

# Flowchart-based programming



Andrea Sterbini – [sterbini@di.uniroma1.it](mailto:sterbini@di.uniroma1.it)

# Flowcharts

Flowcharts show the possible execution paths of the program

Every program has a single input and single output (initial edge)

An edge can be a sub-flowchart/component with single IN/OUT

- single-thread execution (but what about fork/join?)
- Flowgorithm [flowgorithm.org](https://flowgorithm.org)
- Algobuild [algobuild.com](https://algobuild.com)
- Raptor [raptor.martincarlisle.com](https://raptor.martincarlisle.com) (with OOP!)
- Visual Logic [visuallogic.org](https://visuallogic.org)
- PseInt [pseint.SF.net](https://pseint.SF.net) (in Spanish)

# Flowgorithm = Flow-chart + Algorithm

Executable flow-charts

Personalized flow-chart STYLE and COLOURS

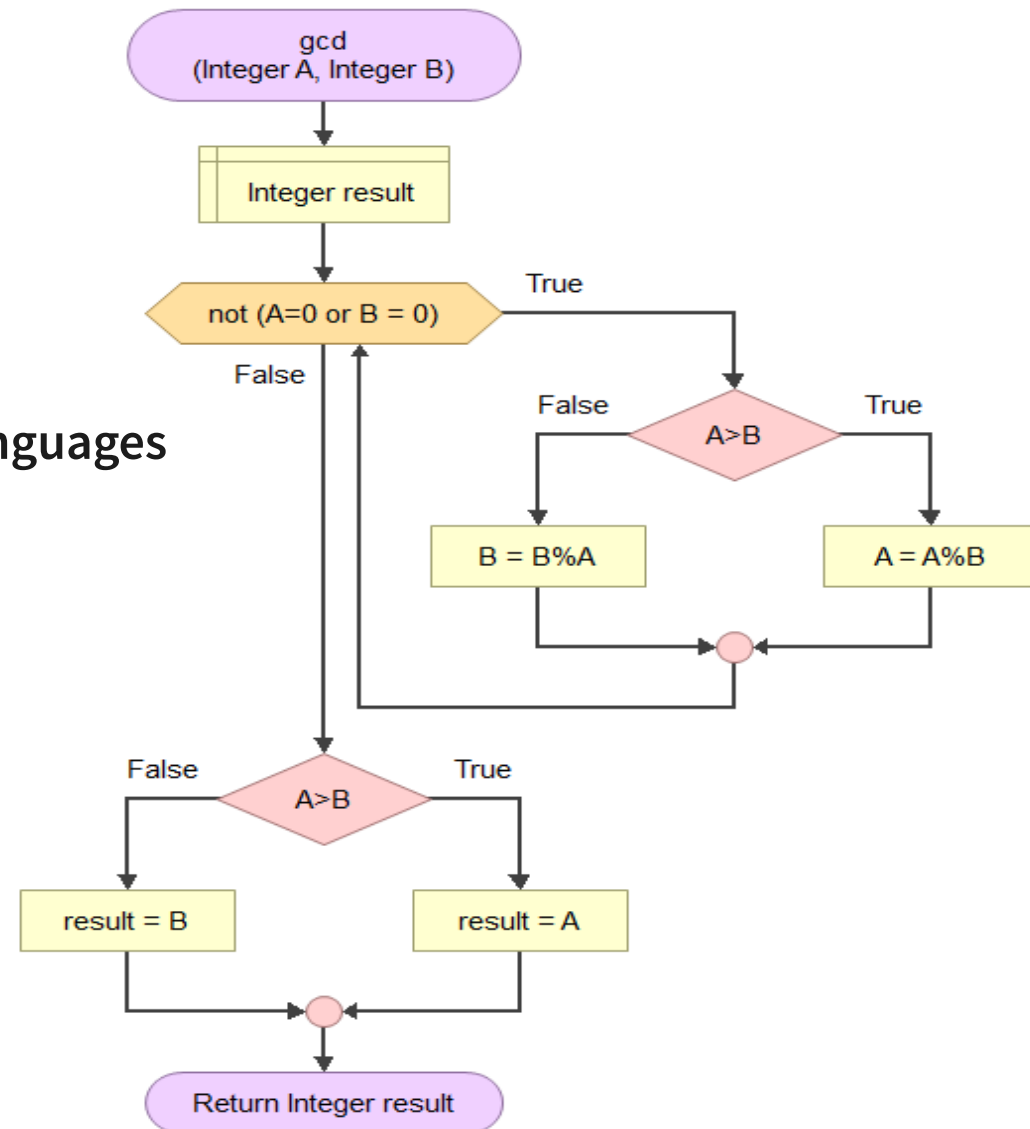
Generate your code in many programming languages

MISSING: loading a program source  
and generating its flow-chart  
(BUT there are tools for that)

- [code2flow.com](https://code2flow.com)

- ...

