

INGRESO Y SALIDA DE DATOS DESDE EL LENGUAJE C++

Elaborado por: Juan Miguel Guanira Erazo - PUCP

El lenguaje C++ pretende marcar una diferencia sustancial con el lenguaje C, en el sentido que los elementos que se empleen para realizar las operaciones de entrada y salida lo hagan de una forma más clara y simple, sin tener que indicar, de manera expresa, cómo se transformarán los caracteres que vienen de la entrada estándar, ni cómo se van a convertir los datos que se envíen a la salida estándar. En otras palabras ya no existirá una cadena de formato que decida cómo se realizarán estas transformaciones sin importar las variables que se empleen en la operación, ni su cantidad.

Por otro lado, a diferencia del lenguaje C, las operaciones de entrada y salida ya no se hacen a través funciones sino que se hacen por medio de "**objetos**" definidos por "**clases**" que han sido diseñadas de manera apropiada para este fin.

BIBLOTECAS DE FUNCIONES Y ESPACIOS DE NOMBRES

Para poder emplear los elementos que permitan la entrada y salida de datos del medio estándar de entrada y salida usaremos primordialmente la biblioteca **iostream**. Sin embargo, para poder manipular los datos, sobre todo para la salida de datos, se podrá emplear la biblioteca **iomanip**, como veremos más adelante.

Por otro lado, debemos tener presente que para el lenguaje C++ se han venido desarrollando una infinidad de bibliotecas de funciones y clases, lo que ha traído como consecuencia que en muchos casos se empleen en un mismo programa dos a más bibliotecas que definan las mismas clases u objetos, lo que trae como consecuencia una interrupción en el proceso de compilación. Para solucionar este problema, el lenguaje ha implementado un mecanismo por medio del cual los elementos definidos en una biblioteca se pueden colocar dentro de un ámbito, identificado con un nombre, ese ámbito se conoce con el nombre de "**espacio de nombres**" (la implementación de un espacio de nombres se desarrollará en el capítulo de funciones, aquí solo lo emplearemos). Para el caso de las bibliotecas **iostream** e **iomanip**, el espacio de nombres que se ha definido en ambos casos se denomina **std** que viene del inglés **standard**.

OBJETO **cout**

El objeto **cout** permite enviar al medio estándar de salida el resultado de expresiones que se le proporcionan, de un modo similar a **printf**. Sin embargo antes de mostrar su sintaxis, debemos aclarar ciertos puntos.

Como se indicó líneas arriba, las operaciones de entrada y salida en C++ no se hacen a través de funciones sino por medio de objetos, en este sentido **cout** es un objeto

definido (también se dice "instanciado") de la clase `ostream` que se encuentran en la biblioteca `iostream`.

Debido a los espacios de nombres, tenemos tres formas de emplear el objeto `cout` (y cualquier otro de esta biblioteca o similar), a continuación las describimos:

1. Acompañando el objeto del espacio de nombres en cada invocación, esto es:

```
#include <iostream>
...
std::cout... // El espacio de nombres std debe aparecer en cada invocación al
              objeto, separándolos con el operador de ámbito (::).
...
std::cout...
```

2. Empleando la cláusula `using` al inicio del módulo y después de la orden de preprocesador `#include <iostream>`, de la siguiente manera:

```
#include <iostream>
using std::cout;
...
cout    // Aquí ya no es necesario emplear el espacio de nombres.
        // Sin embargo cualquier otro elemento de iostream si lo requerirá
```

3. Utilizando la cláusula `using` acompañándolo de cláusula `namespace`, como se indica a continuación:

```
#include <iostream>
using namespace std; // Con esto ningún elemento de iostream requerirá del
                     // espacio de nombres
...
cout...
```

Esta última forma de emplear el objeto `cout` es la que usaremos en adelante, debido a que una vez colocada la cláusula ya no se requerirá emplear el espacio de nombres en ningún elemento de las bibliotecas estándar de C++.

Por otro lado, para poder realizar la operación de salida de datos por medio del objeto `cout`, el lenguaje C++ ha sobrecargado el operador de desplazamiento de bits `<<` de modo que la operación se hará ahora a través de este operador que en adelante será nombrado como "**operador de inserción de flujo**". En el capítulo de funciones se desarrollará el tema de sobrecarga de operadores.

