

# Tècniques de programació

---

Juan Mendoza Bravo  
juanmenbra@yahoo.es

# C++

## Unitat 9 Funcions

## Introducció

---

➤ Quan els programes es compliquen, es fan llargs i complexes, es millor dividir el programa.

➤ Aquestes divisions s'anomenen SUBPROGRAMES.

**Els avantatges que té utilitzar els subprogrames són:**

➤ La solució al programa és més fàcil i simple.

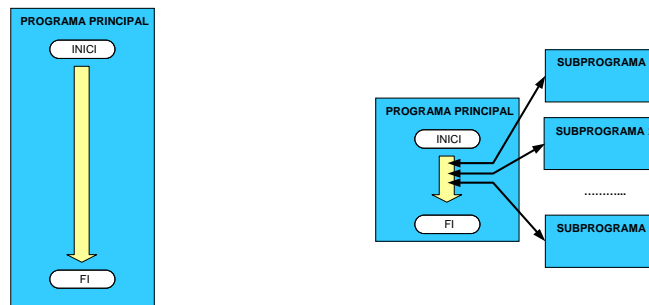
➤ Es poden reutilitzar tantes vegades com es vulgui, fins i tot en altres programes.

➤ Mes fàcil llegir el codi del programa i detectar errors.

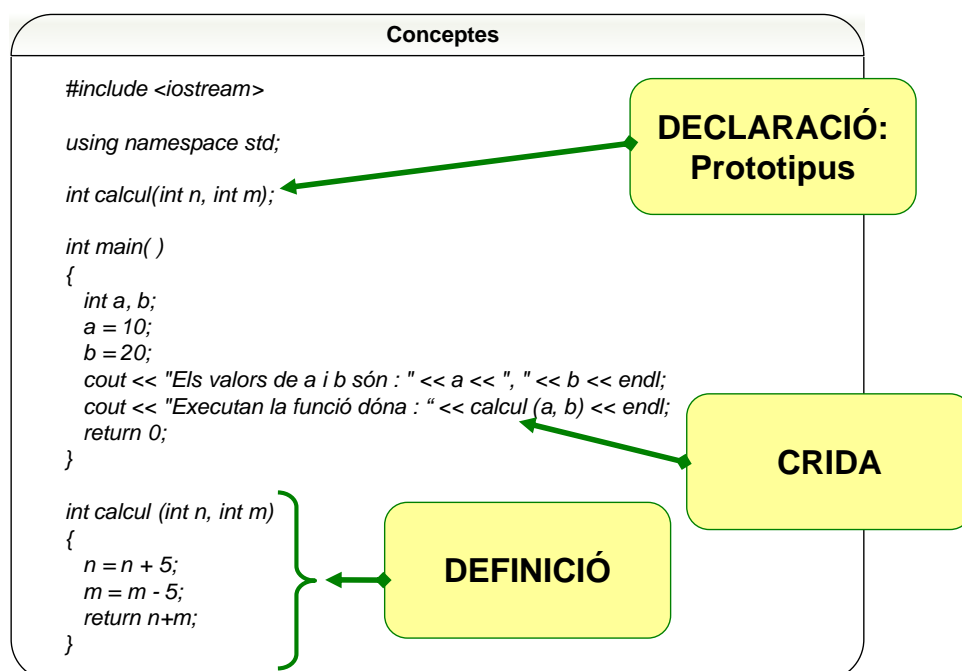
➤ Més útil en solucions de treball en equip, de diferents programadors.

## Introducció

- Molts llenguatges de programació tenen dos tipus de subprogrames:
  - ✓ Procediments.
  - ✓ Funcions.
- Els subprogrames en C++, **només són del tipus funcions.**
- Tots els programes en C++, tenen com a mínim una funció: ***main ()***.

