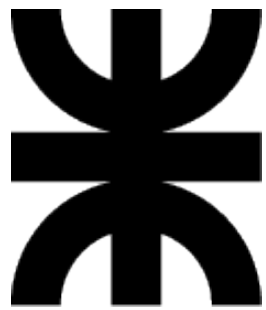
Informática II
Entrada/salida de flujo en C++

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Introducción

- Muchas de las características de E/S de C++ son orientadas a objetos por lo que hace uso de características como referencias y sobrecarga de funciones y de operadores.

Introducción

- Muchas de las características de E/S de C++ son orientadas a objetos por lo que hace uso de características como referencias y sobrecarga de funciones y de operadores.
- C++ utiliza *E/S con seguridad de tipos*, cada operación de E/S se realiza automáticamente de manera sensible al tipo de datos.

Introducción

- Muchas de las características de E/S de C++ son orientadas a objetos por lo que hace uso de características como referencias y sobrecarga de funciones y de operadores.
- C++ utiliza *E/S con seguridad de tipos*, cada operación de E/S se realiza automáticamente de manera sensible al tipo de datos.

stream o flujos

- La E/S en C++ ocurre por medio de *flujos* de bytes.

Introducción

- Muchas de las características de E/S de C++ son orientadas a objetos por lo que hace uso de características como referencias y sobrecarga de funciones y de operadores.
- C++ utiliza *E/S con seguridad de tipos*, cada operación de E/S se realiza automáticamente de manera sensible al tipo de datos.

stream o flujos

- La E/S en C++ ocurre por medio de *flujos* de bytes.
- Un flujo es simplemente una secuencia de bytes.

Introducción

- Muchas de las características de E/S de C++ son orientadas a objetos por lo que hace uso de características como referencias y sobrecarga de funciones y de operadores.
- C++ utiliza *E/S con seguridad de tipos*, cada operación de E/S se realiza automáticamente de manera sensible al tipo de datos.

stream o flujos

- La E/S en C++ ocurre por medio de *flujos* de bytes.
- Un flujo es simplemente una secuencia de bytes.
- En operaciones de entrada los bytes fluyen desde un dispositivo (teclado, disco, etc.) hacia la memoria principal.

Introducción

- Muchas de las características de E/S de C++ son orientadas a objetos por lo que hace uso de características como referencias y sobrecarga de funciones y de operadores.
- C++ utiliza *E/S con seguridad de tipos*, cada operación de E/S se realiza automáticamente de manera sensible al tipo de datos.

stream o flujos

- La E/S en C++ ocurre por medio de *flujos* de bytes.
- Un flujo es simplemente una secuencia de bytes.
- En operaciones de entrada los bytes fluyen desde un dispositivo (teclado, disco, etc.) hacia la memoria principal.
- En operaciones de salida los bytes fluyen desde la memoria principal hacia el dispositivo (pantalla, impresora, disco, etc.).

Introducción

- Muchas de las características de E/S de C++ son orientadas a objetos por lo que hace uso de características como referencias y sobrecarga de funciones y de operadores.
- C++ utiliza *E/S con seguridad de tipos*, cada operación de E/S se realiza automáticamente de manera sensible al tipo de datos.

stream o flujos

- La E/S en C++ ocurre por medio de *flujos* de bytes.
- Un flujo es simplemente una secuencia de bytes.
- En operaciones de entrada los bytes fluyen desde un dispositivo (teclado, disco, etc.) hacia la memoria principal.
- En operaciones de salida los bytes fluyen desde la memoria principal hacia el dispositivo (pantalla, impresora, disco, etc.).
- La aplicación asocia el significado a los bytes (caracteres ASCII, imágenes, voz o video digital, etc.).

