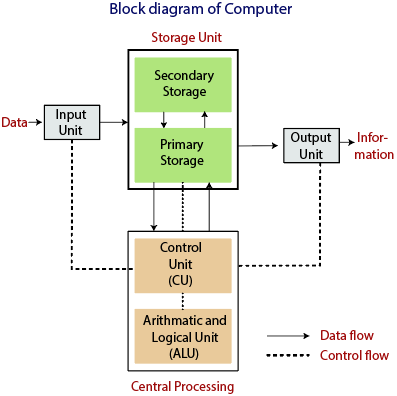
**Computer Programming**

**Computer Block Diagram System**: Mainly computer system consists of three parts, that are central processing unit (CPU), [Input Devices](https://www.tutorialandexample.com/input-devices-of-computer/), and [Output Devices](https://www.tutorialandexample.com/output-devices-of-computer/). The Central Processing Unit (CPU) is divided into two parts again: arithmetic logic unit (ALU) and the control unit (CU). The set of instruction is in the form of raw data.

A large amount of data is stored in the computer memory with the help of primary and secondary storage devices. The CPU is like the heart/brain of the computer. The user does not get the desired output, without the necessary option taken by the CPU.  The Central processing unit (CPU) is responsible for the processing of all the instructions which are given by the user to the computer system.

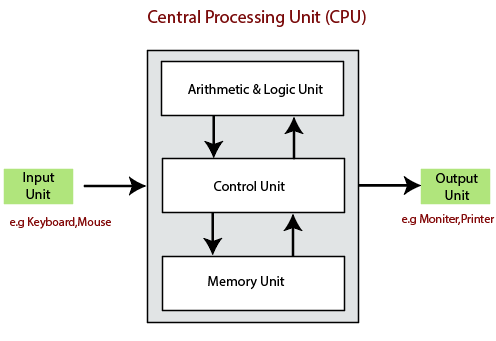


The data is entered through input devices such as the keyboard, mouse, etc. This set of instruction is processed by the CPU after getting the input by the user, and then the computer system produces the output. The computer can show the output with the help of output devices to the user, such as  monitor, printer, etc.

* CPU (Central Processing Unit)
* Storage Unit
* ALU(Arithmetic Logic Unit)
* Control Unit

### Central Processing Unit (CPU)

The computer system is nothing without the [Central processing Unit](https://en.wikipedia.org/wiki/Central_processing_unit) so, it is also known as the brain or heat of computer. The CPU is an electronic hardware device which can perform different types of operations such as arithmetic and logical operation.



The CPU contains two parts: the arithmetic logic unit and control unit. We have discussed briefly the arithmetic unit, logical unit, and control unit which are given below:

### **Control Unit**

The control unit (CU) controls all the activities or operations which are performed inside the computer system. It receives instructions or information directly from the main memory of the computer.

When the control unit receives an instruction set or information, it converts the instruction set to control signals then; these signals are sent to the central processor for further processing. The control unit understands which operation to execute, accurately, and in which order.

### **Arithmetic and Logical Unit**

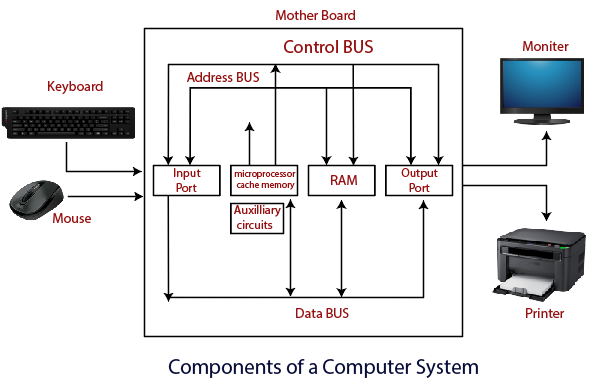
The arithmetic and logical unit is the combinational digital electronic circuit that can perform arithmetic operations on integer binary numbers.It presents the arithmetic and logical operation. The outputs of ALU will change asynchronously in response to the input. The basic arithmetic and bitwise logic functions are supported by ALU.

### **Storage Unit**

The information or set of guidelines are stored in the storage unit of the computer system. The storage unit provides the space to store the data or instruction of processed data. The information or data is saved or hold in computer memory or storage device. The data storage is the core function and fundamental of the computer components.

### **Components of Computer System**

The hardware and software exist on the computer**.** The information which is stored through the device is known as computer software. The hardware components of the computer system are related to electronic and mechanical parts, and the software component is related to data and computer programs. Many elements are connected to the main circuit board of the computer system called a “motherboard.”



* Processor.
* Main Memory.
* Secondary Memory.
* Input Devices.
* Output Devices.

These are mainly five components of the computer system. The computer hardware, computer software, and liveware exist in the element of the computer system.

#### **Processor**

The processor is an electric circuitry within the computer system. The Central processing unit is the central processor or main processor of the computer system. The processor carries out the instructions of the computer program with the help of basic arithmetic and logic, input/output operations.

#### **Main Memory**

The Random Access Memory is the main memory of the computer system, which is known as RAM.  The main memory can store the operating system software, application software, and other information.  The Ram is one of the fastest memory, and it allows the data to be readable and writeable.

1MB =1024 KB= 1024\*1024Byte=1024\*1024\*8 bit

1 Byte = 8 Bit

1 KB = 1024 Byte

1MB = 1024 KB

1GB = 1024 MB

1TB = 1024 GB

1PT = 1024 TB

#### **Secondary memory**

 We can store the data and programs on a long-term basis in the secondary memory. The hard disks and the optical disks are the common secondary devices. It is slow and cheap memory as compare to primary memory. This memory is not connected to the processor directly.

It has a large capacity to store the data. The hard disk has a capacity of 500 gigabytes. The data and programs on the hard disk are organized into files, and the file is the collection of data on the disk. The secondary storage is direct access by the CPU; that’s why it is different from the primary storage.

The hard disk is about 100 times the capacity of the main memory. The main difference between primary and secondary storage is speed and capacity. There are several large blocks of data which are copied from the hard disk into the main memory.

