Fabio Falzoi
Tech lead

C++/Qt Crash Course

ISIS Gobetti Volta
GENNAIO 2021

develer

LEZIONE 2 - PROGRAMMA

- Richiami di C++
- Classi
- QString
- QDateTime

Hello World

- *main* è l'entry-point del programma
- *qDebug()* consente lo stream di testo su output

```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    qDebug() << "Hello World";

    return a.exec();
}
```


develer

Basic Types

- Tipi base: char, int, float, bool
- Ciascun tipo definisce un intervallo di valori rappresentabili e un insieme di operazioni che si possono compiere su di esso

```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    char c = 'a';
    int n = 10;
    float f = 3.14;
    bool b = true;

    qDebug() << "char: " << c;
    qDebug() << "int: " << n;
    qDebug() << "float: " << f;
    qDebug() << "bool: " << b;

    return a.exec();
}
```


develer

Operators

- Alcuni operatori aritmetici e logici comuni

```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);

  qDebug() << 10 + 20;
  qDebug() << 20 - 10;
  qDebug() << 10 * 10;
  qDebug() << 40 / 10;
  qDebug() << 40 % 10;

  qDebug() << (40 < 10);
  qDebug() << (40 >= 10);
  qDebug() << (40 == 10);
  qDebug() << (40 != 10);

  qDebug() << (true && false);
  qDebug() << (true || false);

  return a.exec();
}
```

Conditional Statements

- la keyword *if* permette di inserire un'istruzione condizionale
- Se la condizione è vera, verrà eseguito il blocco che segue *if*, altrimenti verrà eseguito quello che segue *else*
- *else* è facoltativo!

```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    bool b = true;

    if (b == true) {
        qDebug() << "b is true!";
    } else {
        qDebug() << "b is false";
    }

    return a.exec();
}
```

Loop

- la keyword *for* permette di comporre un ciclo
- Un ciclo è un blocco di istruzioni che vengono ripetute finché una condizione rimane vera

```
#include <QCoreApplication>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    // loop until the i variable is less than 10
    for (int i = 0; i < 10; i++) {
        qDebug() << "i value is: " << i;
    }

    return a.exec();
}
```


develer

Functions

- Una funzione è un blocco di codice riutilizzabile, che può essere invocato ripetutamente
- Si compone di una intestazione che comprende il tipo di ritorno, il nome e la lista dei parametri in ingresso

```
#include <QCoreApplication>
#include <QDebug>

// func1 returns nothing and takes nothing
void func1() {
   qDebug() << "func1 has been called!";
}

// func2 returns a boolean and takes nothing
bool func2() {
   return true;
}

// func3 returns an int and takes two ints
int func3(int a, int b) {
   return a + b;
}

int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);

   func1();

   bool res2;
   res2 = func2();
   qDebug() << res2;

   int res3;
   res3 = func3(5, 7);
   qDebug() << res3;

   return a.exec();
}
```


develer

Prime Numbers

1. Scrivete un programma per verificare se un intero, compreso fra 1 e 100 (inclusi) sia o meno un numero primo
2. Un numero primo è un numero intero positivo che ha solo 2 divisori distinti: 1 e se stesso.
3. Scrivete in output “Primo” oppure “Composto” a seconda del risultato.

develer

Classes - Header file

- Una classe consente di aggiungere un tipo personalizzato al linguaggio
- Ciascuna istanza (*oggetto*) della classe avrà i membri e i metodi che specifichiamo noi

```
#ifndef POINT_H
#define POINT_H


class Point
{
   // member values
   float x, y;

public:
   // constructors
   Point();
   Point(float x, float y);

   // setters
   void setX(float a);
   void setY(float b);

   // getters
   float getX();
   float getY();
};

#endif // POINT_H
```

Classes - Source file

- Una classe consente di aggiungere un tipo personalizzato al linguaggio
- Ciascuna istanza (*oggetto*) della classe avrà i membri e i metodi che specifichiamo noi

```
#include "point.h"

Point::Point()
{
   x = 0;
   y = 0;
}

Point::Point(float a, float b) {
   x = a;
   y = b;
}

void Point::setX(float a) {
   x = a;
}

void Point::setY(float b) {
   y = b;
}

float Point::getX() {
   return x;
}

float Point::getY() {
   return y;
}
```

Classes - How to use it

- Possiamo utilizzare la nostra classe istanziando un oggetto
- Quando istanziato, l'oggetto viene valorizzato mediante uno dei costruttori definiti
- Da qui in poi, possiamo richiamare i metodi, definiti per quella classe, sull'oggetto

```
#include <QCoreApplication>
#include <QDebug>

#include "point.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    Point p1;

    qDebug() << p1.getX();
    qDebug() << p1.getY();

    Point p2(5.0, 7.5);

    qDebug() << p2.getX();
    qDebug() << p2.getY();

    return a.exec();
}
```

Prime Numbers

1. Scrivete un programma che contenga la classe Point vista prima
2. Scrivete una funzione che, presi in input 2 argomenti di tipo Point, calcoli la distanza fra i 2 punti

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

develer

QString

- QString class
- Contiene una sequenza di caratteri Unicode
- Fornisce molti metodi per manipolare la stringa contenuta
- *append*, *prepend*

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QString str = "testo";

    qDebug() << str;

    str.prepend("Questo è un ");

    qDebug() << str;

    str.append(" di prova");

    qDebug() << str;

    return a.exec();
}
```

QString

- QString class
- Contiene una sequenza di caratteri Unicode
- Fornisce molti metodi per manipolare la stringa contenuta
- *toLower*, *toUpper*

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QString str = "testo";

    qDebug() << str.toUpper();

    qDebug() << str.toLower();

    return a.exec();
}
```

QString

- QString class
- Contiene una sequenza di caratteri Unicode
- Fornisce molti metodi per manipolare la stringa contenuta
- *length*

```
#include <QCoreApplication>
#include <QDebug>
#include <QString>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QString str = "testo";

    qDebug() << str.length();

    return a.exec();
}
```

QString

- Accesso e modifica dei caratteri di una stringa

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QString str = "testo";

    qDebug() << str[0];
    qDebug() << str[1];

    str[3] = 'Z';

    qDebug() << str;

    return a.exec();
}
```

QString

- Ciclo sui singoli caratteri della stringa

```
#include <QCoreApplication>
#include <QDebug>
#include <QString>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QString str = "There are many stars.";

    for (QChar qc: str) {
        qDebug() << qc << " ";
    }
    qDebug() << "";

    for (int i = 0; i < str.length(); ++i) {
        qDebug() << str.at(i) << " ";
    }
    qDebug() << "";

    return a.exec();
}
```

Characters frequency

1. Contare il numero di vocali contenuto in una stringa e scrivere in output il risultato

develer

Alternating case

1. Data una stringa, scrivere in output la stessa stringa, con i caratteri alternativamente maiuscolo e minuscolo

develer

QDateTime

- QDateTime permette di manipolare date e timestamp

```
#include <QCoreApplication>
#include <QDebug>
#include <QDateTime>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QDateTime dt = QDateTime::currentDateTime();

    qDebug() << dt.toString();

    qDebug() << dt.toString("hh:mm:ss.zzz");

    return a.exec();
}
```

Leap year

1. Dato un intero che rappresenta un anno, determinare se si tratta di un anno bisestile o meno

develer

CONTACTS

Fabio Falzoi

fabio@develer.com

Twitter: @Pippolo84

www.develer.com