Principales diferencias entre la versión de g++ 2.95.4 y 3.2

Uso de espacios de nombres (namespaces)

Cuando un proyecto alcanza un gran tamaño, con miles o millones de líneas, se pueden producir problemas de colisión de nombres (identificadores): variables globales, clases o funciones con el mismo nombre.

En algunos lenguajes de programación (C, Basic, etc.) no existe una solución (el programador la tiene que articular de algún modo). En C++ se han incorporado (julio de 1998) los espacios de nombres, que permiten dividir el espacio general de nombres en subespacios distintos e independientes.

El proceso consiste en declarar un espacio de nombres asignándole un identificador y delimitándolo por un bloque entre llaves. Dentro de este bloque pueden declararse los elementos correspondientes al mismo: variables, clases, funciones, etc. Los elementos declarados dentro de un espacio de nombres son accesibles de diversos modos:

- Mediante el operador de alcance (también llamado "de resolución de ámbito"), "`::`".
- Mediante "`using namespace nombre;`"

Por ejemplo, en el siguiente código se declaran dos espacios de nombres y se muestra cómo se pueden emplear:

```
namespace Espacio1
{
  int a;
}

namespace Espacio2
{
  int a;
}

using namespace Espacio2;
```

```
Espacio1::a = 3;  // Hace falta indicar su namespace
a = 5;  // Se refiere a Espacio2::a
```

Las librerías estándar de C++ están definidas dentro de un espacio de nombres llamado `std`. Por tanto, si se quiere evitar el tener que emplear el operador de ámbito constantemente, hay que añadir la sentencia "`using namespace std;`" justo después de la inclusión (`#include`) de las librerías en un fichero.

Ficheros de cabecera

A partir de ahora se distinguen dos tipos de ficheros de cabecera: C y C++. Los de C no hacen uso de espacios de nombre y llevan el prefijo "c". Por ejemplo, `cstdio`, `cstdlib` o `cstring`. Los de C++ hacen uso de espacios de nombre (el espacio de nombres estándar `std`). En ambos casos, no llevan la extensión `.h`.

Temporalmente, es posible que estén disponibles los ficheros de cabecera antiguos (con la extensión `.h`), pero no se aconseja su uso. Por tanto, a partir de ahora hay que escribir:

```
#include <iostream>     // Para usar: cin, cout, …
#include <cstring>      // Para usar: strcpy(), strcmp(), …
#include <string>       // Para usar la clase string
```

Hay que llevar cuidado y no confundir `string.h` (librería de C), `ctring` (librería de C adaptada para ser usada en C++) y `string` (librería de C++).

Implementación de la librería `iostream`

Se ha detectado una diferencia importante entre la implementación que acompaña a la versión 2.95.4 y la que acompaña a la versión 3.2. Cuando se imprime una cadena (`char *`) que apunta a `NULL`, en 2.95.4 aparece en pantalla "`(null)`", mientras que en la 3.2 no aparece nada y deja de funcionar el flujo de salida (lo que se imprima a continuación no aparece), aunque el programa sigue funcionando.

Por ejemplo, "`cout << (char *) NULL << endl;`" funciona en 2.95.4 (muestra por pantalla la cadena "`(null)`"), pero no funciona en la 3.2.

Realmente, en ambos casos el flujo de salida se corrompe y pasa a un estado "incorrecto". Los siguientes métodos se pueden emplear sobre cualquier flujo (`fstream`, `ifstream` y `ofstream`), incluidos los flujos predefinidos del sistema (`cin`, `cout` y `cerr`), para conocer el estado de un flujo:

- `bool bad()`:devuelve `true` si se ha producido un error fatal en el flujo, `false` en caso contrario.
- `void clear(iostate flags = goodbit)`: limpia los flags asociados al flujo; si no se indica un flag, limpia todos los flags.
- `bool good()`: devuelve `true` si no se han producido errores en el flujo, `false` en caso contrario.