



# TEORIA

## PROGRAMACIÓ CIENTÍFICA

---

T3

Estructures de control




# Estructures de control

## Què són?

- Les instruccions dels algorismes s'executen de forma **seqüencial** (i.e., una després de l'altra)

```
const ... fconst
var ... fvar
inici
    escriure ("Introdueix el valor del radi:");
    llegir (radi);
    perim := 2 * PI * radi;
    escriure ("El perímetre és ", perim);
    area := PI * radi * radi;
    escriure ("L'àrea és ", area);
...
```

Seqüència  
d'instruccions



# Estructures de control

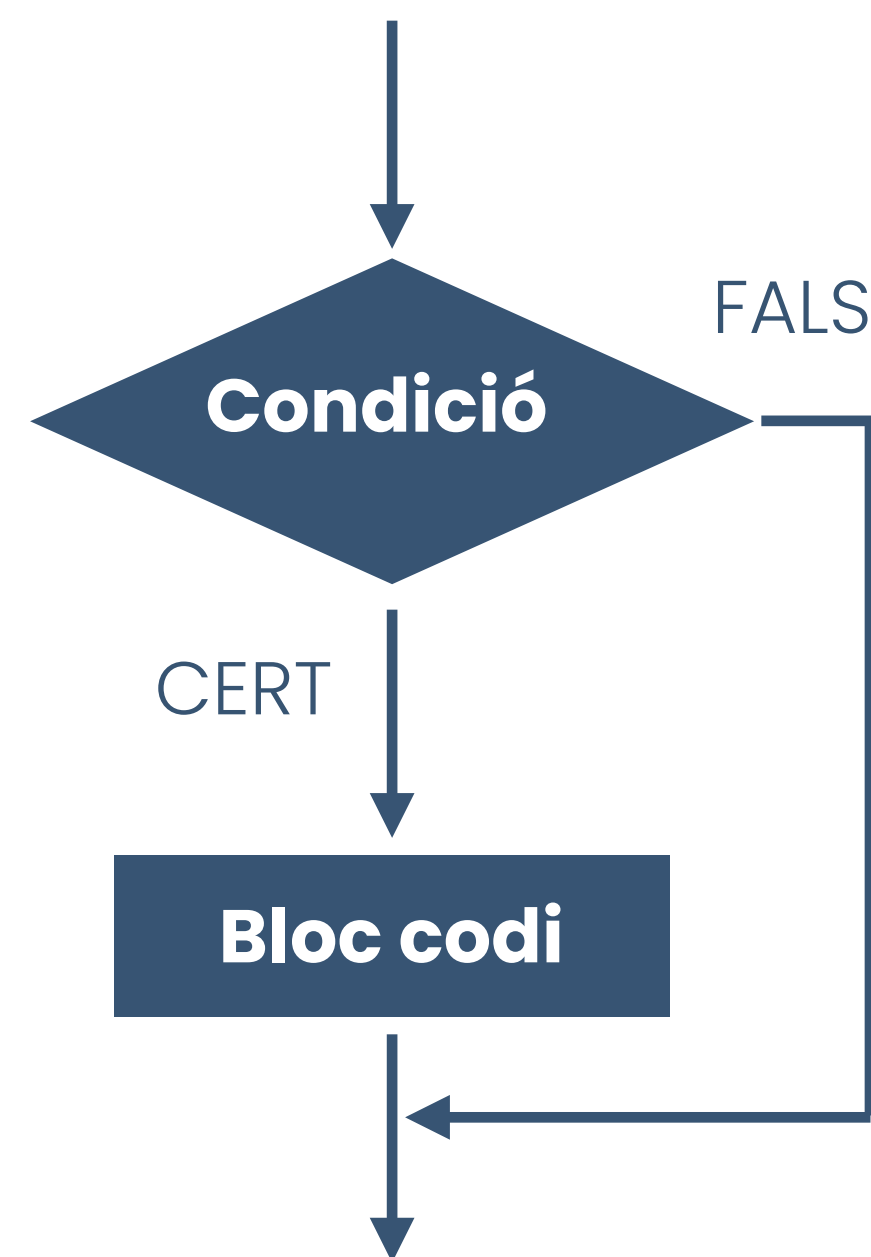
## Què són?

- Les instruccions dels algorismes s'executen de forma **seqüencial** (i.e., una després de l'altra)
- La programació estructurada disposa de blocs que permeten controlar el flux d'execució. Són les **estructures de control**:

# Estructures de control

## Què són?

- Les instruccions dels algorismes s'executen de forma **seqüencial** (i.e., una després de l'altra)
- La programació estructurada disposa de blocs que permeten controlar el flux d'execució. Són les **estructures de control**:
  - El **condicional** o selecció és una estructura de control que permet l'execució d'instruccions segons si es compleixen o no unes condicions.



*Estructura condicional*

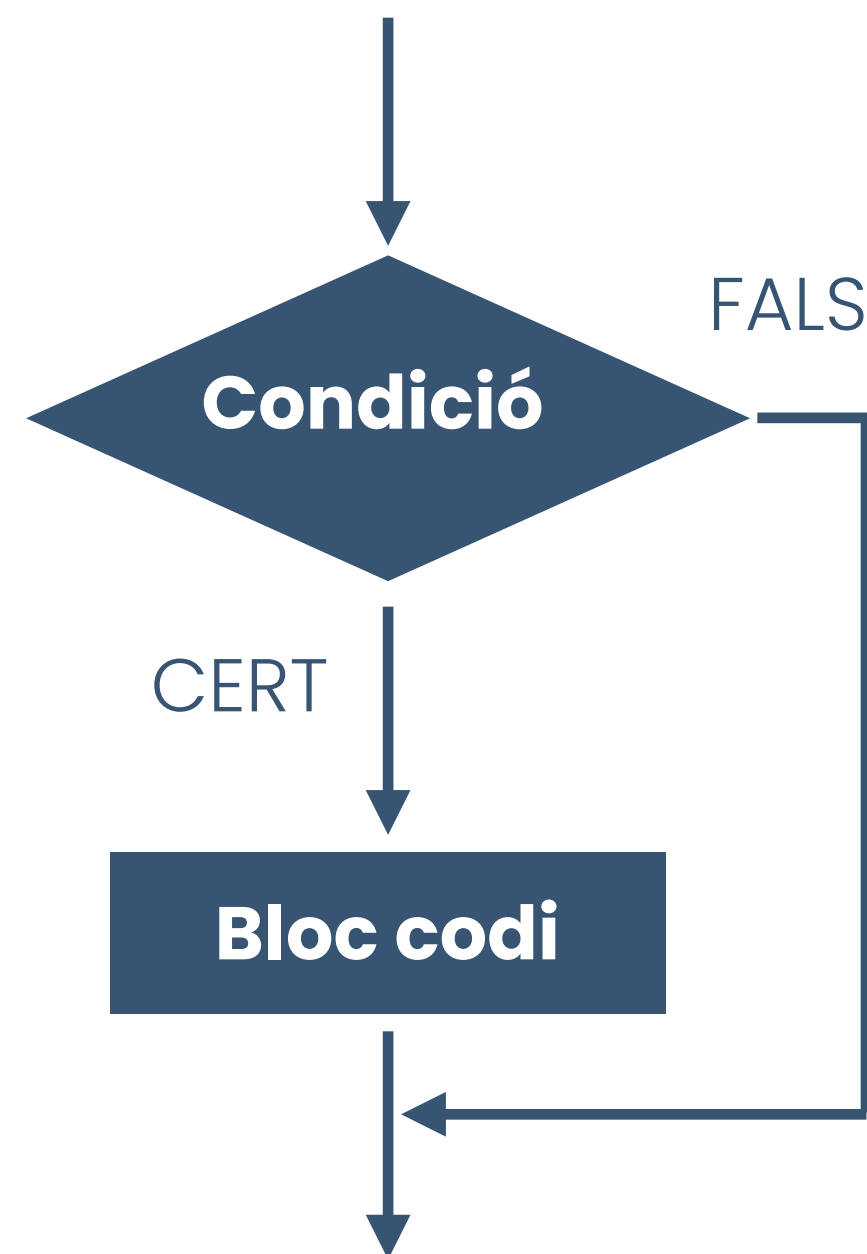
- Tipus:
  - Condicional simple
  - Condicional doble
  - Condicionals anidats
  - Condicional múltiple

# Estructures de control

## Què són?

- Les instruccions dels algorismes s'executen de forma **seqüencial** (i.e., una després de l'altra)
- La programació estructurada disposa de blocs que permeten controlar el flux d'execució. Són les **estructures de control**:

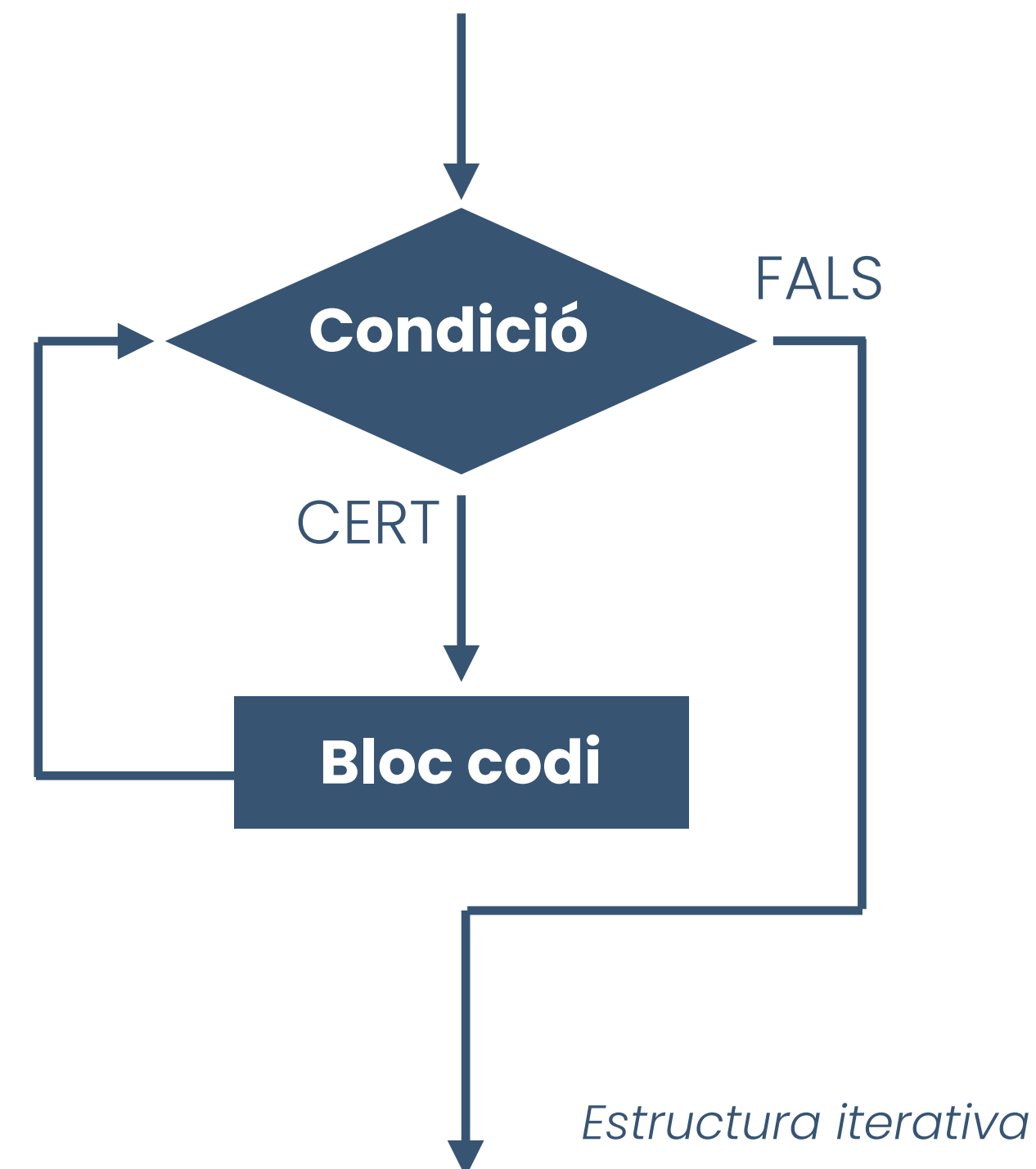
- El **condicional** o selecció és una estructura de control que permet l'execució d'instruccions segons si es compleixen o no unes condicions.