Methods in Computer Science education



# Flowgorithm IDE windows

The image displays the Flowgorithm IDE interface with four main windows:

- FLOWCHART:** A flowchart for calculating the factorial of an integer N. It starts with a process block 'factorial (Integer N)', followed by a data block 'Integer result'. A decision block 'N < 2' branches the flow: if True, it sets 'result = 1'; if False, it sets 'result = N \* factorial(N-1)'. Both paths merge and lead to a final process block 'Return Integer result'.
- TURTLE GRAPHICS:** A window for visual programming using a turtle. It contains a red box labeled 'TURTLE GRAPHICS' and a small orange arrow icon.
- GENERATED PROGRAM:** A window showing the Python code generated from the flowchart. The code defines a recursive function 'factorial(n)' and a main program that prompts the user for an integer and prints the result.
- CONSOLE (I/O):** A window for running the program. It shows a green message box 'Type a positive integer between 0 and 20' and a blue input box containing the number '7'.

On the right side, the **Variable Watch** window is open, showing a **STACK** of variables:

Variable	Value
factorial	
N	5
factorial	
N	6
result	Uninitialized
factorial	
N	7
result	Uninitialized
Main	
N	7
result	Uninitialized

# Only simple data types (and 1 dimensional arrays)

T = Integer, Float, String, Boolean

1 dimensional Array of <T>

NO bigintegers (like Python) → you must consider range of possible values

NO lists or dynamic arrays

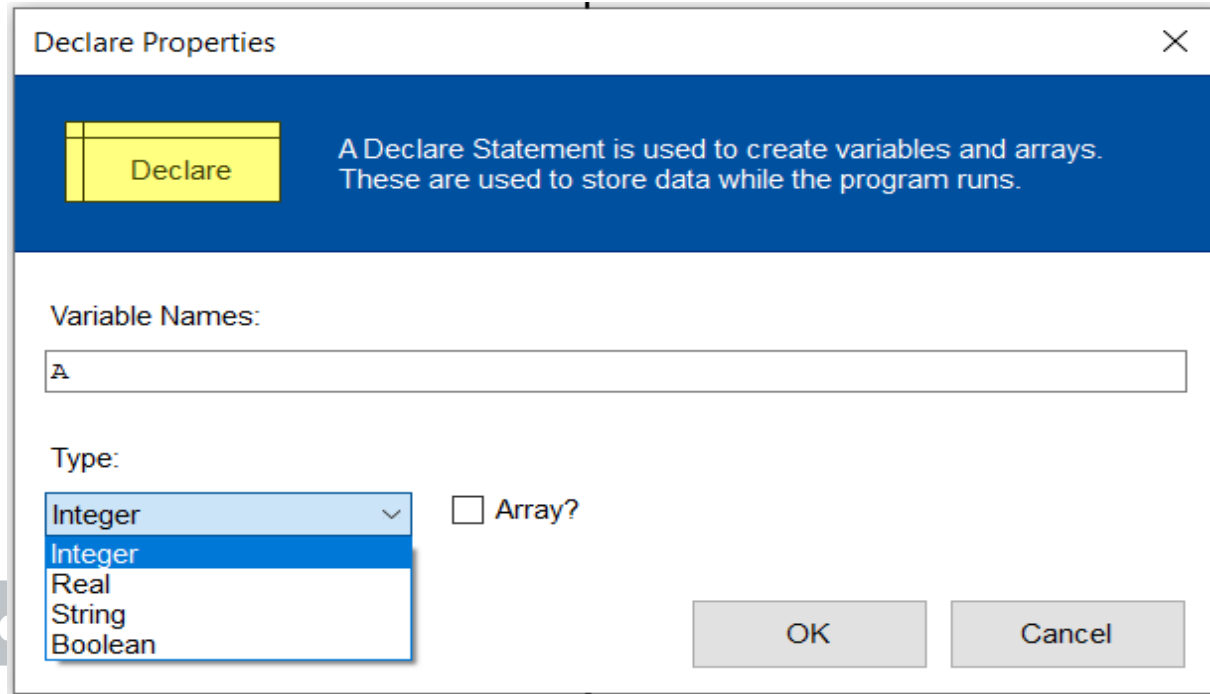
NO heterogeneous arrays

NO multidimensional arrays

NO objects

NO coroutines

NO function objects



The screenshot shows a 'Declare Properties' dialog box. It has a title bar with a close button. Below the title bar is a blue header area with a yellow 'Declare' button and a text box explaining that a Declare Statement is used to create variables and arrays. The main area of the dialog contains a 'Variable Names:' label followed by a text input field containing the letter 'A'. Below this is a 'Type:' label followed by a dropdown menu. The dropdown menu is open, showing a list of data types: Integer, Integer, Real, String, and Boolean. To the right of the dropdown is a checkbox labeled 'Array?'. At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

Declare Properties

**Declare**

A Declare Statement is used to create variables and arrays. These are used to store data while the program runs.

Variable Names:

A

Type:

Integer  
Integer  
Real  
String  
Boolean

☐ Array?

OK Cancel

