

# Programació Bàsica

## UF1 - PROGRAMACIÓ ESTRUCTURADA

### INTRODUCCIÓ

Què és un programa?

```
#include <stdio.h>
int main ()
{
    printf ("Hello World");
    return 0;
}
```

→ Conjunt d' instruccions de tipus lòic executades de manera seqüencial i seguint un ordre per tal d' obtenir un resultat.

→ Una instrucció és un conjunt de dades dins d' una seqüència més en el processador realitzar una acció.

Com s' executa un programa?

→ introducció de dades.

→ Tècnic

→ Bases de dades

→ fitxes

- El processador executa les instruccions.
- Els resultats s' emmagatzemaràn a la RAM.
- Els resultats finals s' emmagatzemaran al disc dur.

## LLENGUATGE DE PROGRAMACIÓ

Llenguatge

- **Compilat** → Una vegada escrit l'algorisme del programa (utilitzant un llenguatge d'alt nivell) es tradueix per un compilador a un llenguatge comprensible per a la màquina.
- **interpretat** → Es aquell en el qual el codi font, (escrit en un llenguatge d'alt nivell) es tradueix per un interpret a llenguatge comprensible per a la màquina (no ens retorna un executable) {és més lent que el compilat}.

Nivells d'abstracció

Hi ha dos nivells d'abstracció, com més alt, més llontan és al nostre llenguatge.

→ Alt nivell → JAVA, C, C++, Python, ...

→ Nivell intermig → } llenguatge originaire } Ensemblejar (en funció de la CPU)

→ Baix nivell → llenguatge binari.

## ALGORITMES

Què és una boba?

→ representació simbòlica d'una entitat

Què és un tipus de dades?

→ són de dades amb les que pot treballar una variable

simples	numeriques	enters (1, 5, 22) (semejant decimal)
		reals (2, 45 / 28, 9)
	no numeriques	booleans (TRUE / FALSE)
		caràcters.
estructurables	internes	estàtiques (registres, vectors, taules).
		dinàmiques (llistes, cues, piles, arbres).
		fixes
	externes	bases de dades

enters → qualsevol nombre natural entre el 0 i l' $\infty$ .

reals → qualsevol nombre entre el - $\infty$  i el  $\infty$

booleans → TRUE o FALSE / 0 o 1

caràcters → conjunt de símbols o lletres.

Què és una expressió?

Una expressió és la representació de diversos operands, que units entre ells mitjançant operadors, per a realitzar una acció sobre ells.

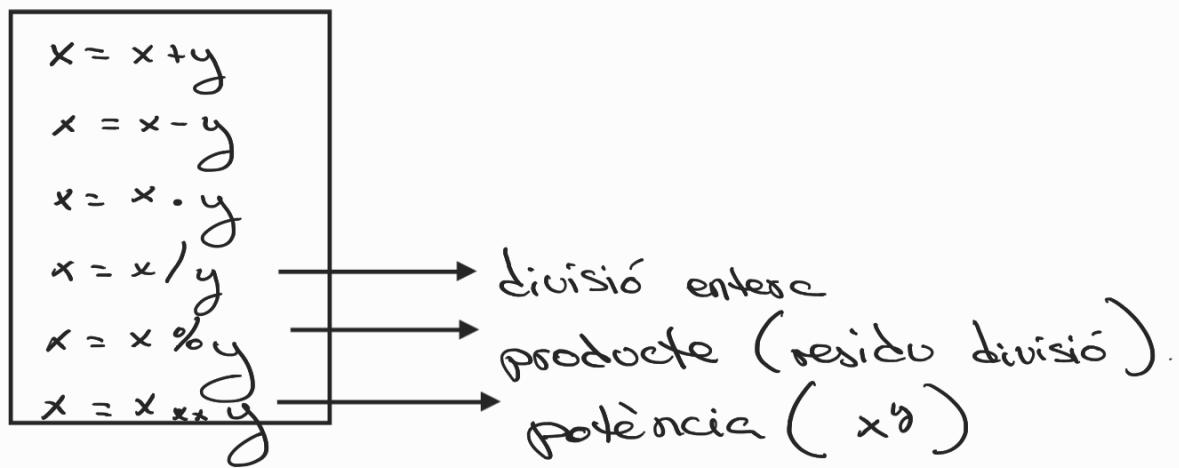
$s_3 == 4$   
 $s / 4.0$   
 $!(40 > 25)$   
 $10 < 20 \&& 40 > 25$   
 $s! = 6$

## Que és una variable?

Una **variable** és un dada emmagatzemada a la memòria que pot ser modificat el seu valor en qualsevol moment de l'execució.

Una **constant** és una variable que NO modifica el seu valor durant l'execució del programa.

