

INFORMÁTICA INDUSTRIAL

Segunda clase de:

PROGRAMACIÓN BÁSICA

C++

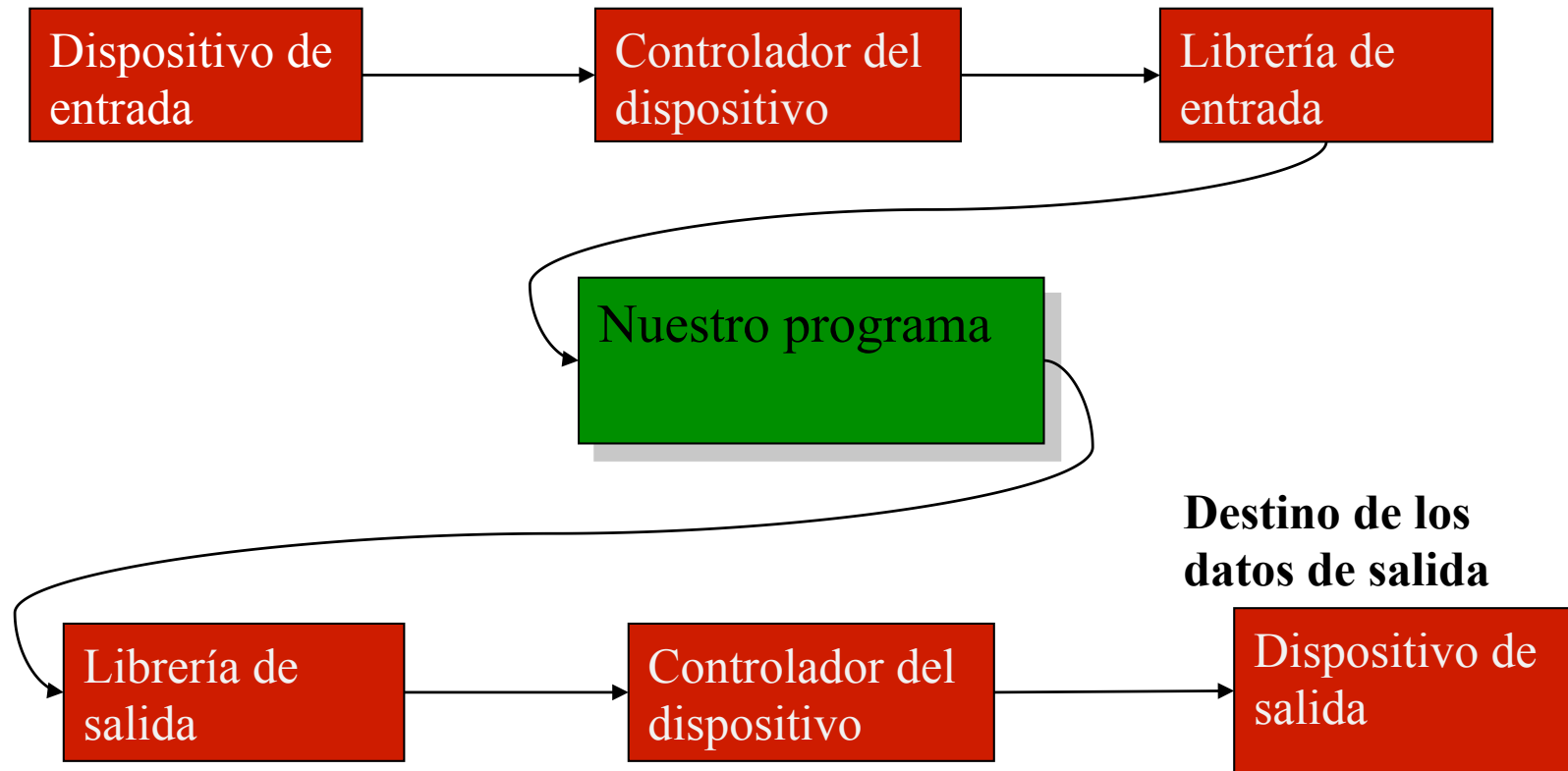
Profesores: F. Abu-Dakka / A. Castro / M. Abderrahim
Departamento de Ingeniería de Sistemas y Automática



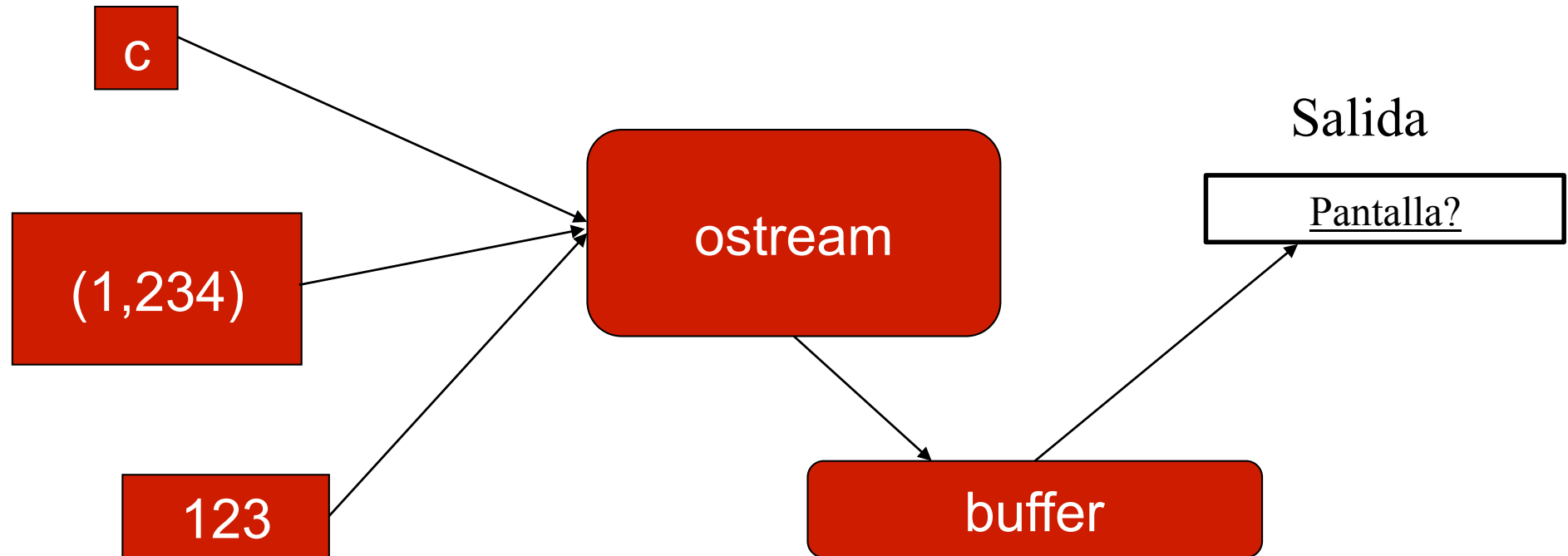
4. Entrada y salida estándar

Entrada y Salida

Fuente de datos:

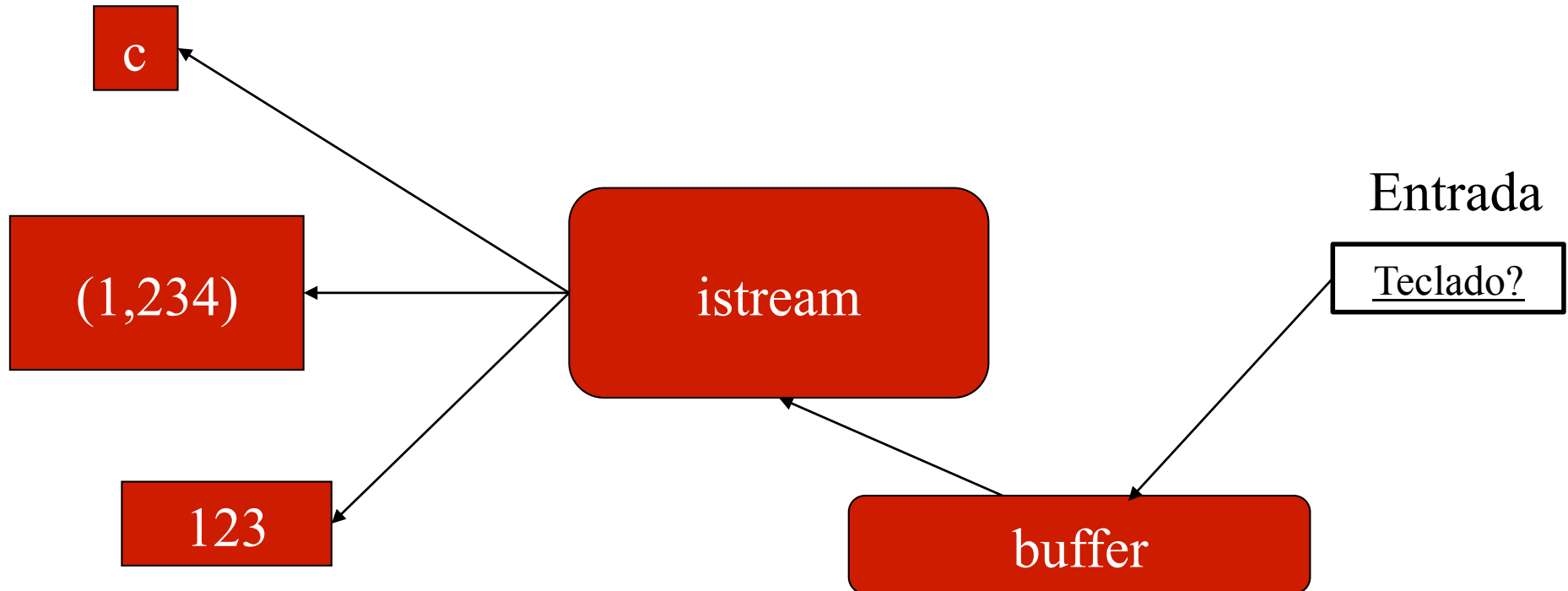


El Modelo de Flujo



- Un “**ostream**” (flujo de salida)
 - Modifica valores de varios tipos en secuencias de caracteres
 - Envía dichos caracteres a una salida
 - *Ej.:* consola, fichero, memoria principal, otra computadora

El Modelo de Flujo



- Un “**istream**” (flujo de entrada)
 - Convierte secuencias de caracteres en valores de varios tipos.
 - Obtiene dichos caracteres desde una entrada
 - *Ej.:* consola, fichero, memoria principal, otra computadora

El Modelo de Flujo

- Lectura y Escritura

- De entidades escritas (mecanografiadas)
 - << (salida) y >> (entrada) más otros operadores
 - Mantiene el tipo del dato
 - Formateado
- Normalmente almacenado (introducido, mostrado en pantalla, etc.) como texto:
 - Pero no necesariamente (se verá en la clase relativa a Ficheros)
- Extensible
 - El programador puede definir sus propias operaciones de E/S a partir de tipos creados por él mismo.
 - Un flujo puede ser utilizado por cualquier dispositivo de E/S o de almacenamiento

Errores en el manejo de las operaciones de E/S

- Fuentes de error
 - Errores humanos
 - Archivos que no cumplen especificaciones
 - Especificaciones no realistas
 - Errores de programación
 - Etc.
- `iostream` reduce todos los errores posibles a cuatro estados:
 - **`good()`** *// la operación es correcta*
 - **`eof()`** *// el programa llega al final del fichero (“end of file”)*
 - **`fail()`** *// algo inesperado ha sucedido en el flujo del programa*
 - **`bad()`** *// algo inesperado y **serio** ha sucedido en el flujo del programa*

Observaciones

- Como programadores, se prefiere SIEMPRE regularidad y simplicidad
 - Pero nuestro trabajo es cumplir con las especificaciones de nuestros usuarios
- Las personas son muy quisquillosas/particulares/exigentes con respecto a la forma de observar/analizar/requerir los resultados obtenidos:
 - A menudo tienen buenas razones para hacerlo así...
 - Reglas tradicionales/convencionales:
 - “¿Qué significa 123,456?”
 - “¿Qué significa (123)?”
 - El mundo (y en nuestro caso, de los formatos de salida) es normalmente dictado por las necesidades.

Salida de la Base Numérica

- Se puede modificar la “base” numérica
 - Base 10 == decimal; dígitos: 0 1 2 3 4 5 6 7 8 9
 - Base 8 == octal; dígitos: 0 1 2 3 4 5 6 7
 - Base 16 == hexadecimal; dígitos: 0 1 2 3 4 5 6 7 8 9 a b c d e f

// ejemplo sencillo:

```
cout << dec << 1234 << "\t(decimal)\n"  
      << hex << 1234 << "\t(hexadecimal)\n"  
      << oct << 1234 << "\t(octal)\n";
```

// El carácter '\t' es “tab” o tabulación (“tabulación carácter”)

// resultados:

1234	(decimal)
4d2	(hexadecimal)
2322	(octal)

Manipuladores

- Se puede cambiar la “base”
 - Base 10 == decimal; dígitos: 0 1 2 3 4 5 6 7 8 9
 - Base 8 == octal; dígitos : 0 1 2 3 4 5 6 7
 - Base 16 == hexadecimal; dígitos : 0 1 2 3 4 5 6 7 8 9 a b c d e f

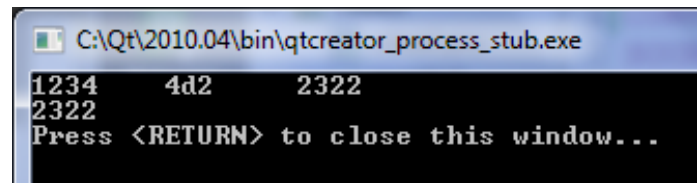
// ejemplo simple:

```
cout << 1234 << '\t'  
      << hex << 1234 << '\t'  
      << oct << 1234 << '\n';
```

```
cout << 1234 << '\n';           // la base octal sigue vigente
```

// resultados:

```
1234    4d2    2322  
2322
```



```
C:\Qt\2010.04\bin\qtcreator_process_stub.exe  
1234    4d2    2322  
2322  
Press <RETURN> to close this window...
```