#### **Input Devices**

The user provides the set of instruction or information to the computer system with the help of input devices such as the keyboard, mouse, scanner, etc. The data representation to the computer system is in the form of binary language after that the processor processes the converted data. The input unit implements the data which is instructed by the user to the system.

We can enter the data from the outside world into the primary storage as the input through input devices. The input devices are the medium of communication between the outside world and the computer system.

There are some important features of input devices which are given below:

1. The input devices receive or accept the data or instruction from the user, who exist in the outside world.
2. These devices convert the data or instruction into the machine-readable form for further processing.
3. The input device performs like the connection between the outside world and our computer system.
4. The keyboard and mouse are common examples of input devices.
5. When the whole procedure is finished, we get the desired output from the output devices such as monitor, printer, etc.

#### **Output Devices**

The output devices produce or generate the desired result according to our input, such as a printer, monitor, etc. These devices convert the data into a human-readable form from [binary code](https://en.wikipedia.org/wiki/Binary_code).

The computer system is linked or connected to the outside world with the help of output devices. The primary examples of output devices are a printer, projector, etc.

These devices have various features which are given below:

1. These devices receive or accept the data in the binary form.
2. The output devices convert the binary code into the human-readable form.
3. These devices produce the converted result and show to the user.

**Difference between Primary and Secondary Memory**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Primary memory** | **Secondary memory** |
| Nature | The primary memory is categorized as volatile & nonvolatile memories. | The secondary memory is always a non-volatile memory. |
| Alias | These memories are also called internal memory. | Secondary memory is known as a Backup memory or Additional memory or Auxiliary memory. |
| Access | Data is directly accessed by the processing unit. | Data cannot be accessed directly by the processor. It is first copied from secondary memory to primary memory. Only then CPU can access it. |
| Formation | It's a volatile memory meaning data cannot be retained in case of power failure. | It's a non-volatile memory so that that data can be retained even after power failure. |
| Storage | It holds data or information that is currently being used by the processing unit. Capacity is usually in 16 to 32 GB | It stores a substantial amount of data and information. Capacity is generally from 200GB to terabytes. |
| Accesses | Primary memory can be accessed by the data bus. | Secondary memory is accessed by I/O channels. |
| Expense | Primary memory is costlier than secondary memory. | Secondary memory is cheaper than primary memory. |

**Operating System**

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux Operating System, Windows Operating System,etc.

**Definition**

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Following are some of important functions of an operating System.

### **1. Memory Management**

It is the management of the main or primary memory. Whatever program is executed, it has to be present in the main memory. Therefore, there can be more than one program present at a time. Hence, it is required to manage the memory.

The operating system:

* Allocates and deallocates the memory.
* Keeps a record of which part of primary memory is used by whom and how much.
* Distributes the memory while multiprocessing.

### **2. Processor Management**

When more than one process runs on the system the OS decides how and when a process will use the CPU. Hence, the name is also **CPU Scheduling**. The OS:

* Allocates and deallocates processor to the processes.
* Keeps record of CPU status.

Certain algorithms used for CPU scheduling are as follows:

* First Come First Serve (FCFS)
* Shortest Job First (SJF)
* Round-Robin Scheduling
* Priority-based scheduling etc.

#### Purpose of CPU scheduling

The purpose of CPU scheduling is as follows:

* Proper utilization of CPU. Since the proper utilization of CPU is necessary. Therefore, the OS makes sure that the CPU should be as busy as possible.
* Since every device should get a chance to use the processor. Hence, the OS makes sure that the devices get fair processor time.
* Increasing the efficiency of the system.

### **3. Device Management**

The processes may require devices for their use. This management is done by the OS. The OS:

* Allocates and deallocates devices to different processes.
* keep records of the devices.
* Decides which process can use which device for how much time.

### **4. File Management**

The files on a system are stored in different directories. The OS:

* keeps records of the status and locations of files.
* Allocates and deallocates resources.

### **5. Security**

The OS keeps the system and programs safe and secure through authentication. A user id and password decide the authenticity of the user.

### **7. Accounting**

As the operating system keeps track of all the functions of a computer system. Hence, it makes a record of all the activities taking place on the system. It has an account of all the information about the memory, resources, errors, etc. Therefore, this information can be used as and when required.

### **8. Other Functions**

Some other functions of the OS can be:

* Error detection.
* keeping a record of system performance.
* Communication between different software etc.

**Compiler**

A compiler is a computer program which helps you transform source code written in a high-level language into low-level machine language.

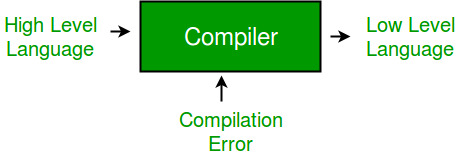
It translates the code written in one programming language to some other language without changing the meaning of the code.

The compiler also makes the end code efficient which is optimized for execution time and memory space.

The compiling process includes basic translation mechanisms and error detection.

A compiler is a translator that converts the high-level language into the machine language.

* High-level language is written by a developer and machine language can be understood by the processor.
* Compiler is used to show errors to the programmer.
* The main purpose of compiler is to change the code written in one language without changing the meaning of the program.



**NUMBER SYSTEM**

When we type some letters or words, the computer translates them in numbers as computers can understand only numbers. A computer can understand the positional number system where there are only a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

The value of each digit in a number can be determined using −

* The digit
* The position of the digit in the number
* The base of the number system (where the base is defined as the total number of digits available in the number system)

**Decimal Number System**

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

(1 x 1000)+ (2 x 100)+ (3 x 10)+ (4 x l)

(1 x 103)+ (2 x 102)+ (3 x 101)+ (4 x l00)

1000 + 200 + 30 + 4

1234

|  |  |
| --- | --- |
| **S.No.** | **Number System and Description** |
| 1 | **Binary Number System**  Base 2. Digits used : 0, 1 |
| 2 | **Octal Number System**  Base 8. Digits used : 0 to 7 |
| 3 | **Hexa Decimal Number System**  Base 16. Digits used: 0 to 9, Letters used : A- F |

## Binary Number System

Characteristics of the binary number system are as follows −

* Uses two digits, 0 and 1
* Also called as base 2 number system
* Each position in a binary number represents a **0** power of the base (2). Example 20
* Last position in a binary number represents a **x** power of the base (2). Example 2x where **x** represents the last position - 1.