# Statements

DECLARE/ASSIGN variable

INPUT & OUTPUT text or number

IF-THEN-ELSE

CALL procedure/function

WHILE-do / counted FOR / DO-while  
(but NO foreach)

COMMENTS & BREAKPOINTS

TURTLE GRAPHICS (new!!!)

FILES I/O (new!!!)

ONLY 1 open file at a time!!!

Methods in Computer Science ed

Clipboard



Miscellaneous

Comment



Breakpoint

Statement

Input / Output

Variables

Control

Looping

Input

Declare

If

While

Output

Assign

Call

For

Do

Turtle Graphics

Files

Turn

Forward

Read

Open

Home

Write

Close

# Expressions and operators

## Function calls

Logic: and, or, not, comparison

Math: +, -, \*, /, %, ^, sign, trigonometry, log/pow, random, round

String: concat, len, char(S, i)

Arrays: size

Conversions: char, ascii, int, float, str, round

Precedence of operators in expressions as usual

# Control flow

Functions?	YES	
args by reference?	NO	(except for arrays like C)
multiple return values?	NO	(single simple types only)
Recursion?	YES	
ONE entry and ONE exit per function/diagram		
NO early return		(use an IF to skip the rest of the code)
NO break		(use an IF to skip the rest of the code)
Multiple assignments?	NO	
Concurrency/multi threading?	NO	
Events?	NO	
Exceptions?	NO	(errors are shown but you cannot catch them)



# Programming style

PROCEDURAL/SEQUENTIAL?	YES	
FUNCTIONAL?	NO	no functions as arguments
STRUCTURED?	YES	
DECLARATIVE?	NO	
EVENT-BASED?	NO	
CONCURRENT?	NO	
MODULARIZATION?	YES	by function/procedure
ANALYSIS		
TOP-DOWN?	YES	
BOTTOM-UP?	NO	
OBJECT-ORIENTED?	NO	no objects

# Debug support

Step-by-step execution (both flow-chart AND generated code)

NOTE: the generated code is NOT executed (only shown)

View Variables content (both simple values and arrays)

Show the Stack content (good to understand recursion)

Breakpoints (then step by step)

Assertions? NO (add if-then by hand)

Exceptions? PARTIAL (some errors are generated, but cannot be handled)

IDE support

Refactoring PARTIAL (cut/paste flowchart into new functions)

# Code is generated by templates

Code generation  
from flow-charts  
to many  
programming  
languages  
(custom also)

Methods in Compute



Ada 95



Applescript



Bash



C#



C++



Fortran 2003



Java



JavaScript



Kotlin



Lua



MATLAB



Nim



Pascal



Perl



PHP



Powershell



Python



QBasic



Ruby



Scala



Smalltalk



Swift



Transact-SQL



TypeScript



VBA



Visual Basic .NET



Gaddis Pseudocode



IBO Pseudocode



Auto Pseudocode



Open...

## Example template: Python

A section with some global info (keywords, ext, case-sensitive ...)