Otros manipuladores

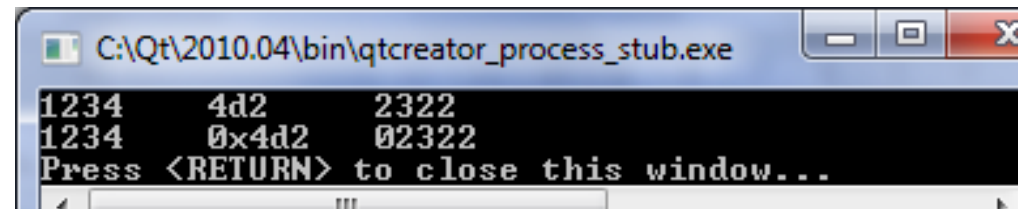
- Se puede cambiar la “base”
 - Base 10 == decimal; dígitos: 0 1 2 3 4 5 6 7 8 9
 - Base 8 == octal; dígitos: 0 1 2 3 4 5 6 7
 - Base 16 == hexadecimal; dígitos: 0 1 2 3 4 5 6 7 8 9 a b c d e f

// ejemplo simple:

```
cout << 1234 << '\t'  
      << hex << 1234 << '\t'  
      << oct << 1234 << endl; // '\n'  
cout << showbase << dec; // muestra las bases  
cout << 1234 << '\t'  
      << hex << 1234 << '\t'  
      << oct << 1234 << '\n';
```

// resultados:

1234	4d2	2322
1234	0x4d2	02322



Manipuladores en punto flotante

- Se puede cambiar el formato de salida de punto flotante
 - **general** – **iostream** elige el mejor formato utilizando **n** digits (es el funcionamiento por defecto)
 - **scientific** – un dígito antes del punto decimal más el exponente; **n** dígitos después.
 - **fixed** – sin exponente; **n** dígitos después del punto decimal.

// ejemplo simple:

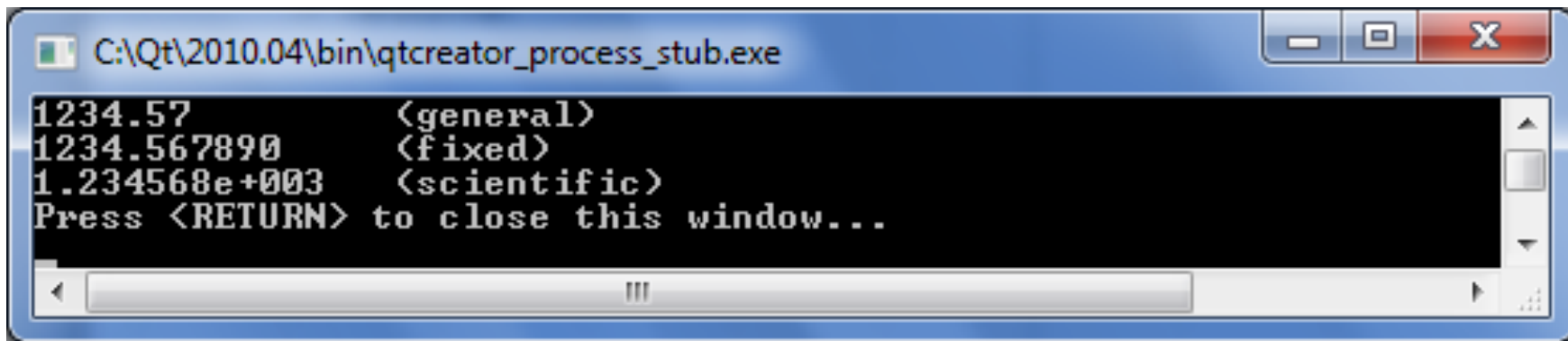
```
cout << 1234.56789 << "\t\t/general)\n"           /\t\t para alinear columnas
      << fixed << 1234.56789 << "\t\t(fixed)\n"
      << scientific << 1234.56789 << "\t\t(scientific)\n";
```

Floating-point Manipulators

// ejemplo simple:

```
cout << 1234.56789 << "\t\t/general\n"           /\t\t para alinear columnas
      << fixed << 1234.56789 << "\t\t/fixed\n"
      << scientific << 1234.56789 << "\t\t/scientific\n";
```

// resultados:



```
C:\Qt\2010.04\bin\qtcreator_process_stub.exe
1234.57      <general>
1234.567890  <fixed>
1.234568e+003 <scientific>
Press <RETURN> to close this window...
```

Amplitud ("Width") del campo de salida

- La anchura del campo de salida es el número de caracteres que pueden ser utilizados por la siguiente operación de salida.
 - Atención: solo funciona para la siguiente salida (no tiene en cuenta la precisión, la base, o el formato de punto flotante)
 - Atención: la salida nunca se trunca para que quepa en el campo
 - (mejor un mal formato que un mal valor)

// ejemplo simple:

```
cout << 123456 << '|' << setw(4) << 123456 << '|'
      << setw(8) << 123456 << '|' << 123456 << "\\n";
cout << 1234.56 << '|' << setw(4) << 1234.56 << '|'
      << setw(8) << 1234.56 << '|' << 1234.56 << "\\n";
cout << "asdfgh" << '|' << setw(4) << "asdfgh" << '|'
      << setw(8) << "asdfgh" << '|' << "asdfgh" << "\\n";
```

// resultados:

```
123456|123456| 123456|123456|
1234.56|1234.56| 1234.56|1234.56|
asdfgh|asdfgh| asdfgh|asdfgh|
```

Observación

- Por este tipo de detalles es por lo que se necesitan libros de texto, manuales, referencias, soporte online, etc., porque:
 - Siempre se suelen olvidar los detalles cuando se necesitan.

5. Cadenas y Arrays (“Arreglos”)

Arrays (“Arreglos”)

- Un conjunto de variables del mismo tipo que se refieren de forma común:

int alumno1, alumno2, alumno3;



int alumnos[3];

- declaracion:

Tipo nombre_de_array [tamaño];

Arrays (“Arreglos”)

- Forma de uso:
 - A menudo, los elementos se utilizan por separado
- `int alumnos[3]`
 - `alumnos[0]` 1º elemento del array
 - `alumnos[1]` 2º elemento del array
 - `alumnos[2]` 3º elemento del array
- El elemento `alumnos[3]`
 - No tiene sentido

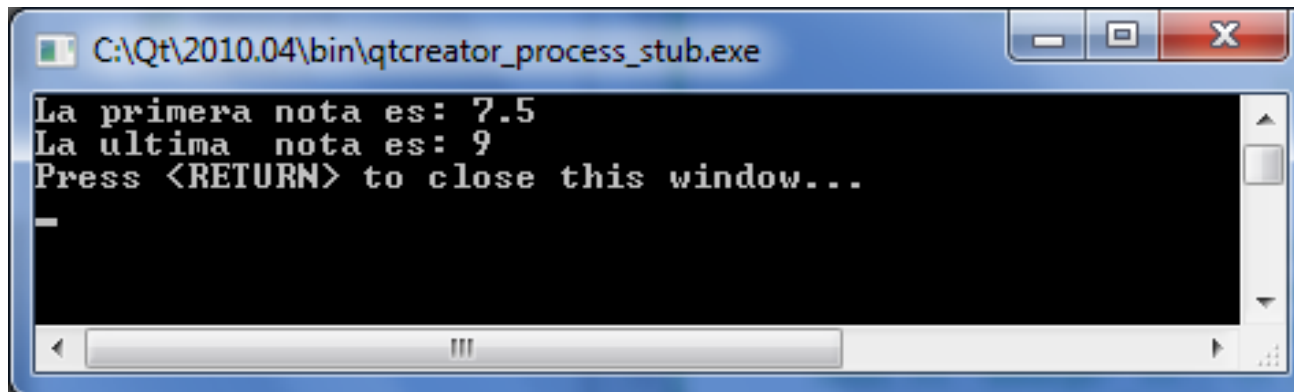
Arrays (“Arreglos”)

