# INFORMÁTICA INDUSTRIAL
# INDUSTRIAL COMPUTING

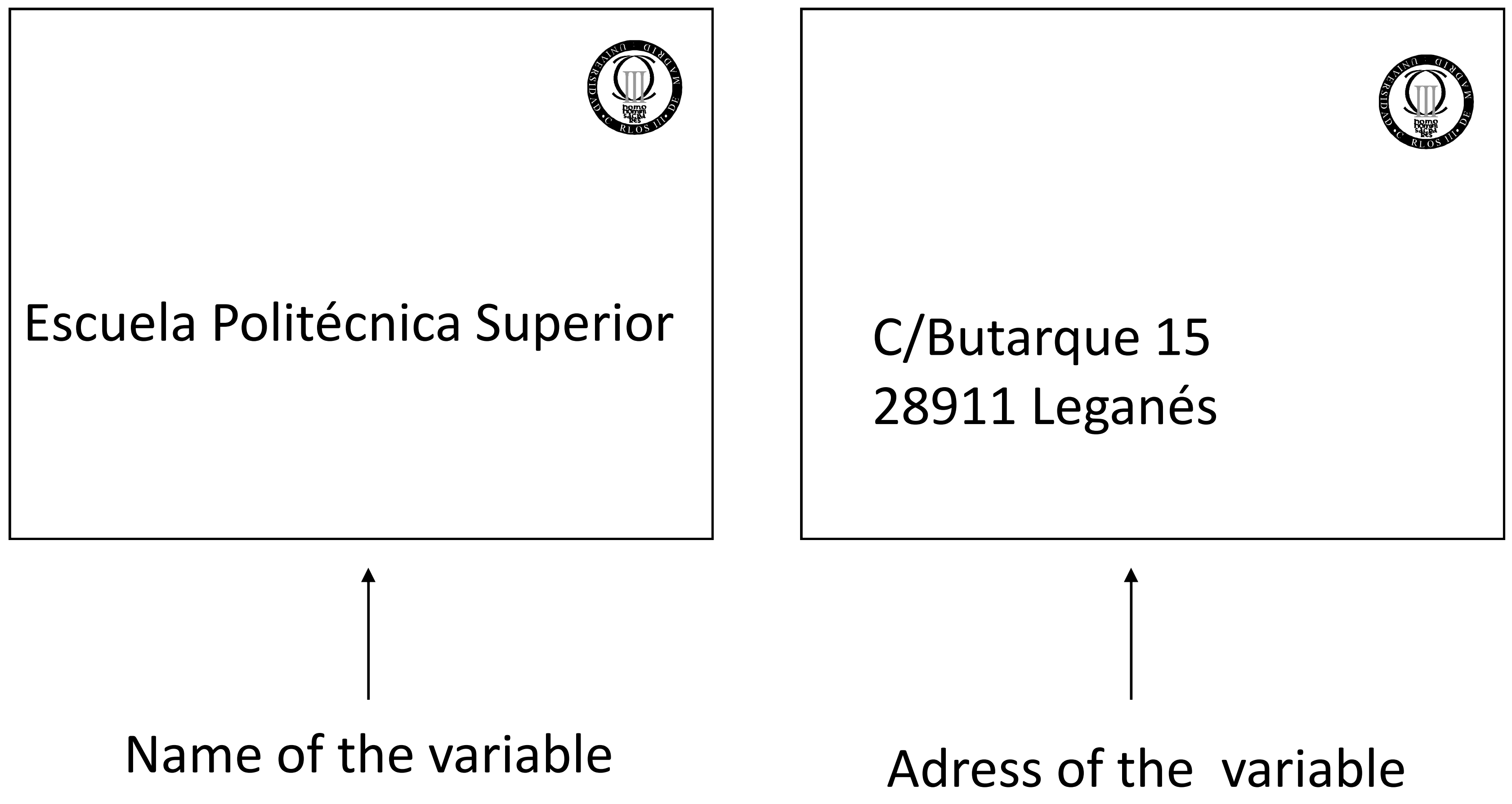
Third class

## BASICS OF
## PROGRAMMING WITH C++

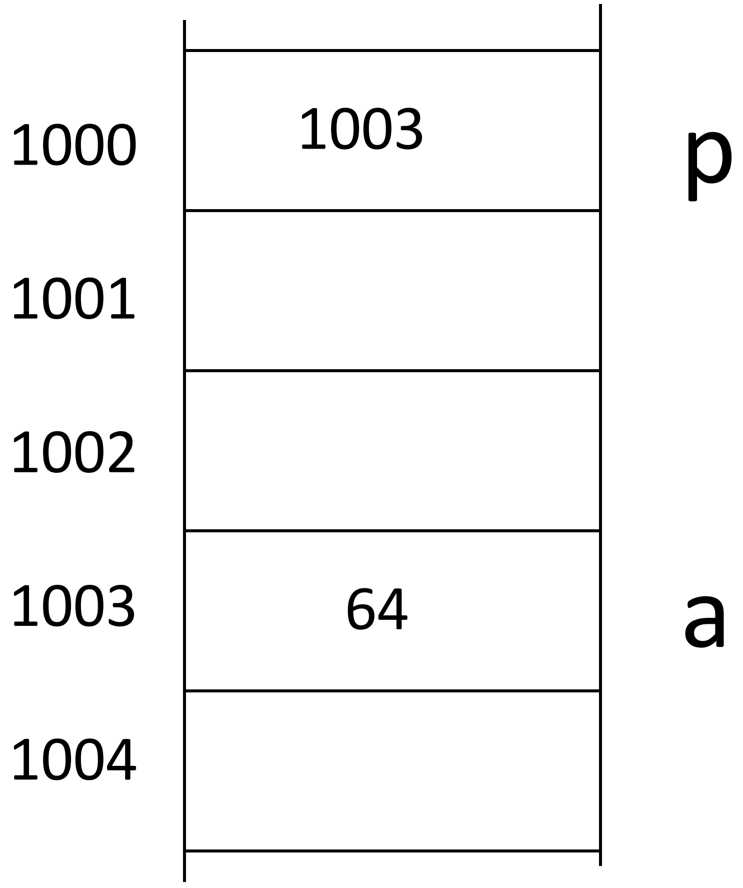
Profesor: Mohamed Abderrahim/Juan Gonzalez
Departamento de Ingeniería de Sistemas y Automática