### Example

Binary Number: (10101)2

Calculating Decimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Binary Number** | **Decimal Number** |
| Step 1 | (10101)2 | ((1 x 24) + (0 x 23) + (1 x 22) + (0 x 21) + (1 x 20))10 |
| Step 2 | 101012 | (16 + 0 + 4 + 0 + 1)10 |
| Step 3 | 101012 | (21)10 |

**Note** − 101012 is normally written as 10101.

## Octal Number System

Characteristics of the octal number system are as follows −

* Uses eight digits, 0,1,2,3,4,5,6,7
* Also called as base 8 number system
* Each position in an octal number represents a **0** power of the base (8). Example 80
* Last position in an octal number represents a **x** power of the base (8). Example 8x where **x** represents the last position - 1

### Example

Octal Number: 125708

Calculating Decimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Octal Number** | **Decimal Number** |
| Step 1 | 125708 | ((1 x 84) + (2 x 83) + (5 x 82) + (7 x 81) + (0 x 80))10 |
| Step 2 | 125708 | (4096 + 1024 + 320 + 56 + 0)10 |
| Step 3 | 125708 | 549610 |

**Note** − 125708 is normally written as 12570.

## Hexadecimal Number System

Characteristics of hexadecimal number system are as follows −

* Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
* Letters represent the numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15
* Also called as base 16 number system
* Each position in a hexadecimal number represents a **0** power of the base (16). Example, 160
* Last position in a hexadecimal number represents a **x** power of the base (16). Example 16x where **x** represents the last position - 1

### Example

Hexadecimal Number: 19FDE16

Calculating Decimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Hexadecimal Number** | **Decimal Number** |
| Step 1 | 19FDE16 | ((1 x 164) + (9 x 163) + (F x 162) + (D x 161) + (E x 160))10 |
| Step 2 | 19FDE16 | ((1 x 164) + (9 x 163) + (15 x 162) + (13 x 161) + (14 x 160))10 |
| Step 3 | 19FDE16 | (65536+ 36864 + 3840 + 208 + 14)10 |
| Step 4 | 19FDE16 | 10646210 |

# **Number Conversion**

There are many methods or techniques which can be used to convert numbers from one base to another. In this chapter, we'll demonstrate the following −

* Decimal to Other Base System
* Other Base System to Decimal
* Other Base System to Non-Decimal
* Shortcut method - Binary to Octal
* Shortcut method - Octal to Binary
* Shortcut method - Binary to Hexadecimal
* Shortcut method - Hexadecimal to Binary

## Decimal to Other Base System

**Step 1** − Divide the decimal number to be converted by the value of the new base.

**Step 2** − Get the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number.

**Step 3** − Divide the quotient of the previous divide by the new base.

**Step 4** − Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

### Example

Decimal Number: 2910

Calculating Binary Equivalent −

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Operation** | **Result** | **Remainder** |
| Step 1 | 29 / 2 | 14 | 1 |
| Step 2 | 14 / 2 | 7 | 0 |
| Step 3 | 7 / 2 | 3 | 1 |
| Step 4 | 3 / 2 | 1 | 1 |
| Step 5 | 1 / 2 | 0 | 1 |

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).

Decimal Number : 2910 = Binary Number : 111012.

(12)3 + (12)3 + (12)3 = (120)3

## Other Base System to Decimal System

**Step 1** − Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

**Step 2** − Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

**Step 3** − Sum the products calculated in Step 2. The total is the equivalent value in decimal.

### Example

Binary Number: 111012

Calculating Decimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Binary Number** | **Decimal Number** |
| Step 1 | 111012 | ((1 x 24) + (1 x 23) + (1 x 22) + (0 x 21) + (1 x 20))10 |
| Step 2 | 111012 | (16 + 8 + 4 + 0 + 1)10 |
| Step 3 | 111012 | 2910 |

Binary Number : 111012 = Decimal Number : 2910

## Other Base System to Non-Decimal System

**Step 1** − Convert the original number to a decimal number (base 10).

**Step 2** − Convert the decimal number so obtained to the new base number.

### Example

Octal Number : 258

Calculating Binary Equivalent −

### Step 1 - Convert to Decimal

|  |  |  |
| --- | --- | --- |
| **Step** | **Octal Number** | **Decimal Number** |
| Step 1 | 258 | ((2 x 81) + (5 x 80))10 |
| Step 2 | 258 | (16 + 5)10 |
| Step 3 | 258 | 2110 |

Octal Number : 258 = Decimal Number : 2110

### Step 2 - Convert Decimal to Binary

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Operation** | **Result** | **Remainder** |
| Step 1 | 21 / 2 | 10 | 1 |
| Step 2 | 10 / 2 | 5 | 0 |
| Step 3 | 5 / 2 | 2 | 1 |
| Step 4 | 2 / 2 | 1 | 0 |
| Step 5 | 1 / 2 | 0 | 1 |

Decimal Number : 2110 = Binary Number : 101012

Octal Number : 258 = Binary Number : 101012

## Shortcut Method ─ Binary to Octal

**Step 1** − Divide the binary digits into groups of three (starting from the right).

**Step 2** − Convert each group of three binary digits to one octal digit.

### Example

Binary Number : 101012

Calculating Octal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Binary Number** | **Octal Number** |
| Step 1 | 101012 | 010 101 |
| Step 2 | 101012 | 28 58 |
| Step 3 | 101012 | 258 |

Binary Number : 101012 = Octal Number : 258

## Shortcut Method ─ Octal to Binary

**Step 1** − Convert each octal digit to a 3-digit binary number (the octal digits may be treated as decimal for this conversion).

