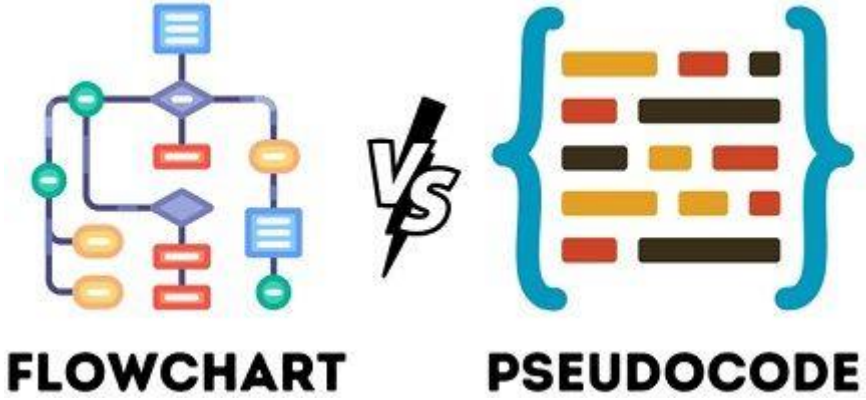
Flowchart

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.

Flowcharts are nothing but the graphical representation of the data or the algorithm for a better understanding of the code visually. It displays step-by-step solutions to a problem, algorithm, or process

Flowchart symbol :
Flowcharts use special shapes to represent different types of actions or steps in a process. Lines and arrows show the sequence of the steps, and the relationships among them. These are known as flowchart symbols.

Oval or Pill Shape : Represents the start or end
Parallelogram Shape : Represents input/output
Rectangle Shape : Represents a process
Diamond Shape : Represents a decision
Arrow : Represents relationship between representative shapes and flow



Common Flowchart Symbol

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectagle represents a process
	Decision	A diamond indicates a decision

Advantage & Disadvantage of Flowchart

Advantages of Flowchart

- It is the most efficient way of communicating the logic of the system.
- It acts as a guide for a blueprint during the program design.
- It also helps in the debugging process.
- Using flowcharts, we can easily analyze the programs.
- Flowcharts are good for documentation.

Disadvantages of Flowchart

- Flowcharts are challenging to draw for large and complex programs.
- It does not contain the proper amount of details.
- Flowcharts are very difficult to reproduce.
- Flowcharts are very difficult to modify.

Pseudocode

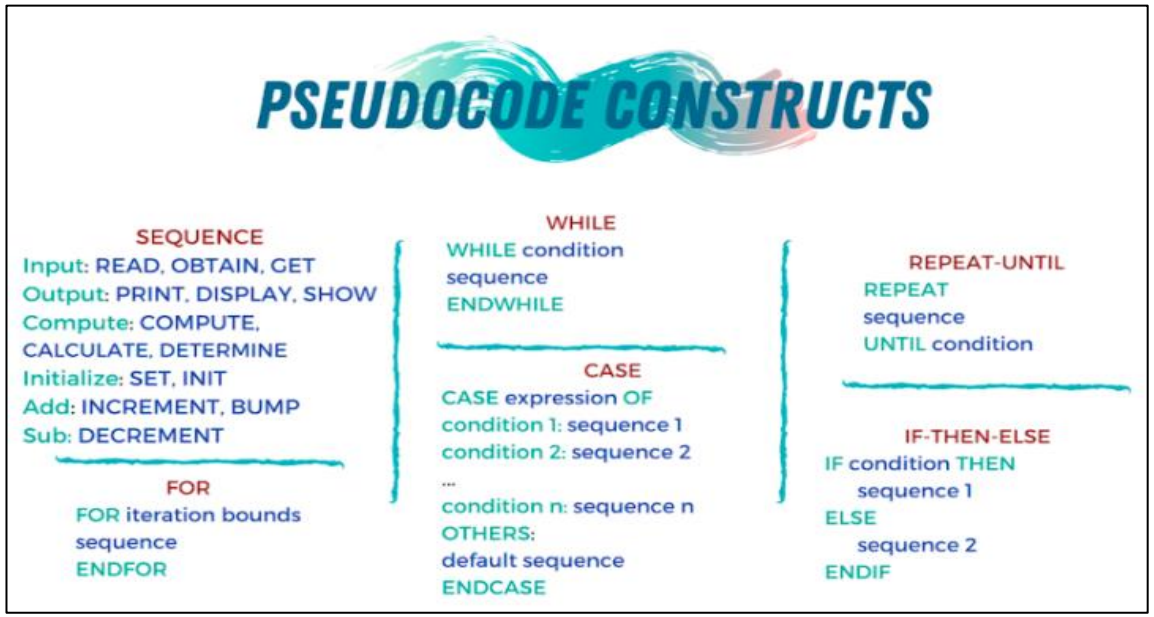
Pseudocode is a technique used to describe the distinct steps of an algorithm in a manner that’ s easy to understand for anyone with basic programming knowledge.

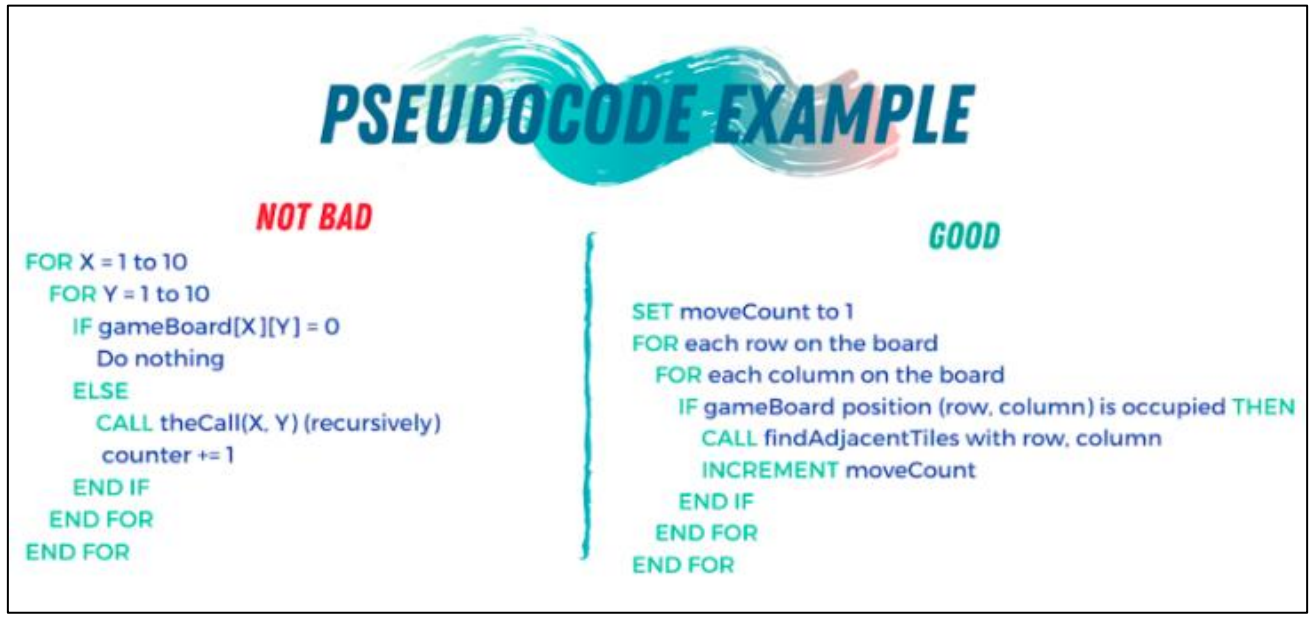
THE MAIN CONSTRUCTS OF PSEUDOCODE

At its core pseudocode is the ability to represent six programming constructs (always written in uppercase): SEQUENCE, CASE, WHILE, REPEAT-UNTIL, FOR, and IF-THEN-ELSE. These constructs – also called keywords –are used to describe the control flow of the algorithm.

1. **SEQUENCE** represents linear tasks sequentially performed one after the other.
2. **WHILE** a loop with a condition at its beginning.
3. **REPEAT-UNTIL** a loop with a condition at the bottom.
4. **FOR** another way of looping.
5. **IF-THEN-ELSE** a conditional statement changing the flow of the algorithm.
6. **CASE** the generalization form of IF-THEN-ELSE.

