

LABORATORIS

PROGRAMACIÓ CIENTÍFICA

Sessió 4: Procediments

PROCEDIMENTS

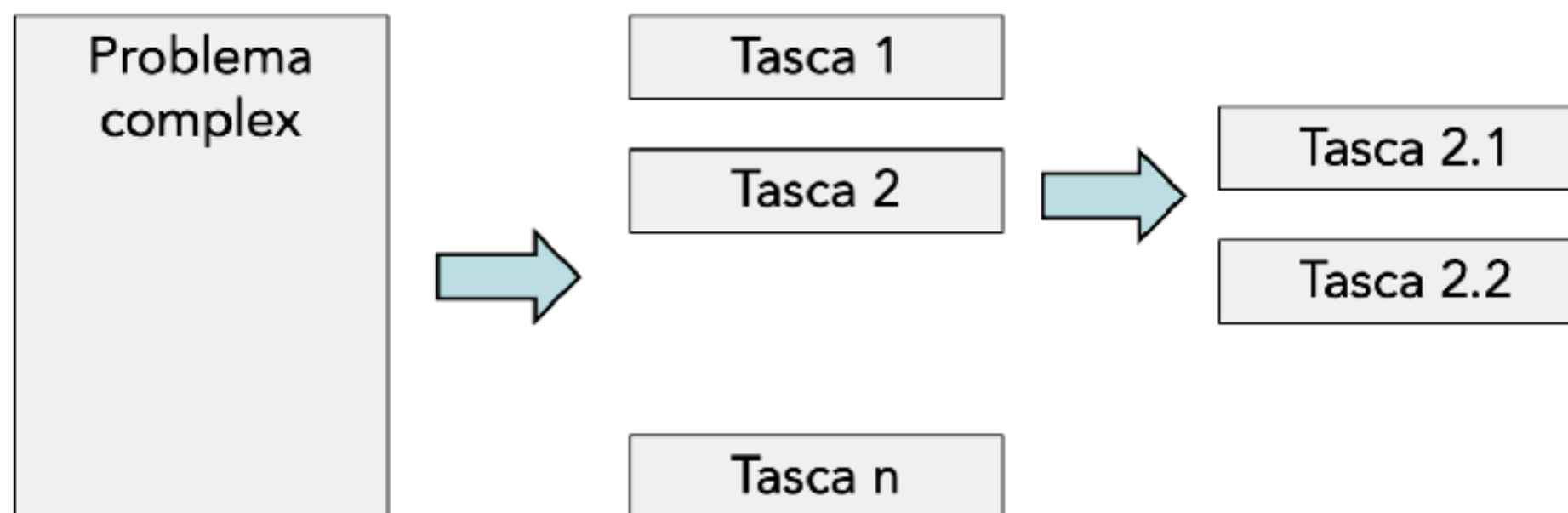
FUNCIONS & ACCIONS

Recordem...

Procediments

Per a què?

- Fins ara hem resolt problemes simples.
- Si volem resoldre problemes més complexos, generarem molt de codi, i llavors és necessari l'ús de procediments.
- A l'hora de resoldre problemes, farem servir el **disseny descendent**



- Al resoldre problemes complexos, les instruccions que resolen parts del problema general les podem **encapsular** en **procediments**.

Avantatges d'usar procediments

Estructurarem millor els programes, per facilitar-ne el seguiment i la seva comprensió (**llegibilitat**).

Reaprofitarem el codi que ja està implementat i testejat (**re-usabilitat**).

Localitzarem i esmenarem errors amb més facilitat (**depurabilitat**).

El procés d'escriure procediments

- 1 Identificar** quin és el **subproblema** que voldrem encapsular en un procediment.
Generalment serà allò que haguem de fer servir més d'una vegada o bé processos complicats que no tenen massa a veure amb el codi i volem separar.

E.g. "Calcular una suma, trobar el valor mínim, calcular la desviació estàndard..."

- 2** Decidir si és **acció** o **funció**. Necessitem que retorni un resultat?