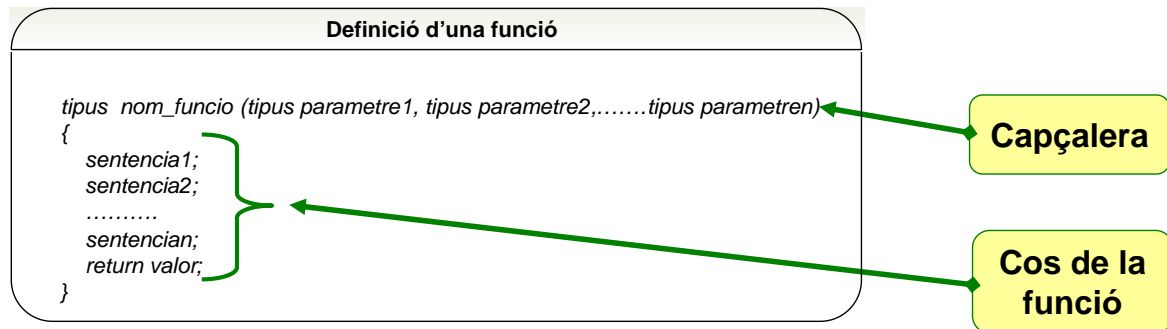


## Conceptes de les funcions



## Definició de funcions

Consisteix en escriure totes les sentències que componen la funció.  
**TASQUES QUE FA LA FUNCIO**



## Definició de funcions

Tipus de valor que retorna

Nom de la funció

Noms i tipus de paràmetres formals

Variable local

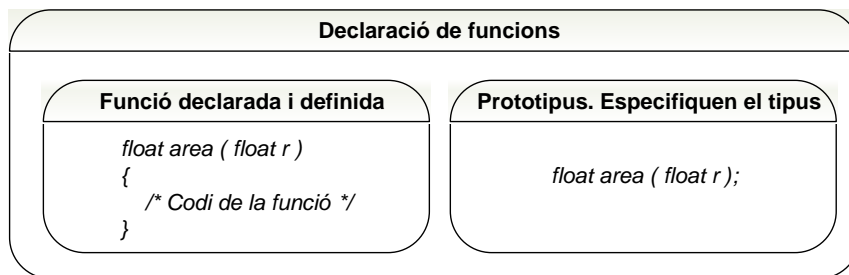
Valor que retorna la funció

```

float suma (float num1, float num2)
{
    float resultat;
    resultat = num1 + num2;
    return resultat;
}
  
```

## Declaració de funcions

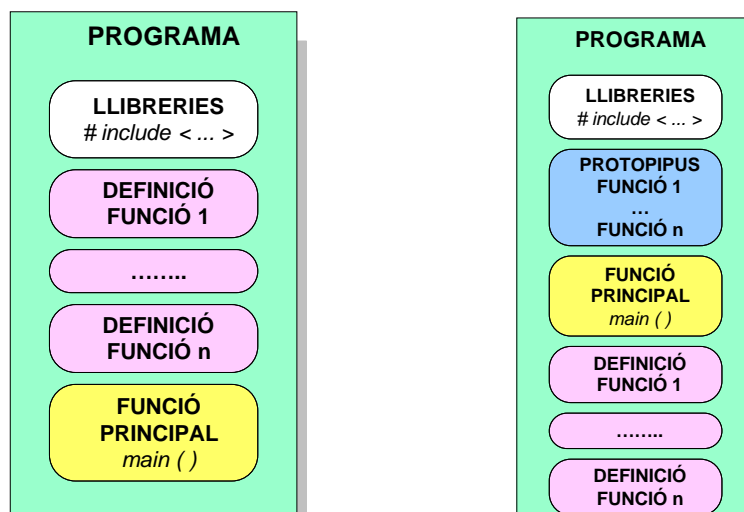
El compilador requereix que tota funció sigui declarada abans d'ésser invocada en el programa. Com qualsevol variable.  
**PROTOTIPUS**



## On definir les funcions ?

Es pot estructurar la definició de les funcions de dues maneres diferents:

- Abans de la funció principal *main ( )*
- Després de la funció principal *main ( )*



## Crida de funcions

Es fa la crida, quan es vol executar la funció.

