CSC-110
COMPUTING FUNDAMENTALS
COMPUTER PROGRAMMING FUNDAMENTALS

INTRODUCTION

- Computer is an electronic device that accepts data, processes it, and generates the relevant output. It can perform both simple and complex tasks with very high speed and accuracy.
- Computers need to be instructed about "how" the task is to be performed. The set of instructions that instruct the computer about the way the task is to be performed is called a program.
- A program is required for processing all kind of tasks—simple tasks like addition of two numbers, and complex tasks like gaming etc.

PROGRAM DEVELOPMENT LIFE CYCLE

- A program is needed to instruct the computer about the way a task is to be performed. The instructions in a program have three essential parts:

1. Instructions to accept the input data that needs to be processed,

2. Instructions that will act upon the input data and process it, and

3. Instructions to provide the output to user.

PROGRAM DEVELOPMENT LIFE CYCLE

- Steps of a program development cycle:
- **Program Analysis**
- Understand the problem
- Have multiple solutions
- Select a solution
- **Program Design**
- Write Algorithm
- Write Flowchart
- Write Pseudo code

PROGRAM DEVELOPMENT LIFE CYCLE

Program Development

- Choose a programming language
- Write the program by converting the pseudo code, and then using the programming language.
- Compile the program and remove syntax errors, if any
- Execute the program.
- Test the program. Check the output results with different inputs. If the output is incorrect, modify the program to get correct results.
- Install the tested program on the user’s computer.

PROGRAM DEVELOPMENT LIFE CYCLE

- **Program Documentation and maintenance**

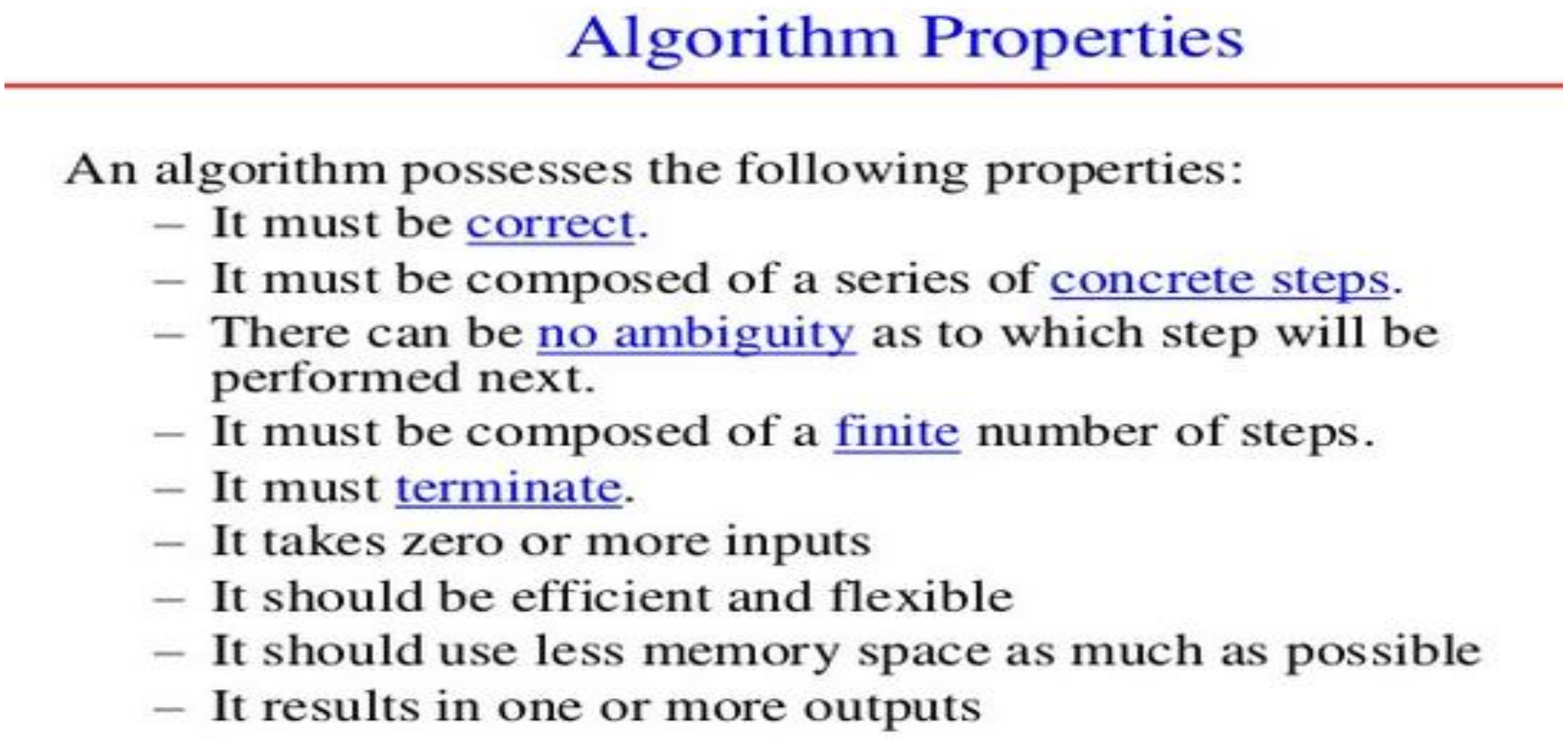
• Document the program, for later use.

• Maintain the program for updating, removing errors, changing requirements etc.

ALGORITHM

- *Algorithm* is an ordered sequence of finite, well defined, unambiguous instructions for completing a task. Algorithm is an English-like representation of the logic which is used to solve the problem.
- It is a step- by-step procedure for solving a task or a problem. The steps must be ordered, unambiguous and finite in number.

ALGORITHM

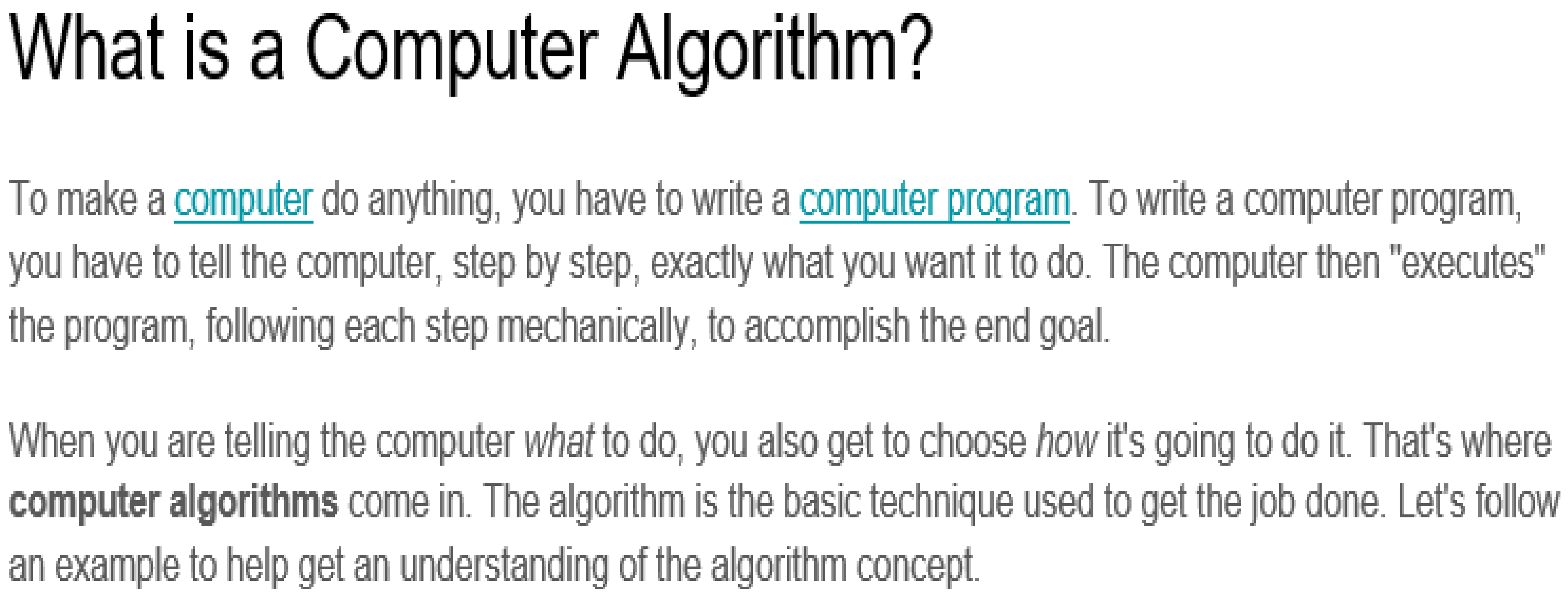


Algorithm Properties

An algorithm possesses the following properties:

- It must be correct.
- It must be composed of a series of concrete steps.
- There can be no ambiguity as to which step will be performed next.
- It must be composed of a finite number of steps.
- It must terminate.
- It takes zero or more inputs
- It should be efficient and flexible
- It should use less memory space as much as possible
- It results in one or more outputs

ALGORITHM

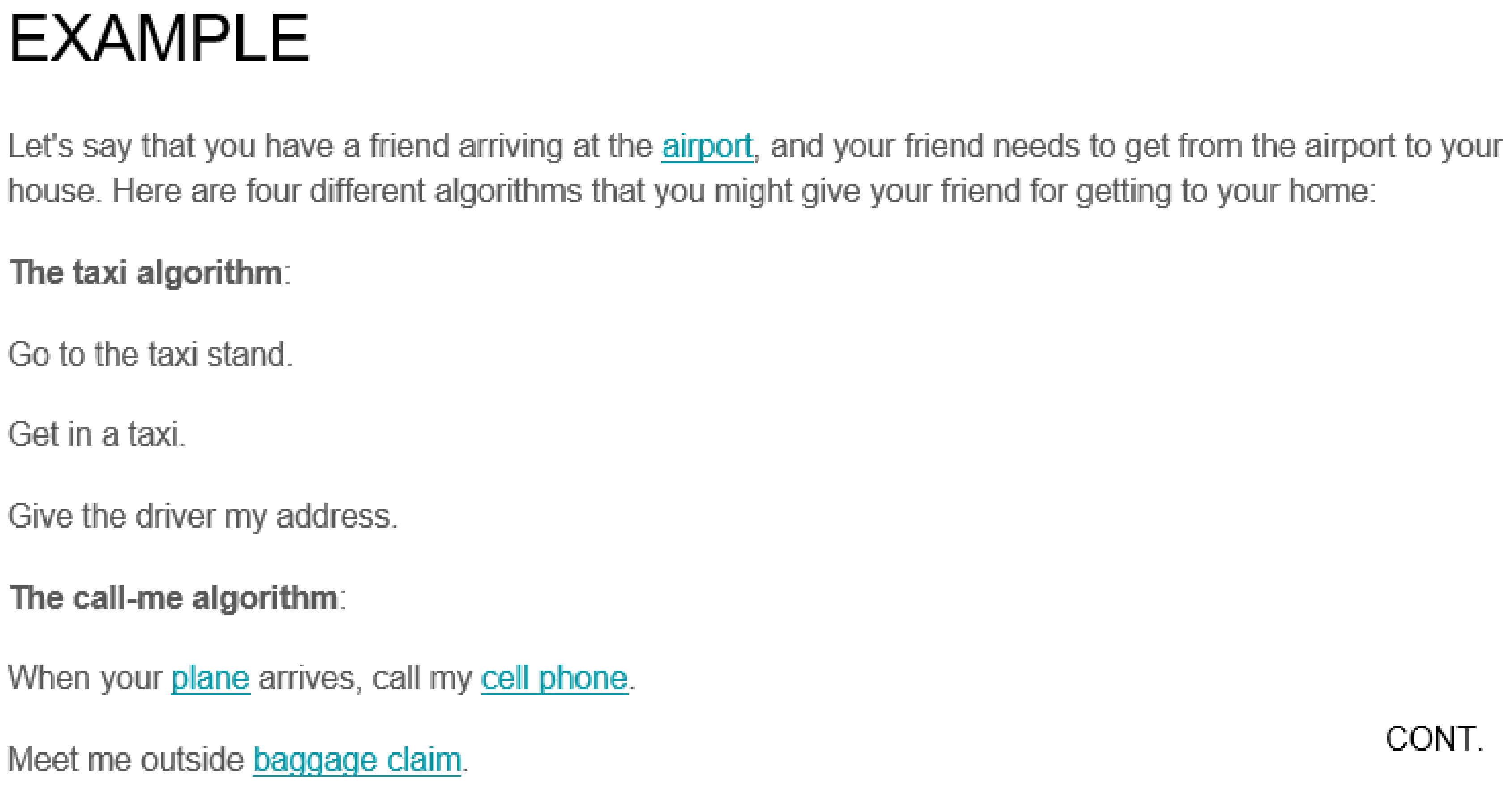


What is a Computer Algorithm?

To make a computer do anything, you have to write a computer program. To write a computer program, you have to tell the computer, step by step, exactly what you want it to do. The computer then "executes" the program, following each step mechanically, to accomplish the end goal.

When you are telling the computer *what* to do, you also get to choose *how* it's going to do it. That's where **computer algorithms** come in. The algorithm is the basic technique used to get the job done. Let's follow an example to help get an understanding of the algorithm concept.

ALGORITHM



EXAMPLE

Let's say that you have a friend arriving at the airport, and your friend needs to get from the airport to your house. Here are four different algorithms that you might give your friend for getting to your home:

The taxi algorithm:

Go to the taxi stand.

Get in a taxi.

Give the driver my address.

The call-me algorithm:

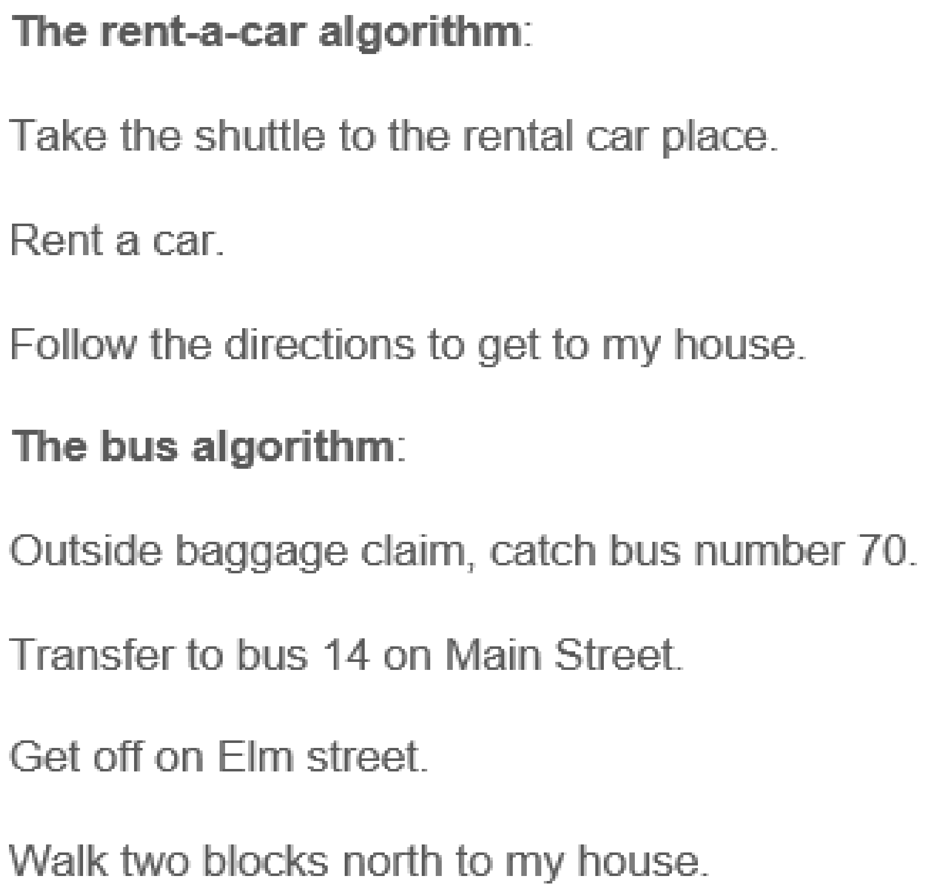
When your plane arrives, call my cell phone.