*Estructura condicional*

- Tipus:
  - Condicional simple
  - Condicional doble
  - Condicionals anidats
  - Condicional múltiple

- El **bucle** (o iteració o loop) permet executar instruccions un determinat nombre de vegades o mentre es compleixi una condició.



*Estructura iterativa*

# CONDICIONALS

# Conditionals

## Conditional simple

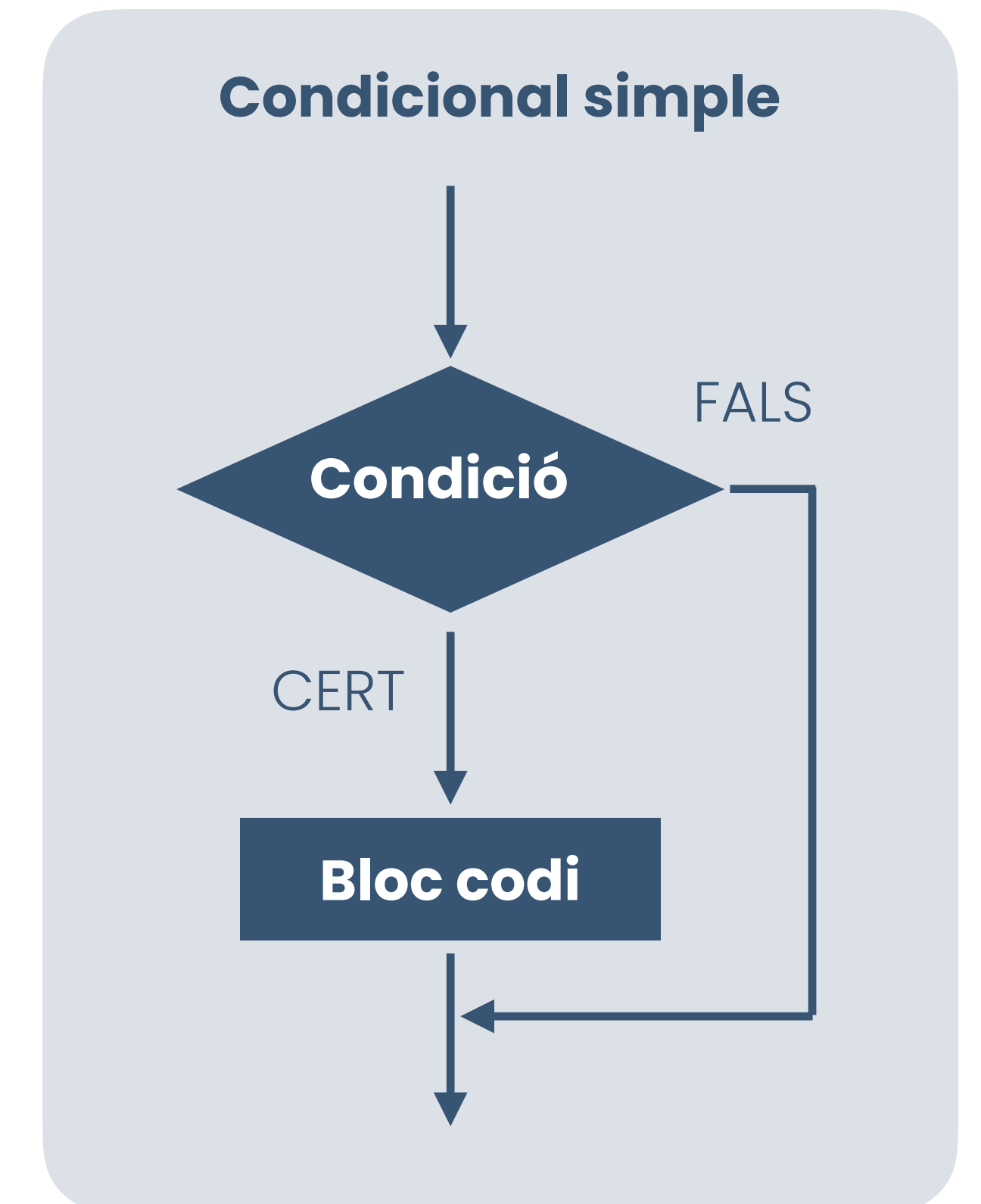
- Els conditionals permeten controlar l'execució de sentències a partir de l'avaluació de condicions.
- Pseudocodi del condicional simple:

```
si (condició) llavors  
    instrucció1;  
    instrucció2;  
    ...  
fsi  
... 
```

La condició és una expressió lògica.

Si es compleix la condició  
s'executen aquestes  
instruccions.

Si no es compleix la condició,  
se salta a aquest punt de  
l'algorisme.



# Conditionals

## Conditional **simple**

- Exemple: Dissenya un algorisme per a obtenir el valor absolut d'un nombre enter introduït per teclat
  - Nota: sense fer servir la funció "valor absolut", com calculem el valor absolut d'un nombre enter?

```
algorisme valor_absolut és
inici
    Obtenir valor
    Calcular valor absolut
    Mostrar resultat
falgorisme
```



# Conditionals

## Conditional **simple**

- Exemple: Dissenya un algorisme per a obtenir el valor absolut d'un nombre enter introduït per teclat
  - Nota: sense fer servir la funció "valor absolut", com calculem el valor absolut d'un nombre enter?

```
algorisme valor_absolut és
inici
    Obtenir valor
    Calcular valor absolut
    Mostrar resultat
falgorisme
```

## Solució

```
algorisme valor_absolut és
var x: enter; fvar
inici
    escriure ("Introdueix el nombre");
    llegir(x);
    si (x < 0) llavors
        x := -x;
    fsi
    escriure ("El valor absolut és ", x);
falgorisme
```

# Condicionals

## Condicional **doble**

- En el condicional doble, si no es compleix la condició, també executem un bloc de codi.
- Pseudocodi del condicional doble:

```
si (condició) llavors  
    instrucció1;  
    instrucció2;  
    ...  
sino  
    instrucció3;  
    instrucció4;  
    ...  
fsi  
...
```

Noteu com fem ús de la **indentació**. Millora la llegibilitat del pseudocodi (i del codi en els llenguatges de programació).

Si no es compleix la condició, s'executaran aquestes instruccions alternatives. Noteu que utilitzem 'sino' com a contracció de 'si no' en pseudocodi. En algunes fonts hi trobareu 'altrament'

# Condicionals

## Condicional **doble**

- En el condicional doble, si no es compleix la condició, també executem un bloc de codi.
- Pseudocodi del condicional doble:

```
si (condició) llavors
```

```
    Codi A
```

```
    ...
```

```
sino
```

```
    Codi B
```

```
    ...
```

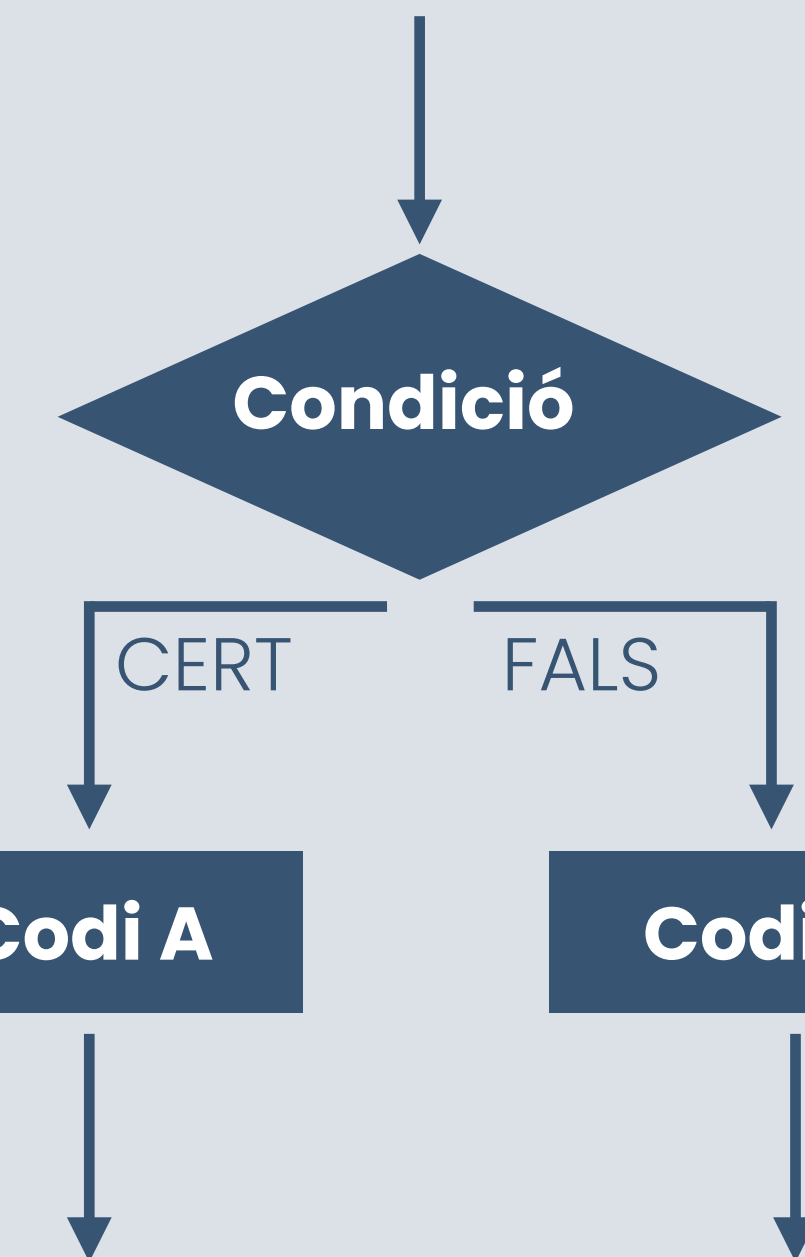
```
fsi
```

```
...
```

Noteu com fem ús de la **indentació**. Millora la llegibilitat del pseudocodi (i del codi en els llenguatges de programació).