# 8. Pointers

## Pointers


Escuela Politécnica Superior
C/Butarque 15
28911 Leganés
Name of the variable
Adress of the variable

## Pointers



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

- *a* is a variable with a value 64
- *p* is a variable that contains the value of the address of an other variable

## Pointers

- A pointer is a variable that points to another variable or function
- Declaration:

```
type * name_of_pointer;
Example:    int * y;
```

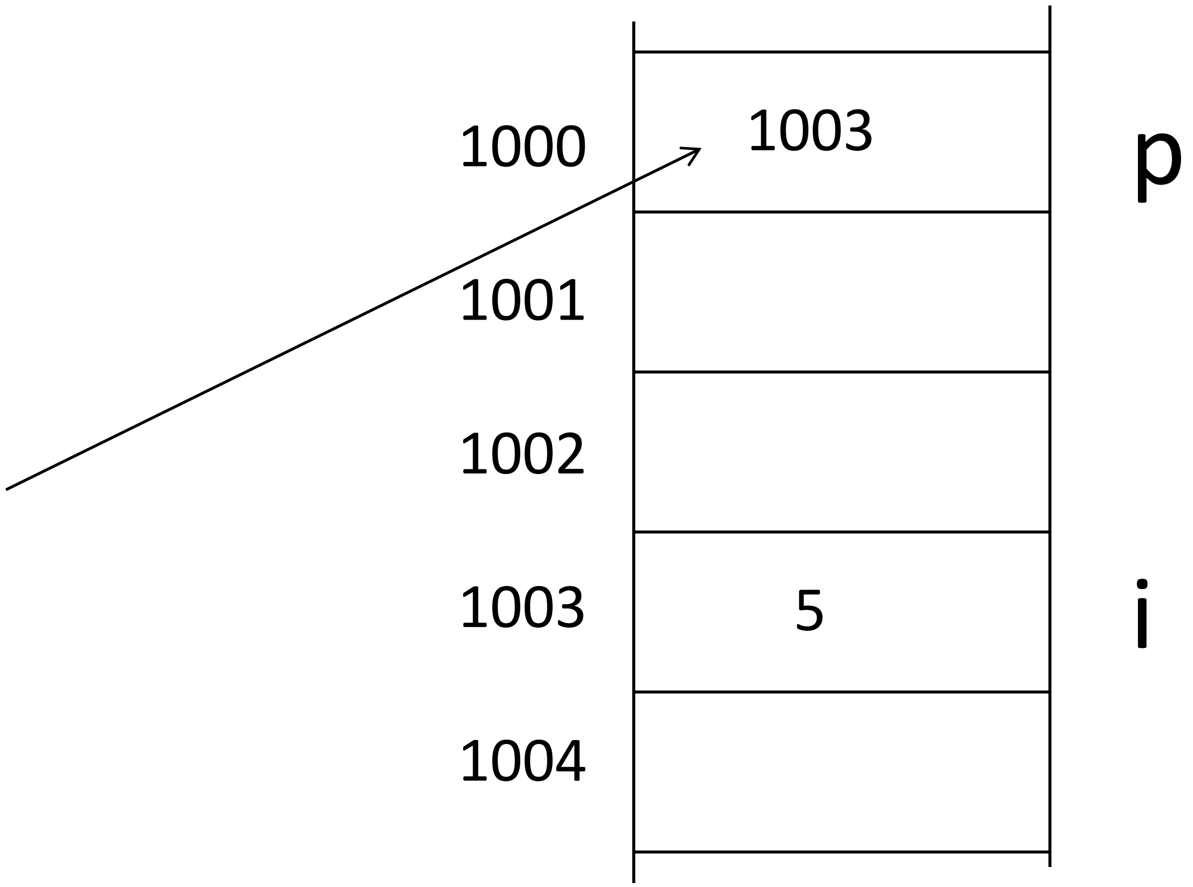
- y is a pointer to a variable of type integer.
- y is NOT an integer

## Pointers

- **&** is used to obtain the address of a variable
- * is used to obtain the value stored in a given memory address

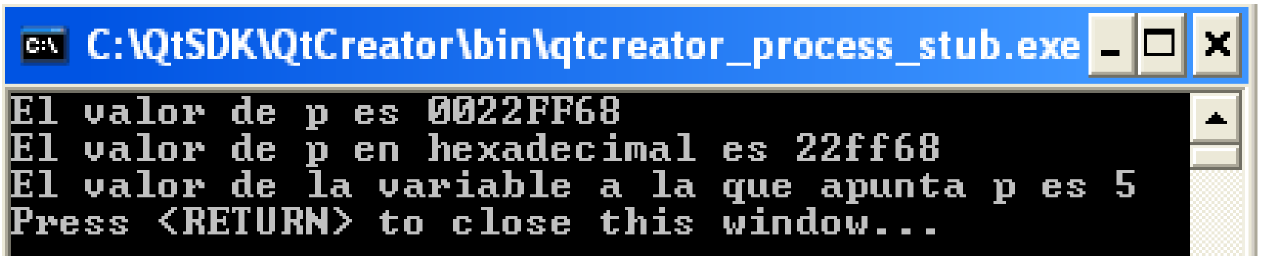
# Initialization of pointers

```
int i;
int *p;
i=5;
p=&i;
```



