ALGORITHMIC THINKING WITH PYTHON

Module 2

Module – 2 - syllabus

ALGORITHM AND PSEUDOCODE REPRESENTATION:- Meaning and Definition of Pseudocode, Reasons for using pseudocode, The main constructs of pseudocode - Sequencing, selection (if-else structure, case structure) and repetition (for, while, repeat- until loops), Sample problems*

FLOWCHARTS** :- Symbols used in creating a Flowchart - start and end, arithmetic calculations, input/output operation, decision (selection), module name (call), for loop (Hexagon), flow-lines, on-page connector, off-page connector.

Algorithm

An **algorithm** describes a systematic way of solving a problem.

It is a step-by step procedure that produces an output when given the necessary inputs.

An algorithm uses pure English phrases or sentences to describe the solution to a problem.

Write an algorithm to evaluate an expression $d = a + b * c$.

1. Start
2. Read the values of a, b and c.
3. Find the product of b and c.
4. Store the product in a temporary variable *temp*.
5. Find the sum of a and *temp*.
6. Store the sum in d.
7. Print the value of d.
8. Stop.

Pseudocode

A **Pseudocode** is a high-level representation of an algorithm that uses a mixture of natural language and programming language-like syntax.

It is more structured than an algorithm.

It uses mathematical expressions with English phrases to capture the essence of a solution concisely.

Why pseudocodes?

Ease of understanding: Since the pseudocode is programming language independent, novice developers can also understand it very easily.

Focus on logic: A pseudocode helps to focus on the algorithm’s logic without bothering about the syntax of a specific programming language.

More legible: Combining programming constructs with English phrases makes pseudocode more legible and conveys the logic precisely.

Why pseudocodes?

Consistent: The constructs used in pseudocode are standardized - useful in sharing ideas among developers from various domains.

Easy translation to a program: Programming constructs ensures mapping the pseudocode to a program straightforward.

Identification of flaws: A pseudocode helps to identify flaws in the solution logic before implementation.

Write a pseudocoden algorithm to evaluate an expression $d = a + b * c$.

```
1. Start
2. Read($a, b, c$)
3. $d = a + b * c$
4. Print($d$)
5. Stop
```

In a pseudocode, **Read** is used to read input values.

Print is used to print a message.
The message to be printed should be enclosed in a pair of double quotes.

Difference between algorithm and pseudocode

Algorithm	Pseudocode
An algorithm is a step-by-step procedure for solving a problem or achieving a specific task, expressed in a finite and well-defined sequence of steps	Pseudocode is a high-level representation of an algorithm, intended to provide a clear and concise description of the logic and flow of an algorithm
It can be expressed in any language or notation, including natural language, mathematical symbols, or programming code.	It uses a combination of programming language and natural language elements
It is a precise and well-defined procedure	It is a more informal and high-level representation of an algorithm
Give an example	Give an example

Constructs of a pseudocode

A good pseudocode should follow the structured programming approach.

Structured coding - improves the readability of pseudocode by ensuring that the execution sequence follows the order in which the code is written.

Such a code is said to have a *linear flow of control*.

Constructs of a pseudocode

The three programming constructs for linear control flow

- Sequencing,
- Selection
- Repetition (loop)

These are also known as *single entry – single exit* constructs.

Sequential control flow

The most elementary construct where the instructions of the algorithm are executed one after other in the order they appear in code

Example

1) Statement 1
2) Statement 2
3) Statement 3
4) ⋮
5) Statement n

- Statement 1 is executed first, which is then followed by Statement 2, so on and so forth until all the instructions are executed.
- No instruction is skipped.
- Every instruction is executed only once.

Pseudocode to find average of three numbers

```
START
    READ num1, num2, num3
    $sum = num1 + num2 + num3$
    $average = sum / 3$
    PRINT (“The Average is ”, average )
END
```

Decision or Selection control flow

A selection/ conditional structure consists of a ***test*** condition together with one or more blocks of statements.

The result of the test determines which of these blocks is executed.

The flow of the program branch to different path based on certain condition.

