Localizaciones en C++

Gerardo Aburruzaga, Inmaculada Medina, Francisco Palomo

Abril de 2004

1. Conceptos

Una *localidad* es un objeto que representa un conjunto de preferencias culturales, como el criterio de comparación de cadenas, la clasificación de caracteres, el separador de decimales o de miles en los números, el símbolo monetario, etc.

En C, y por tanto en C++ por compatibilidad, para la programación de localidades se empleaba sobre todo la función de la biblioteca estándar

```
#include <locale.h>
char* setlocale(int categoria, const char* localizacion);
```

La *categoría* es una macro o constante entera que representa un grupo de características, como `LC_TIME` para todo lo relativo a formatos de fecha y hora, y la *localización* es una cadena de caracteres dependiente del sistema aunque normalizada por ISO que describe el idioma, el país o región, el conjunto de caracteres y quizá alguna característica más; a veces se permiten *alias*, como por ejemplo `"spanish"` para `"es_ES.ISO-8859-1"` o `"es_ES@euro"`. El estándar ISO C o C++ sólo define la localización `"C"`, y el estándar POSIX.1 define además `"POSIX"` (que es igual a la anterior). Además la cadena vacía hace que se tome «del entorno».

Por supuesto en C++ es preferible incluir `<clocale>`, donde *setlocale()* está ahora en el espacio de nombres *std*. Pero de todas formas esta función está diseñada para trabajar con funciones de la biblioteca de C, por lo que seguramente no funcionará con las de la de C++. En este caso hay que usar las que nos proporciona la biblioteca estándar de C++ en `<locale>`.

2. Modo de empleo básico

El manejo de localizaciones en C++ es bastante complejo; aparte de las clases y tipos ya incluidos, se pueden crear nuevas localizaciones y extender las existentes mediante las llamadas *facetas*. Aquí solamente se verá el uso más básico, necesario para que nuestros programas se adapten a la localización existente o a la que se quiera.

En primer lugar hay que incluir por supuesto la cabecera `<locale>`. A continuación conviene definir un objeto de la clase ***locale***. Se podría hacer global, definiéndolo por ejemplo antes de *main()*. Como parámetro del constructor se le pasaría una cadena de caracteres, como la *localización* de *setlocale()*. Lo mejor suele ser la cadena vacía, `""`, que hace que se tome del entorno, con lo cual puede modificarse fácilmente justo antes de ejecutar el programa. Ejemplo:

```
#include <locale>
...

std::locale mi_localidad("");

int main() { ... }
```

Cuando se ejecute el programa:

```
1% a.out
2% LC_ALL="spanish" a.out
3% LC_COLLATE="german" LC_NUMERIC="es_ES" a.out
```

En el primer caso se ejecuta el programa tomando los valores de las variables de entorno relativas a la localización que estén definidas y exportadas. En GNU/Linux pueden verse fácilmente con el programa `locale`.

En el segundo, al programa se le pasa la variable de entorno `LC_ALL` con el valor que se ve, un alias para el español. Esta variable tiene preferencia sobre todas las demás categorías.

En el tercero se emplea el alemán para la clasificación de cadenas de caracteres y el español para los separadores numéricos.

Una vez establecida la localidad para todo el programa, hay que usarla. Se presentan a continuación los casos típicos.

2.1. Comparación de cadenas de caracteres

La forma más fácil es con el *operator() ()* de *locale*, que devuelve -1 si la 1.ª cadena es menor que la 2.ª, $+1$ si es mayor, y 0 si son iguales.

```
void f(const string& s1, const string& s2)
{
  int i = mi_localidad(s1, s2);
  int j = locale("C")(s1, s2); // equivale a   j = s1 < s2;
}
```

Siguiendo con el ejemplo anterior, si *s1* vale `"Álava"` y *s2* vale `"aljibe"`, i valdrá -1 si la localidad del entorno es la española, y j valdrá $+1$. Aquí se ve cómo en cualquier momento se puede cambiar la localización sin más que crear un objeto temporal con el constructor deseado, al que se aplica en este caso el operador de llamada a función.

En el siguiente ejemplo se ve cómo un objeto *locale* se puede emplear como objeto función; se ordena un vector de cadenas `vector<string> v` empleando para comparar los *string* la localidad del entorno:

```
  sort(v.begin(), v.end(), mi_localidad);
```

2.2. Clasificación de caracteres

Los caracteres se agrupan en categorías: alfabéticos, alfanuméricos, numéricos, espacios, blancos, dígitos hexadecimales, etc. En C se empleaban las macros `is`*X* de `<ctype.h>`, siendo X la categoría del carácter, como `alpha` o `digit`.

En C++ se definen funciones similares en la cabecera `<locale>`, pero reciben un 2.º parámetro más: la localización. También hay otra función (*is()*) y una enumeración con los nombres de las categorías de caracteres. Ejemplos:

```
if (isalpha(c, mi_localizacion)) ...  // c es alfabético
bool b = is(space | alnum, c);
```

En la 2.ª línea, *b* es **true** si *c* es alfanumérico o espacio.

2.3. E/S numérica

La clave para la entrada/salida es *imbuir* una localización en un flujo. Esto se hace con *imbue()*. Por ejemplo, para las entrada y salida estándar:

```
cout.imbue(mi_localidad);
cin.imbue(mi_localidad);
...
double d;
cin >> d;
cout << d << endl;
```

Si la entrada es `12.34` la salida será `12` en español, porque el separador de decimales es la coma. Si la entrada es `12,34`, esta será también la salida.

2.4. Salida de fecha y hora

En C++ no hay funciones ni clases especiales estándares para el manejo de fecha y hora, así que hay que utilizar las de `<ctime>` (`<time.h>`). Para que la localización tenga lugar, una forma fácil puede ser establecer la localidad global de C como la del objeto *locale* que se tenía definido. Ejemplo:

```
locale::global(mi_localidad);
char buf[20];
time_t t = time(0);
strftime(buf, 20, "Hoy es %A\n", localtime(&t));
cout << buf;
```

Esto imprimirá en la salida estándar, si la localización del entorno es la española: `Hoy es martes` (o el día que sea que se ejecute el programa, se entiende).

2.5. Conclusión

Se ha presentado una guía muy resumida esperando simplemente que sea útil para las tareas más básicas en cuanto a la localización. Conviene repasar el estilo de localizaciones de C en algún libro sobre la biblioteca estándar de C, y para ampliar conocimientos y darse cuenta de la enorme potencia y posibilidades, el apéndice D de la edición especial del libro *El Lenguaje de Programación C++* de Bjarne Stroustrup.