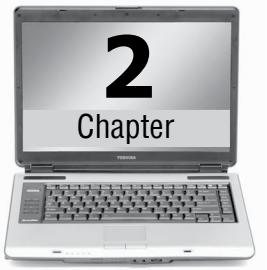


Algorithm and Flow Chart

2.1 ALGORITHM

2.1.1 Introduction

An algorithm is a set of instructions, sometimes called a procedure or a function that is used to perform a certain task. This can be a simple process, such as adding two numbers together, or a complex function, such as adding effects to an image. For example, in order to sharpen a digital photo, the algorithm would need to process each pixel in the image and determine which ones to change and how much to change them in order to make the image look sharper.

Most computer programmers spend a large percentage of their time creating algorithms. (The rest of their time is spent debugging the algorithms that don't work properly.) The goal is to create efficient algorithms that do not waste more computer resources (such as RAM and CPU time) than necessary. This can be difficult, because an algorithm that performs well on one set of data may perform poorly on other data.

As you might guess, poorly written algorithms can cause programs to run slowly and even crash. Therefore, software updates are often introduced, touting "improved stability and performance". While this sounds impressive, it also means that the algorithms in the previous versions of the software were not written as well as they could have been.

Operation

An algorithm generally takes some input, carries out a number of effective steps in a finite

Given a list of numbers, you can easily order them from largest to smallest with the simple instruction "Sort these numbers." A computer, however, needs more detail to sort numbers. It must be told to search for the smallest number, how to find the smallest number, how to compare numbers together, etc. The operation "Sort these numbers" is ambiguous to a computer because the computer has no basic operations for sorting. Basic operations used for writing algorithms are known as primitive operations or primitives. When an algorithm is written in computer primitives, then the algorithm is unambiguous and the computer can execute it.

Algorithms have effectively computable operations

Each operation in an algorithm must be doable, that is, the operation must be something that is possible to do. Suppose you were given an algorithm for planting a garden where the first step instructed you to remove all large stones from the soil. This instruction may not be doable if there is a four ton rock buried just below ground level. For computers, many mathematical operations such as division by zero or finding the square root of a negative number are also impossible. These operations are not effectively computable so they cannot be used in writing algorithms.

Algorithms produce a result

In our simple definition of an algorithm, we stated that an algorithm is a set of instructions for solving a problem. Unless an algorithm produces some result, we can never be certain whether our solution is correct. Have you ever given a command to a computer and discovered that nothing changed? What was your response? You probably thought that the computer was malfunctioning because your command did not produce any type of result. Without some visible change, you have no way of determining the effect of your command. The same is true with algorithms. Only algorithms which produce results can be verified as either right or wrong.

Algorithms halt in a finite amount of time

Algorithms should be composed of a finite number of operations and they should complete their execution in a finite amount of time. Suppose we wanted to write an algorithm to print all the integers greater than 1. Our steps might look something like this:

1. Print the number 2.
2. Print the number 3.
3. Print the number 4.

While our algorithm seems to be pretty clear, we have two problems. First, the algorithm must have an infinite number of steps because there are an infinite number of integers greater than one. Second, the algorithm will run forever trying to count to infinity. These problems violate our definition that an algorithm must halt in a finite amount of time. Every algorithm must reach some operation that tells it to stop.

2.1.3 Advantages and Disadvantages of Algorithms

Advantages

The use of algorithms provides a number of advantages. One of these advantages is in the development of the procedure itself, which involves identification of the processes, major decision points, and variables necessary to solve the problem. Developing an algorithm allows and even forces examination of the solution process in a rational manner. Identification of the processes and decision points reduces the task into a series of smaller steps of more manageable size. Problems that would be difficult or impossible to solve wholesale can be approached as a series of small, solvable subproblems. The required specification aids in the identification and reduction of subconscious biases. By using an algorithm, **decision-making** becomes a more rational process.

In additional to making the process more rational, use of an algorithm will make the process more efficient and more consistent. Efficiency is an inherent result of the analysis and specification process. Consistency comes from both the use of the same specified process and increased skill in applying the process. An algorithm serves as a mnemonic device and helps ensure that variables or parts of the problem are not ignored. Presenting the solution process as an algorithm allows more precise communication. Finally, separation of the procedure steps facilitates division of labor and development of expertise.

A final benefit of the use of an algorithm comes from the improvement it makes possible. If the problem solver does not know what was done, he or she will not know what was done wrong. As time goes by and results are compared with goals, the existence of a specified solution process allows identification of weaknesses and errors in the process. Reduction of a task to a specified set of steps or algorithm is an important part of analysis, control, and evaluation.