## Initialization of pointers

```
#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;
}
```

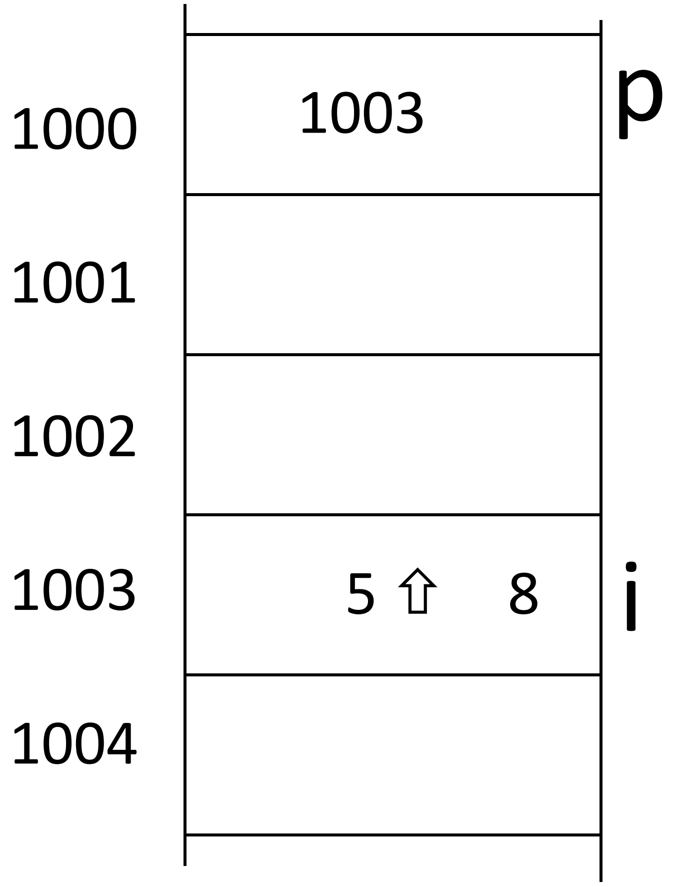

C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe

```
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...
```

# Access to a variables by pointers

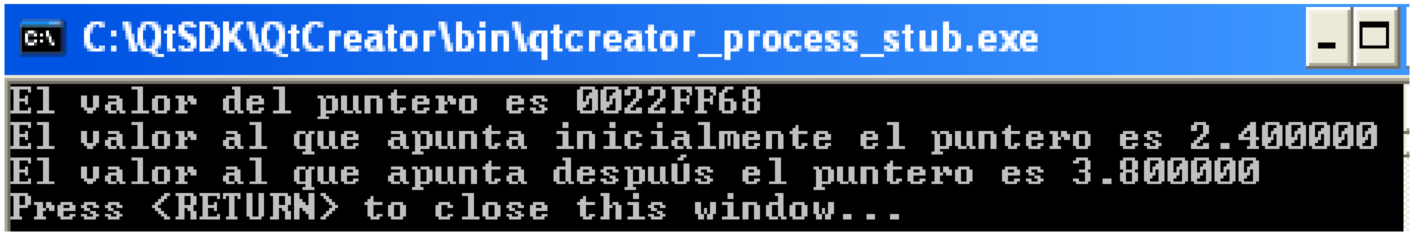
```
int i;
int *p;
i=5;
p=&i;


i=8;
```



# Access to a variables by pointers

```
#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;
}
```



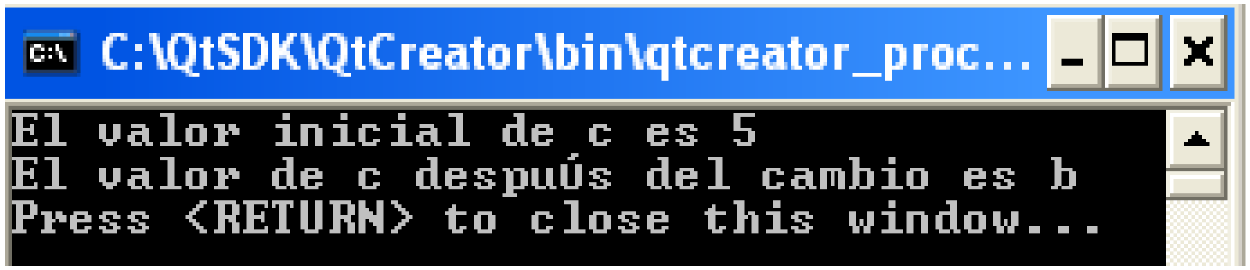
```
El valor del puntero es 0022FF68
El valor al que apunta inicialmente el puntero es 2.400000
El valor al que apunta despuús el puntero es 3.800000
Press <RETURN> to close this window...
```

## Access to a variables by pointers

```
#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;
}
```

## Access to a variables by pointers

```
#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;
}
```



```
El valor inicial de c es 5
El valor de c despuús del cambio es b
Press <RETURN> to close this window...
```

## Pointers and functions

- If we want to change the value of a variable inside a function, the variable should be global, which causes loss of control of the program.
- We can use reference
- The other option is passing the address of the variable instead of the value or reference