- Pueden ser inicializados en un vector de elementos entre “llaves” y separados por comas.

```
char initials[3]={ 'a' , 'c' , 'd' };
```

Arrays (“Arreglos”)

```
int main() {  
  
    float notas[3]={7, 5.5,9};  
    notas[0]=7.5;  
  
    cout << "La primera nota es: " << notas[0] << endl;  
    cout << "La ultima nota es: " << notas[2] << endl;  
    return 0;  
}
```



The screenshot shows a console window titled "C:\Qt\2010.04\bin\qtcreator_process_stub.exe". The output text is: "La primera nota es: 7.5", "La ultima nota es: 9", and "Press <RETURN> to close this window...".

Cadenas

- `char cadena[8];`
- Se suelen emplear “” para delimitar cadenas y ‘’ para delimitar caracteres.
- El último elemento es el carácter nulo ‘\0’
- “hola” es:

‘h’	‘o’	‘l’	‘a’	‘\0’
-----	-----	-----	-----	------
- `sizeof(“hola”)` es **5**
- “b” **no** es igual a ‘b’
 - “b” es un array de un elemento ‘b’+‘\0’
 - ‘b’ es un caracter

Cadenas

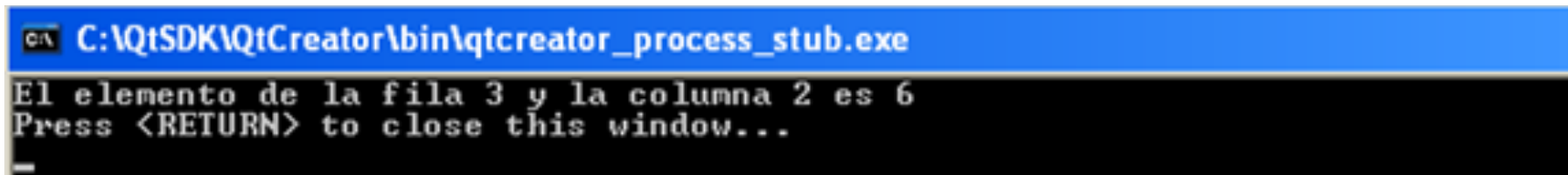
- Es importante el ‘\0’ para que se sepa donde está el final
- Así, en una misma variable, se pueden almacenar cadenas de distinta longitud
- Si no se pone el ‘\0’, al representarla, la cadena sigue hasta que casualmente encuentre otro ‘\0’

Arrays multidimensionales

- `type name[size1][size2][size3];`
- `int matriz[4][2]=`
`{`
`{1,2},`
`{3,4},`
`{5,6},`
`{7,8}`
`};`

Arrays multidimensionales

```
void main(){  
    int matriz[4][2] = {{1,2},{3,4},{5,6},{7,8}};  
    cout<<"el elemento de la fila 3 y la columna 2 es "<<matriz[2][1]<<endl;  
}
```



The screenshot shows a console window titled "C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe". The output text is "El elemento de la fila 3 y la columna 2 es 6" followed by "Press <RETURN> to close this window...". A cursor is visible at the bottom left of the console.

6. Flujo de Control

Flujo de Control

Para cada estudiante

Hacer

Estudiar

Hacer el examen

Si no estás de acuerdo con la calificación ➡

Ir a revisión

Mientras la asignatura no esté aprobada

Operadores para el flujo de control

- Operadores de Relacion
 < > <= >=
- Operadores de Igualdad/Desigualdad
 == (¡¡atención!!, no es =, que es asignación) !=
- Operadores lógicos
- && || !
- Todos devuelven:
 - 0 si la condición es falsa
 - 1 cuando la condición es válida
- En general, cualquier número diferente de cero significa “*true*” (salida “1”)

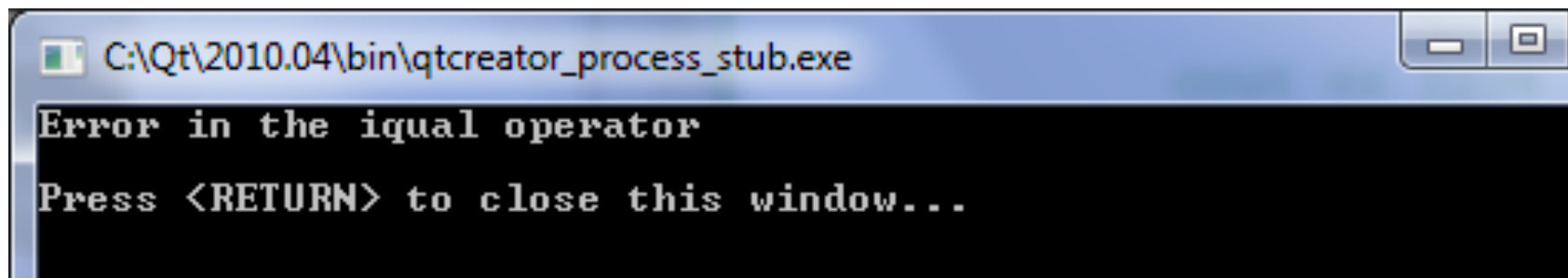
if

if(expresión)
 sentencia1;
 siguientes sentencias;

- Si se cumple “expresión” (salida lógica diferente de cero), se ejecuta “sentencia1” (y luego las siguientes sentencias).
- Si no se cumple “expresión” (salida lógica es cero), se ejecutan directamente las siguientes sentencias, pero no “sentencia1”.

If

```
void main(){  
    int i=0;  
    if(i==1)  
        cout<<"error en el operador de igualdad ";  
}
```



if-else

```
if(expresión)
    sentencia1;
else
    sentencia2;
siguientes sentencias;
```

if-else

Si “expresión” es verdadero, se ejecuta “sentencia1”.

Si no, se ejecuta sentencia2.