Pseudocode

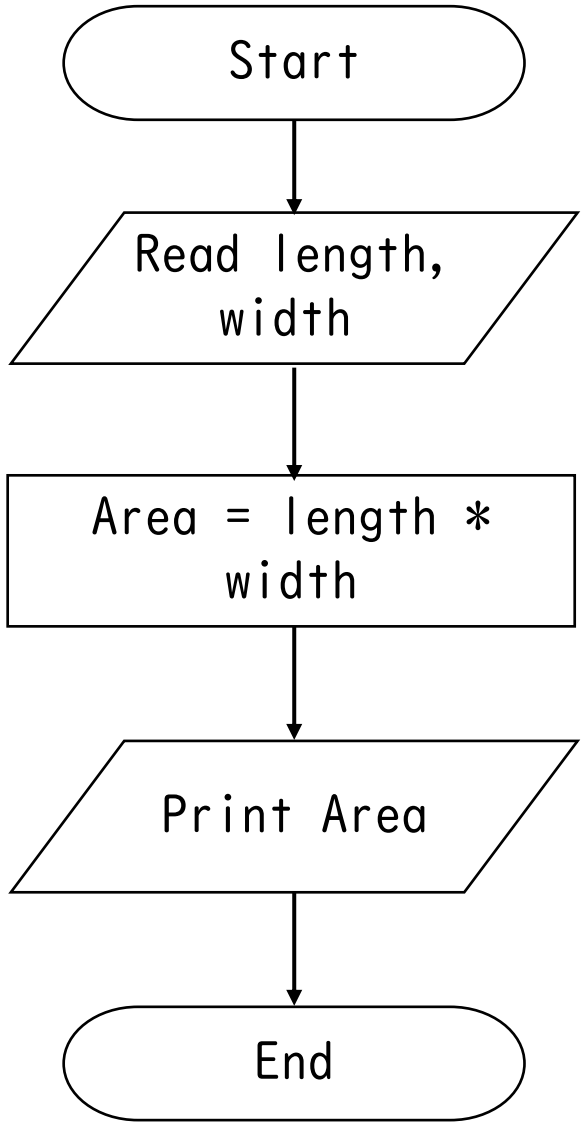




HOW TO WRITE PSEUDOCODE

1. Always capitalize the initial word (often one of the main six constructs).
2. Make only one statement per line.
3. Indent to show hierarchy, improve readability, and show nested constructs.
4. Always end multi-line sections using any of the END keywords (ENDIF, ENDWHILE, etc.).
5. Keep your statements programming language independent.
6. Use the naming domain of the problem, not that of the implementation.
   For instance: “Append the last name to the first name” instead of “name = first+ last.”
7. Keep it simple, concise and readable.

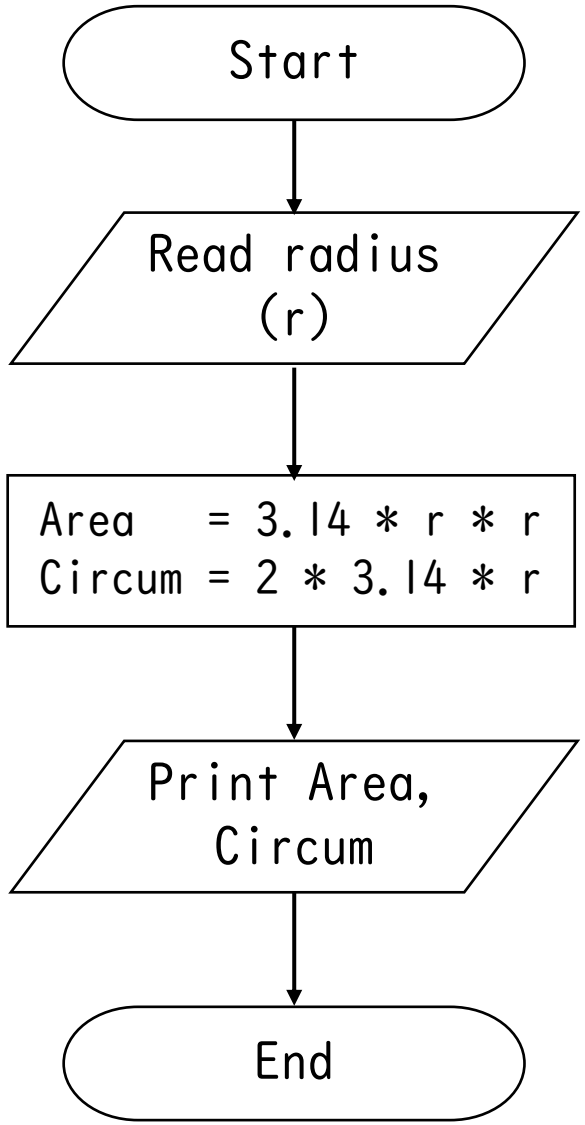
Write an algorithm to find area of a rectangle




```
Step 1 : Start
Step 2 : Get length, width values
Step 3 : Calculate Area = length * width
Step 4 : Print Area
Step 5 : End

BEGIN
READ length, width
CALCULATE Area = length * width
DISPLAY Area
END
```

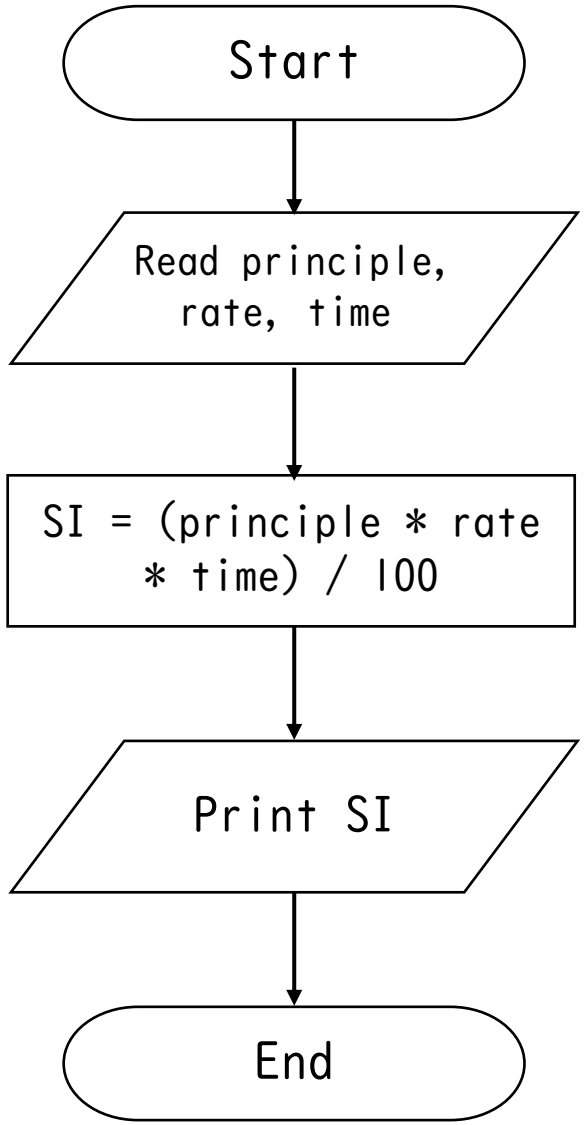
Write an algorithm for Calculating area and circumference of circle




```
Step 1 : Start
Step 2 : Get radius values
Step 3 : Calculate Area = 3.14 * radius * radius
Step 4 : Calculate Circum = 2 * 3.14 * radius
Step 5 : Print Area, Circum
Step 6 : End
```

```
BEGIN
READ radius
CALCULATE Area = 3.14 * radius * radius
CALCULATE Circum = 2 * 3.14 * radius
DISPLAY Area, Circum
End
```

