

# An Introduction to Simple Code Design

## Introduction to Flowcharts

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analysing, designing, documenting or managing a process or program in various fields. Flowcharts originated from computer science as a tool for representing algorithms and programming logic but has extended into other areas.

<https://en.wikipedia.org/wiki/Flowchart>

## History



Flowcharts were a popular means for describing computer algorithms and are still used for this purpose in educational courses. Modern techniques such as UML activity diagrams can be considered as the extension of the flowchart and tend to be utilised for more complex ideas and algorithms.







In the 1970s the popularity of flowcharts decreased when interactive computer terminals and third-generation programming languages became the common tools of the trade, since algorithms can be expressed more concisely as source code in such a language, and also because designing algorithms using flowcharts was more likely to result in spaghetti code because of the need for “goto” statements to describe arbitrary jumps in control flow. Flow charts are still extremely useful for simple algorithms and code snippets.

<https://medium.com/@warren2lynch/a-comprehensive-guide-for-flowchart-over-50-examples-785d6dfdc380>

## Flowchart Notation

Though flowcharts can be useful writing and analysis of a program, drawing a flowchart for complex programs can be more complicated than writing the program itself. Hence, creating flowcharts for complex programs is often ignored.

Symbol	Purpose	Description
	Flow line	Indicates the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Represents the start and the end of a flowchart.

	Input/Output	Used for input and output operation.
	Processing	Used for arithmetic operations and data-manipulations
	Decision	Used for decision making between two or more alternatives.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect the flowchart portion on a different page
	Predefined Process/Function	Represents a group of statements performing one processing task.

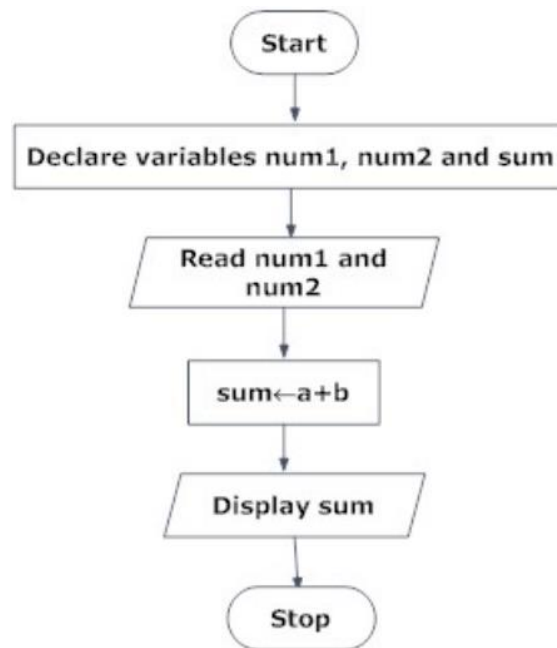
## Advantages of flowchart

Flowchart is an excellent way of communicating the logic of a program.

- Easy and efficient to analyse problem using flowchart.
- During program development cycle, the flowchart plays the role of a blueprint, which makes program development process easier.
- After successful development of a program, it needs continuous timely maintenance during the course of its operation. The flowchart makes program or system maintenance easier.
- It is easy to convert the flowchart into any programming language code

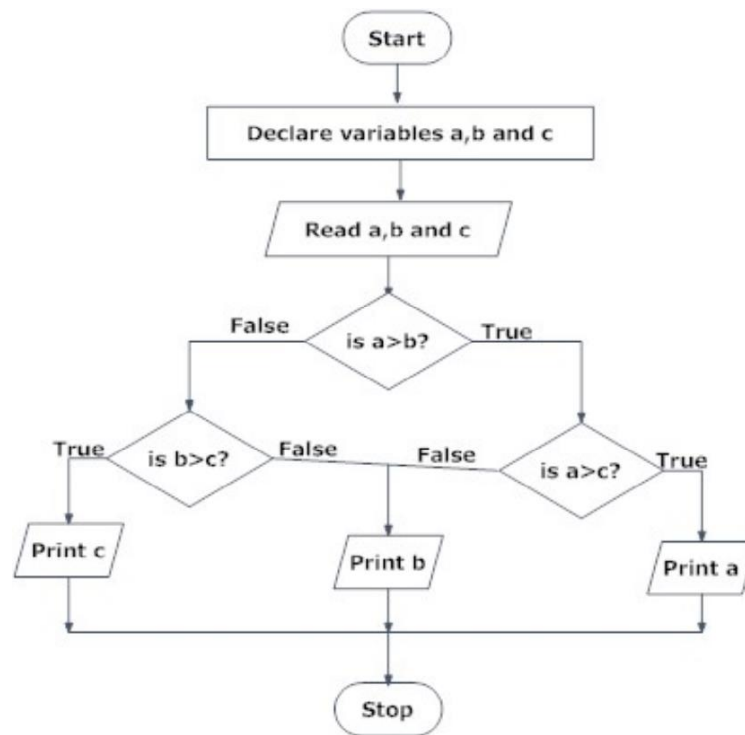
## Flowchart Examples

1. Add two numbers entered by the user.



```
static void Main(string[] args)
{
    int num1, num2, sum;
    Console.WriteLine("Enter Number One");
    num1 = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter Number Two");
    num2 = int.Parse(Console.ReadLine());
    sum = num1 + num2;
    Console.WriteLine("Sum is {0}", sum);
    Console.ReadLine();
}
```