Disadvantages

One disadvantage of algorithms is that they always terminate, which means there are some computational procedures—occasionally even useful ones—which are not algorithms. Furthermore, all computational procedures, whether they terminate or not, can only give computable results, so you cannot, for example, design a program which determines a busy beaver number more quickly than could be done by actually running the associated types of turing machines.

2.2 FLOW CHART

2.2.1 Introduction

A flow chart, or flow diagram, is a graphical representation of a process or system that details the sequencing of steps required to create output. A typical flow chart uses a set of basic symbols to represent various functions, and shows the sequence and interconnection of functions with lines and arrows. Flow charts can be used to document virtually any type of business system, from the movement of materials through machinery in a manufacturing operation to the flow of applicant information through the hiring process in a human resources department.

Each flow chart is concerned with one particular process or system. It begins with the input of data or materials into the system and traces all the procedures needed to convert the input into its final output form. Specialized flow chart symbols show the processes that take place, the actions that are performed in each step, and the relationship between various steps. Flow charts can include different levels of detail as needed, from a high-level overview of an entire system to a detailed diagram of one component process within a larger system. In any case, the flow chart shows the overall structure of the process or system, traces the flow of information and work through it, and highlights key processing and decision points.

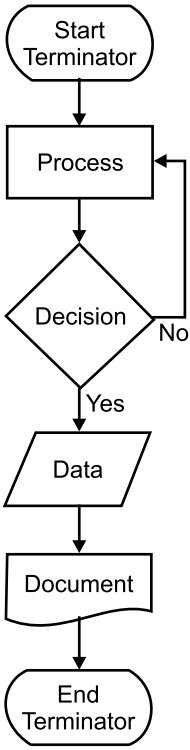
Flow charts are an important tool for the improvement of processes. By providing a graphical representation, they help project teams to identify the different elements of a process and understand the interrelationships among the various steps. Flow charts may also be used to gather information and data about a process as an aid to decision-making or performance evaluation. For example, the owner of a small advertising agency who hopes to reduce the time involved in creating a print ad might be able to use a flow chart of the process to identify and eliminate unnecessary steps. Though flow charts are relatively old design tools, they remain popular among computer programmers working on systems analysis and design. In recent years, many software programs have been developed to assist businesspeople in creating flow charts.

2.2.2 Constructing Flow Charts

Flow charts typically utilize specialized symbols. Some of the main symbols that are used to construct flow charts include:

- A round-edged rectangle to represent starting and ending activities, which are sometimes referred to as terminal activities.
- A rectangle to represent an activity or step.
- Each step or activity within a process is indicated by a single rectangle, which is known as an activity or process symbol.
- A diamond to signify a decision point. The question to be answered or decision to be made is written inside the diamond, which is known as a decision symbol. The answer determines the path that will be taken as a next step.
- Flow lines show the progression or transition from one step to another.

Constructing a flow chart involves the following main steps: (1) Define the process and identify the scope of the flow diagram; (2) Identify project team members that are to be involved in the construction of the process flow diagram; (3) Define the different steps involved in the process and the interrelationships between the different steps (all team members should help develop and agree upon the different steps for the process); (4) Finalize the diagram, involving other concerned individuals as needed and making any modifications necessary; and (5) Use the flow diagram and continuously update it as needed.



2.2.3 Advantages and Disadvantages of Flow Chart

Advantages

1. Communication: Flow Charts are better way of communicating the logic of a system to all concerned.
2. Effective Analysis: With the help of flow chart, problem can be analyzed in more effective way.
3. Proper Documentation: Program flow charts serve as a good program documentation, which is needed for various purposes.
4. Efficient Coding: The flow charts act as a guide or blueprint during the systems analysis and program development phase.
5. Proper Debugging: The flow chart helps in debugging process.
6. Efficient Program Maintenance: The maintenance of operating program becomes easy with the help of flow chart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages

Complex logic: Sometimes, the program logic is quite complicated. In that case, flow chart becomes complex and clumsy.

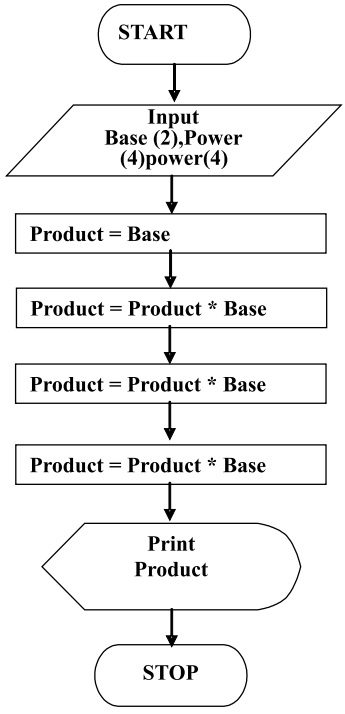
1. Alterations and Modifications: If alterations are required the flow chart may require re-drawing completely.
2. Reproduction: As the flow chart symbols cannot be typed, reproduction of flow chart becomes a problem.
3. The essentials of what is done can easily be lost in the technical details of how it is done.

