
INFORMÁTICA INDUSTRIAL

Tercera clase PROGRAMACIÓN BÁSICA C++

Profesores: F. Abu-Dakka / A. Castro / M. Abderrahim
Departamento de Ingeniería de Sistemas y Automática



8. Punteros

Punteros



Escuela Politécnica Superior



Nombre de la variable



C/Butarque 15
28911 Leganés



Dirección de la variable



Punteros

1000	1003	p
1001		
1002		
1003	64	a
1004		

```
int a = 64;  
int *p = &a;
```

- a es una variable que vale 64
- p es una variable que vale la dirección de memoria de otra variable

Punteros

- Es una variable que señala a otra variable o función
- Declaración:

```
tipo* nombre_del puntero;  
int* p;
```
- p es un puntero a una variable de tipo entero.
- p NO es un entero

Punteros

- `&` se usa para obtener la dirección de variables
- `*` se usa para obtener el valor que hay en una dirección de memoria

Inicialización de punteros

```
int i;
```

```
int *p;
```

```
i=5;
```

```
p=&i;
```

1000

1003

p

1001

1002

1003

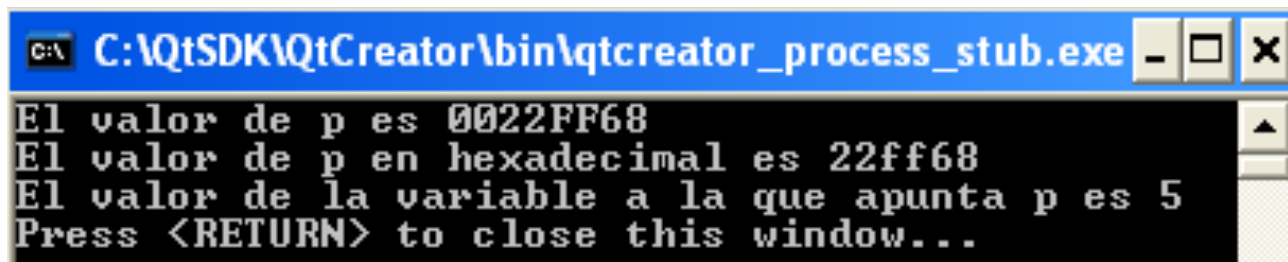
5

i

1004

Inicialización de punteros

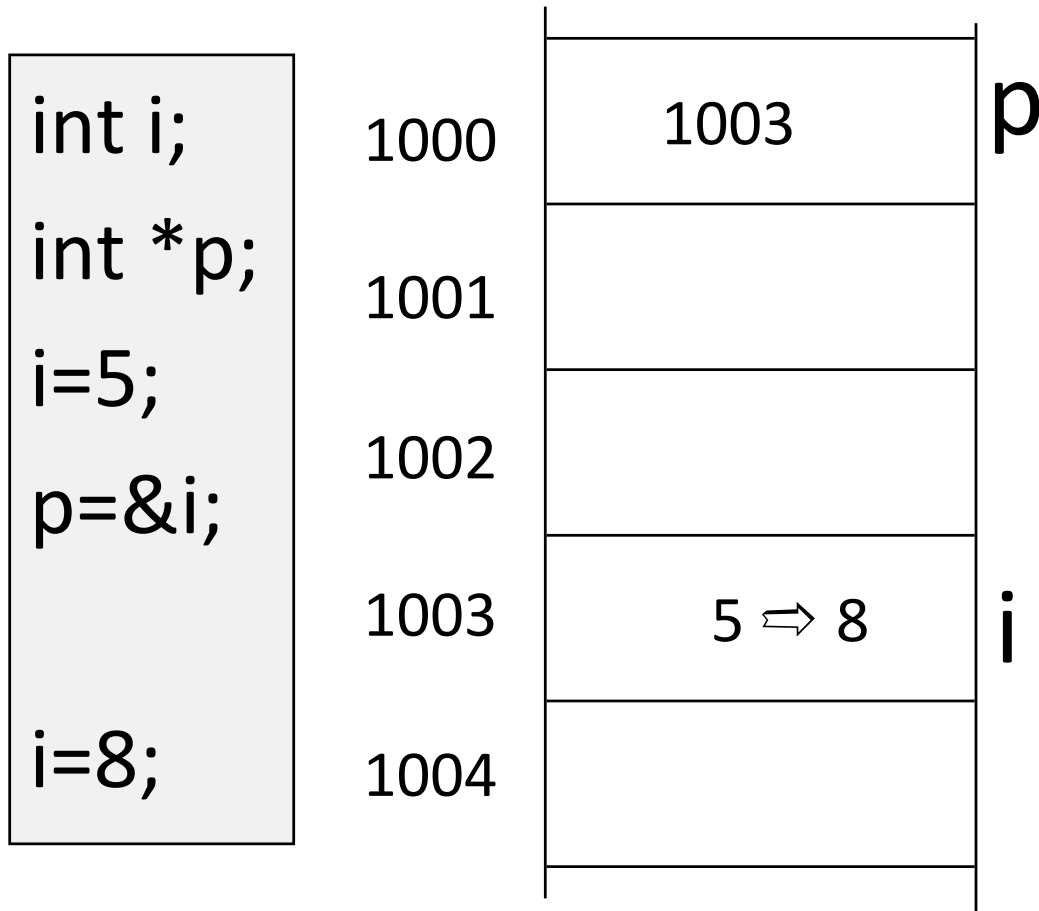
```
#include <iostream>
using namespace std;
int main(){
    int i = 5;
    int *p;
    p = &i;
    cout<<"El valor de p es "<< p << endl;
    cout<<"El valor de p en hexadecimal es "<< hex << p << endl;
    cout<<"El valor de la variable a la que apunta p es "<< *p << endl;
    return 0;
}
```