Selection control flow - types

1. `if` structure
2. `case` structure

`if` structure (3 variants)

```
if (condition)
   True_instructions
endif
```

```
if (condition)
   True_instructions
else
   False_instructions
endif
```

```
if (condition1)
   True_instructions1
else if (condition2)
   True_instructions2
else
   False_instructions
endif
```

Selection control flow - types

`case` structure

```
caseof (expression)
case 1 value1:
    block1
case 2 value2:
    block2 ...
default :
    default_block
endcase
```

`if`

If the test condition is evaluated to **True**, the block of statements following **`if`** are executed. Otherwise, those statements are skipped.

Example:

Write a pseudocode to check if the given number is positive

```
if $(x > 0)$
        Print( “The number $x$ is positive”)
endif
```

Pseudocode that checks if a number is greater than 100

```
BEGIN
    Read a number
    if number > 100 THEN
            PRINT ("Number is greater than 100”)
    endif
END
```

if…else

Write a pseudocode to check if a person is a major or not.

```
if ($age$ >= 18)
        Print("You are a major")
else
        Print("You are a minor")
endif
```

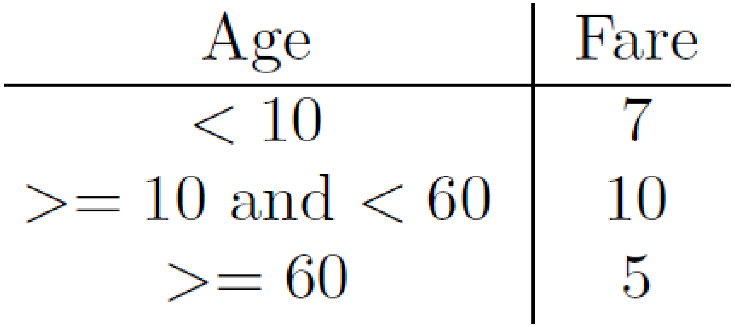
Pseudocode that assigns “A” grade if marks >= 90

```
BEGIN
    PRINT "Enter your marks:"
    INPUT marks
    IF marks >= 90 THEN
        PRINT ("Grade: A“)
    ELSE
        PRINT ("Below A grade.“)
    END IF
END
```

if...else if... else

Write a pseudocode to determine the entry-ticket fare in a zoo based on age.

```
Start
    Read($age$)
    if ($age < 10$)
        $fare = 7$
    else if ($age < 60$)
        $fare = 10$
    else
        $fare = 5$
    endif
    Print($fare$)
Stop
```



Age	Fare
< 10	7
>= 10 and < 60	10
>= 60	5

Pseudocode for a simple traffic light system.

```
BEGIN
    PRINT "Enter the traffic light color (green, yellow, red):"
    INPUT light_color
    IF light_color = "green" THEN
        PRINT "Go!"
    ELSE IF light_color = "yellow" THEN
        PRINT "Slow down!"
    ELSE IF light_color = "red" THEN
        PRINT "Stop!"
    ELSE
        PRINT "Invalid color!"
    END IF
END
```

Repetition

- When a certain block of instructions is to be repeatedly executed - use the repetition or loop construct.
- Each execution of the block is called an **iteration** or a **pass**.
- If the number of iterations (how many times the block is to be executed) is known in advance, it is called **definite iteration**.
- Otherwise, it is called **indefinite** or **conditional iteration**.
- The block that is repeatedly executed is called the *loop body*.

Repetition - types

`while` loop	**while** (*condition*)
	statements
	endwhile

`repeat` loop	**repeat**
	statements
	until (*condition*)

`for` loop	**for** *var* = *begin* to *end*
	statements
	endfor

Difference between `repeat` and `while`

- In the **while** loop, the condition is tested at the beginning; in the **repeat…until** loop, the condition is tested at the end.
- The `while` loop is known as an ***entry controlled*** *loop* and the `repeat-until` loop is known as an ***exit controlled*** *loop*.

`while` loop

- Used to implement indefinite iteration.
- Here, the loop body is executed repeatedly as long as condition evaluates to **True**.
- When the condition is evaluated as False, the loop body is bypassed

Pseudocode to find the sum of first 15 numbers using **while**

```
Start
    Initialize sum as 0
    Initialize counter as 1
    WHILE number is less than or equal to 15
        Add number to sum
        Increment counter by 1
    END WHILE
    Print sum
End
```

Note : sum = 0;
Because it should be initialized with 0, else some random garbage value will be assumed !!

