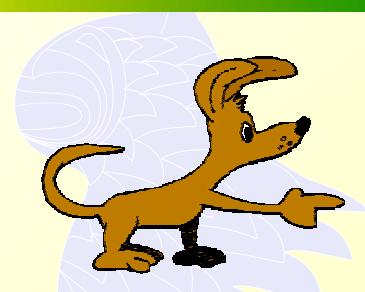


Metodología de la Programación

Tema 1.- Punteros

Los punteros

- Punteros, vectores y cadenas**
- Punteros y struct**
- Punteros y funciones**
- Punteros y const**



ugr Universidad de Granada

DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 1

Los punteros

¿Qué es un puntero?

Es un **dato** que almacena la *dirección de memoria* de otro **dato**.

¿Cómo se declara un puntero?

```
<tipo_base> *varptr;
```

`char var=50; (1 byte)`

`char *p; (4 bytes)`

`p = &var;`

La dirección de var

Dir. Memoria

| | | |
|-------|----------|-----|
| 35672 | 00110010 | var |
| 42173 | 00000000 | p |
| 42174 | 00000000 | |
| 42175 | 10001011 | |
| 42176 | 01011000 | |

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 2

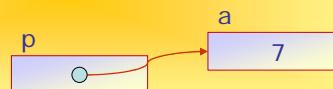
Los punteros 

Operadores de direccionamiento

Operador & (... la dirección de ...)

Devuelve la dirección de un dato de cualquier tipo (es decir, un puntero).

```
int a=7;
int *p=&a;
```



Operador * (... lo apuntado por ... o ... el contenido de ...)

Devuelve el valor del objeto apuntado por un puntero.

```
int b;
b = *p + 2;
*p = b + 3;
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  3

Los punteros 

Ejemplos

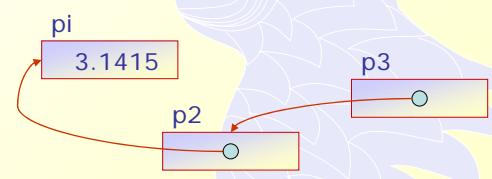
```
char a=10;
char *p1=&a;
```



```
unsigned int x=10;
unsigned int *p4=&x;
```



```
float pi=3.1415;
float *p2=&pi;
float **p3=&p2;
```



```
p1 = &pi; // ¿es correcto?
p3 = &pi; // ¿es correcto?
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  4

Los punteros

Asignación e inicialización

La asignación sólo está permitida entre **punteros de igual tipo**.

```
int a=7;
int *p1=&a;
char *p2=&a;
int *p3=p1;
```

Error → Solución →

```
char *p2 = reinterpret_cast<char*>(&a);
```

Un puntero ha de estar **correctamente inicializado** antes de ser usado.

```
int a=7;
int *p1=&a, *p2;
*p1 = 20;
*p2 = 30;
```

→ Error

Es conveniente inicializar al puntero nulo (**0**) en la declaración.

```
int *p2=0;
```

NULL es el puntero nulo en C.

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 5

Los punteros

Aritmética de punteros

Operadores relacionales: `<`, `>`, `<=`, `>=`, `!=`, `==`

Al usar estos operadores el valor del puntero (la dirección que almacena) se comporta como un número entero.

```
int *p, *q;
```

```
if (p==q) ...
```

```
if (p<q) ...
```

Operadores aritméticos: `+`, `-`, `++`, `--`, `+=`, `-=`

El valor del puntero se comporta **CASI** como un entero.

Al sumar o restar un número **N** al valor del puntero, este se incrementa o decremente **N*sizeof(tipobase)** unidades enteras.

```
char a=7, *p=&a, *q;
p++;
q=p-2;
```

| | | | | | | | |
|-------|---|---|--|--|--|--|--|
| 14562 | | | | | | | |
| 14563 | 7 | a | | | | | |
| 14564 | | | | | | | |

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 6

Los punteros

Aritmética de punteros

```
int a=7, *p=&a, *q;
p++;
q=p-2;
```

```
*p = 4;
```

Error: esa zona de memoria no es nuestra

```
char *pc=reinterpret_cast<char*>(&a);
cout << (int)(*pc) << endl;      pc++;
cout << (int)(*pc) << endl;      pc+=1;
cout << (int)(*pc) << endl;      pc=pc+1;
cout << (int)(*pc) << endl;      pc++;
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 7

Metodología de la Programación

Tema 2.- Punteros

- Los punteros**
- Punteros, vectores y cadenas**
- Punteros y struct**
- Punteros y funciones**
- Punteros y const**

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 8

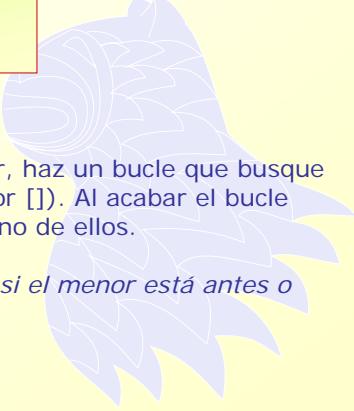
 **Punteros, vectores y cadenas** 
Punteros y vectores

La aritmética de punteros es especialmente interesante para movernos por vectores (en un vector los elementos están consecutivos).

