the steps for solving the problem of adding six test scores. Where one test score is added to sum at a time. Both the algorithm and flowchart should always have a Start step at the beginning of the algorithm or flowchart and at least one stop step at the end, or anywhere in the algorithm or flowchart. Since we want the sum of six test scores, then we should have a container for the resulting sum. In this example, the container is called sum and we make sure that sum should start with a zero value by step 2.

Example 2: The problem with this algorithm is that, some of the steps appear more than once, i.e. step 5 get second number, step 7, get third number, etc. One could shorten the algorithm or flowchart as follows:

1. Start
2. Sum = 0
3. Get a value
4. sum = sum + value
5. Go to step 3 to get next Value
6. Output the sum
7. Stop



This algorithm and its corresponding flowchart are a bit shorter than the first one. In this algorithm, step 3 to 5 will be repeated, where a number is obtained and added to sum. Similarly the flowchart indicates a flowline being drawn back to the previous step indicating that the portion of the flowchart is being repeated. One problem indicates that these steps will be repeated endlessly, resulting in an endless algorithm or flowchart. The algorithm needs to be improved to eliminate this problem. In order to solve this problem, we need to add a last value to the list of numbers given. This value should be unique so that, each time we get a value, we test the value to see if we have reached the last value. In this way our algorithm will be a finite algorithm which ends in a finite number of steps as shown below. There are many ways of making the algorithm finite.

The new list of numbers will be 26, 49, 498, 9387, 48962, 1, -1. The value -1 is a unique number since all other numbers are positive.

- 1. Start
- 2. Sum = 0
- 3. Get a value
- 4. If the value is equal to -1, go to step 7
- 5. Add to sum (sum = sum + value)
- 6. Go to step 3 to get next Value
- 7. Output the sum
- 8. Stop

Corresponding flowchart