Y siempre después, se ejecutan las siguientes sentencias.

if-else

```
#include <iostream>
#include <math.h>

using namespace std;
int main(){
    double numero;
    cin>>numero;
    if(numero < 0)
        cout<<"Numero negativo"<<endl;
    else {
        cout<<"Numero positivo"<<endl;
        cout<<"La raíz cuadrada es "<<sqrt(numero)<<endl;
    }
    return 0;
}
```

If-else anidado

```
if(exp1)
{
    if(exp2)
    {
        sentencia1;
    }
    else
    {
        sentencia2;
    }
}
else
{
    sentencia3;
}
```

If-else anidado

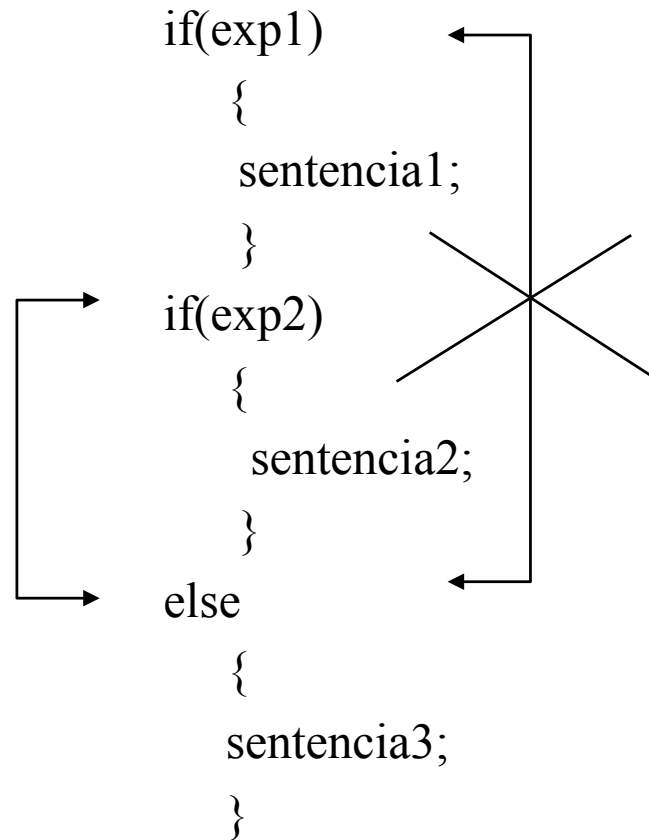
```
#include <iostream>

using namespace std;

int main() {
    int i,j;
    i=3;
    j=-3;
    if(i < 0) {
        if(j < 0)
            cout<<"i j menores que 0"<<endl;
        else
            cout<<"j no es menor que cero, aunque i sí lo sea"<<endl;
    }else
        cout<<"i no es menor que cero"<<endl;
    return 0;
}
```

If-else anidado

La sentencia “else” está asociada con el “if” más cercano



while

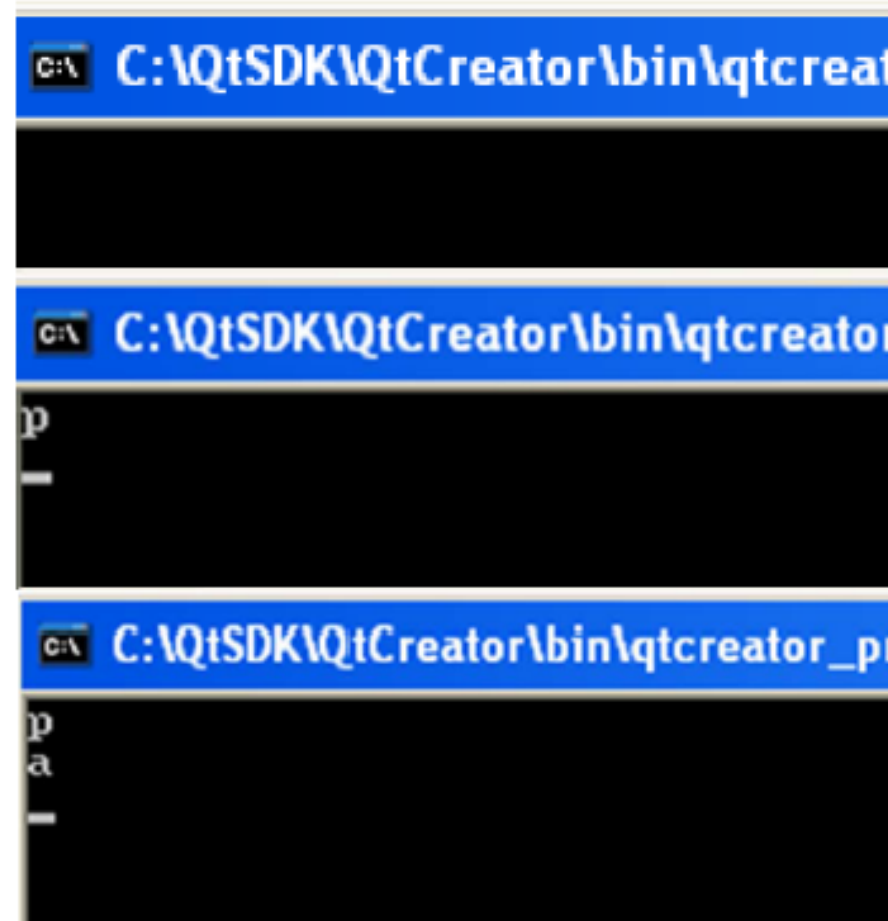
- Si queremos ejecutar sentencias que dependen de una condición

```
while(expresión)
{
    sentencia1;
}
```

while

```
#include <iostream>
using namespace std;

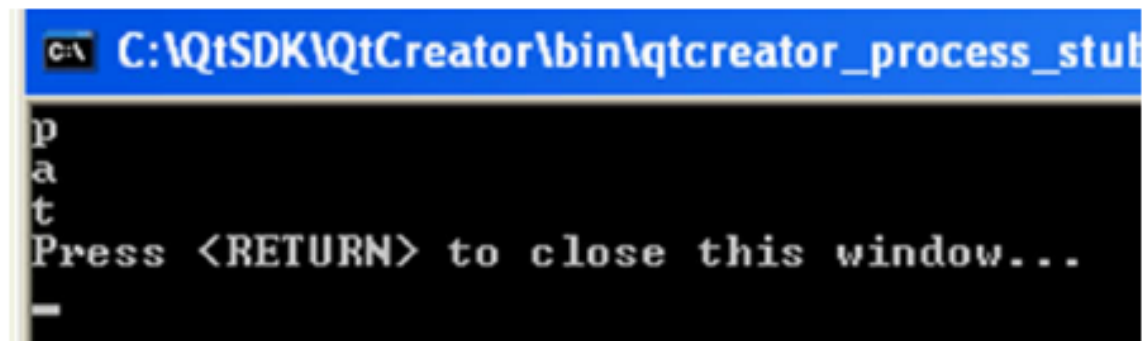
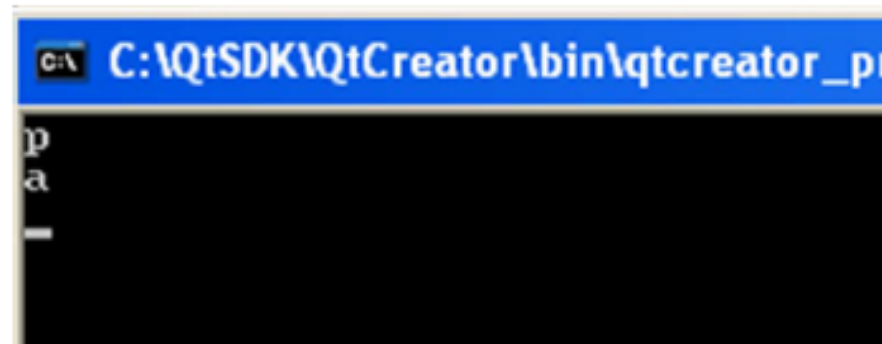
int main()
{
    char c= '\0' ;
    while (c!= 't')
        cin >> c;
    return 0;
}
```



while

```
#include <iostream>
using namespace std;

int main()
{
    char c= '\0';
    while (c!= 't')
        cin >> c;
    return 0;
}
```



for

```
expresión1;  
while(expresión2)  
{  
    sentencia1;  
    expresión3;  
}
```