Modo de uso: (suponiendo que se ha colocado la cláusula `using namespace std`)

`cout << expresión;`

Esta operación permite enviar a la salida estándar un flujo de caracteres provenientes de la conversión del resultado de la expresión en una cadena de caracteres.

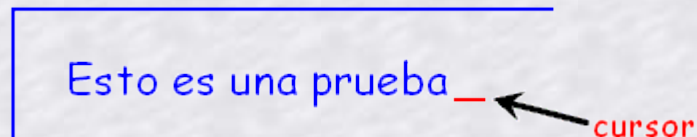
Una vez ejecutada la operación el sistema devuelve una referencia al objeto `cout`, por lo tanto esta operación se podrá concatenar con otras equivalentes de la manera siguiente:

```
cout << expresión1 << expresión2 << expresión3...;
```

Ejemplo:

```
cout << "Esto es una prueba";
```

Al ejecutar la orden se obtiene en la pantalla (medio estándar de salida):

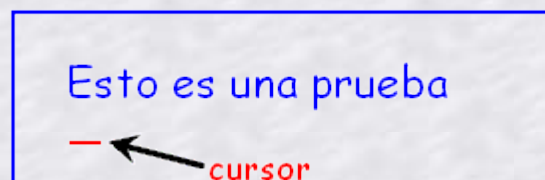
A terminal window with a blue border. Inside, the text "Esto es una prueba" is displayed in blue. At the end of the text is a red horizontal line representing the cursor. A black arrow points from the word "cursor" to this red line.

Como se puede observar la expresión es evaluada y enviada al medio estándar de salida, esto lo hace sin agregarle alguna otra información, como pueden ser espacios o cambios de línea.

Si se desea enviar los caracteres de cambio de línea se puede emplear el caracter `'\n'` como se hace en el lenguaje C, sin embargo el lenguaje C++ proporciona una herramienta que hace lo mismo, esta se denomina `endl`, entonces podríamos ejecutar la orden de la siguiente manera:

```
cout << "Esto es una prueba" << endl;
```

Al ejecutar la orden se obtiene en la pantalla:

A terminal window with a blue border. Inside, the text "Esto es una prueba" is displayed in blue. Below the text is a red horizontal line representing the cursor. A black arrow points from the word "cursor" to this red line.

Por otro lado está el problema de los espacios entre resultados y la tabulación de datos. Los siguientes ejemplos muestran estos casos:

Empecemos con el siguiente código:

```
int a = 23, b = 765, c = 9;  
cout << a << b << c << endl;
```

Al ejecutar la orden se obtiene en la pantalla:

A terminal window with a blue border. Inside, the numbers "237659" are displayed in blue, with no spaces between them. Below the numbers is a red horizontal line representing the cursor.

Como se observa, los datos se pegan, situación que no se puede aceptar. Para intentar solucionar este problema se podría emplear cadenas con espacios o tabuladores (`'\t'`),

sin embargo esto da muy malos resultados sobre todo cuando los datos son muy dispares en tamaño.

Pero antes de ver cómo solucionar este problema debemos entender primero que el operador `<<` se aplica una sola vez con el objeto `cout`, pero como esta operación, luego de enviar el resultado al medio de salida, devuelve una referencia al objeto `cout`, permite concatenarla de modo que parezca que estamos ejecutando una sola instrucción. En otras palabras si escribimos la siguiente instrucción:

```
cout << a << b << c << endl;
```

El compilador va a interpretar esa línea como si hubiéramos escrito:

```
cout << a;  
cout << b;  
cout << c;  
cout << endl;
```

Regresando al tema anterior, el lenguaje C++ nos proporciona herramientas que permite solucionar problema del formato. Estas herramientas se dan en dos categorías: mediante **"funciones miembro"** del objeto `cout` y mediante funciones que proporciona la biblioteca `iomanip`.

