INFORMÁTICA INDUSTRIAL

PROGRAMACIÓN BÁSICA C++ (II)

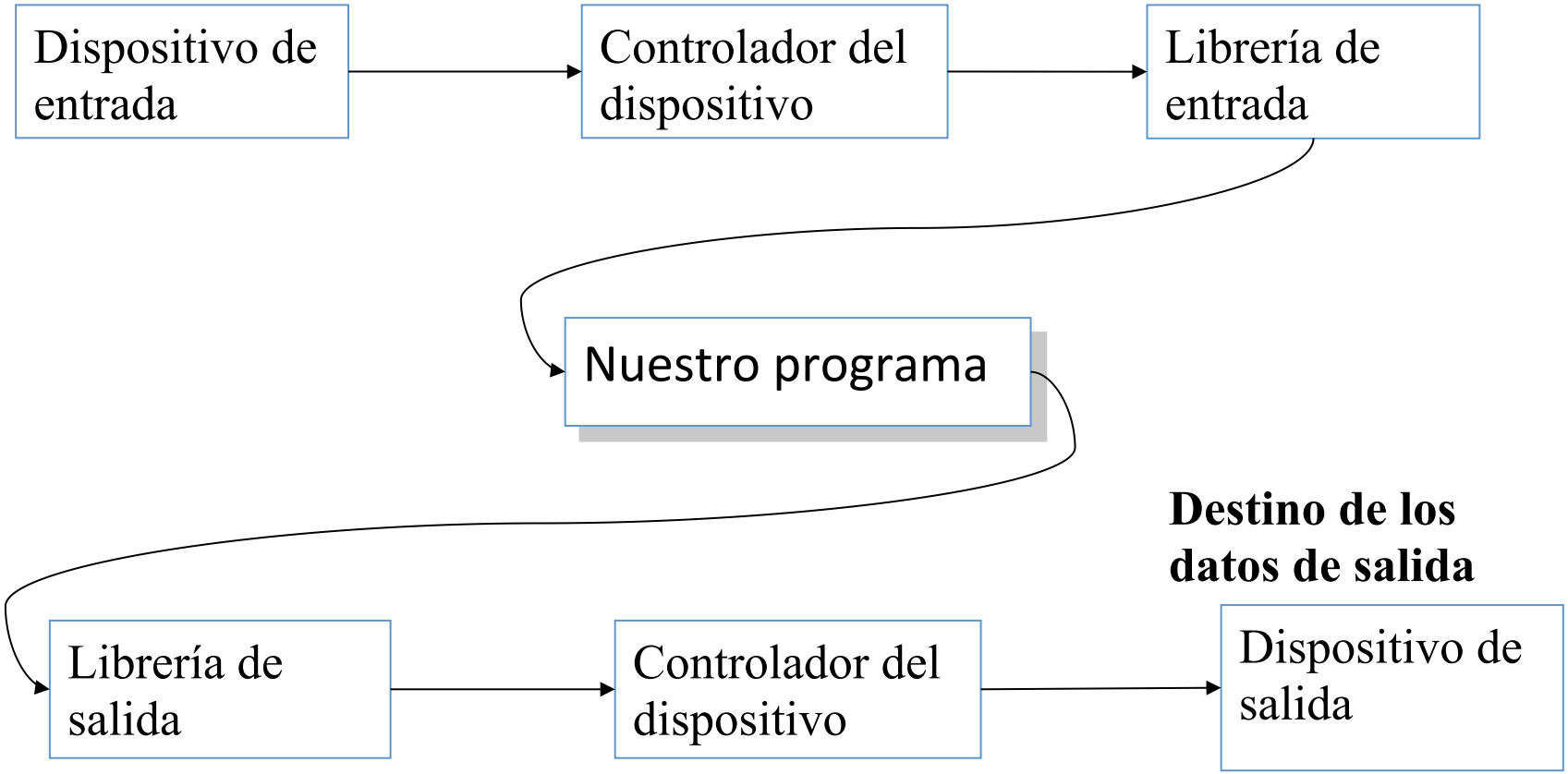
M. Abderrahim, A. Castro, J. C. Castillo
Departamento de Ingeniería de Sistemas y Automática

