

# Flowchart-based learning / 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 output (initial edge)

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

- single-thread execution (but what about fork/join?)

Many executable flowchart editors exists

- Flowgorithm [flowgorithm.org](http://flowgorithm.org)
- Algobuild [algobuild.com](http://algobuild.com)
- Raptor [raptor.martincarlisle.com](http://raptor.martincarlisle.com) (with OOP!)
- Visual Logic [visuallogic.org](http://visuallogic.org)
- PseInt [pseint.SF.net](http://pseint.SF.net) (in Spanish)
- ...

# Flowgorithm = Flow-chart + Algorithm

Executable flow-charts

Personalized flow-chart **STYLE** and **COLOURS**

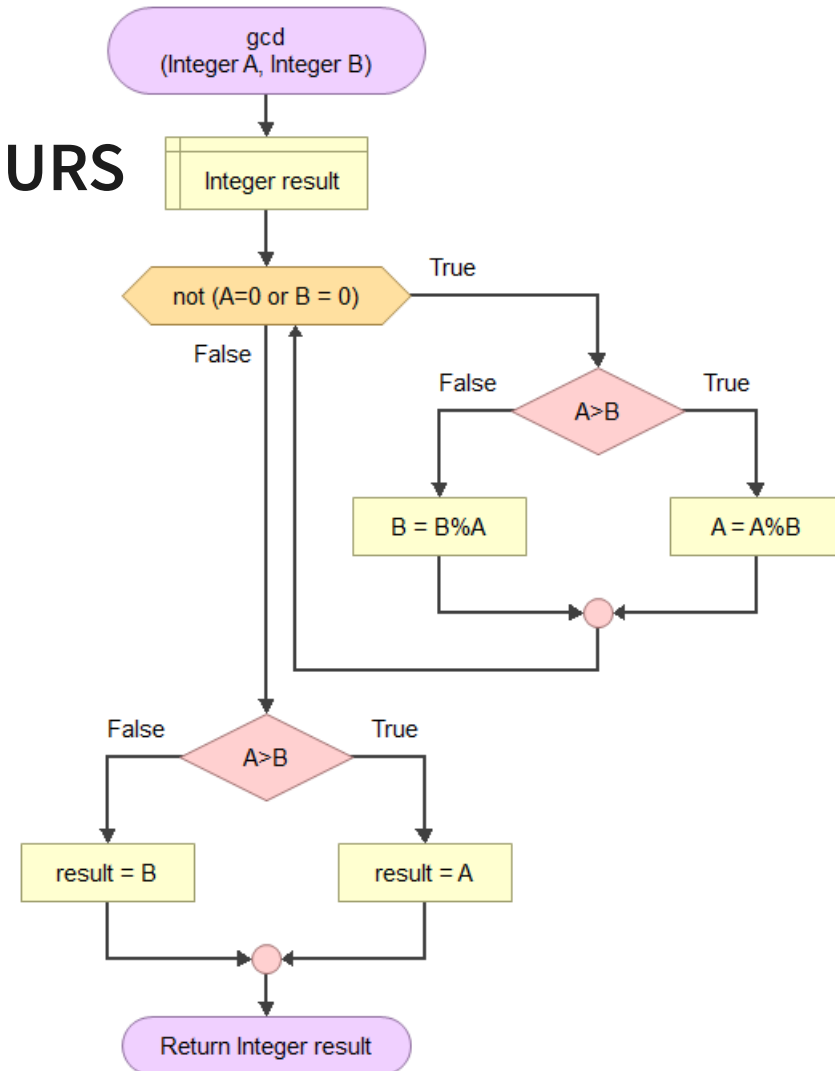
Generate your code in many languages  
(Spoken or Programmed :-)

**MISSING:** loading a program source  
and generating its flow-chart

(BUT there are tools for that)

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

- ...