Accions

Procediments que **no** retornen un resultat

Són procediments que escriuen a pantalla, a fitxer, o bé que modifiquen els propis paràmetres d'entrada que reben

Imprimir un missatge en funció d'una entrada

Funcions

Procediments que retornen un resultat

Hem d'assignar el resultat de la funció a una variable del mateix tipus que el seu retorn

E.g. "Calcular la desviació estàndard"

El procés d'escriure procediments

- 1 Identificar** quin és el **subproblema** que voldrem encapsular en un procediment. Generalment serà allò que haguem de fer servir més d'una vegada o bé processos complicats que no tenen massa a veure amb el codi i volem separar.

E.g. "Calcular una suma, trobar el valor mínim, calcular la desviació estàndard..."

- 2** Decidir si és **acció** o **funció**. Necessitem que retorni un resultat?

- 3 Escriure la capçalera.** Aquí necessitarem **pensar** sobre els paràmetres d'entrada i de sortida. Quins seran, quins valors poden tenir, de quin tipus són?

E.g. En una funció que calculi una suma, quants paràmetres d'entrada tinc? (Nombre d'operands). De quin tipus són (enters, reals?). Quin tipus retornarà? --> Tot això són les meves decisions de disseny i dotaran d'una funcionalitat o altra a la meva funció.

- 4 Escriure el cos.** Aquí escriurem l'algorisme del procediment en sí. Haurem de contemplar les possibles diferències de programació a l'hora d'escriure dins d'una funció vs. escriure al programa principal. (Pas de paràmetres)

El procés d'escriure procediments

2 Decidir si és **acció** o **funció**. Necessitem que retorni un resultat?

3 **Escriure la capçalera.** Aquí necessitarem **pensar** sobre els paràmetres d'entrada i de sortida. Quins seran, quins valors poden tenir, de quin tipus són?

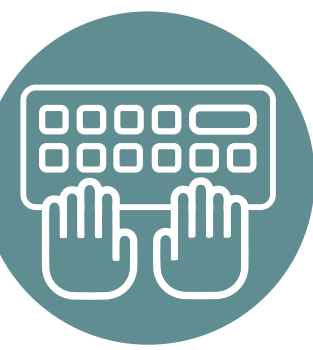
E.g. En una funció que calculi una suma, quants paràmetres d'entrada tinc? (Nombre d'operands). De quin tipus són (enters, reals?). Quin tipus retornarà? --> Tot això són les meves decisions de disseny i dotaran d'una funcionalitat o altra a la meva funció.

4 **Escriure el cos.** Aquí escriurem l'algorisme del procediment en sí. Haurem de contemplar les possibles diferències de programació a l'hora d'escriure dins d'una funció vs. escriure al programa principal. (Pas de paràmetres)

5 **EL PAS MÉS IMPORTANT: TESTEJAR LA FUNCIO QUE HEM ESCRIT!**. Abans d'integrar la funció per complet al nostre programa principal, hem de veure si realment funciona. Necessitem cridar-la amb uns quants valors diferents (jocs de proves) per veure si realment fa el que ha de fer.

6 **Un cop testejada, incorporar-la al programa principal.** Realitzar la crida a la funció des del programa principal.

Definició d'accions



A teoria:

```
acció nom_acció (paràmetres) és  
$ Cos de l'acció  
facció
```

En C:

Fem servir "void"
com a tipus de
retorn per indicar
que no retornem res

Nom del
procediment

Els procediments es
defineixen fora del
"main" o bé a un
altre fitxer

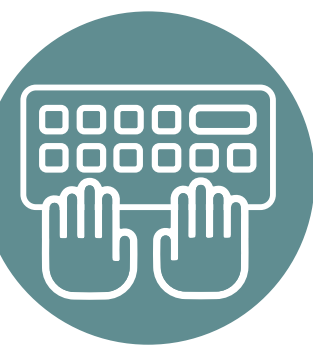
Crida

Paràmetres: en
aquest cas l'acció no
rep cap paràmetre

Exemple

```
1 #include <stdio.h>  
2  
3 void accio_saludar() {  
4     printf("Hola, que tal?\n");  
5 }  
6  
7 int main(){  
8  
9     accio_saludar();  
10    return 0;  
11 }
```