`repeat-until` loop

- Used for indefinite iteration
- Body of loop is repeated as long as *condition* evaluates to **False**.
- When the condition evaluates to **True**, the loop is exited.

Note : When the condition is tested at the end, the instructions in the loop are executed at least once !

Pseudocode to find the sum of first 15 numbers using **repeat until**

```
BEGIN
    sum ← 0
    num ← 1
    REPEAT
        sum ← sum + num
        num ← num + 1
    UNTIL num > 15
    PRINT (“The sum is ”, sum)
END
```

`for` loop

- The **`for`** loop implements definite iteration.
- **`for`** loop use a variable - *loop variable*
- By default the loop variable is incremented/ decremented by **“1”**
- Loop variable can be updated by an amount other than 1 after every iteration

`for` loop with step

```
for var = begin to end by step
    Statements
endfor
```

`for` loop examples

Loop construct	Description	Values taken by var
for var = 1 to 5	var gets incremented by 1 till it reaches 5	1,2,3,4,5
for var = 5 downto 1	var gets decremented by 1 till it reaches 1	5,4,3,2,1
for var = 1 to 10 by 2	var gets increased by 2 till it reaches 10	1,3,5,7,9
for var = 50 downto 40 by 2	var gets decreased by 2 till it reaches 40	50,48,46,44,42 ,40

Pseudocode to display numbers

Display 1 to 50

```
Start
    for $count = 1$ to 50
        Print($count$)
    endfor
Stop
```

Display 50 to 1

```
Start
    for $count = 50$ downto$1$
        Print($count$)
    endfor
Stop
```

Pseudocode to find the sum of first 15 numbers using `for`

```
BEGIN
    SET sum = 0
        FOR number FROM 1 TO 15
            DO sum = sum + number
        END FOR
    PRINT (“The sum”)
END
```

Practice

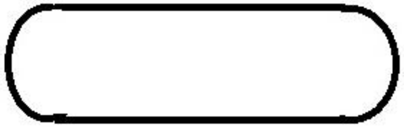
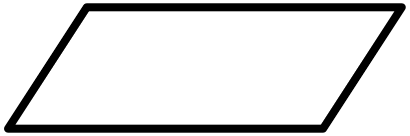
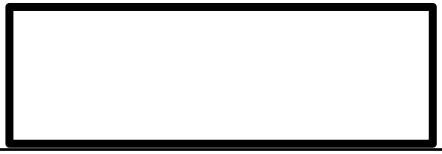
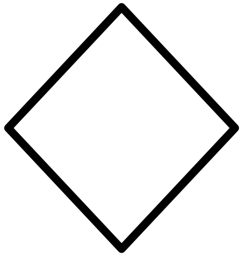
1. Write a pseudocode to check whether a number is divisible by 5. If it is, print "Divisible by 5." Otherwise, print "Not divisible by 5."
2. Write a pseudocode to find the simple interest
3. Write a pseudocode to find the largest of three numbers

Flow Chart

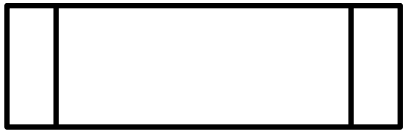
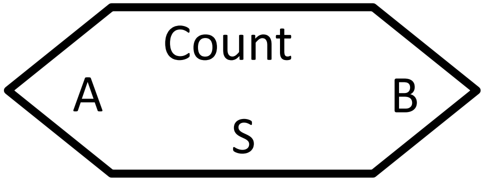
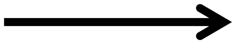
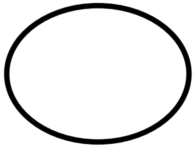
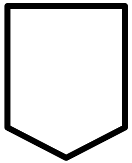
Flowchart

- A flowchart is a diagrammatic representation of an algorithm that depicts how control flows in it.
- Flowcharts are composed of various *blocks* interconnected by *flow-lines*.
- Each block in a flowchart represents some stage of processing in the algorithm.
- Different types of blocks are defined to represent the various programming constructs of the algorithm.