# Code generation by templates

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



C#



Perl



TypeScript



C++



PHP



VBA



Fortran 2003



Powershell



Visual Basic .NET



Java



Python



Gaddis Pseudocode



JavaScript



QBasic



IBO Pseudocode



Lua



Ruby



Auto Pseudocode



MATLAB



Scala



Open...



Nim



Smalltalk



Pascal



Swift

# Example template: Python

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

The program is a template with required imports and missing functions (you can extend it if you like)

Types are mapped to corresponding Python types

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

Functions definition and call templates

Diagram elements map to corresponding templates

**DEMO**

# Simple Data types (and arrays)

T = Integer, Float, String, Boolean

1 dimensional Array of <T>

NO bigintegers (Python)

NO lists or dynamic arrays

NO heterogeneous arrays

NO multidim. arrays

NO objects

NO coroutines

NO function objects

NO files

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 variable

ASSIGN variable

INPUT

OUTPUT

IF

CALL procedure/function

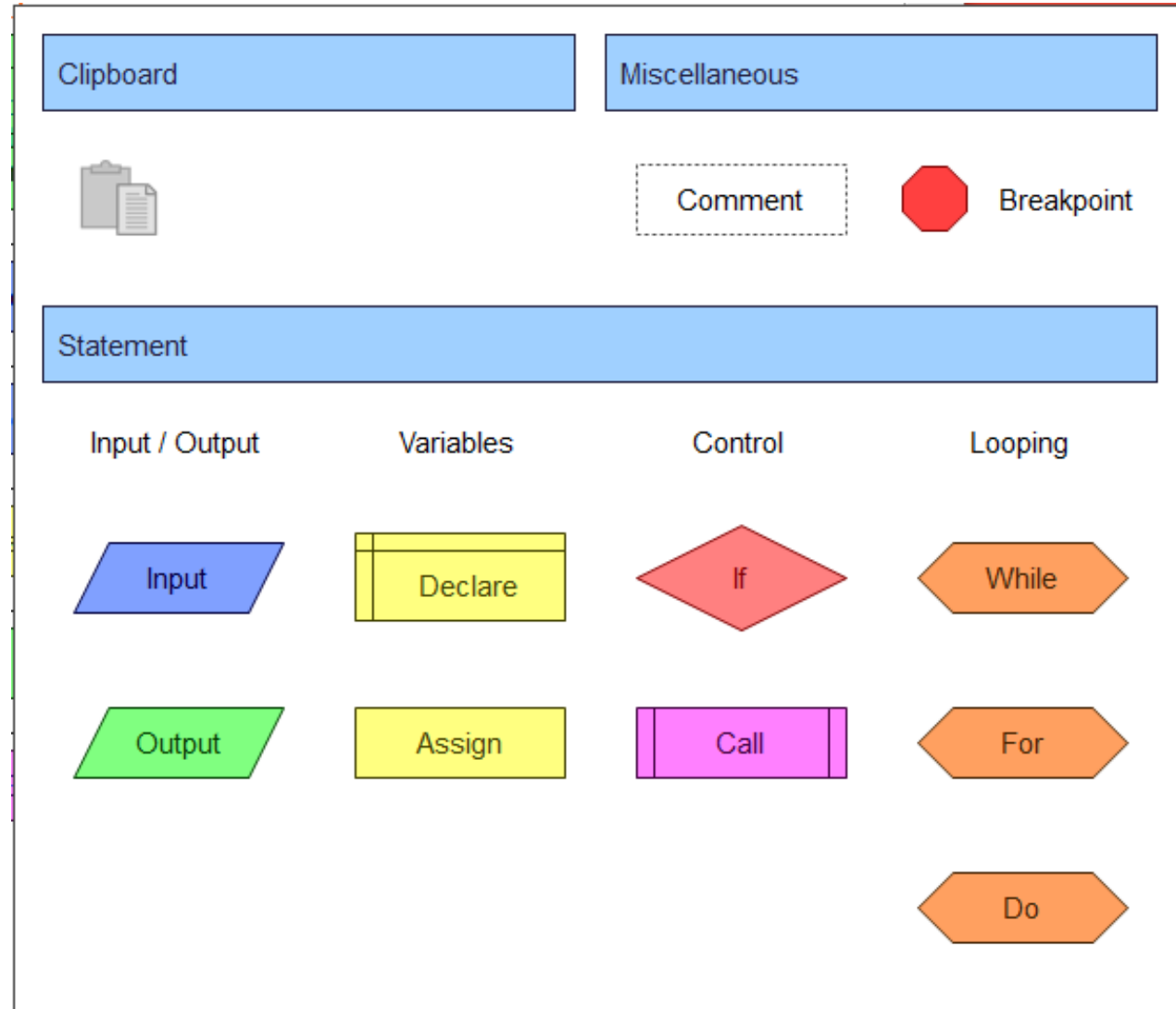
WHILE-do

counted FOR

DO-while

(NO foreach)

COMMENTS



# 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

Precedences as usual



# Control flow

Functions?	YES
args by reference?	NO (except for arrays like C)
multiple return values?	NO (single simple types only)
<u>ONE entry and ONE exit</u> 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
Recursion?	YES
Exceptions?	NO

# 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)

Breakpoints

Assertions? (by hand)

Exceptions? NO

IDE support

Refactoring PARTIAL (cut/paste into new functions)