uc3m | Universidad Carlos III de Madrid

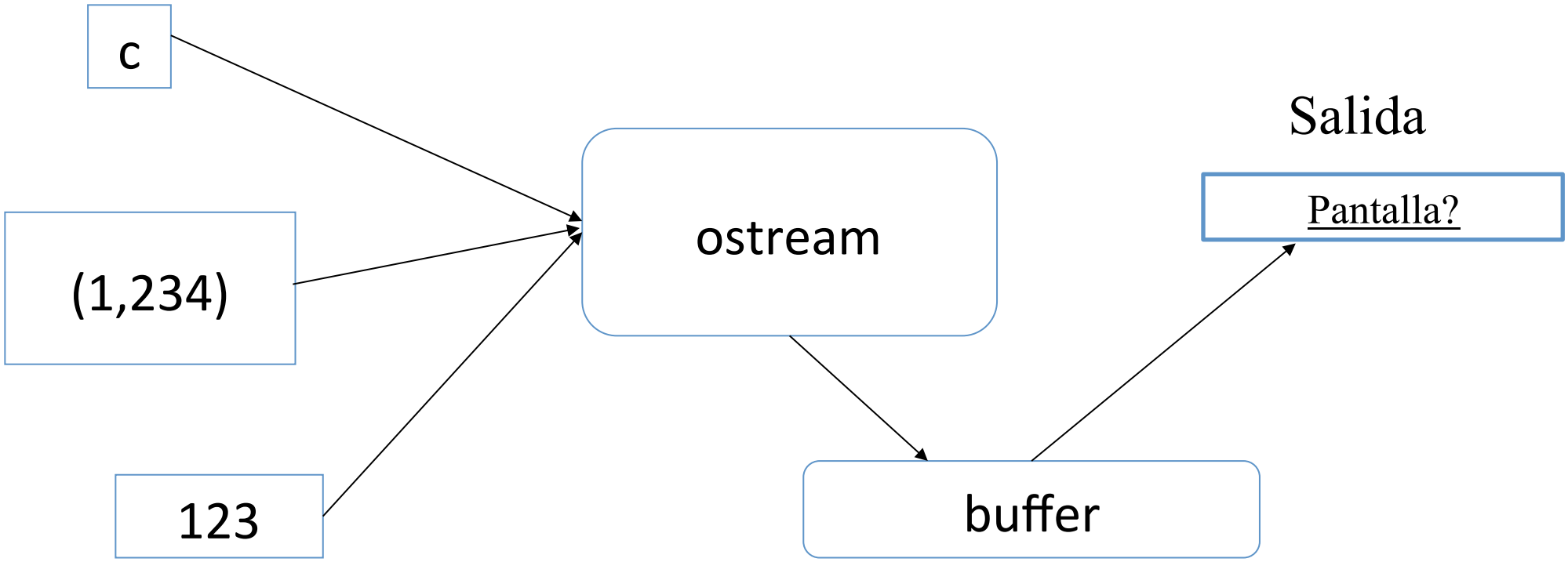
4. Entrada y salida estándar

Entrada y Salida

Fuente de datos:

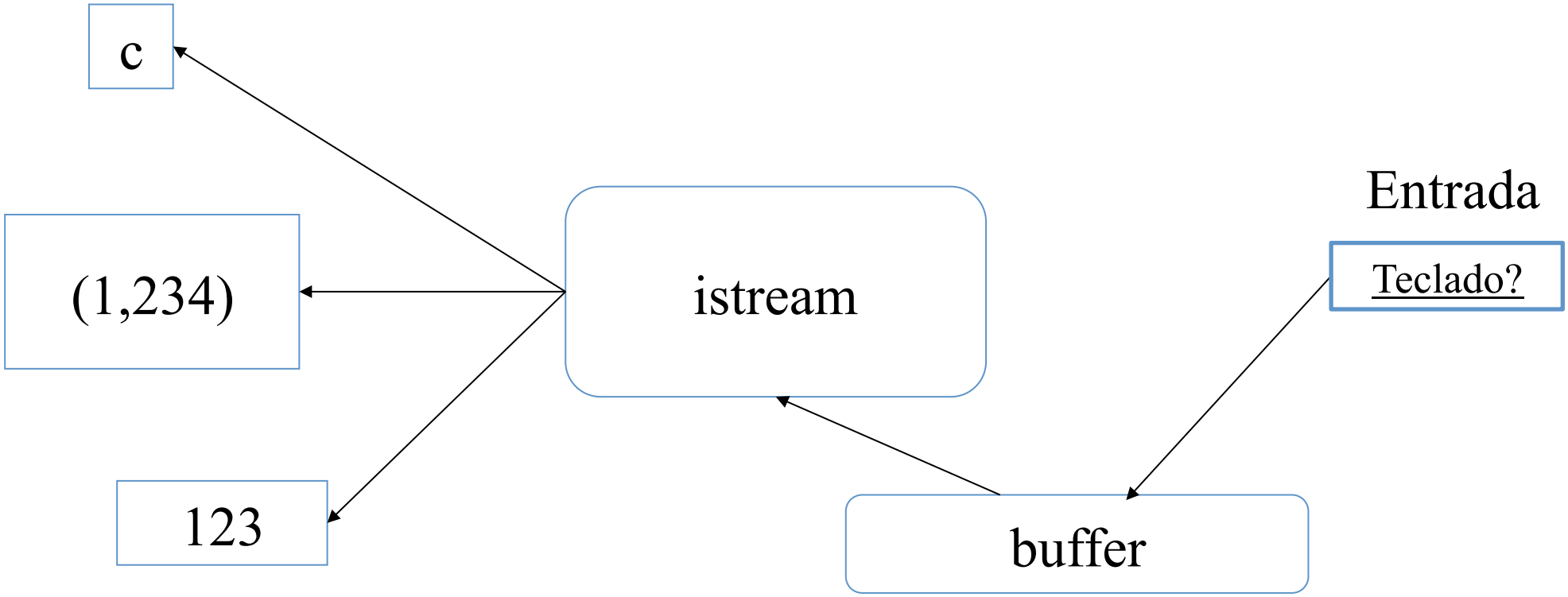


El Modelo de Flujo



- 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

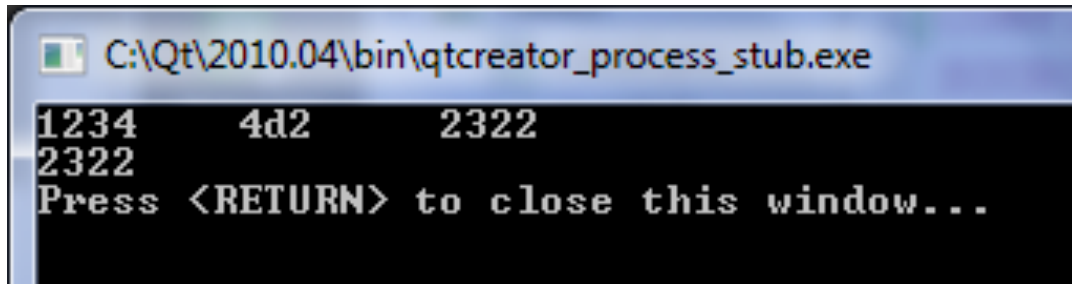
- Se puede modificar la “base” numérica
  - Base 10 == decimal; dígitos: 0 1 2 3 4 5 6 7 8 9
  - Base 8 == octal; dígitos: 0 1 2 3 4 5 6 7
  - Base 16 == hexadecimal; dígitos: 0 1 2 3 4 5 6 7 8 9 a b c d e f

```
// 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

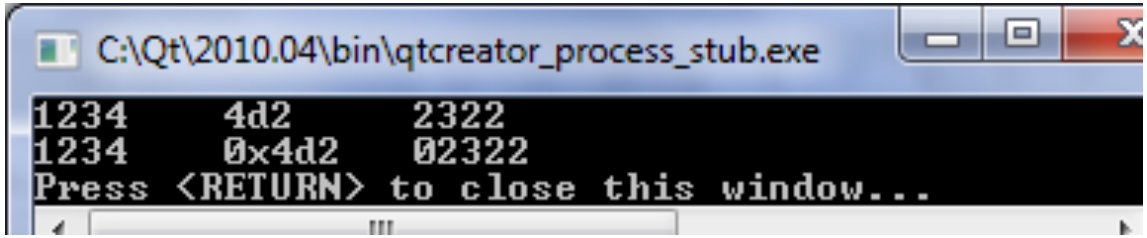
```
// ejemplo simple:
    cout << 1234 << '\t'
         << hex << 1234 << '\t'
       << oct << 1234 << '\n';
    cout << 1234 << '\n';      // la base octal sigue vigente

// resultados:
    1234  4d2 2322
    2322
```



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
```



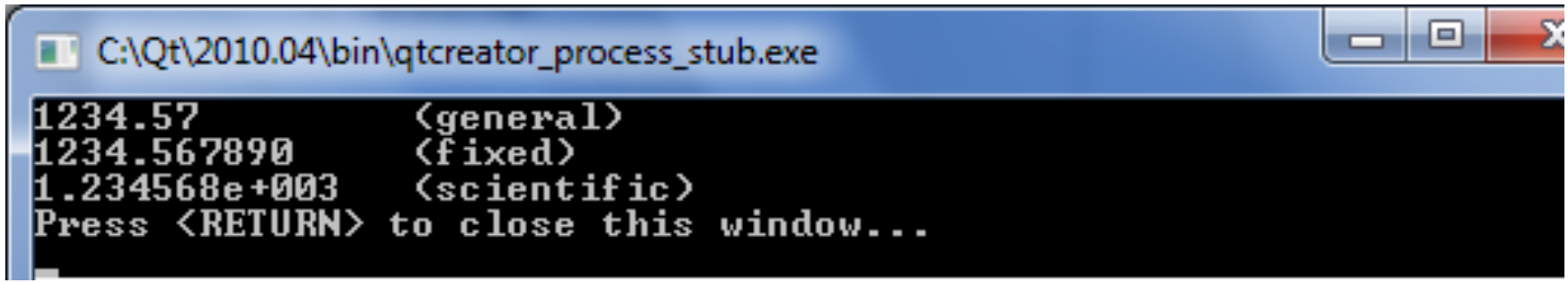
Manipuladores en punto flotante

- 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(fixed)\n"
     << scientific << 1234.56789 << "\t(scientific)\n";
```

Floating-point Manipulators

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

// resultados:
```



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.

INFORMÁTICA INDUSTRIAL

PROGRAMACIÓN BÁSICA C++ (II)

M. Abderrahim, A. Castro, J. C. Castillo
Departamento de Ingeniería de Sistemas y Automática

uc3m | Universidad Carlos III de Madrid