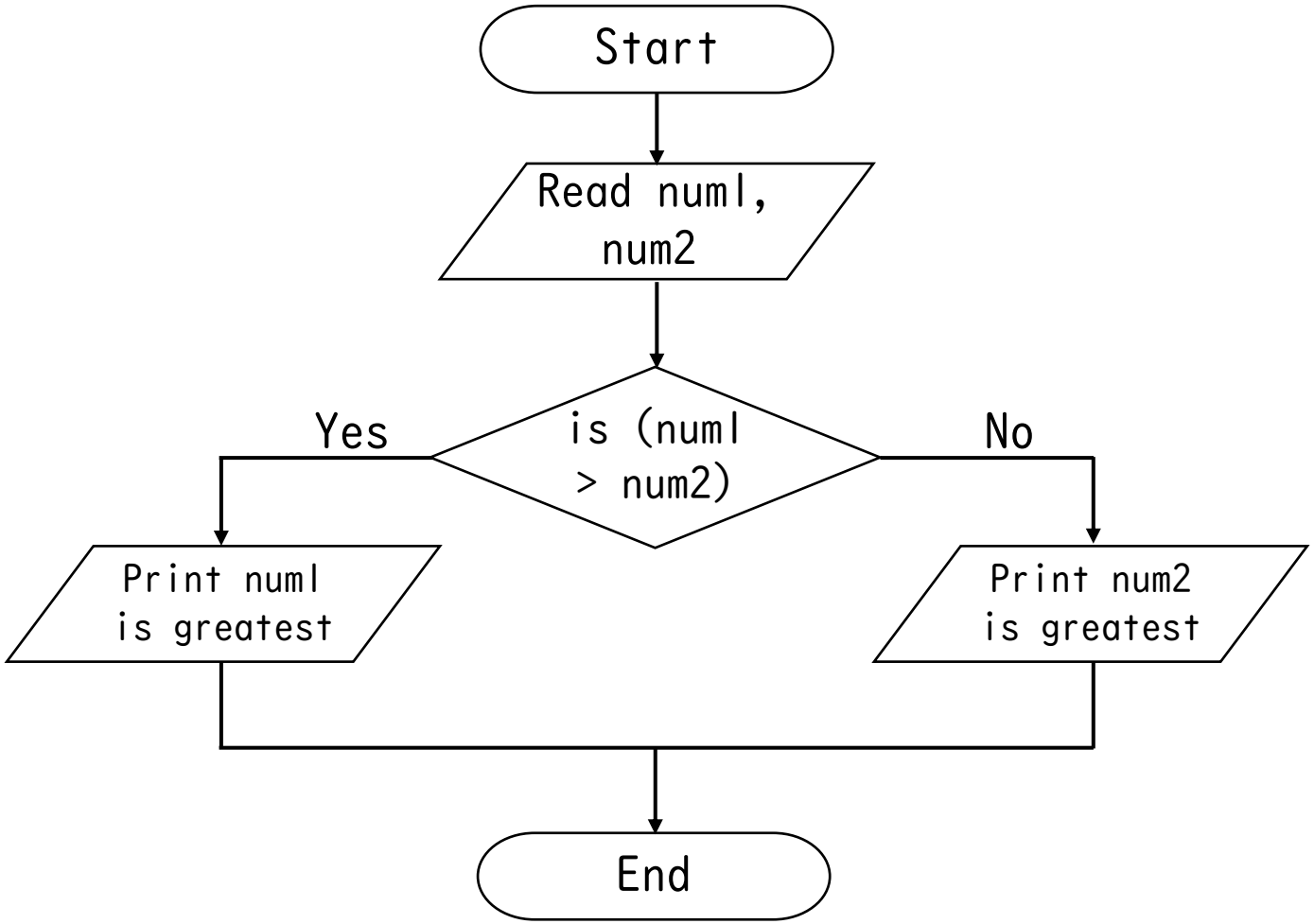
Write an algorithm for Calculating simple interest




```
Step 1 : Start
Step 2 : Get principle, rate, time values
Step 3 : Calculate simpleInterest
         = (principle * rate * time) / 100
Step 4 : Print simpleInterest
Step 5 : End
```

```
BEGIN
READ principle, rate, time
CALCULATE simpleInterest
          = (principle * rate * time)/100
DISPLAY simpleInterest
END
```

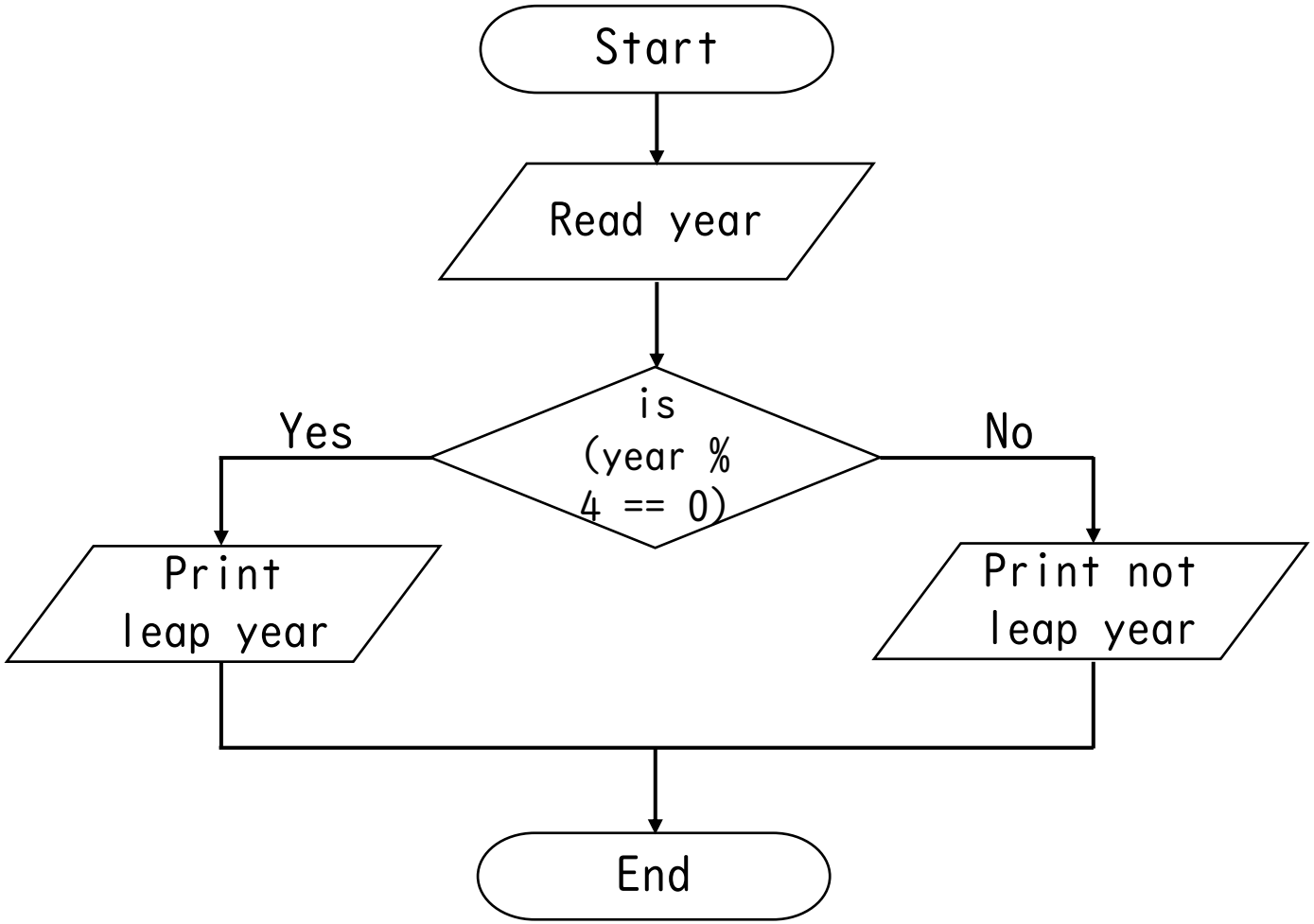
Write an algorithm to check greatest of two numbers




```
Step 1 : Start
Step 2 : Get num1, num2 values
Step 3 : Check if(num1 > num2)
          Print num1 is greatest
Step 4 : else
          Print num2 is greatest
Step 5 : End
```

```
BEGIN
READ num1, num2
IF (num1 > num2) THEN
DISPLAY num1 is greater
ELSE
DISPLAY num2 is greater
END IF
END
```

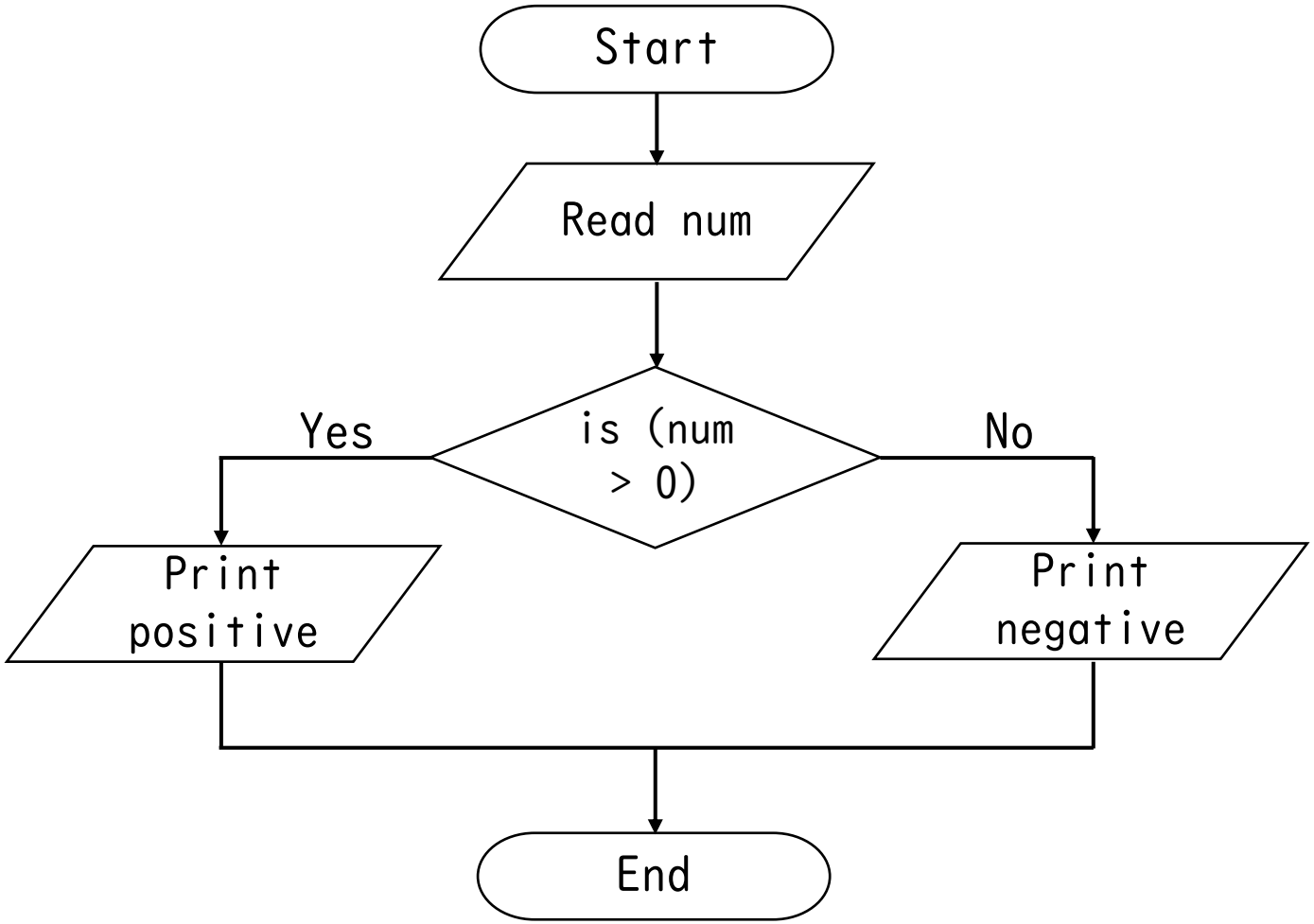
Write an algorithm to check leap year or not




```
Step 1 : Start
Step 2 : Get year values
Step 3 : Check if(year % 4 == 0)
            Print leap year
Step 4 : else
            Print not leap year
Step 5 : End
```

```
BEGIN
READ year
IF (year % 4 == 0) THEN
DISPLAY leap year
ELSE
DISPLAY not leap year
END IF
END
```