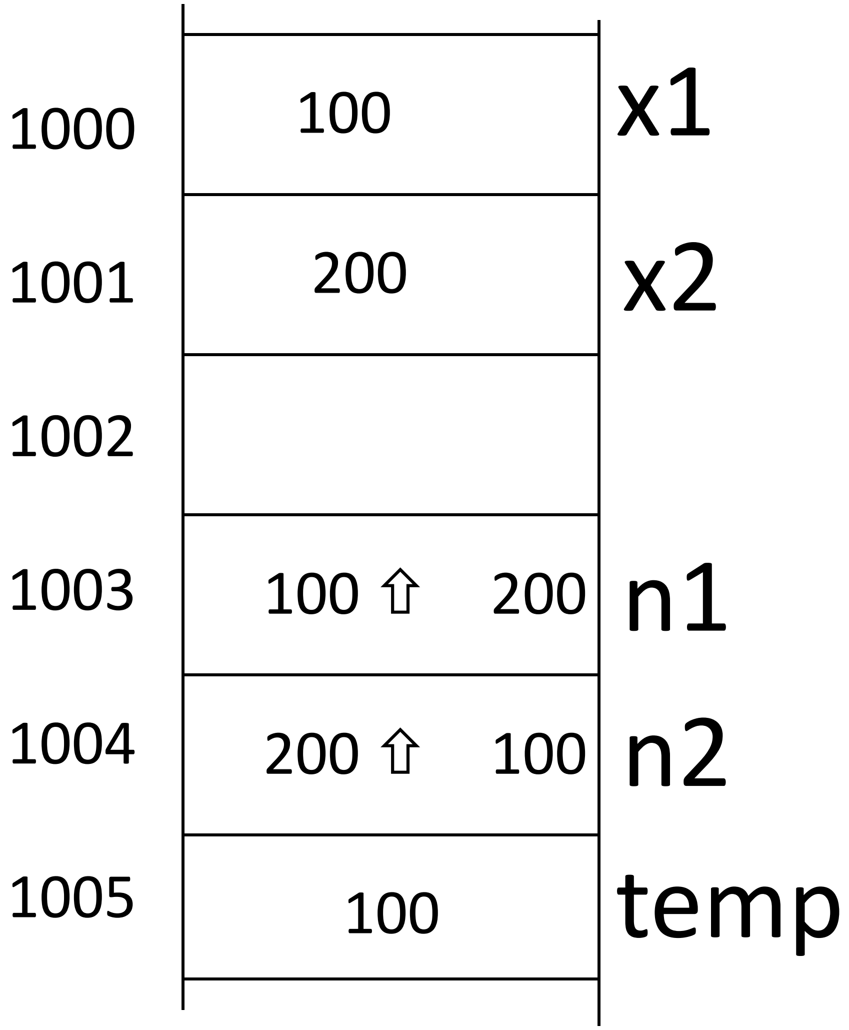
## Pointers and functions

```
main()
{
    int x1=100,x2=200;
    printf(“%d %d\n”,x1,x2);
    exchange(x1,x2);
    printf(“%d %d\n”,x1,x2);
}
```

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

It prints:

# Pointers and functions



```
exchange(x1,x2);

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

## Pointers and functions

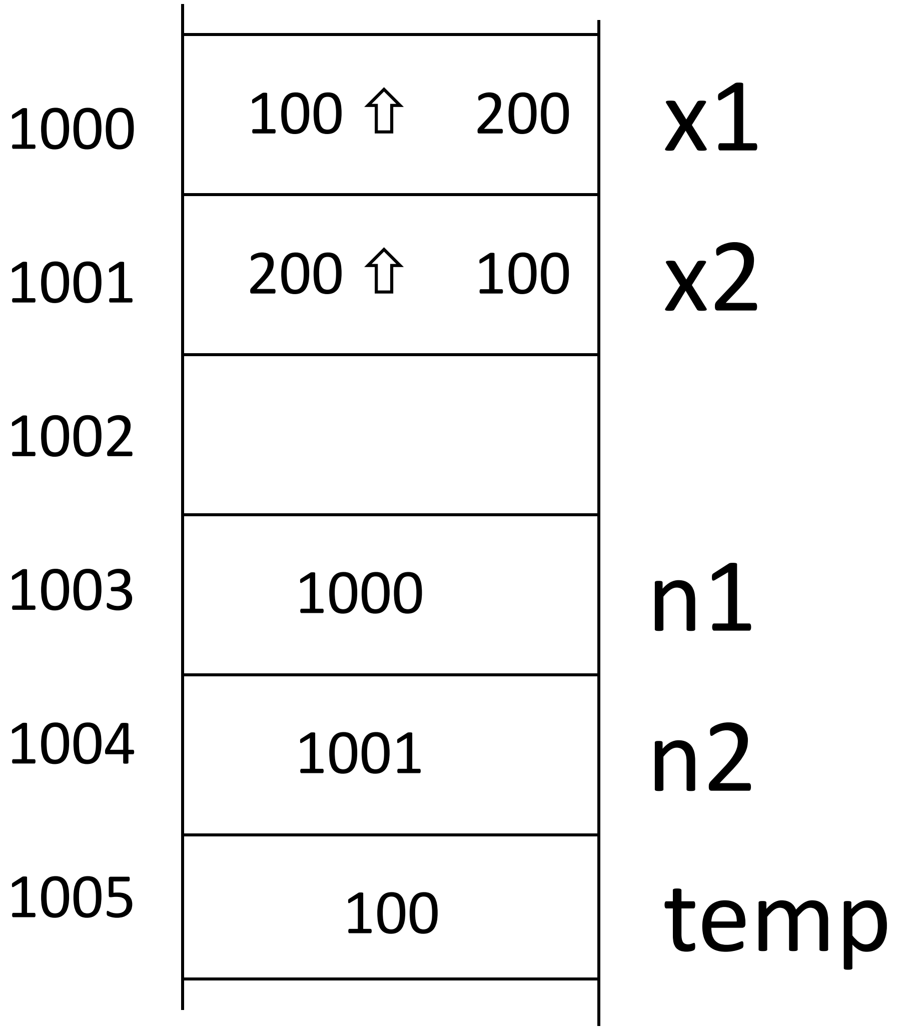
```
main()
{
    int x1=100,x2=200;
    printf("%d %d\n",x1,x2);
    exchange(&x1,&x2);
    printf("%d %d\n",x1,x2);
}
```

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

This prints:

```
100 200
200 100
```

## Pointers and functions



```
exchange(&x1,&x2);

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

## Pointers and functions

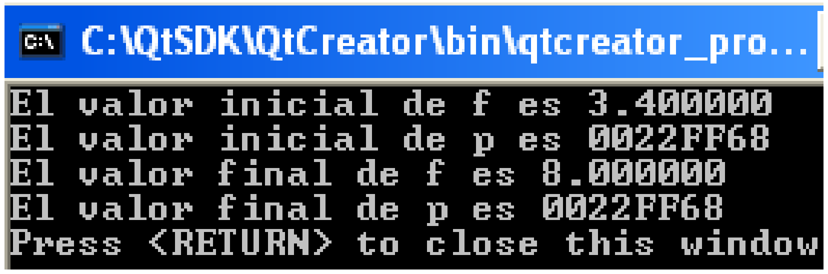
- Two pointers are created in the function
- Although they are not from main, they point to variables in main (when the call is made)
- Therefore, if we change what they point to, we would be modifying the variables of main

## Pointers and functions

```
#include <iostream>
using namespace std;
void FixToEight(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;
        FixToEight(p);
        cout<<"El valor final de f es "<< f << endl;
        cout<<"El valor final de p es "<< p << endl;
        return 0;
}

void FixToEight(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
```