Meet me outside baggage claim.

CONT.

ALGORITHM

EXAMPLE



The rent-a-car algorithm:

Take the shuttle to the rental car place.

Rent a car.

Follow the directions to get to my house.

The bus algorithm:

Outside baggage claim, catch bus number 70.

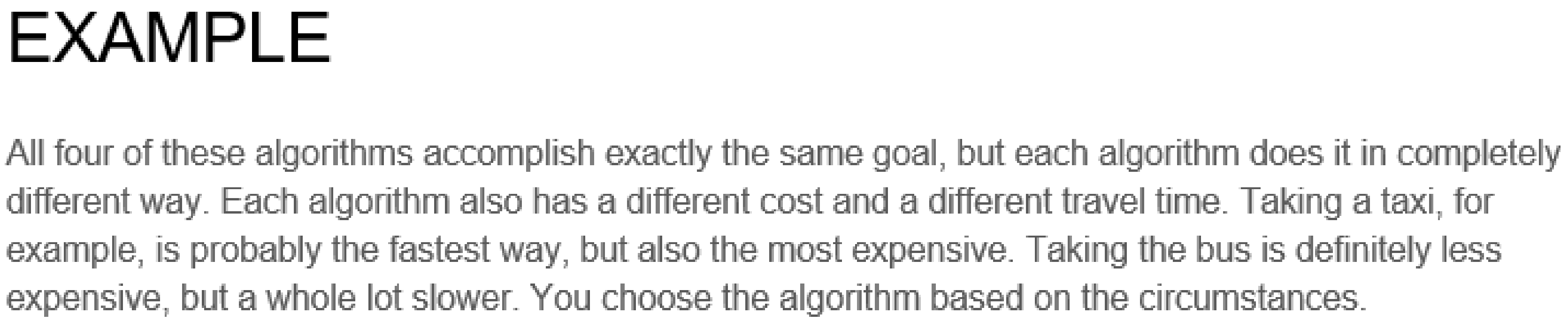
Transfer to bus 14 on Main Street.

Get off on Elm street.

Walk two blocks north to my house.

CONT.

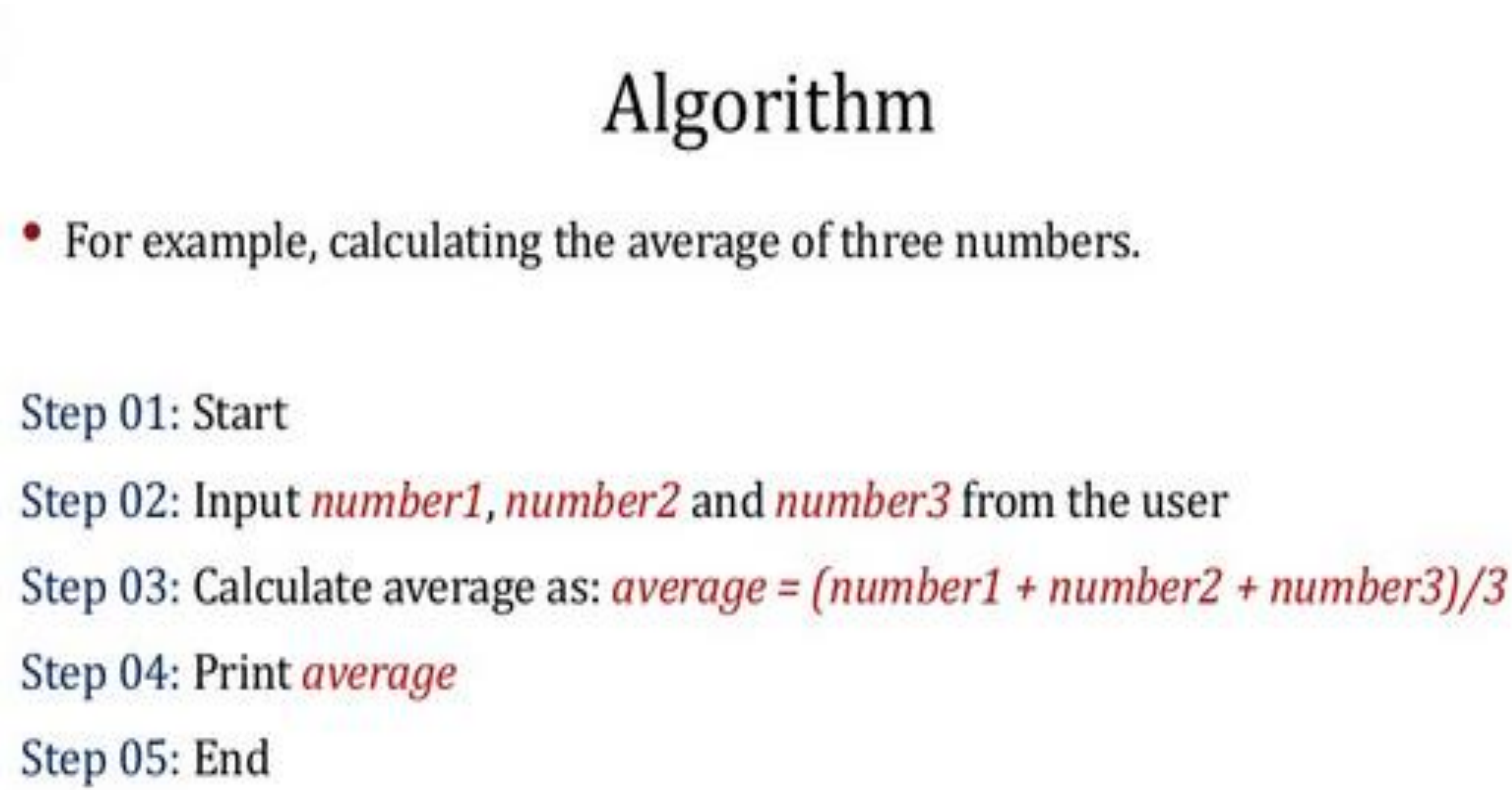
ALGORITHM



EXAMPLE

All four of these algorithms accomplish exactly the same goal, but each algorithm does it in completely different way. Each algorithm also has a different cost and a different travel time. Taking a taxi, for example, is probably the fastest way, but also the most expensive. Taking the bus is definitely less expensive, but a whole lot slower. You choose the algorithm based on the circumstances.

ALGORITHM

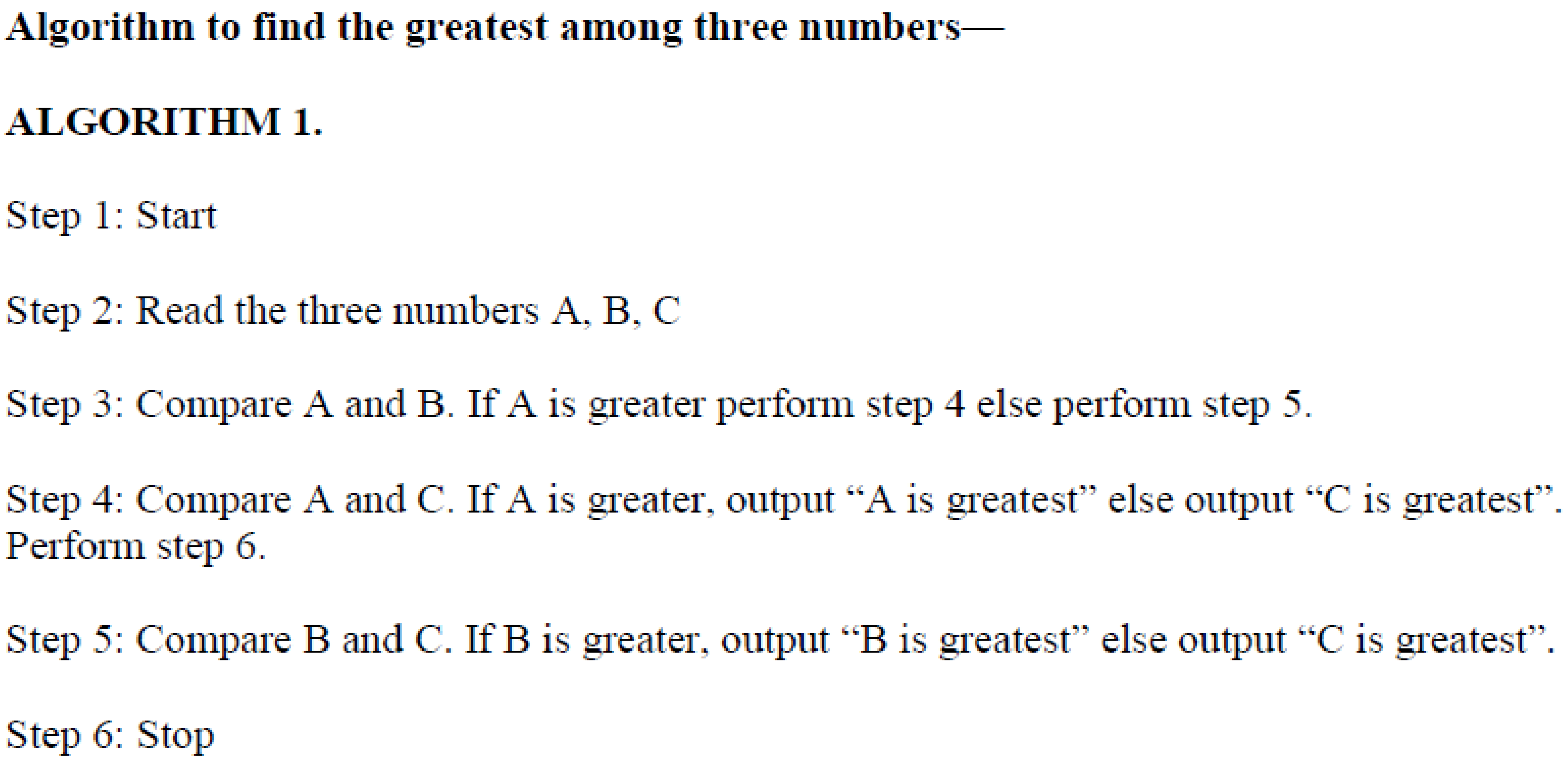


Algorithm

- For example, calculating the average of three numbers.

```
Step 01: Start
Step 02: Input number1, number2 and number3 from the user
Step 03: Calculate average as: average = (number1 + number2 + number3)/3
Step 04: Print average
Step 05: End
```

ALGORITHM



Algorithm to find the greatest among three numbers—

ALGORITHM 1.

Step 1: Start

Step 2: Read the three numbers A, B, C

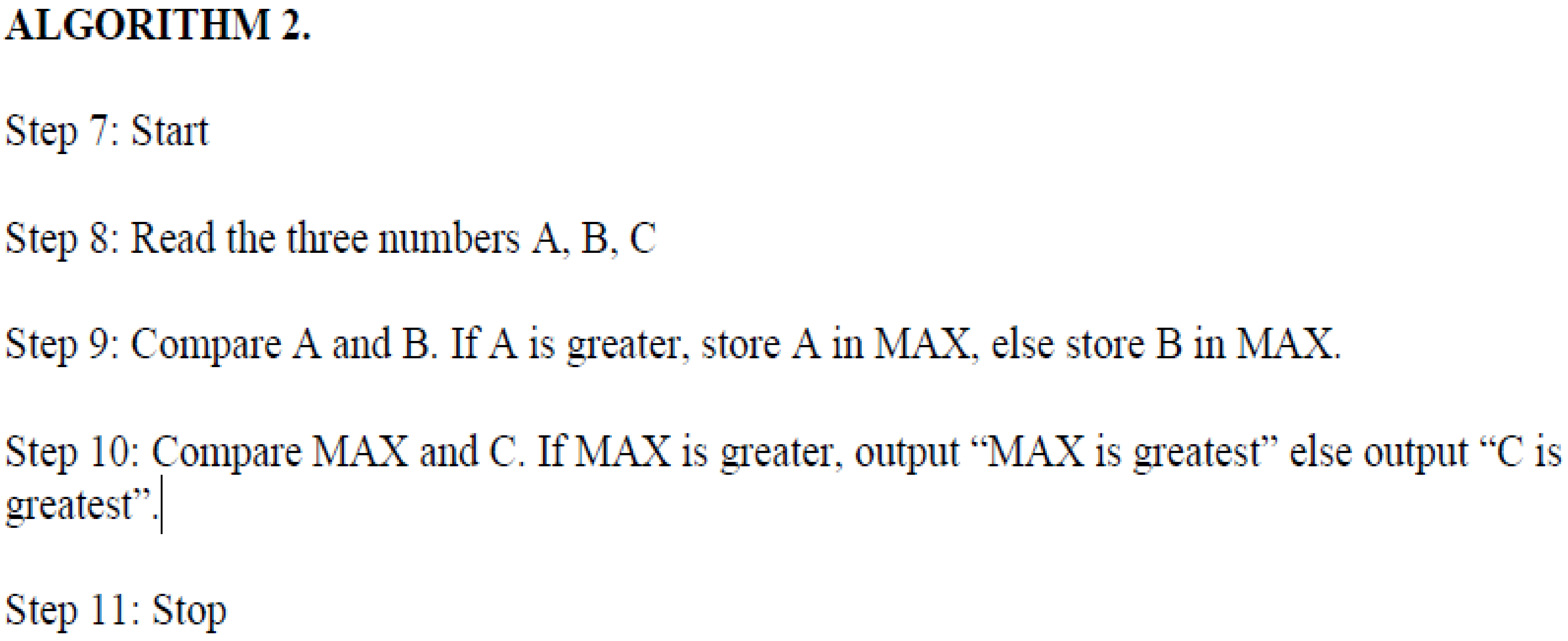
Step 3: Compare A and B. If A is greater perform step 4 else perform step 5.

Step 4: Compare A and C. If A is greater, output “A is greatest” else output “C is greatest”. Perform step 6.

Step 5: Compare B and C. If B is greater, output “B is greatest” else output “C is greatest”.

Step 6: Stop

ALGORITHM



ALGORITHM 2.

Step 7: Start

Step 8: Read the three numbers A, B, C

Step 9: Compare A and B. If A is greater, store A in MAX, else store B in MAX.

Step 10: Compare MAX and C. If MAX is greater, output “MAX is greatest” else output “C is greatest”.