# Literate programming / Documentation?

**Program properties:**

**Title, Author, Description**

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

**Comments in the flow-chart**

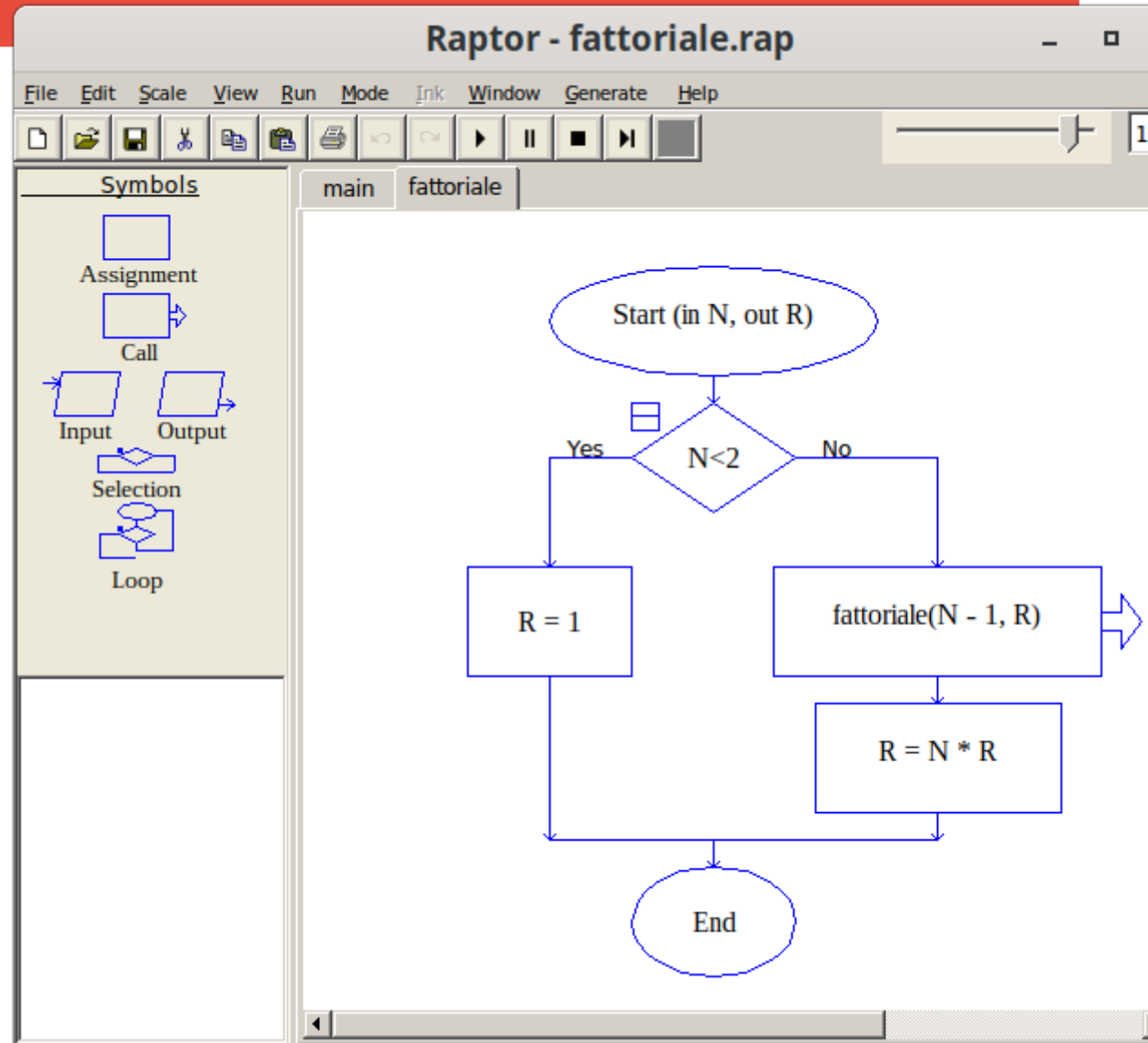
**NO free text**

# Examples

**DEMO**

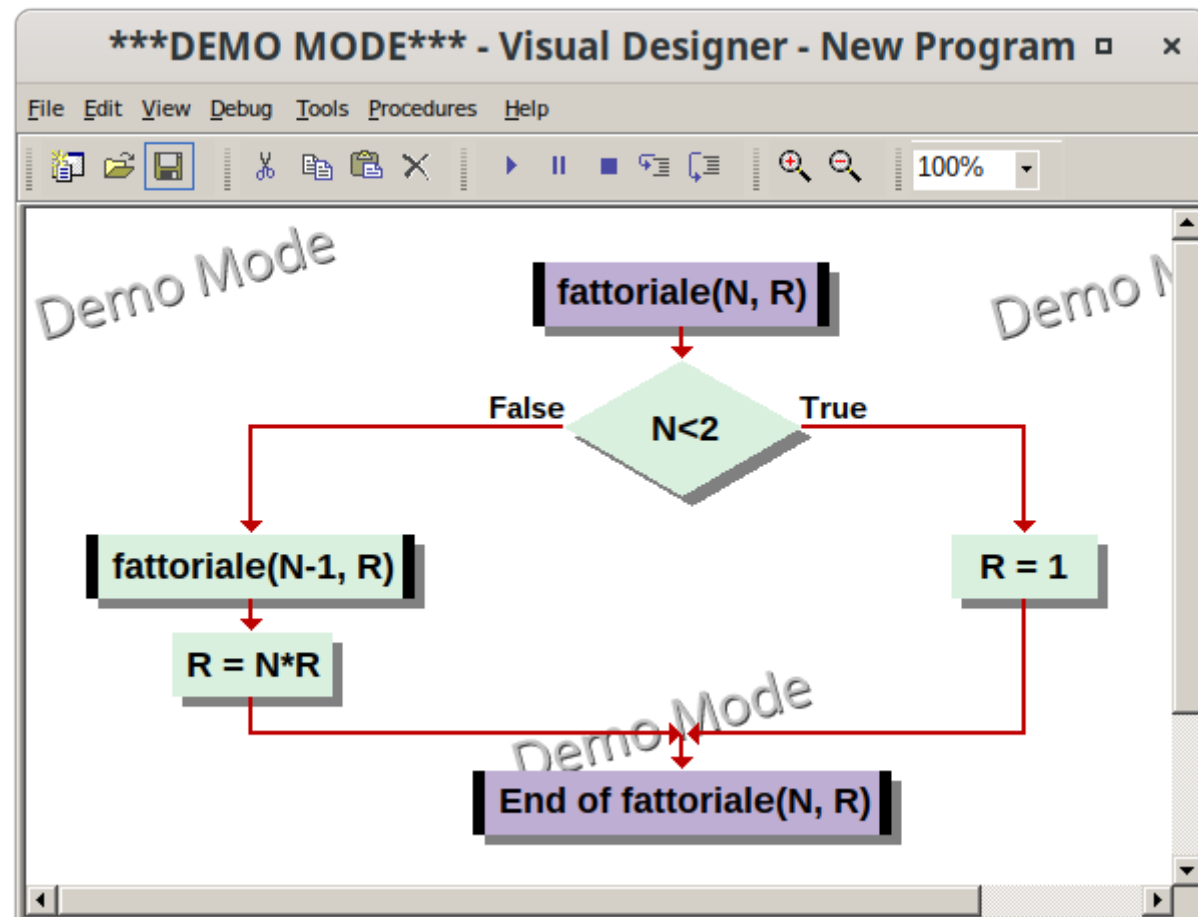
# Raptor

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 + OUT args)	NO?
OOP	NO
Sub-charts	NO
Concurrency	NO
Events	NO
Step-by-step debug	YES
Code generation VB + Pascal	YES



# 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 IDE. The top window, titled 'PSeInt', shows a code editor with a Pascal-like implementation of a factorial function. The code is as follows:

```
1 Proceso main
2   Definir N Como Entero;
3   Leer N;
4   Escribir fattoriale(N);
5 FinProceso

7 SubProceso R <- fattoriale(N)
8   Definir R Como Entero;
9   Si N<2 Entonces