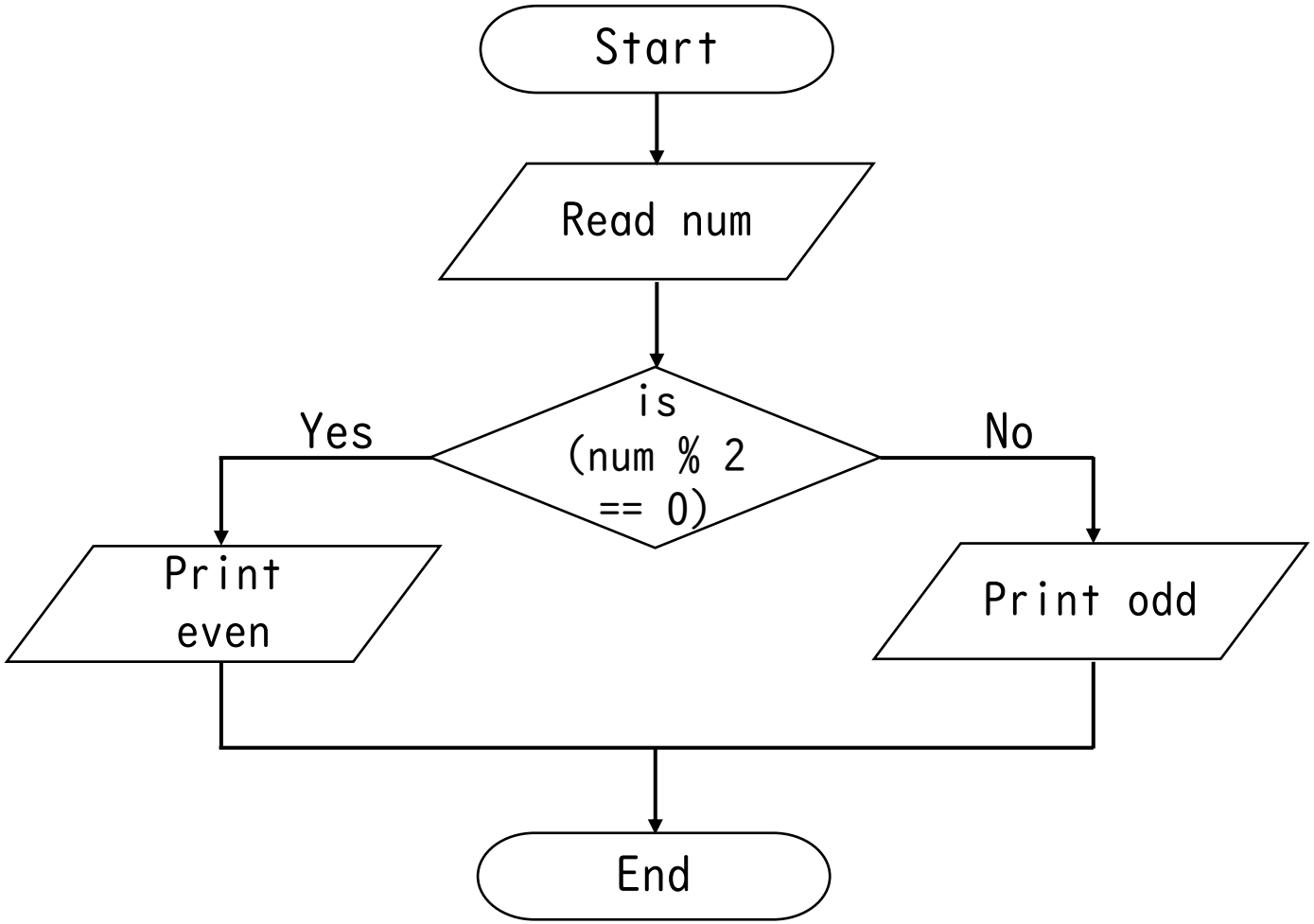
Write an algorithm to check positive or negative number




```
Step 1 : Start
Step 2 : Get num values
Step 3 : Check if(num > 0)
            Print positive number
Step 4 : else
            Print negative number
Step 5 : End
```

```
BEGIN
READ num
IF (num > 0) THEN
DISPLAY positive number
ELSE
DISPLAY negative number
END IF
END
```

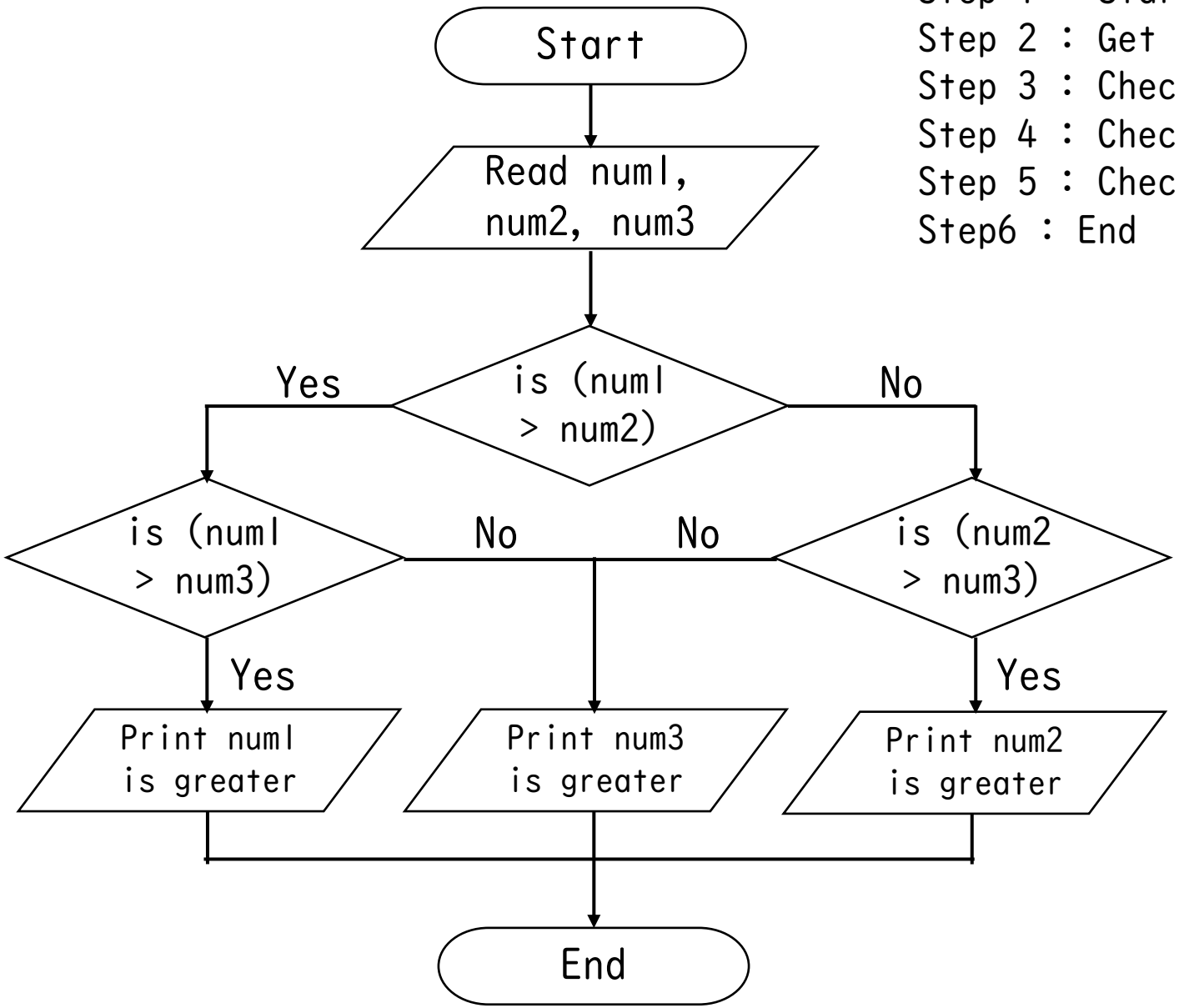
Write an algorithm to check even or odd number




```
Step 1 : Start
Step 2 : Get num values
Step 3 : Check if(num % 2 == 0)
            Print even number
Step 4 : else
            Print odd number
Step 5 : End
```

```
BEGIN
READ num
IF (num % 2 == 0) THEN
DISPLAY even number
ELSE
DISPLAY odd number
END IF
END
```