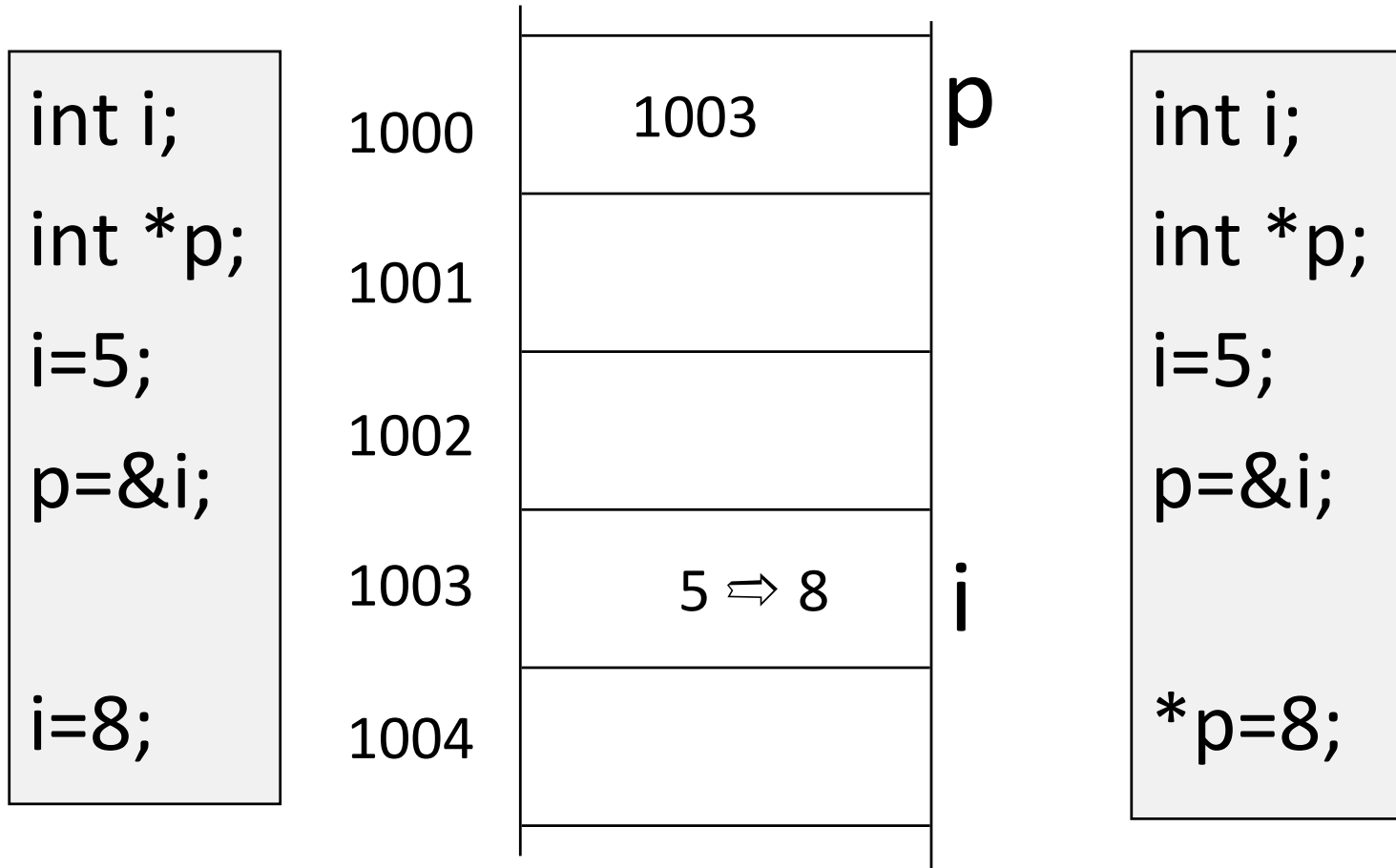
The screenshot shows a command prompt window titled "C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe". The output of the program is displayed as follows:

```
El valor de p es 0022FF68
El valor de p en hexadecimal es 22ff68
El valor de la variable a la que apunta p es 5
Press <RETURN> to close this window...
```

Acceso a variables por punteros



Acceso a variables por punteros

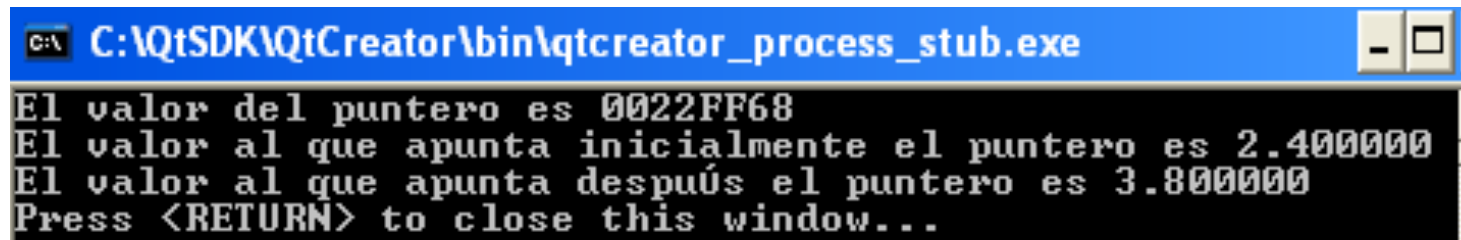


Acceso a variables por punteros

```
#include <iostream>
using namespace std;
int main(){
    float f;
    float *p;
    f = 2.4;
    p = &f;
    cout<<"El valor del puntero es "<< p << endl;
    cout<<"El valor al que apunta inicialmente el puntero es "<< *p << endl;
    f = 3.8;
    cout<<"El valor al que apunta después el puntero es "<< *p << endl;
    return 0;
}
```