Definició d'accions



A teoria:

```
acció nom_acció (paràmetres) és
    $ Cos de l'acció
facció
```

En C:

Com especificar
paràmetres?
tipus nom_formal
Separats per comes

```
1 void nom_accio (int a, float b)
2 {
3     // Cos de l'acció
4 }
```

Exemple

```
1 #include <stdio.h>
2
3 void accio_saludar_molt(int n) {
4     for(int i=0; i<n; i++){
5         printf("Hola, que tal?\n");
6     }
7 }
8
9 int main(){
10     int v = 10;
11     accio_saludar_molt(v);
12     return 0;
13 }
```

- Què imprimeix aquest codi?

Definició d'accions

A teoria:

```
acció nom_acció (paràmetres) és
    $ Cos de l'acció
facció
```

En C:

Com especificar
paràmetres?
tipus nom_formal
Separats per comes

```
1 void nom_accio (int a, float b)
2 {
3     // Cos de l'acció
4 }
```

Exemple II

```
1 #include <stdio.h>
2
3 void suma2(int a, int b) {
4     int resultat = a + b;
5     printf("La suma es: %d\n",
6         resultat);
7 }
8
9 int main(){
10     int a = 10;
11     int b = 20;
12     suma2 (a,b);
13     return 0;
14 }
```

- Què imprimeix aquest codi?



Definició de funcions

A teoria:

```
funció nom_funció (paràmetres)
retorna tipus és
    $ Cos de la funció
ffunció
```

En C:

- En C, la única diferència entre acció i funció és que la funció retorna un tipus diferent de "void".

```
1 int nom_funcio (int a, float b){
2     //Cos de la funció
3     return xxx;
4 }
```

Tipus de retorn:
char, float, int, tipus d'usuari... etc

Paraula clau "return"
Retorna el valor de la funció al procediment superior

Exemple



```
1 #include <stdio.h>
2
3 int suma2(int a, int b) {
4     int resultat = a + b;
5     return resultat;
6 }
7
8 int main(){
9     int a = 10;
10    int b = 20;
11    int r;
12    r = suma2 (a,b);
13    printf("El resultat es: %d\n",r);
14    return 0;
15 }
```

- Què imprimeix aquest codi?

DETALLS IMPORTANTS

DELS PROCEDIMENTS EN C

Àmbit (scope) de les variables

- Què és un àmbit? En qualsevol llenguatge de programació, un àmbit (scope) és una regió del programa on una variable definida existeix, i fora de la regió no es pot accedir a la variable.
- Ja hem vist àmbits abans...

```
1 #include <stdio.h>
2
3 int main(){
4     int a = 10;
5     int b = 20;
6     for (int i=0; i<b; i++){
7         // Fer coses
8     }
9     printf("Puc accedir al valor de i?
    i=%d\n",i)
10     return 0;
11 }
```

```
scope.c:9:45: error: use of undeclared identifier 'i'
    printf("Puc accedir al valor de i? i=%d\n",i);
                                              ^
```

1 error generated.

Àmbit (scope) de les variables

- Què és un àmbit? En qualsevol llenguatge de programació, un àmbit (scope) és una regió del programa on una variable definida existeix, i fora de la regió no es pot accedir a la variable.
- Un procediment defineix un àmbit:
 - En l'exemple anterior, aquí podem veure dos àmbits:

```
1 #include <stdio.h>
2
3 void suma2(int a, int b) {
4     int resultat = a + b;
5     printf("La suma es: %d\n",
6 resultat);
7 }
8
9 int main(){
10     int a = 10;
11     int b = 20;
12     suma2 (a,b);
13     return 0;
14 }
```

Àmbit (scope) de les variables

- Què és un àmbit? En qualsevol llenguatge de programació, un àmbit (scope) és una regió del programa on una variable definida existeix, i fora de la regió no es pot accedir a la variable.