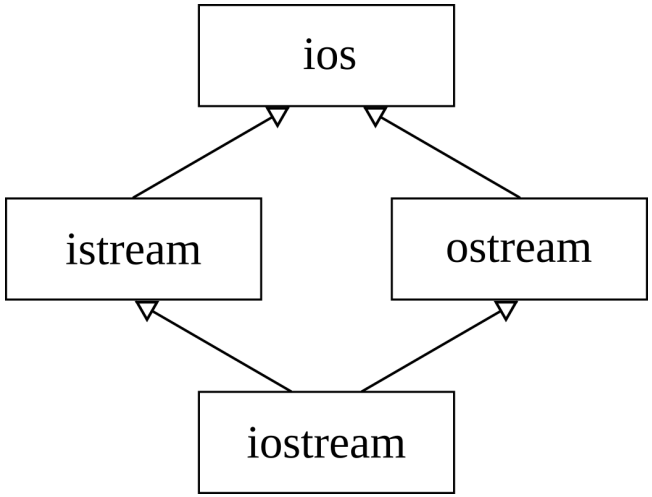
Clases y objetos para E/S

- La biblioteca `iostream` de C++ proporciona las capacidades de E/S.
- El archivo de cabecera `iostream` define los objetos:
  - `cin`: entrada estándar
  - `cout`: salida estándar
  - `cerr`: error estándar sin buffer
  - `clog`: error estándar con buffer
- El archivo de cabecera `iomanip` declara servicios útiles para realizar E/S con formato por medio de manipuladores parametrizados de flujo.

Clases y objetos para E/S

- La biblioteca `iostream` de C++ proporciona las capacidades de E/S.
- El archivo de cabecera `iostream` define los objetos:
  - `cin`: entrada estándar
  - `cout`: salida estándar
  - `cerr`: error estándar sin buffer
  - `clog`: error estándar con buffer
- El archivo de cabecera `iomanip` declara servicios útiles para realizar E/S con formato por medio de manipuladores parametrizados de flujo.

Herencia de clases de la biblioteca de E/S



Clases y objetos para E/S

La sobrecarga de operadores brinda una notación adecuada para realizar operaciones de E/S.

- El objeto `cin` es una instancia de la clase `istream`, y se dice que está *unido con* (o conectado con) el dispositivo de entrada estándar, que generalmente es el teclado:

```
cin >> calificacion; // los datos 'fluyen' en la dirección de
                     // las flechas hacia la derecha
```

Clases y objetos para E/S

La sobrecarga de operadores brinda una notación adecuada para realizar operaciones de E/S.

- El objeto `cin` es una instancia de la clase `istream`, y se dice que está *unido con* (o conectado con) el dispositivo de entrada estándar, que generalmente es el teclado:

```
cin >> calificacion; // los datos 'fluyen' en la dirección de
                     // las flechas hacia la derecha
```

- El objeto `cout` es una instancia de la clase `ostream`, y se dice que está *unido con* el dispositivo de salida estándar, que generalmente es la pantalla:

```
cout << calificacion; // los datos 'fluyen' en la dirección de
                      // las flechas hacia la izquierda
```

Clases y objetos para E/S

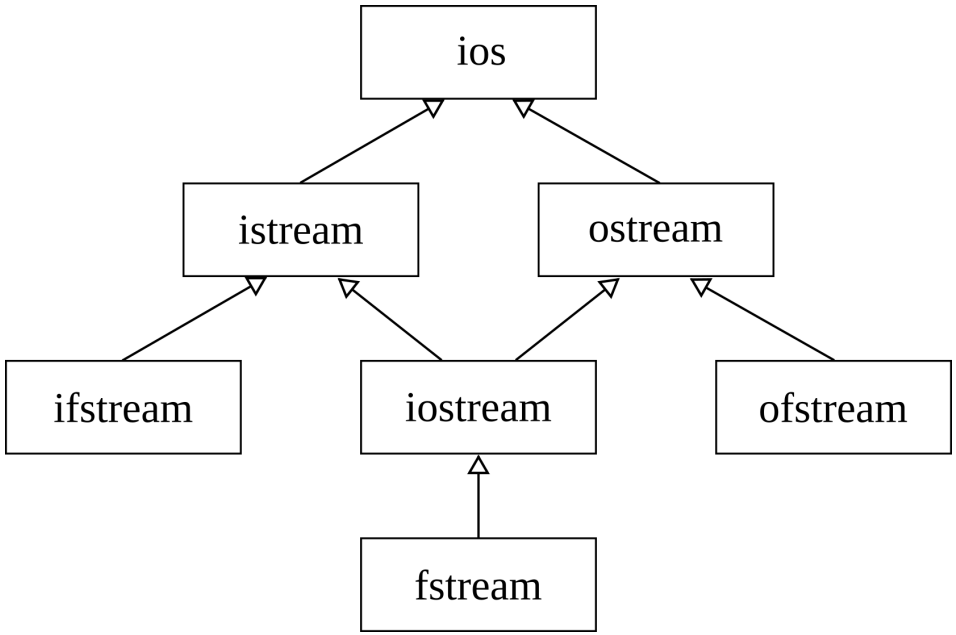
El procesamiento de archivos en C++ utiliza:

- la clase `ifstream` para realizar operaciones de entrada de archivos,
- la clase `ofstream` para operaciones de salida de archivos y
- `fstream` para operaciones de entrada/salida de archivos.

Clases y objetos para E/S

El procesamiento de archivos en C++ utiliza:

- la clase `ifstream` para realizar operaciones de entrada de archivos,
- la clase `ofstream` para operaciones de salida de archivos y
- `fstream` para operaciones de entrada/salida de archivos.



Salida de flujo

La clase `ostream` posibilita las operaciones de salida con formato y sin formato, para:

- salida de tipos de datos estándar

Salida de flujo

La clase `ostream` posibilita las operaciones de salida con formato y sin formato, para:

- salida de tipos de datos estándar
- salida de caracteres con la función miembro `put()`

Salida de flujo

La clase `ostream` posibilita las operaciones de salida con formato y sin formato, para:

- salida de tipos de datos estándar
- salida de caracteres con la función miembro `put()`
- salida sin formato con la función miembro `write()`

Salida de flujo

La clase `ostream` posibilita las operaciones de salida con formato y sin formato, para:

- salida de tipos de datos estándar
- salida de caracteres con la función miembro `put()`
- salida sin formato con la función miembro `write()`
- salida de enteros en formato decimal, octal y hexadecimal

Salida de flujo

La clase `ostream` posibilita las operaciones de salida con formato y sin formato, para:

- salida de tipos de datos estándar
- salida de caracteres con la función miembro `put()`
- salida sin formato con la función miembro `write()`
- salida de enteros en formato decimal, octal y hexadecimal
- salida de valores de punto flotante con diversa precisión, con puntos decimales forzados, en notación científica y en notación fija

Salida de flujo

La clase `ostream` posibilita las operaciones de salida con formato y sin formato, para:

- salida de tipos de datos estándar
- salida de caracteres con la función miembro `put()`
- salida sin formato con la función miembro `write()`
- salida de enteros en formato decimal, octal y hexadecimal
- salida de valores de punto flotante con diversa precisión, con puntos decimales forzados, en notación científica y en notación fija
- salida de datos justificados en campos con anchos no asignados

Salida de flujo

La clase `ostream` posibilita las operaciones de salida con formato y sin formato, para:

- salida de tipos de datos estándar
- salida de caracteres con la función miembro `put()`
- salida sin formato con la función miembro `write()`
- salida de enteros en formato decimal, octal y hexadecimal
- salida de valores de punto flotante con diversa precisión, con puntos decimales forzados, en notación científica y en notación fija
- salida de datos justificados en campos con anchos no asignados
- la salida de datos en campos rellenos con caracteres especificados

Salida de flujo – operador de inserción de flujo

- La salida se realiza con el operador de inserción de flujo, o sea, el operador << sobrecargado
- Se utiliza para mostrar datos de tipos básicos, cadenas, punteros y objetos definidos por el usuario

Salida de flujo – operador de inserción de flujo

- La salida se realiza con el operador de inserción de flujo, o sea, el operador << sobrecargado
- Se utiliza para mostrar datos de tipos básicos, cadenas, punteros y objetos definidos por el usuario

```
1 #include <iostream>
2
3 using std::cout;
4
5 int main()
6 {
7   cout << "Bienvenidos a C++!\n";
8   return 0;
9 }
```

Salida de flujo – operador de inserción de flujo

```
1  #include <iostream>
2
3  using std::cout;
4  using std::endl;
5
6  int main()
7  {
8    cout << "Bienvenidos a C++!" << endl;
9    return 0;
10 }
```