Si no es compleix la condició, s'executaran aquestes instruccions alternatives. Noteu que utilitzem 'sino' com a contracció de 'si no' en pseudocodi. En algunes fonts hi trobareu 'altrament'

## Condicional doble



# Conditionals

## Conditional **doble**

- **Exemple:** dissenya un algorisme que indiqui si dos valors introduïts per teclat són divisibles entre ells o no.



# Conditionals

## Conditional **dobble**

- **Exemple:** dissenya un algorisme que indiqui si dos valors introduïts per teclat són divisibles entre ells o no.

## Solució I

```
algorisme divisible és
    var x, y : enter; fvar
inici
    escriure("Introdueix un nombre enter:");
    llegir(x);
    escriure("Introdueix un altre nombre enter:");
    llegir(y);
    si (x mod y = 0) llavors
        escriure("Són divisibles");
    sino
        escriure("No són divisibles");
    fsi
falgorisme
```

# Conditionals

## Conditional **doble**

- **Exemple:** dissenya un algorisme que indiqui si dos valors introduïts per teclat són divisibles entre ells o no.

### Solució I

```
algorisme divisible és
    var x, y : enter; fvar
inici
    escriure("Introdueix un nombre enter:");
    llegir(x);
    escriure("Introdueix un altre nombre enter:");
    llegir(y);
    si (x mod y = 0) llavors
        escriure("Són divisibles");
    sino
        escriure("No són divisibles");
    fsi
falgorisme
```

### Solució II (guardant en booleà)

```
algorisme divisible és
    var x, y: enter;
    divisible: booleà; fvar    $ Podem desar si és divisible en un
                                $ booleà per usar-ho més endavant
inici
    escriure("Introdueix un nombre enter:");
    llegir(x);
    escriure("Introdueix un altre nombre enter:");
    llegir(y);
    si (x mod y = 0) llavors divisible := cert;
    sino divisible := fals;
    fsi
    si (divisible) llavors escriure("Són divisibles");
    sino escriure("No són divisibles");
    fsi
falgorisme
```

# Conditionals

## Conditional **doble**

- **Exemple II:** Donada una lletra minúscula (i sense accent ni dièresi) introduïda per teclat, s'indiqui si és vocal o consonant.

```
algorisme vocal_consonant és  
var  
    ll: caràcter;  
fvar  
inici  
    escriure ("Introdueix una lletra minúscula i sense accent ni  
dièresi");  
    llegir (ll);  
  
falgorisme
```

# Conditionals

## Conditional **doble**

- **Exemple II:** Donada una lletra minúscula (i sense accent ni dièresi) introduïda per teclat, s'indiqui si és vocal o consonant.

```
algorisme vocal_consonant és  
var  
    ll: caràcter;  
fvar  
inici  
    escriure ("Introdueix una lletra minúscula i sense accent ni  
dièresi");  
    llegir (ll);  
    si (ll='a' o ll='e' o ll='i' o ll='o' o ll='u') llavors  
        escriure ("És una vocal");  
    sino  
        escriure ("És una consonant");  
    fsi  
falgorisme
```



# Condicionals

## Condicionals **anidats**

- Podem incloure estructures condicionals dins d'altres estructures condicionals.

```
si (condició) llavors
```

**Codi A**

```
sino
```

```
  si (condició2) llavors
```

**Codi B**

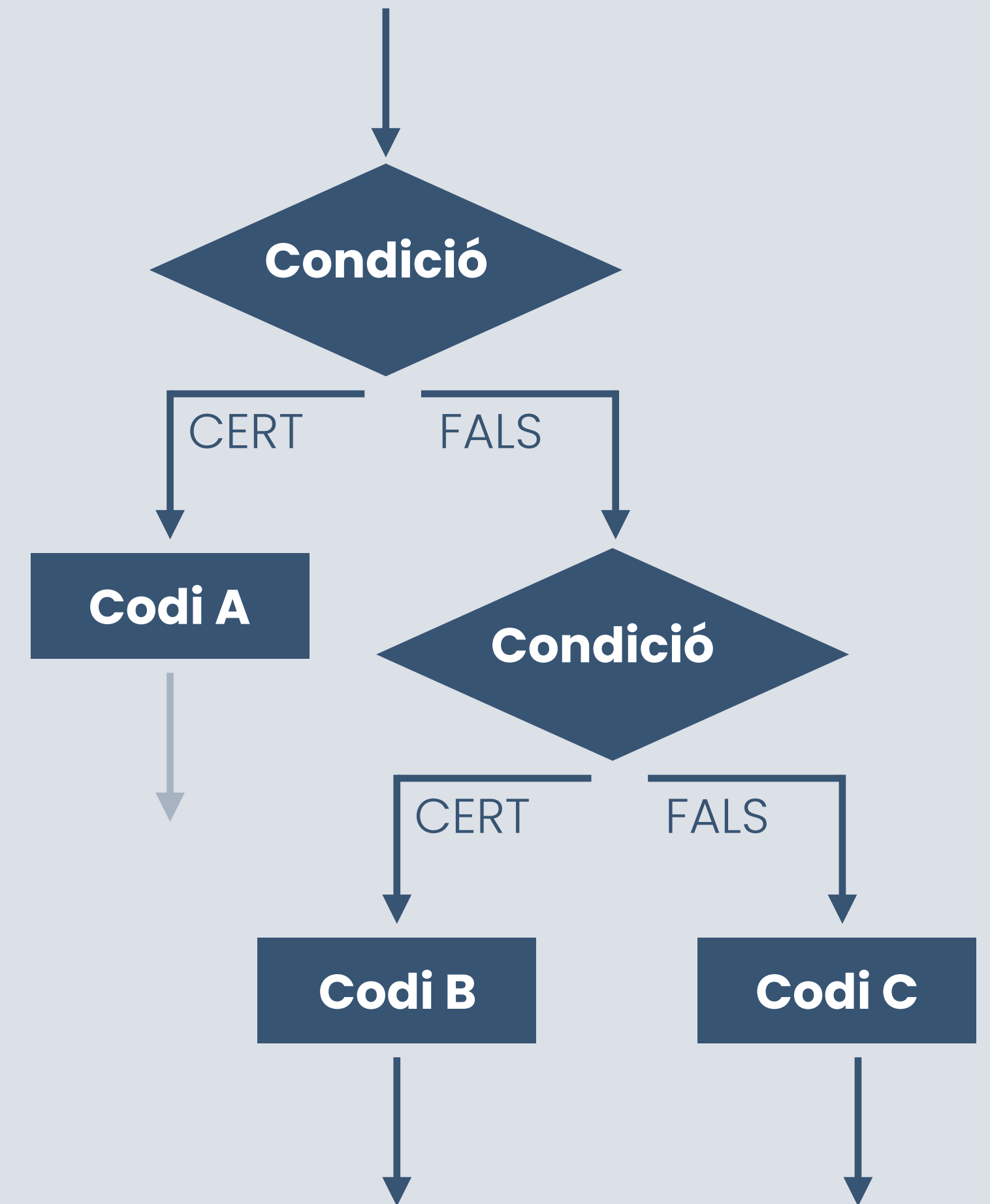
```
sino
```

**Codi C**

```
fsi
```

```
fsi
```

## Condicional **anidat**



# Conditionals

## Conditionals **anidats**

- Podem incloure estructures condicionals dins d'altres estructures condicionals.
- **Exemple:** Donat un numerador i un denominador, escriu un algorisme que indiqui si el resultat de la fracció és major, menor o igual a 1.

```
algorisme resultat_fracció és  
var   num, den: enter; fvar  
inici  
    escriure ("Introdueix el numerador");  
    llegir (num);  
    escriure ("Introdueix el denominador");  
    llegir (den);
```

```
falgorisme
```

# Condicionals

## Condicionals **anidats**

- Podem incloure estructures condicionals dins d'altres estructures condicionals.
- **Exemple:** Donat un numerador i un denominador, escriu un algorisme que indiqui si el resultat de la fracció és major, menor o igual a 1.

```
algorisme resultat_fracció és
var   num, den: enter; fvar
inici
    escriure ("Introdueix el numerador");
    llegir (num);
    escriure ("Introdueix el denominador");
    llegir (den);
    si (num = den) llavors escriure ("Igual a 1");
    sino
        si (num < den) llavors escriure ("Menor que 1");
        sino escriure ("Major que 1");
    fsi
fsi
falgorisme
```

# Condicionals

## Condicionals múltiples (**sino si**)

- Si tenim més d'una condició podem fer servir l'estructura "si", posant més casos amb "sino si"

```
si (condició) llavors
    instruccions1;
sino si (condició2) llavors
    instruccions2;
sino si (condició3) llavors
    instruccions3;
sino
    instruccions4;
fsi
```

```
si (condició) llavors
    Codi A
sino si (condició2) llavors
    Codi B
sino si (condició 3) llavors
    Codi C
sinó
    Codi D
fsi
```



# Condicionals

## Condicionals múltiples (**sino si**)

- Si tenim més d'una condició podem fer servir l'estructura "si", posant més casos amb "sino si"

```
si (condició) llavors
    instruccions1;
sino si (condició2) llavors
    instruccions2;
sino si (condició3) llavors
    instruccions3;
sino
    instruccions4;
fsi
```

- **L'exemple anterior:**

*Donat un numerador i un denominador, escriu un algorisme que indiqui si el resultat de la fracció és major, menor o igual a 1.*

És més normal escriure'l amb una instrucció "sino si".

```
si (num = den) llavors
    escriure("Igual a 1");
sino si (num < den) llavors
    escriure("Major que 1");
sino
    escriure("Menor que 1");
fsi
```

# Condicionals

## Condicionals múltiples (**switch**)

- Si hem d'avaluar un condicional (numèric/caràcter) amb moltes opcions, podem fer servir una estructura de condicional múltiple (o switch)

```
opció (expressió)
    valor1: instrucció1;
        instrucció2;
    ...
    valor2: instrucció3;
        instrucció4;
    ...
    altre cas: instrucció5;
        instrucció6;
    ...
fopció
```

Podem usar la instrucció **opció** quan avaluem com a expressió un nombre o un caràcter.

Els diferents **valors** que pot prendre l'expressió, faran executar les diferents seqüències d'instruccions.

Si l'expressió té un valor no contemplat, s'executaran les instruccions dins d'"altre cas".

Sempre és més fàcil usar el condicional múltiple que moltes estructures si/sino, amb les quals seria fàcil equivocar-se.

# Conditionals

## Condicionals múltiples (switch)

- **Exemple:** Donat un número de l'1 al 7 introduït per teclat, dir a quin dia de la setmana es correspon.

```
algorisme dia_setmana és
var x: enter; fvar
inici
    escriure("Introdueix un nombre (1..7)");
    llegir(x);
```

**falgorisme**

# Conditionals

## Conditionals múltiples (**switch**)

- **Exemple:** Donat un número de l'1 al 7 introduït per teclat, dir a quin dia de la setmana es correspon.

```
algorisme dia_setmana és
var x: enter; fvar
inici
    escriure("Introdueix un nombre (1..7)");
    llegir(x);
    opció (x)
        1: escriure("Dilluns");
        2: escriure("Dimarts");
        ...
        6,7: escriure("Cap de setmana"); $ Tractem dues opcions
    altre cas: escriure("Valor no vàlid");
    fopció
falgorisme
```



# Conditionals

## Conditionals múltiples (**switch**)

Què fa aquest algorisme?