Àmbit 1

```
int a = 10;
```

Àmbit 2

```
int b = 2;
```

Àmbit 3

```
int c = 6;
```

- En C, una variable declarada en un àmbit **n**, és accessible des dels àmbits més interns però no des dels àmbits més externs.
- Per exemple, **a** serà accessible des de l'àmbit 1, l'àmbit 2 i l'àmbit 3
- Per exemple, **b** serà accessible des de l'àmbit 2 i l'àmbit 3, però no des de l'àmbit 1
- Per exemple, **c** serà accessible des de l'àmbit 3, però no des de l'àmbit 1 ni l'àmbit 2

Àmbit (scope) de les variables

- Què és un àmbit? En qualsevol llenguatge de programació, un àmbit (scope) és una regió del programa on una variable definida existeix, i fora de la regió no es pot accedir a la variable.

Àmbit 1

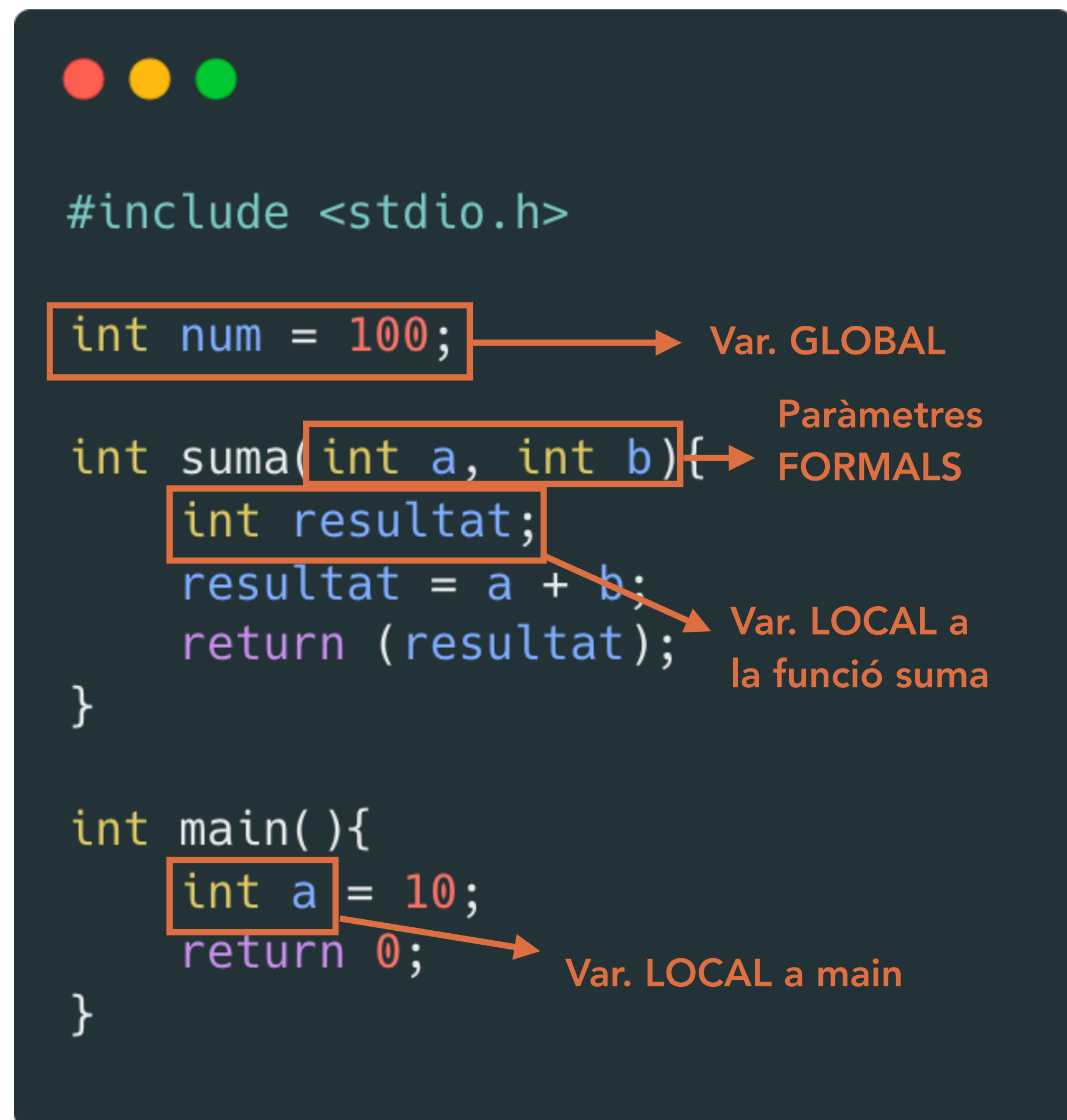
Àmbit 2

Àmbit 3

- Hi ha tres llocs on es poden declarar variables en C, i depenent d'on les declarem tindran accés a un àmbit o un altre.
- Si les declarem...
- ... dins una funció: es diuen **variables locals**.
- ... fora de totes les funcions: es diuen **variables globals**
- ... a la declaració d'un procediment: es diuen **paràmetres formals**

Àmbit (scope) de les variables

- Què és un àmbit? En qualsevol llenguatge de programació, un àmbit (scope) és una regió del programa on una variable definida existeix, i fora de la regió no es pot accedir a la variable.



The diagram shows a C code snippet with annotations for variable scope. The code is as follows:

```
#include <stdio.h>

int num = 100;

int suma(int a, int b){
    int resultat;
    resultat = a + b;
    return (resultat);
}

int main(){
    int a = 10;
    return 0;
}
```

Annotations and arrows:

- `int num = 100;` is boxed in orange, with an arrow pointing to the text **Var. GLOBAL**.
- `int suma(int a, int b){` is boxed in orange, with an arrow pointing to the text **Paràmetres FORMALS**.
- `int resultat;` is boxed in orange, with an arrow pointing to the text **Var. LOCAL a la funció suma**.
