

Flowchart-based programming

By Devin Cook of Sacramento State Univ.



Andrea Sterbini – 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

Many executable flowchart editors

- Flowgorithm flowgorithm.org

- Algbuild algbuild.com

- Raptor raptor.martincarlisle.com (with OOP!)

- Visual Logic visuallogic.org

- PseInt pseint.SF.net (in Spanish)

- ...

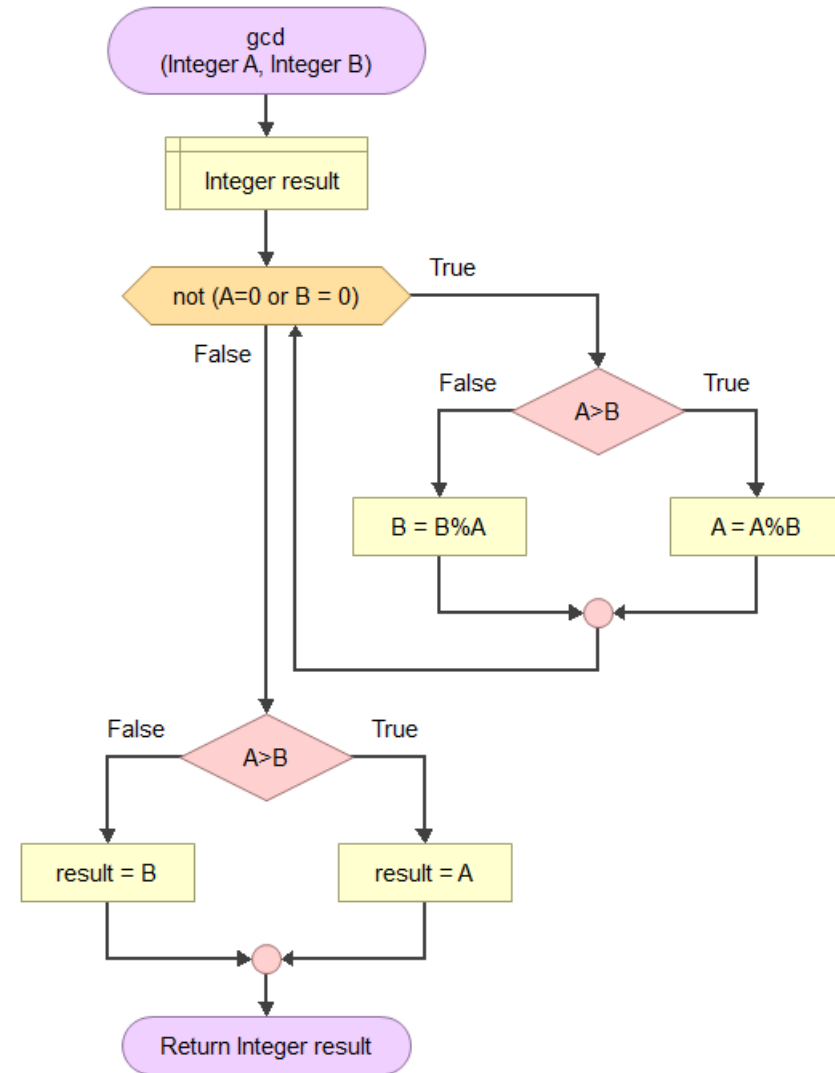
Flowgorithm = Flow-chart + Algorithm

Executable flow-charts

Personalized flow-chart STYLE and COLOURS

Generate your code in many languages

MISSING: load a program and generate its flow-chart



Code generation by templates

Code generation
from flow-charts
to many
programming
languages
(even personalized)



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

Simple Data types (and arrays)