## Operadors



## COMPARACIÓ

$x == y$   
 $x != y$   
 $<, >$   
 $\leq, \geq$

TRUE / FALSE

## ARITMÈTICA

$++ \rightarrow$  increment  
 $-- \rightarrow$  decrement  
 $* * \rightarrow$  potència

## Operacions i ordre

l'ordre de la prioritat de

1	parentesi	( )	les operacions són el següent.
2	càrrega de signe	- ( ) ! ( )	
3	Operadors exponents	$x^a$	$**$
4	producte, divisió i mòdul	$*(\text{div}) / \% (\text{mod})$	
5	suma i resta	+ -	
6	relacions de comparació	> < $\leq$ $\geq$	
7	Negació lògica	NOT ! (unari)	
8	Conjunció	AND ( && )	
9	disjunció	OR (    )	

## ALGEBRA DE BOOLE

### Negació (NOT)

$$0 \rightarrow 1$$

$$1 \rightarrow 0$$

### Suma Lògica (OR)

$$0 + 0 \rightarrow 0$$

$$0 + 1 \rightarrow 1$$

$$1 + 0 \rightarrow 1$$

$$1 + 1 \rightarrow 1$$

### Multiplicació Lògica (AND)

$$0 \times 0 \rightarrow 0$$

$$0 \times 1 \rightarrow 0$$

$$1 \times 0 \rightarrow 0$$

$$1 \times 1 \rightarrow 1$$

## EXERCICI 1

A	B	$A == B$	$A > B$	$A < B$	$A \geq B$	$A \leq B$
4	3	F	T	F	T	F
14	-2	F	T	F	T	F
-78	34	F	F	T	F	T
12	12	T	F	F	T	T

## DÀDES SIMPLES

→ dades que només tenen 1 valor.

Commentaris

entrada

primitius

sortida

assignació

de control

alternatives  
(condicionals)

simples

dobles

múltiples

repetitives

(iteratives).

# ALGORITMES

## ETAPES DEL DISENY DEL PROGRAMA

1. **ANALISI** → Estudi del problema
2. **DISENY** → Passos que haurà de seguir l'algoritme.
3. **PROGRAMACIÓ** → Implementació del pseudocodi a llenguatge d'alt nivell
4. **COMPILACIÓ** → Compilació del codi a alt nivell a llenguatge mègic.
5. **EXECUCIÓ I PROVES** → Execució del codi amb diferents entrades per a comprobar en directe els resultats.

Un algoritme ha de ser:

- Precís
- ben definit
- Finit
- Robust
- Transportable

## EXEMPL

Un autobús viaja a 126 km/h per l'autopista 45 min. Quina distància ha recorregut?

ENTRADA → posició inicial ( $x_0 = 0$ ), velocitat ( $v = 126$ ) i temps ( $t = 45$ )

PROCESS → aplicar la fórmula ( $x_f = x_0 + v \cdot t$ ) convertint tots els dades al sist. universal.

SORTIDA → 94.500 m

ERROS → No hi ha

## EXERCICI 2

Estic a l'aeroport i pego a una cinta transportadora. La cinta fa 60 m. de llarg i triga mig

minut en arribar. A quina velocitat va la cinta?

ENTRADA  $\rightarrow$  60 m / 0'5 min

PROSES  $\rightarrow$  sabem que  $v = \frac{x_0(m)}{t(s)} = \frac{60\text{ m}}{30\text{ s}} = 2\text{ m/s}$

SORTIDA  $\rightarrow$  velocitat = 2 m/s

ERROR  $\rightarrow$  NO



NOTA:

(es correcte en mode científic però  
en pseudocodi fa fata pres text)

## DISSENY

- s'escriu de manera precisa les tasques a realitzar pel program
- s'utilitza la tècnica "top-down" (de dalt a baix)

### EXEMPLG 2

Exemple exercici d'exemple

Nivell 1      1. determinar la distància que ha recorregut un bus en 45 min.

Nivell 2      2.1. determinar dades necessaries.  
                  2.2. Calcular distància  
                  2.3. Indicar resultat a l'usuari

Nivell 3      2.1.1. Assignar dades a variables  
                  2.2.1. Convertir dades a sistema internacional  
                  2.2.2. Aplicar la fórmula  
                  2.3.1. Mostrar el resultat en metres.

## EXERCICI 3

Anunciat exercici 2.

Nivell 1

1. La cinta fa 60 metres de llarg; triga més minut en arribar a l'altre banda. A quina velocitat viafja sobre la cinta?

Nivell 2.