Acceso a variables por punteros

```
#include <iostream>
using namespace std;
int main(){
    float f;
    float *p;
    f = 2.4;
    p = &f;
    cout<<"El valor del puntero es "<< p << endl;
    cout<<"El valor al que apunta inicialmente el puntero es "<< *p << endl;
    f = 3.8;
    cout<<"El valor al que apunta después el puntero es "<< *p << endl;
    return 0;
}
```



The screenshot shows a console window titled "C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe". The output text is as follows:

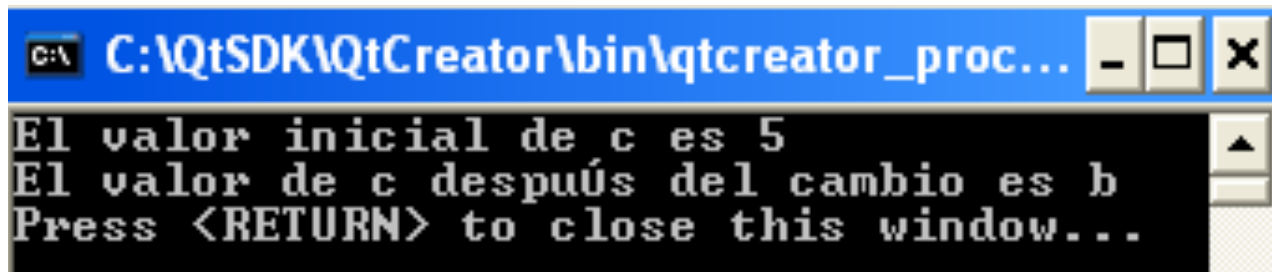
```
El valor del puntero es 0022FF68
El valor al que apunta inicialmente el puntero es 2.400000
El valor al que apunta después el puntero es 3.800000
Press <RETURN> to close this window...
```

Acceso a variables por punteros

```
#include <iostream>
using namespace std;
int main(){
    char c ;
    char *p;
    p = &c;
    *p = '5'
    cout<<"El valor inicial de c es "<< c << endl;
    *p = 'b';
    cout<<" El valor de c después del cambio es "<< c << endl;
    return 0;
}
```

Acceso a variables por punteros

```
#include <iostream>
using namespace std;
int main(){
    char c ;
    char *p;
    p = &c;
    *p = '5'
    cout<<"El valor inicial de c es "<< c << endl;
    *p = 'b';
    cout<<" El valor de c después del cambio es "<< c << endl;
    return 0;
}
```

A screenshot of a Qt Creator console window. The title bar shows the path 'C:\QtSDK\QtCreator\bin\qtcreator_proc...'. The console output displays the program's execution results: 'El valor inicial de c es 5', 'El valor de c después del cambio es b', and a prompt 'Press <RETURN> to close this window...'.

```
C:\QtSDK\QtCreator\bin\qtcreator_proc...
El valor inicial de c es 5
El valor de c después del cambio es b
Press <RETURN> to close this window...
```

Punteros y funciones

- Si se quiere modificar el valor de una variable dentro de una función esta debía ser global, por lo que se pierde el control del programa.
- Se puede usar **referencia**
- Ahora tenemos la opción de hacerlo pasando la dirección de la variable en lugar del **valor o referencia**.

Punteros y funciones

```
main()
{
    int x1 = 100, x2 = 200;
    cout<< x1 << x2;
    exchange(x1, x2);
    cout<< x1<< x2;
}
```

```
void exchange(int n1,int n2)
{
    int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
}
```

Se imprimirá:

Punteros y funciones

```
main()
{
    int x1 = 100, x2 = 200;
    cout<< x1 << x2;
    exchange(x1, x2);
    cout<< x1<< x2;
}
```

```
void exchange(int n1,int n2)
{
    int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
}
```

Se imprimirá: **100 200**
100 200

n1=200 x1=100
n2=100 x2=200

¿Es correcto el formato imprimido?

Punteros y funciones

1000	100	x1	cambiar(x1,x2);
1001	200	x2	void cambiar(int n1,int n2)
1002			{
1003	100 $\Rightarrow$ 200	n1	int temp;
1004	200 $\Rightarrow$ 100	n2	temp=n1;
1005	100	temp	n1=n2;
			n2=temp;
			}

Punteros y funciones

```
main()
{
    int x1 = 100, x2 = 200;
    cout<< x1<< " " << x2 <<endl;
    exchange(x1, x2);
    cout<< x1 << " " << x2;
}
```

```
void exchange(int *n1,int *n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Se imprimirá:



Punteros y funciones

```
main()
{
    int x1 = 100, x2 = 200;
    cout<< x1<< " " << x2 <<endl;
    exchange(x1, x2);
    cout<< x1 << " " << x2;
}
```

```
void exchange(int *n1,int *n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Se imprimirá: **100 200**
200 100

Punteros y funciones

1000	100 $\Rightarrow$ 200	x1	cambiar(&x1,&x2);
1001	200 $\Rightarrow$ 100	x2	void cambiar(int *n1,int *n2)
1002			{
1003	1000	n1	int temp;
1004	1001	n2	temp=*n1;
1005	100	temp	*n1=*n2;
			*n2=temp;
			}

Punteros y funciones

- Se crea en la función dos punteros
- Estos punteros, aunque no sean de main, apuntan a las variables de main
- Por tanto, si modificamos a lo que apuntan los punteros, estamos modificando las variables de main.