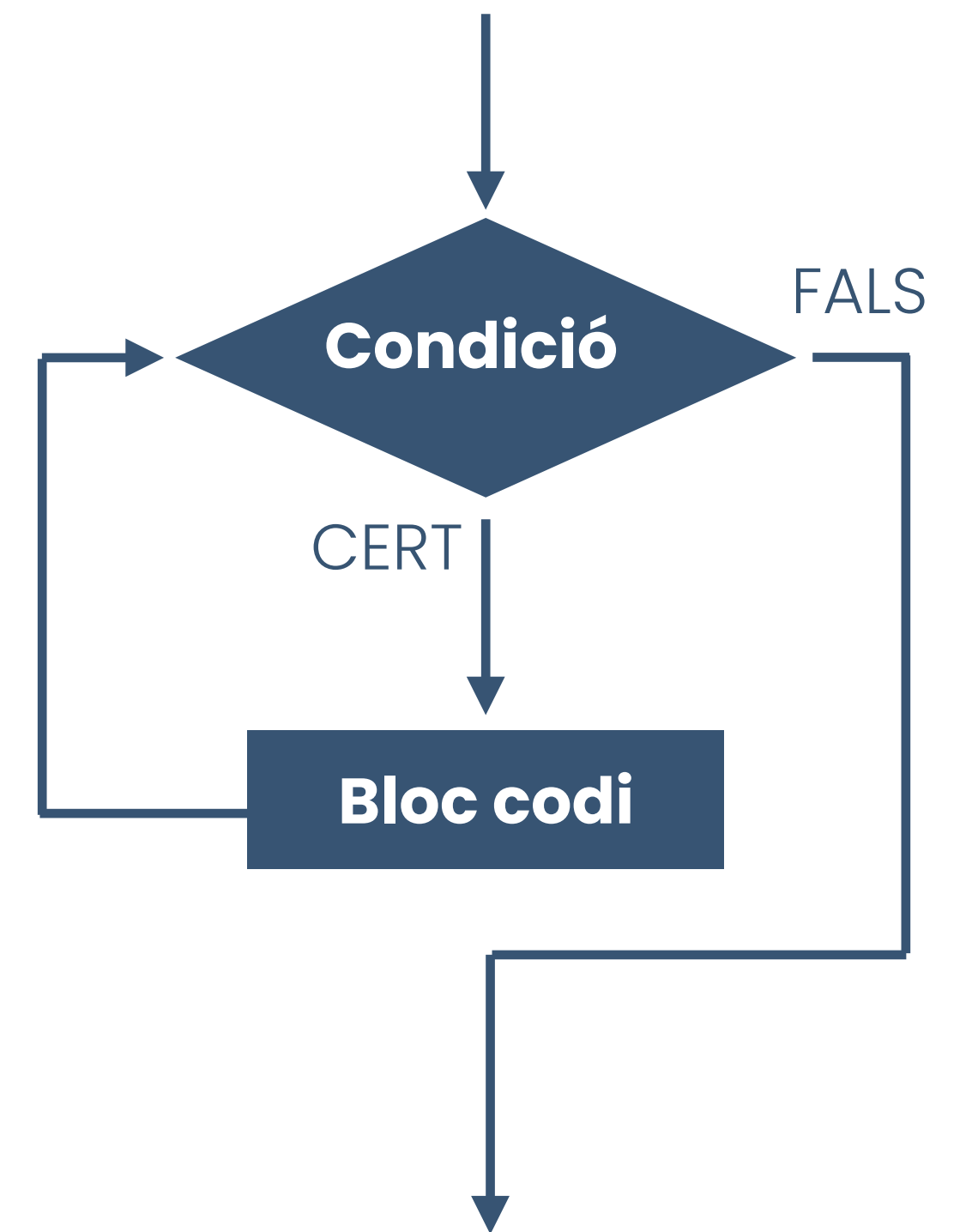
```
algorisme tipus_paraula és
var n_sil: enter; fvar $ Número de síl·labes
inici
    escriure ("Quantes síl·labes té la paraula?");
    llegir (n_sil);
    opció (n_sil)
        1: escriure ("És una monosíl·laba");
        2: escriure ("És una bisíl·laba");
        3: escriure ("És una trisíl·laba");
        4, 5, 6, 7, 8: escriure ("És una polisíl·laba");
        altre cas: ("Segur que en té tantes?");
    fopció
falgorisme
```

**ITERACIONES**

# Bucles

## Estructura

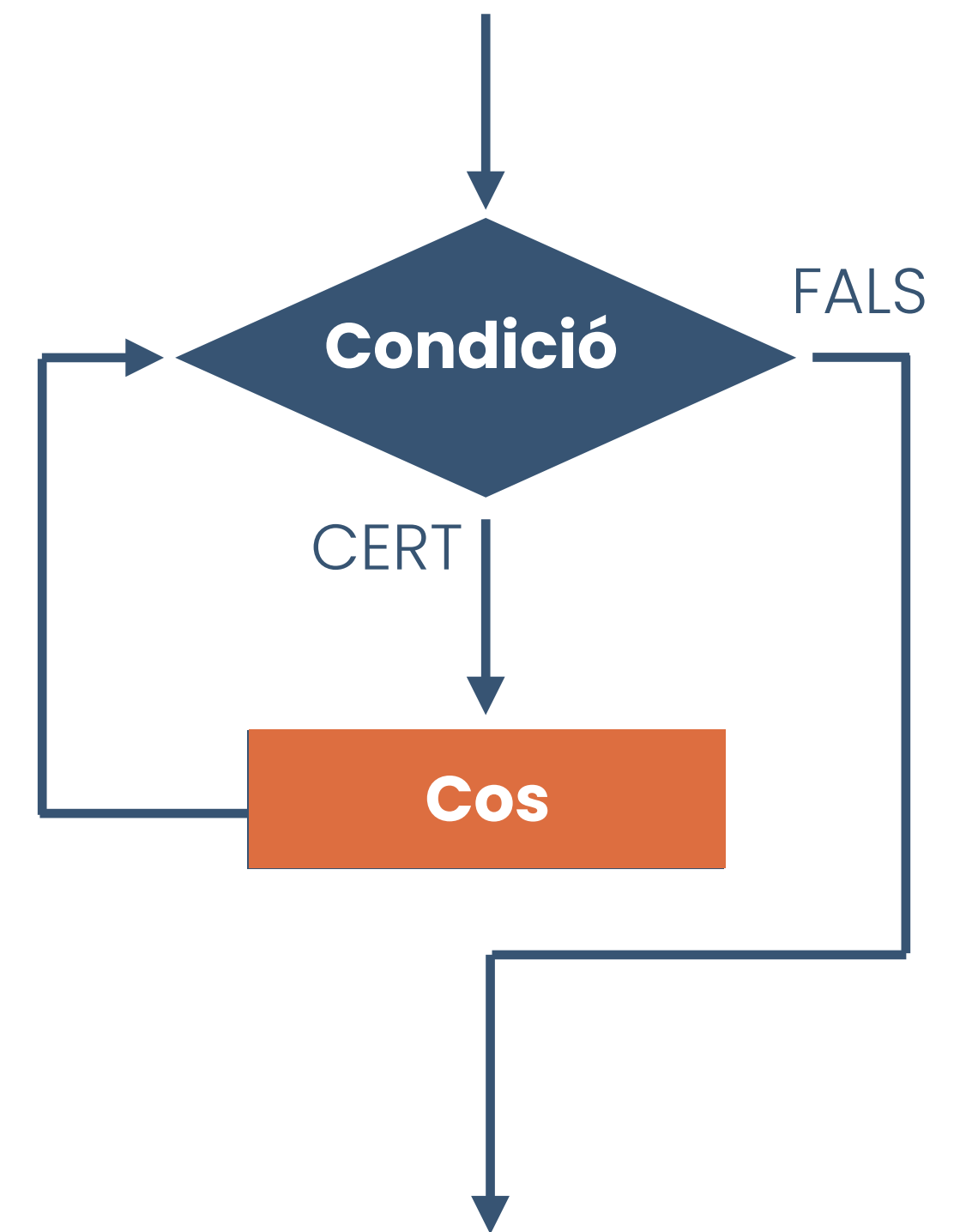
- El bucle és una estructura de control que repeteix una instrucció o conjunt d'instruccions un cert nombre de vegades



# Bucles

## Estructura

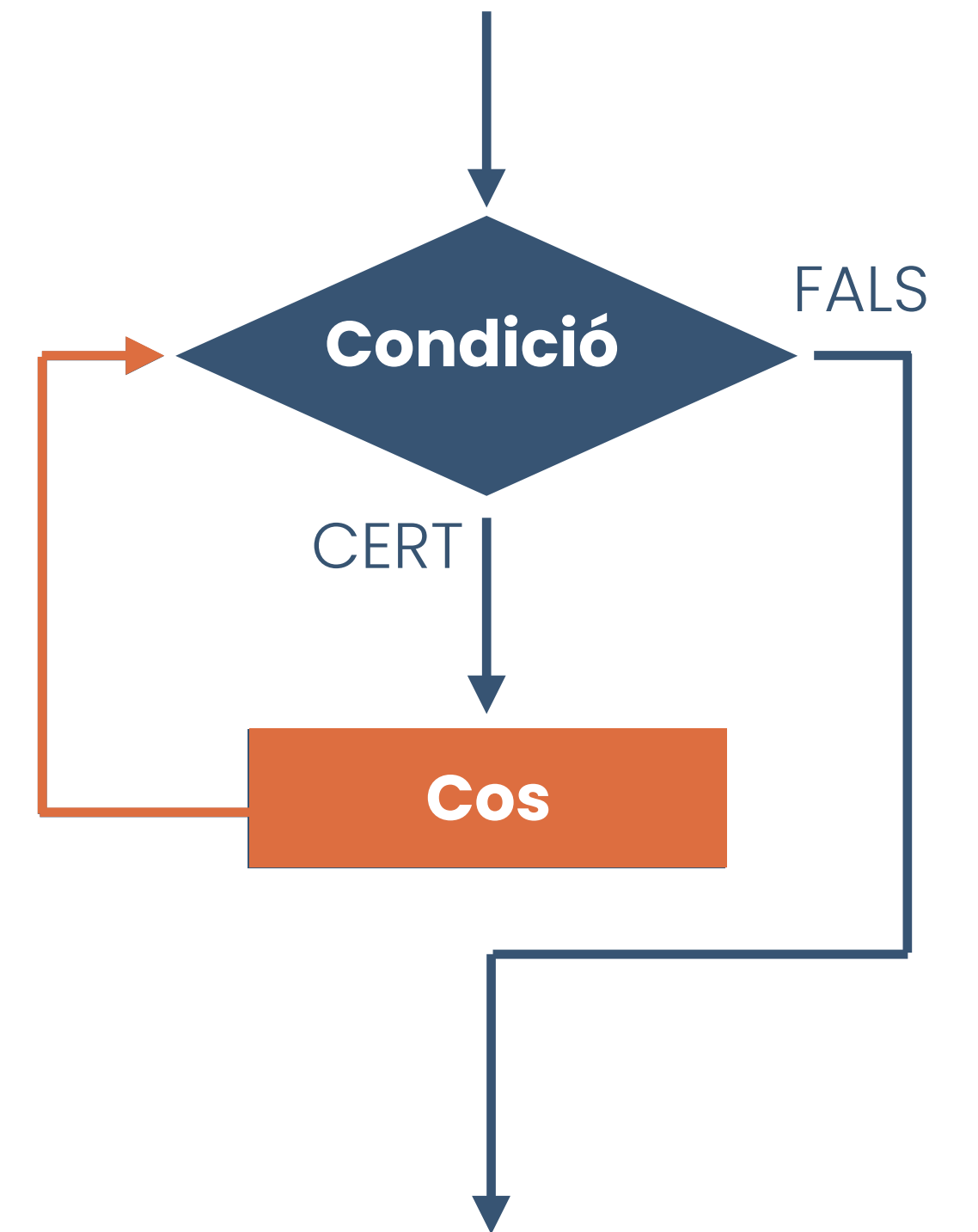
- El bucle és una estructura de control que repeteix una instrucció o conjunt d'instruccions un cert nombre de vegades
  - La instrucció o el bloc d'instruccions s'anomena **cos**.