2.1. Determinar dades.

2.2. Calcular velocitat

2.3. Indicar resultat

Nivell 3

2.1.1. Assignar dades a variables.

2.2.1. Convertir dades a sistema universal

2.2.2. Aplicar la fórmula

2.3.1. Mostrar el resultat

## ESPECIFICACIÓ

Pseudocodi → descripció formal d'un algoritme (d'alt niv.)

function

constant

### EXEMPLE

REAL PI;

end constant

var

INTEGER radius;

REAL area;

endvar

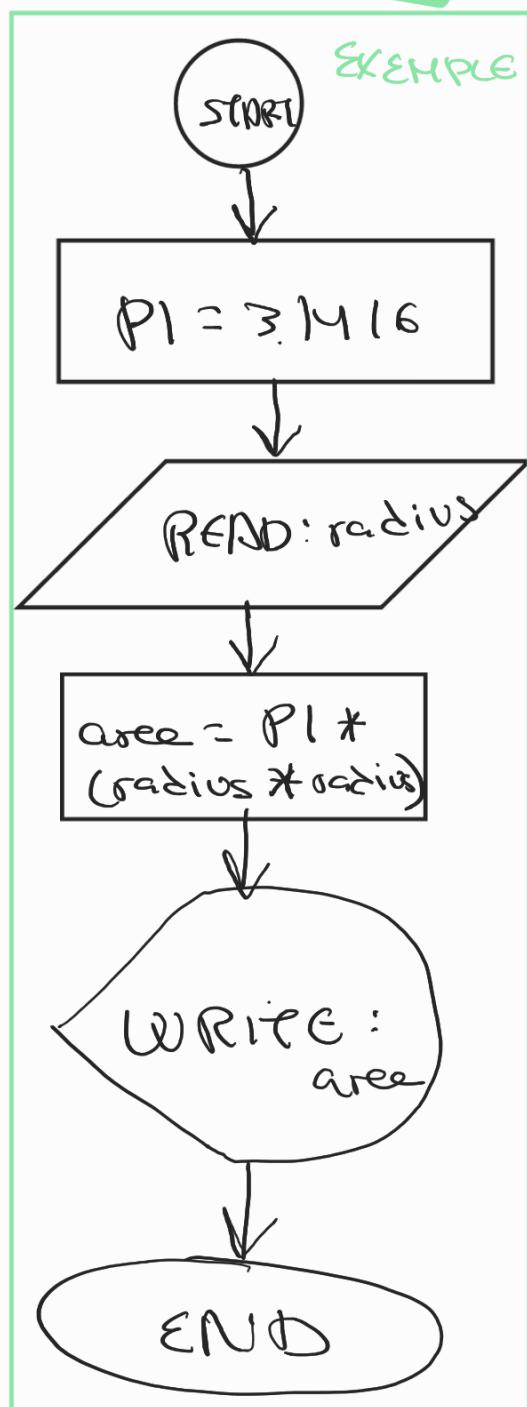
READ radius;

area := PI \* (radius \* radius);

WRITE area;