Punteros y funciones

```
#include <iostream>
using namespace std;
void FijaAOcho(float *pun);

int main(){
    float f = 3.4;
    float *p;
    p = &f;
    cout<<"El valor inicial de f es "<< f << endl;
    cout<<"El valor inicial de p es "<< p << endl;
    FijaAOcho(p);
    cout<<"El valor final de f es "<< f << endl;
    cout<<"El valor final de p es "<< p << endl;
    return 0;
}

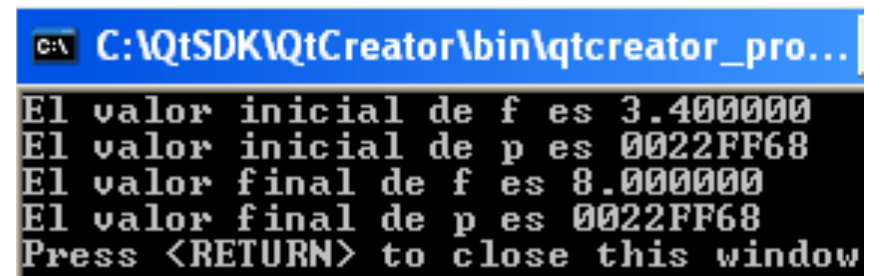
void FijaAOcho(float *pun){
    *pun = 8;
    pun = 0;
}
```

Punteros y funciones

```
#include <iostream>
using namespace std;
void FijaAOcho(float *pun);

int main(){
    float f = 3.4;
    float *p;
    p = &f;
    cout<<"El valor inicial de f es "<< f << endl;
    cout<<"El valor inicial de p es "<< p << endl;
    FijaAOcho(p);
    cout<<"El valor final de f es "<< f << endl;
    cout<<"El valor final de p es "<< p << endl;
    return 0;
}

void FijaAOcho(float *pun){
    *pun = 8;
    pun = 0;
}
```



```
C:\QtSDK\QtCreator\bin\qtcreator_pro...
El valor inicial de f es 3.400000
El valor inicial de p es 0022FF68
El valor final de f es 8.000000
El valor final de p es 0022FF68
Press <RETURN> to close this window
```

Punteros y arrays

- El nombre de un array es un puntero al primer elemento del array

```
char data[5], *p;
```

```
p=&data[0]; es equivalente a p=data;
```