Step 11: Stop

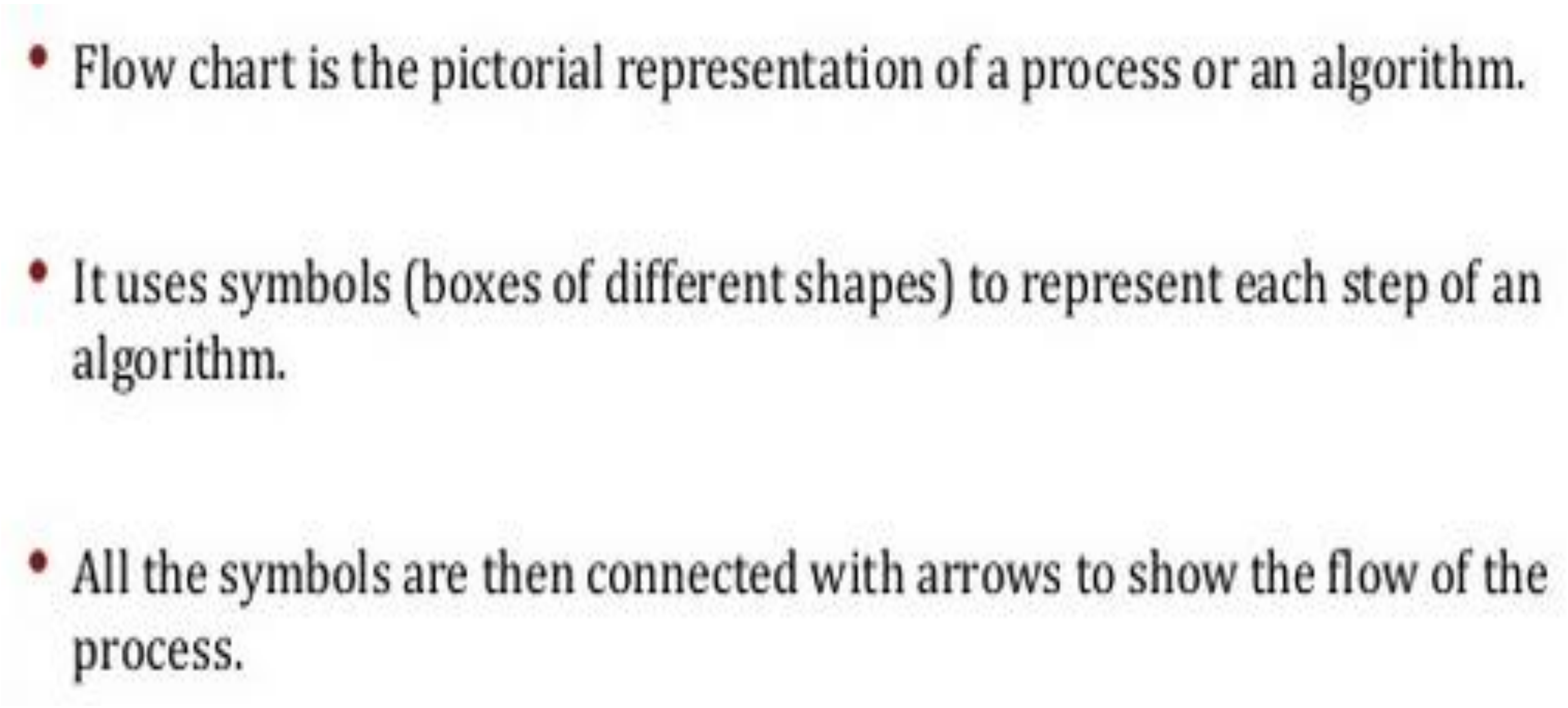
ALGORITHM

- Both the algorithms accomplish the same goal, but in different ways. The programmer selects the algorithm based on the advantages and disadvantages of each algorithm. For example, the first algorithm has more number of comparisons, whereas in the second algorithm an additional variable MAX is required.

CONTROL STRUCTURES

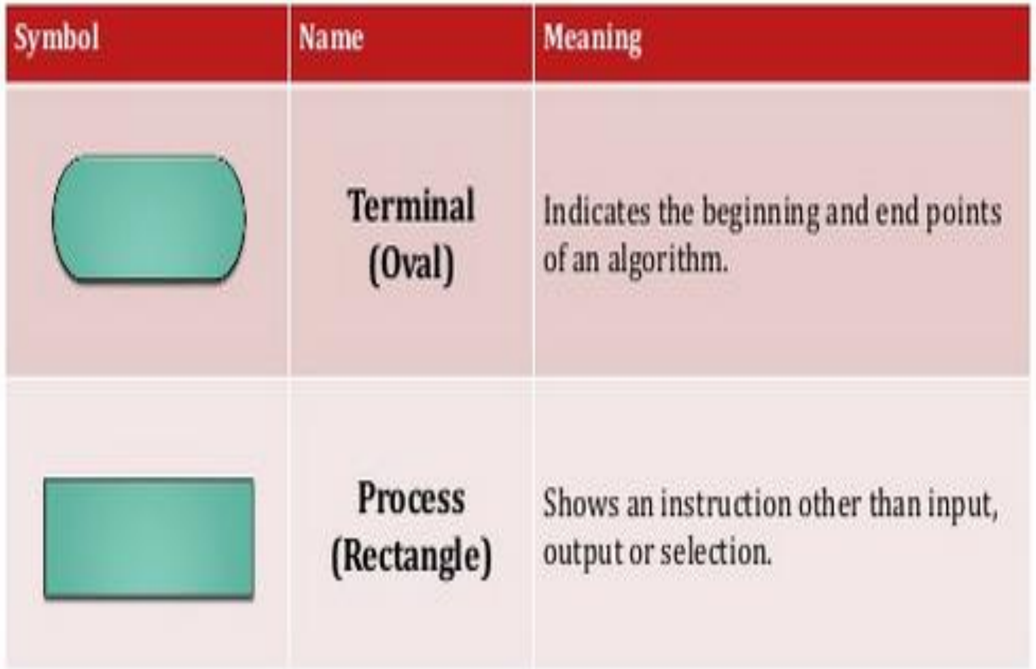
- The logic of a program may not always be a linear sequence of statements to be executed in that order. The logic of the program may require execution of a statement based on a decision. It may repetitively execute a set of statements unless some condition is met. Control structures specify the statements to be executed and the order of execution of statements.
- Flowchart and Pseudo code use control structures for representation. There are three kinds of control structures:
- Sequential—instructions are executed in linear order
- Selection (branch or conditional)—it asks a true/false question and then selects the next instruction based on the answer
- Iterative (loop)—it repeats the execution of a block of instructions.

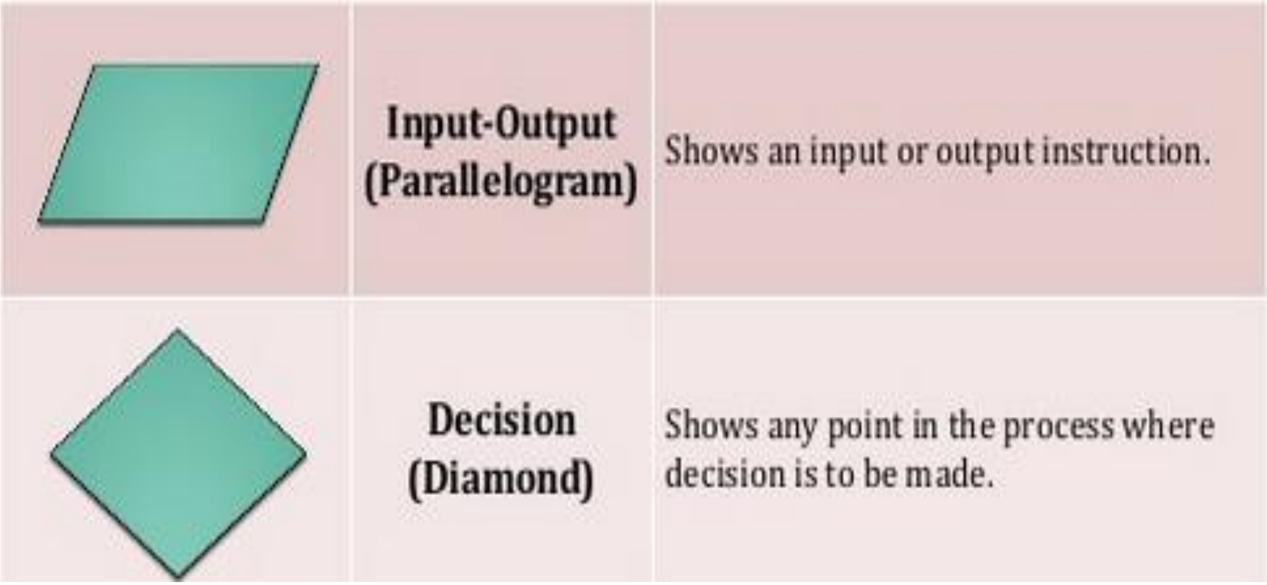
FLOWCHART



- Flow chart is the pictorial representation of a process or an algorithm.
- It uses symbols (boxes of different shapes) to represent each step of an algorithm.
- All the symbols are then connected with arrows to show the flow of the process.

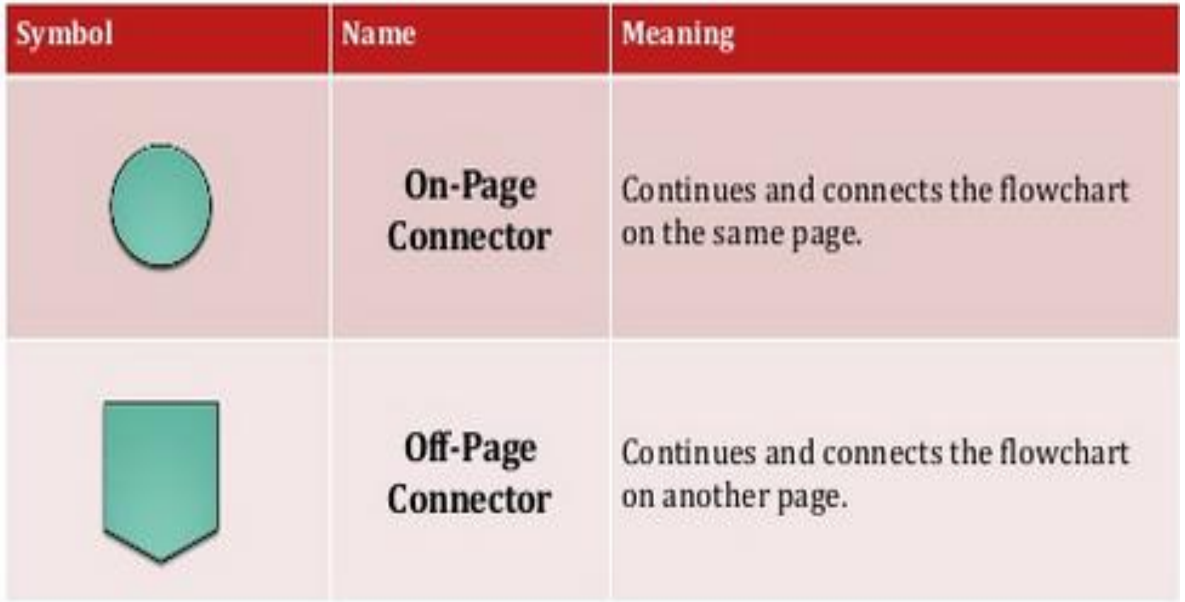
FLOWCHART



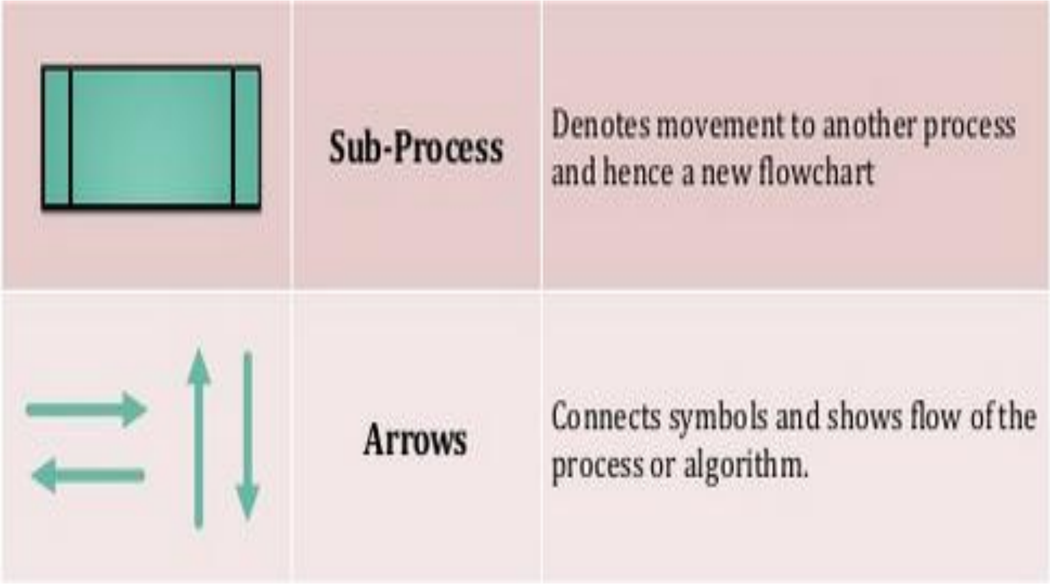


Symbol	Name	Meaning
⬭	Terminal (Oval)	Indicates the beginning and end points of an algorithm.
▭	Process (Rectangle)	Shows an instruction other than input, output or selection.
▱	Input-Output (Parallelogram)	Shows an input or output instruction.
◇	Decision (Diamond)	Shows any point in the process where decision is to be made.

FLOWCHART

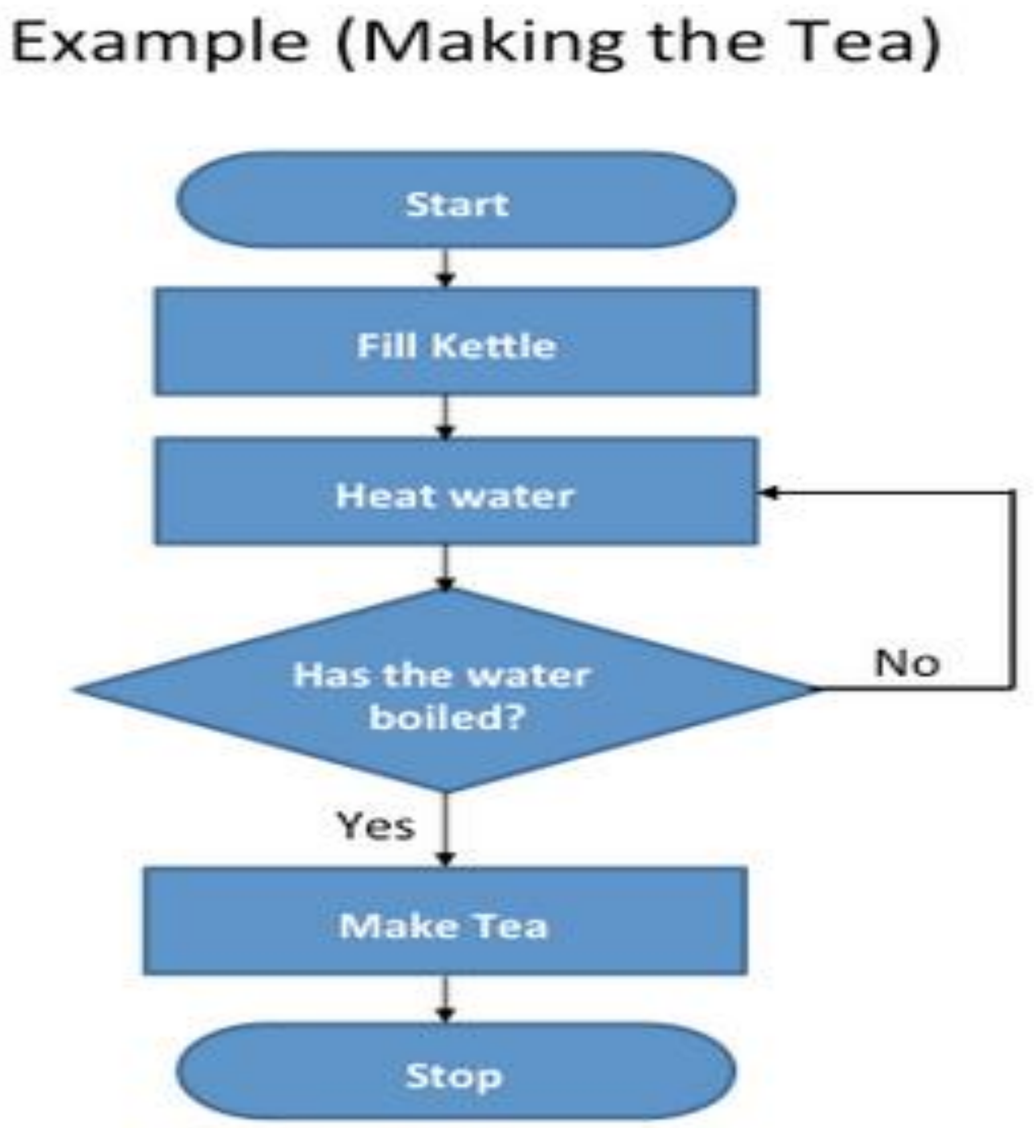


Symbol	Name	Meaning
○	**On-Page Connector**	Continues and connects the flowchart on the same page.
⬠	**Off-Page Connector**	Continues and connects the flowchart on another page.

