# SNS COLLEGE OF TECHNOLOGY

**Coimbatore-35.**
**An Autonomous Institution**

## COURSE NAME : 23ITT101 PROGRAMMING IN CAND DATASTRUCTURES

## I YEAR/ II SEMESTER

## UNIT-I INTRODUCTION TO C

## Algorithm, Pseudocode and Flowchart

Dr.B.Vinodhini
Associate Professor
Department of Computer Science and Engineering

# ***Building Blocks of Algorithms***

Algorithms can be constructed from basic building blocks namely,

1. Statements:
2. State
3. Control Flow
4. Functions

# *Building Blocks of Algorithms*

**1.Statements:**

Statement is a single action in a computer.

1. Input Data
2. Process Data
3. Output Data

**2.State:**

Transition from one process to another process under specified condition with in a time is called state

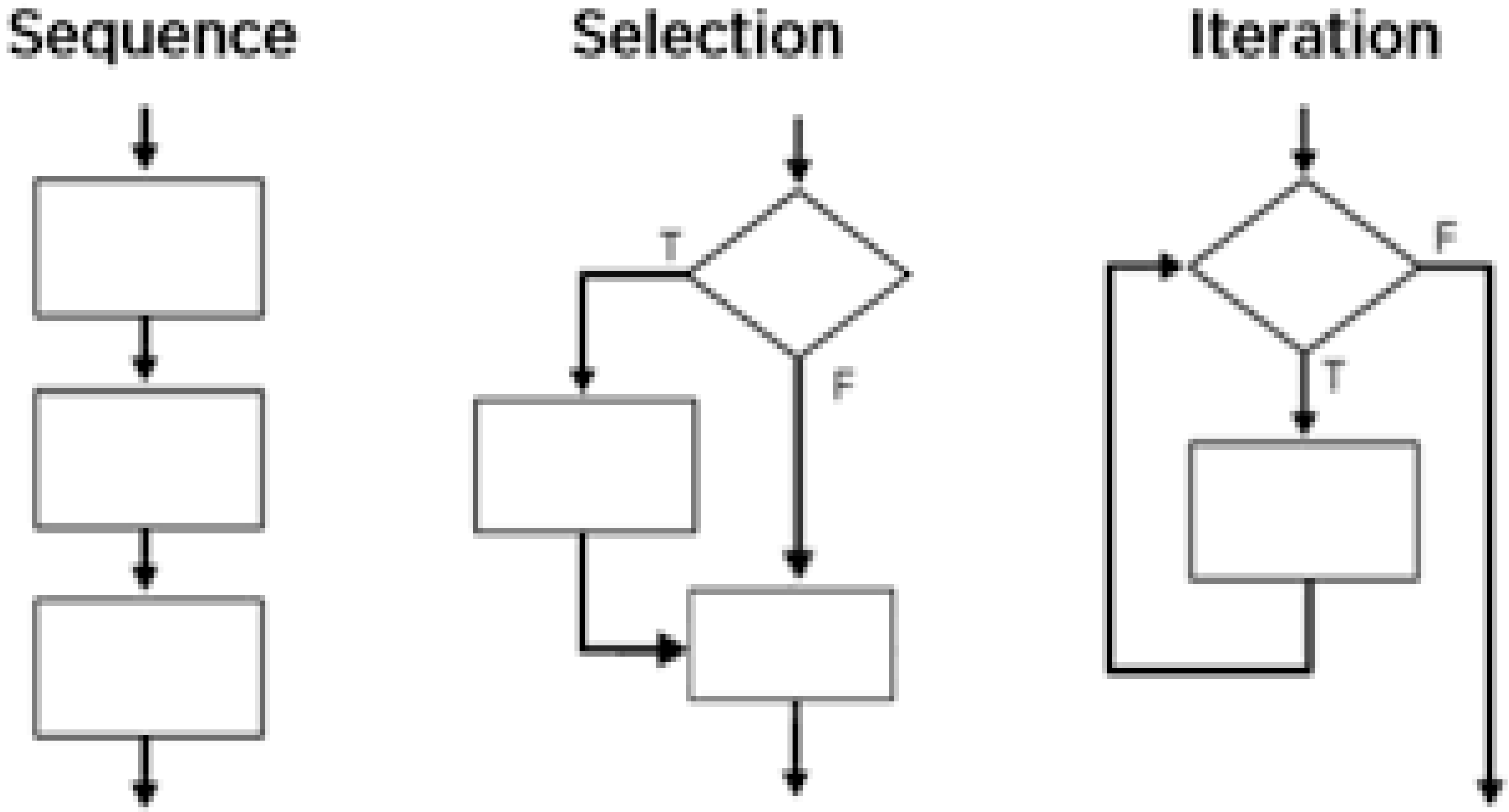
**3.Control flow:**

The process of executing the individual statements in a given order is called control flow