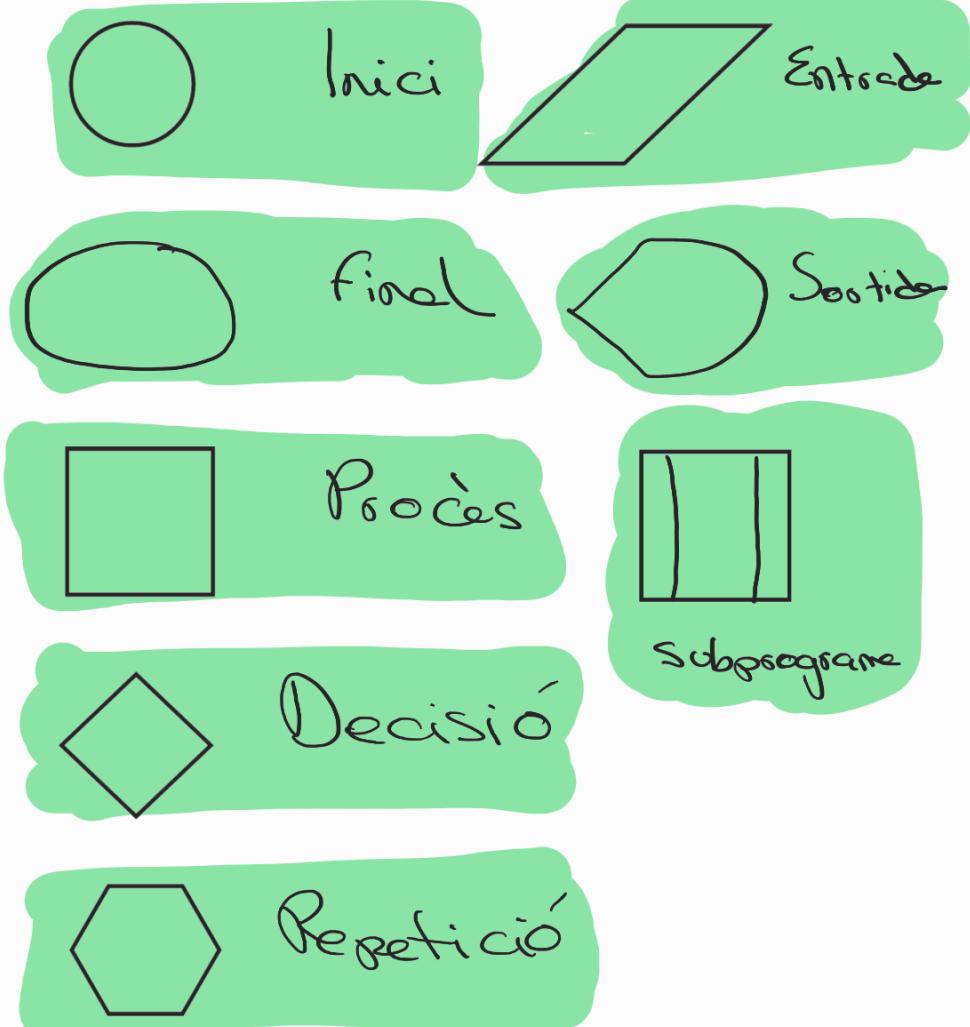
end function

Diagramme de flux → representació gràfica d'un algoritme



## SÍMBOLES

→ ordre de l'algoritme



## ESTRUCTURES DE SELECCIÓ

Controlen la seqüència d'execució d'altres instruccions, segons una determinada condició. Podeu ser simples, múltiples o dobles.

simples

## SIMPLES

function

var

INTEGER age;

endvar

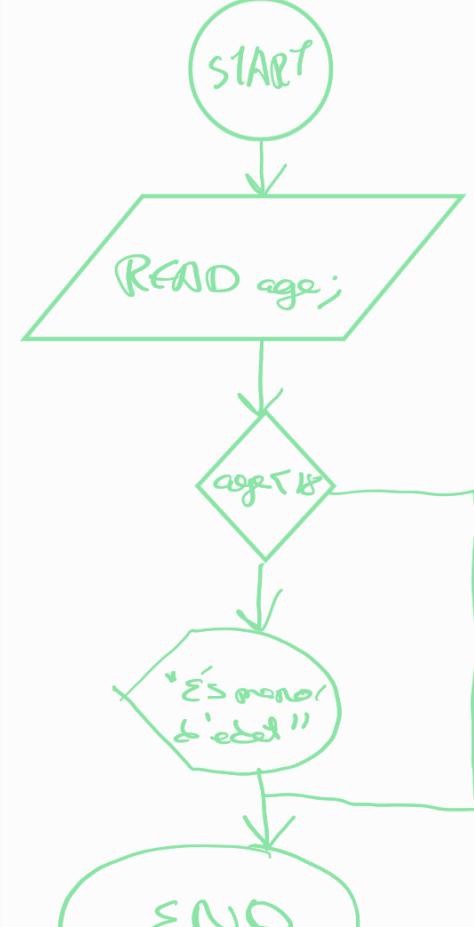
READ age;

if age < 18 then

WRITE "Es menor d'edat";

endif

endfunction



## DOBLES

function

var

INTEGER age;

endvar

READ age;

if age < 18 then

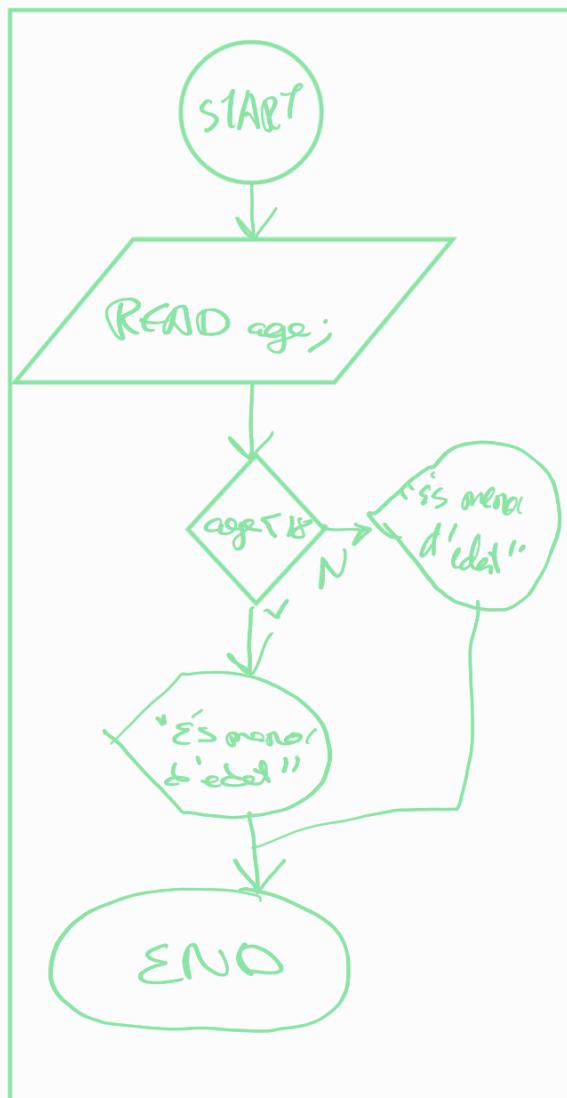
WRITE "Es menor";