- `int a = 10;` is boxed in orange, with an arrow pointing to the text **Var. LOCAL a main**.

- Hi ha tres llocs on es poden declarar variables en C, i depenent d'on les declarem tindran accés a un àmbit o un altre.
- Si les declarem...
- ... dins una funció: es diuen **variables locals**.
- ... fora de totes les funcions: es diuen **variables globals**
- ... a la declaració d'un procediment: es diuen **paràmetres formals**

Àmbit (scope) de les variables

Variables locals

- Si declarem una variable dins d'una funció, aquestes variables només existiran dins de la funció i no seran accessibles fora d'ella.

```
1 #include <stdio.h>
2
3 // Procediment tonto
4 void accio_tonta (){
5     int x = 10;
6     printf("x=%d\n",x);
7 }
8
9 int main(){
10     accio_tonta();
11     printf("Puc accedir al valor de x? i=%d\n",x);
12     return 0;
13 }
```

error: use of undeclared identifier 'x'
printf("Puc accedir al valor de x? i=%d\n",x);

- Què diu? Puc accedir-hi o no?

- Des d'un àmbit superior (el main) no puc accedir a una variable que és local d'un àmbit inferior (l'acció tonta)

Àmbit (scope) de les variables

Variables globals

- Les variables globals són les que són accessibles des de **tots** els àmbits d'un programa.
- Els seus valors es mantenen mentre duri el programa i són accessibles des de totes les funcions.
- Es defineixen fora del main.

```
1 #include <stdio.h>
2
3 /* Variable global */
4 int g;
5
6 int main () {
7     /* Variables locals al main */
8     int a, b;
9     a = 10;
10    b = 20;
11    g = a + b; // Puc accedir a g?
12
13    printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
14    return 0;
15 }
```

- Puc accedir a "g"?

value of a = 10, b = 20 and g = 30

Àmbit (scope) de les variables

Variables globals

- Les variables globals són les que són accessibles des de **tots** els àmbits d'un programa.
- Els seus valors es mantenen mentre duri el programa i són accessibles des de totes les funcions.
- Es defineixen fora del main.



OJO CUIDAO!