The control can be executed in three ways

1. Sequence
2. Selection
3. Iteration

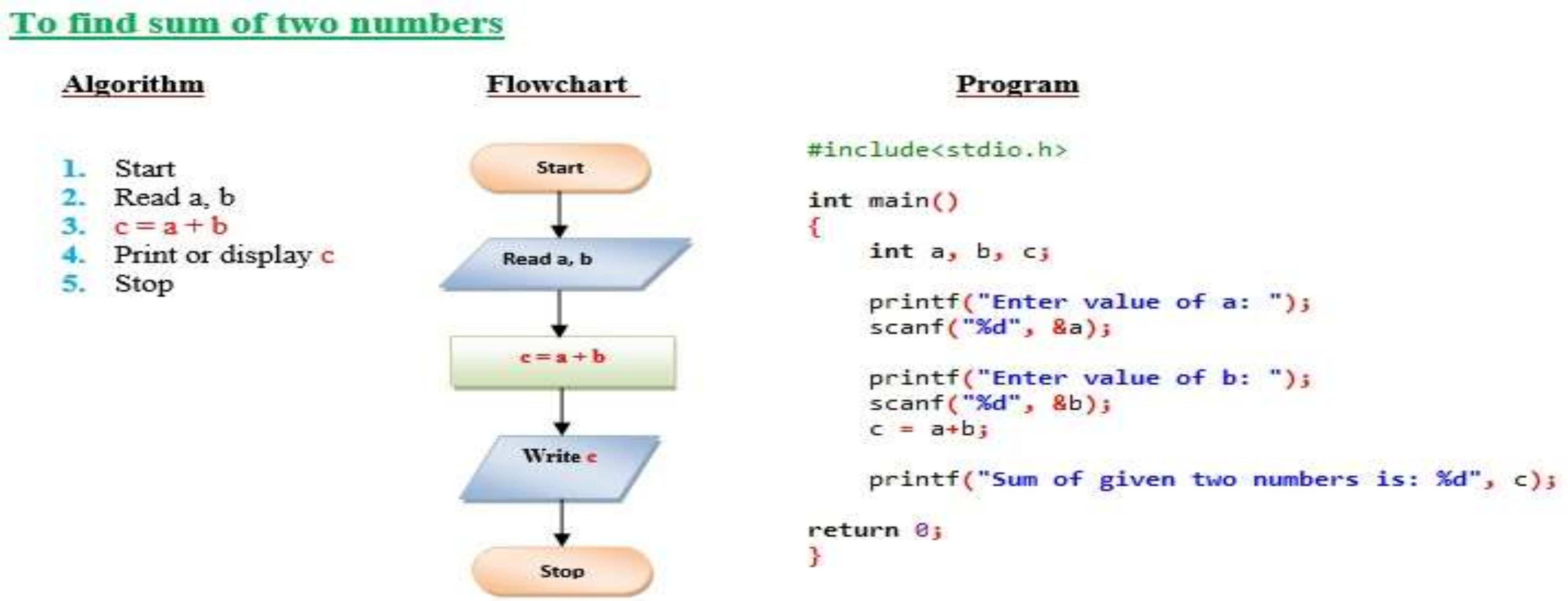
# *Building Blocks of Algorithms*



# *Building Blocks of Algorithms*

**1.Sequence:** All the instructions are executed one after another is called sequence execution

**Example:** Algorithm for Addition of TWO NUMBERS



## To find sum of two numbers

**Algorithm**

1. Start
2. Read a, b
3. c = a + b
4. Print or display c
5. Stop

**Flowchart**

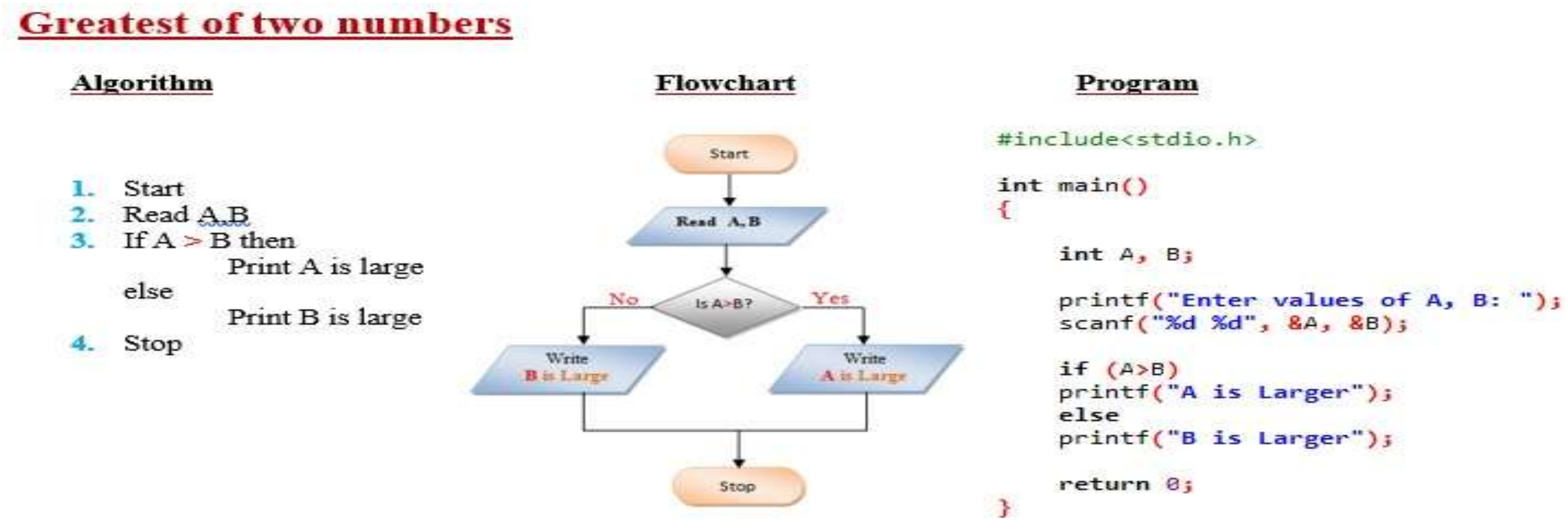
**Program**

```
#include<stdio.h>

int main()
{
    int a, b, c;

    printf("Enter value of a: ");
    scanf("%d", &a);

    printf("Enter value of b: ");
    scanf("%d", &b);
    c = a+b;

    printf("Sum of given two numbers is: %d", c);

return 0;
}
```

# Building Blocks of Algorithms

**2.Selection:** A selection statement causes the program control to be transferred to a specific part of the program based upon the condition. If the conditional test is true, one part of the program will be executed, otherwise it will execute the other part of the program.

**Example:** Algorithm for Greatest of TWO NUMBERS



## Greatest of two numbers

**Algorithm**

```
1.  Start
2.  Read A,B
3.  If A > B then
            Print A is large
    else
            Print B is large
4.  Stop
```

**Flowchart**

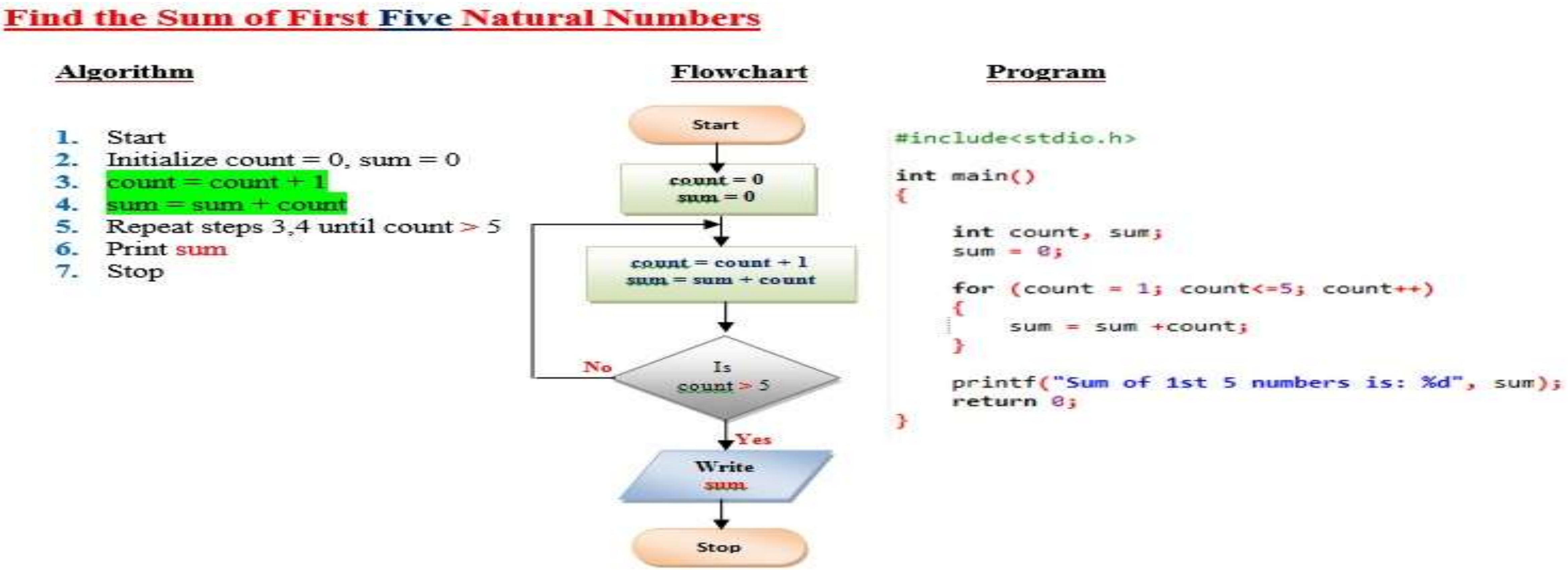
**Program**

```
#include<stdio.h>

int main()
{

     int A, B;

     printf("Enter values of A, B: ");
     scanf("%d %d", &A, &B);

     if (A>B)
     printf("A is Larger");
     else
     printf("B is Larger");

     return 0;
}
```

# Building Blocks of Algorithms

**3.Iteration:**In programs, certain set of statements are executed again and again based upon conditional test. It executed more than one time. This type of execution is called looping or iteration.