Se ejecuta al principio

Se ejecuta en cada iteración

```
for(exp1;exp2;exp3)  
{  
    sentencia 1;  
}
```

En cada iteración, se verifica para decidir ejecutar o no una nueva iteración

for

- El uso común de “for” es en bucles con un número conocido de iteraciones

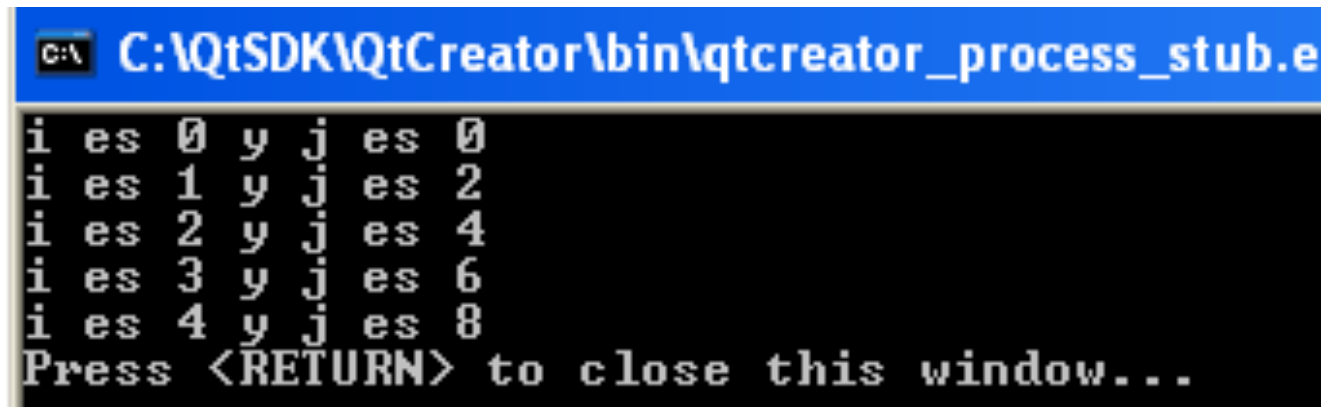
```
int a[10];
for(i=0; i<10; ++i)
{
    cout << "elemento" << " " << i << "es" << " " << a[i] << endl;
}
```

Diagram illustrating the components of the for loop:

- test**: Points to the condition `i < 10`.
- sentencias**: Points to the body of the loop (the code inside the curly braces).

for

```
#include <iostream>
using namespace std;
int main() {
    int i,j;
    for(i=0,j=0; i<5 && j<20; i++, j+=2)
        cout<<"i es "<<i<<" j es "<<j<<endl;
    return 0;
}
```



```
C:\QtSDK\QtCreator\bin\qtcreator_process_stub.e
i es 0 y j es 0
i es 1 y j es 2
i es 2 y j es 4
i es 3 y j es 6
i es 4 y j es 8
Press <RETURN> to close this window...
```

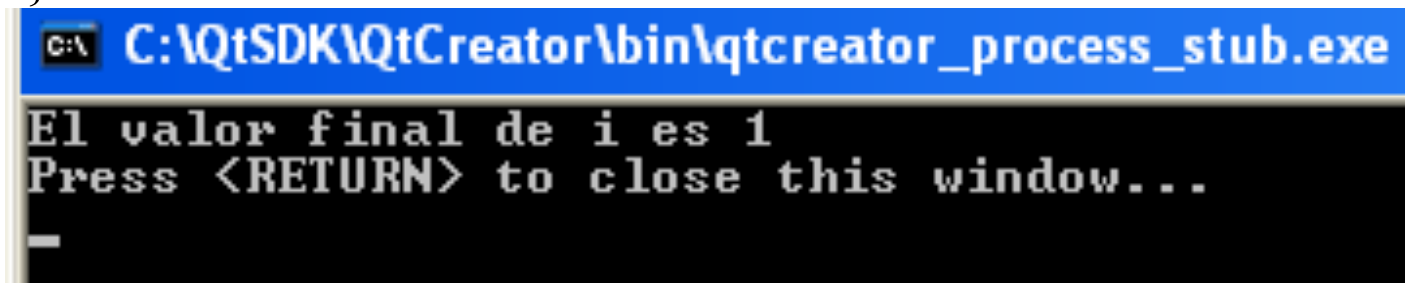
do-while

- Es similar a “while”, pero se utiliza cuando necesitamos ejecutar un conjunto de instrucciones por lo menos una vez (incluso cuando la expresión es falsa

```
do
{
    sentencia1;
}
while(expresión);
```

do-while

```
#include <iostream>
using namespace std;
int main() {
    int i=0;
    do{
        i= i +1;
    }while(i<0);
    cout<<"El valor final de i es "<<i<<endl;
    return 0;
}
```

A screenshot of a console window from Qt Creator. The title bar is blue and contains the text 'C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe'. The console output is in a monospaced font on a black background. It shows 'El valor final de i es 1' followed by 'Press <RETURN> to close this window...'. A single underscore character is visible on the line below the prompt.

“break” y “continue”

- “break” causa la salida de un bucle anidado (**interno**)

```
while(1){  
    cin >> x;  
    if(x<0.0)  
        break;  
    else  
        cout<<"raiz cuadrada"<<sqrt(x)<<endl;  
}
```

“break” y “continue”

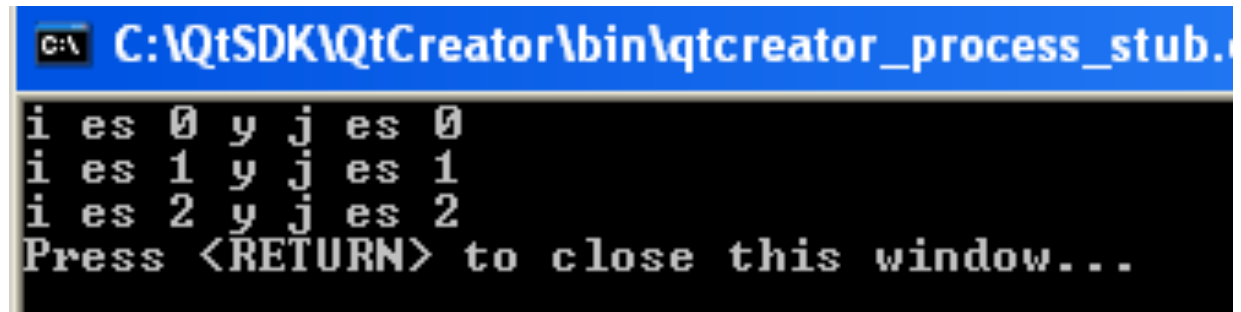
- **continue** interrumpe la iteración actual del bucle a la siguiente iteración

```
for(i=0;i<1000;i++){  
    c=getchar();  
    if('0'<=c && c<='9')  
        continue;  
}
```

