Entrada y Salida estándar en C++

Un programa en C++ puede realizar operaciones de entrada y salida de varias formas distintas. A continuación describiremos lo que se conoce como flujos. Un flujo de entrada no es más que la serie de entradas que alimentan un ordenador para que el programa las utilice. En esta sección supondremos que la entrada proviene del teclado. Asimismo, un flujo de salida es la serie de salidas que el programa genera. En esta sección supondremos que las salidas se envían a la pantalla de un terminal.

Salidas con cout

Los valores de variables se pueden enviar a la pantalla empleando cout. Es posible enviar a la pantalla cualquier combinación de variables y cadenas.
Por ejemplo:

```
cout << num_dulces << "dulces\n";
```

Esta instrucción le dice a la computadora que despliegue dos cosas: el valor de la variable **num_dulces** y la cadena "**dulces\n**". Esta instrucción es equivalente a:

```
cout << num_dulces ;
cout << "dulces\n";
```

Se pueden incluir expresiones aritméticas, como se muestra en el siguiente ejemplo:

```
cout << "El precio total es: " << (precio1 + precio2);
```

El doble símbolo **<<** se conoce como **operador de inserción.**

En los ejemplos anteriores hemos utilizado una secuencia de carácteres especial '**\n**'**.** Esta secuencia indica al ordenador que salte a una nueva línea en la salida.
Existen otros secuencias especiales. Todas ellas empiezan con **'\'** y se les llama secuencias de escape. No deben aparecer espacios entre **'\'** y el carácter.

Algunas secuencias de escape:

\n	**nueva línea**
\t	**tabulación horizontal.**
****	**diagonal invertida**
\"	**comillas dobles**

Ejemplo:

```
// Envia una línea en blanco a la salida estándar
//(pantalla)
cout << "\n";
```

Otra forma de enviar un salto de línea es la utilización de **endl**.

```
// Envia una línea en blanco a la salida estándar (pantalla)
cout << endl;
```

Formateo de salidas con funciones de flujos

La organización de la salida de un programa en C++ es su **formato.** En C++ podemos controlar el formato con órdenes que determinan detalles tales como el número de espacios entre los elementos y el número de dígitos después del punto decimal
Veamos algunas funciones o métodos asociadas a los flujos de salida:

- ***precision***: Fijar el número de cifras después de la coma: precision

  ```
  cout.precision(2);
  ```

- ***width:*** Indica al flujo cuantos espacios debe usar al enviar un elemento a la salida.

  ```
  cout.width(4);
  ```

  El método ***width*** solamente afecta a la siguiente salida por pantalla.

- **fill:** Indica el carácter con lo que completa los espacios en blanco cuando se manda un elemento a la salida y se fija el número de espacios con la función width.

  ```
  cout.fill('a');
  ```

  Al igual que el método width, el método ***fill*** solamente afecta a la siguiente salida por pantalla.

- **setf:** es una abreviatura de *set flags*, que significa *establecer banderas*. Una bandera es algo que indica que se debe efectuar algo de una de dos posibles maneras. Si damos una determinada bandera como argumento de setf, dicha bandera le dirá a la computadora que escriba las salidas en ese flujo de alguna forma específica. El efecto sobre el flujo depende de la bandera.
  Banderas:

  ios::fixed : hace que el flujo envíe a la salida los número en formato de punto fijo (no notación e).

ios::showpoint : le dice al flujo que siempre incluya un punto decimal en los números en punto flotante.

ios::showpos : le dice al flujo que escribe un signo + para los números positivos.

ios::left: si esta bandera está establecida y se da algún valor de anchura de campo con una llamada a la función miembro width, el siguiente número que se despliegue estará en el extremo derecho del espacio especificado con width.

ios::rigth: igual que la bandera anterior pero a la derecha.

```
cout.setf(ios::fixed);
cout.setf(ios::showpos);
cout.setf(ios::showpoint);
cout.setf(ios::left);
cout.setf(ios::right);
```

- **unsetf:** Cualquier bandera establecida se puede desactivar con esta función miembro. Por ejemplo:

```
cout.unsetf(ios::showpos);
```

Nota: una vez establecida una bandera o valor se mantiene en todo el programa a no ser que se utilice la función miembro unsetf o se varie el valor con el método adecuado.

Entrada con cin

`cin` es el flujo de entrada estandar. Supondremos que la entrada estandar es el teclado. Veamos un ejemplo:

```
cin >> num_pasteles;
```

En la sentencia anterior lo que hacemos es leer un dato introducido por teclado y almacenarlo en una variable '**`num_pasteles`**'.

Entrada y Salida de caracteres

Todos los datos se envian como caracteres. Cuando un programa envía a la salida el número 10, en realidad lo que envía son los dos caracteres '1' y '0'. Qué la computadora interprete el número 10 o los caracteres '1' y '0' depende del programa.

C++ ofrece algunos recursos para la entrada y salida de datos de caracteres.

Funciones miembro o métodos:

- **get:** permite a un programa leer un carácter de entrada y guardarlo en una variable de tip char. Está función toma un argumento, que debe ser una variable de tipo char. Por ejemplo:

```
char siguiente_simbolo;

cin.get(siguiente_simbolo);
```

Comparación de **`cin >>`** y **`cin.get`**:

```
char c1,c2,c3,c4,c5;

//Introduce 4 caracteres en 2 lineas diferentes

cin.get(c1);
cin.get(c2);
cin.get(c3);
cin.get(c4);

//Imprimo los caracteres leidos

cout << "Los 4 caracteres leidos son:\n";

cout.put(c1);
cout.put(c2);
cout.put(c3);
cout.put(c4);
```

Con cin >>:

```
//Prueba con cin

char c1, c2, c3, c4;

//Introduce 4 caracteres separados por espacios
cin >> c1;
cin >> c2;
cin >> c3;
cin >> c4;

cout << "Los 4 caracteres leidos son:\n";

cout << c1 << c2 << c3 << c4;
```

La diferencia consiste en que el primero lee los espacios y '\n' y los almacena en las variables y el segundo ignora estos caracteres y los utiliza como carácteres separadores.

- **put:** es análoga a la función miembro get sólo que se emplea para salida. Mediante put un programa puede enviar a la salida un carácter. La función miembro put recibe un argumento que debe ser una expresión de tipo char (constante o variable). Por ejemplo:

```
char ch;

ch='a';
cout.put(ch);
```

- **ignore:** este método permite descartar caracteres existentes en el buffer de entrada.

```
char ch1,ch2;

cout << " Introduce 2 caracteres en líneas diferentes:\n";

cin.get(ch1);
cin.ignore(); //Ignora '\n'
cin.get(ch2);
cin.ignore(); // Ignora '\n'

cout << "Los caracteres leídos son:\n";

cout.put(ch1);
cout.put(' '); //Escribo un espacio de separacion
cout.put(ch2);
```