T = Integer, Float, String, Boolean

Array of <T>

NO bigintegers

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

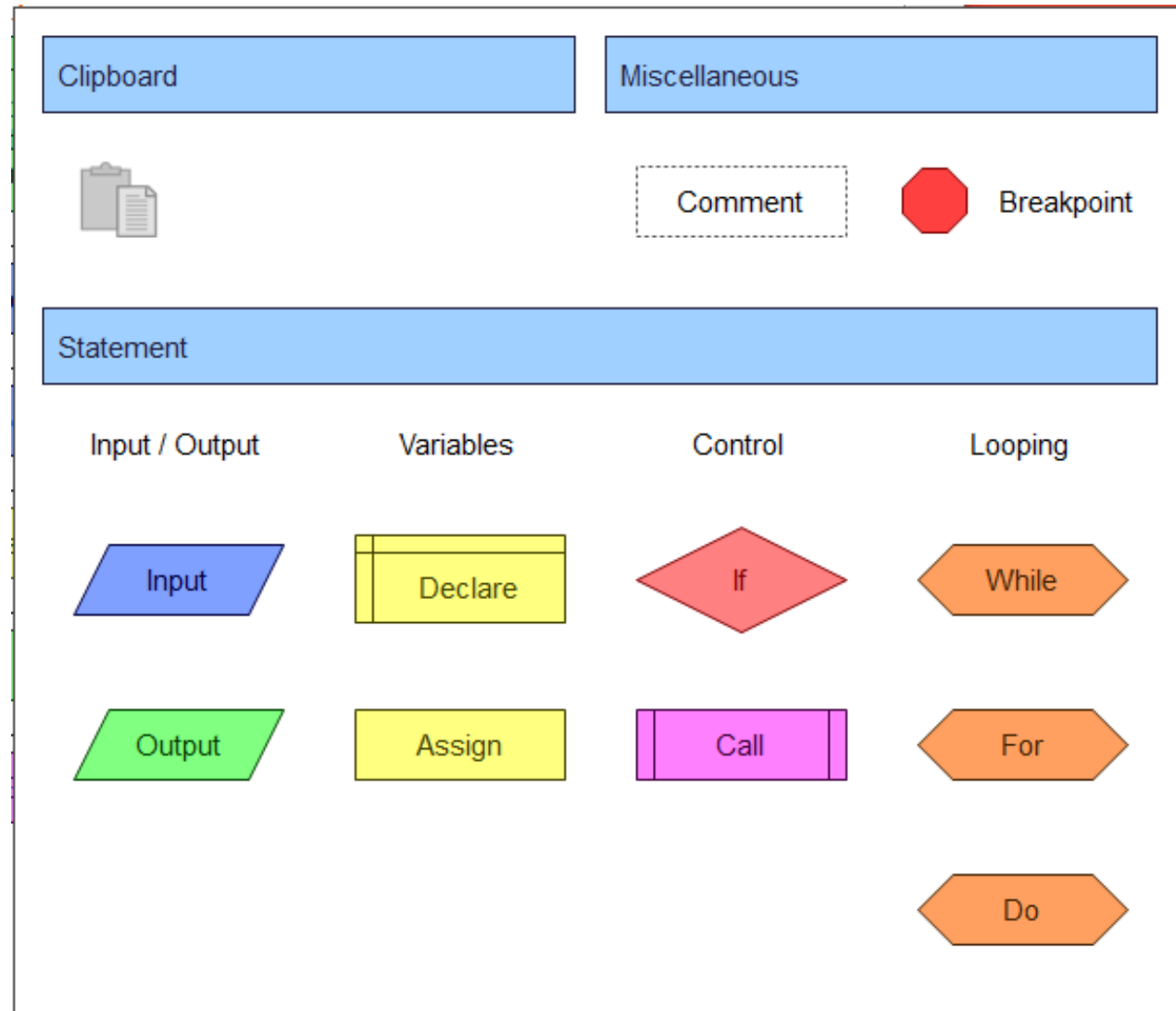
WHILE-do

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)
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