**Step 2** − Combine all the resulting binary groups (of 3 digits each) into a single binary number.

### Example

Octal Number : 258

Calculating Binary Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Octal Number** | **Binary Number** |
| Step 1 | 258 | 210 510 |
| Step 2 | 258 | 0102 1012 |
| Step 3 | 258 | 0101012 |

Octal Number : 258 = Binary Number : 101012

## Shortcut Method ─ Binary to Hexadecimal

**Step 1** − Divide the binary digits into groups of four (starting from the right).

**Step 2** − Convert each group of four binary digits to one hexadecimal symbol.

### Example

Binary Number : 101012

Calculating hexadecimal Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Binary Number** | **Hexadecimal Number** |
| Step 1 | 101012 | 0001 0101 |
| Step 2 | 101012 | 110 510 |
| Step 3 | 101012 | 1516 |

Binary Number : 101012 = Hexadecimal Number : 1516

## Shortcut Method - Hexadecimal to Binary

**Step 1** − Convert each hexadecimal digit to a 4-digit binary number (the hexadecimal digits may be treated as decimal for this conversion).

**Step 2** − Combine all the resulting binary groups (of 4 digits each) into a single binary number.

### Example

Hexadecimal Number : 1516

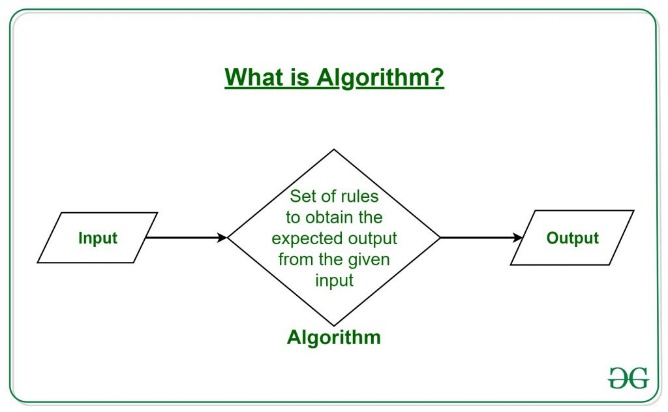
Calculating Binary Equivalent −

|  |  |  |
| --- | --- | --- |
| **Step** | **Hexadecimal Number** | **Binary Number** |
| Step 1 | 1516 | 110 510 |
| Step 2 | 1516 | 00012 01012 |
| Step 3 | 1516 | 000101012 |

Hexadecimal Number : 1516 = Binary Number : 101012

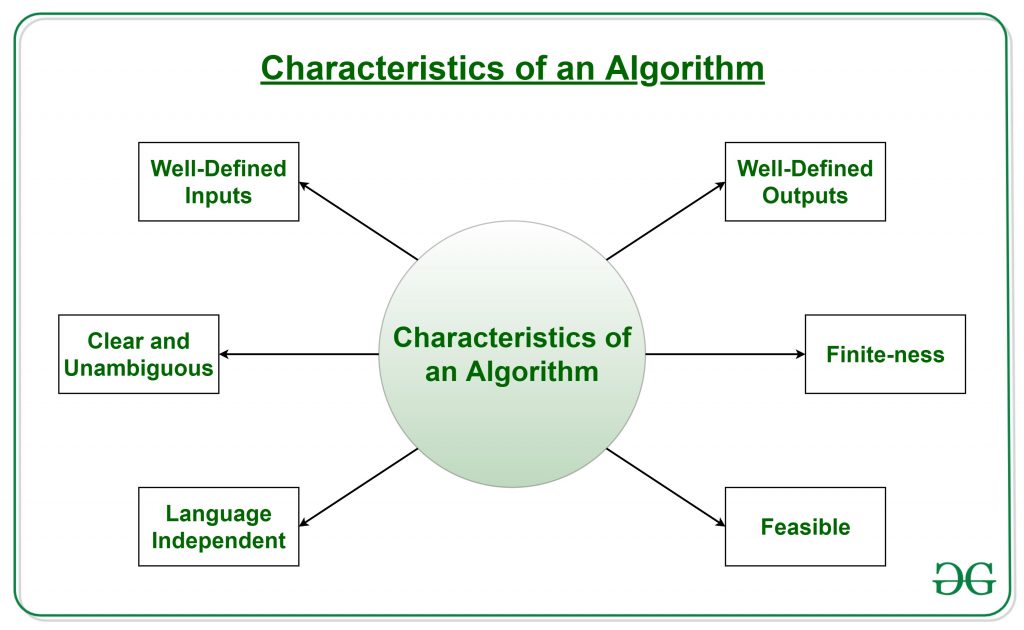
**ALGORITHM**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.



## Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should 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 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 Write an Algorithm?

There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.

As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.

We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution.

### **Example**

Let's try to learn algorithm-writing by using an example.

**Problem** − Design an algorithm to add two numbers and display the result.

**Step 1** − START

**Step 2** − declare three integers **a**, **b** & **c**

**Step 3** − define values of **a** & **b**

**Step 4** − add values of **a** & **b**

**Step 5** − store output of step 4 to **c**

**Step 6** − print **c**

**Step 7** − STOP

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as −

**Step 1** − START

**Step 2** − get values of **a** & **b**

**Step 3** − c ← a + b

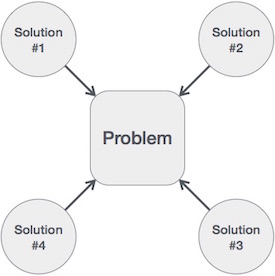
**Step 4** − display c

**Step 5** − STOP

In design and analysis of algorithms, usually the second method is used to describe an algorithm. It makes it easy for the analyst to analyze the algorithm ignoring all unwanted definitions. He can observe what operations are being used and how the process is flowing.

Writing **step numbers**, is optional.

We design an algorithm to get a solution of a given problem. A problem can be solved in more than one ways.



Hence, many solution algorithms can be derived for a given problem. The next step is to analyze those proposed solution algorithms and implement the best suitable solution.

## Algorithm Analysis

Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation. They are the following −

* ***A Priori* Analysis** − This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed etc.
* ***A Posterior* Analysis** − This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required are consider.

Algorithm analysis deals with the execution or running time of various operations involved. The running time of an operation can be defined as the number of computer instructions executed per operation.

## Flowchart

**Flowchart** is a diagrammatic representation of sequence of logical steps of a program.

Flowcharts use simple geometric shapes to depict processes and arrows to show relationships and process/data flow.