**Example:** Algorithm for sum of FIRST FIVE NATURAL NUMBERS



**Find the Sum of First Five Natural Numbers**

**Algorithm**

1. Start
2. Initialize count = 0, sum = 0
3. count = count + 1
4. sum = sum + count
5. Repeat steps 3,4 until count > 5
6. Print sum
7. Stop

**Flowchart**

**Program**

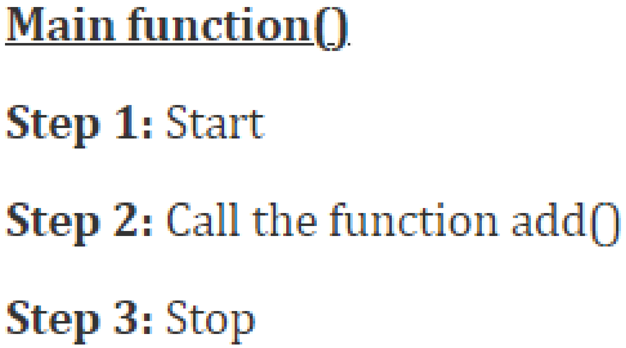
```
#include<stdio.h>

int main()
{

    int count, sum;
    sum = 0;

    for (count = 1; count<=5; count++)
    {
        sum = sum +count;
    }

    printf("Sum of 1st 5 numbers is: %d", sum);
    return 0;
}
```

# *Building Blocks of Algorithms*

**4.Functions:** Function is a sub program which consists of block of code(set of instructions) that performs a particular task. For complex problems, the problem is been divided into smaller and simpler tasks during algorithm design.