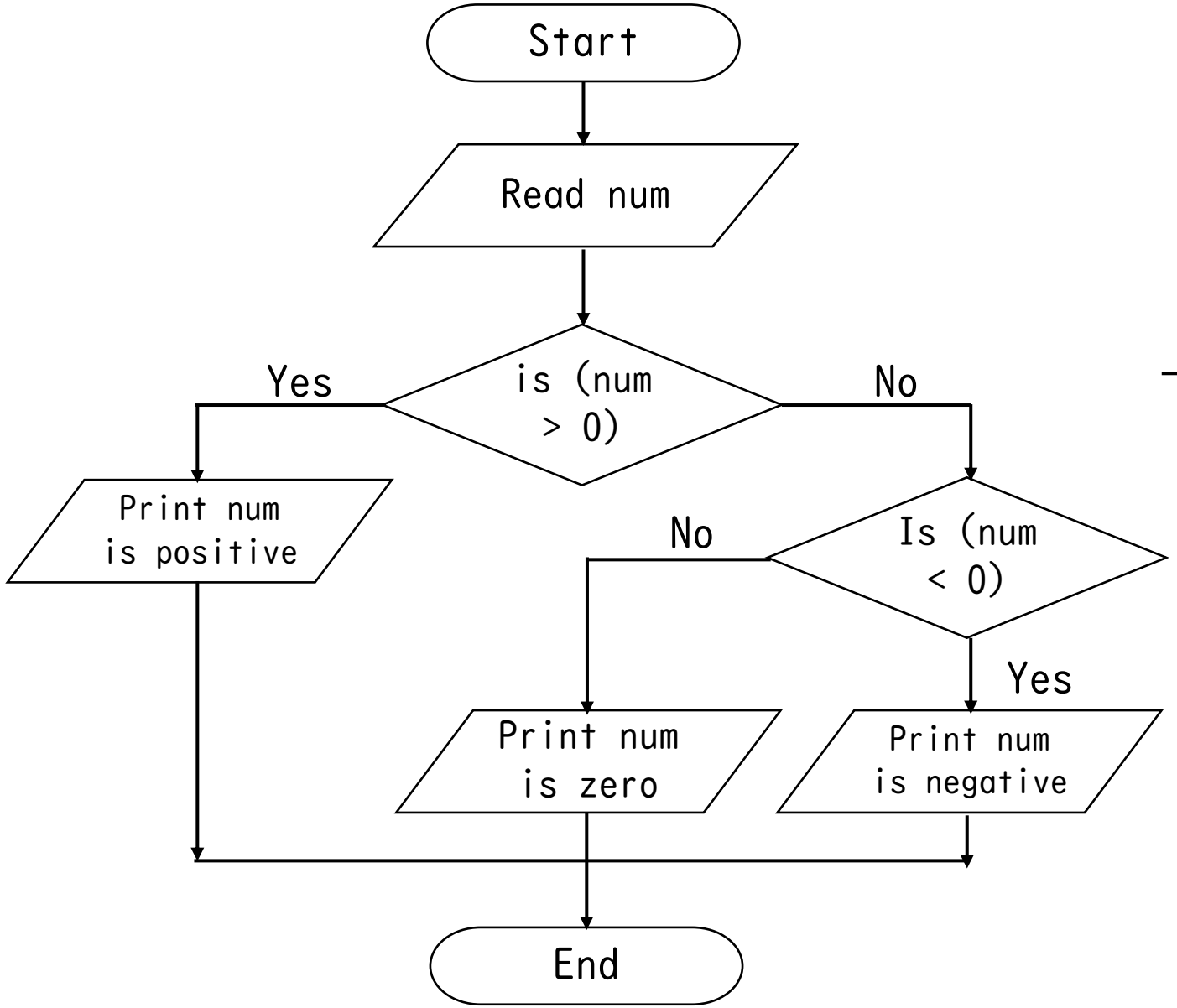
Write an algorithm to check greatest of three numbers




```
Step 1 : Start
Step 2 : Get num1, num2, num3  values
Step 3 : Check if(num1 > num2) goto Step4 else goto Step5
Step 4 : Check if(num1 > num3) Print num1 else Print num2
Step 5 : Check if(num2 > num3) Print num2 else num3
Step6 : End
```

```
BEGIN
READ num1, num2, num3
IF (num1 > num2) THEN
     IF (num1 > num3) THEN
          DISPLAY num1 is greater
     ELSE
          DISPLAY num3 is greater
     END IF
ELSE
     IF(num2 > num3)
          DISPLAY num2 is greater
     ELSE
          DISPLAY num3 is greater
     END IF
END IF
END
```

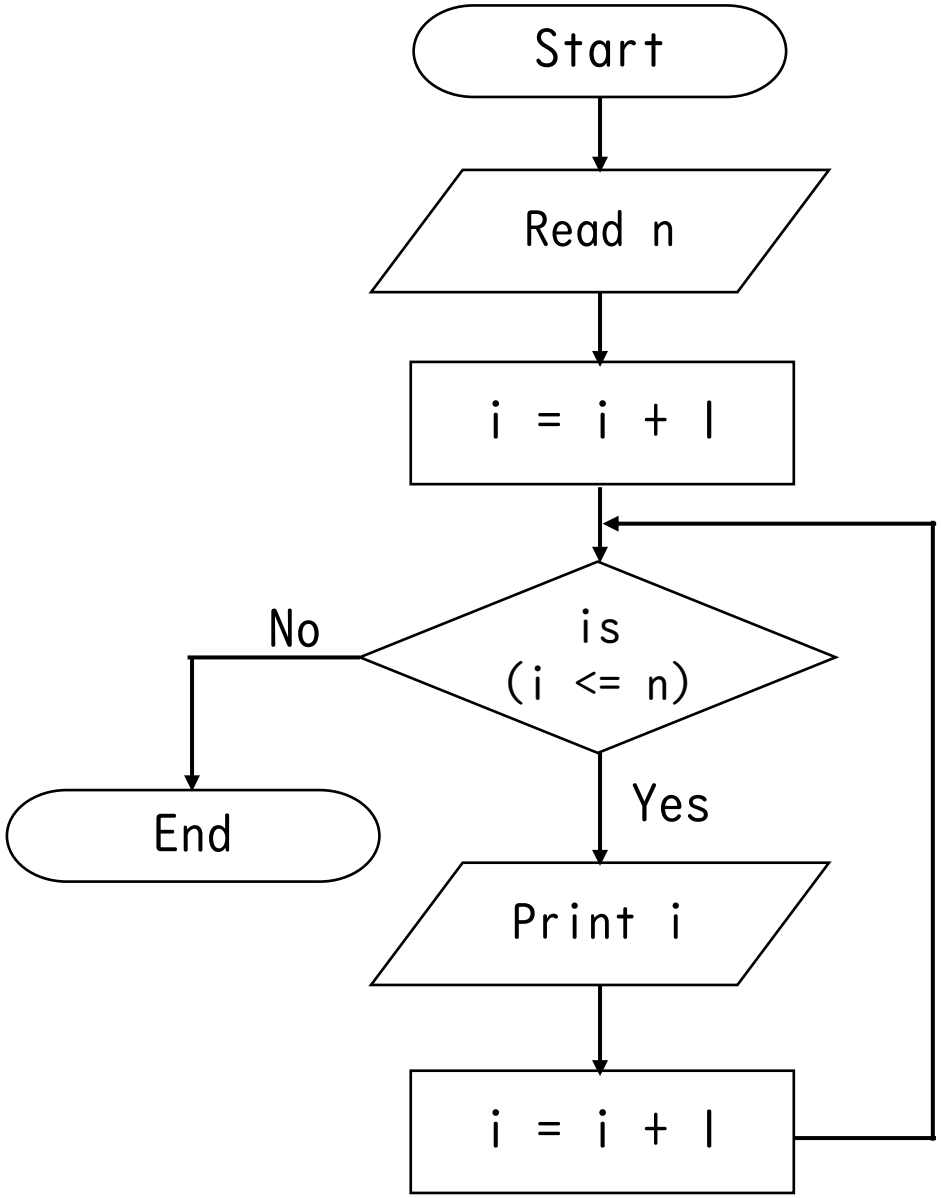
Write an algorithm to check whether given number is +ve, -ve or zero




```
Step 1 : Start
Step 2 : Get num values
Step 3 : Check if(num > 0)
           Print num is +ve else goto Step4
Step 4 : Check if(num < 0) Print num is -ve
Step 5 : else Print num is zero
Step 6 : End
```

```
BEGIN
READ num
IF (num > 0) THEN
    DISPLAY num is positive
ELSE
    IF(num < 0)
        DISPLAY num is negative
       ELSE
        DISPLAY num is zero
       END IF
END IF
END
```