## Flowchart Symbols

Here is a chart for some of the common symbols used in drawing flowcharts.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Symbol Name** | **Purpose** |
| Start Stop | Start/Stop | Used at the beginning and end of the algorithm to show start and end of the program. |
| Process | Process | Indicates processes like mathematical operations. |
| Input/ Output | Input/ Output | Used for denoting program inputs and outputs. |
| Decision | Decision | Stands for decision statements in a program, where answer is usually Yes or No. |
| Arrow | Arrow | Shows relationships between different shapes. |
| On-page Connector | On-page Connector | Connects two or more parts of a flowchart, which are on the same page. |
| Off-page Connector | Off-page Connector | Connects two parts of a flowchart which are spread over different pages. |

### **Guidelines for Developing Flowcharts**

These are some points to keep in mind while developing a flowchart −

* Flowchart can have only one start and one stop symbol
* On-page connectors are referenced using numbers
* Off-page connectors are referenced using alphabets
* General flow of processes is top to bottom or left to right
* Arrows should not cross each other

**Example 1: Write an algorithm to display “GGV Bilaspur”**

Step 1: START

Step 2: display “GGV Bilaspur”

Step 3: STOP

Display GGV Bilaspur

Hello.C

#include<stdio.h>

int main()

{

printf(“GGV Bilaspur”);

return 0;

}

**Example 2: Write an algorithm to calculate Area of Circle.**

Step 1: START

Step 2: Input value of radius r

Step 3: A=3.14 \* r\*r

Step 4: Display value of A

Step 5: STOP

Input Radius r

A=3.14\*r\*r

Display Area A

#include<stdio.h>

int main()

{

float r,A;

printf(“Enter the value of Radius:”);

scanf(“%f”,&r);

A=3.14\*r\*r;

printf(“Area of Circle is : %f ”, A);

return 0;

}

**Example 3: Write an Algorithm to find Average of five number.**

Step 1: START

Step 2: Input five numbers A,B,C,D,E

Step 3: Avg =(A+B+C+D+E)/5

Step 4: Display Values of Avg.

Step 5: STOP

Input A,B,C,D,E

Avg=(A+B+C+D+E)/5

Display Avg

**Example 4: Write an Algorithm to convert temperature F to C**

Step 1: START

Step 2: Input value of F

Step 3: C=(F-32)\*5/9

Step 4: Display the value of C

Step 5: STOP

Input F

C=(F-32)\*5/9

Display C

/\* Program to convert F to C \*/

#include<stdio.h>

int main()

{

float F,C;

printf("enter temprature in F:");

scanf("%f",&F);

C=(F-32)\*5/9;

printf("The value of C is %f",C);

return 0;

}

**Example 5: Write an algorithm to check given number is Even or Odd.**

Step 1: START

Step 2: Input number N

Step 3: if N%2 == 0 goto Step 4 otherwise goto Step 5

Step 4: Display given number N is Even goto Step 6

Step 5: Display given Number N is Odd

Step 6: STOP

Input N

Is N%2===0

True

False

Odd

Even

## /\*Program to check given number is Even or Odd.\*/

## #include<stdio.h>

## int main()

## {

## int N;

## printf("enter a Number:");

## scanf("%d",&N);

## if(N%2 == 0)

## {

## printf("Number %d is Even",N);

## }

## else

## {

## printf("Number %d is Odd",N);

## }

## 

## return 0;

## }

**Example 6: Write an algorithm to check given alphabet is ‘A’.**

Step 1: START

Step 2: Input character T

Step 3: if T== ‘A’ goto Step 4 otherwise goto Step 5

Step 4: Display given alphabet is ‘A’ goto Step 6

Step 5: Display given alphabet is not ‘A’

Step 6: STOP

Input character T

Is T==’A’

True

False

Entered Alphabet is not A

Entered Alphabet is A

/\*Program to check given alphabet is 'A'\*/

#include<stdio.h>

int main()

{

char T;

printf("Enter an Alphabet:");

scanf("%c",&T);

if(T == 'A')

{

printf("Entered Alphabet is 'A'");

}

else

{

printf("Entered Alphabet is NOT 'A'");

}

return 0;

}

**Example 7: Write an algorithm to swap two number using third variable.**

Step 1: START

Step 2: Input two number A and B

Step 3: T=B

Step 4: B=A

Step 5: A=T

Step 6: Display value of A and B

Step 7: STOP

Input A and B

T=B B=A A=T

Display value of A and B

/\*Program to swap two number using third variable\*/

#include<stdio.h>

int main()

{

int A,B,T;

printf("Enter two Numbers:");

scanf("%d%d",&A,&B);

T=B;

B=A;

A=T;

printf("After Swaping Value of A=%d and B=%d is",A,B);

return 0;

}

**Example 8: Write an algorithm to swap two number without using third variable.**

Step 1: START

Step 2: Input two number A and B

Step 3:A=A+B

Step 4: B=A-B

Step 5: A=A-B

Step 6: Display the value of A and B

Step 7: STOP

Input A and B

A=A+B B=A-B A=A-B

Display value of A and B

/\*Program to swap two number without using third variable\*/

#include<stdio.h>

int main()

{

int A,B;

printf("Enter two Numbers:");

scanf("%d%d",&A,&B);

A=A+B;

B=A-B;

A=A-B;

printf("After Swaping Value of A=%d and B=%d",A,B);

return 0;

}

**Example 10: Write an algorithm to check given number is divisible by 3 and 8.**

Step 1: START

Step 2: Input number N

Step 3: is N%3 == 0 and N%8== 0 then goto Step 4 otherwise goto Step 5

Step 4: Display given number N is divisible by 3 and 8 got step 6

Step 5: Display given number N is not divisible by 3 and 8

Step 6: STOP

Input Number N

Is N%3==0 && N%8==0

True

False

given number N is not divisible by 3 and 8

given number N is divisible by 3 and 8

/\* Program to check given number N is divisible by 3 and 8\*/

#include<stdio.h>

int main()

{

int N;

printf("Enter a Number:");

scanf("%d",&N);

if(N%3 == 0 && N%8 ==0)