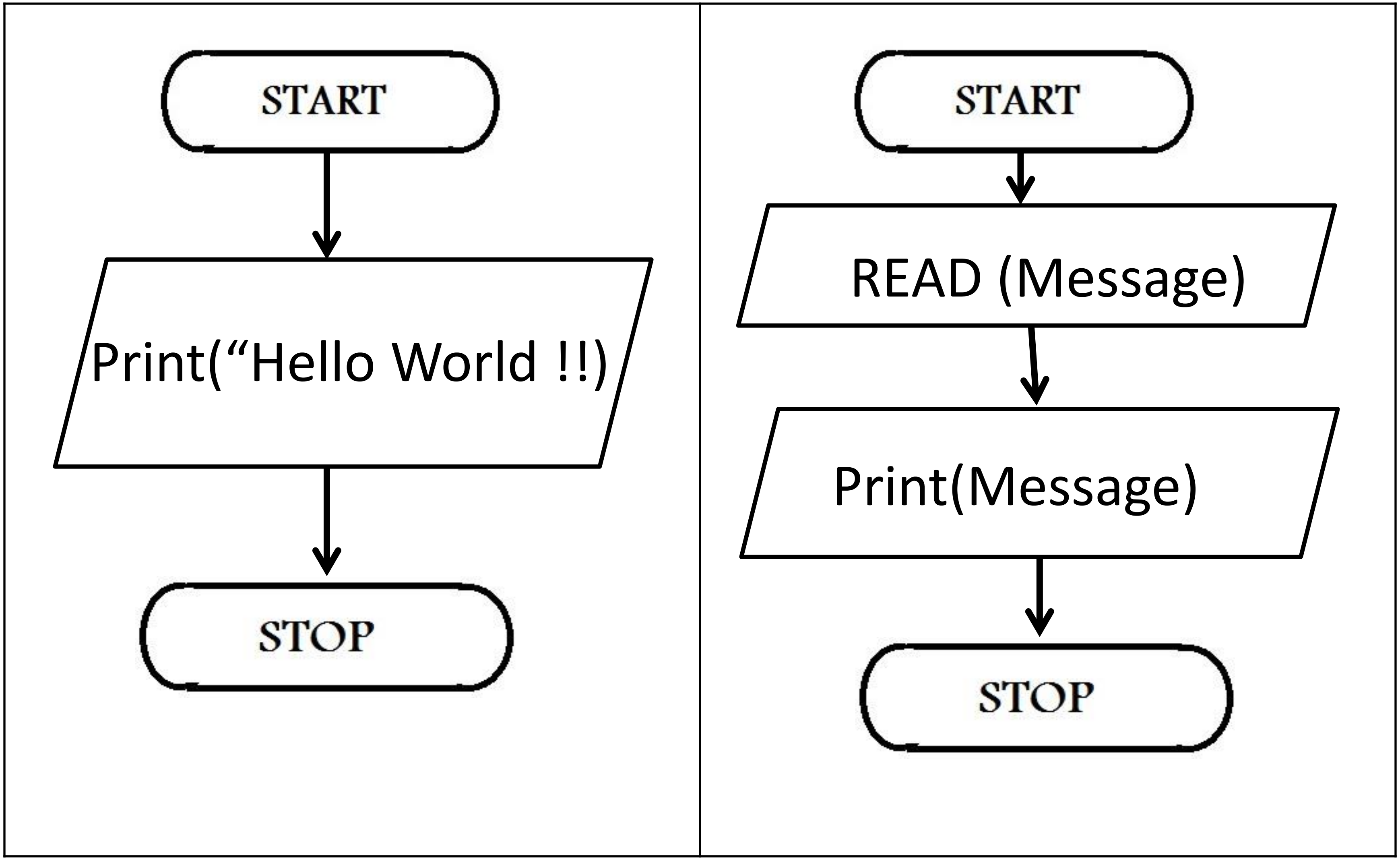
Flowchart Symbols

Flow chart symbol	Description
	Flattened ellipse indicates the start and end of a module.
	Parallelogram denotes an input/output operation.
	Rectangle is used to show arithmetic calculations.
	Diamond indicates a decision box with a condition to test. It has two exits. One exit leads to a block specifying the actions to be taken when the tested condition is True and the other exit leads to a second block specifying the actions for False case.