Punteros y arrays

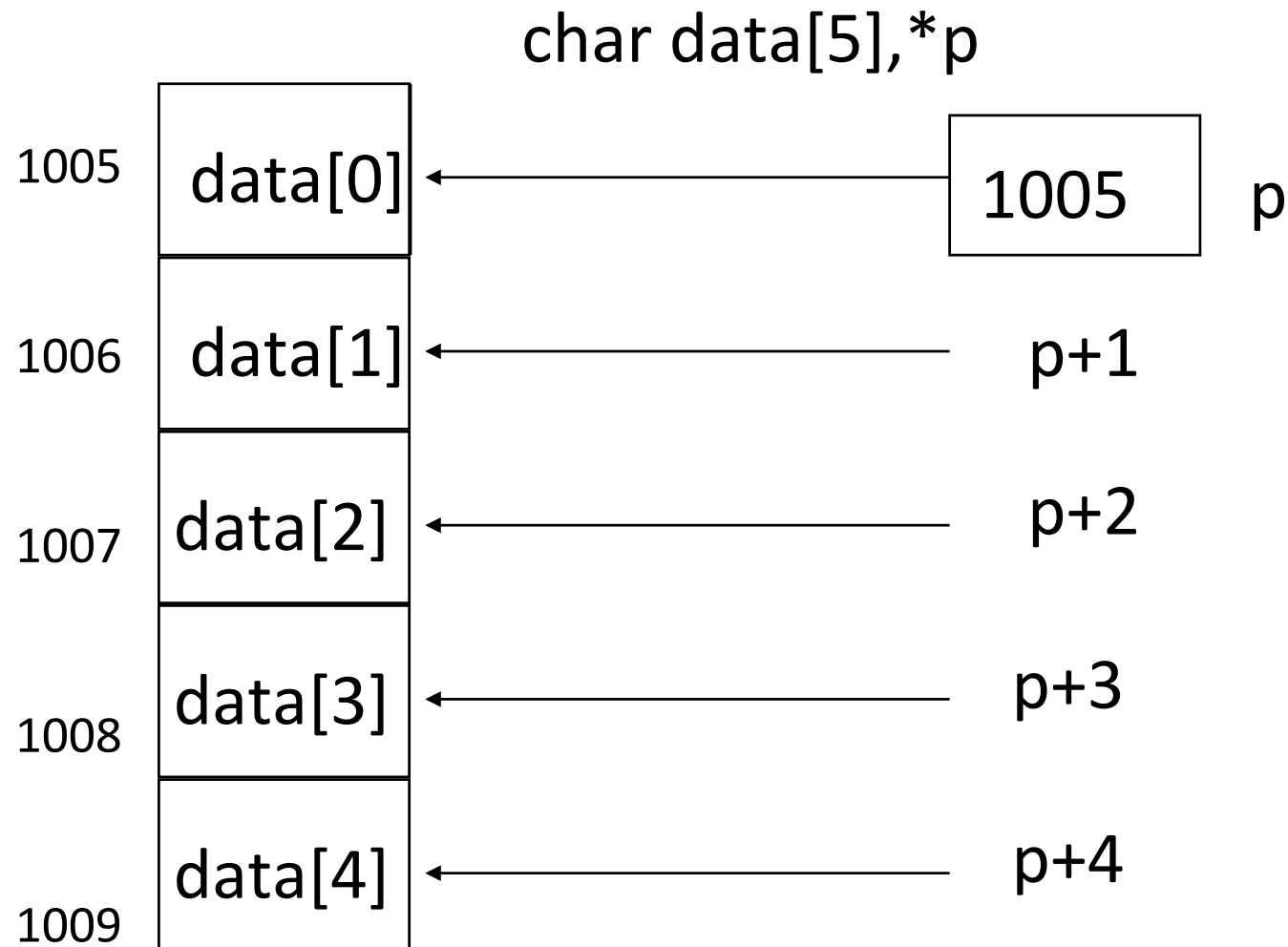
```
char data[5], *p;
```

`p=&data[0];` es *equivalente a* `p=data;`

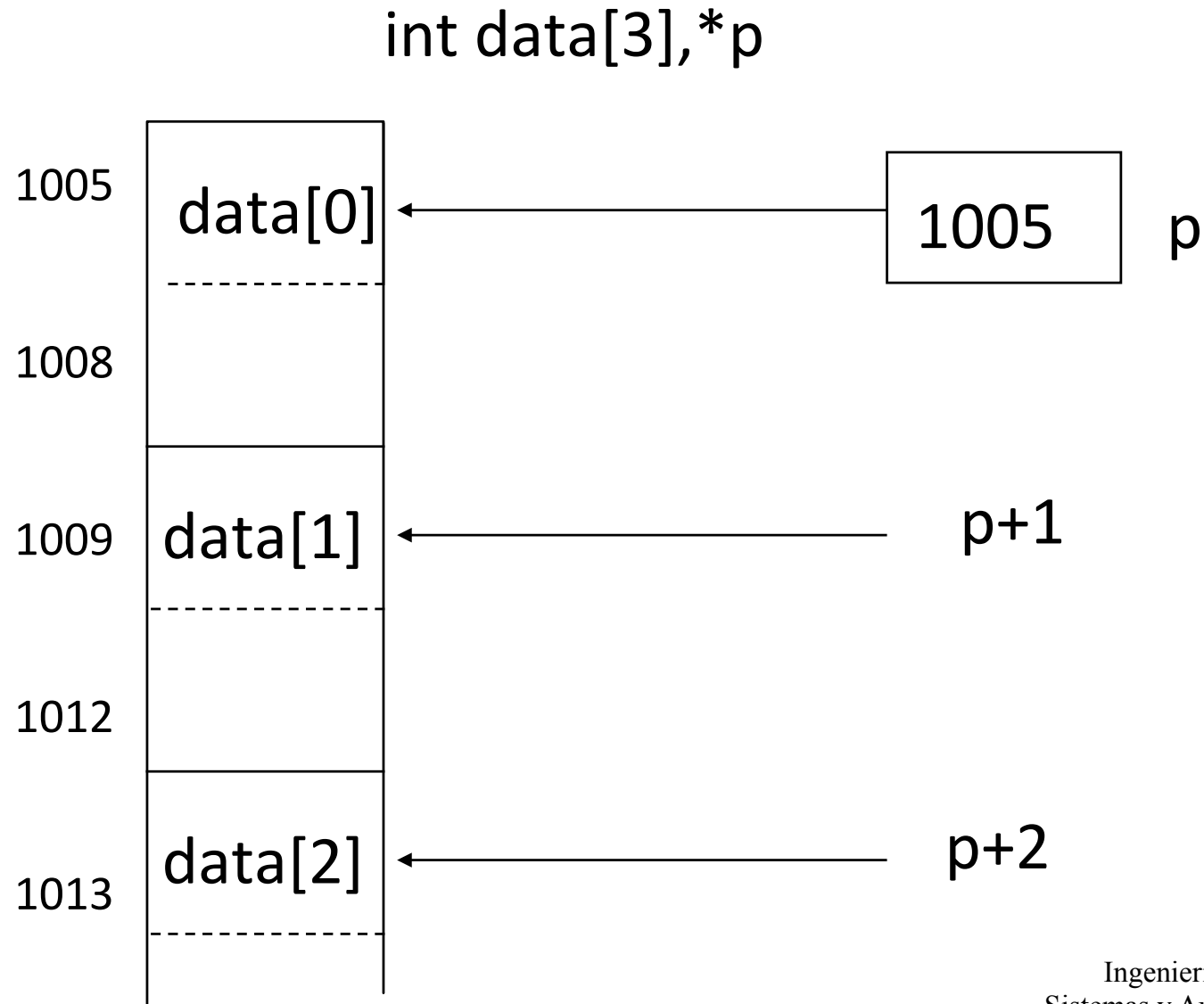


data
&data[0]

Punteros y arrays



Punteros y arrays



Punteros y arrays

```
int i,b[20],suma;
```

```
suma=0;
```

```
for(i=0;i<20;++i)
```

```
    suma=suma+b[i];
```

```
int i,b[20],suma,*p;
```

```
suma=0;
```

```
p=b;
```

```
for(i=0;i<20;++i)
```

```
    suma=suma+*(p+i);
```

OJO:

- $*(p+i) \neq *p+i$
- $b[i] \neq b[0]+i$

Arrays como parámetros en funciones

- Tres formas **equivalentes**
 - `funcion (int a[10])`
 - `funcion (int a[])`
 - `funcion (int *a)`
- Si un array se pasa como parámetro, siempre se podrá modificar dentro de la función
 - Se modifica el array, no la copia, luego los cambios persisten en la función que llama

Punteros y arrays bidimensionales

```
char dias[7][10]={“domingo”, “lunes”, “martes”, “miércoles”,
“jueves”, “viernes”, “sábado”}
```

1000	d	o	m	i	n	g	o	'\0'		
1010	l	u	n	e	s	'\0'				
1020	m	a	r	t	e	s	'\0'			
1030	m	i	e	r	c	o	l	e	s	'\0'
1040	j	u	e	v	e	s	'\0'			
1050	v	i	e	r	n	e	s	'\0'		
1060	s	a	b	a	d	o	'\0'			

● ¿Qué es dias[0]?