**Benefits of Using Functions:**

- Code reuse
- Reduction in line of code
- Easy to debug and test

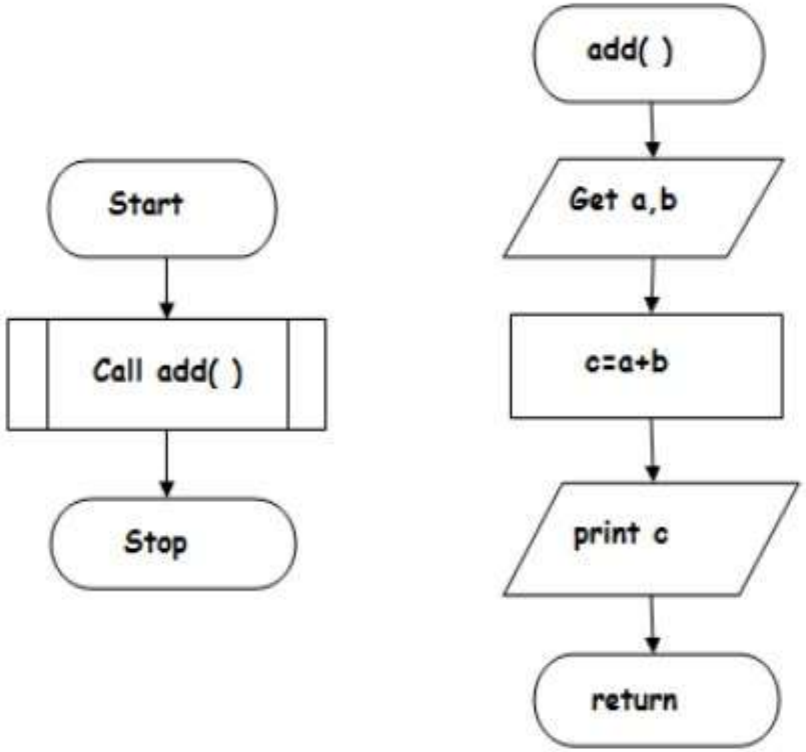


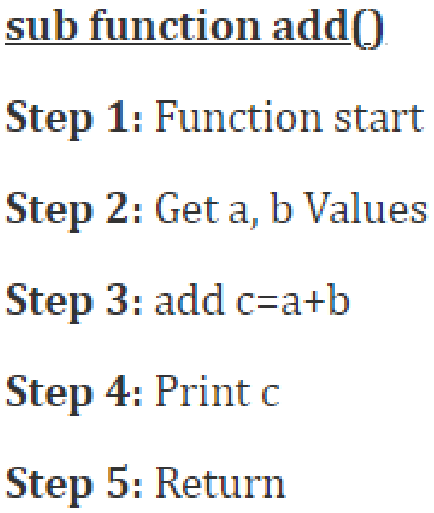
**Main function()**

**Step 1:** Start

**Step 2:** Call the function add()

**Step 3:** Stop





**sub function add()**

**Step 1:** Function start

**Step 2:** Get a, b Values

**Step 3:** add c=a+b

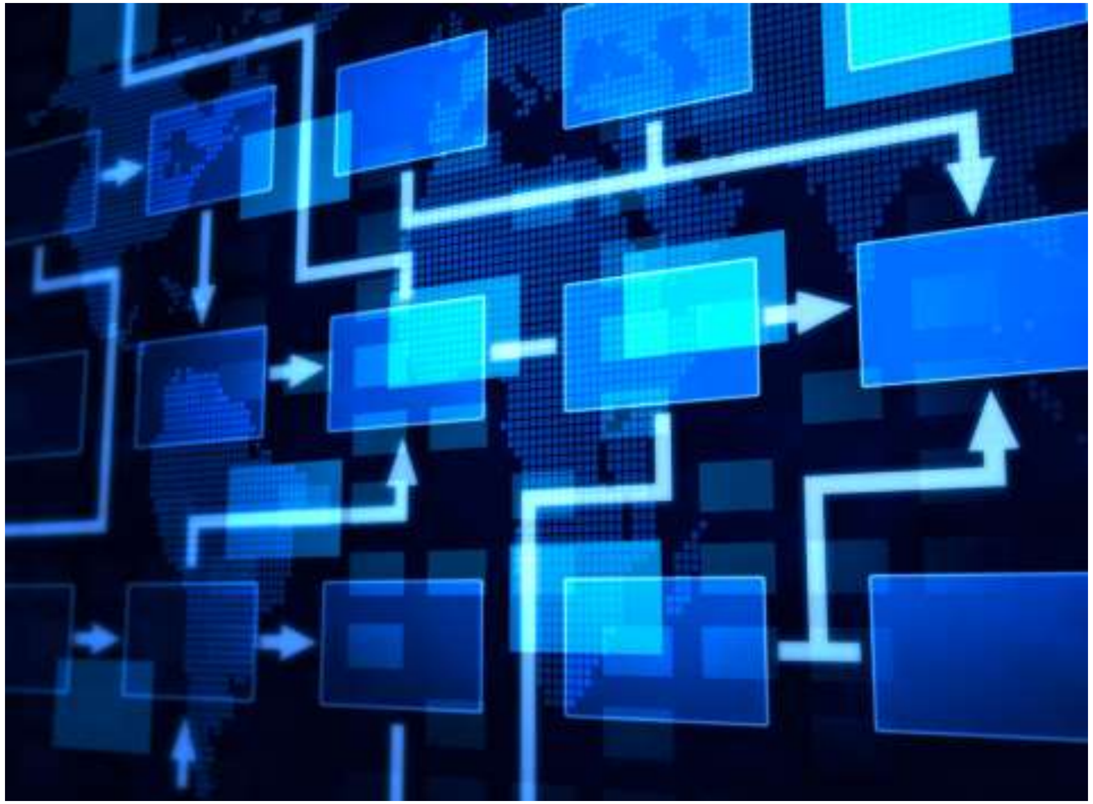
**Step 4:** Print c

**Step 5:** Return

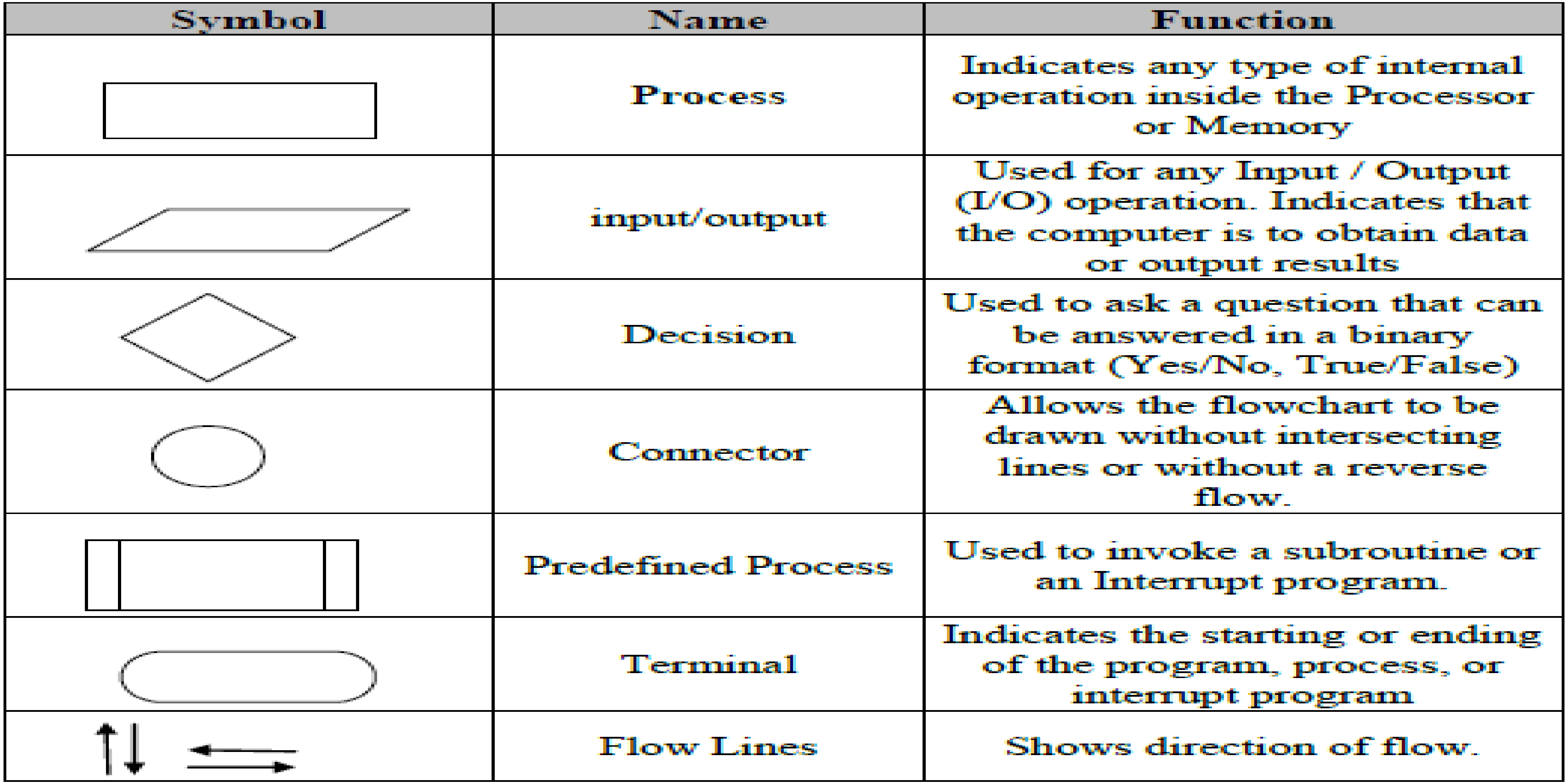
# *Flow Chart*

**What is Flow Chart?**

- Flow chart is defined as graphical representation of the logic for problem solving.
- The purpose of flowchart is making the logic of the program clear in a visual representation.



# Flow Chart



| 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. |

# *Flow Chart*

## Rules for drawing a flowchart

- The flowchart should be clear, neat and easy to follow.
- The flowchart must have a logical start and finish.
- Only one flow line should come out from a process symbol.
- Only one flow line should enter a decision symbol.
- two or three flow lines may leave the decision symbol
- Only one flow line is used with a terminal symbol.
- Intersection of flow lines should be avoided.

# *Flow Chart*

.