En el primer caso se emplea: `cout.width(n)`, donde `cout.width` representa el llamado a ejecución de la función miembro (o también denominado método) `width` que pertenece a (es miembro de) `cout`. `n` es un valor entero que indica la cantidad mínima de caracteres a emplear en la salida del valor (es equivalente al uso de `%nd` en el Lenguaje C). El ejemplo siguiente muestra el efecto de esta función:

```
int a = 2351, b = 765;  
cout << a;  
cout.width(10);  
cout << a << endl;  
cout << b << endl;
```

Al ejecutar el código se puede observar:

```
2351  
2351  
765  
—
```

Aquí podemos ver algunas cosas muy significativas, la primera es que la instrucción de formato y la instrucción que envía el valor de salida van por separado. La segunda y más importante es que el formato se aplica solo a la operación de salida inmediata, al resto no se aplica por lo que deberá anteponerlo en cada instrucción de salida.

Observe este otro ejemplo:

```
int a = 2351, b = 765, c = 1234;
cout.width(10);
cout << a << b << endl;
cout << c << endl;
```

Al ejecutar el código obtenemos:

```
2351765
1234
_
```

Observe que el único valor afectado por el formato es el de la variable "a".

Luego, para conseguir que se aplique a todas las variables se deberá hacer lo siguiente:

```
int a = 2351, b = 765, c = 1234;
cout.width(10);
cout << a ;           // Sin el cambio de línea: endl
cout.width(10);
cout << b << endl;
cout.width(10);
cout << c << endl;
```

Con lo que obtendremos lo siguiente:

```
2351      765
1234
_
```

En el segundo caso emplearemos una función definida en la biblioteca **iomanip**. Se trata de **setw(n)**. Esta función, al igual que **cout.width(n)**, define la mínima cantidad de caracteres en a que se representará el valor, sin embargo la forma de emplearla es mucho más sencilla y práctica, como veremos a continuación:

```
int a = 2351, b = 765, c = 1234;
cout << setw(10) << a << setw(10) << b << endl;
cout << setw(10) << c << endl;
```

```
2351      765
1234
_
```

El resultado es igual al del caso anterior pero del código podemos observar que con esta forma se puede concatenar en una sola instrucción, sin embargo, como en el primer caso el formato solo se aplica a la expresión contigua.

Es bueno indicar que **setw** forma parte del espacio de nombres **std** de la biblioteca **iomanip**, por lo que al igual que con **cout** se requiere incluir este espacio de nombres explícitamente en el programa en cualquiera de sus formas.

Banderas de formato

Las banderas de formato (**format flags**) son constantes que permiten definir ciertos atributos que influenciarán en la forma cómo aparecerán los datos en el medio de salida.

El ejemplo siguiente muestra alguno de ellos:

```
int a = 39963, b = 765, c = 1234;
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << left << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << right << setw(10) << a << setw(10) << b << setw(10) << c << endl;
```

La salida de este programa será:

	39963	765	1234
39963	765	1234	
	39963	765	1234

Observe que el formato se mantiene en todos los casos, sin embargo el uso de los elementos **"left"** y **"right"** harán que los resultados se alineen a la izquierda o a la derecha respectivamente. También se puede ver que una vez que se ejecuta el elemento, la alineación se mantenga hasta que se use de otro elemento lo cambie.

Un segundo ejemplo de estas banderas podemos verlas a continuación:

```
int a = 39963, b = 765, c = 1234;
cout << setw(10) << a << setw(10) << b << c << endl;
cout << hex << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << uppercase << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << oct << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << dec << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << nouppercase;
```

Al ejecutar el programa se verá lo siguiente:

39963	765	1234
9c1b	2fd	4d2
9C1B	2FD	4D2
116033	1375	2322
39963	765	1234

Las banderas, "**hex**", "**oct**" y "**dec**" controlan la base (16, 8 y 10) en la que aparecerán los valores en la salida. Las banderas "**uppercase**", y "**nouppercase**" determinan la forma en que se mostrarán las letras en el formato hexadecimal, mayúsculas o minúsculas respectivamente. Observe también que los formatos se mantienen hasta que se levante otra bandera.

Finalmente mostraremos cómo poder rellenar los espacios colocados en los formatos con otro caracter diferente:

```
int a = 39963, b = 765, c = 1234;
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout.fill('0');
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout.fill(' ');
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << setfill('X');
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << setfill(' ');
```

La salida será como sigue:

39963	765	1234
0000039963	000000765	0000001234
39963	765	1234
XXXXX39963	XXXXXXX765	XXXXXX1234

Formato en valores de punto flotante

Por defecto, los valores de punto flotante aparecerán con un formato pre establecido. El ejemplo siguiente muestra este detalle:

```
double f = 3.1415926535;
double g = 0.0000012345;
```

```
cout << f << endl;
cout << -f << endl;
cout << g << endl;
cout << -g << endl;
```

La salida será:

```
3.14159
-3.14159
1.2345e-006
-1.2345e-006
```

Para dar format a valores de punto flotante, C++ requiere del uso de dos herramientas, una que define la precisión en la que aparecerá el número (***cout.precision(n)*** o ***setprecision(n)***) y la otra que interpretará el significado de la precisión (***fixed***).