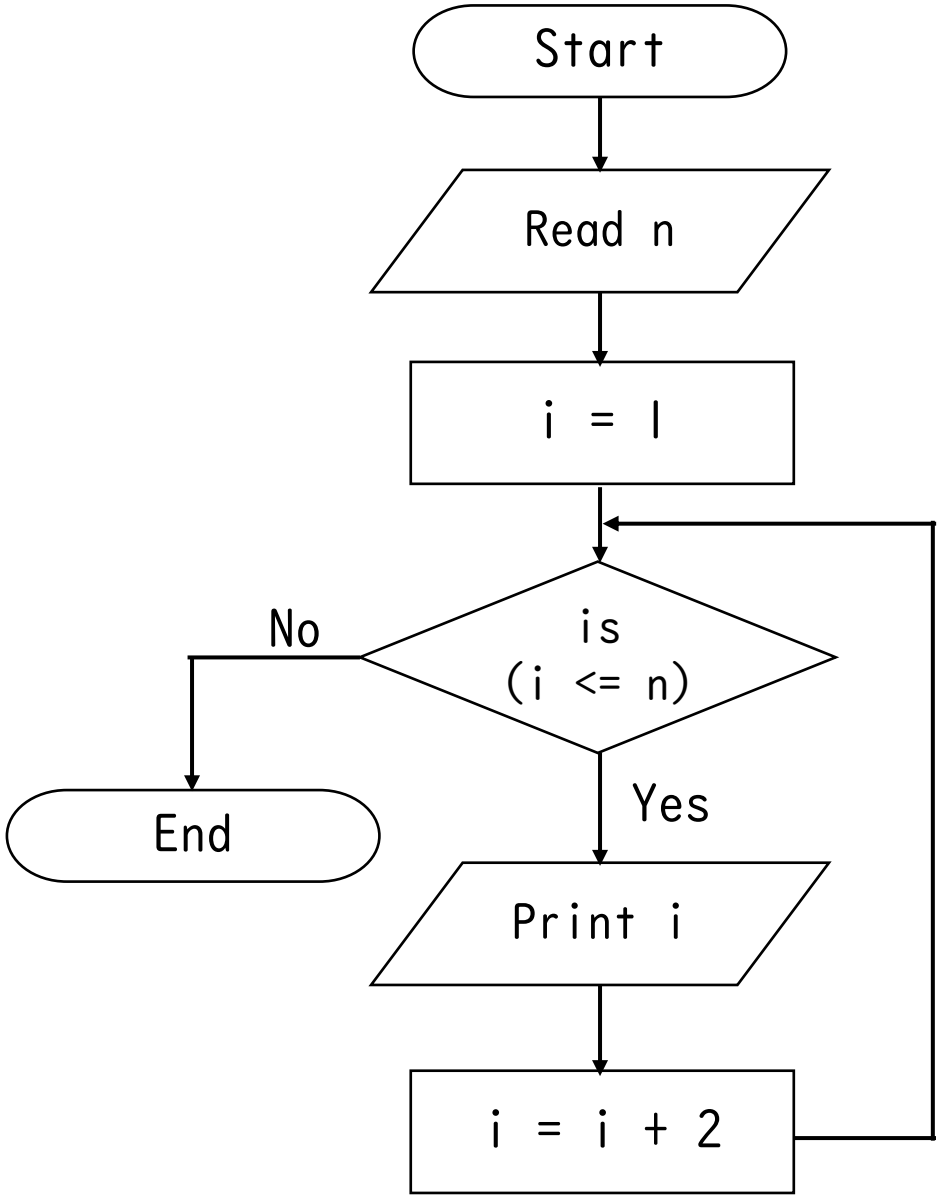
Write an algorithm to print all natural numbers up to n




```
Step 1 : Start
Step 2 : Get n values
Step 3 : Initialize i= 1
Step 4 : If(i <= n) goto Step 5
         else goto Step 8
Step 5 : Print value of i
Step 6 : Increment value of i by 1
Step 7 : Goto Step 4
Step 8 : End
```

```
BEGIN
READ n
INITIAIZE i = 1
WHILE (i <=  n) DO
     PRINT i
     i=i + 1
END WHILE
END
```

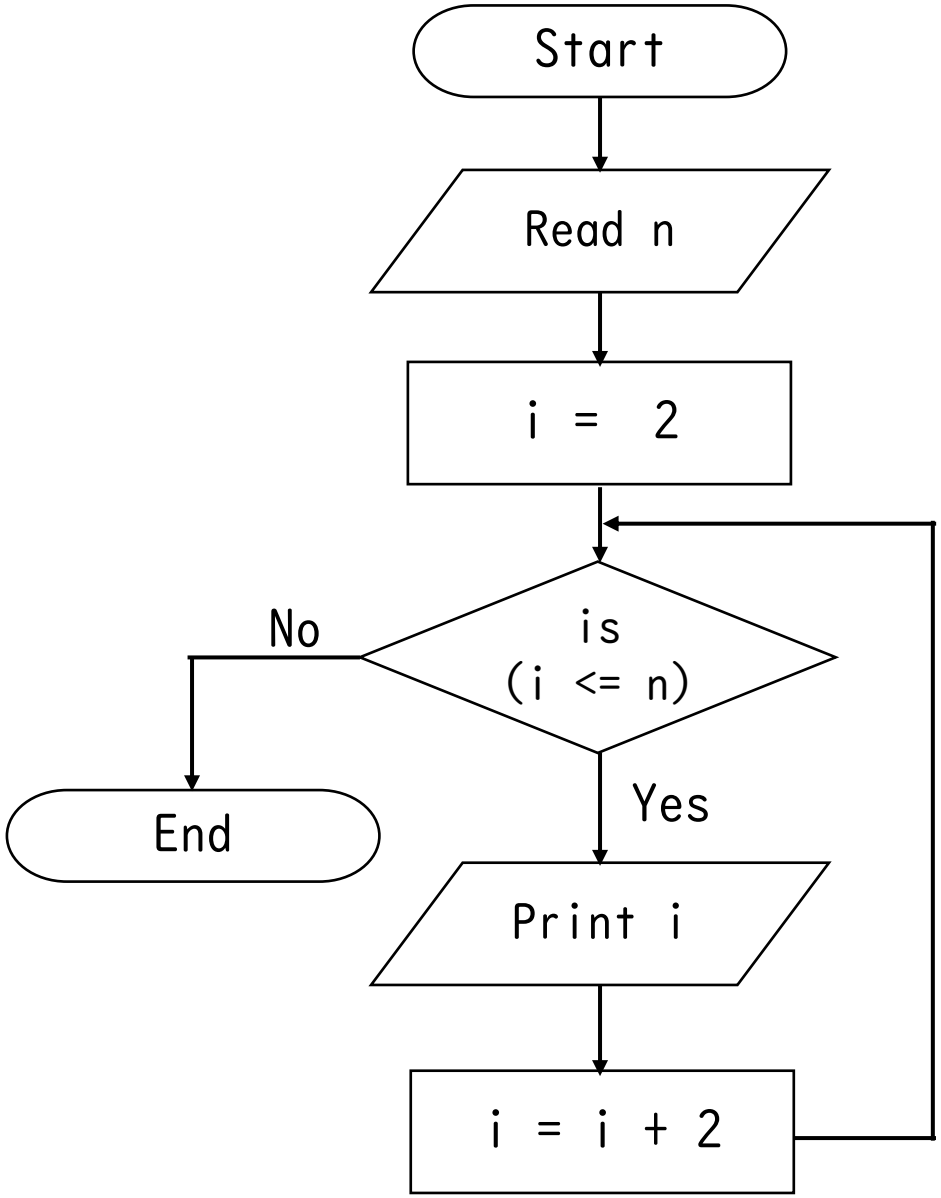
Write an algorithm to print N odd numbers




```
Step 1 : Start
Step 2 : Get n values
Step 3 : Initialize i = 1
Step 4 : If(i <= n) goto Step 5
         else goto Step 8
Step 5 : Print value of i
Step 6 : Increment value of i by 2
Step 7 : Goto Step 4
Step 8 : End
```

```
BEGIN
READ n
INITIAIZE i = 1
WHILE (i <=  n)
     PRINT i
     i=i + 2
END WHILE
END
```