10      R <- 1;
11   SiNo
12      R <- N*fattoriale(N-1);
13   FinSi
14 FinSubProceso
```

The left sidebar of the PSeInt window shows a 'Variables' panel with the following entries:

- Sb fattoriale
- 42 N
- 42 R
- Po main
- N

The bottom window, titled 'PSDraw - main', shows a flowchart corresponding to the code above. The flowchart starts with a process box 'SubProceso R <- fattoriale(N)', followed by 'Definir R Como Entero', then a decision diamond 'N<2'. If the condition is true (V), it goes to 'R <- 1'. If false (F), it goes to 'R <- N\*fattoriale(N-1)'. Both paths lead to 'FinSubProceso'.



# 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

The screenshot shows the AlgoBuild interface for a factorial function. The window title is `/home/andrea/AlgoBuild/fattoriale.algobuild`. The menu bar includes `File`, `Modifica`, `Run`, `Lingua`, `Utente`, and `Aiuto`. The toolbar contains icons for file operations and execution. The main workspace is divided into three panels:

- Flowchart:** A flowchart for the function `FUNC fattoriale(N)`. It starts with a decision diamond `N<2`. If true (T), it sets `M=1`. If false (F), it calls `R = fattoriale(N-1)`, then calculates `M=N*R`. Both paths lead to `RET M`.
- Code:** The following code is displayed:

```
FUNC fattoriale(N)
- IF N<2
- M=1
- ELSE
- CALL R = fattoriale(N-1)
- M=N*R
- END IF
- RET M
```
- Debug Log:** A step-by-step execution log for `START:main` with `INPUT: N VALUE: 4.0`. It shows recursive calls: `CALL: fattoriale(4.0)`, `CALL: fattoriale(3.0)`, and `CALL: fattoriale(2.0)`. The log also shows the state of variables `N` and `R` at each step.
- VARIABLES:** A panel on the right shows the current state: `N=4.0` and `R=24.0`.

**Demo**

**DEMO**