- Es considera **mala pràctica** definir variables globals quan no es necessiten.
- Quan es necessiten? Quan la variable global conté un valor al qual hi ha d'accedir tothom sovint (e.g. clau de sessió, etc). És a dir, vosaltres de moment no en necessiteu.
- Quin perill tenen? Si tothom hi pot accedir (llegir i modificar), i hi ha un malfuncionament, és extremadament difícil trobar-ne el responsable (perquè tothom hi té accés)
- En general, **no en feu servir**. Fer servir variables locals és més farragós, però molt més segur.

Àmbit (scope) de les variables

Breu comentari sobre noms de variables i el seu scope

```
1 #include <stdio.h>
2
3 /* global variable declaration */
4 int g = 20;
5
6 int main () {
7
8     /* local variable declaration */
9     int g = 10;
10
11     printf ("value of g = %d\n", g);
12
13     return 0;
14 }
```

- **Pregunta:** puc declarar una variable amb el mateix nom dins de diferents scopes? Aquest programa compila?
- **Resposta:** sí que puc, són àmbits diferents.
- **Pregunta:** Ok, en aquest cas, quin valor imprimirà el printf? g=20 o g=10?
- **Resposta:** El valor de la variable local té preferència

value of g = 10


Àmbit (scope) de les variables

Paràmetres formals

- Els paràmetres formals d'una funció es consideren variables locals d'aquella funció
- **Pregunta:** Ok, en aquest cas, aquest codi és vàlid? Puc declarar dues variables amb el mateix nom (**a** i **b**) dins de la funció?
- **Resposta:** No. Com que els paràmetres formals a i b ja es consideren variables locals de la funció, si vull declarar variables amb el mateix nom tinc un error: no es poden declarar variables amb el mateix nom dins el mateix àmbit.

```
formalparams.c:4:6: error: redefinition of 'a'
    int a;
    ^
formalparams.c:3:15: note: previous definition is here
int suma2(int a, int b) {
    ^
formalparams.c:5:8: error: redefinition of 'b' with a different type:
'float' vs 'int'
    float b;
    ^
formalparams.c:3:22: note: previous definition is here
int suma2(int a, int b) {
```

```
1 #include <stdio.h>
2
3 int suma2(int a, int b) {
4     // Declaro variables locals
5     int a;
6     float b;
7     return (a+b);
8 }
9
10 int main(){
11     int a = 10;
12     int b = 20;
13     int r;
14     r = suma2 (a,b);
15     printf("El resultat es: %d\n",r);
16     return 0;
17 }
```




Intent de declaració de variables locals amb el mateix nom que els paràmetres formals

Declaració de múltiples funcions

Més d'una funció i overload

- Fins ara només hem definit una única funció.
En podem definir més.

Exemple



```
1 #include <stdio.h>
2
3 int suma(int a, int b) {
4     return a+b;
5 }
6
7 int resta(int a, int b){
8     return a-b;
9 }
10
11 int main(){
12     int a = 10;
13     int b = 20;
14     printf("Suma: %d\n", suma(a,b));
15     printf("Resta: %d\n", resta(a,b));
16     return 0;
17 }
```


Definició de dues funcions al mateix arxiu: suma i resta

Declaració de múltiples funcions

Més d'una funció i overload

- Fins ara només hem definit una única funció. En podem definir més.
- Això ens porta a una qüestió important:
Pregunta: puc definir més d'una funció amb el mateix nom però paràmetres diferents? (el que s'anomena *Function Overload*?)
 - Per exemple, voldria poder definir la funció suma amb tres operands.
- **Resposta:** No es pot fer en C. En altres llenguatges de programació sí, però en C no.

Exemple




```
1 #include <stdio.h>
2
3 int suma(int a, int b) {
4     return a+b;
5 }
6
7 int suma(int a, int b, int c) {
8     return a+b+c;
9 }
10
11 int main(){
12     int a = 10;
13     int b = 20;
14     printf("Suma: %d\n", suma(a,b));
15     printf("Resta: %d\n", resta(a,b));
16     return 0;
17 }
```