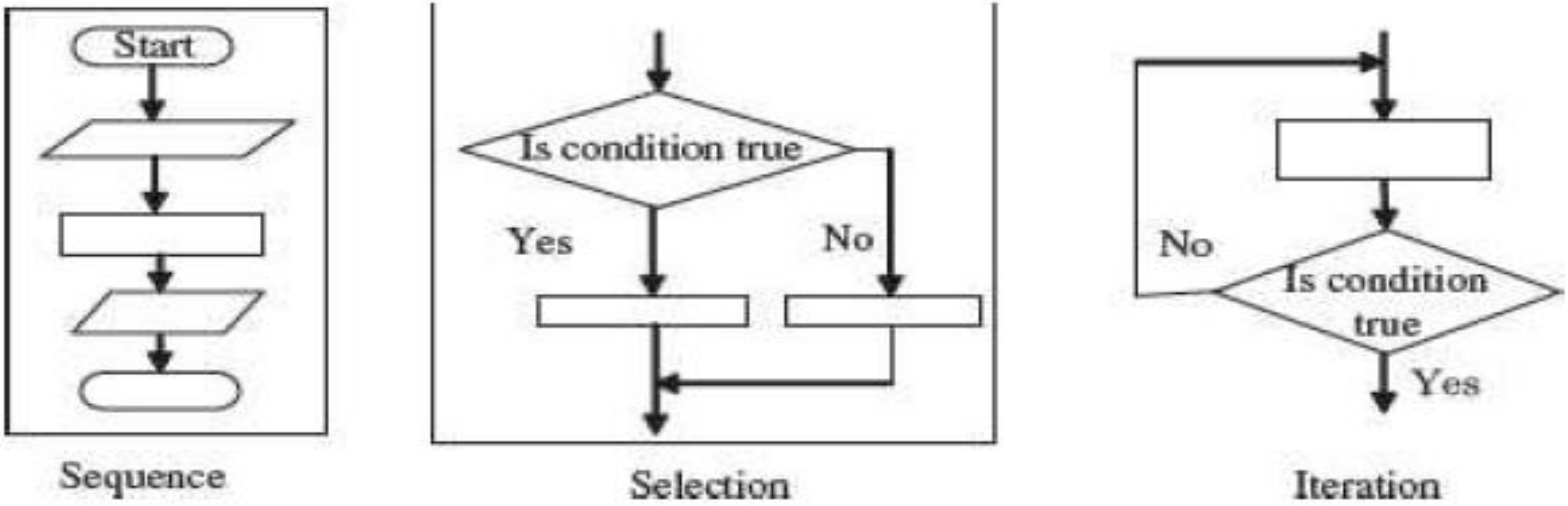


▭	**Sub-Process**	Denotes movement to another process and hence a new flowchart
⇄ ⇅	**Arrows**	Connects symbols and shows flow of the process or algorithm.

FLOWCHART

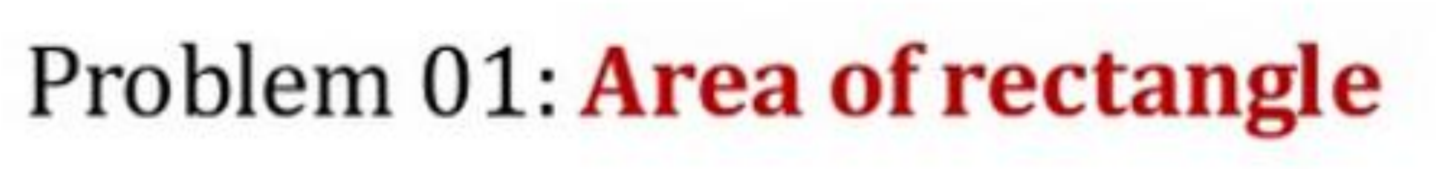

Example (Making the Tea)
Start
Fill Kettle
Heat water
Has the water boiled?
No
Yes
Make Tea
Stop

FLOWCHART

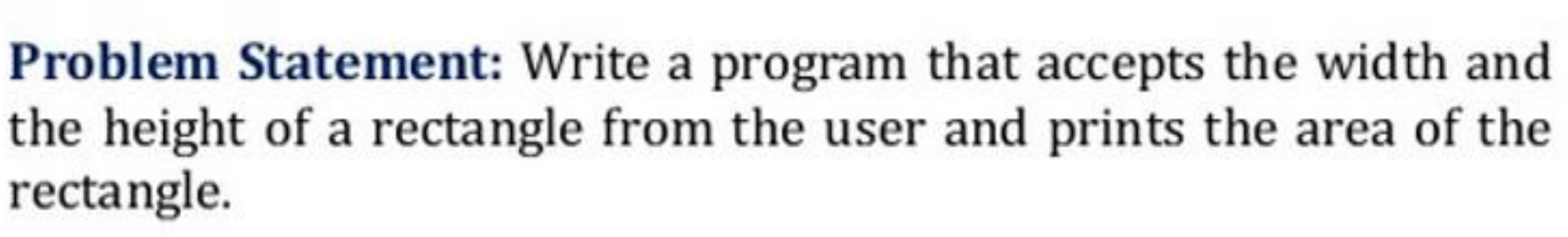


Control structures in flowchart.

FLOWCHART

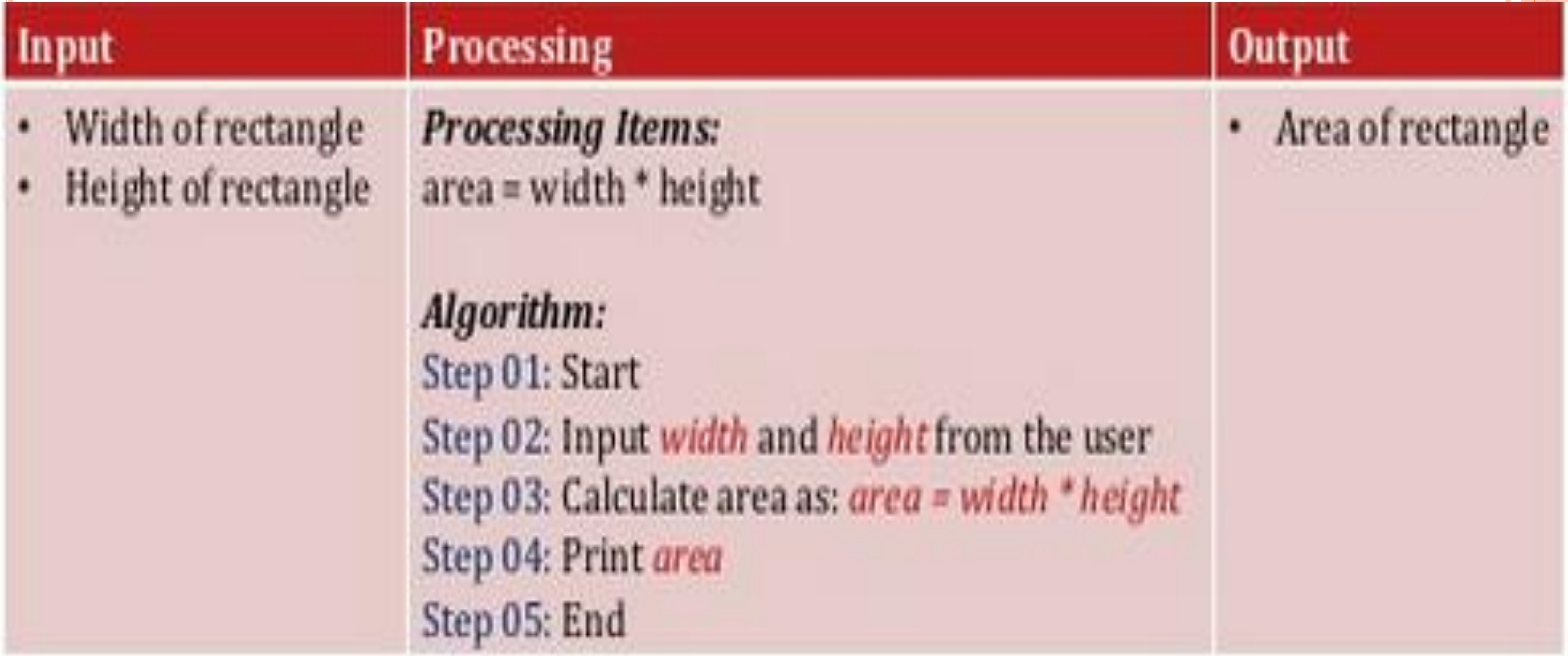


Problem 01: **Area of rectangle**



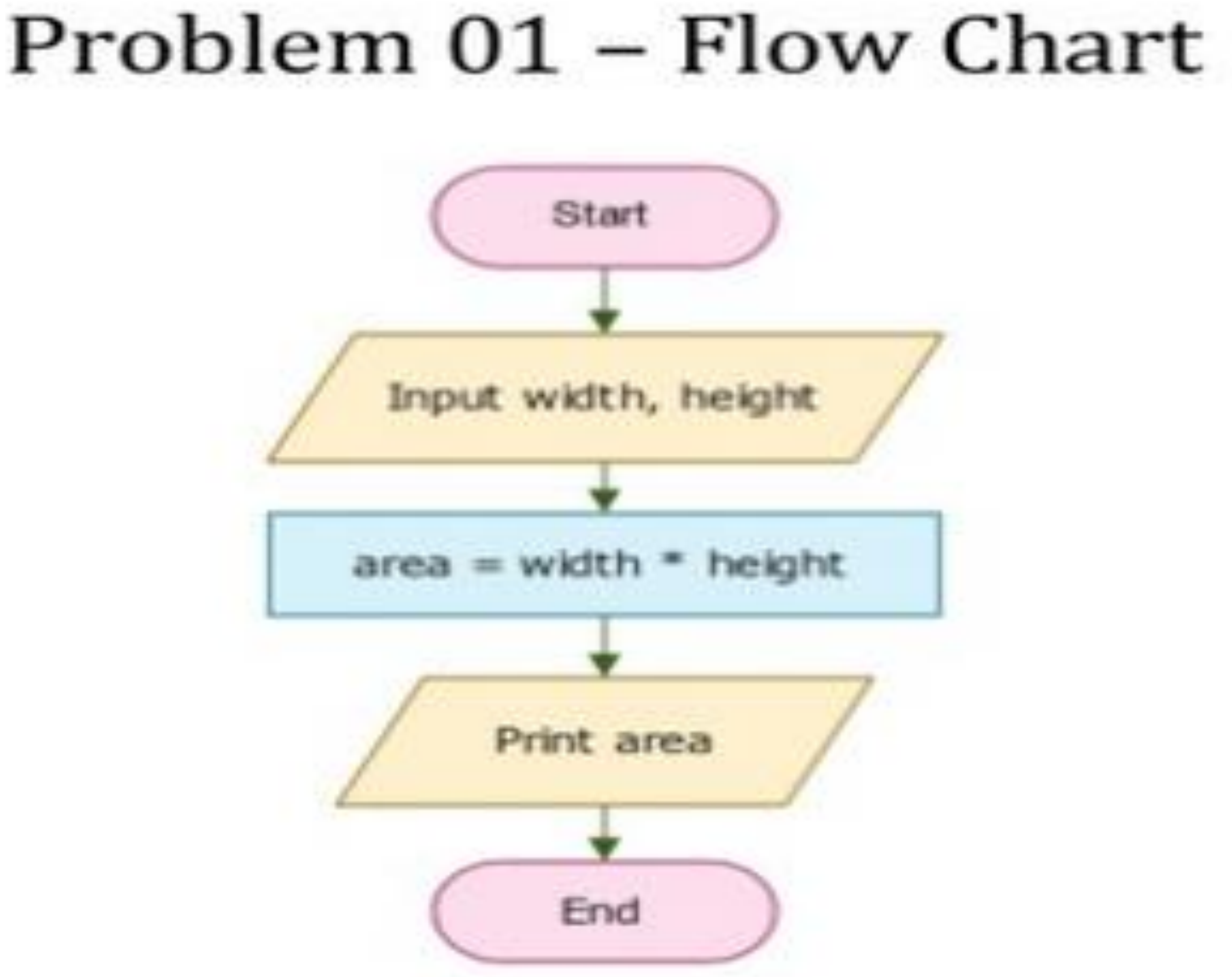
Problem Statement: Write a program that accepts the width and the height of a rectangle from the user and prints the area of the rectangle.

FLOWCHART



Input	Processing	Output
• Width of rectangle • Height of rectangle	***Processing Items:*** area = width * height ***Algorithm:*** Step 01: Start Step 02: Input *width* and *height* from the user Step 03: Calculate area as: *area = width * height* Step 04: Print *area* Step 05: End	• Area of rectangle

FLOWCHART



Problem 01 – Flow Chart

Start

Input width, height

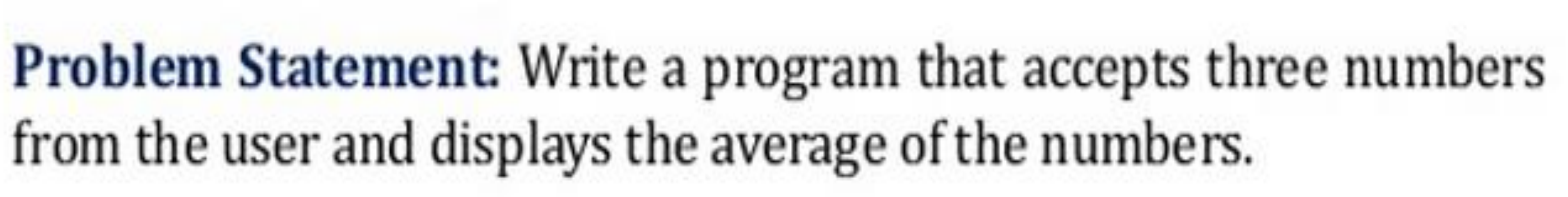
area = width * height

Print area

End

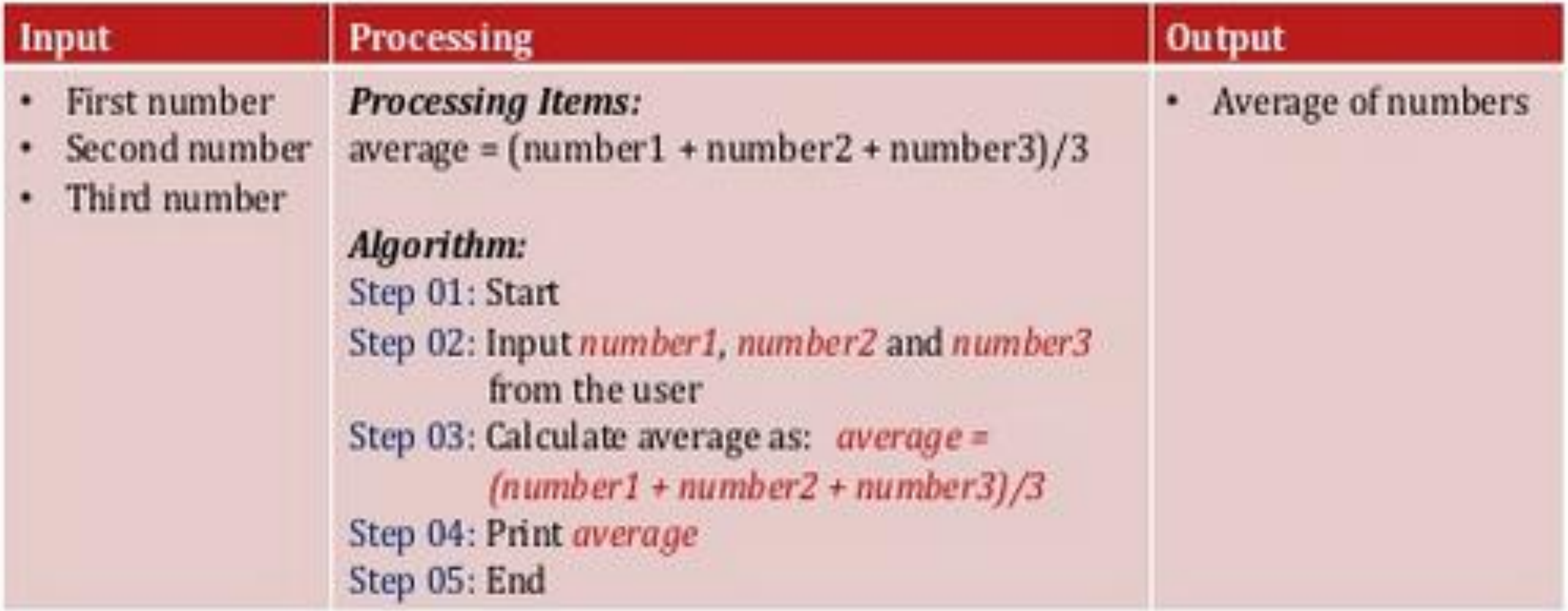
FLOWCHART

Problem 02 : **Average of three numbers**



Problem Statement: Write a program that accepts three numbers from the user and displays the average of the numbers.