2.3 EXAMPLES

Example 1: Write an algorithm and draw a flow chart to calculate 2^4.

Algorithm: Step 1: *Input* Base (2), Power (4)
Step 2: Product= Base
Step 3: Product = Product * Base
Step 4: Product = Product * Base
Step 5: Product = Product * Base
Step 6: *Print* Product

Flow chart

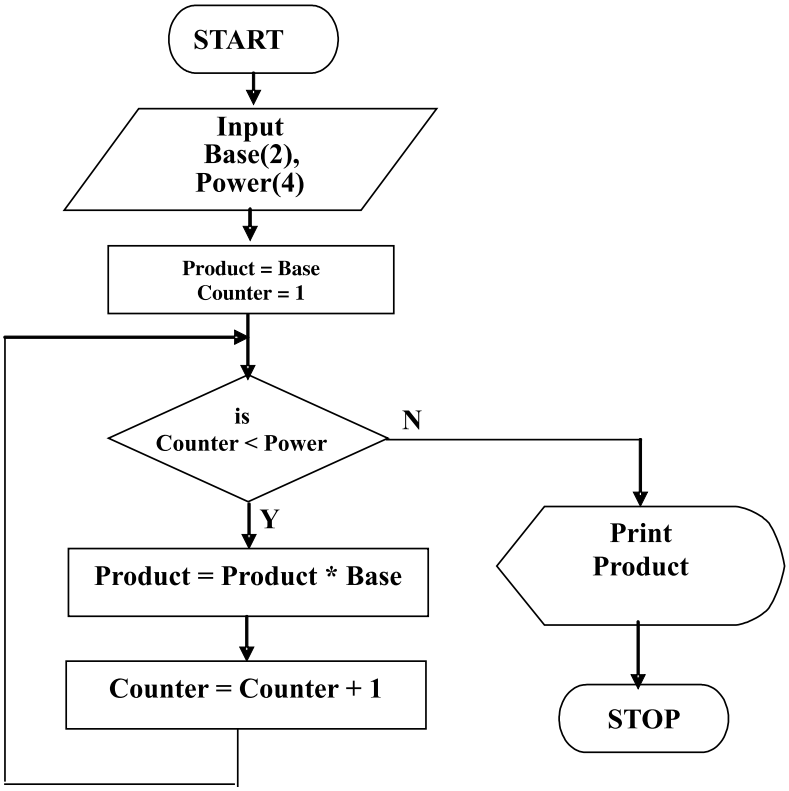


- Question: What happens if you want to calculate 2 to the power of 1000 (2^{1000})
- Answer: Use a LOOP (repeated execution of the same set of instructions)

Example 2: Write an algorithm and draw a flow chart to calculate 2^4 using a loop approach?

Algorithm:
Step 1: *Input* Base (2), Power (4)
Step 2: Product = Base
Step 3: Counter = 1
Step 4: *While* (Counter < Power)
Repeat steps 4 through 6
Step 5: Product = Product * Base
Step 6: Counter = Counter + 1
Step 7: *Print* Product