Flowchart Symbols

Flow chart symbol	Description
	Rectangle with vertical side-lines denotes a module. A module is a collection of statements written to achieve a task. It is known by the name ***function*** in the programming domain.
	Hexagon denotes a **`for`** loop. The symbol shown here is the representation of the loop: **for** *count* = *A* **to** *B* **by** *S*
	Flowlines are indicated by arrows to show the direction of data flow. Each flowline connects two blocks.
	On-page connector used when one part of a long flowchart is drawn on one column of a page and the other part in the other column of the same page.
	Off-page connector used when the flowchart is very long and spans multiple pages.

Flow chart to print a message



START

Print(“Hello World !!)

STOP

START

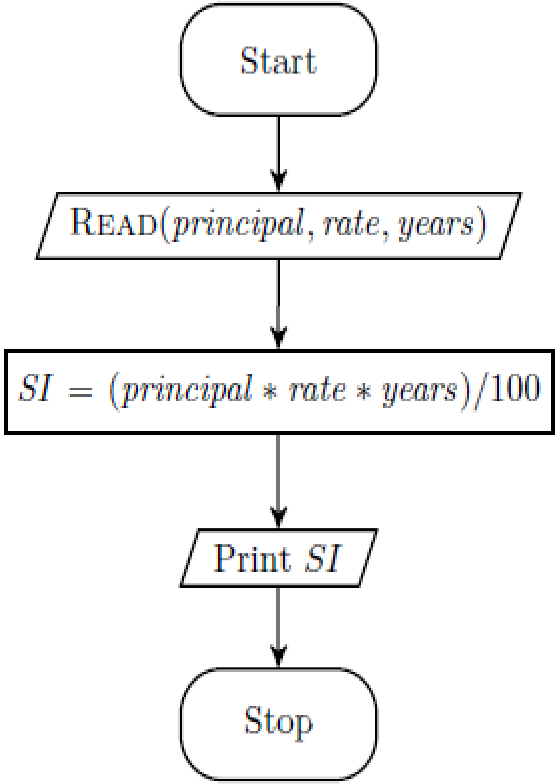
READ (Message)

Print(Message)

STOP

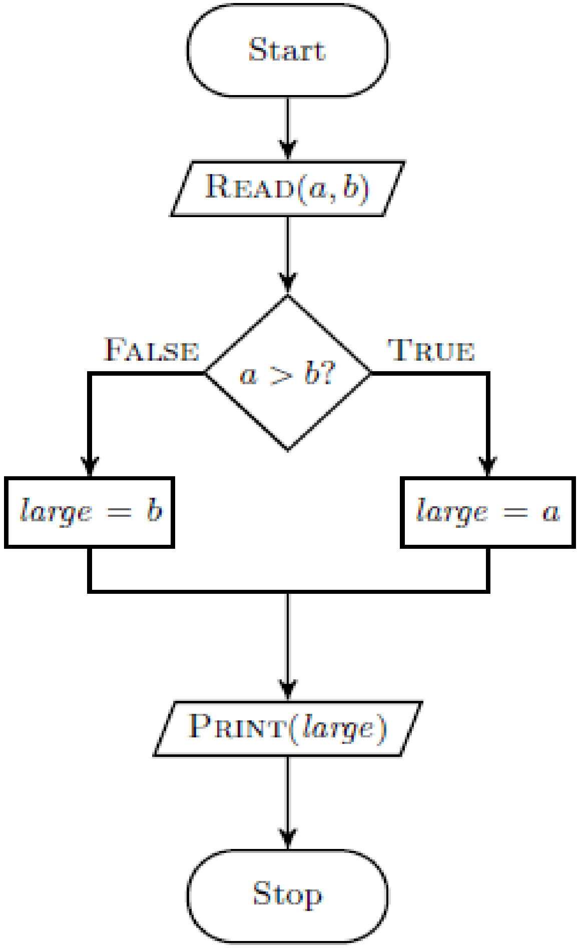
Flow chart to calculate simple interest

```
1 Start
2 Read($principal$ , $rate$, $years$)
3 $SI = (principal * rate * years) / 100$
4 Print($SI$ )
5 Stop
```


Start
READ($principal, rate, years$)
$SI = (principal * rate * years)/100$
Print SI
Stop