Variable on  
s'emmagatzema  
el valor de retorn

Nom de la funció

Noms i tipus de paràmetres actuals

*variable* = *nom\_funcio* ( *parametres* )

Quan la funció necessita rebre dades o arguments per operar, aquests paràmetres es poden passar de dos maneres:

- Per valor.
- Per referència.

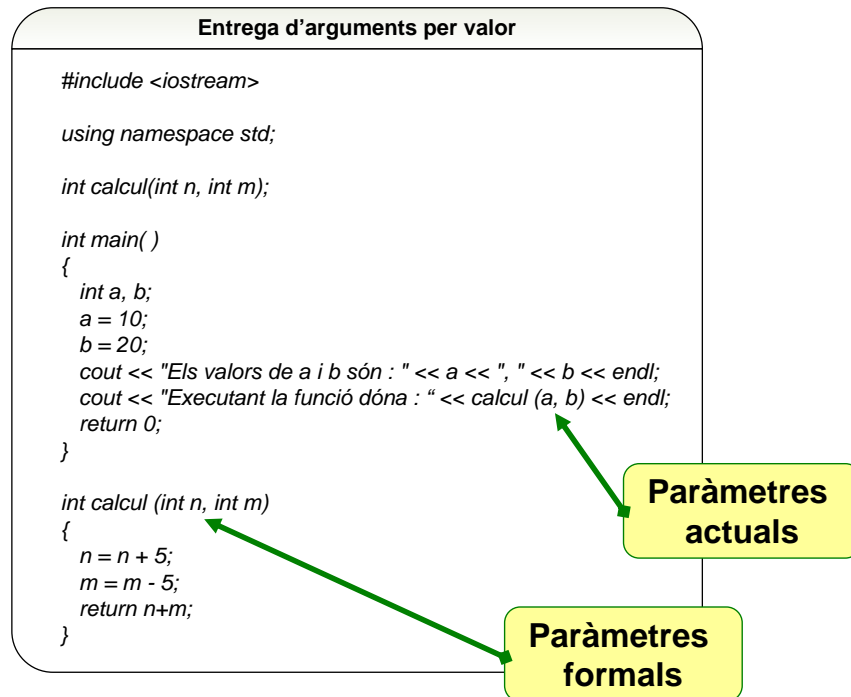
## Crida de funcions – Per valor

Els arguments o paràmetres actuals es copien en els paràmetres formals de la funció.  
Només es fa una còpia.