2. Find the largest among three different numbers entered by the user.



```

static void Main(string[] args)
{
    int a, b, c;
    Console.WriteLine("Enter number a");
    a = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter Number b");
    b = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter Number c");
    c = int.Parse(Console.ReadLine());
    if (a > b)
        if (a > c)
            Console.WriteLine("Print a {0}", a);
        else
            Console.WriteLine("Print a {0}", b);
    else if (b > c)
        Console.WriteLine("Print a {0}", c);
    else
        Console.WriteLine("Print a {0}", b);
    Console.ReadLine();
}
  
```

For more examples:

<https://medium.com/@warren2lynch/a-comprehensive-guide-for-flowchart-over-50-examples-785d6dfdc380>

## Introduction to Algorithms

The word “algorithm” relates to the name of the mathematician Al-khowarizmi, which means a procedure or a technique. Software Programmers commonly uses an algorithm for planning and solving various problems. An algorithm is a sequence of steps to solve a particular problem and represents an ordered set of unambiguous steps that produces a result and terminates in a finite time.

An Algorithm has the following characteristics

- **Input:** An algorithm may or may not require input
- **Output:** Each algorithm is expected to produce at least one result
- **Definiteness:** Each instruction must be clear and unambiguous.
- **Finiteness:** If the instructions of an algorithm are executed, the algorithm should terminate after finite number of steps

The algorithm and flowchart include following three types of control structures.

1. **Sequence:** In the sequence structure, statements are placed one after the other and the execution takes place starting from up to down.
2. **Selection:** In branch control, there is a condition and according to a condition, a decision of either TRUE or FALSE is achieved. In the case of TRUE, one of the two branches is explored; but in the case of FALSE condition, the other alternative is taken. Generally, the ‘IF-THEN’ is used to represent branch control.
3. **Iteration (Loops):** The Loop or Repetition allows a statement(s) to be executed repeatedly based on certain loop condition e.g. WHILE, FOR loops.

## Algorithm Notation (Pseudo Code)

Pseudo code is a term which is often used in programming and algorithm based fields. It is a methodology that allows the programmer to represent the implementation of an algorithm. It has no syntax like any of the programming language and thus can't be compiled or interpreted by the computer.

An algorithm is a well-defined sequence of steps that provides a solution for a given problem, whereas a pseudocode is one of the methods that can be used to represent an algorithm. Here are some basic words used in pseudo code:

<b>START</b>	This is the start of your pseudocode.
<b>INPUT</b>	This is data retrieved from the user through typing or through an input device.
<b>READ / GET</b>	This is input used when reading data from a data file.
<b>PRINT, DISPLAY, SHOW</b>	This will show your output to a screen or the relevant output device.
<b>COMPUTE, CALCULATE</b>	This is used to calculate the result of an expression.
<b>SET, INIT</b>	To initialize values
<b>INCREMENT</b>	To increase the value of a variable
<b>DECREMENT</b>	To decrease the value of a variable

There are a few basic rules for writing pseudo code:

Do's:	
Use control structures	IF – ELSE, IF – ELSEIF – ELSE
Use proper naming conventions	SUM, TOTAL, MAX
Use indentation and white space	SWITCH 10: do this 20: do the other
Keep it simple and concise	INPUT firstName
Don'ts:	
Don't make the code too abstract	RETURN Epsilon- Sigma * 98 <sup>2</sup>
Don't be too generalised	GET data from source

<https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/>

## Advantages of Algorithm

- It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
- An algorithm uses a definite procedure.
- It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
- Every step in an algorithm has its own logical sequence so it is easy to debug.

The language used to write algorithm is simple and similar to day-to-day life language. The variable names are used to store the values. These variables can change in the solution steps. We can use keyword INPUT or READ or GET to accept input(s) /value(s) and keywords PRINT or WRITE or DISPLAY to output the result(s).

## Algorithm Examples

1. Add two numbers entered by the user.

Algorithm:

```
Step-1    Start
Step-2    Input two numbers A & B
Step-3    SUM = A + B
Step-4    Display SUM
Step-5    Stop
```

```
static void Main(string[] args)
{
    int A, B, SUM;
    Console.WriteLine("Enter Number A");
    A = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter Number B");
    B = int.Parse(Console.ReadLine());
    SUM = A + B;
    Console.WriteLine("Sum is {0}", SUM);
    Console.ReadLine();
}
```

2. Find the smallest of two numbers entered by the user

Algorithm:

Step-1 Start

Step-2 Input two numbers NUM1 & NUM2

Step-3 IF NUM1 < NUM2 THEN  
    print smallest is NUM1

ELSE  
    print smallest is NUM2

ENDIF

Step-4 Stop

```
static void Main(string[] args)
{
    int num1, num2;
    Console.WriteLine("Enter number num1");
    num1 = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter Number num2");
    num2 = int.Parse(Console.ReadLine());
    if (num1 > num2)
        Console.WriteLine("Print smallest is {0}", num1);
    else
        Console.WriteLine("Print smallest is {0}", num2);
    Console.ReadLine();
}
```

END OF DOCUMENT