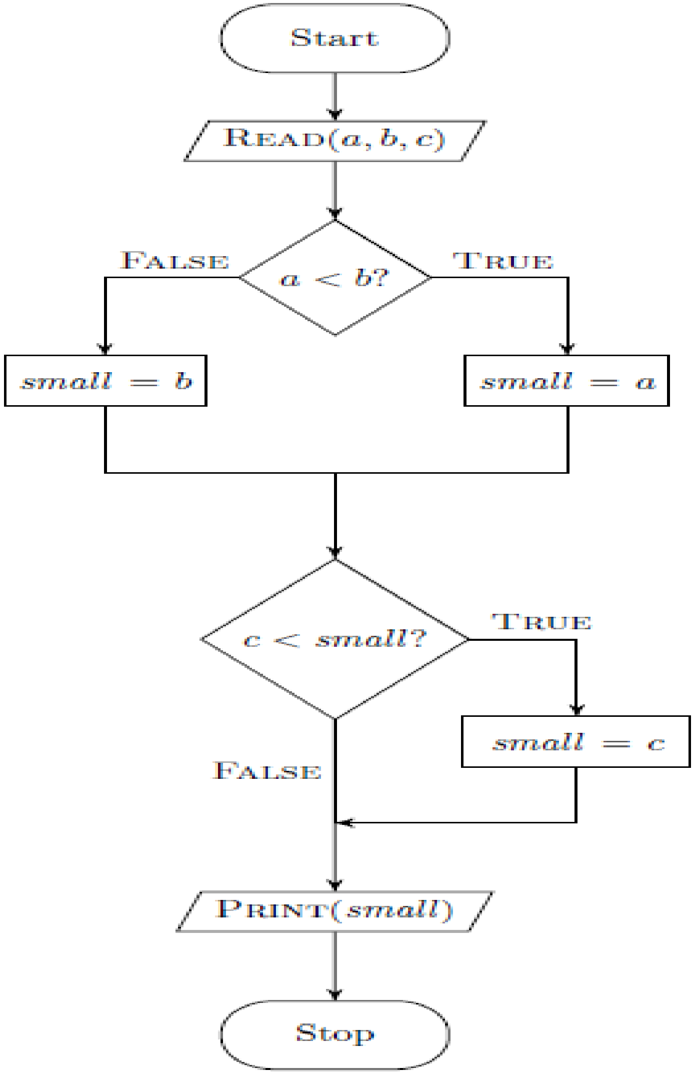
Flow chart to find largest of two numbers

```
Start
Read($a$, $b$)
if ($a > b$)
    $large = a$
else
    $large = b$
endif
Print($large$)
Stop.
```



Flow chart for smallest of three numbers

```
SmallestThree
Start
Read($a, b, c$)
if ($a < b$)
        $small = a$
else
        $small = b$
endif
if ($c < small$)
        $small = c$
endif
Print($small$)
Stop.
```