else

WRITE "Es mayor d'edat";

endif

endfunction

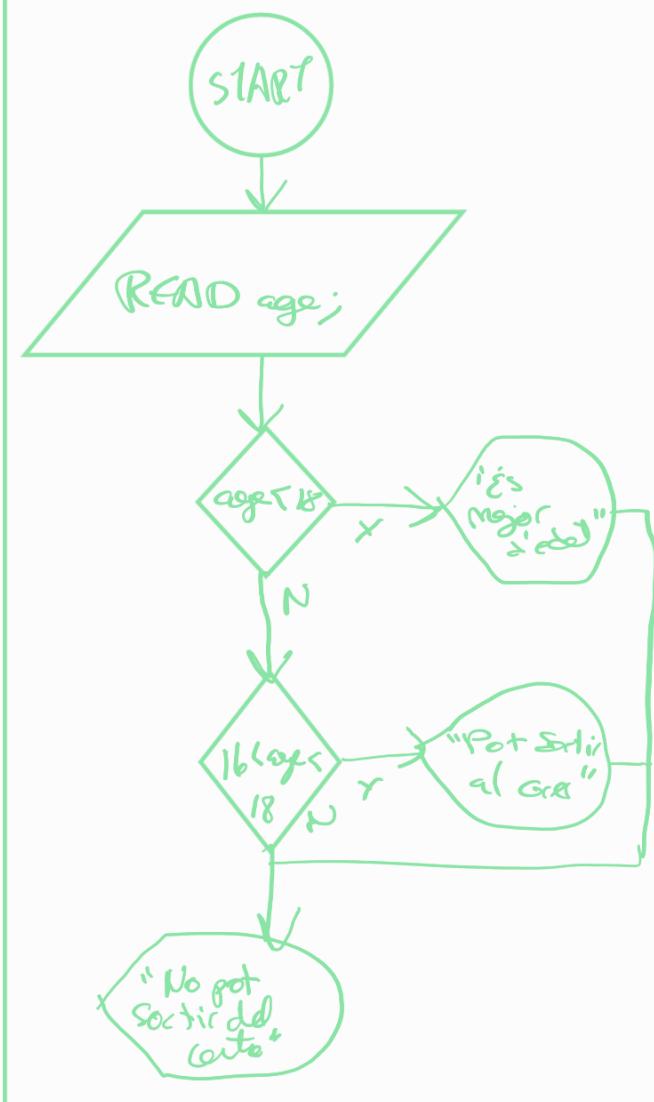


## MULTIPLES

function

junction  
var

```
    INTEGER age;  
    entvar  
    READ age;  
    if age < 18 then  
        WRITE "Es menor";  
    else  
        if age < 18 AND age > 16 then  
            WRITE "Pot sortir al centre";  
        else  
            WRITE "No pot sortir del centre";  
        endif  
    endif  
endfunction
```



## ESTRUCTURES DE REPETICIO

Els bucles o iteracions es repeteixen un cert nombre de cops dins d'un algoritme.

**While (munta)** → les operacions s'executen mentre es compleix la condició (sigui certa):

```
while count <= 10 do:  
    WRITE count;  
    count := count + 1;  
endwhile
```

En el diagrama s'unirà el principi i el final del bucle amb una línia discontinua

**Do...while (repetir fins)** → s'executaran les operacions una primera vegada i després

s'avaluarà si es compleix la condició:

```
do  
    WRITE count;  
    Count := Count + 1;  
    while Count <= 10;
```

En el diagrames de flux es representa mitjançant una línia discontinua des de la condició fins al principi de les accions.

**For (par)** → s'executará tantes vegades com indica el comptador:

```
for Count := 1 to 10 do  
    WRITE Count;  
    Count := Count + 1;  
endfor
```

En el diagrames de flux es representa amb una línia discontinua des de les accions fins abans del comptador.