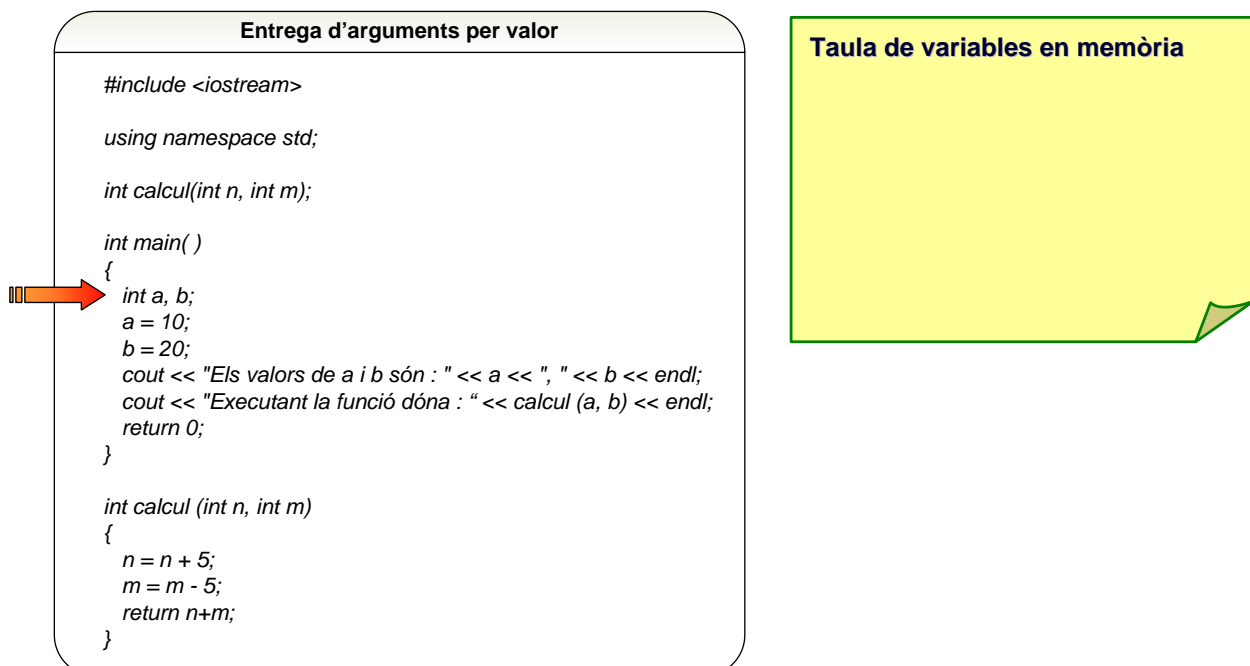
**A tenir molt en compte:**

- Les còpies dels paràmetres es fan per ordre. No pel nom.
- Al ser còpia, els paràmetres formals i actuals són variables distintes.
- Es poden modificar el paràmetres formals, però en cap cas es podran modificar el paràmetres actuals.

## Crida de funcions – Per valor



## Crida de funcions – Per valor



## Crida de funcions – Per valor

## Entrega d'arguments per valor

```
#include <iostream>

using namespace std;

int calcul(int n, int m);

int main( )
{
    int a, b;
    a = 10;
    b = 20;
    cout << "Els valors de a i b són : " << a << ", " << b << endl;
    cout << "Executant la funció dóna : " << calcul (a, b) << endl;
    return 0;
}

int calcul (int n, int m)
{
    n = n + 5;
    m = m - 5;
    return n+m;
}
```

## Taula de variables en memòria

main ( )	
a	b
10	20

Els valors de a i b són: 10 , 20

## Crida de funcions – Per valor

## Entrega d'arguments per valor

```
#include <iostream>

using namespace std;

int calcul(int n, int m);

int main( )
{
    int a, b;
    a = 10;
    b = 20;
    cout << "Els valors de a i b són : " << a << ", " << b << endl;
    cout << "Executant la funció dóna : " << calcul (a, b) << endl;
    return 0;
}

int calcul (int n, int m)
{
    n = n + 5;
    m = m - 5;
    return n+m;
}
```

## Taula de variables en memòria

main ( )		calcul ( )	
a	b	n	m
10	20	10	20

Els valors de a i b són: 10 , 20

## Crida de funcions – Per valor

## Entrega d'arguments per valor

```
#include <iostream>

using namespace std;

int calcul(int n, int m);

int main()
{
    int a, b;
    a = 10;
    b = 20;
    cout << "Els valors de a i b són : " << a << ", " << b << endl;
    cout << "Executant la funció dóna : " << calcul(a, b) << endl;
    return 0;
}

int calcul (int n, int m)
{
    n = n + 5;
    m = m - 5;
    return n+m;
}
```



## Taula de variables en memòria

main ( )		calcul ( )	
a	b	n	m
10	20	15	15
		20	20

Els valors de a i b són: 10 , 20

## Crida de funcions – Per valor

## Entrega d'arguments per valor

```
#include <iostream>

using namespace std;

int calcul(int n, int m);

int main()
{
    int a, b;
    a = 10;
    b = 20;
    cout << "Els valors de a i b són : " << a << ", " << b << endl;
    cout << "Executant la funció dóna : " << calcul(a, b) << endl;
    return 0;
}

int calcul (int n, int m)
{
    n = n + 5;
    m = m - 5;
    return n+m;
}
```



40

## Taula de variables en memòria

main ( )		<del>calcul ( )</del>	
a	b	n	m
10	20	15	15
		20	20

Els valors de a i b són: 10 , 20

## Crida de funcions – Per valor

### Entrega d'arguments per valor

```
#include <iostream>

using namespace std;

int calcul(int n, int m);

int main( )
{
    int a, b;
    a = 10;
    b = 20;
    cout << "Els valors de a i b són : " << a << ", " << b << endl;
    cout << "Executant la funció dóna : " << calcul (a, b) << endl;
    return 0;
}

int calcul (int n, int m)
{
    n = n + 5;
    m = m - 5;
    return n+m;
}
```

### Taula de variables en memòria

main ( )	
a	b
10	20

Els valors de a i b són: 10 , 20  
Executant la funció dóna: 40

## Crida de funcions – Per referència

A la funció s'entreguen les direccions del paràmetres, no els valors. Això li permet modificar els valors d'aquestes variables que s'entreguen.