```
int v[10] = {3,5,2,7,6,7,5,1,2,5};
int *p=&(v[0]);
for (int i=0; i<10; i++, p++)
    cout << *p << endl;
```

Ejercicio: Dado un vector como el anterior, haz un bucle que busque el máximo y el mínimo (sin usar el operador []). Al acabar el bucle tendremos un puntero apuntando a cada uno de ellos.

¿Cómo se puede saber (de forma sencilla) si el menor está antes o después que el mayor?



Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  9

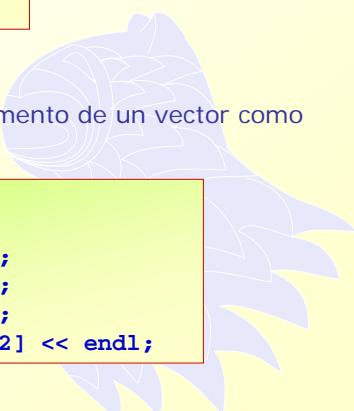
 **Punteros, vectores y cadenas** 
Punteros y vectores

En C++ podemos usar un vector como un puntero al primer elemento:

```
int v[10] = {3,5,2,7,6,7,5,1,2,5};
cout << *v << endl;
cout << *(v+3) << endl;
```

En C++ podemos usar un puntero a un elemento de un vector como un vector que comienza en ese elemento:

```
int v[10] = {3,5,2,7,6,7,5,1,2,5};
int *p;
p=&(v[4]);      cout << *p << endl;
p=v+2;          cout << *p << endl;
p++;            cout << *p << endl;
p=&(v[3])-2;    cout << p[0] << p[2] << endl;
```

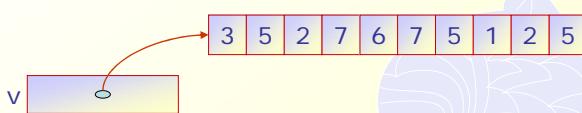


Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  10

Punteros, vectores y cadenas  **Punteros y vectores** 

En definitiva: podemos ver un vector como un puntero constante al primer elemento del mismo:

```
int v[10] = {3,5,2,7,6,7,5,1,2,5};
```



v es lo mismo que &(v[0])

```
int a=3;
v = &a; → Error, v es const y no puede modificarse
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  11

Punteros, vectores y cadenas  **Punteros y vectores** 

Recorrer e imprimir los elementos de un vector:

```
int v[10] = {3,5,2,7,6,7,5,1,2,5};
for (int i=0; i<10; i++)
    cout << v[i] << endl;
```

```
int v[10] = {3,5,2,7,6,7,5,1,2,5};
int *p=v;
for (int i=0; i<10; i++)
    cout << *(p++) << endl;
```

```
int v[10] = {3,5,2,7,6,7,5,1,2,5};
int *p=v;
for (; p<v+10; ++p)
    cout << *p << endl;
```

Ejercicio: Sumar 8 a todos los elementos de un vector de enteros.
Ejercicio: Invertir un vector (sin usar el operador []).

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  12

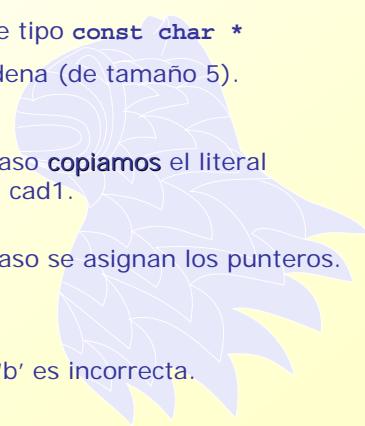
 **Punteros, vectores y cadenas** 

Una cadena (*estilo C*) es un vector de **char** acabado en 0.

```
char *cad1;
```

Una constante literal de tipo cadena es de tipo **const char ***

"Hola" es una constante literal de cadena (de tamaño 5).



`char cad1[]="Hola";` → En este caso **copiamos** el literal "Hola" en cad1.

`char *cad2="Hola";` → En este caso se asignan los punteros.
`cad2[2] = 'b';`

En el segundo, la asignación de 'b' es incorrecta.