**Advantages of flowchart:**

1. Communication
2. Effective analysis
3. Proper documentation
4. Efficient Coding
5. Proper Debugging

**Disadvantages of flowchart:**

1. Complex logic
2. Alterations and Modifications
3. Reproduction
4. Cost

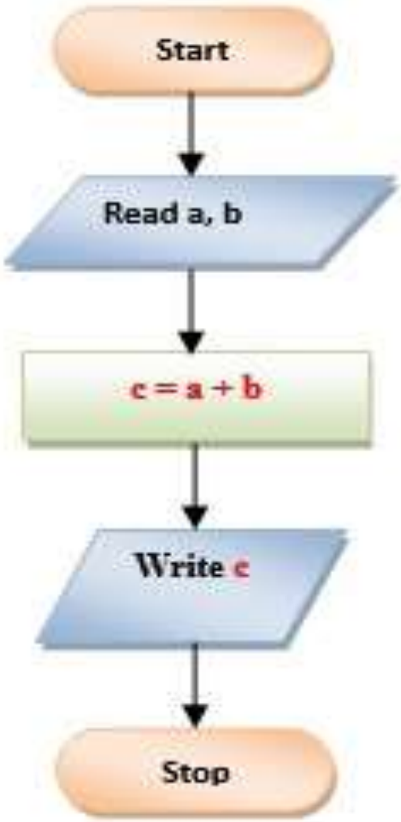
# *Flow Chart*

## To find sum of two numbers

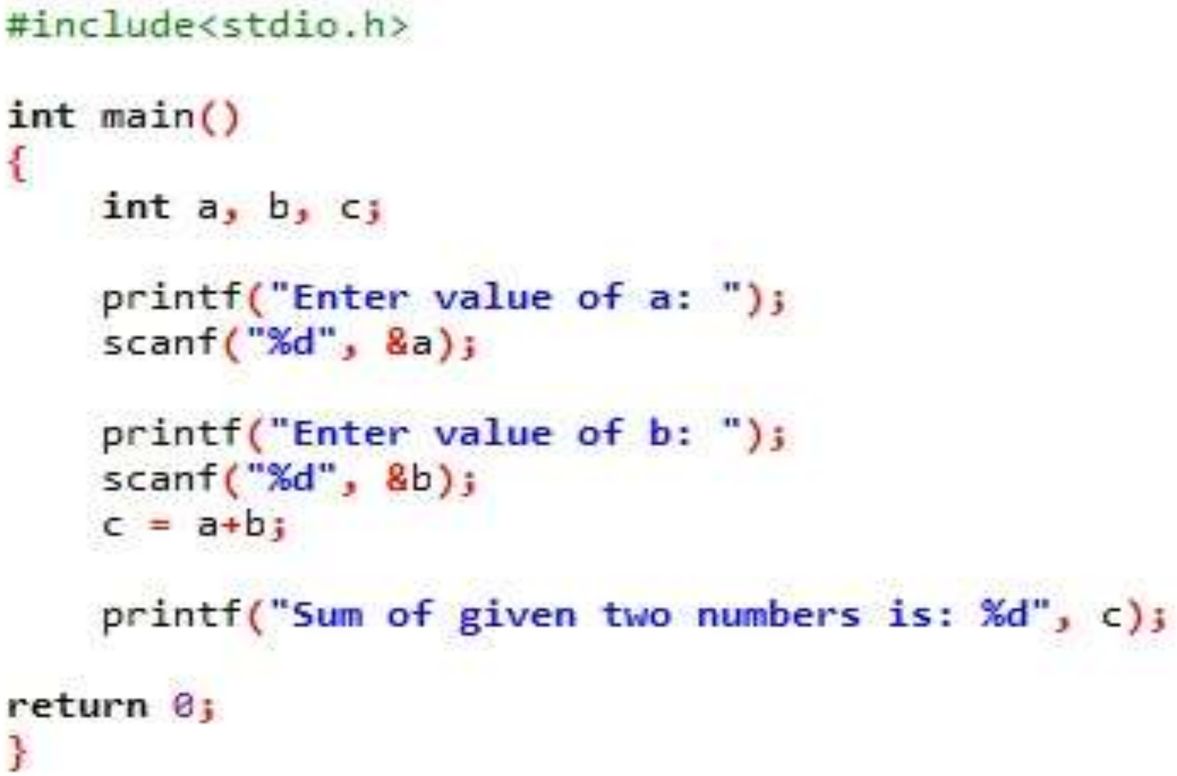
### Algorithm

1. Start
2. Read a, b
3. c = a + b
4. Print or display c
5. Stop

### Flowchart



### Program

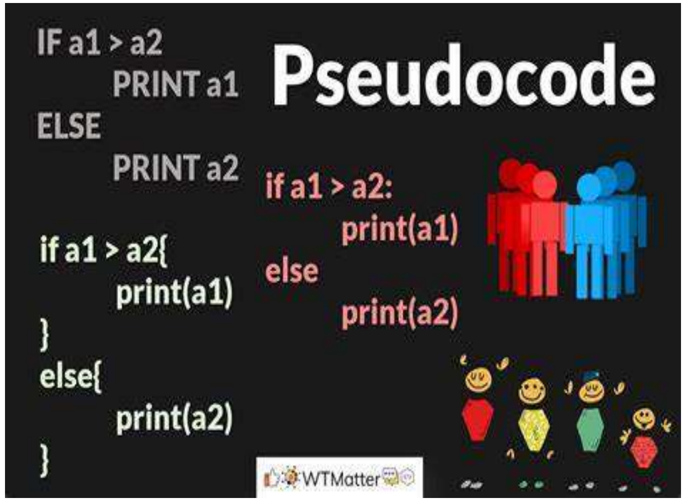


```
#include<stdio.h>

int main()
{
    int a, b, c;

    printf("Enter value of a: ");
    scanf("%d", &a);

    printf("Enter value of b: ");
    scanf("%d", &b);
    c = a+b;

    printf("Sum of given two numbers is: %d", c);

return 0;
}
```

# *Pseudo Code*

## What is Pseudo Code?

- Pseudo code consists of short, readable and formally styled English languages used for explain an algorithm.
- It does not include details like variable declaration, subroutines.
- It is easier to understand for the programmer or non programmer to understand the general working of the program.
- It is not a machine readable
- Pseudo code can’t be compiled and executed.
- No standard syntax.



Pseudocode

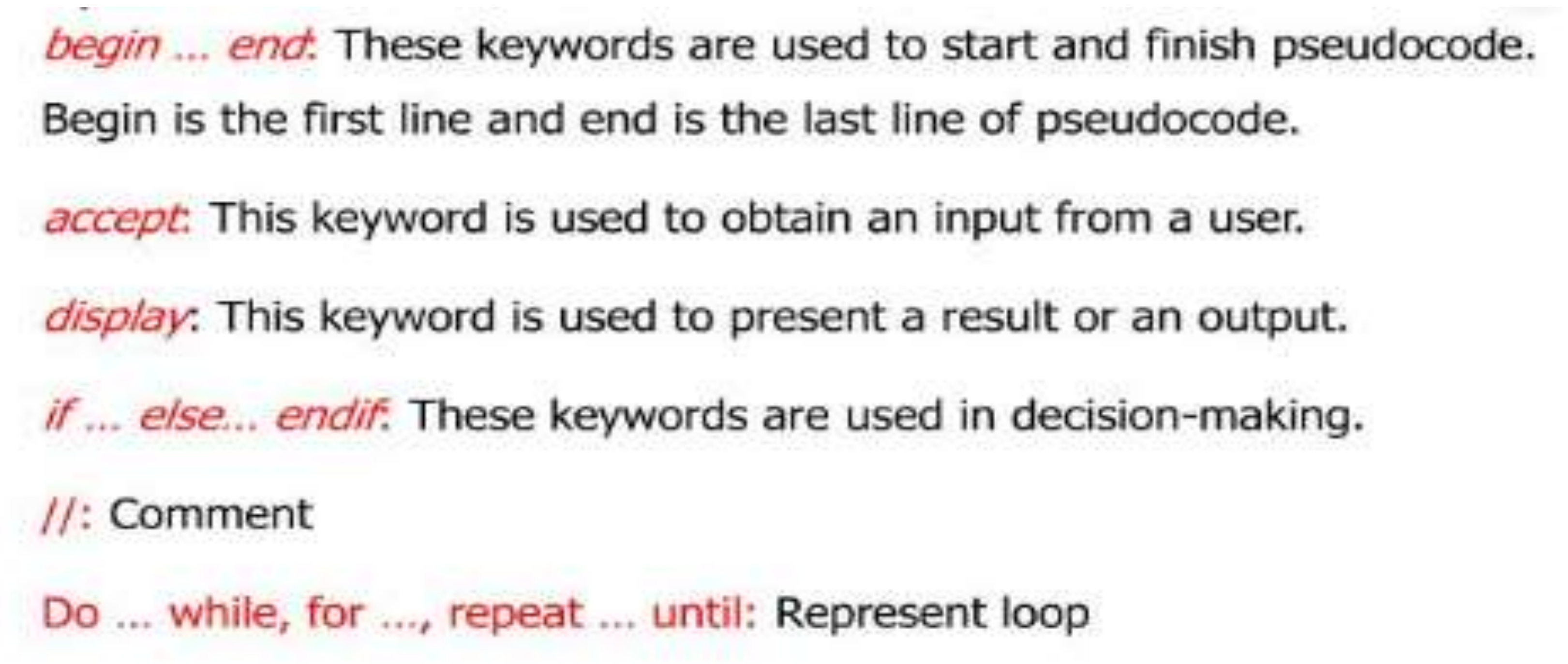
```
IF a1 > a2
      PRINT a1
ELSE
      PRINT a2
```

```
if a1 > a2{
      print(a1)
}
else{
      print(a2)
}
```

```
if a1 > a2:
      print(a1)
else
      print(a2)
```

# *Pseudo Code*

## Common keywords used in pseudocode



*begin ... end*: These keywords are used to start and finish pseudocode. Begin is the first line and end is the last line of pseudocode.