The template contains required imports and definitions for some missing functions (you can extend it if you like)

Types are mapped to corresponding Python types

Diagram elements map to corresponding templates

Each Flowgorithm expression operator or intrinsic function is mapped to the corresponding Python one (with precedence levels)

Functions definition and call templates

DEMO

# Literate programming / Documentation?

Program properties:

Title, Author, Description

BUT: they are NOT present in the generated code!!!

Comments in the flow-chart

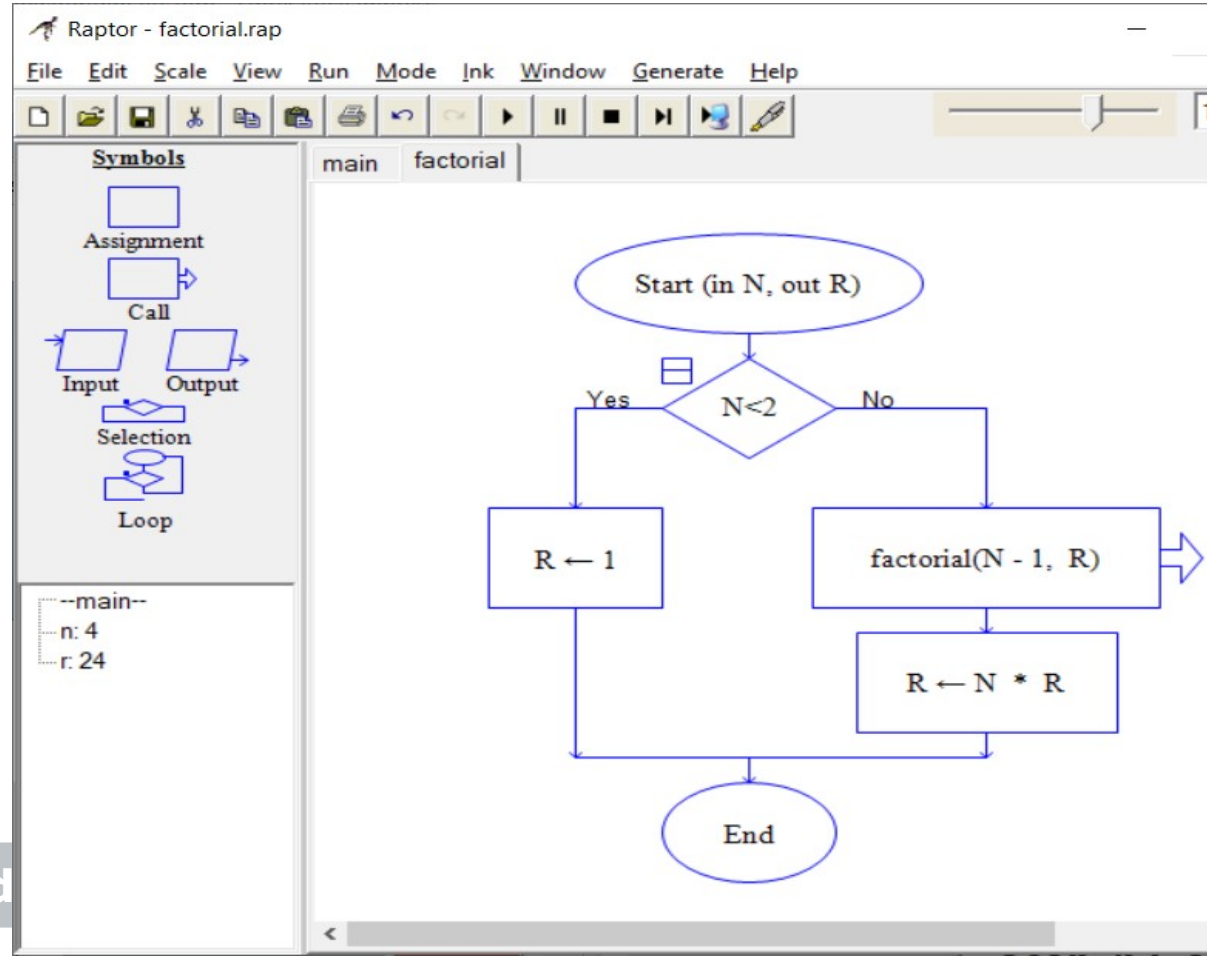
NO free text

# Examples

DEMO  
(segue)