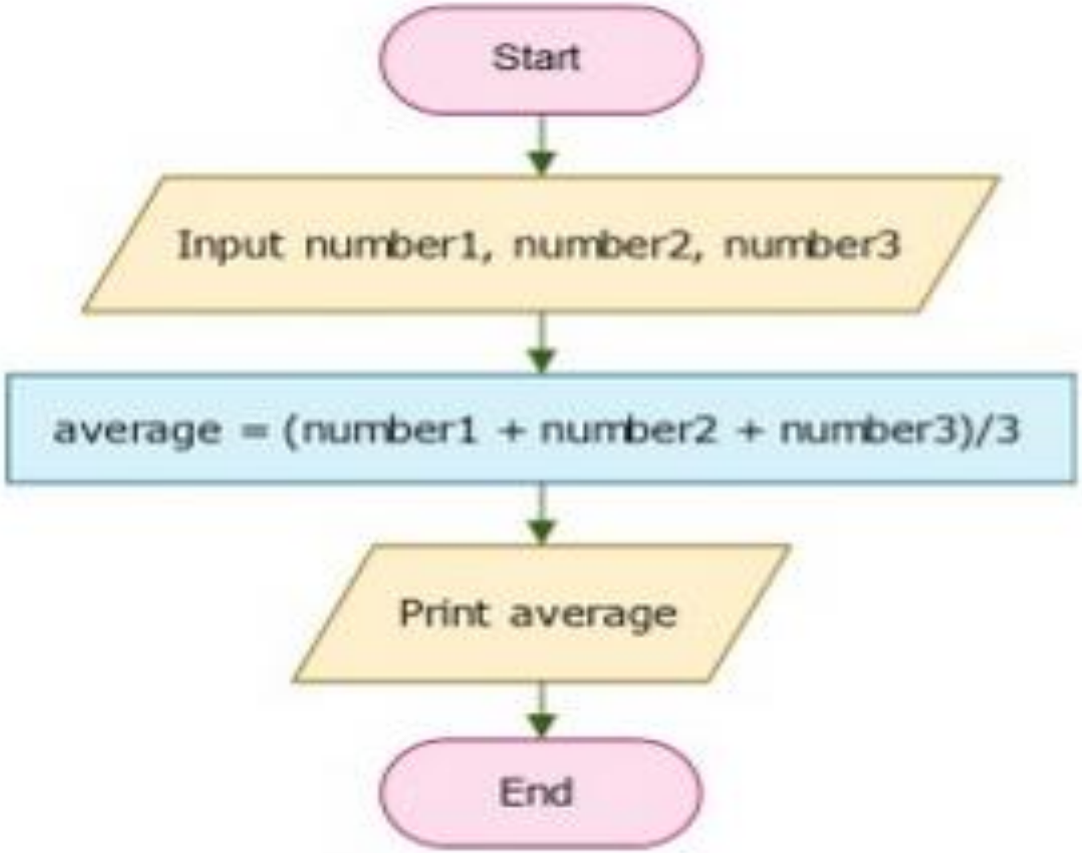
FLOWCHART



Input	Processing	Output
• First number • Second number • Third number	***Processing Items:*** average = (number1 + number2 + number3)/3 ***Algorithm:*** Step 01: Start Step 02: Input *number1*, *number2* and *number3* from the user Step 03: Calculate average as: *average = (number1 + number2 + number3)/3* Step 04: Print *average* Step 05: End	• Average of numbers

FLOWCHART

Problem 02 – Flow Chart



Start

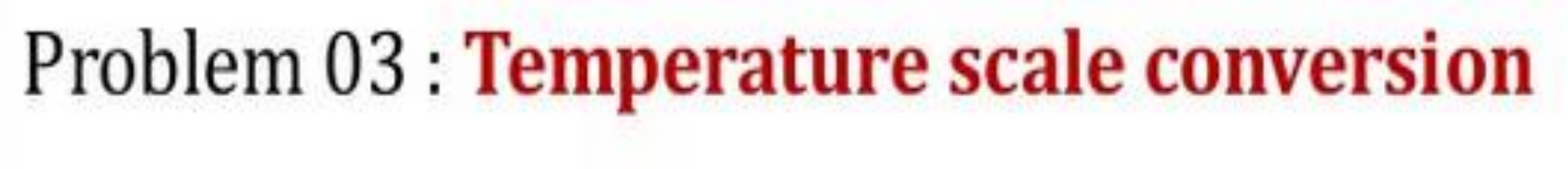
Input number1, number2, number3

average = (number1 + number2 + number3)/3

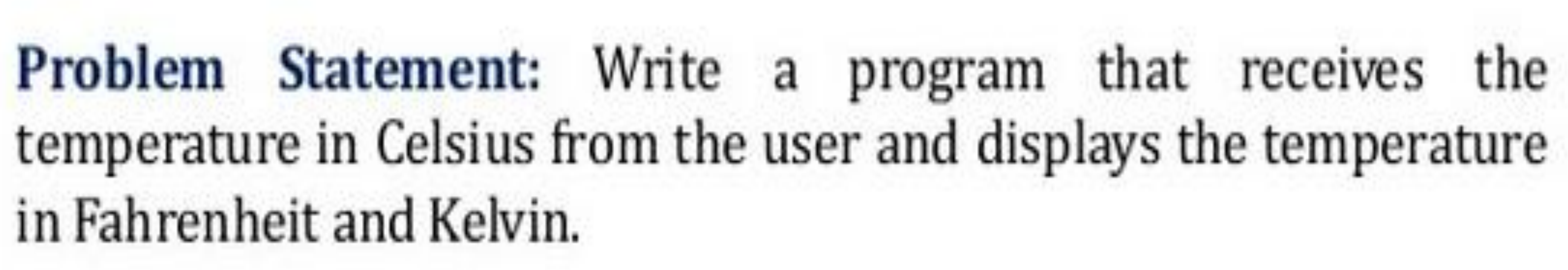
Print average

End

FLOWCHART

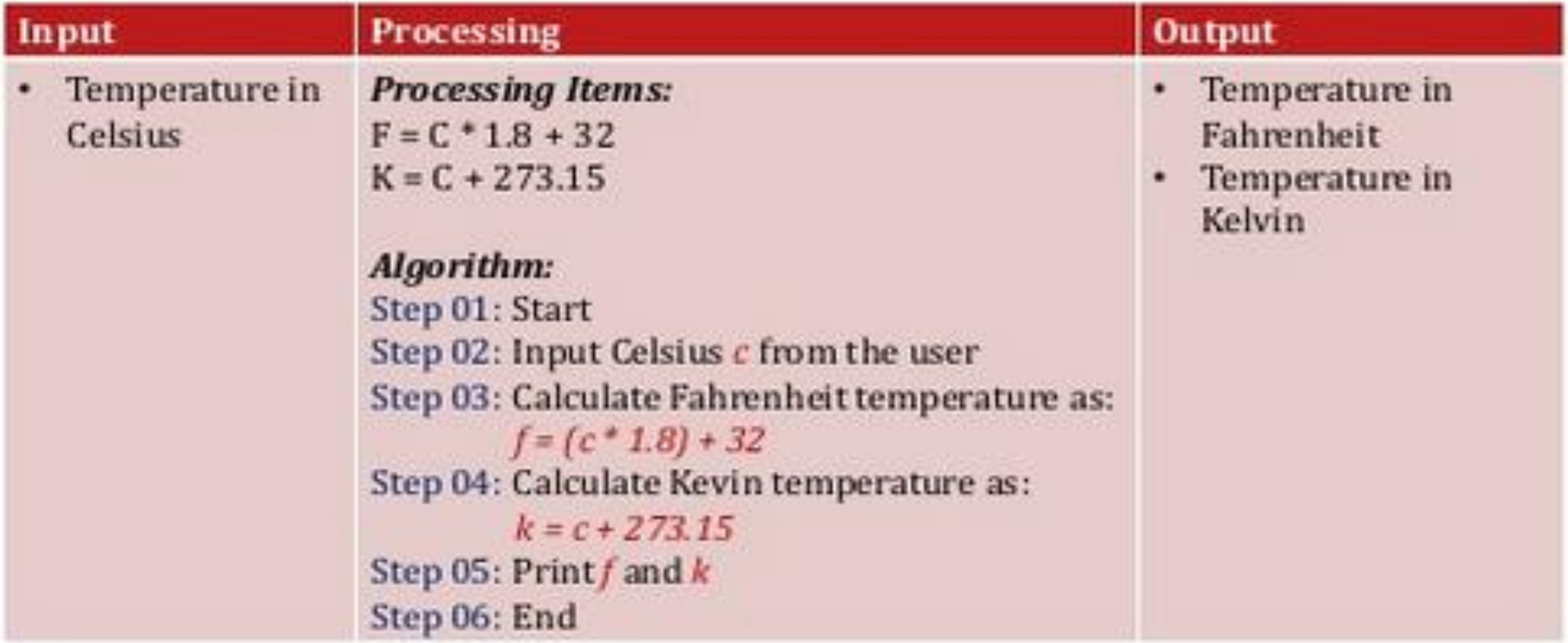


Problem 03 : **Temperature scale conversion**



Problem Statement: Write a program that receives the temperature in Celsius from the user and displays the temperature in Fahrenheit and Kelvin.

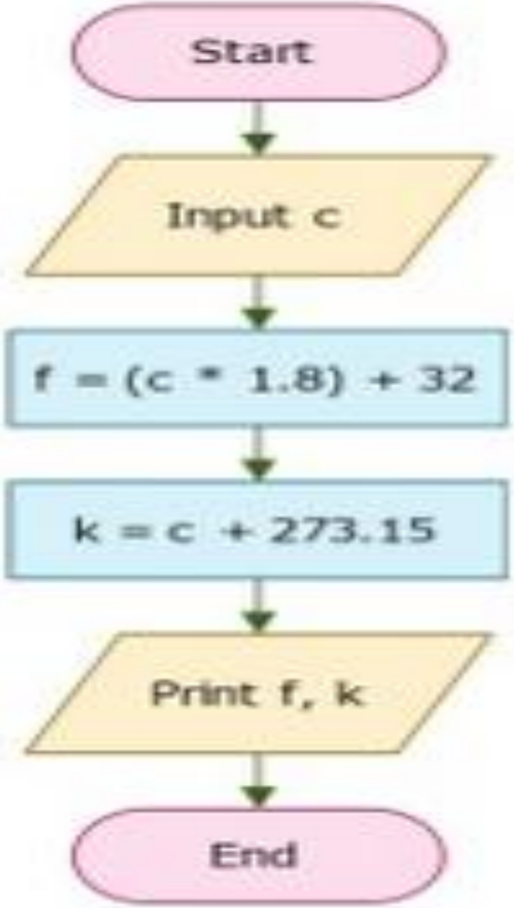
FLOWCHART



Input	Processing	Output
- Temperature in Celsius	***Processing Items:*** F = C * 1.8 + 32 K = C + 273.15 ***Algorithm:*** Step 01: Start Step 02: Input Celsius *c* from the user Step 03: Calculate Fahrenheit temperature as: $f = (c * 1.8) + 32$ Step 04: Calculate Kevin temperature as: $k = c + 273.15$ Step 05: Print *f* and *k* Step 06: End	- Temperature in Fahrenheit - Temperature in Kelvin

FLOWCHART

Problem 03 – Flow Chart



Start

Input c

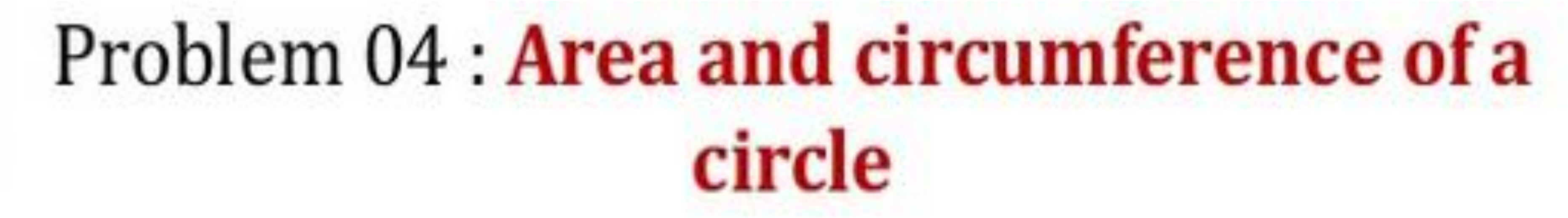
f = (c * 1.8) + 32

k = c + 273.15

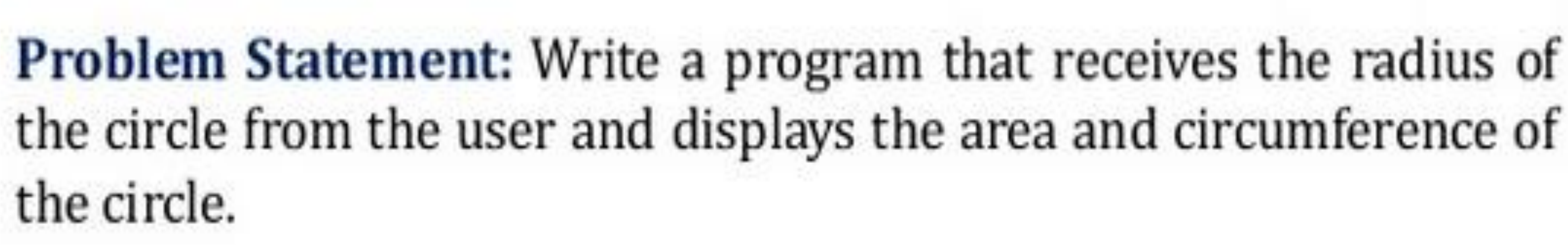
Print f, k

End

FLOWCHART

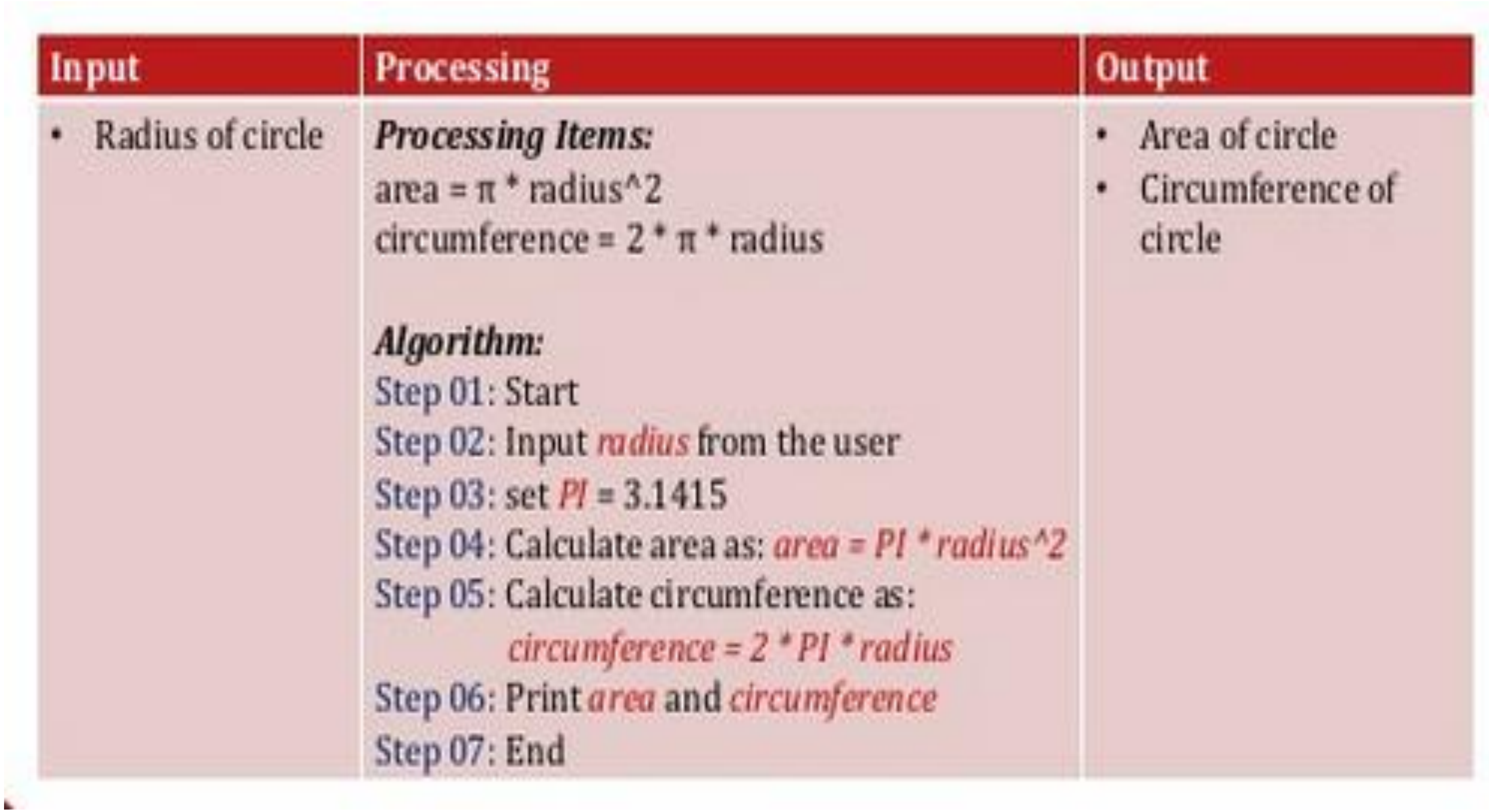


Problem 04 : **Area and circumference of a circle**



Problem Statement: Write a program that receives the radius of the circle from the user and displays the area and circumference of the circle.