## Pointers and arrays

- The name of an array is a pointer to the first element of the array

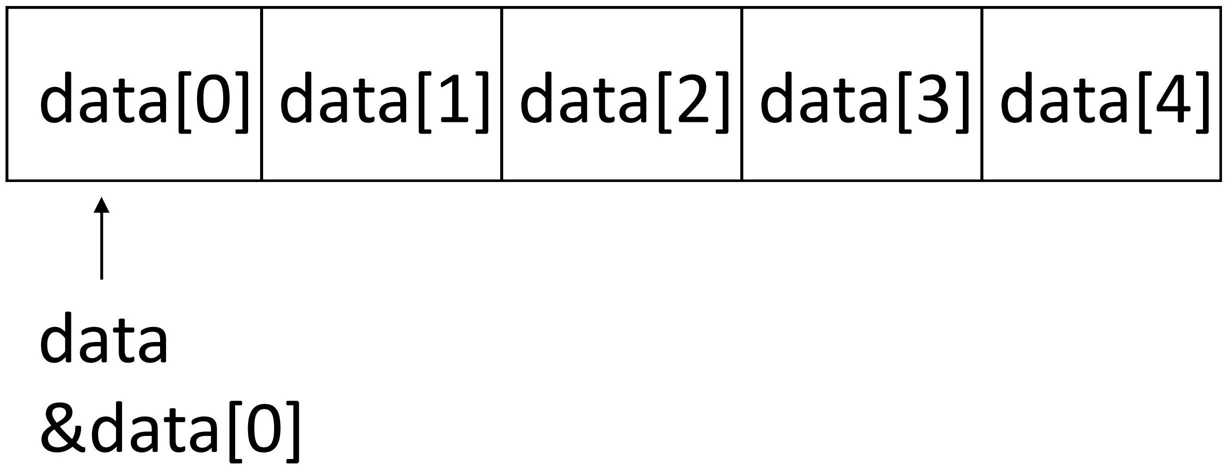
```
char data[5],*p;

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

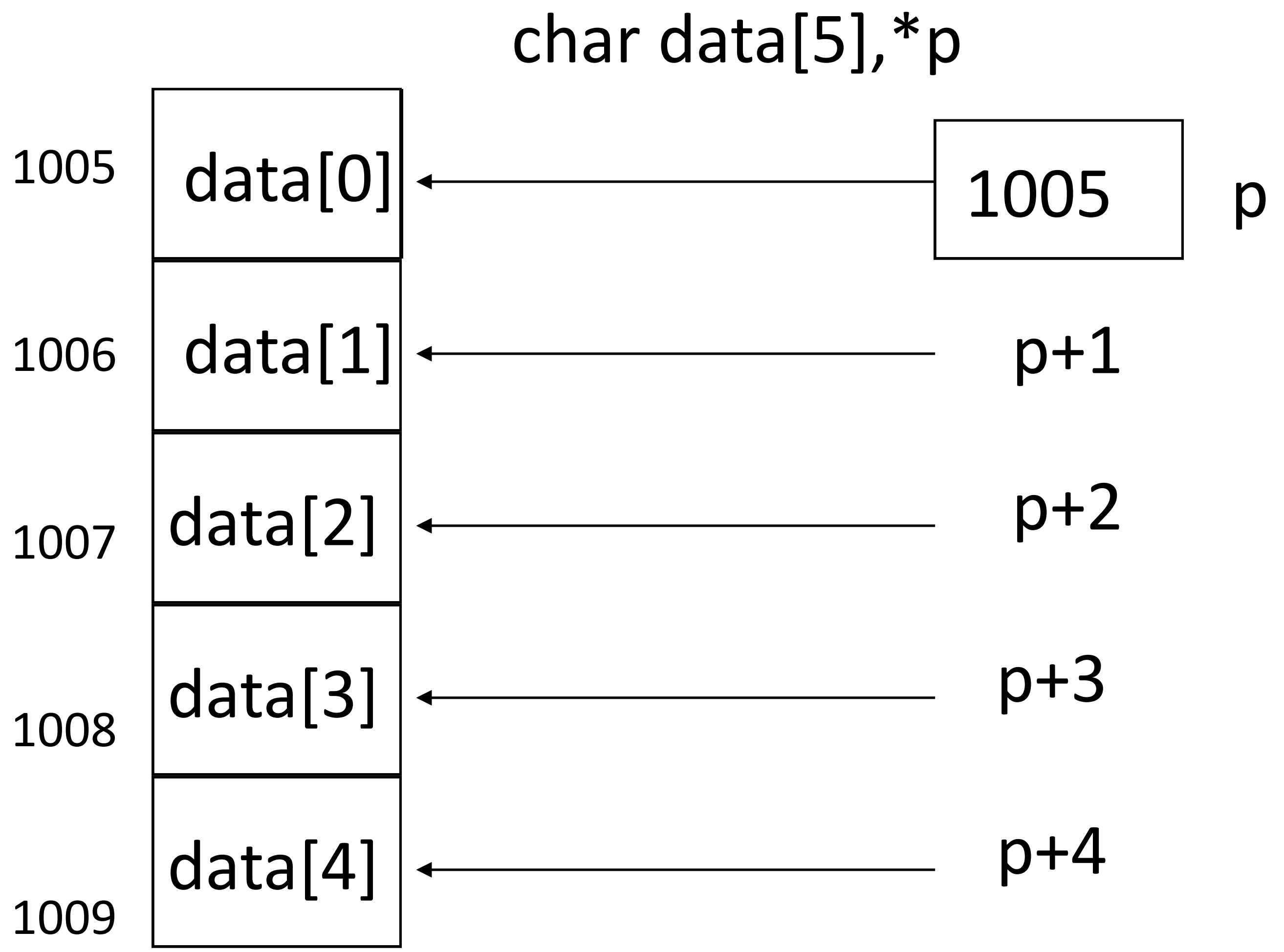
## Pointers and arrays

char data[5],*p;

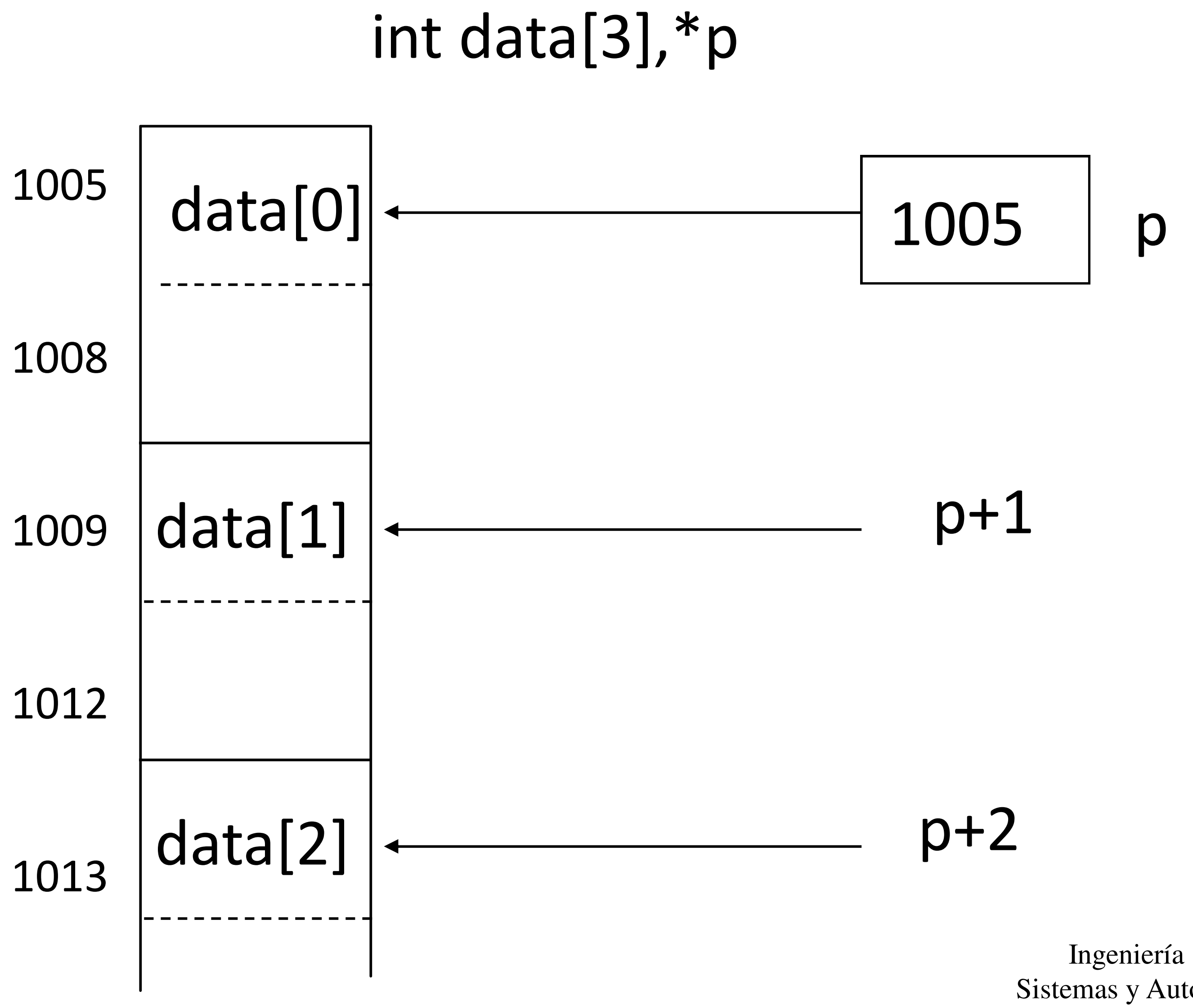
p=&data[0]; is *equivalent to* p=data;



# Pointers and arrays



# Pointers and arrays



# Pointers and 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);
```

**Be Careful:**

**OJO:**

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

## Arrays as function parameters

- Three equivalent forms
  - function (int a[10])
  - function (int a[])
  - function (int *a)
- If an array is passed as a parameter, it can always be changed from inside the function
  - The array is modified, not the copy, and then the changes persist in the calling function