### A tenir molt en compte:

- D'aquesta forma podem fer que una funció retorni tots el valors que es desitgin.
- La variable formal és un punter, que rep la còpia de la direcció de la variable actual, que és també un punter.
- En la definició de la funció, hem d'utilitzar el operador de punters "\*" cada vegada que utilitzem les variables formals.
- En la crida a la funció, hem d'utilitzar l'operador de direcció "&" davant de les variables que volem passar per referència.

## Crida de funcions – Per referència

### Entrega d'arguments per referència

```
#include <iostream>

using namespace std;

void intercanvi ( int *a, int *b);

int main( )
{
    int x, y;
    x = 5;
    y = 10;
    cout << "Els valors de x i y són : " << x << " , " << y << endl;
    intercanvi (&x , &y);
    cout << "Executant la funció dóna x i y són : ";
    cout << x << " , " << y << endl;
    return 0;
}

void intercanvi ( int *a, int *b);
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

### Taula de variables en memòria

## Crida de funcions – Per referència

### Entrega d'arguments per referència

```
#include <iostream>

using namespace std;

void intercanvi ( int *a, int *b);

int main( )
{
    int x, y;
    x = 5;
    y = 10;
    cout << "Els valors de x i y són : " << x << " , " << y << endl;
    intercanvi (&x , &y);
    cout << "Executant la funció dóna x i y són : ";
    cout << x << " , " << y << endl;
    return 0;
}

void intercanvi ( int *a, int *b);
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

### Taula de variables en memòria

main ( )	x	y
	5	10

Els valors de x i y són: 5 , 10

## Crida de funcions – Per referència

### Entrega d'arguments per referència

```
#include <iostream>

using namespace std;

void intercanvi ( int *a, int *b);

int main( )
{
    int x, y;
    x = 5;
    y = 10;
    cout << "Els valors de x i y són : " << x << " , " << y << endl;
    intercanvi (&x, &y);
    cout << "Executant la funció dóna x i y són : ";
    cout << x << " , " << y << endl;
    return 0;
}

void intercanvi ( in *a, in *b)
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

### Taula de variables en memòria

intercanvi ( )	*a	*b
	↓	↓
main ( )	x	y
	5	10

Els valors de x i y són: 5 , 10

## Crida de funcions – Per referència

### Entrega d'arguments per referència

```
#include <iostream>

using namespace std;

void intercanvi ( int *a, int *b);

int main( )
{
    int x, y;
    x = 5;
    y = 10;
    cout << "Els valors de x i y són : " << x << " , " << y << endl;
    intercanvi (&x, &y);
    cout << "Executant la funció dóna x i y són : ";
    cout << x << " , " << y << endl;
    return 0;
}

void intercanvi ( in *a, in *b)
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

### Taula de variables en memòria

intercanvi ( )	*a	*b	aux
	↓	↓	
main ( )	x	y	5
	10	5	

Els valors de x i y són: 5 , 10

## Crida de funcions – Per referència

### Entrega d'arguments per referència

```
#include <iostream>

using namespace std;

void intercanvi (int *a, int *b);

int main()
{
    int x, y;
    x = 5;
    y = 10;
    cout << "Els valors de x i y són : " << x << ", " << y << endl;
    intercanvi (&x , &y);
    cout << "Executant la funció dóna x i y són : ";
    cout << x << ", " << y << endl;
    return 0;
}

void intercanvi (int *a, int *b);
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

### Taula de variables en memòria

<del>intercanvi ( )</del>	<del>*a</del>	<del>*b</del>	<del>aux</del>
<del>main ( )</del>	<del>x</del>	<del>y</del>	<del>5</del>
	10	5	

Els valors de x i y són : 5 , 10

## Crida de funcions – Per referència

### Entrega d'arguments per referència