# Bucles

## Estructura

- El bucle és una estructura de control que repeteix una instrucció o conjunt d'instruccions un cert nombre de vegades
  - La instrucció o el bloc d'instruccions s'anomena **cos**.
  - Cada repetició s'anomena iteració

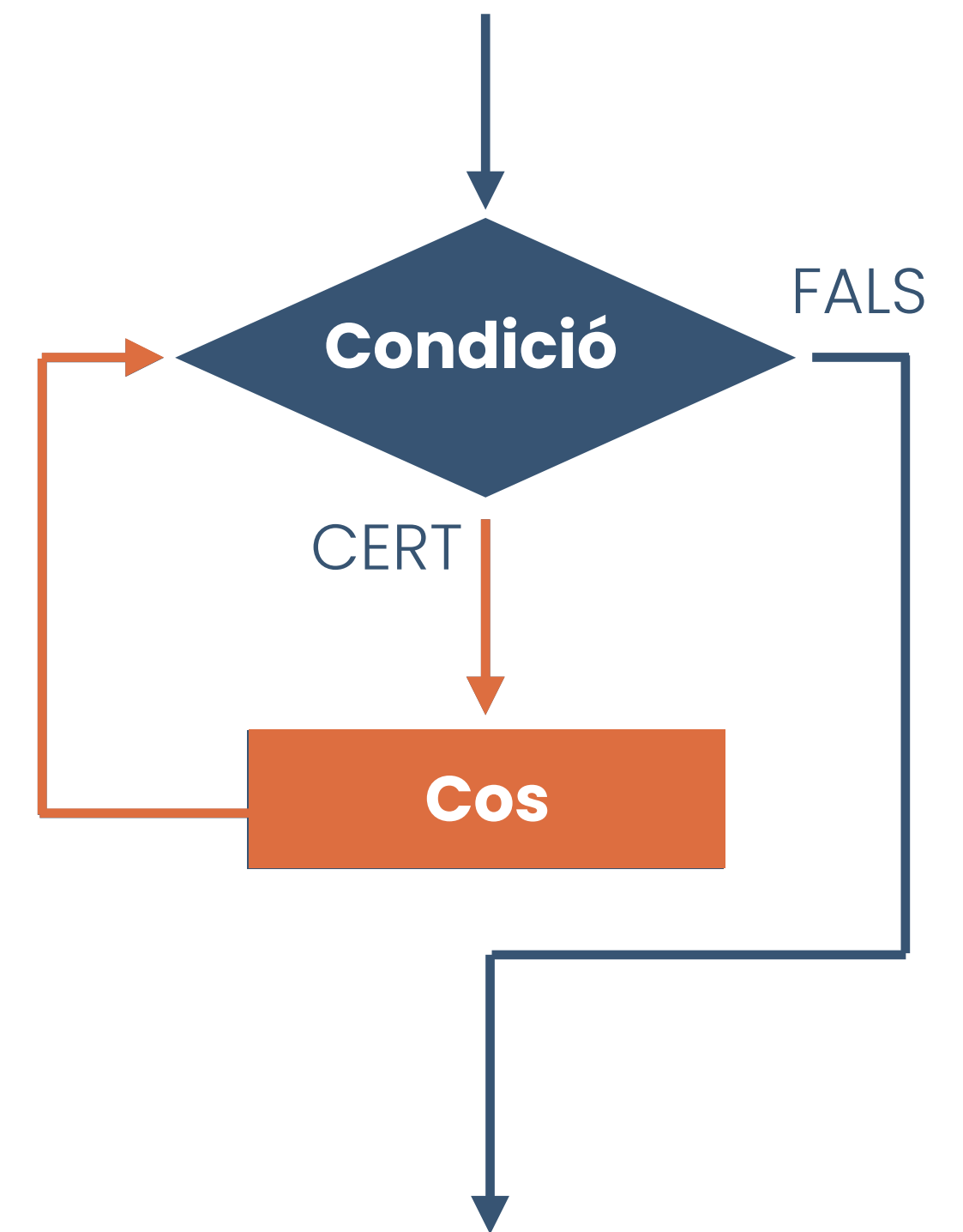




# Bucles

## Estructura

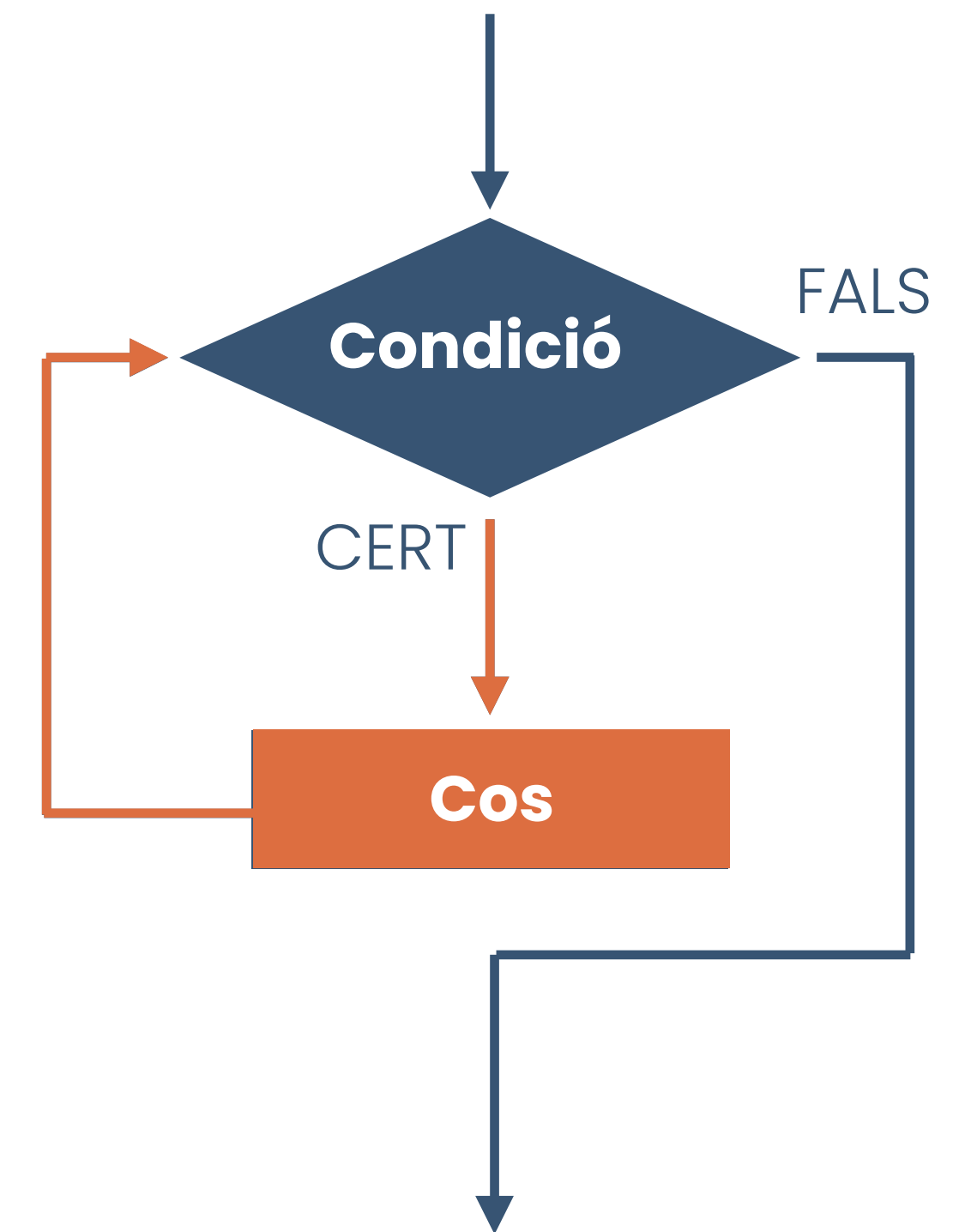
- El bucle és una estructura de control que repeteix una instrucció o conjunt d'instruccions un cert nombre de vegades
  - La instrucció o el bloc d'instruccions s'anomena **cos**.
  - Cada repetició s'anomena iteració
  - Els bucles repeteixen les instruccions si es compleix una condició, és a dir, **mentre la condició és certa**.



