# Section 1: Introduction

## 8. QT overview

- QT là 1 thư viện trong C++

- 3 phần chính : core, wigets, QML

Text

Description automatically generated

## 9. Course Overview

Diagram

Description automatically generated

# Section 2: Introduce to C++

## 35. C++ QT build process

Diagram

Description automatically generated

QT MOC: QT meta ofject compiler

Graphical user interface, text, application, email

Description automatically generated

QT tool

A person holding a pen

Description automatically generated

Test is source folder and above is build folder

# Section 8: Class

OOP

1. Các đặc tính cơ bản của OOP

* **Tính đóng gói (Encapsulation)**
* **Tính kế thừa (Inheritance)**
* **Tính đa hình (Polymorphism )**
* **Tính trừu tượng(Abstraction)**

You cannot call signal directly

8.9 Qobject

The **Q\_OBJECT macro** must appear in the private section of a class definition that declares its own signals and slots or that uses other services provided by Qt's meta-object system.

Example;

#include <QObject>

class Counter : public [QObject](https://doc.qt.io/qt-6/qobject.html#QObject)

{

Q\_OBJECT

public:

Counter() { m\_value = 0; }

int value() const { return m\_value; }

public slots:

void setValue(int value);

signals:

void valueChanged(int newValue);

private:

int m\_value;

};

**The Meta-Object System**

**https://doc.qt.io/qt-6/metaobjects.html**

Qt's meta-object system provides the signals and slots mechanism for inter-object communication, run-time type information, and the dynamic property system.

The meta-object system is **based on** **three** things:

* The **QObject class** provides a base class for objects that can take advantage of the meta-object system.
* The **Q\_OBJECT macro** inside the private section of the class declaration is used to enable meta-object features, such as dynamic properties, signals, and slots.
* The **Meta-Object Compiler (moc**) supplies each QObject subclass with the necessary code to implement meta-object features.

**Meta-Object Compiler (moc)**

**https://doc.qt.io/qt-6/moc.html**

The moc tool reads a C++ source file. If it finds one or more class declarations that contain the Q\_OBJECT macro, it **produces** another C++ source file which contains the meta-object code for each of those classes.

This **generated source** file is either #include'd into the class's source file or, more usually, compiled and linked with the class's implementation. Lúc này parameter type sẽ được xem như là class type

# Explicit

Trong C++, 1 constructor chỉ có required parameter được xem là **implicit conversion function**.

class Human

{

public:

    Human()         { qInfo() << "Hello"; }

    explicit Human(int age) { qInfo() << age; }

    void GetAge() {qInfo() << this->age;}

private:

    int age;

    QString name;

};

int main(int argc, char \*argv[])

{

    QCoreApplication a(argc, argv);

    Human h1;

    /\*implicit conversion\*/

    Human h2 = 10;                  //10 is parameter of constructor

    Human h3(15);                   //15 is parameter of constructor

    h3.GetAge();

    return a.exec();

}

Việc chuyển đồi này có thể là tốt hay xấu tùy mục đích sử dụng.

Để ngăn việc chuyển đổi này xảy ra có thể thêm tiền tố explicit trước constructor.

explicit Human(int age) { qInfo() << age; }

Signal And Slot