Salida de flujo – operador de inserción de flujo

```
1  #include <iostream>
2
3  using std::cout;
4  using std::endl;
5
6  int main()
7  {
8    cout << "Bienvenidos a C++!" << endl;
9    return 0;
10 }
```

- El manipulador de flujo `endl` imprime un carácter de nueva línea y además vacía el búfer de salida
- El buffer de salida también puede vaciarse con `cout << flush;`

Salida de flujo – operador de inserción de flujo

Ver los siguientes ejemplos:

1. Operador de inserción/extracción de flujo en cascada: `fig21.07.cpp`, `fig21_09.cpp`
2. Salida de variables `char *`: `fig21_08.cpp`

Salida de flujo – operador de inserción de flujo

Ver los siguientes ejemplos:

1. Operador de inserción/extracción de flujo en cascada: `fig21.07.cpp`, `fig21_09.cpp`
2. Salida de variables `char *`: `fig21_08.cpp`

Se pueden imprimir caracteres utilizando la función miembro `put()`, p.e.:

```
cout.put('A')
```

Las llamadas a `put()` pueden realizarse en cascada como en:

```
cout.put('A').put('\n');
```

Entrada de flujo – operador de extracción de flujo

- La entrada se realiza con el operador de extracción de flujo, o sea, el operador >> sobrecargado.

Entrada de flujo – operador de extracción de flujo

- La entrada se realiza con el operador de extracción de flujo, o sea, el operador >> sobrecargado.
- En general se ingnoran los *caracteres blancos* (p.e. espacios, tabuladores, nueva línea, etc.). Es posible modificar este comportamiento.

Entrada de flujo – operador de extracción de flujo

- La entrada se realiza con el operador de extracción de flujo, o sea, el operador `>>` sobrecargado.
- En general se ingnoran los *caracteres blancos* (p.e. espacios, tabuladores, nueva línea, etc.). Es posible modificar este comportamiento.
- Devuelve cero (falso) cuando se encuentra un fin de archivo en un flujo, de lo contrario devuelve una referencia del objeto que recibió el mensaje de extracción (p.e. `cin` en la expresion `cin >> calificacion`).

Entrada de flujo – operador de extracción de flujo

```
1  #include <iostream>
2
3  using std::cout;
4  using std::cin;
5  using std::endl;
6
7  int main()
8  {
9    int x, y;
10
11   cout << "Introduzca dos enteros: ";
12   cin >> x >> y;
13   cout << "La suma de " << x << " y " << y << " es: "
14     << (x + y) << endl;
15
16   return 0;
17 }
```

Entrada de flujo – operador de extracción de flujo

```cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
  int x, y;

  cout << "Introduzca dos enteros: ";
  cin >> x >> y;
  cout << "La suma de " << x << " y " << y << " es: "
    << (x + y) << endl;

  return 0;
}
```

Ver los ejemplos: `fig21_09.cpp`, `fig21_10.cpp` y `fig21_11.cpp`

Entrada de flujo – funciones `get` y `getline`

- `get()` sin argumento devuelve el caracter ingresado (incluso si es un caracter blanco).

Entrada de flujo – funciones `get` y `getline`

- `get()` sin argumento devuelve el caracter ingresado (incluso si es un caracter blanco).
- `get()` devuelve `EOF` cuando se lee el fin de archivo en el flujo.

Entrada de flujo – funciones `get` y `getline`

- `get()` sin argumento devuelve el caracter ingresado (incluso si es un caracter blanco).
- `get()` devuelve `EOF` cuando se lee el fin de archivo en el flujo.
- `getline()` lee una cadena completa incluyendo espacios.

Entrada de flujo – funciones `get` y `getline`

- `get()` sin argumento devuelve el caracter ingresado (incluso si es un caracter blanco).
- `get()` devuelve `EOF` cuando se lee el fin de archivo en el flujo.
- `getline()` lee una cadena completa incluyendo espacios.
- `getline()` inserta un carácter nulo después de la línea en el arreglo de caracteres.

Entrada de flujo – funciones `get` y `getline`

- `get()` sin argumento devuelve el caracter ingresado (incluso si es un caracter blanco).
- `get()` devuelve `EOF` cuando se lee el fin de archivo en el flujo.
- `getline()` lee una cadena completa incluyendo espacios.
- `getline()` inserta un carácter nulo después de la línea en el arreglo de caracteres.