# Bucles

## Estructura

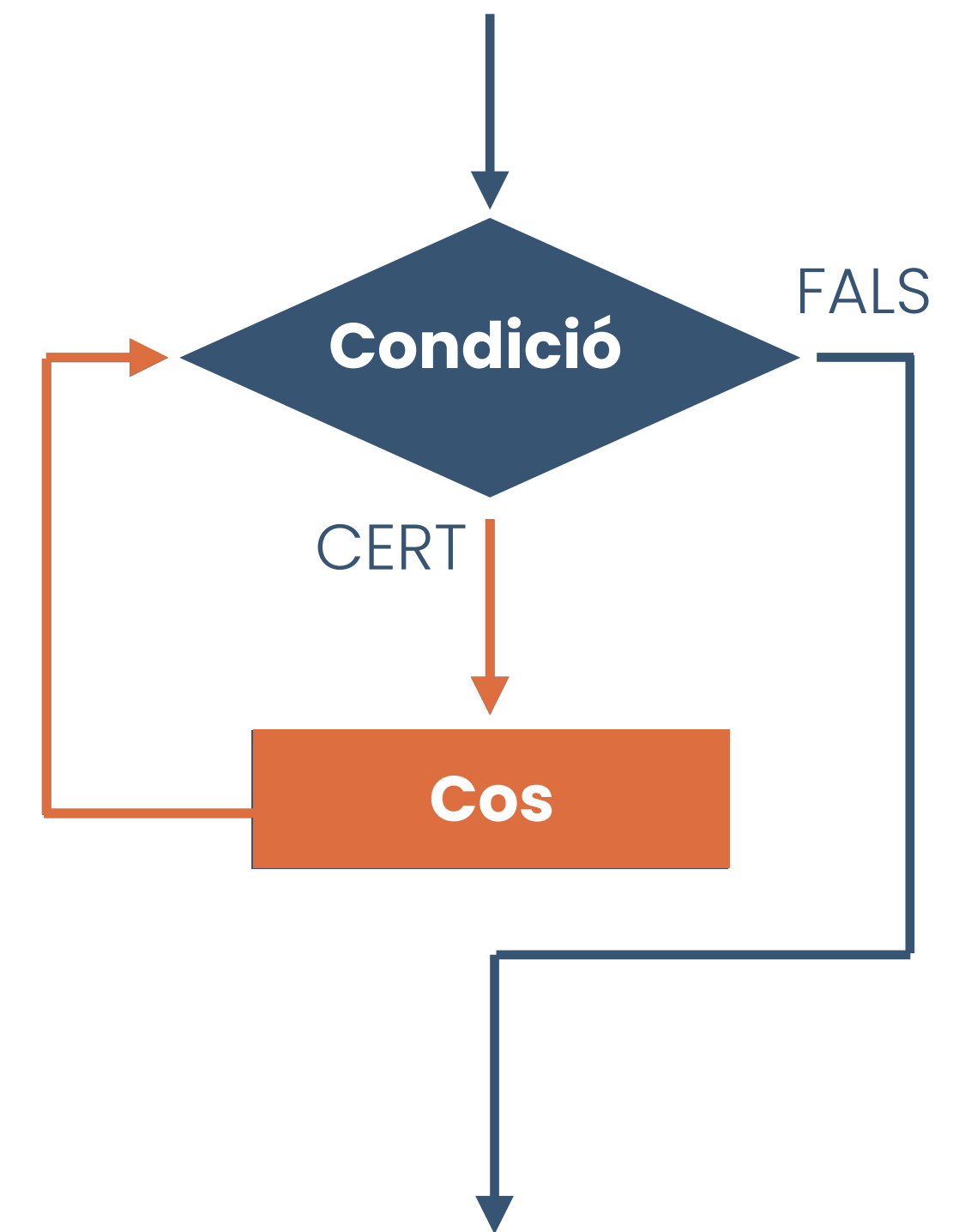
- El bucle és una estructura de control que repeteix una instrucció o conjunt d'instruccions un cert nombre de vegades
  - La instrucció o el bloc d'instruccions s'anomena **cos**.
  - Cada repetició s'anomena iteració
  - Els bucles repeteixen les instruccions si es compleix una condició, és a dir, **mentre la condició és certa**.
- La **condició** pot estar formada per una variable booleana, o una comparació de valors i variables, etc.
  - Sovint implica una variable anomenada variable de control (índex, comptador...)



# Bucles

## Estructura

- El bucle és una estructura de control que repeteix una instrucció o conjunt d'instruccions un cert nombre de vegades
  - La instrucció o el bloc d'instruccions s'anomena **cos**.
  - Cada repetició s'anomena iteració
  - Els bucles repeteixen les instruccions si es compleix una condició, és a dir, **mentre la condició és certa**.
- La **condició** pot estar formada per una variable booleana, o una comparació de valors i variables, etc.
  - Sovint implica una variable anomenada variable de control (índex, comptador...)



- Tipus de bucles:
  - Bucle "mentre" (while)
  - Bucle "per" (for)

# Bucles

## Bucle "mentre"

- El cos del bucle es repeteix mentre la condició del bucle sigui certa: mentre es compleixi la condició.
- Ja no es repeteix quan la condició del bucle és falsa, llavors sortim del bucle

# Bucles

## Bucle "mentre"

- El cos del bucle es repeteix mentre la condició del bucle sigui certa: mentre es compleixi la condició.
- Ja no es repeteix quan la condició del bucle és falsa, llavors sortim del bucle

### Condicional vs. bucle

**si** (**no** mal\_de\_panxa) :



**f****si**

**mentre** (**no** mal\_de\_panxa) :



**f****mentre**

*Quantes bosses de patates m'he menjat en cada cas?*



# Bucles

## Bucle "mentre"

- El cos del bucle es repeteix mentre la condició del bucle sigui certa: mentre es compleixi la condició.
- Ja no es repeteix quan la condició del bucle és falsa, llavors sortim del bucle
- Bucle "mentre" en pseudocodi:

```
Inicialització variable control
mentre (condició) fer
    Cos del bucle
    Actualització variable control
fmentre
```

# Bucles

## Bucle "mentre"

- **Exemple:** Mostra per pantalla tants asteriscs com l'usuari indiqui per teclat

### Estructura

```
algorisme mostrar_asteriscs és
inici
    Obtenir nombre d'asteriscs
    mentre (condició) fer
        Mostrar un asterisc per pantalla
    fmentre
falgorisme
```

### Solució

```
algorisme mostrar_asteriscs és
    var i, num: enter; fvar
inici
    escriure("Indica quants asteriscs vols:");
    llegir(num);
    i:= 0;   $ Inicialització variable de control
    mentre (i ≤ num) fer
        escriure("*");
        i :=i+1;           $ Actualització variable de control
    fmentre
    $ acabem quan i=num
falgorisme
```

# Bucles

## Bucle "mentre"

- **Exemple II:** Calcular la suma dels enters parells positius inferiors a un valor enter indicat per teclat

Amb un exemple ho podem veure més clar:

Valor = 15

Nombres 2, 4, 6, 8, 10, 12, 14

La suma és 56

# Bucles

## Bucle "mentre"

- **Exemple II:** Calcular la suma dels enters parells positius inferiors a un valor enter indicat per teclat

Amb un exemple ho podem veure més clar:

Valor = 15

Nombres 2, 4, 6, 8, 10, 12, 14

La suma és 56

- Quines **dades** necessitem?

```
algorisme sumar_parells és  
  var n, i, suma: enter; fvar
```

‘suma’ és una variable que fa les funcions d’acumulador. Sobre aquesta variable s’hi aniran sumant els nombres que siguin parells.

# Bucles

## Bucle "mentre"

- **Exemple II:** Calcular la suma dels enters parells positius inferiors a un valor enter indicat per teclat

### Solució

```
algorisme sumar_parells és
    var n, i, suma: enter; fvar
inici
    escriure ("Introdueix el valor límit");
    llegir(n);
    suma := 0;          $ Inicialitzem a zero
    i := 2;             $ Inicialitzem la variable de control
    mentre (i < n) fer
        suma := suma + i;
        i := i + 2;

    fmentre
    escriure ("El resultat és ", suma);
falgorisme
```



# Bucles

## Bucle "mentre"

- **Exemple III:** Càlcul d'una potència ( $b^e$ )

Hem de calcular-la a partir de multiplicar la base tantes vegades com ens indiqui l'exponent. Això de "tantes vegades com" ens indica que ho hem de fer amb un bucle.

Com ho calcularíem? Vegem un parell de casos

$2^3$      $b = 2, e = 3$

resultat := 2

resultat :=  $2 \times 2 = 4$

resultat :=  $4 \times 2 = 8$

$2^0$      $b = 2, e = 0$

resultat := 1

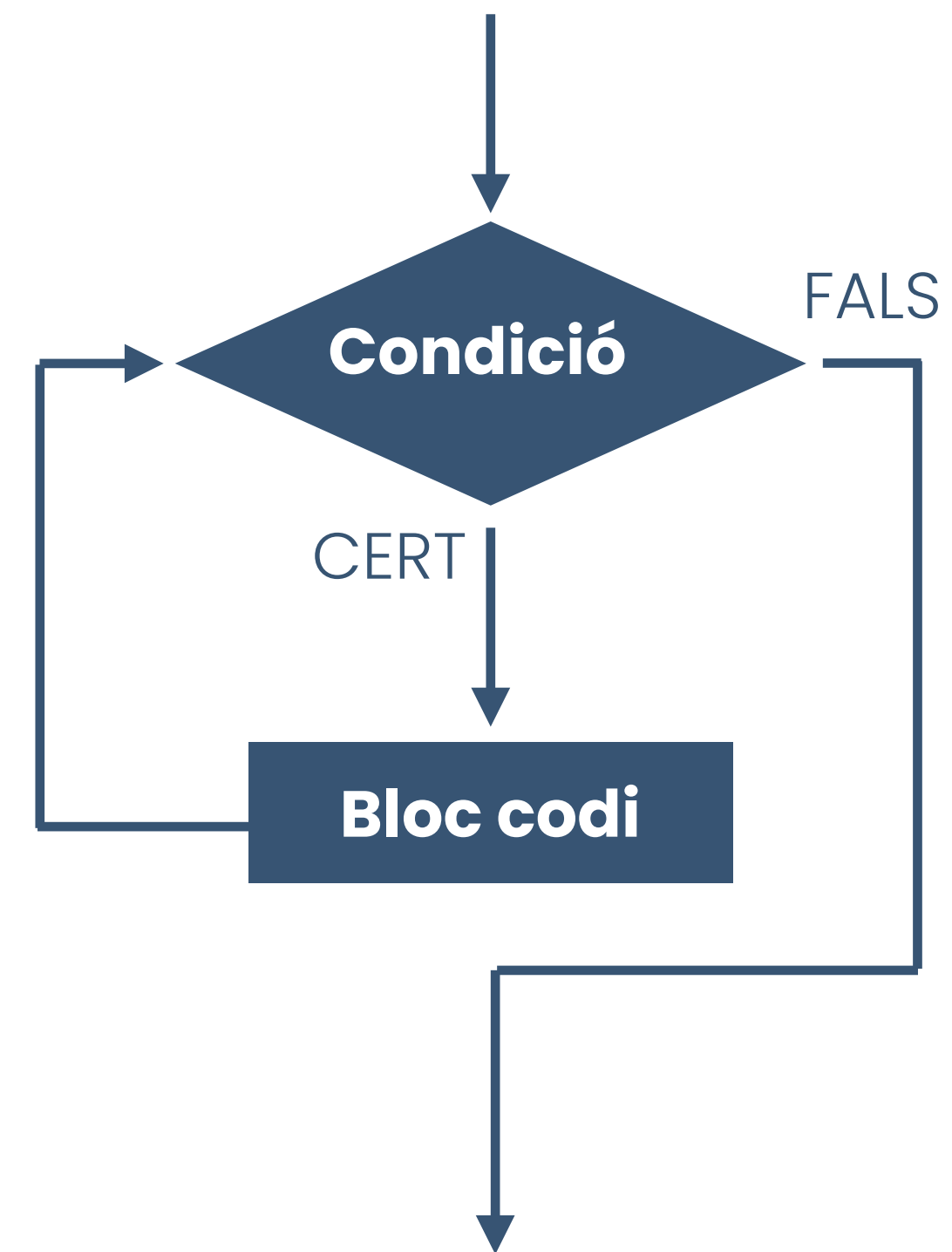
(qualsevol nombre elevat a 0 és 1)