*accept*: This keyword is used to obtain an input from a user.

*display*: This keyword is used to present a result or an output.

*if ... else... endif*: These keywords are used in decision-making.

//: Comment

Do ... while, for ..., repeat ... until: Represent loop

# *Pseudo Code*

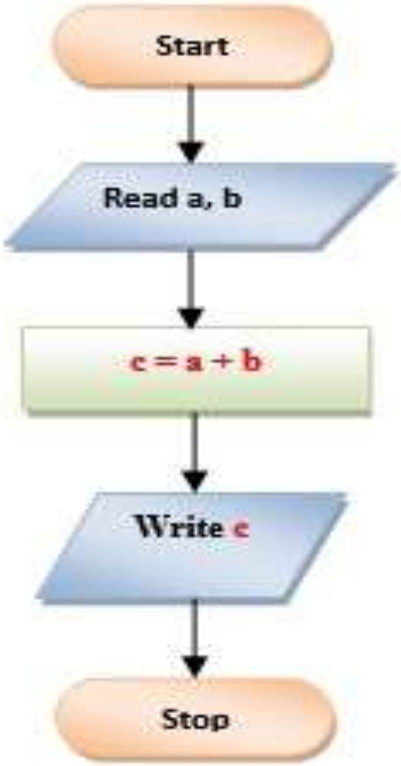
**Example for Sequence Method:**

**To find sum of two numbers**

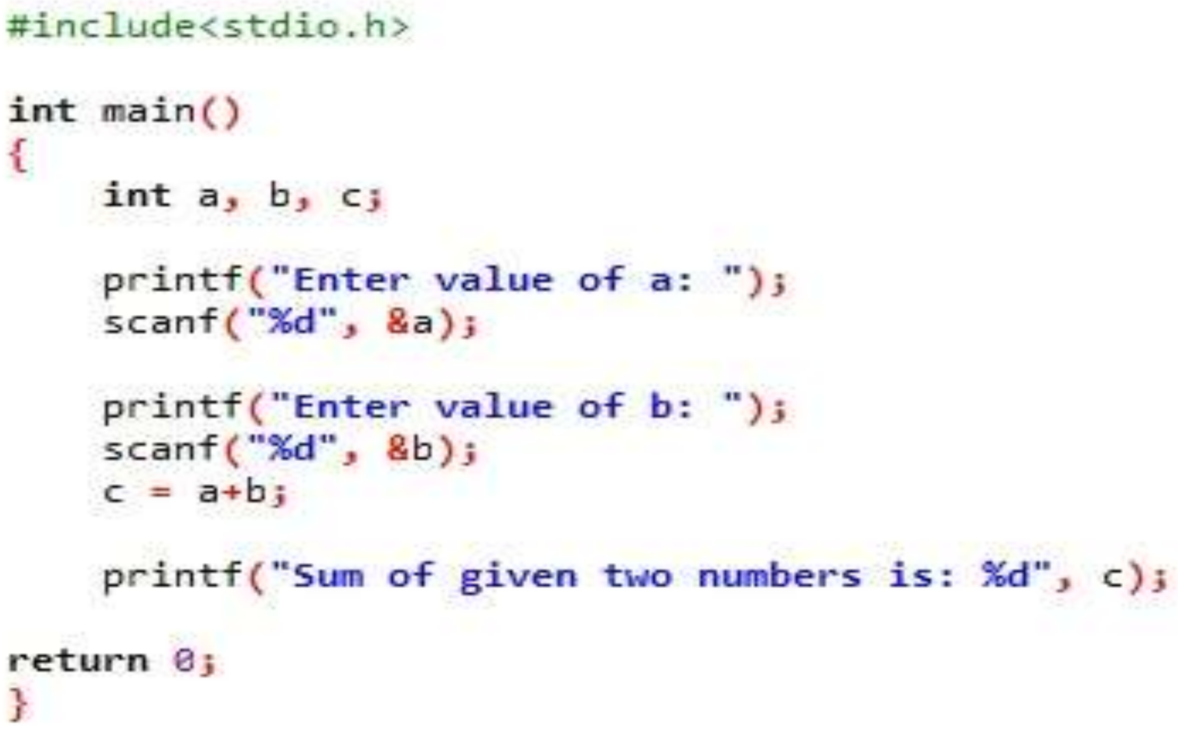
Pseudo code

```
BEGIN

GET a,b

ADD c=a+b

PRINT c

END
```

**Flowchart**



**Program**



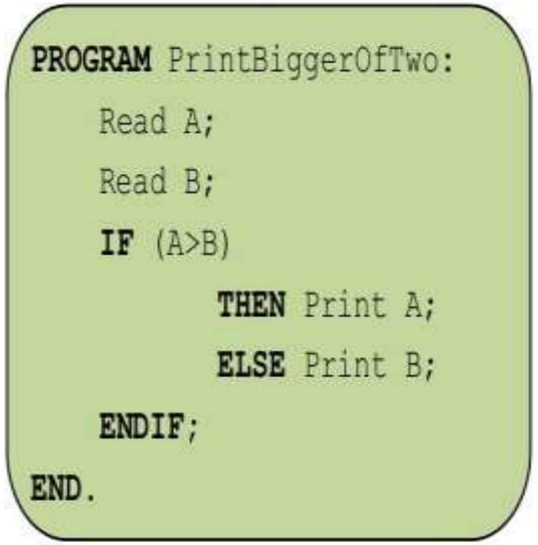
```
#include<stdio.h>

int main()
{
    int a, b, c;

    printf("Enter value of a: ");
    scanf("%d", &a);

    printf("Enter value of b: ");
    scanf("%d", &b);
    c = a+b;

    printf("Sum of given two numbers is: %d", c);

return 0;
}
```