Function overload: escriure funcions amb el mateix nom i paràmetres diferents

Declaració vs. definició de funcions

- Fins ara sempre hem escrit les funcions abans de que comenci el programa "main", que és des d'on les cridem.
- **Pregunta:** Què passa si les posem després?
- **Resposta:** No compila. El primer cop que veu "resta(a,b)" es pensa que és una declaració i no la troba correcta.

```
error: implicit declaration of function 'resta' is
      invalid in C99 [-Werror,-Wimplicit-function-declaration]
      printf("Resta: %d\n", resta(a,b));
```



```
1 #include <stdio.h>
2
3 int main(){
4     int a = 10;
5     int b = 20;
6     printf("Resta: %d\n", resta(a,b));
7     return 0;
8 }
9
10 int resta(int a, int b){
11     return a-b;
12 }
```

Què passa si faig servir la funció "resta" i la seva definició va després de la funció des d'on la crido?

Declaració vs. definició de funcions

- Haver de definir sempre la funció abans de fer-la servir és limitant.
- **Pregunta:** Què podem fer-hi?
- **Resposta:** Podem declarar una funció i no definir-la fins més tard.

```
1 #include <stdio.h>
2
3 // DECLARACIÓ de la funció
4 int resta (int a, int b);
5
6 int main(){
7     int a = 10;
8     int b = 20;
9     printf("Resta: %d\n", resta(a,b));
10    return 0;
11 }
12
13 // DEFINICIÓ de la funció
14 int resta(int a, int b){
15     return a-b;
16 }
```

Exemple de declaració i definició d'una funció separades.

Declaració vs. definició de funcions

- Durant la **declaració** d'una variable/funció...
 - S'informa al compilador del nom de la variable/funció, el tipus que té i el valor inicial que té (si s'ha especificat). És a dir, la declaració dona detalls de les propietats d'una variable/funció.
- Durant la **definició** d'una variable/funció...
 - Es reserva un espai de memòria per aquella funció.
- En el **llenguatge C**...
 - La declaració i definició d'una variable no es pot separar:
`int a; // Declaració + definició`
 - En canvi, en una funció sí que es pot separar (veure l'exemple de la dreta).

```
1 #include <stdio.h>
2
3 // DECLARACIÓ de la funció
4 int resta (int a, int b);
5
6 int main(){
7     int a = 10;
8     int b = 20;
9     printf("Resta: %d\n", resta(a,b));
10    return 0;
11 }
12
13 // DEFINICIÓ de la funció
14 int resta(int a, int b){
15     return a-b;
16 }
```

Exemple de declaració i definició d'una funció separades.

PAS DE PARÂMETRES

Recordeu...

Pas de paràmetres

Pas per valor vs. pas per referència

- Distinció **molt important**
- En programació, distingim entre dos mecanismes de pas de paràmetres:

Pas per valor

Paràmetres **no** modificables

Qualsevol canvi que fem dins el procediment no tindrà efecte quan sortim del procediment

```
x := 1;  
y := 2;  
...  
intercanvia (x, y);  
...  
$ x val 1  
$ y val 2
```

Pas per referència

Paràmetres modificables

Els canvis que fem dins el procediment es mantindran al sortir-ne

```
x := 1;  
y := 2;  
...  
intercanvia (x, y);  
...  
$ x val 2  
$ y val 1
```

Cada llenguatge de programació gestiona el pas de paràmetres de manera diferent

En llenguatge C...

... es fa servir pas per valor

Això vol dir que mai podrem modificar els paràmetres?

No! Es fan servir adreces de memòria (anomenats punters o apuntadors) i llavors es pot accedir al contingut i modificar-lo

Pas de paràmetres en C

Pas per valor

- En C, el pas de paràmetres sempre es fa per valor. Això vol dir que no podem modificar allò que se'ns passa.
- Anem a comprovar-ho.
- Què imprimeix aquest codi?
 - a) Valor de x=10, y=20
 - b) Valor de x=20, y=10

Valor de x=10, y=20

- No s'ha pogut fer el swap perquè el pas de paràmetres es fa per valor.

```
1 #include <stdio.h>
2 // Accio swap que no funciona
3 void accio_swap(int a, int b){
4     int aux;
5     aux = a;
6     a = b;
7     b = aux;
8 }
9
10 int main(){
11     int x, y;
12     x = 10;
13     y = 20;
14     accio_swap(x,y);
15     printf("Valor de x=%d, y=%d\n",x,y);
16     return 0;
17 }
```