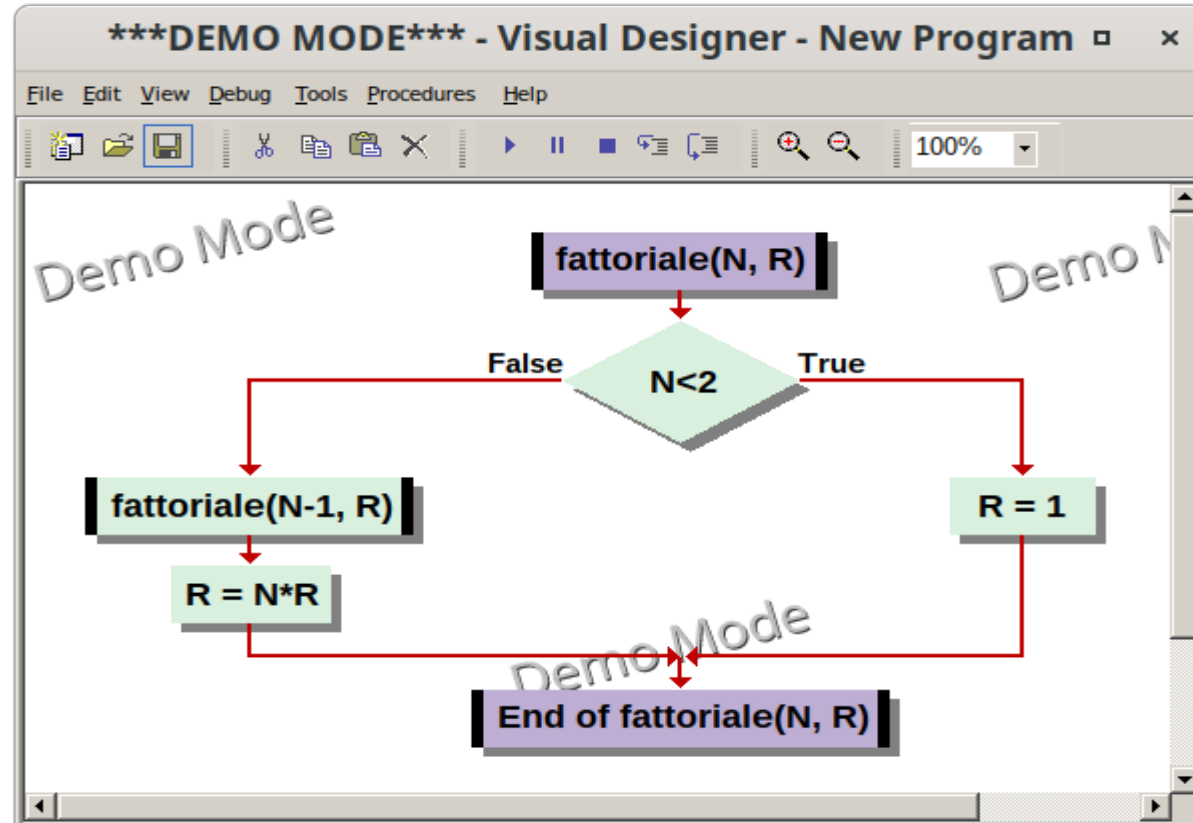
# Raptor

Procedures (with IN/OUT args)	YES
Recursion	YES
Functions (procedures + OUT args!!!)	NO?
<u>OOP</u>	<u>YES</u>
<u>Sub-charts</u>	<u>YES</u>
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation	YES
Ada, C#, C++, Java, VBA	



# Visual Logic

Procedures (with IN/OUT args)	YES
Recursion	YES
Functions (procedures + <u>OUT</u> args!!!)	NO?
OOP	NO
Sub-charts	NO
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation	YES
Visual Basic + Pascal	





# PseInt (Spanish only)

Procedures	YES
Recursion	YES
Functions	YES
OOP	NO
Sub-charts	NO
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation	YES
- C, C++, C#, Java, JavaScript, MatLab	
- Pascal, PHP, Python 2/3	
- Qbasic, Visual Basic ...	