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  13

 **Metodología de la Programación** 

Tema 2.- Punteros

- Los punteros**
- Punteros, vectores y cadenas**
- Punteros y struct**
- Punteros y funciones**
- Punteros y const**



Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  14

Punteros y struct 
El operador ->

También podemos tener punteros a **struct**.

```

struct A {
    int x;
    float y;
};

A v1;
A *pa;
pa = &v1;

```

```

struct B {
    int x;
    A *p;
};

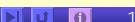
B v2;
B *pb;
pb = &v2;
v2.p = &v1; ← (*pb).p = &v1;

```

Operador ->

Si p es un puntero a un **struct** podemos acceder a sus miembros:

| | |
|----------------------|--------------------------------------|
| (*p).miembro | <i>Cuidado con el paréntesis !!!</i> |
| p->miembro | |

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  15

Punteros y struct 
Ejemplo

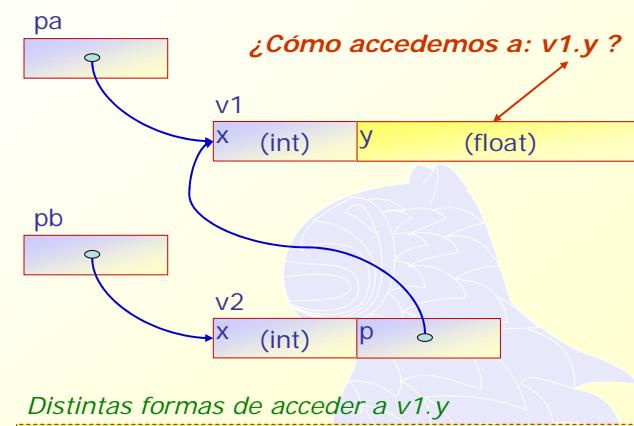
struct A {
 int x;
 float y;
};

A v1;
A *pa;
pa = &v1;

struct B {
 int x;
 A *p;
};

B v2;
B *pb;
pb = &v2;
v2.p = &v1;

¿Cómo accedemos a: v1.y ?



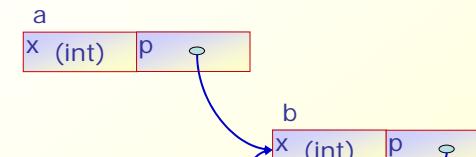
Distintas formas de acceder a v1.y

| | | |
|---------|-------------|-----------------|
| v1.y | v2.p->y | pb->p->y |
| (*pa).y | (*(v2.p)).y | (*pb).p->y |
| pa->y | | (**((*pb).p)).y |

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  16

Punteros y struct 

Ejemplo

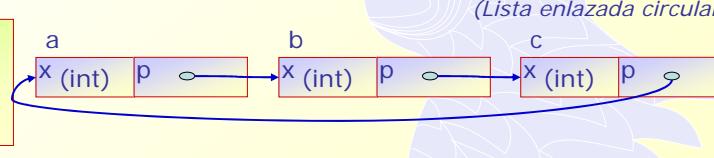


```

struct Celda {
    int x;
    Celda *p;
};

Celda a, b;
a.p = &b;
b.p = &b;

```

```

Celda a,b,c;
a.p = &b;
b.p = &c;
c.p = &a;

```

(Lista enlazada circular)

¿Cómo acceder al miembro *x* de *b*?

b.x a.p->x c.p->p->x (*((*(c.p)).p)).x

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 17

Punteros y struct 

Ejemplo

Representa gráficamente la disposición de la memoria

```

struct Celda {
    int d;
    Celda *p1, *p2, *p3;
} a, b, c, d;

a.d = b.d = c.d = d.d = 0;

a.p1 = &c;           c.p3 = &d;           a.p2 = a.p1->p3;
d.p1 = &b;           a.p3 = c.p3->p1;   a.p3->p2 = a.p1;
a.p1->p1 = &a;     a.p1->p3->p1->p2 = c.p3->p1;
c.p1->p3->p1 = &b; (*((*(c.p3->p1)).p2->p3)).p3 = a.p1->p3;
d.p2 = b.p2;
(*(a.p3->p1)).p2->p2->p3 = (*a.p3->p2).p3->p1->p2;

a.p1->p2->p2->p1->d = 5;    d.p1->p3->p1->p2->p1->d = 7;
(*(d.p1->p3)).p3->d = 9;    c.p1->p2->p3->d = a.p1->p2->d - 2;
(*(c.p2->p1)).p2->d = 10;

cout << "a=" << a.d << " b=" << b.d << " c=" << c.d << " d=" << d.d << endl;

```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 18

Punteros y struct

Ejemplo

```

struct SA {
    int dat;
    SB *p1;
};

struct SB {
    int dat;
    SA *p1;
    SC *p2;
};

struct SC {
    SA *p1;
    SB *p2;
    SD *p3;
};

struct SD {
    int *p1;
    SB *p2;
};

```

Representa gráficamente la disposición de la memoria

Es necesaria la declaración adelantada !