Ver los ejemplos: `fig21_12.cpp`, `fig21_13.cpp` y `fig21_14.cpp`

Manipuladores de *stream* (flujo)

C++ proporciona varios *manipuladores de flujo* que realizan tareas de formato, brindando capacidades como:

- establecer el ancho de un campo y precisiones

Manipuladores de *stream* (flujo)

C++ proporciona varios *manipuladores de flujo* que realizan tareas de formato, brindando capacidades como:

- establecer el ancho de un campo y precisiones
- establecer y restablecer banderas de formato

Manipuladores de *stream* (flujo)

C++ proporciona varios *manipuladores de flujo* que realizan tareas de formato, brindando capacidades como:

- establecer el ancho de un campo y precisiones
- establecer y restablecer banderas de formato
- establecer el carácter de relleno en un campo

Manipuladores de *stream* (flujo)

C++ proporciona varios *manipuladores de flujo* que realizan tareas de formato, brindando capacidades como:

- establecer el ancho de un campo y precisiones
- establecer y restablecer banderas de formato
- establecer el carácter de relleno en un campo
- insertar una nueva línea en el flujo de salida y vaciar el flujo

Manipuladores de *stream* (flujo)

C++ proporciona varios *manipuladores de flujo* que realizan tareas de formato, brindando capacidades como:

- establecer el ancho de un campo y precisiones
- establecer y restablecer banderas de formato
- establecer el carácter de relleno en un campo
- insertar una nueva línea en el flujo de salida y vaciar el flujo
- insertar un carácter nulo en el flujo de salida e ignorar espacios en el flujo de entrada

Base de un *stream* de enteros

- Los enteros normalmente se interpretan como valores decimales (base 10)

Base de un *stream* de enteros

- Los enteros normalmente se interpretan como valores decimales (base 10)
- Para cambiar la base en la que se interpretan los enteros en un flujo, se puede utilizar:
  1. el manipulador `hex` para establecer la base en hexadecimal (base 16)
  2. el manipulador `oct` para establecer la base octal (base 8)
  3. el manipulador `dec` para restablecer la base del flujo a decimal
  4. el manipulador de flujo `setbase()` el cual puede tomar un argumento entero 10, 8 o 16

Base de un *stream* de enteros

- Los enteros normalmente se interpretan como valores decimales (base 10)
- Para cambiar la base en la que se interpretan los enteros en un flujo, se puede utilizar:
  1. el manipulador `hex` para establecer la base en hexadecimal (base 16)
  2. el manipulador `oct` para establecer la base octal (base 8)
  3. el manipulador `dec` para restablecer la base del flujo a decimal
  4. el manipulador de flujo `setbase()` el cual puede tomar un argumento entero 10, 8 o 16

Para utilizar los manipuladores de flujo es necesario incluir el archivo de cabecera `<iomanip>`.

Base de un *stream* de enteros

- Los enteros normalmente se interpretan como valores decimales (base 10)
- Para cambiar la base en la que se interpretan los enteros en un flujo, se puede utilizar:
  1. el manipulador `hex` para establecer la base en hexadecimal (base 16)
  2. el manipulador `oct` para establecer la base octal (base 8)
  3. el manipulador `dec` para restablecer la base del flujo a decimal
  4. el manipulador de flujo `setbase()` el cual puede tomar un argumento entero 10, 8 o 16

Para utilizar los manipuladores de flujo es necesario incluir el archivo de cabecera `<iomanip>`.

Ver código fuente ejemplo: `fig21_16.cpp`.

Precisión de punto flotante

- Es posible controlar la *precisión* de los números de punto flotante (es decir, el número de dígitos a la derecha del punto decimal), por medio:

Precisión de punto flotante

- Es posible controlar la *precisión* de los números de punto flotante (es decir, el número de dígitos a la derecha del punto decimal), por medio:
  1. del manipulador de flujo `setprecision` o
  2. de la función miembro `precision()`.