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

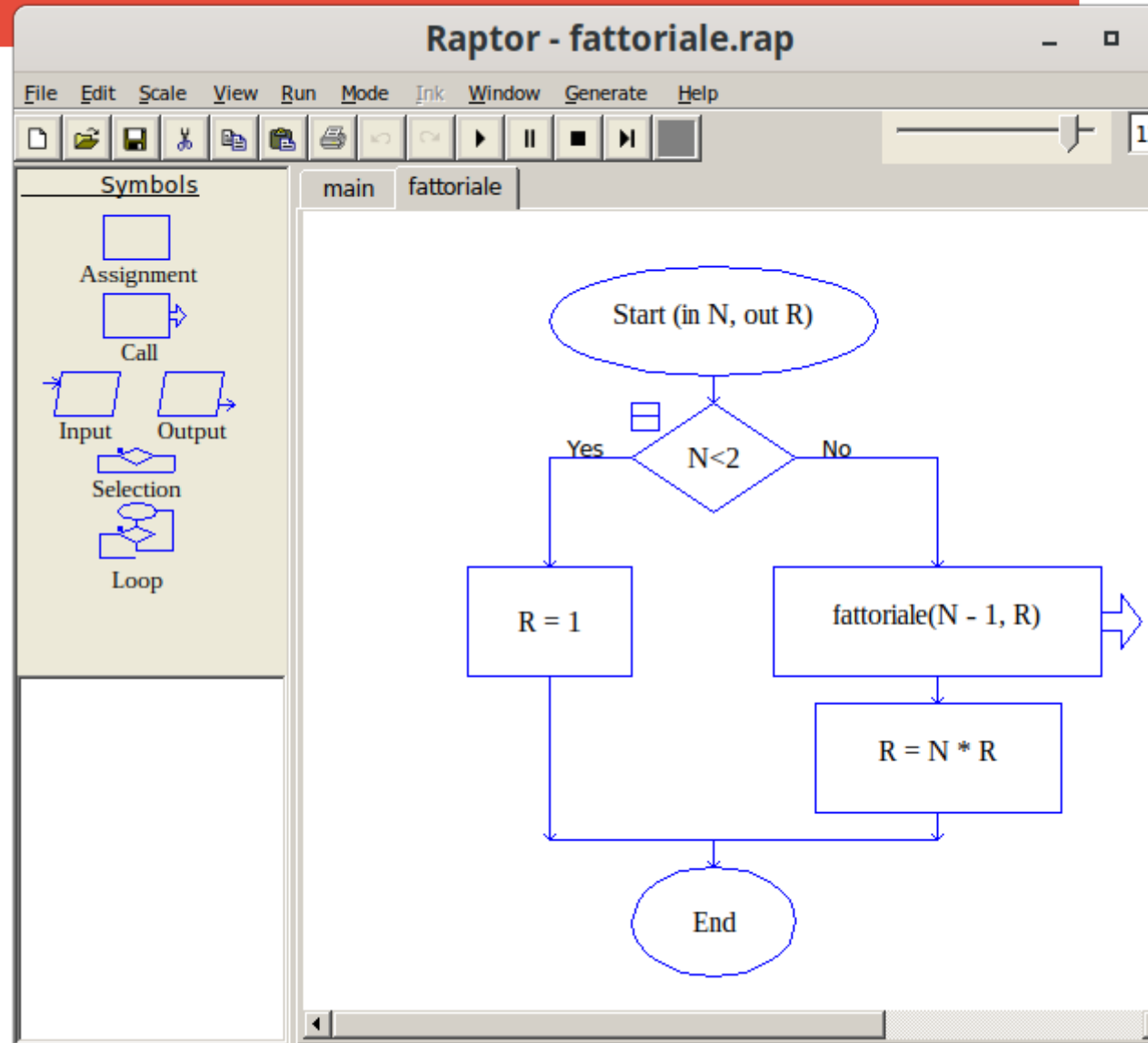
BUT: they are NOT present in the generated code!!!