Variables

```

SA a;           SB b;
SC c;           SD d;
int dat;

```

```

a.dat = b.dat = dat = 0;
a.p1 = &b;          b.p1 = &a;          b.p2 = &c;
c.p1 = b.p1;      c.p2 = &(*(a.p1));  c.p3 = &d;
d.p1 = &dat;        d.p2 = &(*(c.p1)->p1);
*(d.p1) = 9;      (*(b.p2)->p3->p2)->p1).dat = 1;
(*((*(c.p3->p2)).p2->p3)).p1) = (*(b.p2)).p1->dat + 5;

cout<<"a.dat="<<a.dat<<"b.dat="<<b.dat<<"dat="<<dat<<endl;

```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 19

Metodología de la Programación

Tema 2.- Punteros

- Los punteros**
- Punteros, vectores y cadenas**
- Punteros y struct**
- Punteros y funciones**
- Punteros y const**



ugr Universidad de Granada DECSAI Universidad de Granada

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 20

Punteros y funciones 

Paso de parámetros

Paso por valor

```
int doble(int x) {
    x = x*x;
    return x;
}

...
int y=3, z;
z = doble(y);
...
```

Paso por referencia

```
void func(int *a) {
    int b=4;
    *a = 5;
    a = &b;
    *a = *a + 3;
}

int x=3, *y=&x;
func(y);
func(&x);
```

¿Paso por valor o por referencia?

Implementa una función que reciba un puntero a entero y que:

- 1.- Ponga a cero el dato apuntado.
- 2.- Ponga a cero el puntero.

```
int a=6;
int *q;
q = &a;
HacerCero(q);
cout << a << q;      // Debería salir 0 0
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  21

Punteros y funciones 

Devolución de punteros a variables locales

¿Qué ocurre en las siguientes implementaciones?

```
int func(int x)
{
    int a;
    a = x*2;
    return a;
}

...
int x;
x = func(3);
cout << x << endl;
```

Devolvemos una copia de una variable local

```
int *func(int x)
{
    int a;
    a = x*2;
    return &a;
}

...
int *x;
x = func(3);
cout << *x << endl;
```

Devolvemos un puntero a una variable local

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  22

Punteros y funciones

Parámetros de tipo vector

```

int doble(int v[], int p)
{
    return v[p]*2;
}

int doble(int *v, int p)
{
    return v[p]*2;
}

```

El vector v se pasa por valor.
v se comporta como un puntero.
Los elementos de v NO se pasan como parámetros.

```

...
int v[8] = {5,4,3,7,6,8,2,3};
cout << doble(v,2) << endl;
cout << doble(&v[3],2) << endl;

```

¿Se modifica v[3] en el siguiente código (el paso de v es por valor)?

```

void duplica(int v[], int p) {
    v[p] *= 2;
}

int v[6] = {5,4,3,7,6,8};
duplica(v,3);
cout << v[3] << endl;

```

Metodología de la Programación

Tema 2.- Punteros

Los punteros

Punteros, vectores y cadenas

Punteros y struct

Punteros y funciones

Punteros y const

ugr Universidad de Granada

DECSAI Universidad de Granada

Punteros y const

Diferencias entre el valor apuntado y el puntero

Cuando tratamos con punteros tenemos dos cosas:

- 1.- El dato puntero.
- 2.- El dato que es apuntado.

Puede ocurrir que:

- Ninguno sea **const**.
- Sólo el dato apuntado sea **const**.
- Sólo el puntero sea **const**.
- Los dos sean **const**.

| |
|------------------------|
| double *p; |
| const double *p; |
| double *const p; |
| const double *const p; |

Para que un puntero sea constante se cambia * por *const en la declaración:

const int *
p;
↑

int *
const
p;
↓

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 25

Punteros y const

Punteros const y no const

Un puntero **const** es distinto de un puntero **no const**.

¿Qué ocurre en las siguientes situaciones?

| | | | | |
|---|--|--|--|--|
| int *p; const int a=2; p = &a; | int *v1; int * const v2; const int * v3; const int * const v4; | | | |
| const int *p; int a; p = &a; *p = 7; a = 8; | v1 = v2; v1 = v3; v1 = v4; *v1=*v2; *v1=*v3; *v1=*v4; | v2 = v1; v2 = v3; v2 = v4; *v2=*v1; *v2=*v3; *v2=*v4; | v3 = v1; v3 = v2; v3 = v4; *v3=*v1; *v3=*v2; *v3=*v4; | v4 = v1; v4 = v2; v4 = v3; *v4=*v1; *v4=*v2; *v4=*v3; |

Es posible asignar un puntero no const a uno const pero no al revés (en la asignación se hace una conversión implícita).

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 26