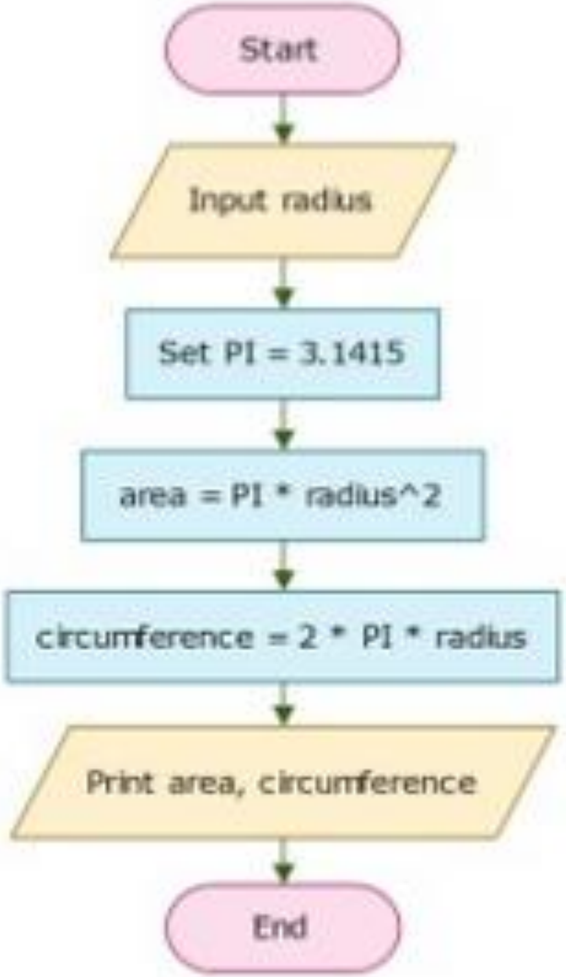
FLOWCHART



Input	Processing	Output
• Radius of circle	***Processing Items:*** area = π * radius^2 circumference = 2 * π * radius ***Algorithm:*** Step 01: Start Step 02: Input *radius* from the user Step 03: set *PI* = 3.1415 Step 04: Calculate area as: *area = PI * radius^2* Step 05: Calculate circumference as: *circumference = 2 * PI * radius* Step 06: Print *area* and *circumference* Step 07: End	• Area of circle • Circumference of circle

FLOWCHART

Problem 04 – Flow Chart



Start

Input radius

Set PI = 3.1415

area = PI * radius^2

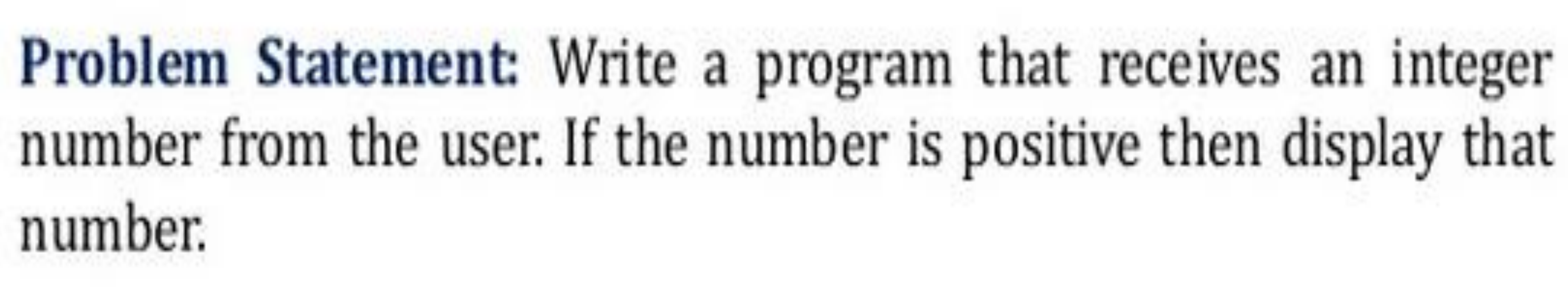
circumference = 2 * PI * radius

Print area, circumference

End

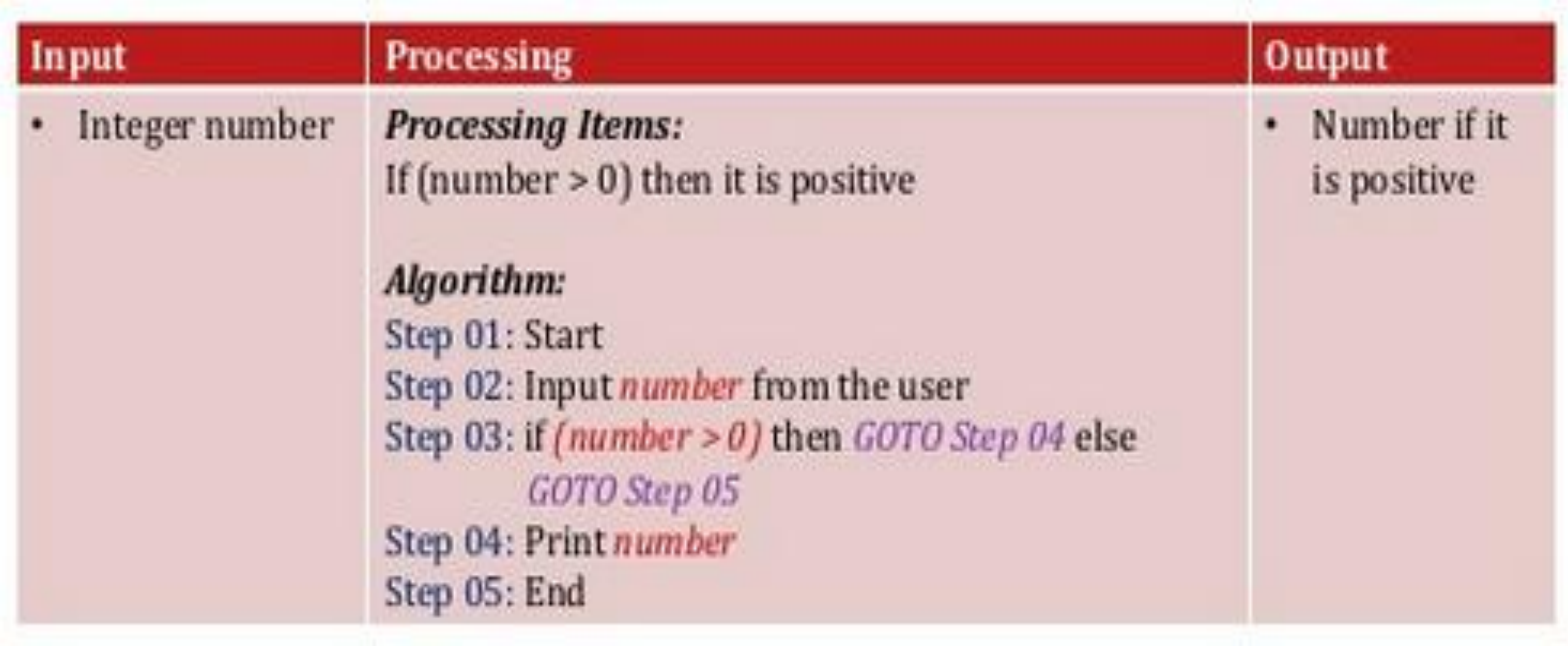
FLOWCHART

Problem 08 : **Positive numbers**



Problem Statement: Write a program that receives an integer number from the user. If the number is positive then display that number.

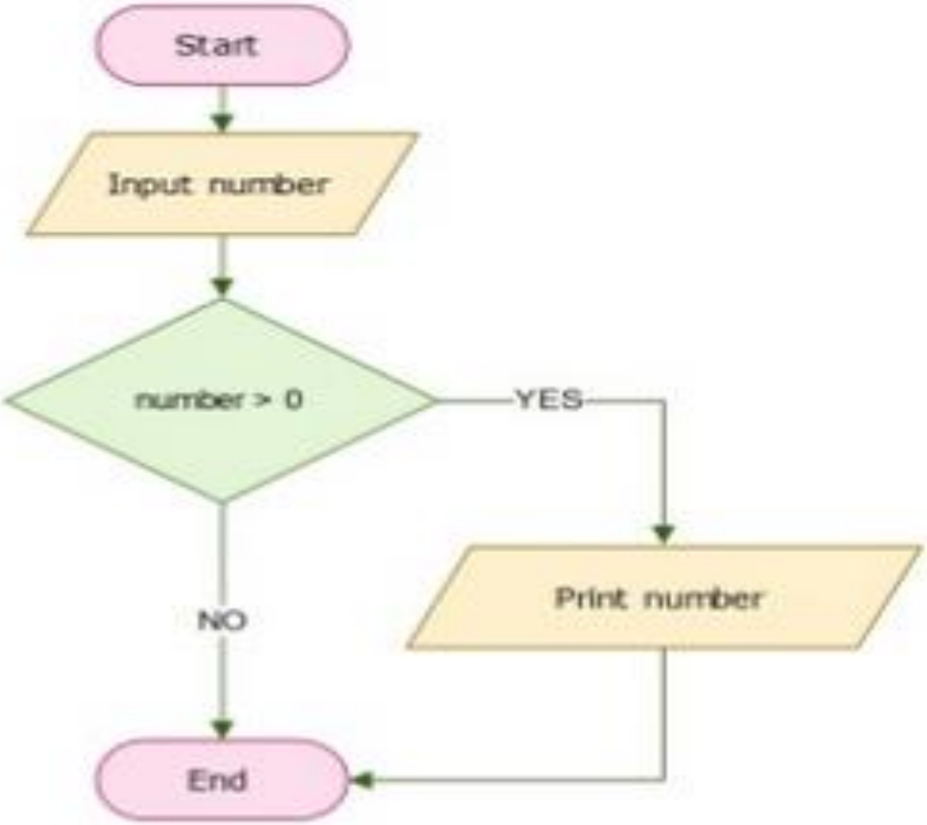
FLOWCHART



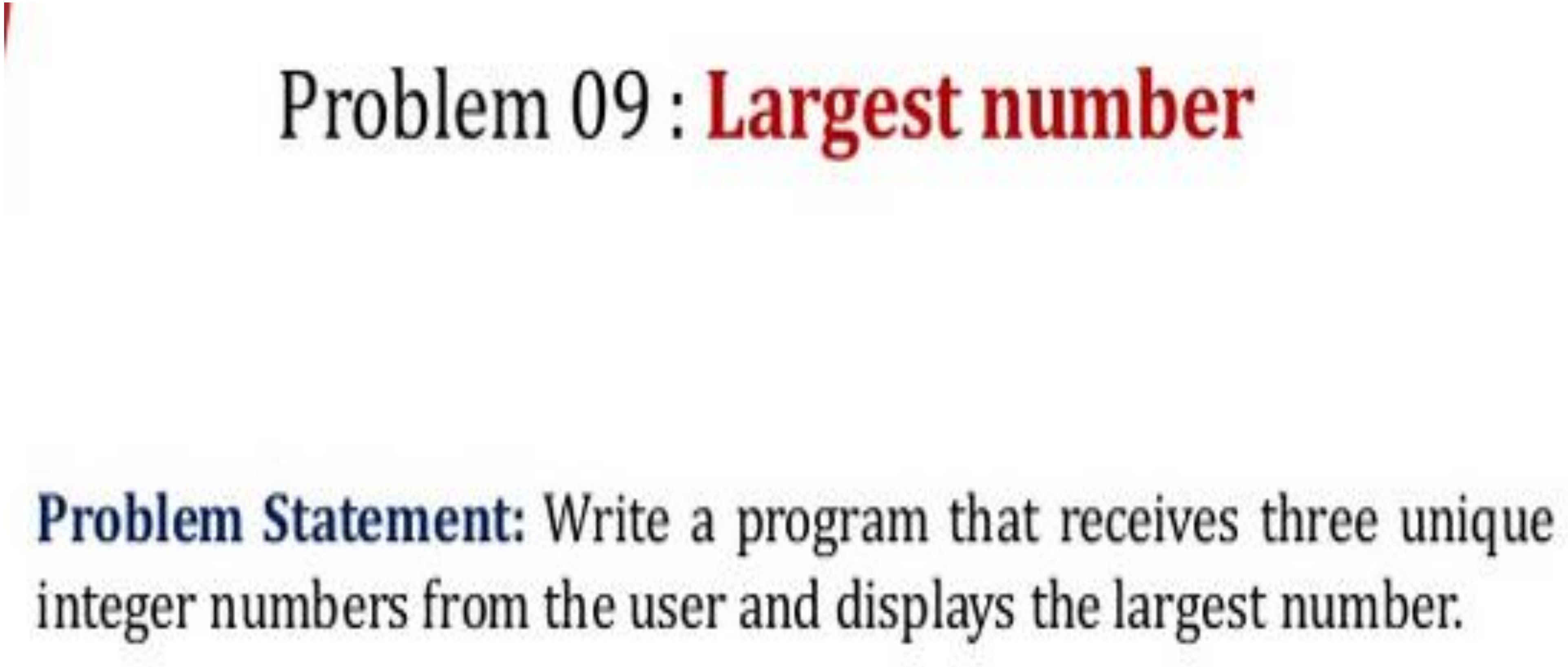
Input	Processing	Output
- Integer number	***Processing Items:*** If (number > 0) then it is positive ***Algorithm:*** Step 01: Start Step 02: Input *number* from the user Step 03: if *(number > 0)* then *GOTO Step 04* else *GOTO Step 05* Step 04: Print *number* Step 05: End	- Number if it is positive

