PROGRAMACIÓN DE CÓDIGOS QR

EN C++

Periféricos y Dispositivos de Interfaz Humana

Pablo Fernández Tello

Índice de Contenidos

¿Qué es un código QR? ¿Para qué sirve?

Un código QR (Quick Response code), es la evolución del código de barras. Es un módulo para almacenar información en una matriz de datos o en un código de barras bidimensional. La matriz se lee en el dispositivo móvil por un lector específico, y de forma inmediata nos lleva a una aplicación en Internet, un mapa de localización, un correo electrónico, una página web o un perfil en una red social. Fue creado en 1994 por la compañía japonesa denso wave, subsidiaria de Toyota. Presenta tres cuadrados en las esquinas que permiten detectar la posición del código al lector. El objetivo de los creadores, un equipo de dos personas dirigido por Masahiro Hara, fue que el código permitiera que su contenido se leyera a alta velocidad.

Composición y características

La estructura general de un código QR es una matriz bidimensional de módulos de dos colores contrastados, en principio blancos y negros. Hay varias versiones de códigos QR según la cantidad de módulos que forman la matriz: van desde la versión 1 (con una matriz de 21 x 21 módulos) hasta la versión 40 (con 177 x 177 módulos). Las versiones de más módulos admiten mayor cantidad de información en el código. Los códigos más extendidos para el uso del público en general suelen ser los de 25 x 25 y de 29 x 29, para capturar desde el teléfono móvil en cualquier situación (paquetes de productos, folletos de mano, tarjetas o carteles de pared).

Una característica clave de los códigos QR es su **capacidad para ser escaneados rápidamente** desde múltiples ángulos, facilitando su uso en una amplia variedad de contextos. Se utilizan en muchas aplicaciones, como marketing, pagos móviles, información de productos, entradas para eventos y más. Al escanear un código QR con un dispositivo habilitado para ello, el usuario puede acceder de forma rápida y sencilla a la información contenida en el código.

Bidimensionalidad. A diferencia de los códigos de barras tradicionales que son unidimensionales (lineales), los códigos QR son bidimensionales y contienen información en dos dimensiones (horizontal y vertical). Esto les permite almacenar más datos.

Tolerancia a errores. Los códigos QR tienen niveles de corrección de errores, lo que significa que pueden ser leídos incluso si están parcialmente dañados o sucios. Hay cuatro niveles de corrección de errores: L (bajo), M (medio), Q (alto), H (muy alto), cada uno con un porcentaje específico de recuperación.

Escalabilidad. Los códigos QR pueden variar en tamaño y complejidad, desde simples hasta altamente detallados, dependiendo de la cantidad de datos que contengan. Se pueden crear en diferentes tamaños para adaptarse a diversas necesidades y entornos.

Actualmente, el código QR es el más famoso de código de barras 2D en el mundo y su éxito se remonta a la década del 2000 en Japón, donde se convirtieron en un estándar y hasta 2010 comenzaron su expansión en los Estados Unidos y Europa, principalmente en anuncios.

Programación de códigos QR en C++

Algunos de los lenguajes más comunes para este tipo de programación es **Python**, conocido por su simplicidad y versatilidad, y cuenta con bibliotecas populares para la generación de códigos QR, como qrcode. Además, se usa ampliamente en aplicaciones web, lo que lo hace ideal para integraciones más complejas.
También lo es Java. **JavaScript** es el lenguaje del lado del cliente para la mayoría de los navegadores web. Con bibliotecas como qrcode.js es fácil generar códigos QR directamente en aplicaciones web, lo que permite una interacción rápida y sin necesidad de recursos adicionales del servidor.
Sin embargo, yo he optado por C++ ya que también es un lenguaje bastante competente para este tipo de tareas y es el que más domino.

Programar códigos QR en C++ implica generar y, en algunos casos, leer códigos QR mediante bibliotecas especializadas. En C++, existen varias bibliotecas y herramientas que permiten la creación y lectura de códigos QR como son **libqrencode** o **QREncoder**. Estas bibliotecas te permiten generar códigos QR a partir de datos como texto, URLs, información de contacto, etc.

¿Qué tiene la biblioteca?

La Biblioteca tiene muchas funcionalidades, podemos encontrar todas ellas y la descripción de las mismas en su página web oficial o en github. Sin embargo, voy a comentar algunas de las principales y de las que he usado en el ejemplo de prueba más adelante:

1. Estructura Principal del Código QR

Una **estructura QRcode** con las siguientes variables:

- **int version**: Representa la versión del código QR. Las versiones varían de 1 a 40, donde la versión determina el tamaño y la capacidad de datos del código QR.
- **int width**: El ancho del código QR en módulos (celdas).
- **Unsigned char* data**: Un puntero a un array que contiene los datos del código QR.

2. Generación de Códigos QR

La función principal para la generación de códigos QR, **QRcode_encodeString,** toma datos de entrada y los convierte en un objeto QRcode. Sus argumentos son:

- **const char* text**: El texto o datos que se convertirán en código QR.
- **int version**: La versión del código QR a generar. Si se pasa 0, la librería determina la mejor versión automáticamente.
- **QrecLevel level**: El nivel de corrección de errores. Puede ser QR_ECLEVEL_L (bajo), QR_ECLEVEL_M (medio), QR_ECLEVEL_Q (cuasi alto) o QR_ECLEVEL_H (alto).
- **QRencodeMode hint:** Sugiere el modo de codificación. Los modos posibles son: QR_MODE_NUM (modo numérico), QR_MODE_AN (modo alfanumérico), QR_MODE_8 (modo de 8 bits), QR_MODE_KANJI (para caracteres japoneses específicos)
- **int caseSensitive**: Indica si el código QR es sensible a mayúsculas y minúsculas (1 para sí, 0 para no).