Largest of *N* numbers

40	50	30	60	70

Store largest value as variable LARGE

Assume 40 is LARGE

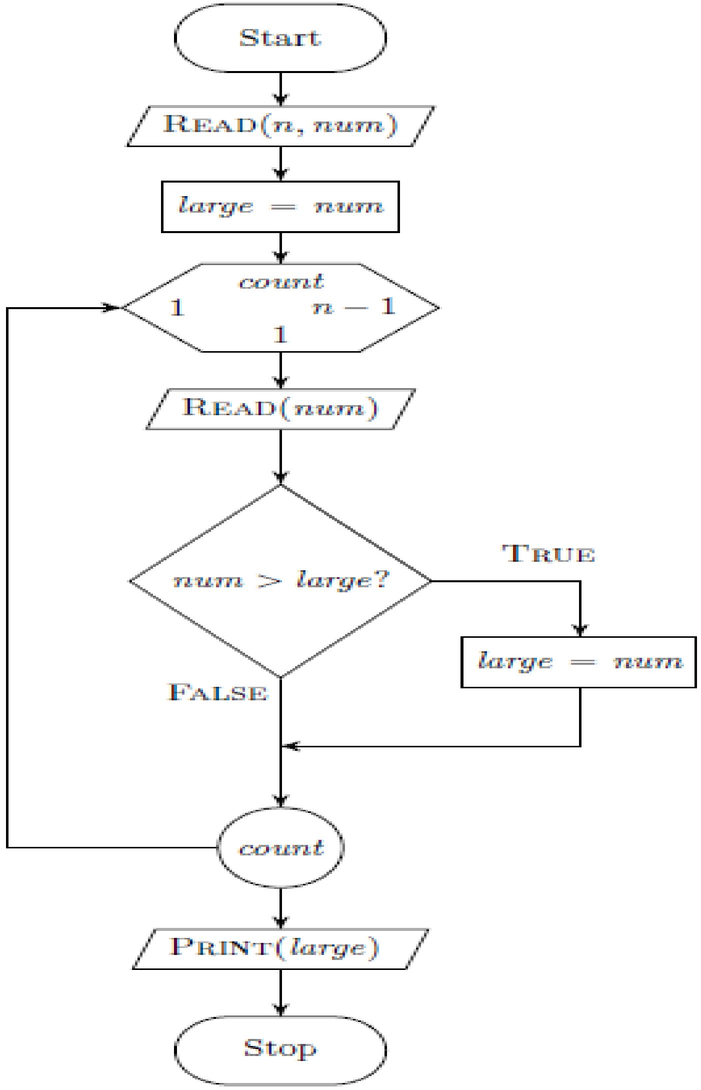
Compare LARGE with all elements

					LARGE ⇩
40	**50**	30	60	70	**50**
40	50	**30**	60	70	**50**
40	50	30	**60**	70	**60**
40	50	30	60	**70**	**70**

The largest element is 70

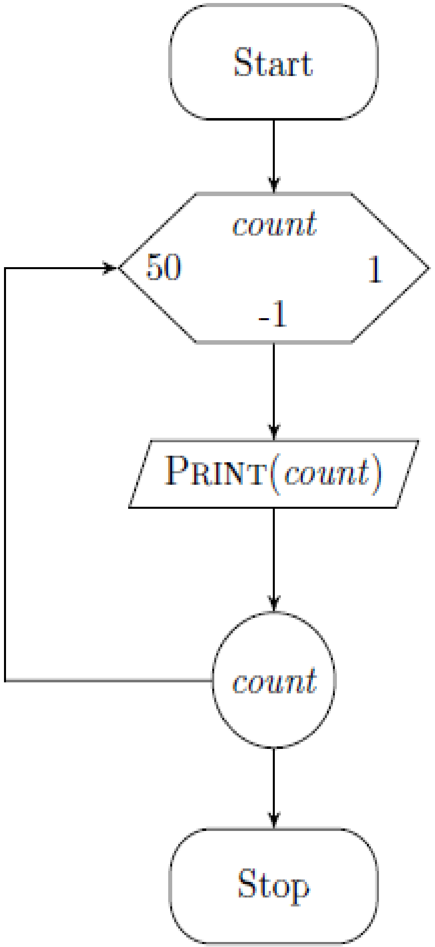
Flow chart for Largest of N numbers

```
LARGEN
Start
READ($n, num$)
$large = num$
for $count = 1$ to $n - 1$
    READ($num$)
    if ($num > large$)
        $large = num$
    endif
endfor
PRINT($large$)
Stop
```