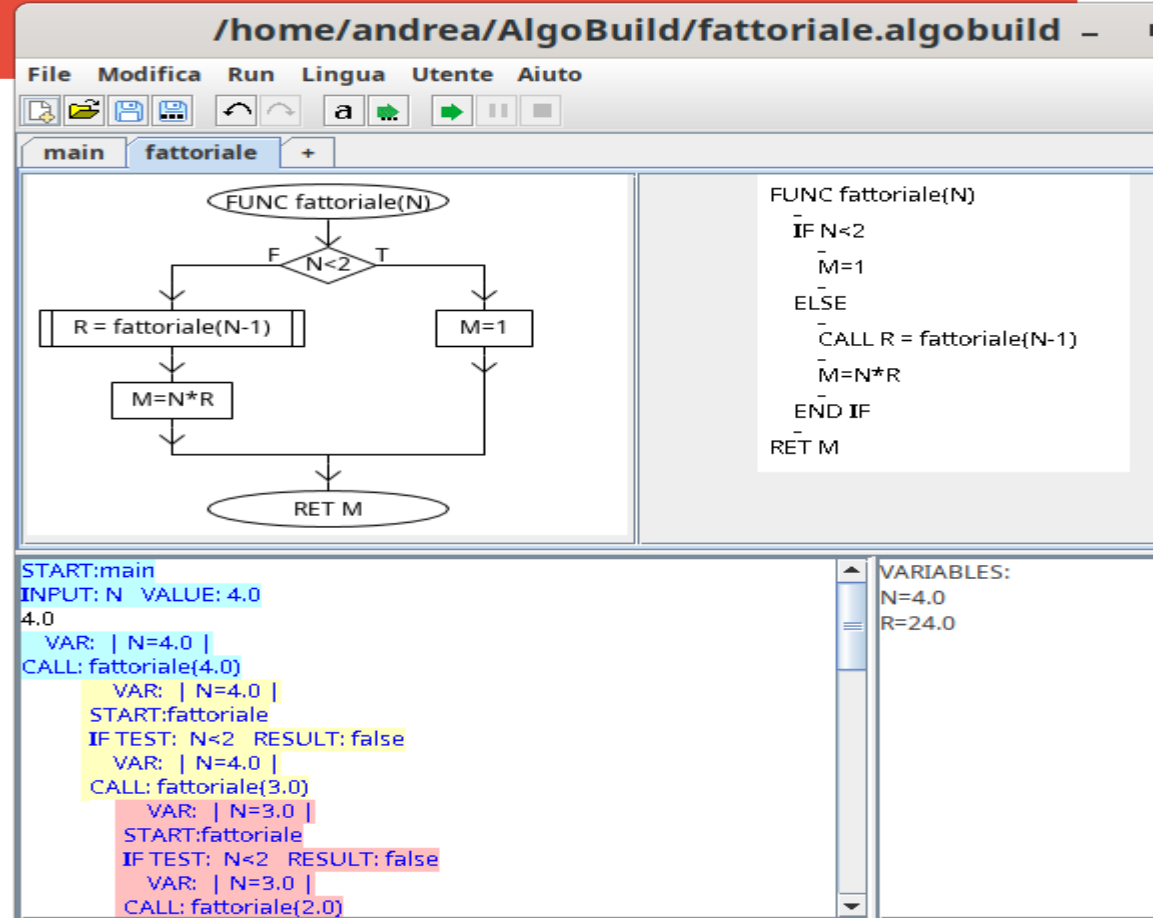
Methods in Computer Science edu

The image displays two windows from the PSeInt software suite. The top window, titled 'PSeInt', shows a code editor with a Pascal-style factorial function named 'fattoriale.psc'. The code includes a main process and a sub-process 'fattoriale' that uses recursion. The bottom window, titled 'PSDraw - main', shows a flowchart corresponding to the code in the sub-process. The flowchart starts with a call to 'SubProceso R ← fattoriale(N)', followed by a declaration 'Definir R Como Entero', a decision diamond 'N < 2', and two paths: one for 'N < 2' (labeled 'V') leading to 'R ← 1', and another for 'N < 2' (labeled 'F') leading to 'R ← N \* fattoriale(N-1)'. Both paths converge at 'FinSubProceso'.

```
graph TD
    Start([SubProceso R ← fattoriale(N)]) --> DefR[Definir R Como Entero]
    DefR --> Cond{N < 2}
    Cond -- V --> R1[R ← 1]
    Cond -- F --> R2[R ← N * fattoriale(N-1)]
    R1 --> End([FinSubProceso])
    R2 --> End
```

# AlgoBuild

Functions	YES
Recursion	YES
Simple data types	
- numbers, strings, 1D arrays	
Complex types	NO
OOP	NO
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation	NO
Nice tracing of recursion with indentation	



DEMO