## PRECONDICIÓ I POSTCONDICIÓ

**Precondició** → condició que ha de satisfer-se just abans d'executar el codi.

**Postcondició** → condició que ha de donar-se immediatament després de l'execució del codi.

Poc. rep 2 nombres enters. el divisor ha de ser diferent de 0  
Post. retorna el cocient.

## JOC DE PROVES

El joc de proves documenta els casos i els resultats que om dengui.

CAS

ENTRADA

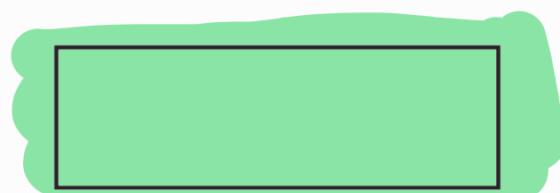
SORTIDA

CORRECTE?

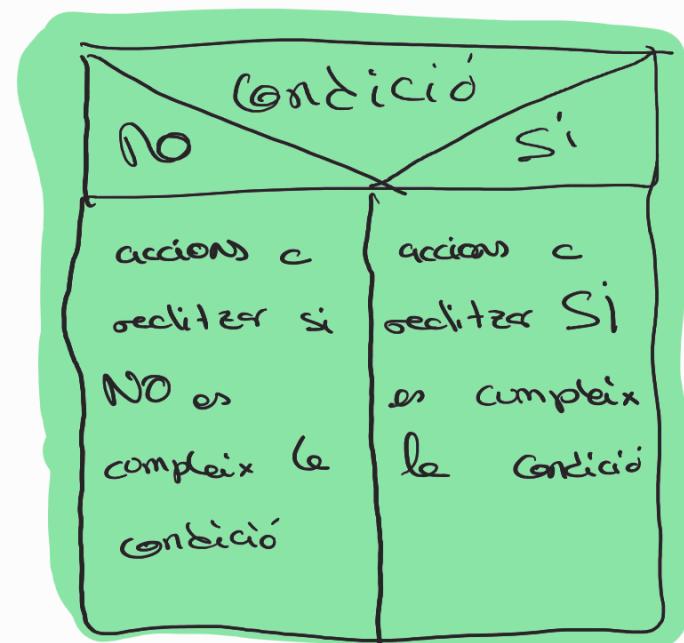
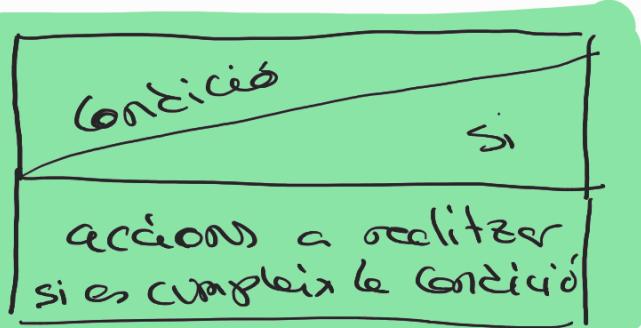
1	age = 15	"menor edad"	si
2	age = 21	"mayor edad"	si
3	age = 12	"menor edad"	si
4	age = 5	"menor edad"	si
5	age = 18	"mayor edad"	si

## DIAGRAMA DE NASSI SCHNEIDERMAN

Descripció d'una tasca:



Instruccions alternatives:



Variable =

valor1	valor2	valor3	valor4	...	alternativa
--------	--------	--------	--------	-----	-------------

acció acció acció acció acció.

mentre condició fer

accions a repetir

while (mentre)

accions a repetir

mentre condició

do ... while (repetir fins)

per ini = val, final, passos a fer

accions a repetir

for (per)

## EXEMPLE

Donat un nombre  
introduït per teclat,  
mostre per pantalla  
si és positiu, negatiu