- Solo se utiliza con **for**, **while** y **do-while**

break and continue

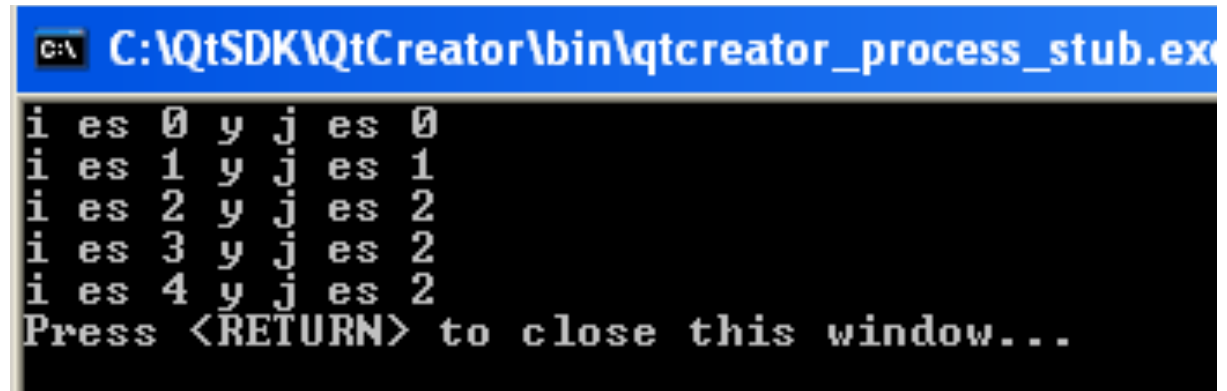
```
#include <iostream>
using namespace std;
int main() {
    int i,j=0;
    for(i=0; i<5; i++){
        cout<<"i es "<<i<<" j es "<<j<<endl;
        if(j>1)
            break;
        j++; //j+=1; o j=j+1;
    }
    return 0;
}
```



```
C:\QtSDK\QtCreator\bin\qtcreator_process_stub...
i es 0 j es 0
i es 1 j es 1
i es 2 j es 2
Press <RETURN> to close this window...
```

break and continue

```
#include <iostream>
using namespace std;
int main() {
    int i,j=0;
    for(i=0; i<5; i++){
        cout<<"i es "<<i<<" j es "<<j<<endl;
        if(j>1)
            continue;
        j++; //j+=1; o j=j+1;
    }
    return 0;
}
```



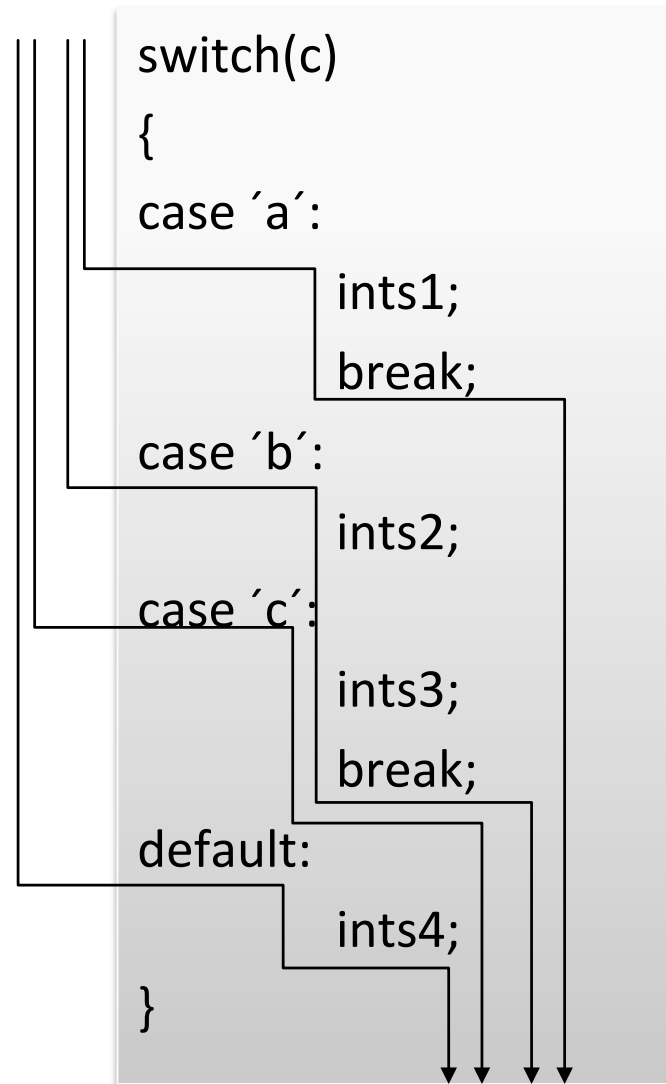
```
C:\QtSDK\QtCreator\bin\qtcreator_process_stub.exe
i es 0 j es 0
i es 1 j es 1
i es 2 j es 2
i es 3 j es 2
i es 4 j es 2
Press <RETURN> to close this window...
```

switch

- Generaliza el uso de múltiples sentencias “if-else”
- Normalmente se utiliza cuando existen varias opciones
- Verifica y chequea el valor de una variable introducida, permitiendo además valores por defecto.

switch

```
switch(c)
{
case 'a':
    ints1;
    break;
case 'b':
    ints2;
case 'c':
    ints3;
    break;
default:
    ints4;
}
```