FLOWCHART

Problem 08 – Flow Chart



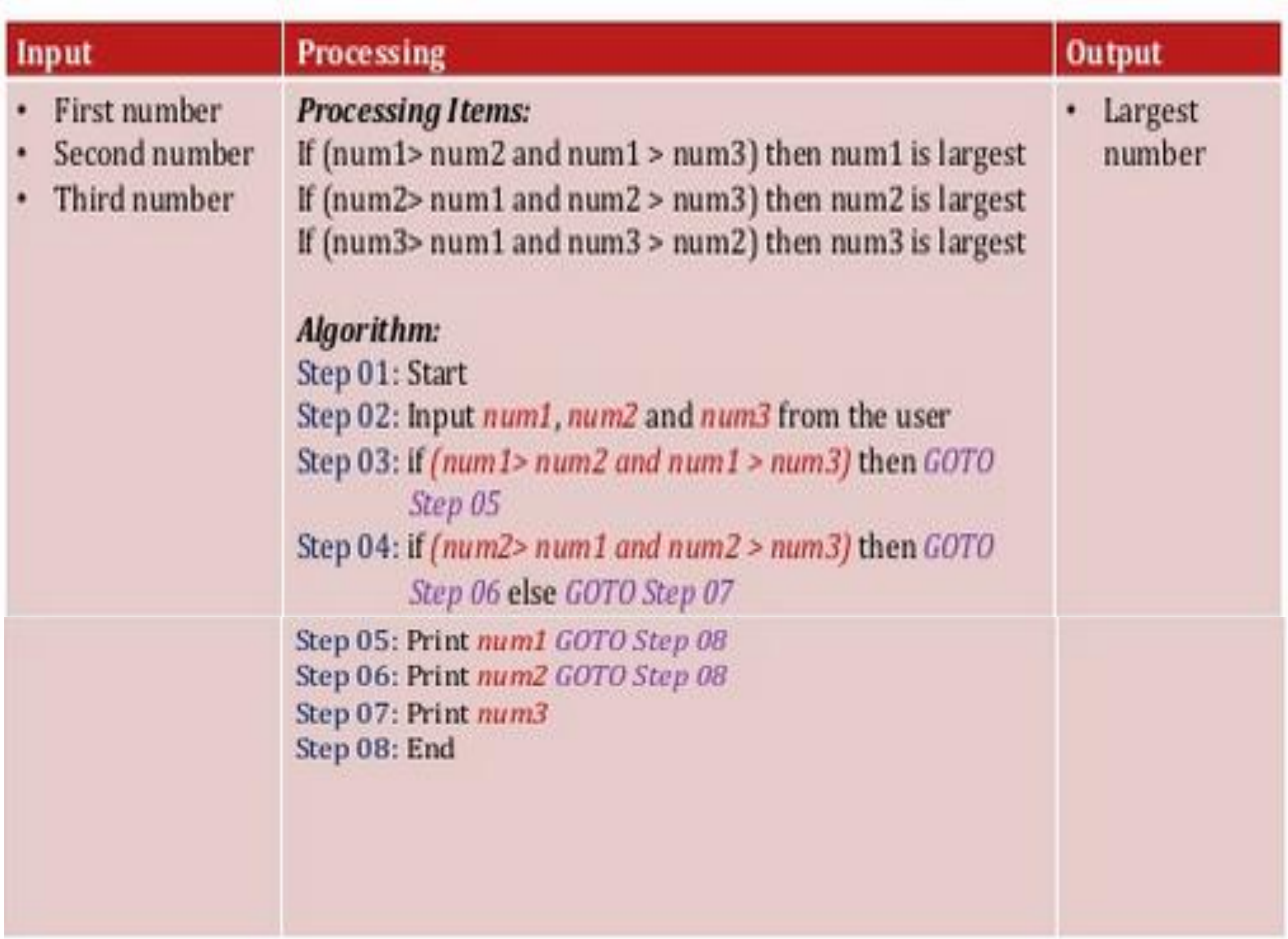
FLOWCHART



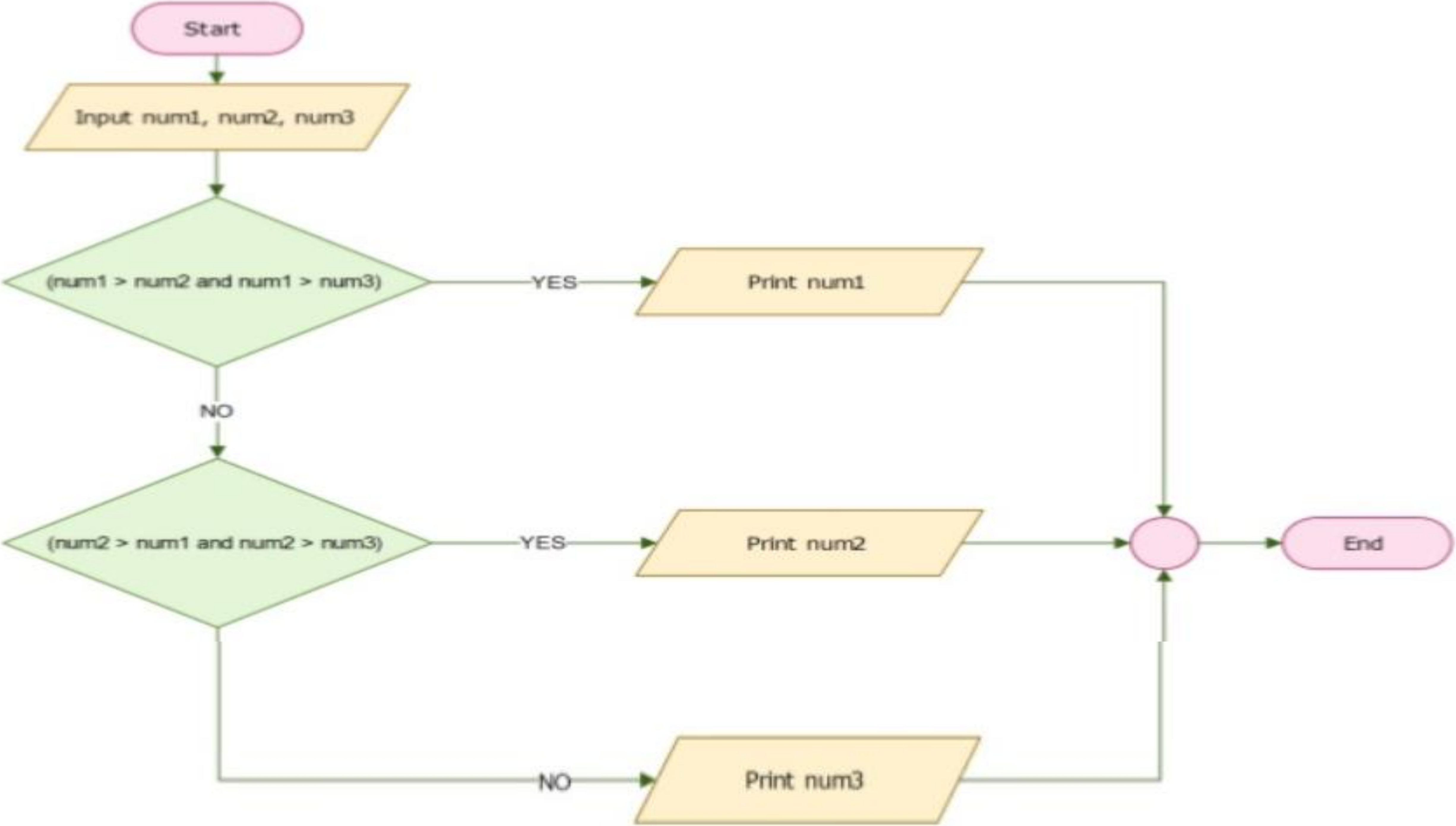
Problem 09 : Largest number

Problem Statement: Write a program that receives three unique integer numbers from the user and displays the largest number.

FLOWCHART

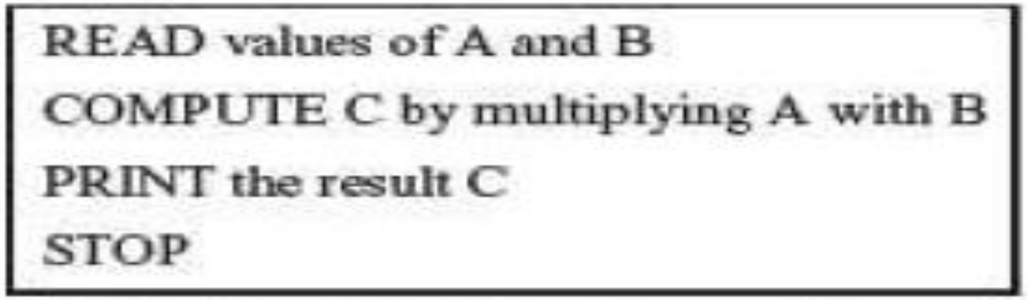


Input	Processing	Output
• First number • Second number • Third number	***Processing Items:*** If (num1> num2 and num1 > num3) then num1 is largest If (num2> num1 and num2 > num3) then num2 is largest If (num3> num1 and num3 > num2) then num3 is largest ***Algorithm:*** Step 01: Start Step 02: Input *num1*, *num2* and *num3* from the user Step 03: if *(num1> num2 and num1 > num3)* then *GOTO Step 05* Step 04: if *(num2> num1 and num2 > num3)* then *GOTO Step 06* else *GOTO Step 07*	• Largest number
	Step 05: Print *num1 GOTO Step 08* Step 06: Print *num2 GOTO Step 08* Step 07: Print *num3* Step 08: End	


Start
Input num1, num2, num3
(num1 > num2 and num1 > num3)
YES
Print num1
NO
(num2 > num1 and num2 > num3)
YES
Print num2
NO
Print num3
End

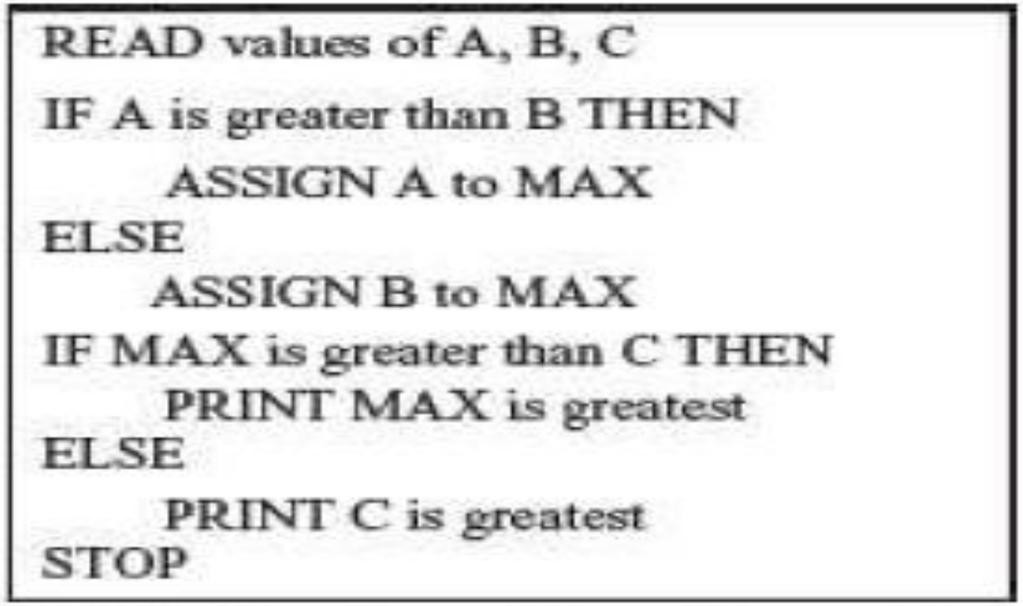
PSEUDO CODE

- Pseudo code consists of short, readable and formally-styled English language used for explaining an algorithm. Pseudo code does not include details like variable declarations, subroutines etc.
- Pseudo code is a short-hand way of describing a computer program.
- It is used to give a sketch of the structure of the program, before the actual coding. It uses the structured constructs of the programming language but is not machine-readable. Pseudo code cannot be compiled or executed. Thus, no standard for the syntax of pseudo code exists. For writing the pseudo code, the programmer is not required to know the programming language in which the pseudo code will be implemented later.



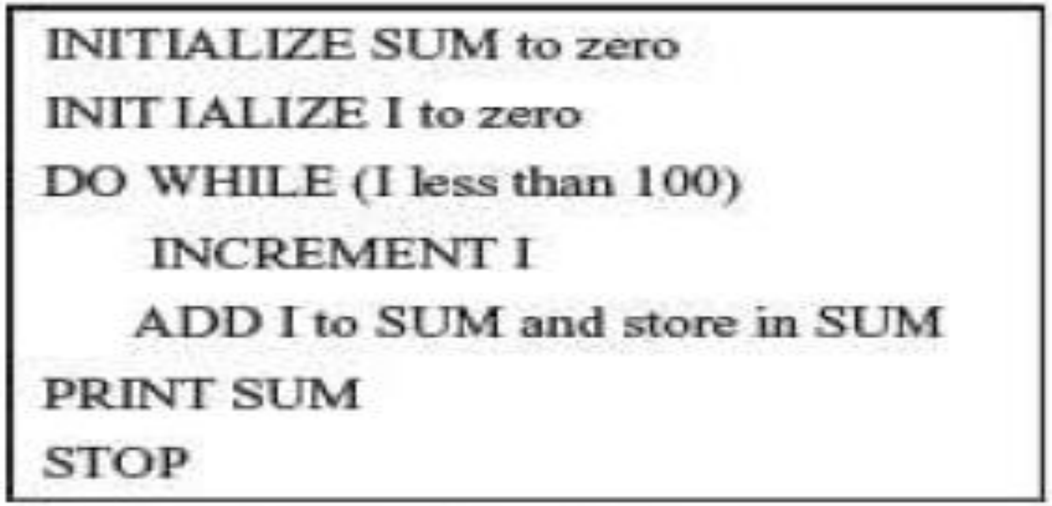
```
READ values of A and B
COMPUTE C by multiplying A with B
PRINT the result C
STOP
```

(i) Find product of any two numbers



```
READ values of A, B, C
IF A is greater than B THEN
        ASSIGN A to MAX
ELSE
       ASSIGN B to MAX
IF MAX is greater than C THEN
        PRINT MAX is greatest
ELSE
        PRINT C is greatest
STOP
```

(ii) Find maximum of any three numbers



```
INITIALIZE SUM to zero
INIT IALIZE I to zero
DO WHILE (I less than 100)
       INCREMENT I
      ADD I to SUM and store in SUM
PRINT SUM
STOP
```

(iii) Find sum of first 100 integers

PSEUDO CODE

- A pseudo code is easily translated into a programming language. But, as there are no defined standards for writing a pseudo code, programmers may use their own style for writing the pseudo code, which can be easily understood. Generally, programmers prefer to write pseudo code instead of flowcharts.

DIFFERENCE BETWEEN ALGORITHM, FLOWCHART, AND PSEUDO CODE

- An algorithm is a sequence of instructions used to solve a particular problem. Flowchart and Pseudo code are tools to document and represent the algorithm. In other words, an algorithm can be represented using a flowchart or a pseudo code. Flowchart is a graphical representation of the algorithm. Pseudo code is a readable, formally styled English like language representation of the algorithm.