{

printf("Given number %d is divisible by 3 and 8",N);

}

else

{

printf("Given number %d is not divisible by 3 or 8",N);

}

return 0;

}

**Example 10: Write an algorithm to display “Hello GGV” 10 times.**

Step 1: START

Step 2 I=1

Step 3: I<=10 goto Step 4 otherwise Step 7

Step 4: Print “Hello GGV”

Step 5:I=I+1

Step 6: goto Step 3

Step 7: STOP

I=1

I<=10

False

True

Print Hello GGV

I=I+1

## /\* Program to display Hello GGV 10 times\*/

## #include<stdio.h>

## int main()

## {

## int i=1;

## while(i<=10)

## {

## printf("Hello GGV\n");

## i=i+1;

## }

## return 0;

## }

**Example 11: Write an algorithm to display 1 to 10.**

Step 1: START

Step 2 I=1

Step 3: I<=10 goto Step 4 otherwise goto Step 7

Step 4: print value of I

Step 5: I=I+1

Step 6: goto step 3

Step 7: STOP

I=1

I<=10

False

True

Print value of I

I=I+1

**/\* Write a Program to display 1 to 10. \*/**

**#include<stdio.h>**

**int main()**

**{**

**int I=1;**

**while(I<=10)**

**{**

**printf("%d\t",I);**

**I=I+1;**

**}**

**return 0;**

**}**

**Example 11: Write an algorithm to display table of N.**

Step 1: START

Step 2: Input number N

Step 3: I=1

Step 4: I<=10 goto Step 5 otherwise goto Step 8

Step 5: print N\*I

Step 6: I=I+1

Step 7: goto Step 4

Step 8: STOP

Input Number N

I=1

I<=10

False

True

Print value of N\*I

I=I+1

/\* Program to display table of N\*/

#include<stdio.h>

int main()

{

int i=1,N;

printf("enter the Value of N");

scanf("%d",&N);

while(i<=10)

{

printf("%d \* %d = %d\n",N,i,N\*i);

i=i+1;

}

return 0;

}

**Example 12: Write an algorithm to calculate factorial of N.**

Step 1: START

Step 2: Input number N

Step 3: I=1 Fact=1

Step 4: I<=N goto Step 5 otherwise goto Step 8

Step 5: Fact = Fact \* I

Step 6: I=I+1

Step 7: goto Step 4

Step 8: Display value of fact

Step 9: STOP

Input Number N

I=1, Fact=1

I<=N

False

Display Value of Fact

True

Fact=Fact\*1

I=I+1

/\* Program to caculate factorial of N\*/

#include<stdio.h>

int main()

{

int i=1,N,fact=1;

printf("enter the Value of N");

scanf("%d",&N);

while(i<=N)

{

fact=fact\*i;

i=i+1;

}

printf("factorial of %d is %d",N,fact);

return 0;

}

**Example 13: Write an algorithm to calculate sum of following series**

**1+2+3+4+5…+N**

Step 1: START

Step 2: Input number N

Step 3: I=1 sum=0

Step 4: I<=N goto Step 5 otherwise goto Step 8

Step 5: sum = sum + I

Step 6: I=I+1

Step 7: goto Step 4

Step 8: Display value of fact

Step 9: STOP

/\* Program to caculate sum of following series1+2+3+4...N\*/

#include<stdio.h>

int main()

{

int i=1,N,sum=0;

printf("enter the Value of N");

scanf("%d",&N);

while(i<=N)

{

sum=sum+i;

i=i+1;

}

printf("Sum of %d natural Number is %d ",N,sum);

return 0;