```
#include <iostream>

using namespace std;

void intercanvi (int *a, int *b);

int main()
{
    int x, y;
    x = 5;
    y = 10;
    cout << "Els valors de x i y són : " << x << ", " << y << endl;
    intercanvi (&x , &y);
    cout << "Executant la funció dóna x i y són : ";
    cout << x << ", " << y << endl;
    return 0;
}

void intercanvi (int *a, int *b);
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

### Taula de variables en memòria

intercanvi ( )		
main ( )	x	y
	10	5

Els valors de x i y són : 5 , 10  
Executant la funció dóna x i y són :  
10 , 5

## Relació entre funcions i vectors/taules

Els **vectors**, normalment, s'entreguen i retornen per referència.  
Hi ha maneres diferents d'especificar els paràmetres.

**Incloent un punter en els arguments,  
com qualsevol altra variable.**

*Tipus nom\_funcio (tipus **\*vector**, parametre2.....)*

**Indicant amb claudadors que  
l'argument és un vector.**

*Tipus nom\_funcio (tipus **vector [ ]**, parametre2.....)*

## Relació entre funcions i vectors/taules

En les **taules**, cal especificar totes les dimensions  
excepte la primera que és opcional.  
Hi ha maneres diferents d'especificar els paràmetres.

**Incloent un punter en els arguments, com qualsevol  
altra variable i incloent el n<sup>o</sup> de columnes.**

*Tipus nom\_funcio (tipus (**\*taula**)[10], parametre2.....)*

**Indicant amb claudadors que l'argument  
és una taula i incloent en n<sup>o</sup> de columnes.**

*Tipus nom\_funcio (tipus **taula [ ] [10]**, parametre2.....)*

## Relació entre funcions i vectors/taules

### Relació funcions i vectors

```
#include <iostream.h>

void per_dos (int vector [ ]);

void main(void)
{
    int sumands[ 3 ] = { 0 };
    cout<<"Introdueixi els 3 números que vulgui doblar:"<<endl;
    cin>>sumands[0]>>sumands[1]>>sumands[2];
    per_dos(sumands);
    cout<<"Els nous valors són "<<endl<<sumands[0]<<endl;
    cout<<sumands[1]<<endl<<sumands[2]<<endl;
}

void per_dos (int vector [ ])
{
    int i;

    for (i=0 ; i<3 ; i++)
        {
            vector [ i ] = 2 * vector[i ];
        }
}
```

### Relació funcions i taules

```
#include <iostream.h>

void per_dos (int taula [ ][ 3 ]);

void main(void)
{
    int elements[ 2 ][ 2 ] = { 0 };
    cout<<"Introdueixi els números que vulgui doblar:"<<endl;
    cin>>elements[ 0 ][ 0 ]>>elements[ 0 ][ 1 ];
    cin>>elements[ 1 ][ 0 ]>>elements[ 1 ][ 1 ];
    per_dos(elements);
    cout<<"Els nous valors són "<<endl;
    cout<<elements[ 0 ][ 0 ]<<" "<<elements[ 0 ][ 1 ]<<endl;
    cout<<elements[ 1 ][ 0 ]<<" "<<elements[ 1 ][ 1 ]<<endl;
}

void per_dos (int taula [ ][ 3 ])
{
    int i , j;

    for (i=0 ; i<2 ; i++)
        {
            for (j=0 ; j<2 ; j++)
                {
                    taula[ i ][ j ] = 2 * taula[ i ][ j ];
                }
        }
}
```