Precisión de punto flotante

- Es posible controlar la *precisión* de los números de punto flotante (es decir, el número de dígitos a la derecha del punto decimal), por medio:
  1. del manipulador de flujo `setprecision` o
  2. de la función miembro `precision()`.
- Una llamada a cualquiera de ellas ocasiona que se establezca la precisión para todas las operaciones de salida subsiguientes, hasta la siguiente llamada para establecer la precisión.

Precisión de punto flotante

- Es posible controlar la *precisión* de los números de punto flotante (es decir, el número de dígitos a la derecha del punto decimal), por medio:
  1. del manipulador de flujo `setprecision` o
  2. de la función miembro `precision()`.
- Una llamada a cualquiera de ellas ocasiona que se establezca la precisión para todas las operaciones de salida subsiguientes, hasta la siguiente llamada para establecer la precisión.
- La función miembro `precision()` sin argumentos devuelve la precisión actual establecida.

Precisión de punto flotante

- Es posible controlar la *precisión* de los números de punto flotante (es decir, el número de dígitos a la derecha del punto decimal), por medio:
  1. del manipulador de flujo `setprecision` o
  2. de la función miembro `precision()`.
- Una llamada a cualquiera de ellas ocasiona que se establezca la precisión para todas las operaciones de salida subsiguientes, hasta la siguiente llamada para establecer la precisión.
- La función miembro `precision()` sin argumentos devuelve la precisión actual establecida.

Ver código fuente ejemplo: `fig21_17.cpp`.

Banderas de estado de formato

Algunas de las banderas de estado de formato son:

- `ios::left`, `ios::right`, `ios::dec`, `ios::hex`, `ios::oct`, `ios::showbase`, `ios::showpoint`, `ios::scientific`, `ios::fixed`

Banderas de estado de formato

Algunas de las banderas de estado de formato son:

- `ios::left`, `ios::right`, `ios::dec`, `ios::hex`, `ios::oct`, `ios::showbase`, `ios::showpoint`, `ios::scientific`, `ios::fixed`

Las cuales se definen como una enumeración de la clase `ios`.

Banderas de estado de formato

Algunas de las banderas de estado de formato son:

- `ios::left`, `ios::right`, `ios::dec`, `ios::hex`, `ios::oct`, `ios::showbase`, `ios::showpoint`, `ios::scientific`, `ios::fixed`

Las cuales se definen como una enumeración de la clase `ios`.

- Estas banderas se pueden controlar por medio de las funciones miembro `flags()`, `setf()` y `unsetf()`

Banderas de estado de formato

Algunas de las banderas de estado de formato son:

- `ios::left`, `ios::right`, `ios::dec`, `ios::hex`, `ios::oct`, `ios::showbase`, `ios::showpoint`, `ios::scientific`, `ios::fixed`

Las cuales se definen como una enumeración de la clase `ios`.

- Estas banderas se pueden controlar por medio de las funciones miembro `flags()`, `setf()` y `unsetf()`
- Algunos programadores prefieren utilizar manipuladores de flujo

Banderas de estado de formato

Algunas de las banderas de estado de formato son:

- `ios::left`, `ios::right`, `ios::dec`, `ios::hex`, `ios::oct`, `ios::showbase`, `ios::showpoint`, `ios::scientific`, `ios::fixed`

Las cuales se definen como una enumeración de la clase `ios`.

- Estas banderas se pueden controlar por medio de las funciones miembro `flags()`, `setf()` y `unsetf()`
- Algunos programadores prefieren utilizar manipuladores de flujo
- Se puede utilizar la operación de nivel de bits or, `|`, para combinar varias opciones

Banderas de estado de formato

Algunas de las banderas de estado de formato son:

- `ios::left`, `ios::right`, `ios::dec`, `ios::hex`, `ios::oct`, `ios::showbase`, `ios::showpoint`, `ios::scientific`, `ios::fixed`

Las cuales se definen como una enumeración de la clase `ios`.

- Estas banderas se pueden controlar por medio de las funciones miembro `flags()`, `setf()` y `unsetf()`
- Algunos programadores prefieren utilizar manipuladores de flujo
- Se puede utilizar la operación de nivel de bits or, `|`, para combinar varias opciones

Ver código fuente ejemplo: `fig21_25.cpp` y `fig21_26.cpp`.