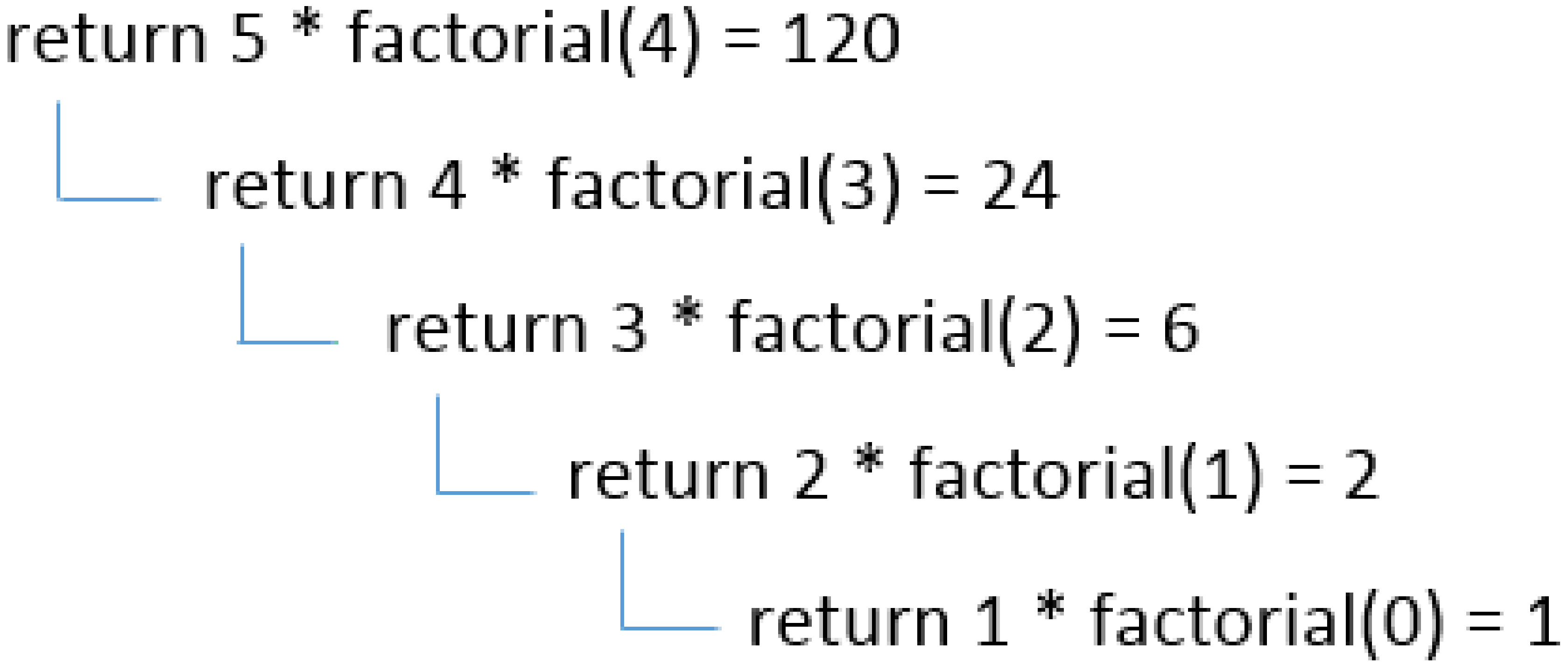


Flow chart to print numbers in descending order

```
Start
for $count = 50$ downto 1
    Print($count$)
endfor
Stop
```



To find Factorial



return 5 * factorial(4) = 120

return 4 * factorial(3) = 24

return 3 * factorial(2) = 6

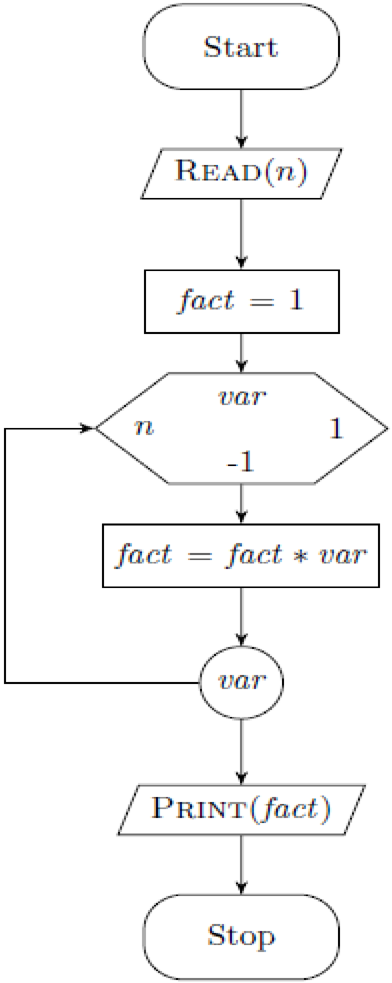
return 2 * factorial(1) = 2

return 1 * factorial(0) = 1

1 * 2 * 3 * 4 * 5 = 120

Flow chart to find factorial of a number

```
Start
Read($n$)
$fact = 1$
for $var = n$ downto 1
        $fact = fact * var$
endfor
Print($fact$)
Stop
```



Start
READ(n)
$fact = 1$
var
n
1
-1
$fact = fact * var$
var
PRINT($fact$)
Stop