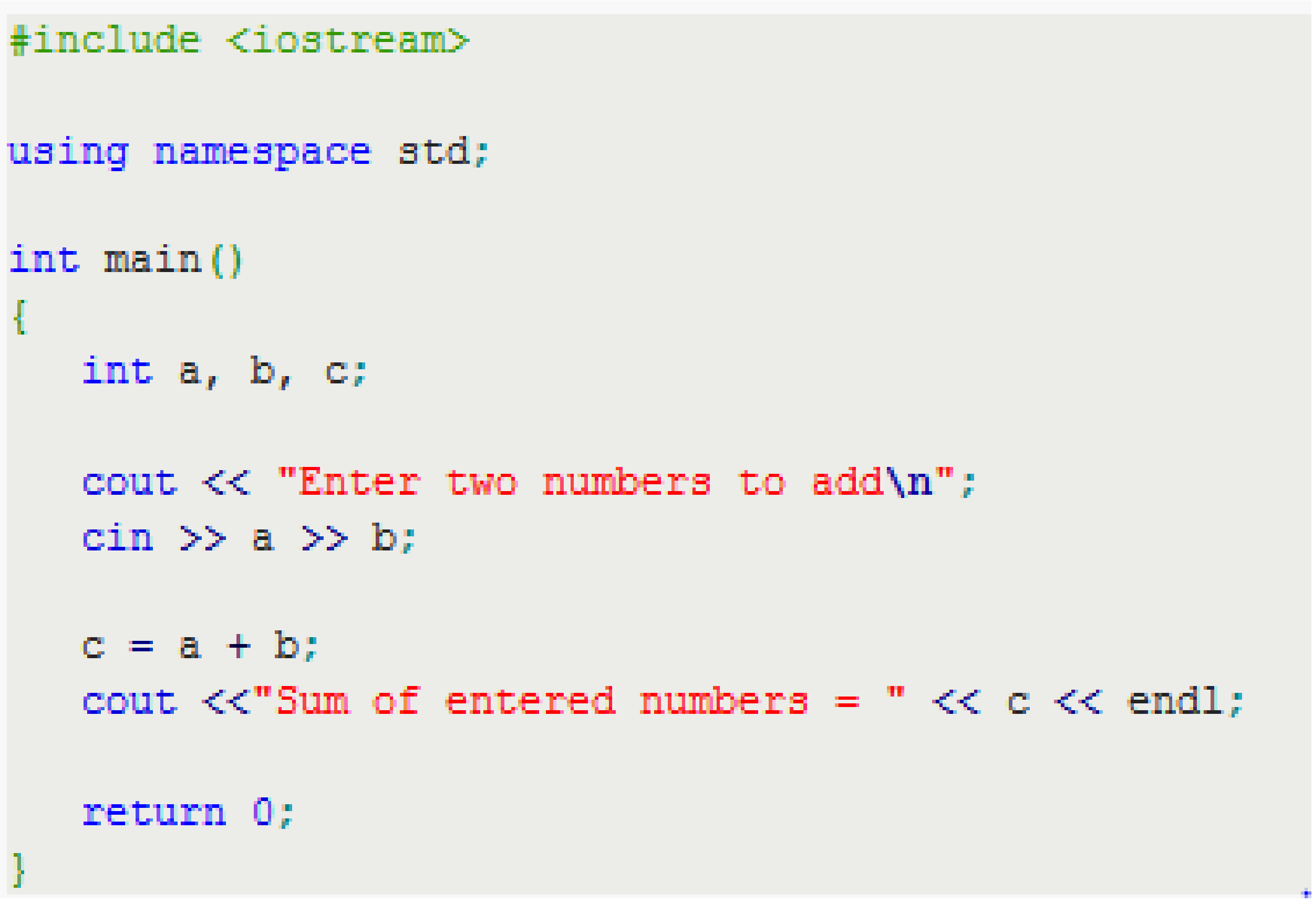
Programming Paradigms

- There are three types of Programming Paradigms or styles/patterns of programming:
- 1- Procedural Programming
- 2- Functional Programming
- 3- Object Oriented Programming

Programming Paradigms

- **Procedural programming** uses a list of instructions to tell the computer what to do step by step.
- Procedural programming is also referred to as imperative or structured programming.
- Examples of procedural languages that use the style of procedural programming include Fortran, COBOL and C, which have been around since the 1960s and 70s.

PROGRAMMING PARADIGMS



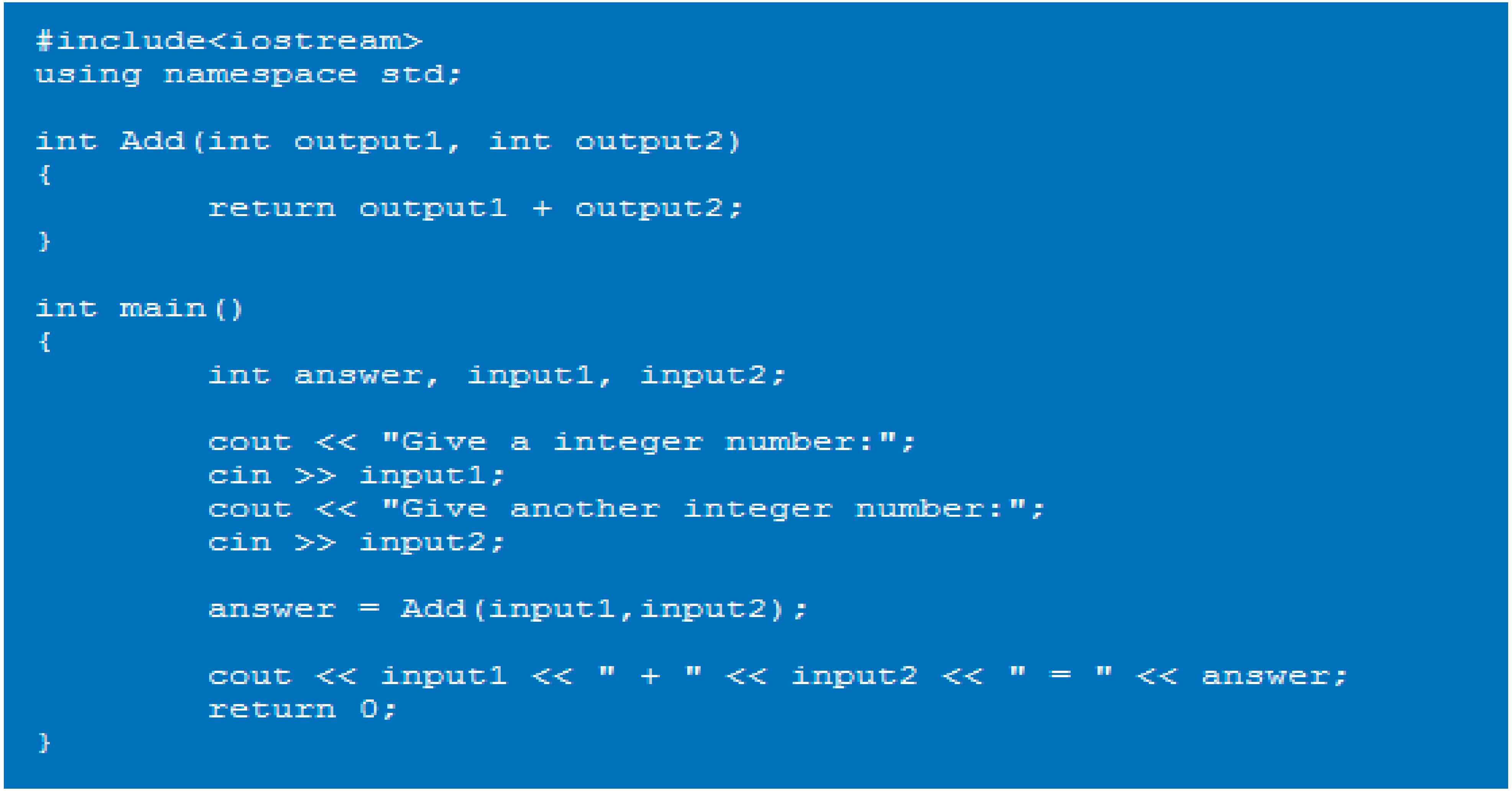
```
#include <iostream>

using namespace std;

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

    cout << "Enter two numbers to add\n";
    cin >> a >> b;

    c = a + b;
    cout <<"Sum of entered numbers = " << c << endl;

    return 0;
}
```

EXAMPLE OF PROCEDURAL PROGRAMMING

Programming Paradigms

- **Functional programming** is an approach to problem solving that treats every computation as a mathematical function.
- It is when functions are used as the fundamental building blocks of a program..
- Examples of functional programming languages include Erlang, Haskell, Lisp and Scala. .

PROGRAMMING PARADIGMS- EXAMPLE



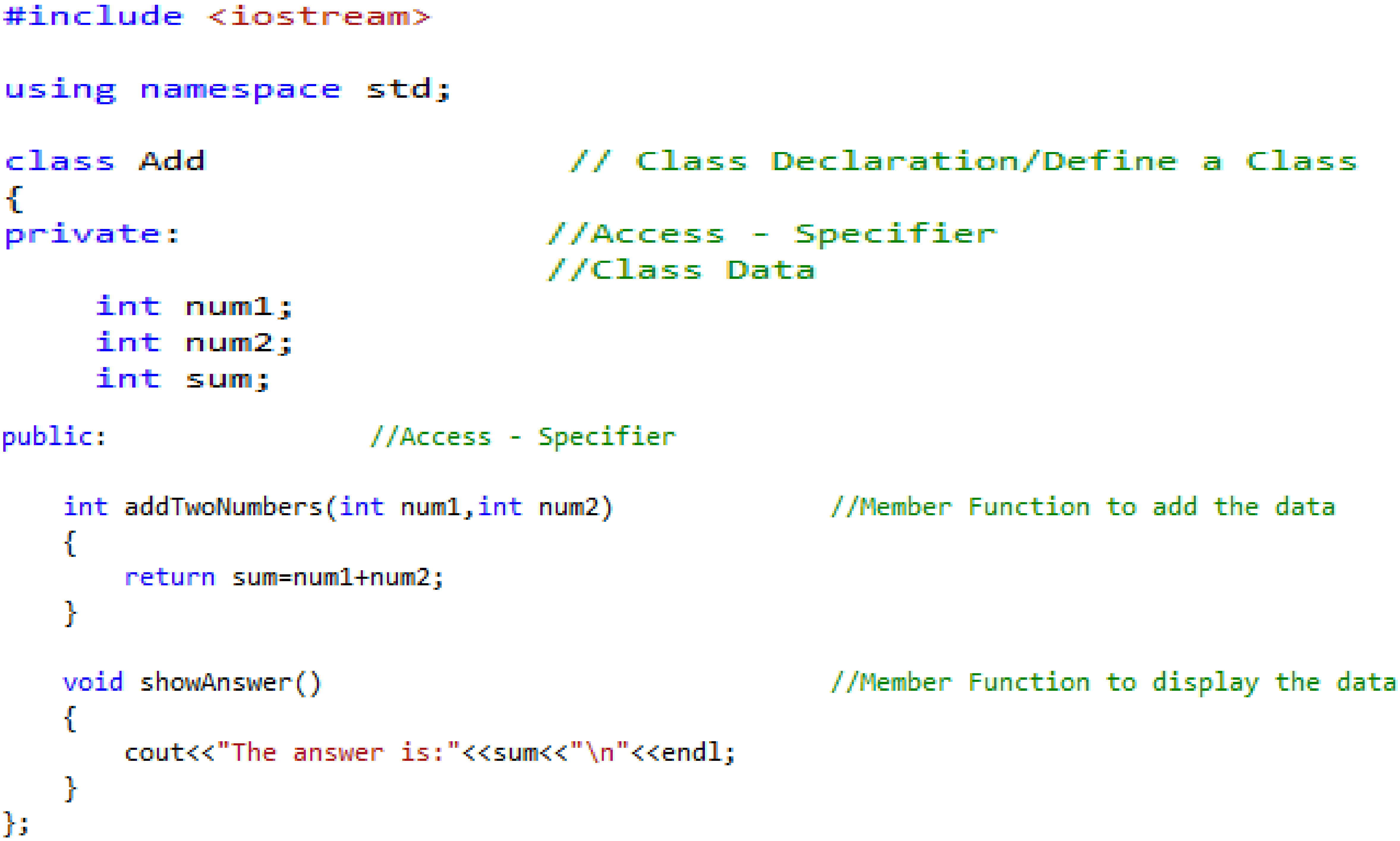
```
#include<iostream>
using namespace std;

int Add(int output1, int output2)
{
        return output1 + output2;
}

int main()
{
        int answer, input1, input2;

        cout << "Give a integer number:";
        cin >> input1;
        cout << "Give another integer number:";
        cin >> input2;

        answer = Add(input1,input2);

        cout << input1 << " + " << input2 << " = " << answer;
        return 0;
}
```

Programming Paradigms

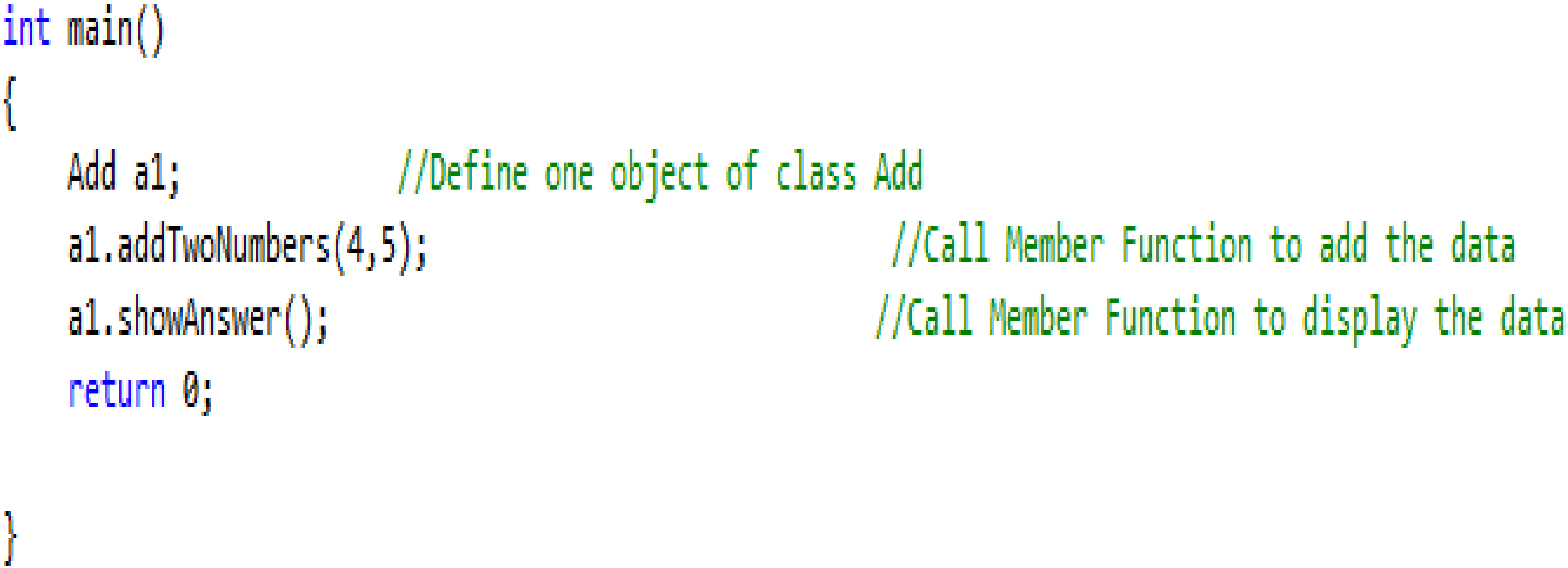
- **Object Oriented Programming: The** fundamental idea behind object-oriented languages is to combine into a single unit both data and the functions that operate on that data.
- A computer language is object-oriented if they support the four specific object properties called *abstraction*, *polymorphism*, *inheritance*, and *encapsulation*.

Programming Paradigms

- *Abstraction* allows dealing with the complexity of the object. Abstraction allows picking out the relevant details of the object, and ignoring the non-essential details.
- *Encapsulation* means information hiding.
- *Polymorphism* means, many forms. It refers to an entity changing its form depending on the circumstances.
- The *Inheritance* feature of object-oriented software allows a new class, called the derived class, to be derived from an already existing class known as the base class.



```
#include <iostream>

using namespace std;

class Add                   // Class Declaration/Define a Class
{
private:                   //Access - Specifier
                           //Class Data
    int num1;
    int num2;
    int sum;

public:               //Access - Specifier

    int addTwoNumbers(int num1,int num2)          //Member Function to add the data
    {
        return sum=num1+num2;
    }

    void showAnswer()                             //Member Function to display the data
    {
        cout<<"The answer is:"<<sum<<"\n"<<endl;
    }
};
```



```
int main()
{
    Add a1;             //Define one object of class Add
    a1.addTwoNumbers(4,5);                           //Call Member Function to add the data
    a1.showAnswer();                                //Call Member Function to display the data
    return 0;


}
```