Punteros y arrays bidimensionales

- `dias[0]` es un puntero al primer carácter de la primera fila
- `cout << dias[0];` `domingo`

<code>dias[0]</code>	d	o	m	i	n	g	o	'\0'		
<code>dias[1]</code>	l	u	n	e	s	'\0'				
<code>dias[2]</code>	m	a	r	t	e	s	'\0'			
<code>dias[3]</code>	m	i	e	r	c	o	l	e	s	'\0'
<code>dias[4]</code>	j	u	e	v	e	s	'\0'			
<code>dias[5]</code>	v	i	e	r	n	e	s	'\0'		
<code>dias[6]</code>	s	a	b	a	d	o	'\0'			

Punteros y arrays bidimensionales

```
char dias[7][10]={"domingo","lunes","martes","miércoles",  
                 "jueves","viernes","sábado"};
```

```
char *DIAS[7]={"domingo","lunes","martes","miércoles",  
              "jueves","viernes","sábado"};
```

*DIAS[7] es un array de siete punteros a cadenas de caracteres

Arrays bidimensionales y funciones

```
#include <iostream>
using namespace std;
void asignavalor(int mat[][4], int k, int l);
int main(){
    int mat[3][4],i,j;
    asignavalor(mat,3,4);
    for(i=0;i<3;++i){
        for(j=0;j<4;++j)
            cout<<mat[i][j];
        cout<<"\n";
    }
    return 0;
}
```

```
void asignavalor(int mat[][4],int k, int l){
    int i,j;
    for(i=0;i<k;++i)
        for(j=0;j<l;++j)
            mat[i][j]=j+i*1;
}
```

Arrays bidimensionales y funciones

```
#include <iostream>
using namespace std;
void asignavalor(int *mat, int k, int l);
int main(){
    int mat[3][4],i,j;
    asignavalor(&mat[0][0],3,4);
    for(i=0;i<3;++i){
        for(j=0;j<4;++j)
            cout<<mat[i][j];
        cout<<"\n";
    }
    return 0;
}
```

```
void asignavalor(int *mat,int k, int l){
    int i,j;
    for(i=0;i<k;++i)
        for(j=0;j<l;++j)
            *(mat+i*l+j)=j+i*l;
}
```

Funciones que retornan punteros

- La declaración:

int * funcion(float a)

- Declara que la función tiene como parámetro un float y devuelve un puntero a un entero

Funciones que retornan punteros

```
#include <iostream>
using namespace std;
float *DevuelveDireccion(float vector[5]) ;

int main(){
    float v[5] ;
    cout<< "La dirección del primer elemento de v es "<< &v[0] << endl;
    cout<< "La dirección del primer elemento de v es " DevuelveDireccion(v) << endl;
    return 0;
}

float *DevuelveDireccion(float vector[5])
{
    float *pun;
    pun= &vector[1]
    return (pun);
}
```

Funciones que retornan punteros

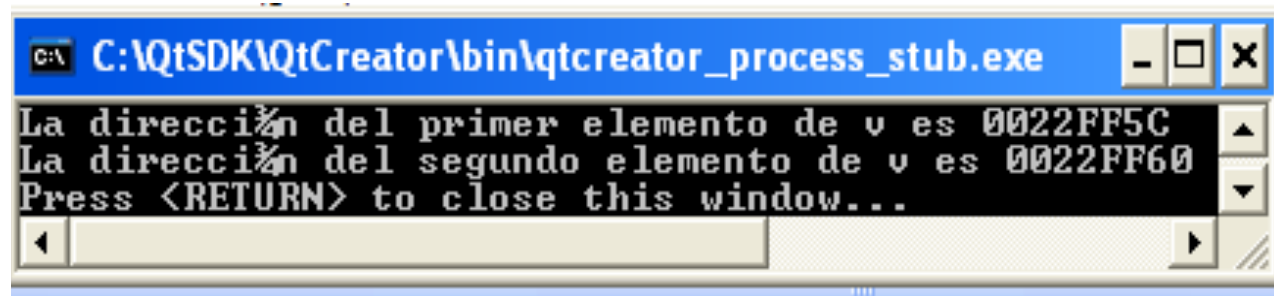
```
#include <iostream>

using namespace std;

float *DevuelveDireccion(float vector[5]) ;

int main(){
    float v[5] ;
    cout<< "La dirección del primer elemento de v es "<< &v[0] << endl;
    cout<< "La dirección del segundo elemento de v es " << DevuelveDireccion(v) << endl;
    return 0;
}

float *DevuelveDireccion(float vector[5])
{
    float *pun;
    pun= &vector[1]
    return (pun);
}
```



The screenshot shows a console window titled "C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe". The output text is as follows:

```
La dirección del primer elemento de v es 0022FF5C
La dirección del segundo elemento de v es 0022FF60
Press <RETURN> to close this window...
```

9. Estructuras

Estructuras

- Con los arrays se quería relacionar datos de igual tipo que tienen algo en común.
 - float notas[20]
- Las estructuras permiten generalizar mezclando datos de distintos tipos.

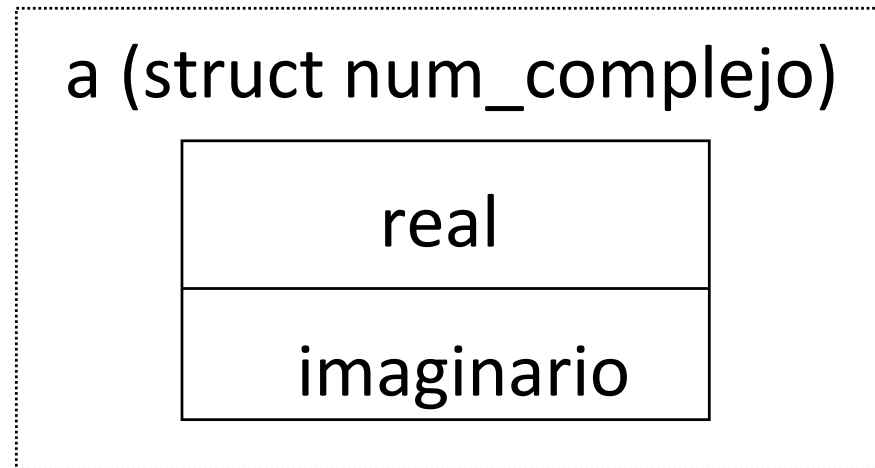
Estructuras

```
struct num_complejo  
{  
    float real;  
    float imaginario;  
};
```

```
num_complejo a,b,c;  
int i,j;
```

```
typedef struct  
{  
    float real;  
    float imaginario;  
} num_complejo;  
  
num_complejo a,b,c;  
int i,j;
```

