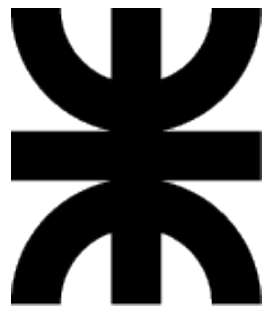
Informática II

Programación gráfica con Qt
Puerto serie

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Puerto serie con Qt

La biblioteca Qt brinda diferentes funcionalidades relacionadas al puerto serie, como: configuración, operaciones de E/S y obtención y configuración de las señales de control de los pines RS-232, etc. (documentación)

Puerto serie con Qt

La biblioteca Qt brinda diferentes funcionalidades relacionadas al puerto serie, como: configuración, operaciones de E/S y obtención y configuración de las señales de control de los pines RS-232, etc. (documentación)

En los proyectos basados en `CMake` se necesita incluir el módulo `SerialPort`. (Instalar paquete de Ubuntu `qt6-serialport-dev`)

Puerto serie con Qt

La biblioteca Qt brinda diferentes funcionalidades relacionadas al puerto serie, como: configuración, operaciones de E/S y obtención y configuración de las señales de control de los pines RS-232, etc. (documentación)

En los proyectos basados en `CMake` se necesita incluir el módulo `SerialPort`. (Instalar paquete de Ubuntu `qt6-serialport-dev`)

Las clases C++ para la programación del puerto serie son:

- `QSerialPort`: funcionalidad para acceder al puerto.
- `QSerialPortInfo`: brinda información de los puertos existentes.

Clase QSerialPortInfo

Archivo port_info.cpp

```
#include <QCoreApplication>
#include <QTextStream>
#include <QSerialPortInfo>
#include <QList>

int main(int argc, char *argv[])
{
    QCoreApplication app(argc, argv);
    QTextStream out(stdout);

    QList<QSerialPortInfo> port_info = QSerialPortInfo::availablePorts();

    out << "# of serial ports: " << port_info.count() << Qt::endl;
    for(int i = 0; i < port_info.size(); i++)
    {
      // mostrar info
    }

    return 0;
}
```

Clase QSerialPortInfo

Archivo CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)

project(port_info VERSION 1.0.0 LANGUAGES CXX)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 COMPONENTS Core SerialPort REQUIRED)

add_executable(port_info port_info.cpp)
target_link_libraries(port_info PRIVATE Qt6::Core Qt6::SerialPort)
```

Clase QSerialPortInfo

Archivo CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)

project(port_info VERSION 1.0.0 LANGUAGES CXX)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 COMPONENTS Core SerialPort REQUIRED)

add_executable(port_info port_info.cpp)
target_link_libraries(port_info PRIVATE Qt6::Core Qt6::SerialPort)
```

Ejecutar el programa con una placa Arduino UNO conectada a la PC, se detecta 33 puertos, el correspondiente a la placa Arduino es:

```
> ./port_info
# of serial ports: 33

Port: ttyUSB0
Location: /dev/ttyUSB0
Description: USB2.0-Serial
Manufacturer: 1a86
Serial number:
Vendor Identifier: 1a86
Product Identifier: 7523
```

(Plantilla de clases)

El ejemplo anterior utiliza *Plantilla de clases*, `QList`, la cual se define como:

```
template <typename T>
class QList {
  ...
};
```

(Plantilla de clases)

El ejemplo anterior utiliza *Plantilla de clases*, `QList`, la cual se define como:

```
template <typename T>
class QList {
  ...
};
```

(documentación)

(Plantilla de clases)

El ejemplo anterior utiliza *Plantilla de clases*, `QList`, la cual se define como:

```
template <typename T>
class QList {
  ...
};
```

(documentación)

Otro ejemplo de plantilla es la clase `vector` de la biblioteca estándar de C++:

```
template <typename T>
class vector {
  ...
};
```

(Plantilla de clases)

El ejemplo anterior utiliza *Plantilla de clases*, `QList`, la cual se define como:

```
template <typename T>
class QList {
  ...
};
```

(documentación)

Otro ejemplo de plantilla es la clase `vector` de la biblioteca estándar de C++:

```
template <typename T>
class vector {
  ...
};
```

(documentación)

(Plantilla de clases)

Se puede declarar una clase `Arreglo` mediante plantilla como:

```
template< typename T >
class Arreglo {
    public:
        Arreglo(int = 10 ) // constructor predeterminado

        ~Arreglo() { delete [] ptrArreglo; } // destructor

        // funciones miembros

    private:
        int tamanio; // nro de elementos en la arreglo
        T *ptrArreglo; // apuntador a la arreglo
};
```

Clase `QSerialPort` (abrir y cerrar)

Dado un objeto `port` de la clase `QSerialPort`.

Clase `QSerialPort` (abrir y cerrar)

Dado un objeto `port` de la clase `QSerialPort`.

Configurar el puerto:

```
port.setPortName("ttyUSB0"); // Harcodeado :(
port.setBaudRate(QSerialPort::Baud9600);
port.setDataBits(QSerialPort::Data8);
port.setParity(QSerialPort::NoParity);
port.setStopBits(QSerialPort::OneStop);
port.setFlowControl(QSerialPort::NoFlowControl);
```

Clase `QSerialPort` (abrir y cerrar)

Dado un objeto `port` de la clase `QSerialPort`.

Configurar el puerto:

```
port.setPortName("ttyUSB0"); // Harcodeado :(
port.setBaudRate(QSerialPort::Baud9600);
port.setDataBits(QSerialPort::Data8);
port.setParity(QSerialPort::NoParity);
port.setStopBits(QSerialPort::OneStop);
port.setFlowControl(QSerialPort::NoFlowControl);
```

Abrir el puerto:

```
port.open(IODevice::ReadWrite);
```

Clase `QSerialPort` (abrir y cerrar)

Dado un objeto `port` de la clase `QSerialPort`.

Configurar el puerto:

```
port.setPortName("ttyUSB0"); // Harcodeado :(
port.setBaudRate(QSerialPort::Baud9600);
port.setDataBits(QSerialPort::Data8);
port.setParity(QSerialPort::NoParity);
port.setStopBits(QSerialPort::OneStop);
port.setFlowControl(QSerialPort::NoFlowControl);
```

Abrir el puerto:

```
port.open(IODevice::ReadWrite);
```

Cerrar el puerto:

```
if(port->isOpen())
  port->close();
```

Clase `QSerialPort` (escribir y leer)

Ver ejemplos