## Pointers and  bi-dimensional arrays

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´ | | | |

**What is dias[0]?**

# Pointers and bi-dimensional arrays

- dias[0] is a pointer to the first character of the first line
- cout **<<** dias[0]; domingo

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| dias[0] | d | o | m | i | n | g | o | ´\0´ | | |
| dias[1] | l | u | n | e | s | ´\0´ | | | | |
| dias[2] | m | a | r | t | e | s | ´\0´ | | | |
| dias[3] | m | i | e | r | c | o | l | e | s | ´\0´ |
| dias[4] | j | u | e | v | e | s | ´\0´ | | | |
| dias[5] | v | i | e | r | n | e | s | ´\0´ | | |
| dias[6] | s | a | b | a | d | o | ´\0´ | | | |

## Pointers and  bi-dimensional arrays

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

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

*DIAS[7]  is an array of 7 pointers to strings

Mohamed Abderrahim,  Universidad Carlos III de Madrid

## bi-dimensional arrays y functions

```
#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*l;
}
```

## bi-dimensional arrays y functions

```
#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;
}
```

## Functions with pointer return

- The declaration:

  **int * function(float a)**

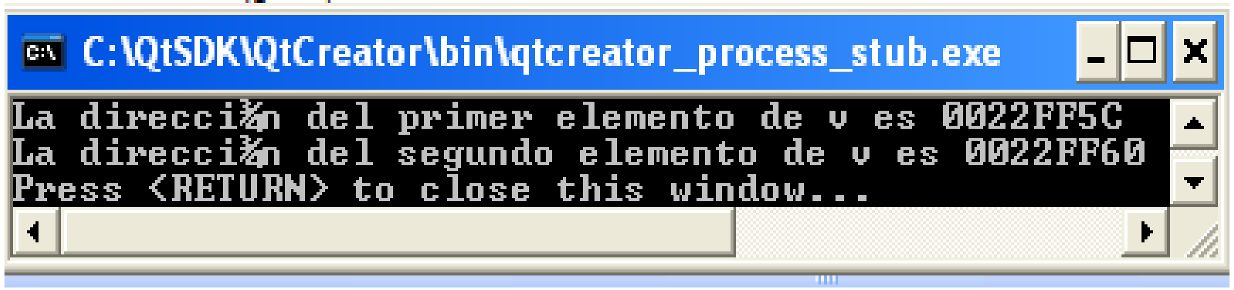
- Declares that the function has a float **a** parameter and returns a pointer

Mohamed Abderrahim, Universidad Carlos III de Madrid

# Functions with pointer return

```
#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);
}
```



```
La direcci¾n del primer elemento de v es 0022FF5C
La direcci¾n del segundo elemento de v es 0022FF60
Press <RETURN> to close this window...
```

# 9. Structures

## Structures

- With arrays we relate data of the same type that have something in common.
  - float marks[20]
- Structures allow to generalize and mix data from different types.

## Structures

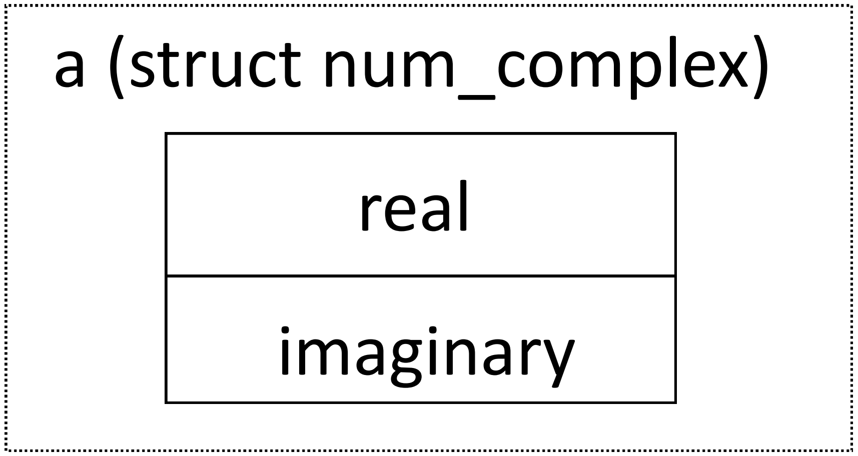
```
struct num_complex
{
   float real;
   float imaginary;
  };

struct num_complex a,b,c;
int i,j;
```

```
typedef struct
{
   float real;
   float imaginary;
  } num_complex;

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

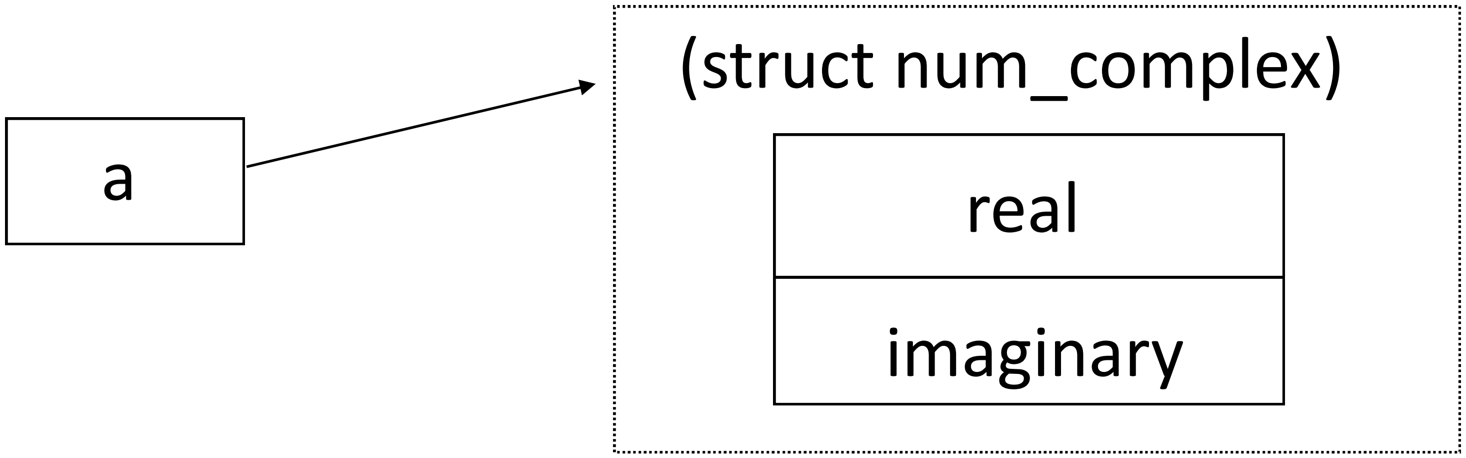
## Access to field



**variable_structure.name_field**

module=sqrt(a●real* a●real +a●imaginary* a●imaginary)

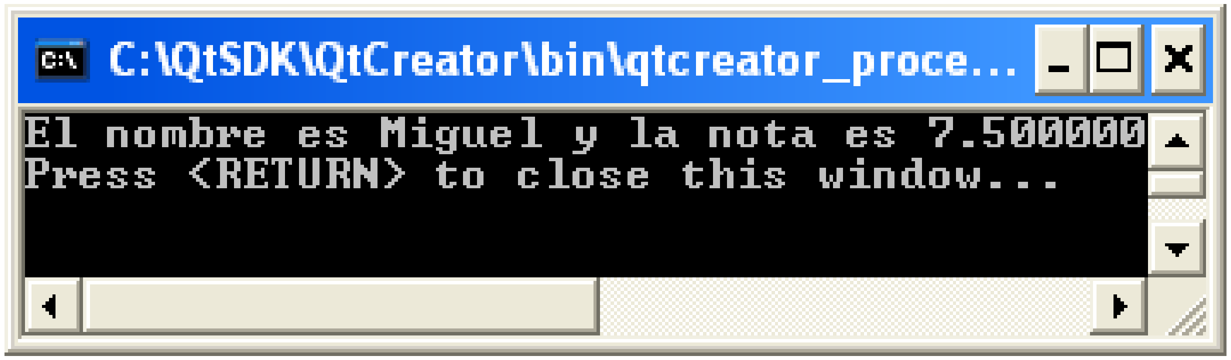
# Structures and Pointers



**pointer_ to_structure->name_field**

module=sqrt(a->real* a->real +a->imaginary* a->imaginary)