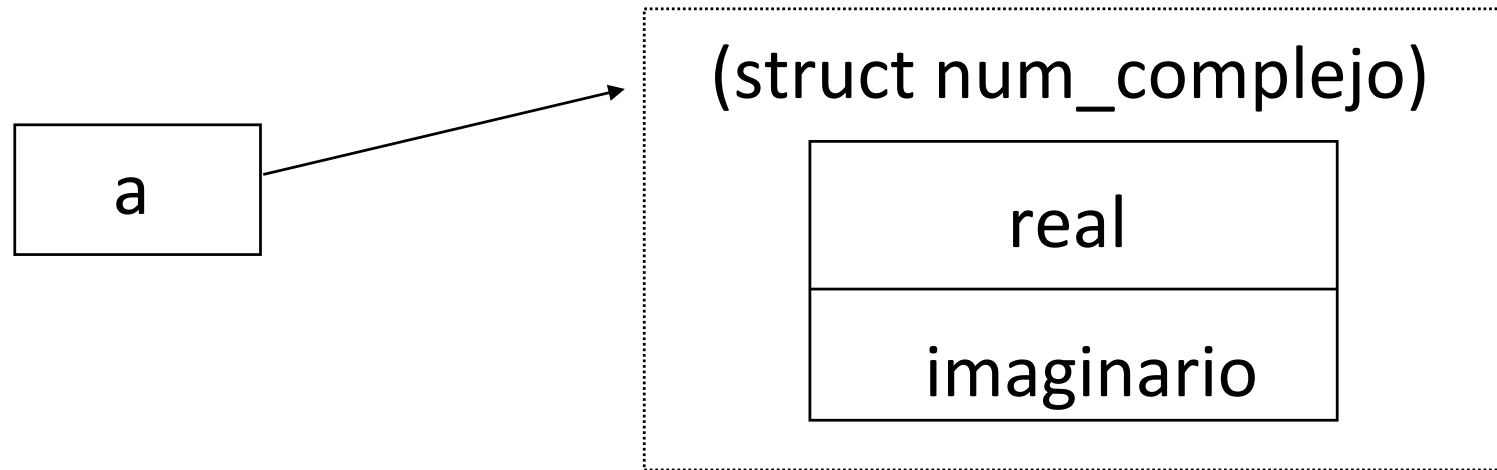
Acceso a los campos



variable_estructura.nombre_campo

`modulo=sqrt(a●real* a●real +a●imaginario* a●imaginario)`

Estructuras y punteros



puntero_a_estructura->nombre_campo

`modulo=sqrt(a->real* a->real +a->imaginario* a->imaginario)`

```

#include <iostream>
using namespace std;
typedef struct alumno
{
    char nombre[15];
    float nota;
};
int main(){
    alumno a ;
    a.nombre[0] = 'M' ;
    a.nombre[1] = 'i' ;
    a.nombre[2] = 'g' ;
    a.nombre[3] = 'u' ;
    a.nombre[4] = 'e' ;
    a.nombre[5] = 'l' ;
    a.nombre[6] = '\0' ;
    a.nota = 7.5 ;
    alumno *punt;
    punt=&a;
    cout<< "El nombre es "<< punt->nombre << ;
    cout<< " y la nota es " << punt->nota << endl;
    return 0;
}

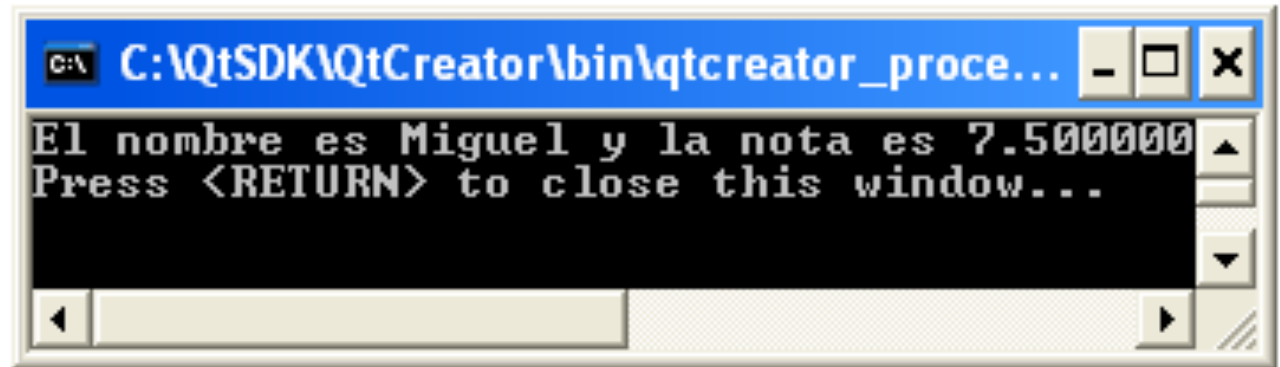
```



```

#include <iostream>
using namespace std;
typedef struct alumno
{
    char nombre[15];
    float nota;
};
int main(){
    alumno a ;
    a.nombre[0] = 'M' ;
    a.nombre[1] = 'i' ;
    a.nombre[2] = 'g' ;
    a.nombre[3] = 'u' ;
    a.nombre[4] = 'e' ;
    a.nombre[5] = 'l' ;
    a.nombre[6] = '\0' ;
    a.nota = 7.5 ;
    alumno *punt;
    punt=&a;
    cout<< "El nombre es "<< punt->nombre ;
    cout<< " y la nota es " << punt->nota << endl;
    return 0;
}

```



Estructuras y funciones

- Pueden pasarse y recibirse como variables “normales”

complejo suma (complejo a, complejo b)

```
{  
    complejo sum;  
    sum.real=a.real+b.real;  
    sum.imaginario=a.imaginario+b.imaginario;  
    return(sum)  
}
```