# *Pseudo Code*

**Example for Selection Method:**

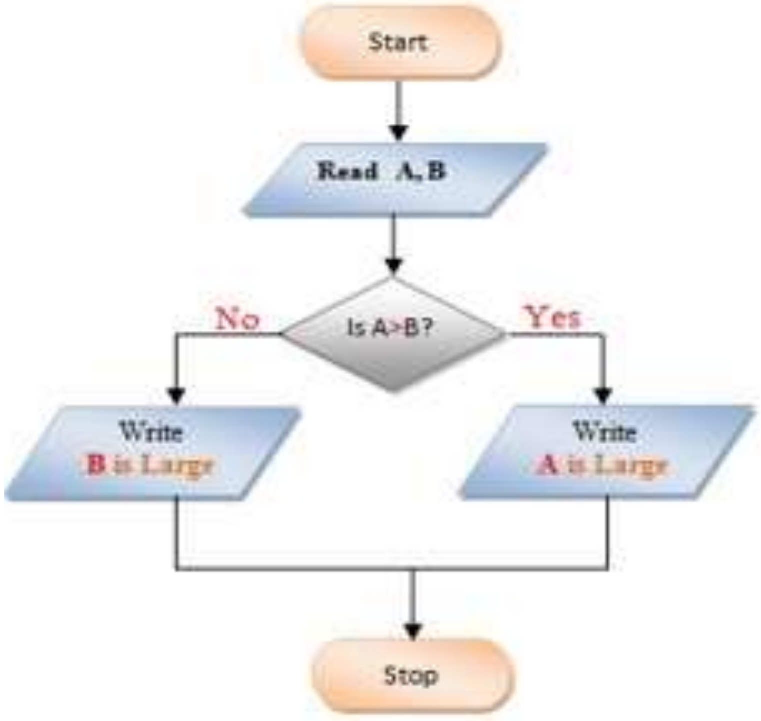
## Greatest of two numbers

### Pseudocode

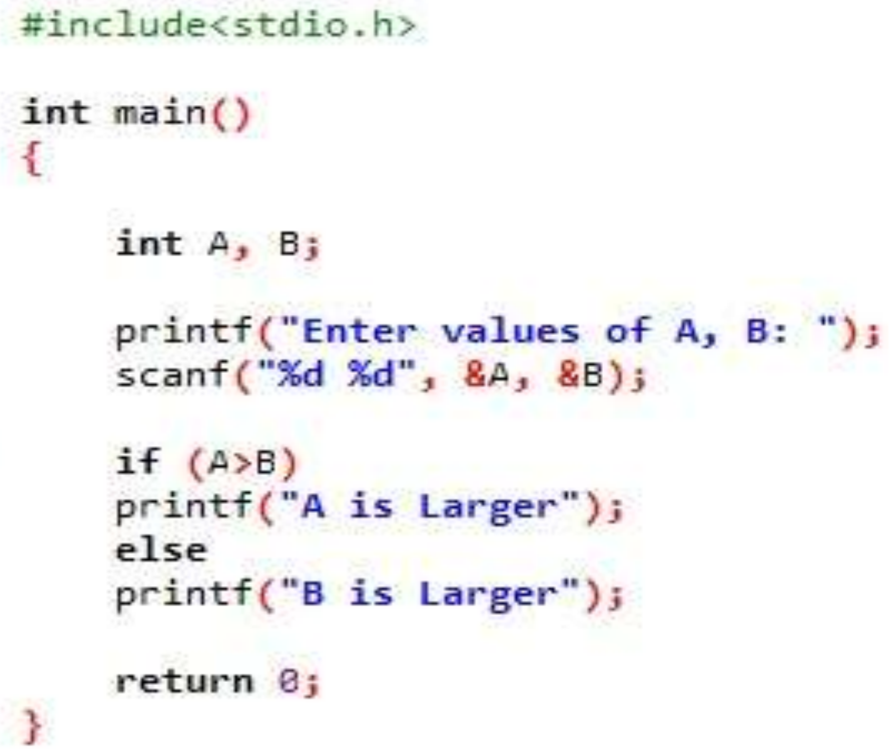


```
PROGRAM PrintBiggerOfTwo:
    Read A;
    Read B;
    IF (A>B)
        THEN Print A;
        ELSE Print B;
    ENDIF;
END.
```

### Flowchart



### Program



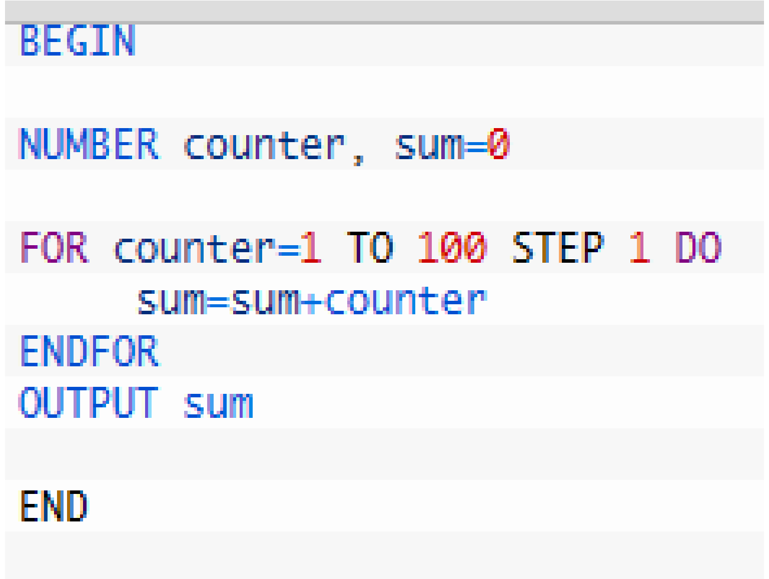
```
#include<stdio.h>

int main()
{

    int A, B;

    printf("Enter values of A, B: ");
    scanf("%d %d", &A, &B);

    if (A>B)
    printf("A is Larger");
    else
    printf("B is Larger");

    return 0;
}
```

# Pseudo Code

**Example for Iteration Method:**

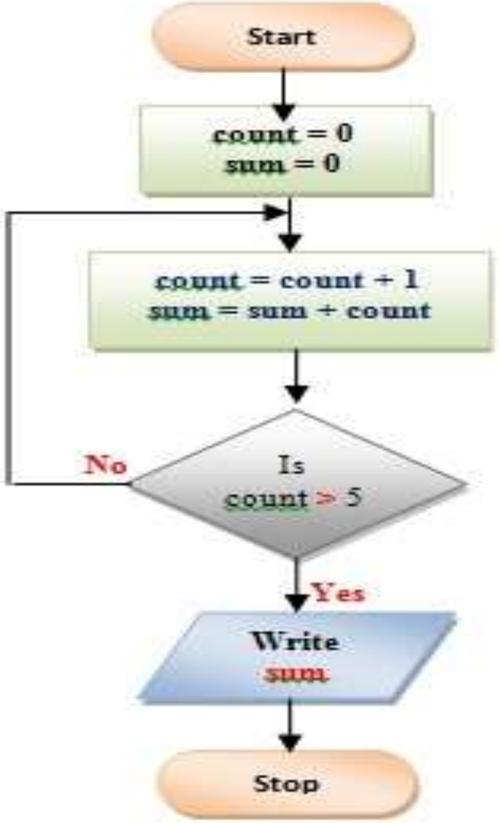
## Find the Sum of First Five Natural Numbers

Pseudo code

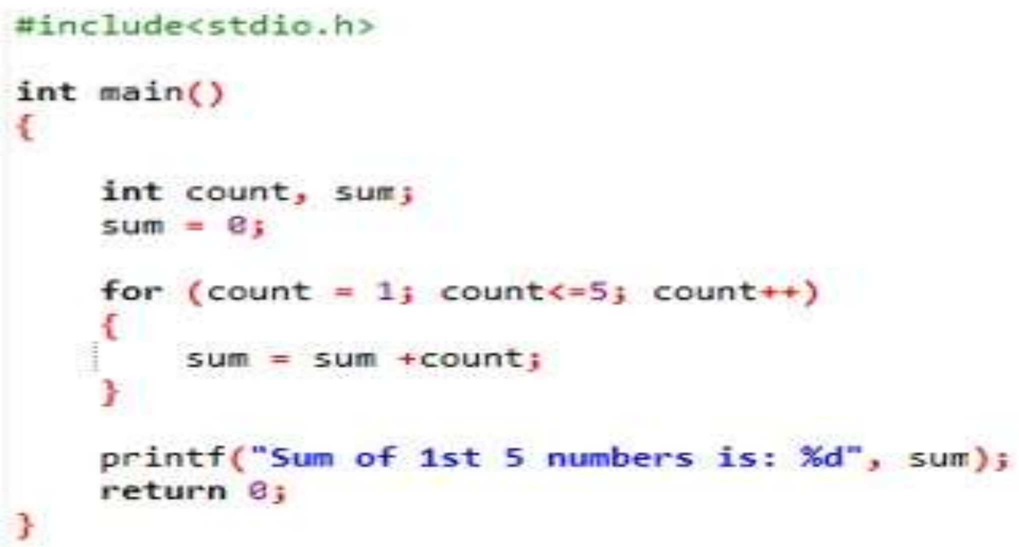


```
BEGIN

NUMBER counter, sum=0

FOR counter=1 TO 100 STEP 1 DO
     sum=sum+counter
ENDFOR
OUTPUT sum

END
```