Por defecto ***cout.precision(n)*** y ***setprecision(n)***, define la cantidad de cifras o dígitos que se mostrarán en el medio de salida, ***entiéndase que esto se refiere al número de dígitos, no al número de cifras decimales***. El siguiente ejemplo muestra este detalle:

```
double f = 314.15926535;
```

```
cout << setw(14) << f << endl;    // Que es equivalente a:
cout.precision(10);               // cout << setprecision(10) << setw(14) << f << endl;
cout << setw(14) << f << endl;    //
cout.precision(3);                // cout << setprecision(3) << setw(14) << f << endl;
cout << setw(14) << f << endl;    //
```

```
314.159
314.1592653
314
```

La figura anterior confirma lo explicado, ***cout.precision(n)*** y ***setprecision(n)*** no definen el número de decimales sino el número de dígitos que aparecerán en la salida.

Por eso se requiere otra herramienta, en este caso ***fixed***. Al activar esta bandera, el valor de precisión se interpretará como el número de decimales.

El siguiente programa muestra este efecto:


```
double f = 3.1415926535;
// Que es equivalente a:
cout << fixed;
cout.precision(5); // cout << setprecision(5) << setw(14) << f << endl;
cout << setw(14)
cout.precision(10); // cout << setprecision(10) << setw(14) << f << endl;
cout << setw(14) << f << endl; //
cout.precision(3); // cout << setprecision(3) << setw(14) << f << endl;
cout << setw(14) << f << endl; //
cout.unsetf(ios::fixed); // Se desactiva la orden fixed
```

La ejecución mostrará lo siguiente:

```

      3.14159
3.1415926534
      3.142
_
```

OBJETO **cin**

El objeto cin, al igual que la función scanf, permite leer, uno por uno, los caracteres de un flujo de caracteres que ingresa del medio estándar de entrada. Los caracteres son convertidos de acuerdo al tipo de variable que acompaña al objeto.

Modo de uso: (suponiendo que se ha colocado la cláusula using namespace std)

cin >> expresión;

Como se observa, aquí también se ha sobrecargado un operador, en este caso el operador de bits >>, que en adelante será nombrado como "**operador de extracción de flujo**". De igual manera, como en cout, la operación devolverá una referencia al objeto cin, por lo que la operación se podrá concatenar de la siguiente manera:

cin >> expresión1 >> expresión2 >> expresión3...;

Ejemplo:

```
int a;
double f;
cin >> a >> f;
```

Al ejecutar la orden, el sistema detendrá el programa para que el usuario pueda colocar los datos en el buffer de entrada, luego de presionar la tecla **ENTER [↵]** el programa convertirá los caracteres del flujo de entrada en la representación binaria correspondiente al tipo de dato de la variable y lo asignará a ella. Los caracteres ingresados deberán corresponder con el tipo de la variable de lo contrario el proceso se detendrá, asignando a la variable lo que se haya podido convertir hasta ese

momento. El proceso termina satisfactoriamente cuando se encuentre un separador (espacio en blanco, cambio de línea o tabulador).

Como hemos indicado, el objeto `cin` devuelve una referencia al mismo objeto, por esta razón no podemos contar con un valor de respuesta que permita saber si la operación se realizará con éxito o no.