Taules com a paràmetres

- Si volem passar taules com a paràmetres de funcions, especifiquem a la capçalera que és una taula (amb `[]`)
- Hem d'especificar també la mida, si no, no podem recórrer-la.



```
1 #include <stdio.h>
2
3 /* Funcio que calcula l'average */
4 double getAverage(int arr[], int mida) {
5
6     int i;
7     double avg;
8     double suma = 0;
9
10    for (i = 0; i < mida; ++i) {
11        suma += arr[i];
12    }
13
14    avg = suma / mida;
15    return avg;
16 }
17
```

- Per cridar la funció, passem la taula **sense** `[]`

```
17
18 int main () {
19
20     /* Un array d'enters de 5 elements */
21     int balance[5] = {1000, 2, 3, 17, 50};
22     double avg;
23
24     /* Cridem a la funció */
25     avg = getAverage( balance, 5 );
26
27     /* Imprimir resultat */
28     printf( "Average value is: %f \n", avg );
29     return 0;
30 }
```

Taules com a paràmetres



OJO CUIDAO!

- Si la taula és bi-dimensional, hem de passar per paràmetre el nombre de columnes.

```
1 ...  
2 int maxim(int m[][], int fil, int col ){  
3     ...  
4 }
```

error: array has incomplete element type 'int []'

```
1 ...  
2 int maxim(int m[][NUM_COL], int fil){  
3     ...  
4 }
```

Taules de caràcters com a paràmetres

- Els strings són taules de caràcters. Per tant es passen de la mateixa manera.
- No cal passar la mida perquè podem esbrinar-la iterant fins trobar el caràcter sentinella `'\0'` o bé fent servir la funció **strlen** de la llibreria **string.h**

```
#include <stdio.h>
#define LEN 100

int llargada (char s[]){
    int i;
    i = 0;
    while (s[i]!='\0'){
        i++;
    }
    return i;
}
```

```
int main(){

    char nom[LEN] = "Perico de los Palotes";
    printf("Llargada del nom: %d\n", llargada(nom));
    return 0;
}
```


Taules com a paràmetres

- **Pregunta:** Puc modificar una taula des d'una funció?
- **Resposta:** Sí! Compte!!!
Considereu les taules com paràmetres d'entrada i sortida
- **Pregunta legítima:** I com pot ser? No havíem quedat que els paràmetres es passen per valor?
- **Resposta:** Sí, clar, però... ai, no us vull fer un spoiler, però... què és el nom del vector?

```
printf("Que val el vector balance    = %u\n", balance);  
printf("Que val  balance[0] = %u\n", balance[0]);
```

Què imprimeix aquest codi?

```
Que val el vector balance    = 3866085936  
Que val  balance[0] = 1000
```

Podem modificar un vector des de dins d'un procediment perquè quan el passem per paràmetre ja estem passant la seva adreça de memòria, i quan hi accedim fent `[]` des de la funció, ja estem accedint al seu valor original

```
17  
18 int main () {  
19  
20     /* Un array d'enters de 5 elements */  
21     int balance[5] = {1000, 2, 3, 17, 50};  
22     double avg;  
23  
24     /* Cridem a la funció */  
25     avg = getAverage( balance, 5 );  
26  
27     /* Imprimir resultat */  
28     printf( "Average value is: %f \n", avg );  
29     return 0;  
30 }
```

Programa principal del codi anterior

Pas de paràmetres en C

Pas per referència

- Existeix un mecanisme per poder modificar variables, i és obtenir l'adreça de memòria on resideix la variable en comptes del seu valor.
- Per a les taules, ja hem vist que és automàtic.
- Per la resta de variables, necessitem fer servir "punter" per això.
- Tornarem aquí quan sapiguem fer servir punters



Ai, quins nervis!

A CASA...

EXERCICIS L6 del Moodle

EL PROPER DIA...

PUNTERS! PER FI!