## Relació entre funcions i vectors/taules

### Relació funcions i vectors

```
#include <iostream.h>

void per_dos (int vector [ ]);

void main(void)
{
    int sumands[ 3 ] = { 0 };
    cout<<"Introdueixi els 3 números que vulgui doblar:"<<endl;
    cin>>sumands[0]>>sumands[1]>>sumands[2];
    per_dos(sumands);
    cout<<"Els nous valors són "<<endl<<sumands[0]<<endl;
    cout<<sumands[1]<<endl<<sumands[2]<<endl;
}

void per_dos (int vector [ ])
{
    int i;

    for (i=0 ; i<3 ; i++)
        {
            vector [ i ] = 2 * vector[i ];
        }
}
```

### Relació funcions i taules

```
#include <iostream.h>

void per_dos (int taula [ ][ 3 ]);

void main(void)
{
    int elements[ 2 ][ 2 ] = { 0 };
    cout<<"Introdueixi els números que vulgui doblar:"<<endl;
    cin>>elements[ 0 ][ 0 ]>>elements[ 0 ][ 1 ];
    cin>>elements[ 1 ][ 0 ]>>elements[ 1 ][ 1 ];
    per_dos(elements);
    cout<<"Els nous valors són "<<endl;
    cout<<elements[ 0 ][ 0 ]<<" "<<elements[ 0 ][ 1 ]<<endl;
    cout<<elements[ 1 ][ 0 ]<<" "<<elements[ 1 ][ 1 ]<<endl;
}

void per_dos (int taula [ ][ 3 ])
{
    int i , j;

    for (i=0 ; i<2 ; i++)
        {
            for (j=0 ; j<2 ; j++)
                {
                    taula[ i ][ j ] = 2 * taula[ i ][ j ];
                }
        }
}
```

## Relació entre funcions i vectors/taules

### Relació funcions i vectors

```
#include <iostream.h>

void per_dos (int vector [ ]);

void main(void)
{
    int sumands[ 3 ] = { 0 };
    cout<<"Introdueixi els 3 números que vulgui doblar:"<<endl;
    cin>>sumands[0]>>sumands[1]>>sumands[2];
    per_dos(sumands);
    cout<<"Els nous valors són "<<endl<<sumands[0]<<endl;
    cout<<sumands[1]<<endl<<sumands[2]<<endl;
}

void per_dos (int vector [ ])
{
    int i;

    for (i=0 ; i<3 ; i++)
        {
            vector [ i ] = 2 * vector[i ];
        }
}
```

### Relació funcions i taules

```
#include <iostream.h>

void per_dos (int taula [ ][ 3 ]);

void main(void)
{
    int elements[ 2 ][ 2 ] = { 0 };
    cout<<"Introdueixi els números que vulgui doblar:"<<endl;
    cin>>elements[ 0 ][ 0 ]>>elements[ 0 ][ 1 ];
    cin>>elements[ 1 ][ 0 ]>>elements[ 1 ][ 1 ];
    per_dos(elements);
    cout<<"Els nous valors són "<<endl;
    cout<<elements[ 0 ][ 0 ]<<" "<<elements[ 0 ][ 1 ]<<endl;
    cout<<elements[ 1 ][ 0 ]<<" "<<elements[ 1 ][ 1 ]<<endl;
}

void per_dos (int taula [ ][ 3 ])
{
    int i , j;

    for (i=0 ; i<2 ; i++)
        {
            for (j=0 ; j<2 ; j++)
                {
                    taula[ i ][ j ] = 2 * taula[ i ][ j ];
                }
        }
}
```

## Tipus de variables: globals i locals

### VARIABLES GLOBALES

- Són definides fora de qualsevol funció. (incloent *main ( )*).
- El seu àmbit és tot el programa.
- Existeixen i es poden utilitzar en tot el programa.

### VARIABLES LOCALS

- Són definides dintre de qualsevol funció o bloc d'instruccions (incloent *main ( )*).
- El seu àmbit és només en la funció o bloc on està definida.
- Només existeixen en la funció o bloc on estan definides.

### RECOMANACIONS

- Evitar sempre les variables globals.

## Tipus de variables: globals i locals

```
#include <iostream.h>

int q;           // Variable global

void main(void)
{
    int a;       // Variable local en main

    a = 10;
    q = 0;

    {
        int b;   // Variable local en aquest bloc

        b = 50;
        a = 20;
        q = 1;
    }
}
```