Flow chart

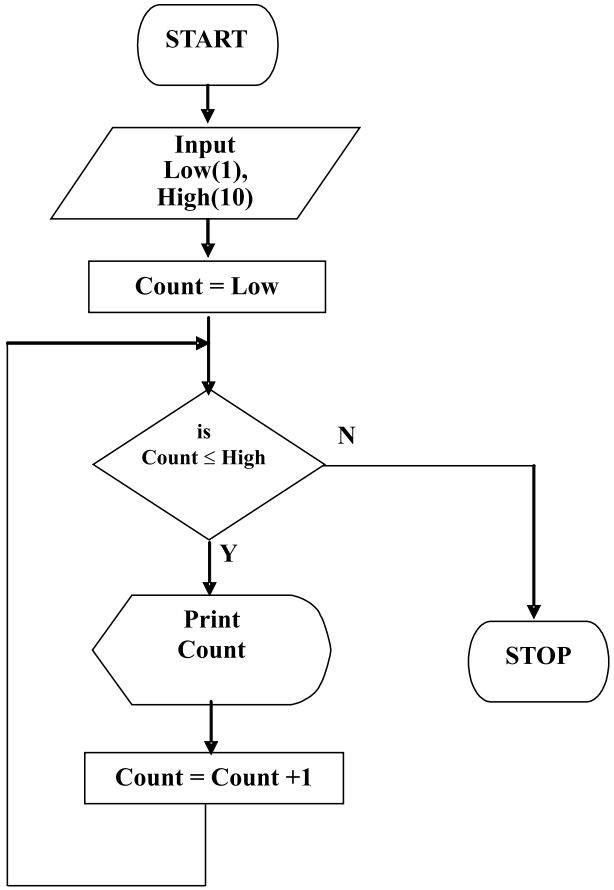


Example 3: Write down an algorithm and draw a flow chart to find and print the largest of three numbers. Read numbers one by one. (Use 5, 7, 3 as the numbers read)

Example 4: Write down an algorithm and draw a flow chart to count and print from 1 to 10.

Algorithm: Step 1: *Input* Low(1), High(10)
Step 2: Count = 1
Step 4: *While* (Count ≤ High)
Repeat steps 4 through 6
Step 5: *Print* Count
Step 6: Count = Count + 1

Flow chart

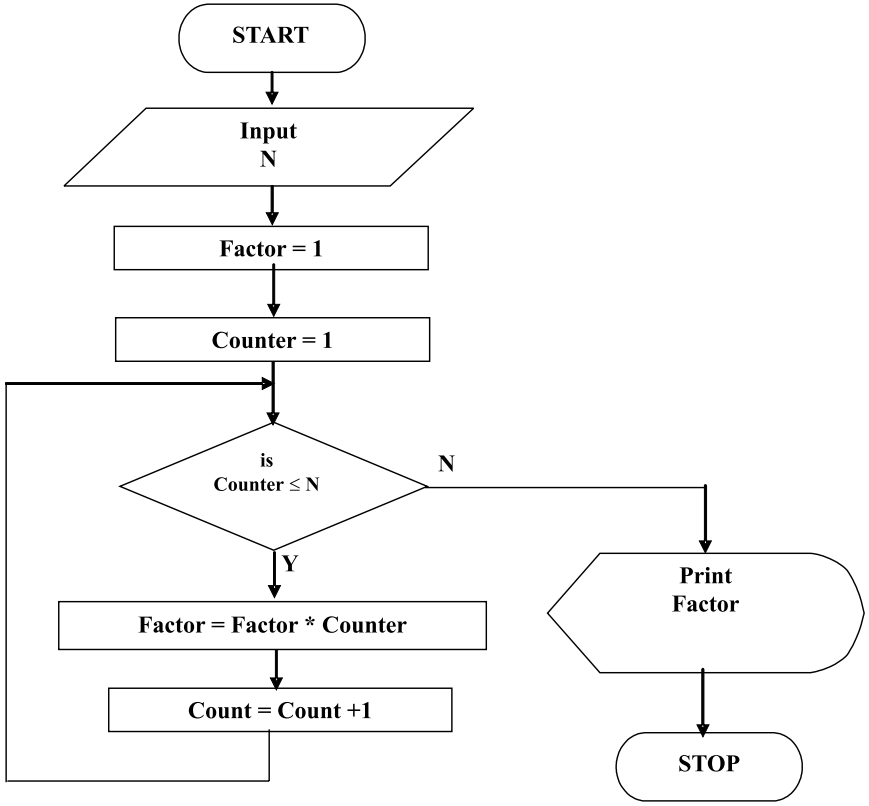


Example 5: Write an algorithm and draw a flow chart to calculate the factorial of a number (N). Verify your result by a trace table by assuming N = 5.

Hint: The factorial of N is the product of numbers from 1 to N.

Algorithm: Step 1: *Input* N
Step 2: Factor = 1
Step 3: Counter = 1
Step 4: *While* (Counter ≤ N)
Repeat steps 4 through 6
Step 5: Factor = Factor * Counter
Step 6: Counter = Counter + 1
Step 7: Print (N, Factor)

Flow chart



REVIEW EXERCISE

1. Write an algorithm and draw a flow chart to print the square of all numbers from LOW to HIGH. (Test your algorithm with LOW = 1 and HIGH = 10).
2. Write an algorithm and draw a flow chart to print the SUM of numbers from LOW to HIGH. (Test your algorithm with LOW = 3 and HIGH = 9.)
3. Write an algorithm and draw a flow chart to print all numbers between LOW and HIGH that are divisible by NUMBER.