**Flowchart**



**Program**



```
#include<stdio.h>

int main()
{

     int count, sum;
     sum = 0;

     for (count = 1; count<=5; count++)
     {
          sum = sum +count;
     }

     printf("Sum of 1st 5 numbers is: %d", sum);
     return 0;
}
```

# *Algorithmic Problem Solving*

Algorithmic problem solving is solving problem that require the formulation of an algorithm for the solution

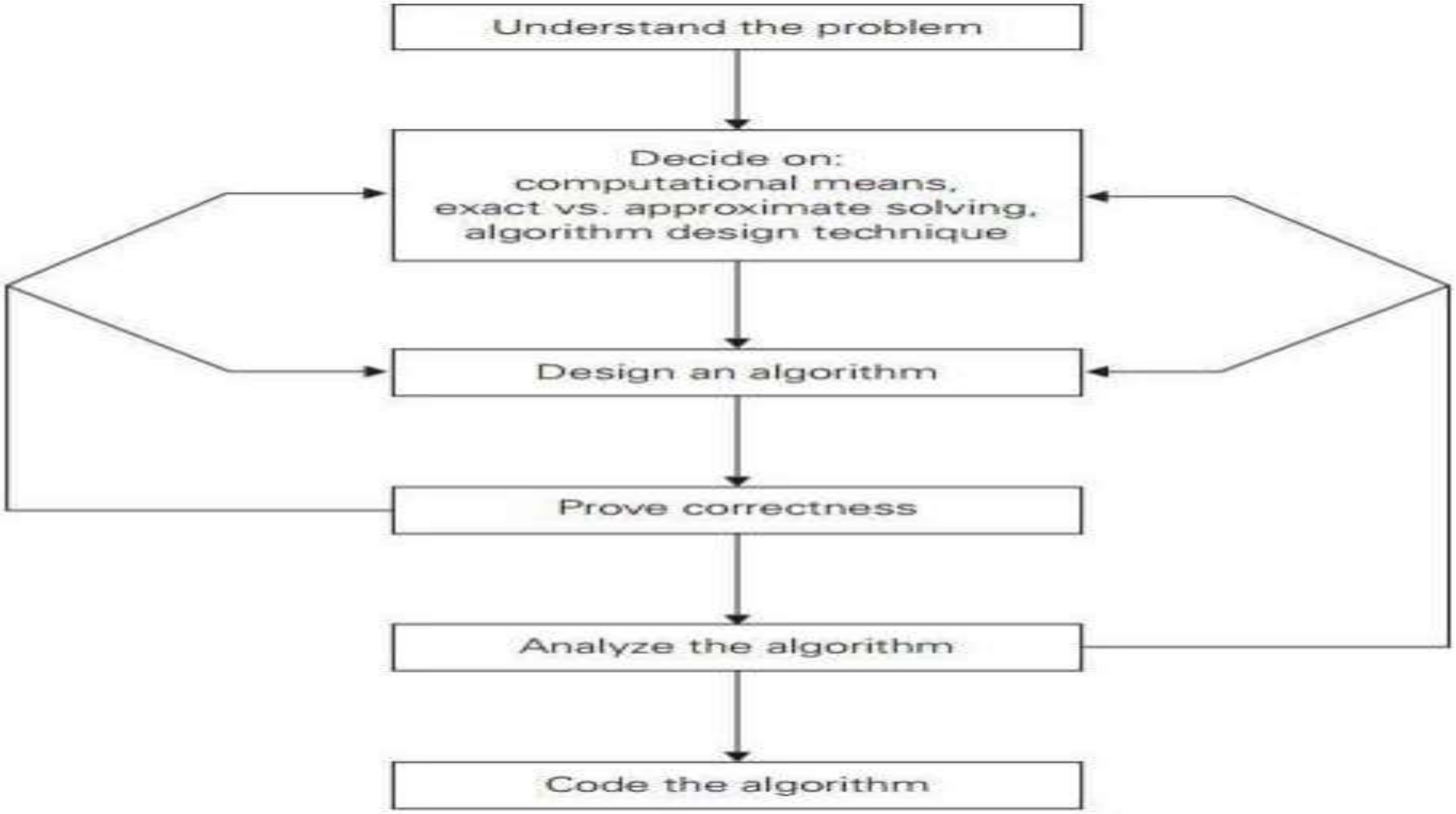


**FIGURE 1.2** Algorithm design and analysis process.

# Algorithmic Problem Solving

**1.Understanding the Problem**

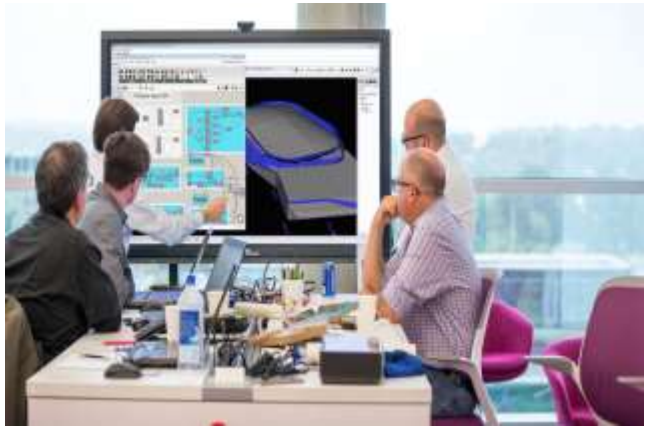
- It is the process of finding the input of the problem that the algorithm solves.
- It is very important to specify exactly the set of inputs the algorithm needs to handle.

**2.Ascertaining the Capabilities of the Computational Device**

- If the instructions are executed one after another, it is called sequential algorithm.
- If the instructions are executed concurrently, it is called parallel algorithm.

**3.Choosing between Exact and Approximate Problem Solving**

- To choose between solving the problem exactly or solving it approximately.
- Based on this, the algorithms are classified as exact *algorithm* and *approximation algorithm.*



# Algorithmic Problem Solving

**4.Deciding a data structure:**

- Data structure plays a vital role in designing and analysis the algorithms.
- Algorithm+ Data structure=programs.

**5.Algorithm Design Techniques**

- Learning these techniques is of utmost importance for the following reasons.
- First, they provide guidance for designing algorithms for new problems.
- Second, algorithms are the cornerstone of computer science.

**6.Methods of Specifying an Algorithm**

- *Pseudocode*
  - *Flowchart*

# *Algorithmic Problem Solving*

**7.Proving an Algorithm’s Correctness**

- Once an algorithm has been specified, you have to prove its ***correctness***.
- A common technique for proving correctness is to use mathematical induction because an algorithm’s iterations provide a natural sequence of steps needed for such proofs.
- It cannot prove the algorithm’s correctness conclusively.

**8.Analysing an Algorithm**

- ***Efficiency***.
  1. *Time efficiency*
  2. *Space efficiency*
- ***simplicity***.

**9.Coding an Algorithm**

- Most algorithms are destined to be ultimately implemented as computer programs. Programming an algorithm presents both a peril and an opportunity.

Thank You!