switch

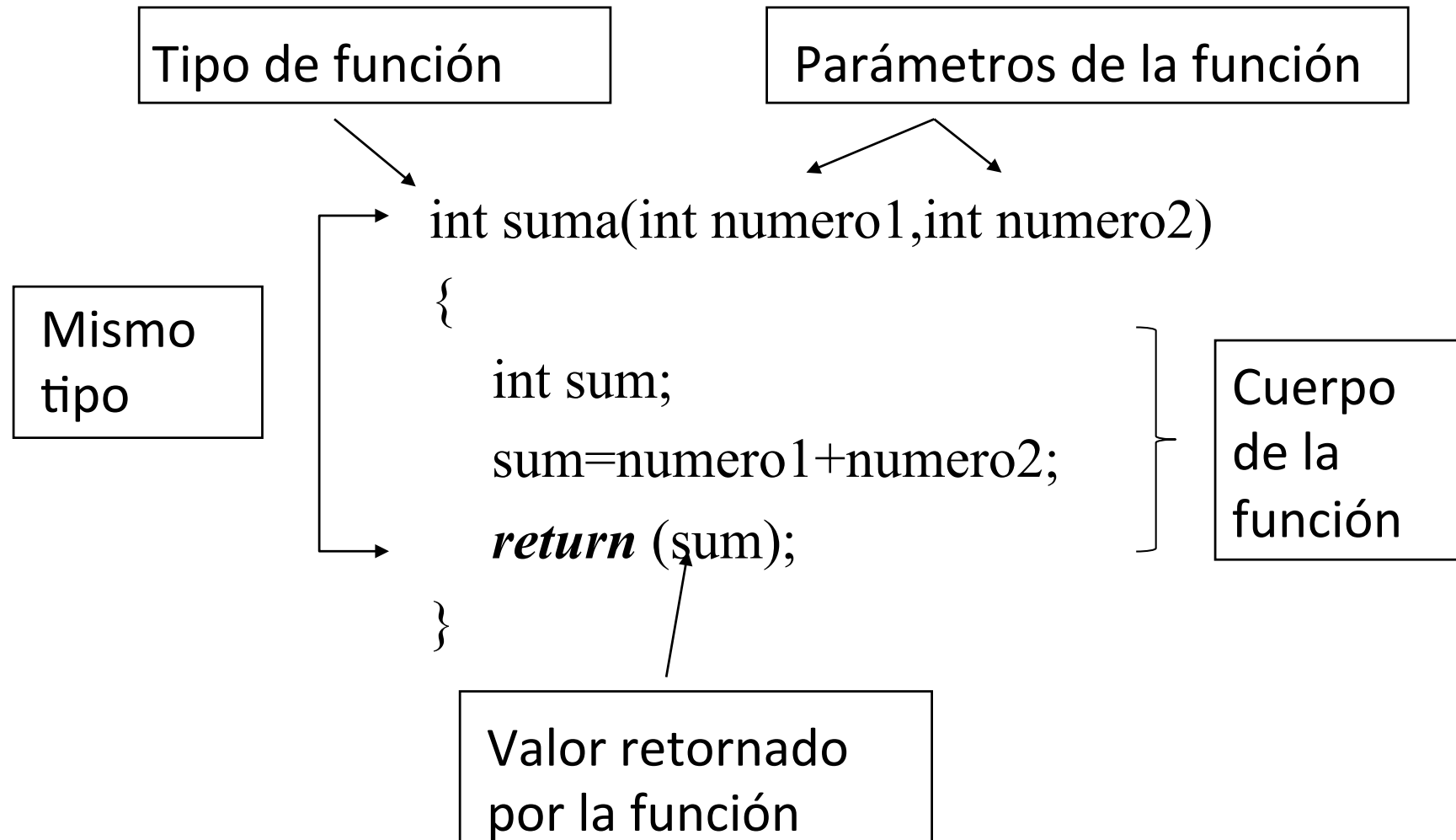
- Es típico poner “break” al final de cada caso para evitar entrar en el siguiente.
- Es muy util para menús con selección (“Escribe 1 para calcular la suma, 2 para calcular la resta, ...”).

7. Funciones

Funciones

- El lenguaje C++ está basado en el uso de funciones:
 - Funciones de sistema
 - Funciones de dispositivo/equipo.
 - Diseñadas por el usuario.
- Ideas:
 - No reinventemos la rueda
 - No repetir innecesariamente
 - Siempre la mayor claridad posible

Funciones



Prototipos de las funciones

```
int suma(int numero1,int numero2);←
```

Declaración de la función
Tipo nombre (parametros);

```
main()
```

```
{  int i, j=2;
```

```
    i=suma(j,6); ←
```

```
}
```

Llamada a la función: el
número y tipo de los
parámetros son verificados

```
int suma(int numero1,int numero2)
```

```
{
```

```
    int sum;
```

```
    sum=numero1+numero2;
```

```
    return(sum);
```

```
}
```

Implementación o
definición de la función

Variables locales y globales

- Las variables definidas en el cuerpo de la función son accesibles únicamente dentro de la función (variables **locales**).
- Las variables definidas fuera de todas las funciones son conocidas por todas las funciones (variables **globales**).

Variables locales y globales

```
int i, j;
```

i,j variables globales .
Pueden ser utilizadas por *main* y *funcion*

```
main()
```

```
{
```

```
    int a;
```

```
}
```

a es una variable local de *main*.
funcion no la conoce

```
int funcion()
```

```
{
```

```
    int b;
```

```
    int a;
```

```
}
```

b es una variable local in *funcion*
Pero *main* no sabe que existe

Las variables locales con el mismo nombre
pueden existir en muchas funciones.

Argumentos de la función

- Una función es ejecutada mediante la invocación de su nombre y argumentos.
- Los argumentos son pasados mediante parámetros (variables o constantes)
 - Cada argumento es evaluado y su valor es utilizado de forma local.
 - Los parámetros formales no son pasados a las funciones (la variable)

Argumentos y llamadas a función

```
main()
```

```
{
```

```
int a,b,sum;
```

```
a=5;
```

```
b=8;
```

```
sum=suma(a,b);
```

```
}
```

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

```
{
```

```
int sum;
```

```
sum=a+b;
```

```
return(sum);
```

```
}
```

Pila de memoria
en main

a	b	sum
5	?	?
5	8	?
5	8	?
5	8	?
5	8	13

pila de memoria
en suma

a	b	sum
?	?	?
?	?	?
5	8	?
5	8	13
5	8	13

Argumentos y llamadas a función

```
main()
{
    int a,b,c;
    a=5;
    b=8;
    c=funcion(a,b);
}

int funcion(int a, int b)
{
    int sum;
    a = a+1; b = b+1;
    sum=a+b;
    return(sum);
}
```

Pila de memoria
en main

a	b	c
5	?	?
5	8	?
5	8	?
5	8	?
5	8	?
5	8	15

pila de memoria
en function

a	b	sum
?	?	?
?	?	?
5	8	?
6	9	?
6	9	15
6	9	15



Paso de argumentos por referencia

- C++ dispone de un tipo especial de variable, llamada “referencia”, que puede ser utilizada como parámetro de la función que referencia al valor de la variable original.
- Una “referencia” es un “alias” de otra variable.
- Cualquier cambio realizado a través de la referencia se actualiza en la variable referenciada.
- Para declarar una referencia a una variable, se inserta el símbolo “&” delante del nombre de la variable a referenciar.

Paso de argumentos por referencia

```
#include <iostream>
using namespace std;

void main( )
{
    int count = 1;
    int &alias_for_count = count;
    cout << count << alias_for_count << endl;

    alias_for_count++;
    cout << count << alias_for_count << endl;

    count++;
    cout << count << alias_for_count << endl;
}
```



1	1
2	2
3	3

Paso de argumentos por referencia

- Cuando se pasa un argumento por referencia, el argumento debe ser una variable.
- Cuando se pasa el argumento por valor, el argumento puede ser un literal, una variable, una expresión, o incluso el valor de retorno de otra función

Referencias constantes como parámetros

```
#include <iostream>
using namespace std;

void max(const int &num1, const int &num2)
{
    if (num1 > num2)
        cout << num1 << endl;
    else
        cout << num2 << endl;
}

int main( )
{
    int x = 1;
    int y = 2;
    max(x, y);

    return 0;
}
```

Argumentos por defecto

- C++ permite declarar funciones con valores de argumentos por defecto
- Se pasan valores por defecto como parámetros de la función cuando dicha función es invocada sin argumentos

Argumentos por defecto

```
1  #include <iostream>
2  using namespace std;
3
4  void printArea(double radius = 1)
5  {
6      double area = radius * radius * 3.14159;
7      cout << area << endl;
8  }
9
10 int main( )
11 {
12     printArea( );
13     printArea(4);
14
15     return 0;
16 }
```

area is 3.14159
area is 50.2654

Arrays como argumento de funciones

- Tres formas:
 - Declarando el array con sus dimensiones
 - `function(int t[10])`
 - Declarando el array sin dimensiones
 - `function(int t[])`
 - A través de punteros
 - `function(int *t)`