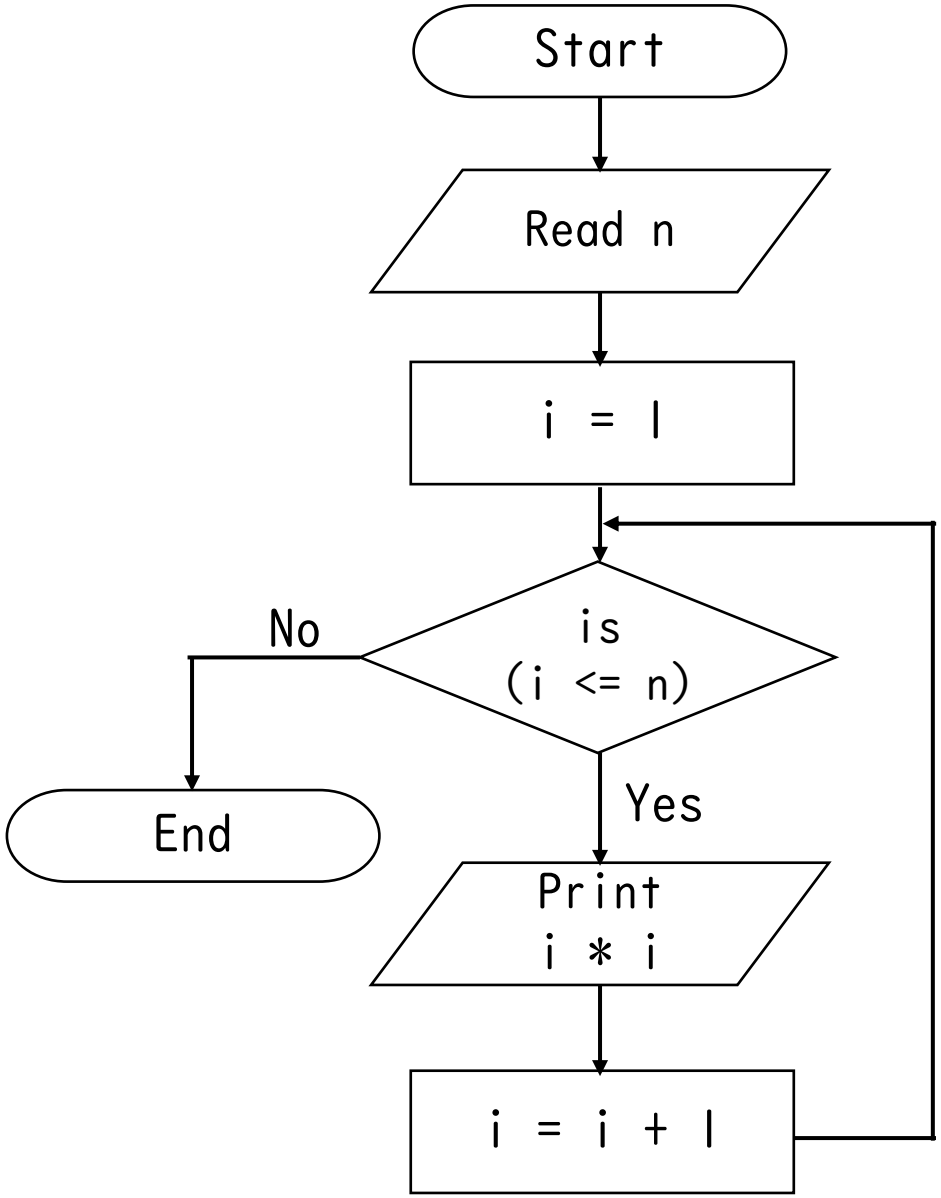
Write an algorithm to print N even numbers




```
Step 1 : Start
Step 2 : Get n values
Step 3 : Initialize i = 2
Step 4 : If(i <= n) goto Step 5
         else goto Step 8
Step 5 : Print value of i
Step 6 : Increment value of i by 2
Step 7 : Goto Step 4
Step 8 : End
```

```
BEGIN
READ n
INITIAIZE i = 2
WHILE (i <=  n) DO
     PRINT i
     i=i + 2
END WHILE
END
```

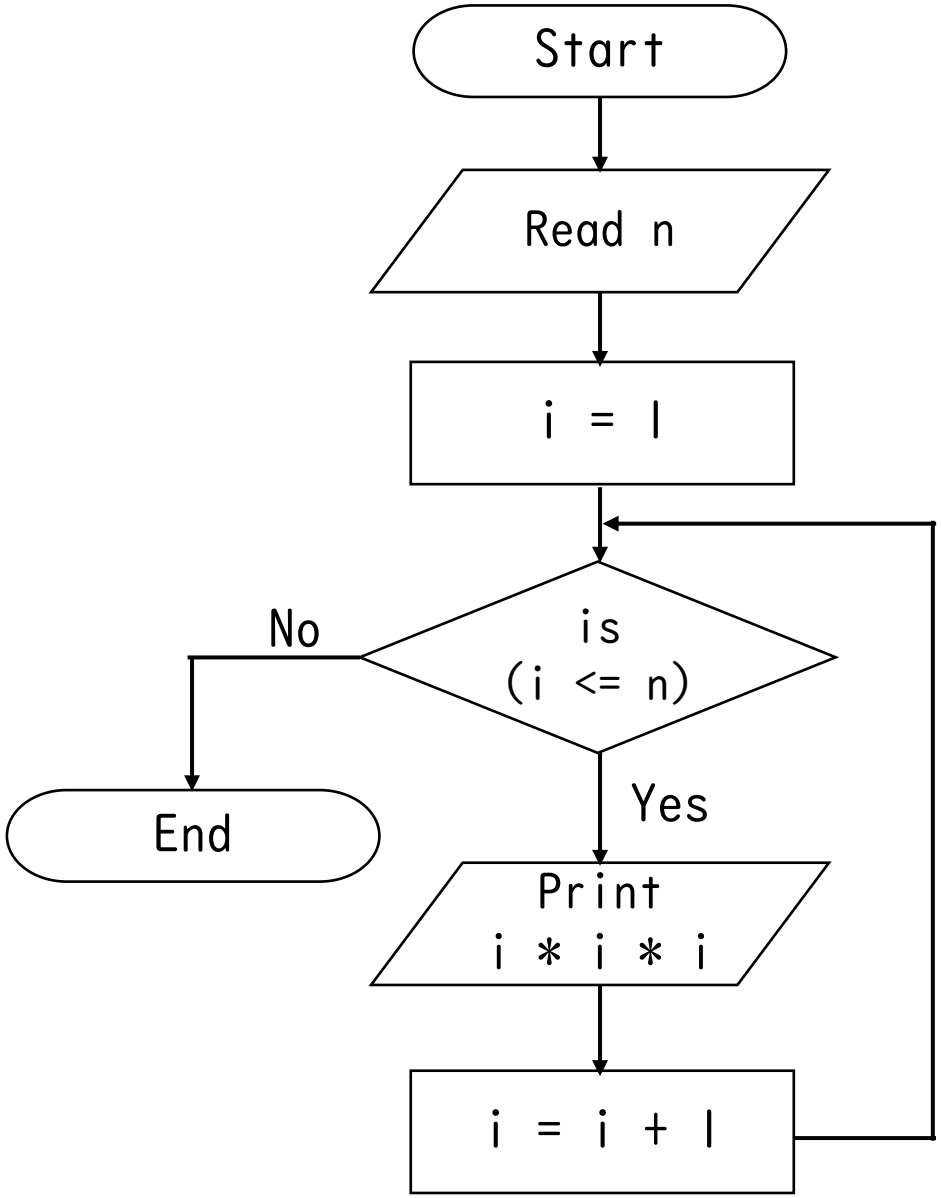
Write an algorithm to print square of N number




```
Step 1 : Start
Step 2 : Get n values
Step 3 : Initialize i = 1
Step 4 : If(i <= n) goto Step 5
         else goto Step 8
Step 5 : Print value of i * i
Step 6 : Increment value of i by 1
Step 7 : Goto Step 4
Step 8 : End
```

```
BEGIN
READ n
INITIAIZE i = 1
WHILE (i <=  n) DO
     PRINT i * i
    i = i + 1
END WHILE
END
```