Comments in the flow-chart

NO free text

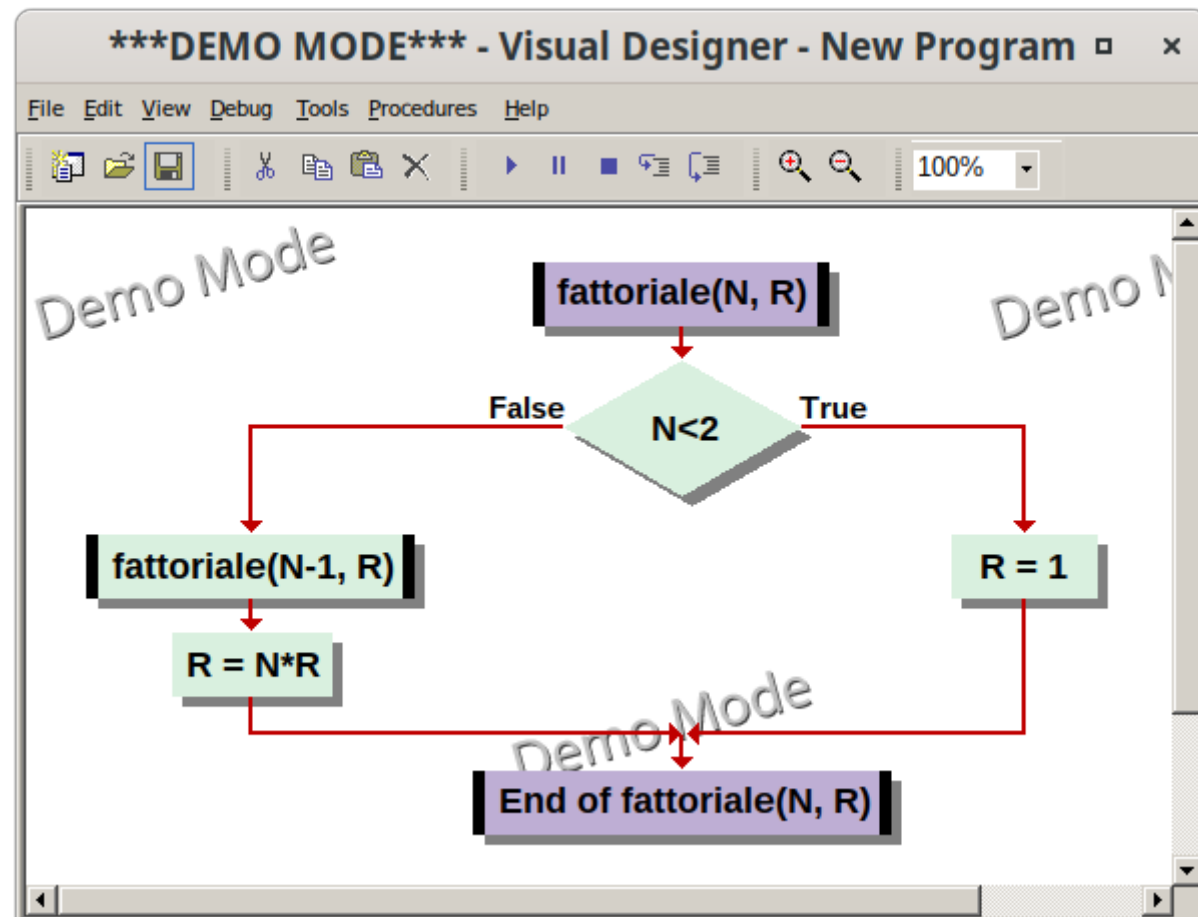
Raptor

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

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. 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 that visualizes the logic of the 'SubProceso R <- fattoriale(N)' function. The flowchart consists of the following steps:

- Start: SubProceso R <- fattoriale(N)
- Process: Definir R Como Entero
- Decision: N < 2 (True/False)
- If True (V): Process R <- 1
- If False (F): Process R <- N * fattoriale(N-1)
- End: 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 application window titled "/home/andrea/AlgoBuild/fattoriale.algobuild". The interface includes a menu bar (File, Modifica, Run, Lingua, Utente, Aiuto) and a toolbar with icons for file operations and execution. The main workspace is divided into three sections:

- Flowchart:** A flowchart for the function "FUNC fattoriale(N)". It starts with a decision diamond "N<2". If true (T), it proceeds to a process box "M=1". If false (F), it proceeds to a call box "R = fattoriale(N-1)", then to a process box "M=N*R". Both paths merge and lead to an output oval "RET M".
- Code:** A text editor showing the implementation of the function:

```
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 the main program with input N=4.0:

```
START:main  
INPUT: N VALUE: 4.0  
4.0  
VAR: | N=4.0 |  
CALL: fattoriale(4.0)  
  VAR: | N=4.0 |  
  START:fattoriale  
  IF TEST: N<2 RESULT: false  
  VAR: | N=4.0 |  
  CALL: fattoriale(3.0)  
    VAR: | N=3.0 |  
    START:fattoriale  
    IF TEST: N<2 RESULT: false  
    VAR: | N=3.0 |  
    CALL: fattoriale(2.0)
```
- Variables:** A panel on the right showing the current state of variables:

```
VARIABLES:  
N=4.0  
R=24.0
```

Demo

DEMO