inici

LEGIR n

n == 0

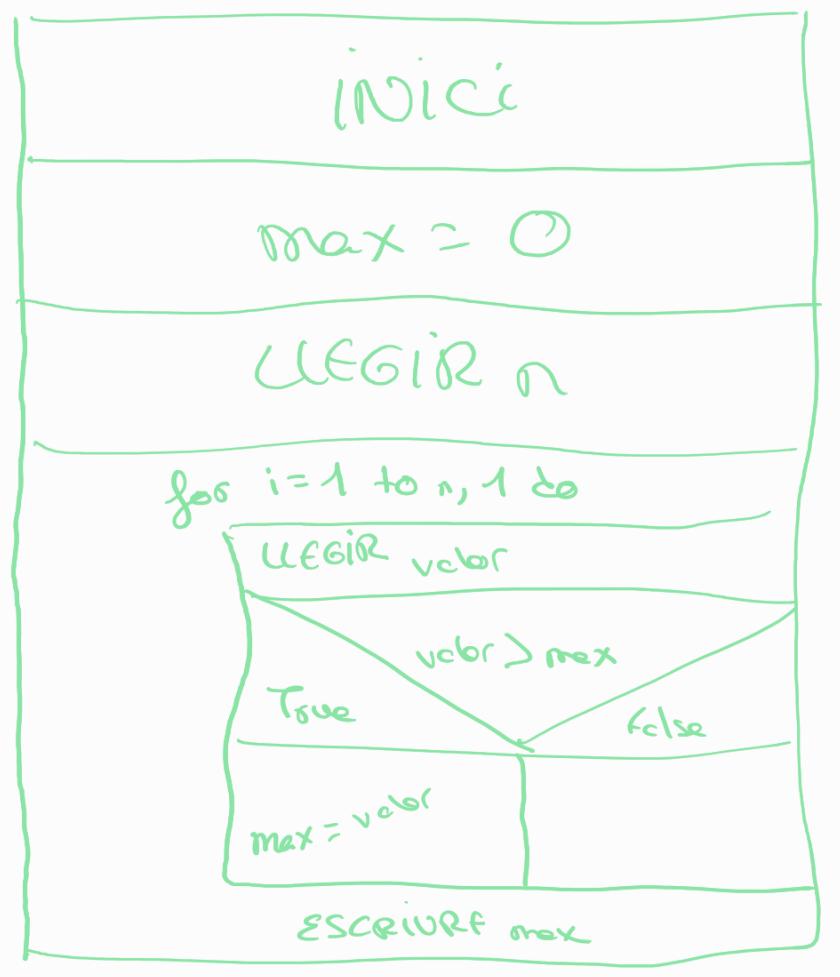
No

si



## EXEMPLE

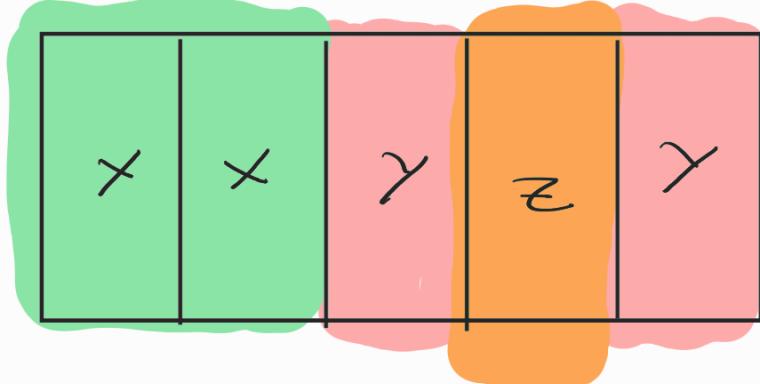
Trobar i mostrar el màxim d'un conjunt de  $N$  números ( $N \geq 2$ ) introduits per teclat.



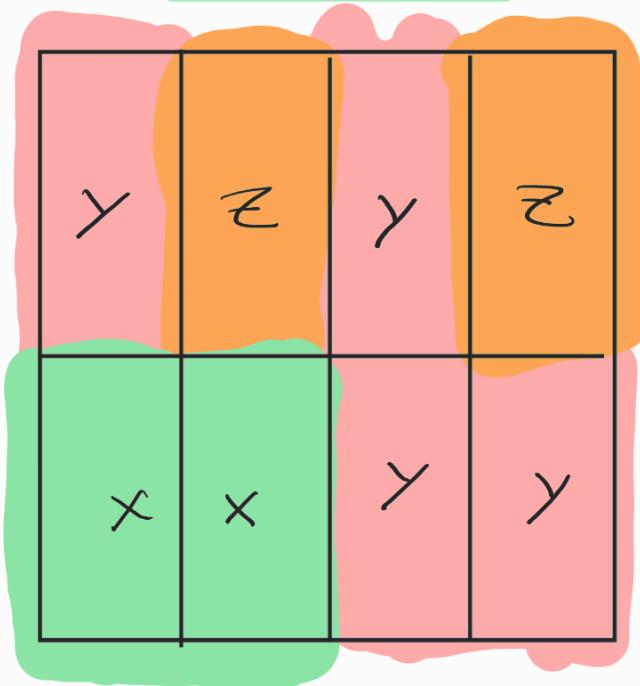
tipus de variable estructurada és aquella formada per diferents vèlors de tipus primitiu i/o compostos.

variables estructurades } arrays } Los 2 són estrutges ja que hi assignen un tamany fixo  
matrizes } matrizes } no un valor.