● Estructuras y funciones

- Pueden pasarse y recibirse como punteros

```
void suma (complejo a, complejo b, complejo *sum)
{
    sum->real=a.real+b.real;
    sum->imaginario=a.imaginario+b.imaginario;
}
```

Memoria dinámica

```
int * funcion (....)
{
    int * resultado = new int[5]; // Creación dinámica
    //hacer algo;
    return resultado;
}
```

- C++ almacena las variables locales en una **pila (STACK)**, pero el operador “new” reserva **memoria de almacenamiento dinámico (HEAP)**
- Cuando se crea un array de forma dinámica, su tamaño se hace en el tiempo de operación
- Cuando se crea un array, su tamaño debe de ser siempre conocido en tiempo de compilación (es decir, no es un valor variable)

Uso de las memorias

Heap

- No está limitada (salvo la memoria disponible del sistema)
- Gestión dinámica de la memoria
 - malloc/calloc/realloc/**new** => free/**delete**
 - OJO. **memory leak** => memoria no liberada no puede ser reasignada
- Menos rápida
- Accesible desde cualquier función en cualquier parte del código
- Variables de gran tamaño, dinámicas o que se van a utilizar en varias funciones

Stack

- Almacena variables temporales o locales
- Funcionamiento FILO
- Gestión por el sistema
- Limitada en tamaño
 - OJO. **stack overflow** (desbordamiento del stack) => CRASH!
- Muy rápida
- Cuando una función termina, todas sus variables son eliminadas.

Datos

- Variables globales
- Variables estáticas

Código

- Las instrucciones

Video

Memoria dinámica

```
delete pValue; //Puntero  
delete [] list; //Array
```

- Solo se puede utilizar el comando “delete” con el puntero que apunta a la memoria que ha sido creada mediante el comando “new”

```
int main()  
{  
    int * b = new int;  
    char* c = new char[12];  
  
    *b = 5;  
    delete b;  
    delete [] c;  
}
```

Memoria dinámica

```
int *p = new int; p→0013FF60  
*p = 45;  
p = new int; p→0013FF64
```

- Este procedimiento implica un problema de pérdida de memoria
- La posición original de memoria en la que se almacena el valor 45 (es decir, 0013FF60) no es accesible (debido a que se ha asignado nuevo espacio de memoria al mismo puntero p en 0013FF64) y la memoria original no puede ser borrada.

Argumentos de la línea de comandos

- C++ permite acceder desde el programa a los argumentos que se especifican en la línea de comandos.

mi_programa *fichero dato_1 dato_2*

Argumentos de la línea de comandos

- `main(int argc, char *argv[])`
 - `argc` es el número de comandos introducidos.
 - Ojo: el nombre del programa cuenta como un argumento
 - `*argv[]` es un puntero a un array de punteros a carácter
 - Cada puntero hace referencia a una cadena

Argumentos de la línea de comandos

```
main(int argc, char *argv[])
{
    int i;
    std::cout<<" numero de parámetros es "<< argc<<"\n";
    for (i=0;i<argc;++i)
        std::cout<<"argumento "<< i << "es "<< argv[i]<<"\n";
}
```

```
./mi_programa fichero.txt 5
```

```
numero de parámetros es 3
argumento 0 es mi_programa
argumento 1 es fichero.txt
argumento 2 es 5
```

El Pre-procesador

- Originalmente usado para simplificar el desarrollo de compiladores
- Actualmente parte del compilador
- Es posible escribir programas con él
- Los comandos empiezan por #

#include

- `#include<filename>`
Lee un fichero de texto “suministrado por el sistema” para utilizarlos en el fichero fuente
- Son del tipo `nombre.h`
- Se ponen definiciones de variables y prototipos
- `#include“filename”`

#define

- Definir variables

```
#define PI 3.1415
```

- Se sustituye el texto por el valor
- Ojo con las divisiones

```
#define RAD_TO_GRAD 180/3.14
```

```
#define RAD_TO_GRAD (180/3.14)
```

#define

```
#include <iostream>
#define PI 3.1416

main(int argc, char *argv[])
{
    float radio, area;
    std::cout<<" Introduzca el radio: ";
    std::cin>> radio;
    area = PI * radio * radio;
    std::cout<<"El area es "<< area <<"\n";

    return 0;
}
```

Output::

```
Introduzca el radio: 2
El area es 12.566400
```

#ifdef

- Se usa para incluir código de forma condicional

```
#ifdef SUNOS
```

```
/*Seccion de código específica de Sunos*/
```

```
#endif
```

- Depuraciones

Macros

- El pre-procesador reconoce.

```
#define SQR(a,b) a * a + b * b
```

$$\text{SQR}(a,3) \Rightarrow a * a + 3 * 3$$
$$\text{SQR}(a+1,3) \Rightarrow a+1 * a+1 + 3 * 3$$
$$(2a + 10 \text{ \textbf{NO} } a^2 + 2a + 10)$$

Macros

```
#include <iostream>
#define SUMA(a,b) a + b

main(int argc, char *argv[])
{
    int a = 2;
    std::cout<< SUMA(a,3)<< std::endl;
    std::cout<< SUMA(a+1,3) <<"\n";

    return 0;
}
```

Organización del código

- Prototipos de las funciones:

<ficheros>.h / <ficheros>.hpp

- Implementación de las funciones:

<ficheros>.c / <ficheros>.cpp

- Función main :

<fichero_principal>.c / <fichero_principal>.cpp

Organización del código

functions.h

```
int suma(int a, int b);
```

functions.cpp

```
#include "functions.h"
int suma(int a, int b)
{
    return a+b;
}
```

principal.cpp

```
#include <iostream>
#include "functions.h"

int main()
{
    int i=2, j=5;
    int s = suma(i,j);
    std::cout <<"la suma es "
              << s <<std::endl ;
}
```