}

## Example 14 : Write a Program to reverse the given Number N.

## /\*Write a Program to reverse the given Number N \*/

## #include<stdio.h>

## int main()

## {

## int N,rev=0,rem,T;

## printf("Enter the Value of N ");

## scanf("%d",&N);

## T=N;

## while(N!=0)

## {

## rem = N%10;

## rev = rev\*10+rem;

## N=N/10;

## }

## printf("Reverse of %d is %d",T,rev);

## return 0;

## }

## Example 15: Write a Program to check given number N is palindrome or NOT.

## /\*Write a Program to check given number N is palindrome or NOT \*/

## #include<stdio.h>

## int main()

## {

## int N,rev=0,rem,T;

## printf("Enter the Value of N ");

## scanf("%d",&N);

## T=N;

## while(N!=0)

## {

## rem = N%10;

## rev = rev\*10+rem;

## N=N/10;

## }

## if(T==rev)

## {

## printf("%d is palindrome number",T);

## }

## else

## {

## printf("%d is NOT a palindrome number",T);

## }

## return 0;

## }

## Example 16:Write a Program to calculate ab

## /\* Program to caculate a power b\*/

## #include<stdio.h>

## int main()

## {

## int a,b,i=1,power=1;

## printf("enter the Value of a and b");

## scanf("%d%d",&a,&b);

## while(i<=b)

## {

## power=power\*a;

## i=i+1;

## }

## printf("%d power %d is %d ",a,b,power);

## return 0;

## }

## Example 17: Write a Program to check given Number N is Prime or NOT.

## /\* Program to check given Number N is Prime or NOT\*/

## #include<stdio.h>

## int main()

## {

## int N,count=0,i=1;

## printf("enter the Value of N");

## scanf("%d",&N);

## while(i<=N)

## {

## if(N%i == 0)

## {

## count=count+1;

## }

## i=i+1;

## }

## if(count == 2)

## {

## printf("%d is Prime Number",N);

## }

## else

## {

## printf("%d is NOT Prime Number",N);

## }

## return 0;

## }

## Example 18 : Write a Program to find sum of following series: 12+22 …N2

## /\* Program to caculate sum of following series1\*1+2\*2+3\*3+4\*4...N\*N\*/

## #include<stdio.h>

## int main()

## {

## int i=1,N,sum=0;

## printf("enter the Value of N");

## scanf("%d",&N);

## while(i<=N)

## {

## sum=sum+i\*i;

## i=i+1;

## }

## printf("Sum of %d natural Number is %d ",N,sum);

## return 0;

## }

## Example 19: Write an Algorithm to add all digits of given number N

Step 1 :START

Step 2: Input Number N

Step 3:sum=0

Step 4:if N != 0 then goto Step 5 otherwise got Step 9

Step 5: rem = N%10

Step 6: sum = sum+rem

Step 7: N= N/10

Step 8: goto Step 4

Step 9: Display value of sum

Step 10: STOP

## Example 20: Write an Algorithm to find sum of following series:

## 11 + 22 + 33 .. NN

## Step 1: START

## Step 2: Input Number N

## Step 3: sum=0,I=1

## Step 4: if I <= N then got Step 5 otherwise goto Step 13

## Step 5: J=1, pow=1

## Step 6: if J <= I then goto Step 7 otherwise got Step 10

## Step 7: pow= pow \*I

## Step 8: J=J+1

## Step 9: goto Step 6

## Step 10: sum=sum+pow

## Step 11: I=I+1

## Step 12:goto Step 4

## Step 13: Display the Value of sum

## Step 14:STOP

## Example 21: Write an Algorithm to display following pattern

## \*

## \*\*

## \*\*\*

## Step 1:START

## Step 2: Input Number of Rows N

## Step 3: I=1

## Step 4: if I<=N goto Step 5 otherwise goto Step 13

## Step 5: J=1

## Step 6 :J<=I got Step 7 otherwise got Step 10

## Step 7: print \*

## Step 8: J=J+1

## Step 9: goto Step 6

## Step 10:come to the next line (\n)

## Step 11: I=I+1

## Step 12: goto Step 4

## Step 13: STOP

## Example 22: Write an Algorithm to Check Given Number Is Armstrong number or not.

## 153 13 + 53 + 33 =153

## Step 1: START

## Step 2: Input Number N

## Step 3: sum=0, T=N

## Step 4: if N != 0 then goto Step 5 otherwise goto Step 9

## Step 5 : rem=N%10

## Step 6 : sum=sum+rem\*rem\*rem

## Step 7: N=N/10

## Step 8: goto Step 4

## Step 9 : if T==sum then goto Step 10 otherwise goto Step 11

## Step 10: Display Given number N is Armstrong Number goto Step 12

## Step 11: Display Given number N is NOT Armstrong Number

## Step 12 : STOP

## Example 23: Write an Algorithm to display following pattern

## \*\*\*

## \*\*

## \*

## Step 1: Start

## Step 2: Input Number of Rows N

## Step 3: I=N

## Step 4 : if I>=1 then got Step 5 otherwise goto Step 13

## Step 5: J=1

## Step 6: If J<=I then goto Step 7 otherwise goto Step 10

## Step 7: print \*

## Step 8:J=J+1

## Step 9: goto Step 6

## Step 10 :I=I-1

## Step 11: New line command

## Step 12: goto Step 4

## Step 13: STOP

## Example 24: Write an algorithm to find the sum of odd number:

## 1+3+5+7…

## Step 1: START

## Step 2: Input Number N

## Step 3: I=1, sum=0

## Step 4: if I<=N then goto Step 5 otherwise goto Step 8

## Step 5: sum=sum+I

## Step 6: I=I+2

## Step 7: goto Step 4

## Step 8: Display sum

## Step 9: STOP

## Example 25: Write an algorithm to find the sum of even number:

## 2+4+6+8…

## Step 1: START

## Step 2: Input Number N

## Step 3: I=2, sum=0

## Step 4: if I<=N then goto Step 5 otherwise goto Step 8

## Step 5: sum=sum+I

## Step 6: I=I+2

## Step 7: goto Step 4

## Step 8: Display sum

## Step 9: STOP

## Example 26: Write an algorithm to display Fibonacci series

## 1 1 2 3 5 8 13 21 ….

## Example 27:

## /\* Write a program to Print A to Z using for loop. \*/

## #include<stdio.h>

## int main()

## {

## int i;

## for(i=65;i<91;i++)

## {

## printf("%c ",i);

## }

## return 0;

## }

## /\* Write a program to Print

## A

## AB

## ABC

## ABCD \*/

## #include<stdio.h>

## int main()

## {

## int n,i,j;

## printf("enter the number of rows: ");

## scanf("%d",&n);

## for(i=1;i<=n;i++)

## {

## for(j=65;j<65+i;j++)

## {

## printf("%c",j);

## }

## printf("\n");

## }

## return 0;

## }

## /\* Write a program to Print

## ABCD

## ABC

## AB

## A \*/

## #include<stdio.h>

## int main()

## {

## int n,i,j;

## printf("enter the number of rows: ");

## scanf("%d",&n);

## if(n<=26)

## {

## for(i=1;i<=n;i++)

## {

## for(j=65;j<=65+n-i;j++)

## {

## printf("%c",j);

## }

## printf("\n");

## }

## }

## else

## {

## printf("please enter the value of n between 1 to 26");

## }

## return 0;

## }

## /\* Write a program to check given alphabet is between P to T or NOT \*/

## #include<stdio.h>

## int main()

## {

## char T;

## printf("enter one Alphabet ");

## scanf("%c",&T);

## if(T>=80 && T<=84)

## {

## printf("Enter Alphabet is between P to T");

## }

## else

## {

## printf("Enter Alphabet is NOT between P to T");

## }

## return 0;

## }

## ARRAY