### ARRAY



### MATRIZ



L'array (vector) permet emmagatzemar en forma de seqüència, una quantitat predefinida de vèlors pertanyents al mateix tipus de dades.

INTEGER arrayInt [S]; sense iniciar  
INTEGER arrayInt [S] = {10, 20, 30, 40, 50};

### EXAMPLE

Algorisme que emmagatzeme 6 vèlors naturals introduïts per telet en un array:

function

constant

```
INTEGER size := 6;
```

```
endconstant
```

```
var
```

```
INTEGER arrayInt [size], i := 0;
```

```
endvar
```

```
while i < size do
```

```
    WRITE "Introdueix un nombre";
```

```
    READ arrayInt [i];
```

```
    i := i + 1;
```

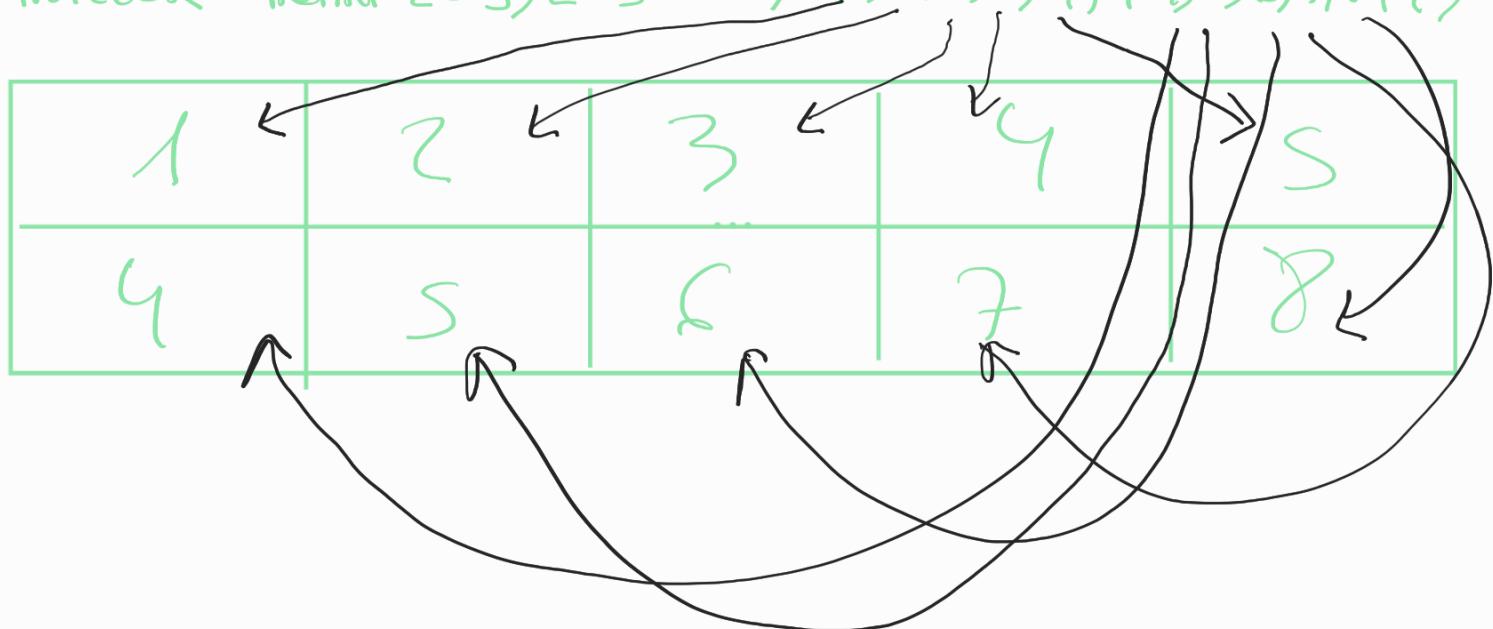
```
endwhile
```

```
endfunction
```

Una matrícula permet emmagatzemar en forma de taula, una quantitat predeterminada de valors pertanyents al mateix tipus de dades.

```
INTEGER matInt [2], [5]; // sense inicialitzar
```

```
INTEGER matInt [2], [5] := { {1, 2, 3, 4, 5}, {6, 7, 8} };
```



Algorisme que emmagatzeme les notes dels 6 mòduls professionals de 4 alumnes per teclat.

function

constant

INTEGER ROWS := 4, Cols := 6;

endconstant

var

INTEGER notes [ROWS] [Cols], i := 0, j := 0;

endvar

for i := 0 to ROWS do

    for j := 0 to Cols do

        READ notes [i] [j];

        j := j + 1;

    endfor

    i := i + 1;

endfor

endfunction