Y la salida:

- **Un puntero** a una estructura QRcode que representa el código QR generado. Si ocurre un error, devuelve NULL.

3. Liberacion de Memoria

Una vez que se ha creado un código QR, es importante liberar la memoria utilizada, para ello, tenemos la función **QRcode_free**. Argumentos:

- **QRcode* qrcode**: El objeto QRcode a liberar

4. Exportación de Códigos QR

Actualmente en la librería libqrencode no existe ninguna función que nos permita exportar el QR. Habría que implementarlo a mano, por ejemplo, para que se imprima en ASCII:

```
void QRcode_print(const QRcode* qrcode) {
    if (!qrcode) {
        cerr << "QR no válido" << endl;
        return;
    }

    for (int y = 0; y < qrcode->width; ++y) {
        for (int x = 0; x < qrcode->width; ++x) {
            if (qrcode->data[y * qrcode->width + x] & 1) {
                cout << "██";
            } else {
                cout << "  ";
            }
        }
        cout << endl;
    }
}
```

Si queremos exportarlo a formato PNG por ejemplo, sería una tarea más compleja y tendríamos que ayudarnos de otras herramientas como la librería libpng.

Ejemplo

A continuación he hecho un pequeño programa a modo de ejemplo donde creo un archivo de texto llamado qrcode.txt en el que se imprime el cógido QR en formato ASCII.

Al escanearlo nos lleva a la página web de Youtube.

```cpp
#include <qrencode.h>
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    const char* text = "https://www.youtube.com";

    QRcode* qrcode = QRcode_encodeString(text, 0, QR_ECLEVEL_L, QR_MODE_8, 1);

    if (!qrcode) {
        cerr << "Error al generar el código QR" << endl;
        return 1;
    }

    ofstream out("qrcode.txt");
    int width = qrcode->width;
    for (int y = 0; y < width; ++y) {
        for (int x = 0; x < width; ++x) {
            if (qrcode->data[y * width + x] & 1) {
                out << "██";
            } else {
                out << "  ";
            }
        }
        out << endl;
    }

    QRcode_free(qrcode);
    cout << "Código QR generado en qrcode.txt" << endl;

    return 0;
}
```

La mayor parte del código queda explicada con las funciones descritas anteriormente. El único punto a destacar es cómo se imprime el código QR generado por la función QRcode_encodeString.

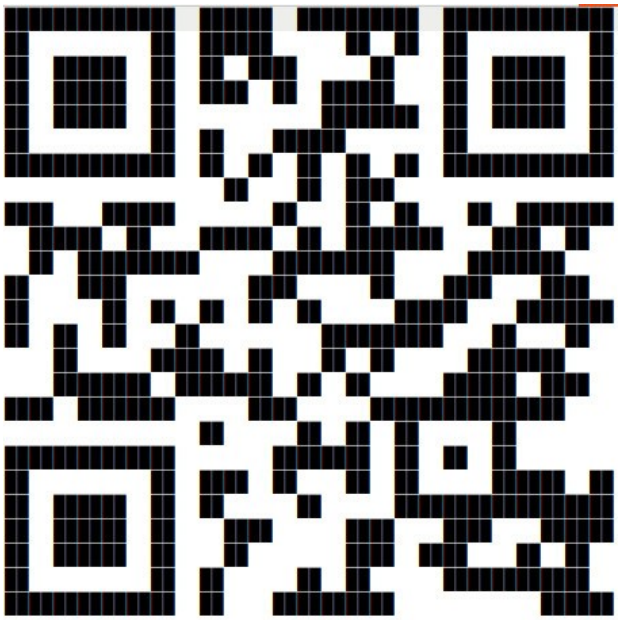
Con un bucle anidado recorremos las filas y las columnas del QR y para cada combinación de y y x, verificamos si el módulo correspondiente es negro o blanco:

- qrcode→data es un array unidimensional que representa la matriz del código QR. Para obtener el valor del módulo en la posición (y,x), calculamos el índice como y*qrcode→width+x.
- Si el valor en esta posición tiene el bit menos significativo como 1, significa que es un módulo negro. En este caso, imprime un bloque que visualmente representa el módulo negro.
- Si el valor tiene el bit menos significativo como 0, es un módulo blanco. Imprime dos espacios para representar el módulo blanco.

Compilamos el código enlazando con la biblioteca libqrencode de la siguiente manera:

g++ qr.cpp -o qr -lqrencode

Ejecutamos el programa: ./qr y mostramos el contenido del archivo: cat qrcode.txt



Podríamos haber puesto cualquier URL únicamente cambiando el contenido de la variable text del código.

Por último concluir diciendo que, como he mencionado anteriormente, libqrencode cuenta con funcionalidades más complejas para la generación de códigos QR y nosotros solo hemos visto algunas de las más comunes. Algunas de estas funcionalidades son:

- **Soporte para archivos SVG y otros formatos gráficos:** Permite generar el código QR en diferentes formatos gráficos, como SVG, lo cual facilita su integración en aplicaciones web y otros medios.
- **Soporte para salida binario y hexadecimal:** Esto es útil para guardar el código QR como datos binarios o hexadecimal para procesamientos adicionales.
- **Incrustación de imágenes y logotipos:** Aunque libqrencode no soporta directamente la incrustación de imágenes o logotipos en los códigos QR, puede ser utilizado como base para lograrlo, en combinación con otras herramientas gráficas.