**Deures per casa**

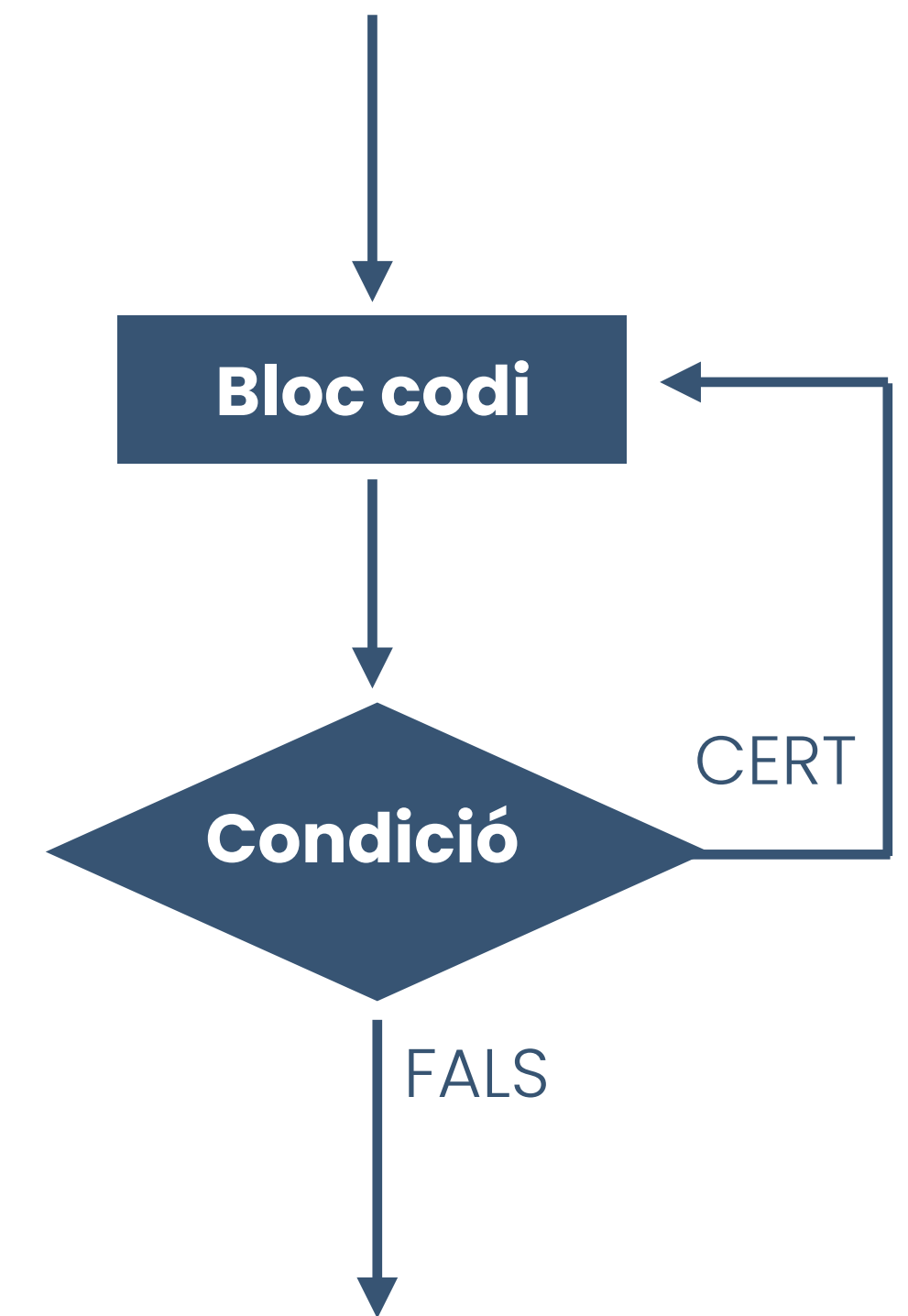
# Bucles

## Variant de "mentre": "fer mentre"

- Bucle "do while": Es tracta d'una variant del mentre que, **com a mínim, executa el cos del bucle una vegada.**



*"mentre" / "while"*



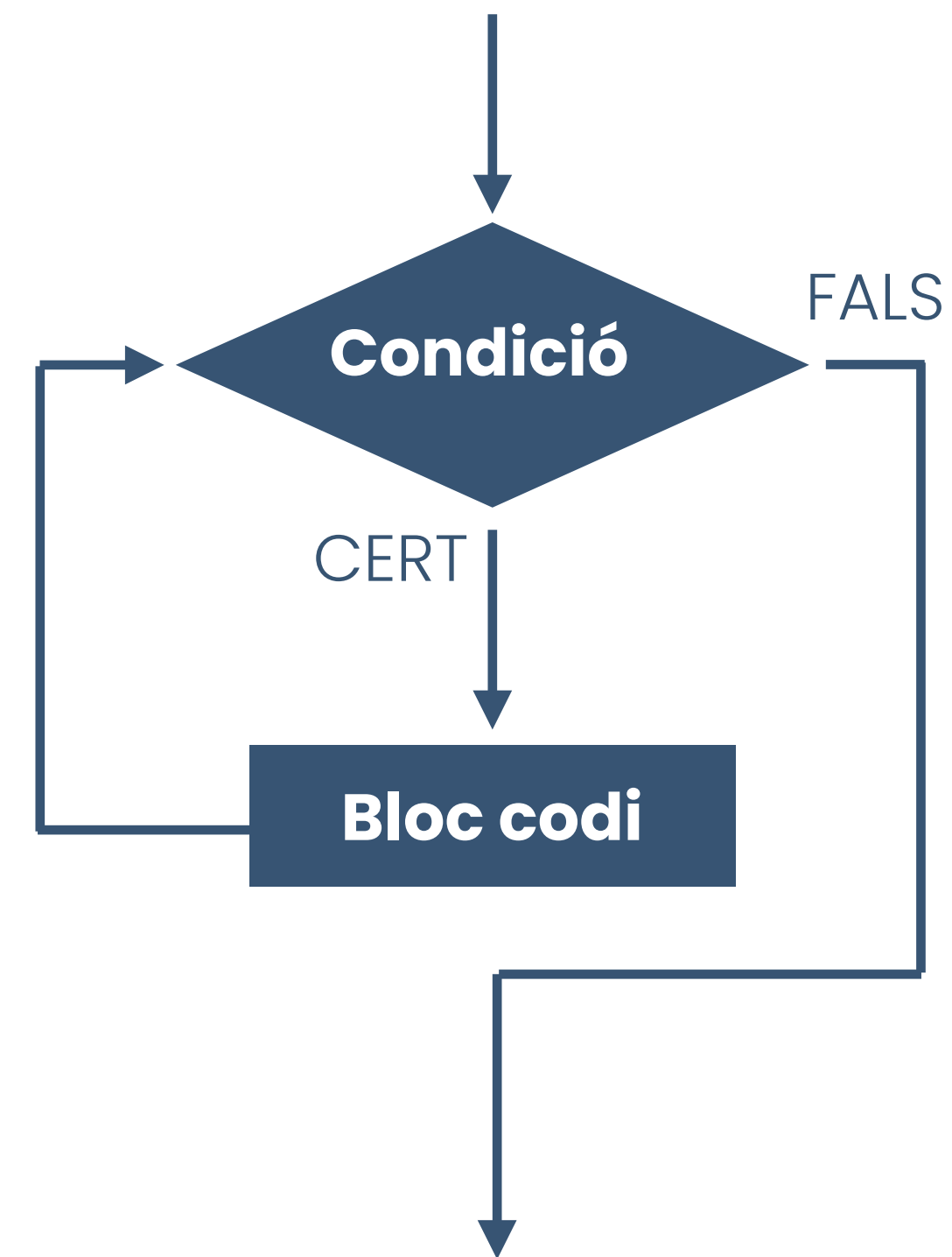
*"fer mentre" / "do while"*

# Bucles

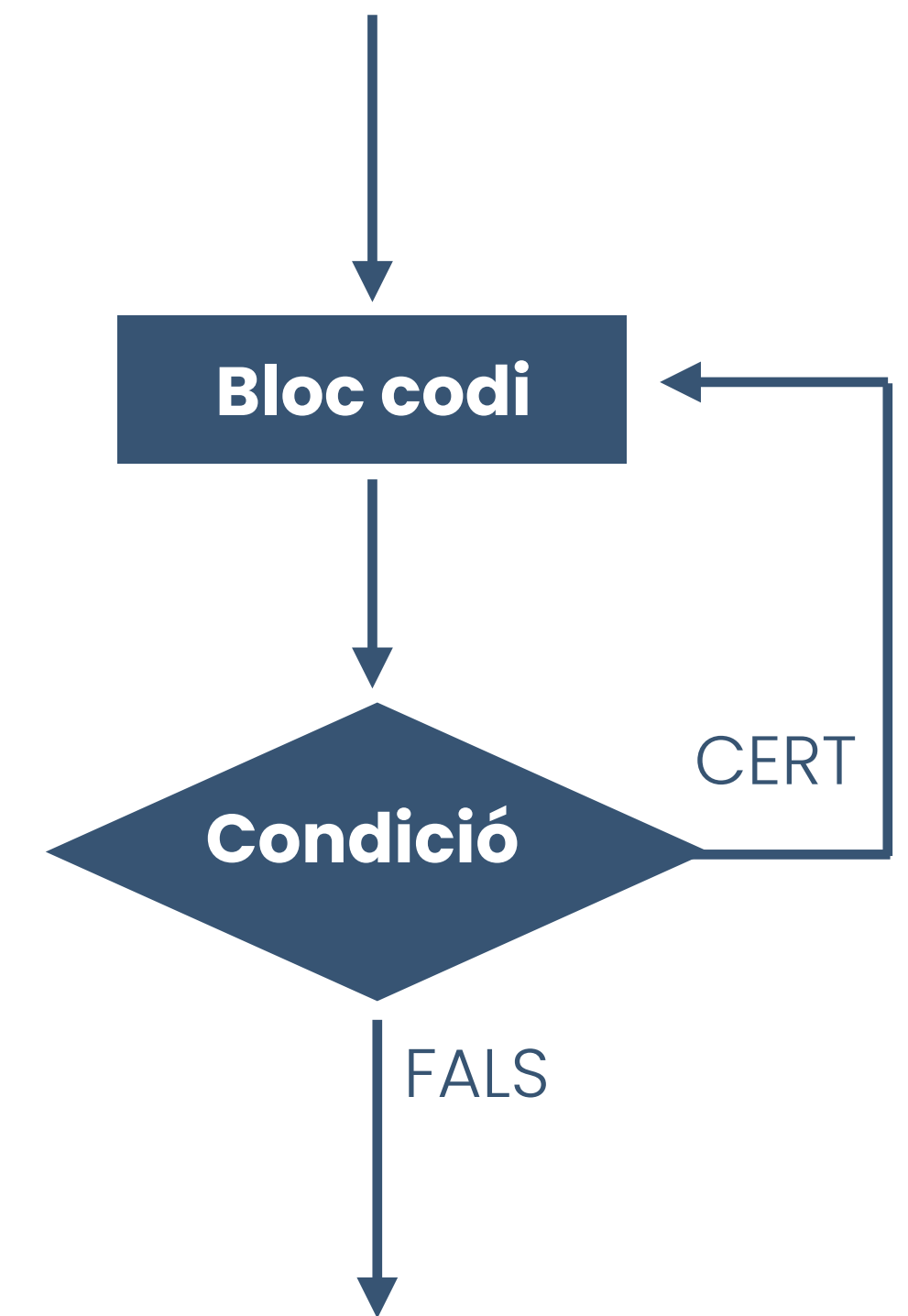
## Variant de "mentre": "fer mentre"