```
#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;
}
```



```
El nombre es Miguel y la nota es 7.500000
Press <RETURN> to close this window...
```

# Structures and Pointers

- A structure can be passed and returned like “normal” variables

```
complex suma (complex a, complex b)
{
    complex sum;
    sum.real=a.real+b.real;
    sum.imaginary=a.imaginary+b.imaginary;
    return(sum)
}
```

## Structures and Functions

- Structures can be passed and returned like pointers

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

## Dynamic Memory Allocation

```
int * reverse(.......)
{
    int *result = new int[5]; // Dynamic creation
        do something;
    return result;
 }
```

- C++ allocates local variables in the stack, but the new operator reserves memory in the heap.
- When you create a dynamic array, its size is determined at runtime.
- When you create a regular array, its size must be known at compile time (i.e., cannot be a variable).

# Dynamic Memory Allocation

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

- Use the delete keyword only with the pointer that points to the memory created by the new operator.

# Dynamic Memory Allocation

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

- This procedure incurs *memory leak* problem.
- The original memory space that holds value 45 (at 0013FF60) is not accessible (due to new memory space for the same pointer variable p at 0013FF64), the original memory cannot be deleted.

## Command line arguments

- C++ allows access through the program to command line arguments

**my_program** *fichero dato_1 dato_2*

## Command line arguments

- main(int argc, char *argv[])
  - argc is the number of introduced arguments.
    - Note: the name of the program is also counted as an argument
  - *argv[] is a pointer to an array of pointers to characters
    - Each pointer refer to a string

# Command line arguments

```
int main(int argc, char *argv[])
{
   int i;
   cout <<"Number of parameters is "  << argc << endl ;
   for (i=0;i<argc;++i)
        cout <<"argument " << i <<“is “ << argv[i] << endl;
    return 0;
}
```

```
./my_program fichero.txt 5

Number of parameters is 3
argument 0 is my_program
argument 1 is fichero.txt
argument 2 is 5
```

## The Pre-processor

- Originally used to simplify the development of compilers
- Currently it is part of the compiler
- It is possible to write programs with it
- The commands start with #

## #include

- #include<filename>
  Read a text file supplied by the system to use it in the source file
- They are of type name.h
- Include definitions of variables and prototypes
- #include“filename”

## #define

- Define variables

```
#define PI 3.1415
```

- Text is substituted by the value
- Be careful with divisions

```
#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

- Used to include part othe code in a conditional manner

```
#ifdef SUNOS
```

-
  ```
  /*   Section of code specific to Sunos   */
  #endif
  ```

- Debugging

## Macros

- The pre-processor recognize

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

SQR(a,3) $\Rightarrow$ a * a + 3 * 3

SQR(a+1,3) $\Rightarrow$ a+1 * a+1 + 3 * 3

(2a + 10 **NO** $a^2 + 2a + 10$)

## Organization of the code

- Function Prototypes:

  *<files>.h / <files>.hpp*

- Implementation of the functions:

  *<files>.c / <files>.cpp*

- Function main :

  *<man_file>.c/<main_file>.cpp*

## Code Organization

**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 ;

}
```