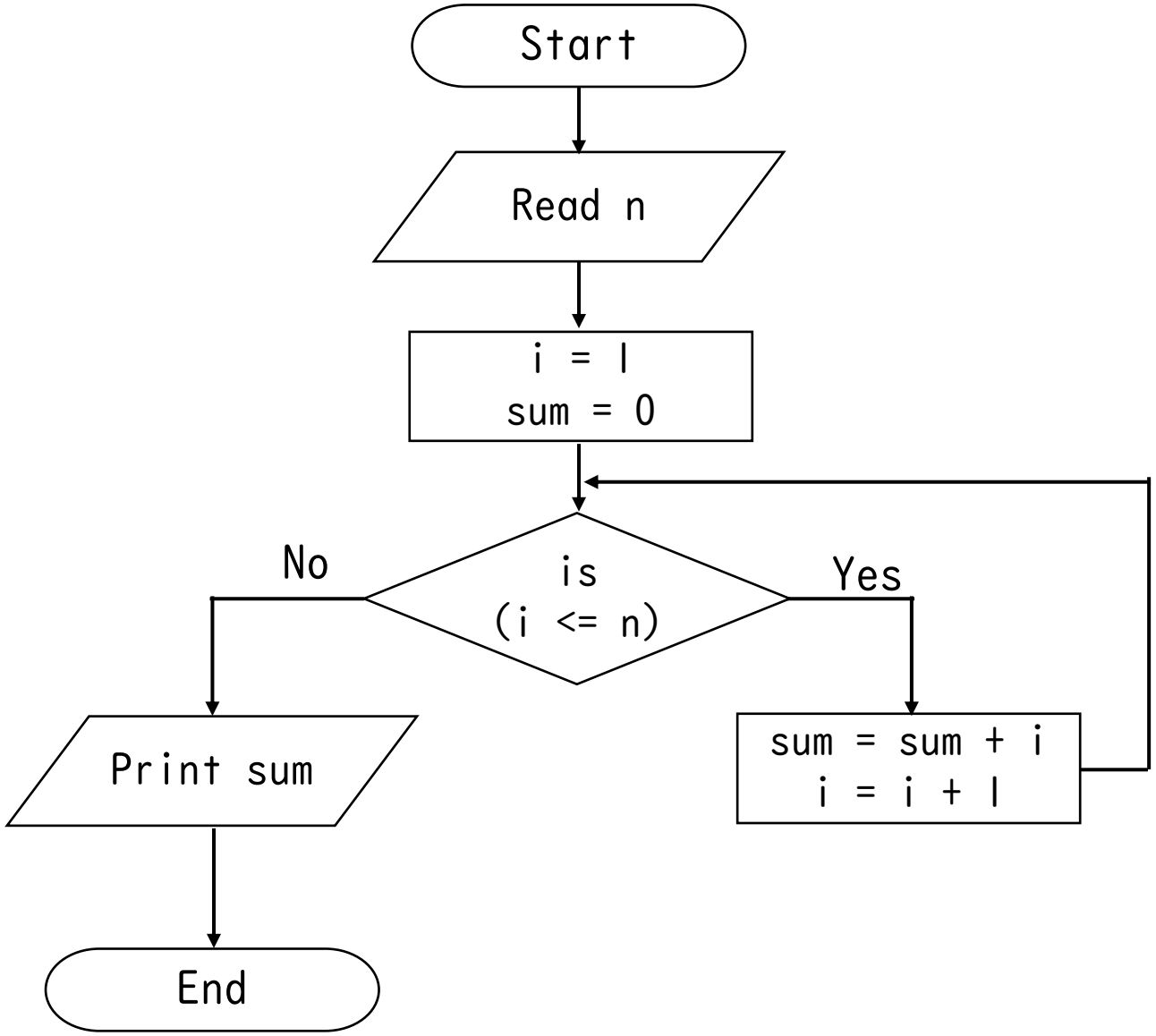
Write an algorithm to print cubes of N number




```
Step 1 : Start
Step 2 : Get n values
Step 3 : Initialize i = 1
Step 4 : If(i <= n) goto Step 5
         else goto Step 8
Step 5 : Print value of i * i * i
Step 6 : Increment value of i by 1
Step 7 : goto Step 4
Step 8 : End
```

```
BEGIN
READ n
INITIAIZE i = 1
WHILE (i <=  n) DO
     PRINT i * i
     i = i  + 1
END WHILE
END
```

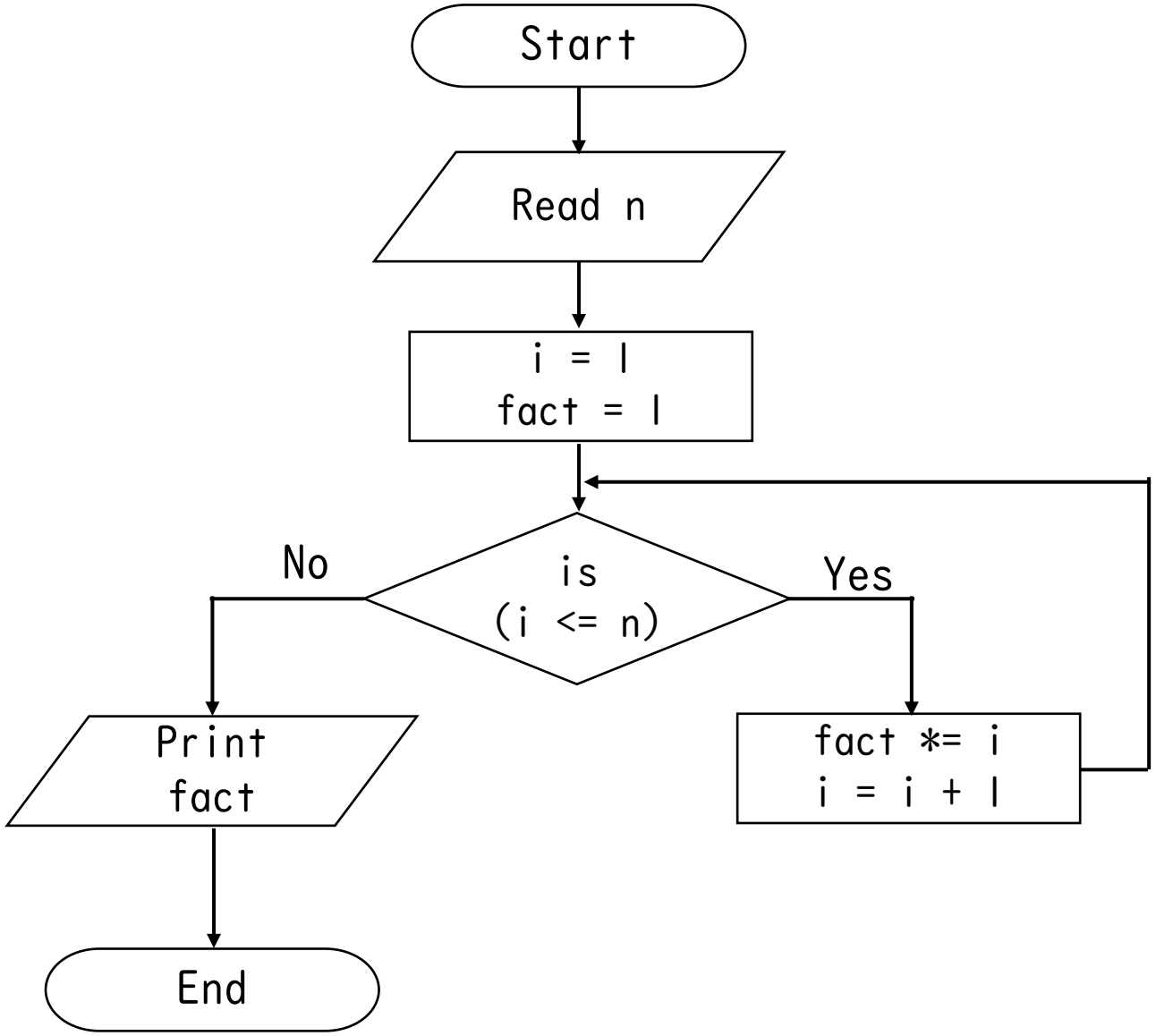
Write an algorithm to find the sum of N given number




```
Step 1 : Start
Step 2 : Get n values
Step 3 : Initialize i = 1, sum = 0
Step 4 : If(i <= n) goto Step 5
         else goto Step 8
Step 5 : Calculate sum += i
Step 6 : Increment value of i by 1
Step 7 : goto Step 4
Step 8 : Print value of sum
Step 9 : End
```

```
BEGIN
READ n
INITIAIZE i = 1, sum = 0
WHILE (i <=  n) DO
     sum += i
     i = i  + 1
END WHILE
PRINT sum
END
```

Write an algorithm to find factorial of a given number




```
Step 1 : Start
Step 2 : Get n values
Step 3 : Initialize i = 1, fact = 1
Step 4 : If(i <= n) goto Step 5
         else goto Step 8
Step 5 : Calculate fact *= i
Step 6 : Increment value of i by 1
Step 7 : goto Step 4
Step 8 : Print value of fact
Step 9 : End

BEGIN
READ n
INITIAIZE i = 1, fact = 1
WHILE (i <=  n) DO
     fact *= i
     i = i  + 1
END WHILE
PRINT fact
END
```