- Bucle "do while": Es tracta d'una variant del mentre que, **com a mínim, executa el cos del bucle una vegada.**
- En pseudocodi:

```
fer  
    Cos del bucle  
mentre (condició)
```



*"mentre" / "while"*



*"fer mentre" / "do while"*

# Bucles

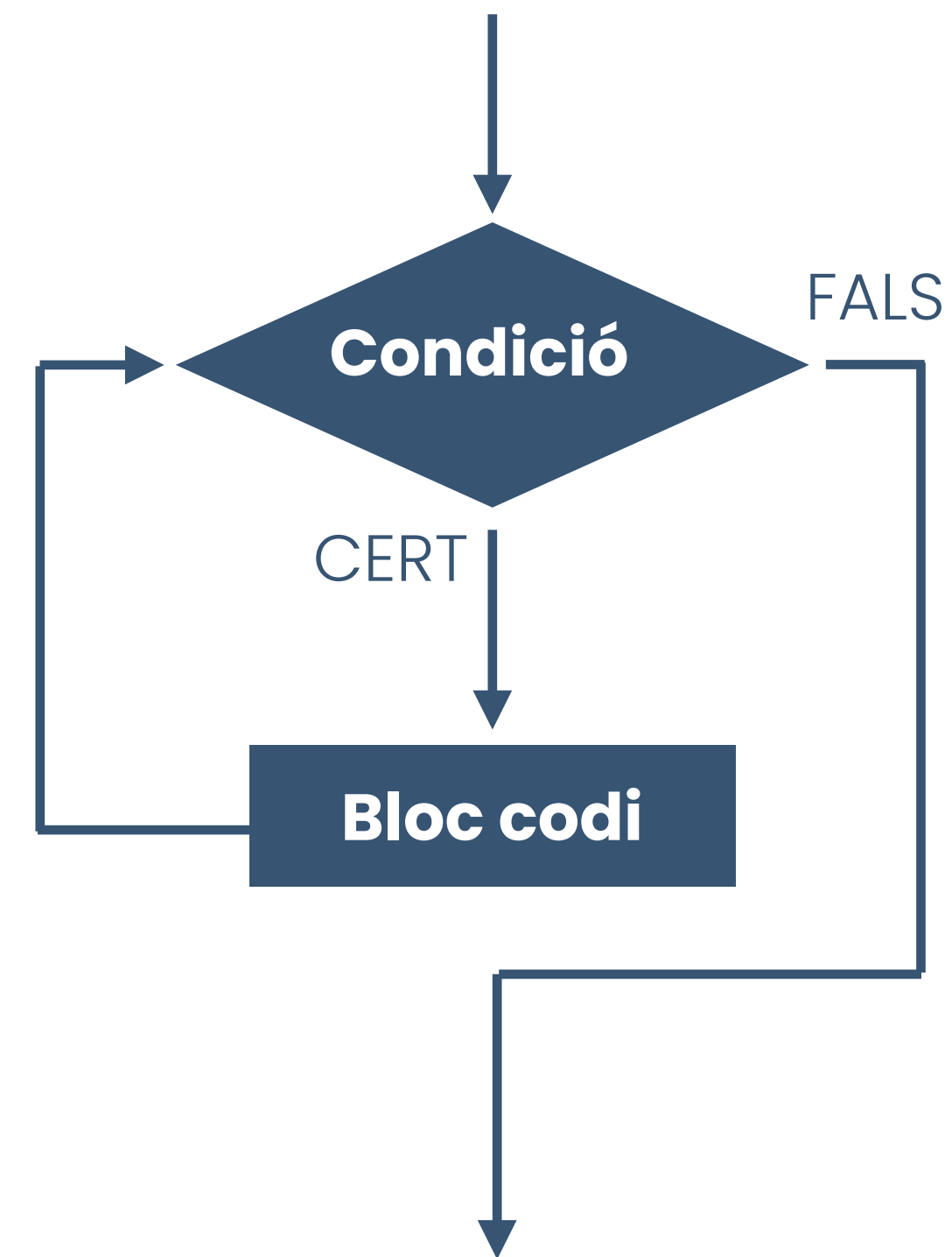
## Variant de "mentre": "fer mentre"

- Bucle "do while": Es tracta d'una variant del mentre que, **com a mínim, executa el cos del bucle una vegada.**
- En pseudocodi:

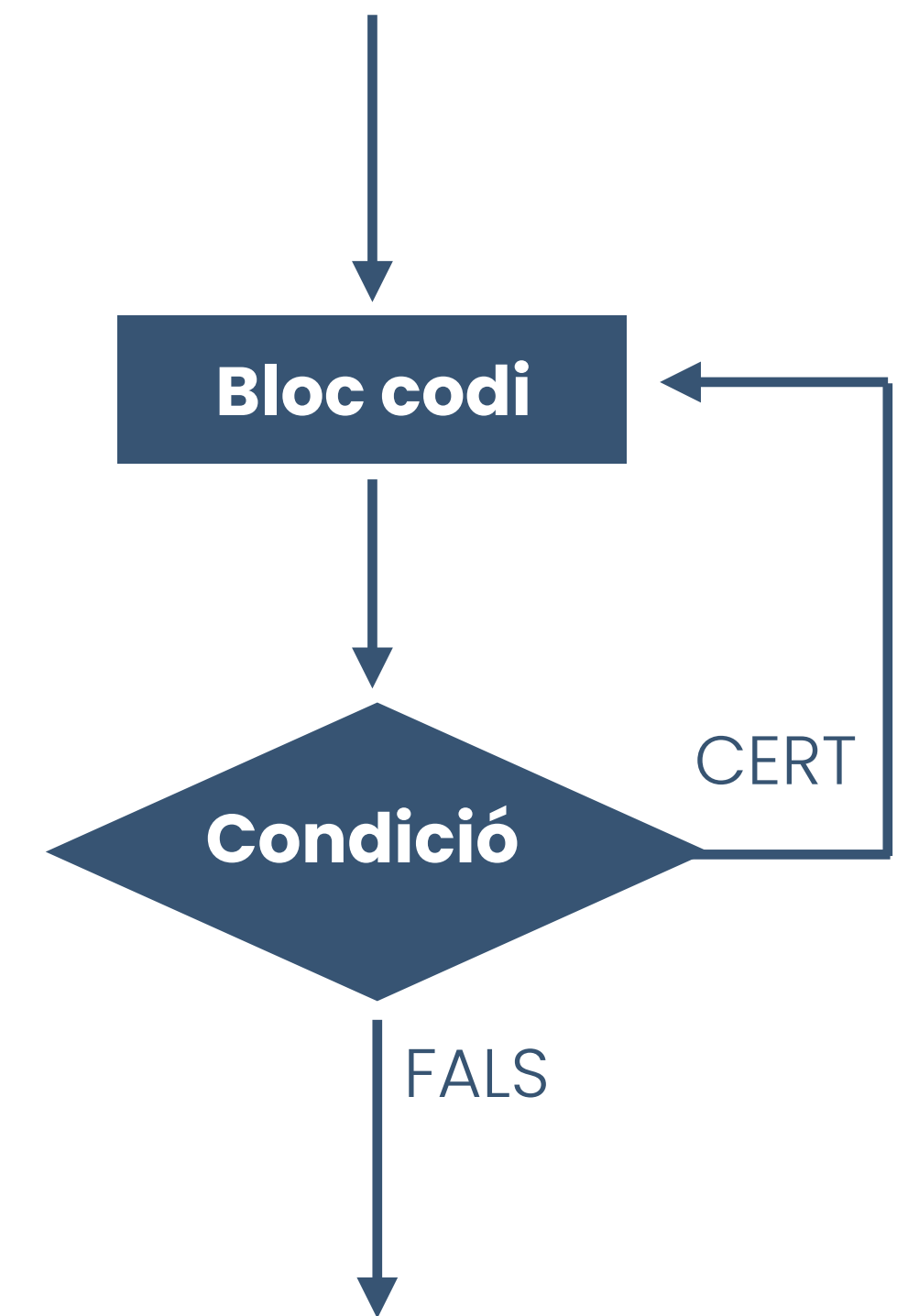
```
fer  
    Cos del bucle  
mentre (condició)
```

- Farem servir aquesta estructura en casos excepcionals
  - Cas paradigmàtic: el menú d'opcions.

```
Simple Menu Driven Program  
-----  
1. Sum  
2. Subtract  
3. Multiply  
4. Divide  
5. Exit  
-----  
Select option (1-5) : 3  
Enter Two Number : 3 5  
Multiplication  
3 * 5 = 15
```



"mentre" / "while"



"fer mentre" / "do while"

# Bucles

## Bucle "per" (for)

- Es tracta d'una altra manera de descriure bucles.
- El seu ús es restringeix a **quan coneixem el nombre d'iteracions** que cal fer i la variable de control s'incrementa fins arribar a un límit

```
per (i:= 0; i<n_iteracions; i:= i+1) fer  
    Cos del bucle  
fper
```

Usant un bucle per no oblidarem actualitzar la variable de control ni d'inicialitzar-la!



# Bucles

## Bucle "per" (for)

- **Exemple:** Mostra per pantalla tants asteriscs com l'usuari indiqui per teclat

```
algorisme mostrar_asteriscs és  
    var i, num: enter; fvar  
inici  
    escriure("Indica quants asteriscs vols:");  
    llegir(num);  
    per (i:=0; i<num; i:=i+1) fer  
        escriure("*");  
    fper  
falgorisme
```

# Conditionals: RESUM

## Condicional simple

```
si (condició) llavors  
    instruccions1;  
fsi
```

## Condicional doble

```
si (condició) llavors  
    instruccions1;  
sino  
    instruccions2;  
fsi
```

## Condicional múltiple (I)

```
si (condició) llavors  
    instruccions1;  
sino si (condició2) llavors  
    instruccions2;  
sino  
    instruccions_n;  
fsi
```

## Condicional múltiple (II)

```
opció (expressió)  
    valor1: instruccions1;  
    valor2: instruccions2;  
    ...  
    valorN: instruccionsn;  
en altre cas:  
    instruccions2;  
fopcio
```

# Iteracions: RESUM

Si **desconeixem** el nombre d'iteracions finals

De 0 a N iteracions

```
mentre (condició_NO_fi) fer  
    instruccions;  
fmentre
```

De 1 a N iteracions

```
fer  
    instruccions;  
mentre (condició_NO_fi)
```

Si **coneixem** el nombre d'iteracions finals

```
per (index:=valor_inicial; condició_no_fi; increment_index)  
    instruccions;  
fper
```