ENTRADA Y SALIDA DE CARACTERES

La entrada y salida de caracteres desde C++ se puede realizar, de igual manera que con los números, con los objetos `cin` y `cout` empleando una variable una variable de tipo `char`, sin embargo existen algunos métodos que pueden ser muy útiles dado el caso. A continuación describiremos algunos de éstos:

Método `cin.get()`:

El método toma un caracter del buffer de entrada y lo entrega al programa. El siguiente ejemplo muestra su efecto:

```
char c = 'A';  
cout << "Ingrese un texto: ";  
c = cin.get();  
cout << "C = " << c << endl;
```

Al ejecutarlo, primero se verá el mensaje en la ventana y el programa se detendrá, como se ve a continuación:

```
Ingrese un texto: _
```

Luego de ingresar el texto y presionar la tecla **ENTER** [↵] se verá lo siguiente:

```
Ingrese un texto: Hola↵  
C = H
```

Método `cout.put(c)`:

El método toma el caracter contenido en la variable "`c`" y lo envía al medio de salida. El siguiente ejemplo muestra su uso:

```
char c = 'A';  
cout << "Ingrese un texto: ";  
c = cin.get();  
cout << "C = "  
cout.put(c);  
cout << endl;
```

Al ejecutarlo se mostrará de manera idéntica al resultado del programa anterior.

Método `cin.unget()`:

El método envía al buffer de entrada el último carácter extraído por el método `cin.get()`.

Los ejemplos siguientes mostrarán el efecto de este método:

// Ejemplo 1, sin usar `cin.unget()`

```
char a, b, c;

cout << "Ingrese un texto: ";
a = cin.get();
b = cin.get();
c = cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
```

Al ejecutarlo, se obtendrá lo siguiente:

```
Ingrese un texto: Hola↵
A = H
B = o
C = l
```

// Ejemplo 2, usando `cin.unget()`

```
char a, b, c;

cout << "Ingrese un texto: ";
a = cin.get();
cin.unget();           // ⇐
b = cin.get();
c = cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
```

Al ejecutarlo, se obtendrá lo siguiente:

```
Ingrese un texto: Hola↵
A = H
B = H
C = o
```


Método `cin.peek()`:

El método obtiene una copia del carácter del buffer de entrada, sin extraerlo y lo entrega al programa.

En el ejemplo siguiente se podrá ver el efecto de este método:

```
char a, b, c d;  
cout << "Ingrese un texto: ";  
d = cin.peek();  
a = in.get();  
b = cin.get();  
c = cin.get();  
cout << "A = " << a << endl;  
cout << "B = " << b << endl;  
cout << "C = " << c << endl;  
cout << "D = " << d << endl;
```

La ejecución de este código mostrará lo siguiente:

```
Ingrese un texto: Hola  
A = H  
B = o  
C = l  
D = H
```

Método `cin.putback(c)`:

El método es similar a `cin.unget()` con la diferencia que el usuario puede enviar al buffer de entrada el carácter que desee.

En el ejemplo siguiente se podrá ver su efecto:

```
char a, b, c d;  
cout << "Ingrese un texto: ";  
a = cin.get();  
cin.putback('M');  
b = cin.get();  
c = cin.get();  
d = cin.get();  
cout << "A = " << a << endl;  
cout << "B = " << b << endl;  
cout << "C = " << c << endl;  
cout << "D = " << d << endl;
```

La ejecución de este código mostrará lo siguiente:

```
Ingrese un texto: Hola↵
A = H
B = M
C = o
D = l
—
```

Función **ws**:

La función permite extraer del buffer de entrada todos los espacios (espacios en blanco, tabuladores y cambios de línea) que vaya encontrando hasta encontrar un caracter no blanco.

Observe los siguientes códigos:

```
// Ejemplo 1, sin usar ws
char a, b, c d;
cout << "Ingrese un texto: ";
a = cin.get();
b = cin.get();
c = cin.get();
d = cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
cout << "D = " << d << endl;
```

El resultado será el siguiente:

```
Ingrese un texto: H__ola↵
A = H
B = _
C = _
D = o
—
```

Diagrama de anotación: Una llave roja agrupa los caracteres de espacio en blanco (los dos guiones bajos) en la línea de entrada "H__ola". Una flecha roja apunta desde esta llave hacia la etiqueta "espacios".

```
// Ejemplo 2, usando ws
char a, b, c d;
cout << "Ingrese un texto: ";
a = cin.get();
cin << ws;
b = cin.get();
c = cin.get();
d = cin.get();
```

```
cout << "A = " << a << endl;  
cout << "B = " << b << endl;  
cout << "C = " << c << endl;  
cout << "D = " << d << endl;
```

El resultado será el siguiente:

Ingrese un texto: H__ola.↵

A = H

B = o

C = l

D = a

—