## #include<stdio.h>

## int main()

## {

## char aman[10];

## int i,count=0;

## printf("Enter 10 characters: ");

## for(i=0;i<10;i++)

## {

## scanf("%c",&aman[i]);

## }

## 

## for(i=9;i>=0;i--)

## {

## if(aman[i]=='a'||aman[i]=='A')

## {

## count++;

## }

## }

## printf("Total Number of letter a or A is %d ",count);

## return 0;

## }

## /\*count total number of EVEN \*/

## #include<stdio.h>

## int main()

## {

## int aman[10];

## int i,count=0;

## printf("Enter 10 values: ");

## for(i=0;i<10;i++)

## {

## scanf("%d",&aman[i]);

## }

## for(i=0;i<=9;i++)

## {

## if(aman[i]%2==0)

## {

## count++;

## }

## }

## printf("Total even number is %d",count);

## return 0;

## }

## String

## /\* Program to calculate the length of string \*/

## #include<stdio.h>

## int main()

## {

## char arr[] = "VARANASHI";

## int count=0,i=0;

## while(arr[i] != '\0')

## {

## count++;

## i++;

## }

## printf("Total length of %s is %d",arr,count);

## return 0;

## }

## /\* Program to replaces all odd place in string \*/

## #include<stdio.h>

## int main()

## {

## char arr[] = "VARANASHI";

## int count=0,i=0;

## while(arr[i] != '\0')

## {

## if((i+1)%2 ==1)

## {

## arr[i]='~';

## }

## i++;

## }

## printf("After replacing all odd places %s",arr);

## return 0;

## }

## /\* Program to copy one string to another variable \*/

## #include<stdio.h>

## #include<string.h>

## int main()

## {

## char arr[20],trr[20],crr[20];

## int count=0,i=0;

## printf("enter the first name: ");

## scanf("%s",arr);

## printf("enter the last name: ");

## scanf("%s",trr);

## while(arr[i]!='\0')

## {

## crr[i]=arr[i];

## i++;

## }

## while(trr[count]!='\0')

## {

## crr[i]=trr[count];

## count++;

## i++;

## }

## crr[i]='\0';

## printf("%s",crr);

## return 0;

## }

## /\* Program to Calculate the length of String using strlen() \*/

## #include<stdio.h>

## #include<string.h>

## int main()

## {

## char arr[50];

## int count;

## printf("Enter Your Name: ");

## scanf("%s",arr);

## count=strlen(arr);

## printf("length of %s is %d",arr,count);

## return 0;

## }

## /\* Program to Copy one string array to another String array using strcpy() \*/

## #include<stdio.h>

## #include<string.h>

## int main()

## {

## char arra[50],arrb[50];

## int i=0;

## printf("Enter Your Name: ");

## scanf("%s",arra);

## strcpy(arrb,arra);

## printf("String A= %s and String B = %s",arra,arrb);

## return 0;

## }

## /\* Program to reverse the string using strrev() \*/

## #include<stdio.h>

## #include<string.h>

## int main()

## {

## char arra[50];

## printf("Enter Your Name: ");

## scanf("%s",arra);

## strrev(arra);

## printf("reverse is %s" ,arra);

## return 0;

## }

## /\* Program to concatnate two string using strcat() \*/

## #include<stdio.h>

## #include<string.h>

## int main()

## {

## char arra[50],arrb[50];

## printf("Enter Your First Name: ");

## scanf("%s",arra);

## printf("Enter Your Last Name: ");

## scanf("%s",arrb);

## strcat(arra,arrb);

## printf("concatnate of given string is :%s " ,arra);

## return 0;

## }

## /\* Program to compare two string using strcmp() \*/

## #include<stdio.h>

## #include<string.h>

## int main()

## {

## char arra[50],arrb[50];

## int t;

## printf("Enter Your First Name: ");

## scanf("%s",arra);

## printf("Enter Your Last Name: ");

## scanf("%s",arrb);

## t=strcmp(arra,arrb);

## if(t==0)

## {

## printf("both string are same");

## }

## if(t>0)

## {

## printf("First String is bigger than Second String");

## }

## if(t<0)

## {

## printf("First String is smaller than Second String");

## }

## return 0;

## }

## /\* Program to add two 3\*3 matrix \*/

## #include<stdio.h>

## int main()

## {

## int a[3][3],b[3][3],c[3][3],i,j;

## printf("Enter the Value of First Matrix a[3][3]: ");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## scanf("%d",&a[i][j]);

## }

## }

## 

## printf("Enter the Value of Second Matrix b[3][3]: ");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## scanf("%d",&b[i][j]);

## }

## }

## 

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## c[i][j]=a[i][j]+b[i][j];

## }

## }

## 

## printf("sum of Matrix a and b is: \n");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## printf("%d ",c[i][j]);

## }

## printf("\n");

## }

## return 0;

## }

## /\* Program to calculate transpose of 3\*3 matrix \*/

## #include<stdio.h>

## int main()

## {

## int a[3][3],i,j;

## printf("Enter the Value of First Matrix a[3][3]: ");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## scanf("%d",&a[i][j]);

## }

## }

## printf("Transpose of a Matrix is : \n");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## printf("%d ",a[j][i]);

## }

## printf("\n");

## }

## return 0;

## }

## /\* Program to calculate sum of all element of 3\*3 matrix \*/

## #include<stdio.h>

## int main()

## {

## int a[3][3],i,j,sum=0;

## printf("Enter the Value of First Matrix a[3][3]: ");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## scanf("%d",&a[i][j]);

## }

## }

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## sum=sum+a[i][j];

## }

## }

## printf("Sum of All element of a Matrix is: %d",sum);

## return 0;

## }

## /\* Program to calculate sum of all element of 3\*3 matrix \*/

## #include<stdio.h>

## int main()

## {

## int a[3][3],i,j,sum=0;

## printf("Enter the Value of First Matrix a[3][3]: ");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## scanf("%d",&a[i][j]);

## }

## }

## for(i=0;i<3;i++)

## {

## sum=sum+a[i][i];

## 

## }

## printf("Sum of All digonal element of a Matrix is: %d",sum);

## return 0;

## }

## /\* Program to multiply two 3\*3 matrix \*/

## #include<stdio.h>

## int main()

## {

## int a[3][3],b[3][3],c[3][3],i,j,k,sum;

## printf("Enter the Value of First Matrix a[3][3]: ");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## scanf("%d",&a[i][j]);

## }

## }

## printf("Enter the Value of Second Matrix [3][3]: ");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## scanf("%d",&b[i][j]);

## }

## }

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## sum=0;

## for(k=0;k<3;k++)

## {

## sum=sum + a[i][k]\*b[k][j];

## }

## c[i][j]=sum;

## }

## }

## 

## printf("Multipliplication of Matrix is : \n");

## for(i=0;i<3;i++)

## {

## for(j=0;j<3;j++)

## {

## printf("%d ",c[i][j]);

## }

## printf("\n");

## }

## return 0;

## }