qCharts

Progetto PAO-2013

1 Scopo

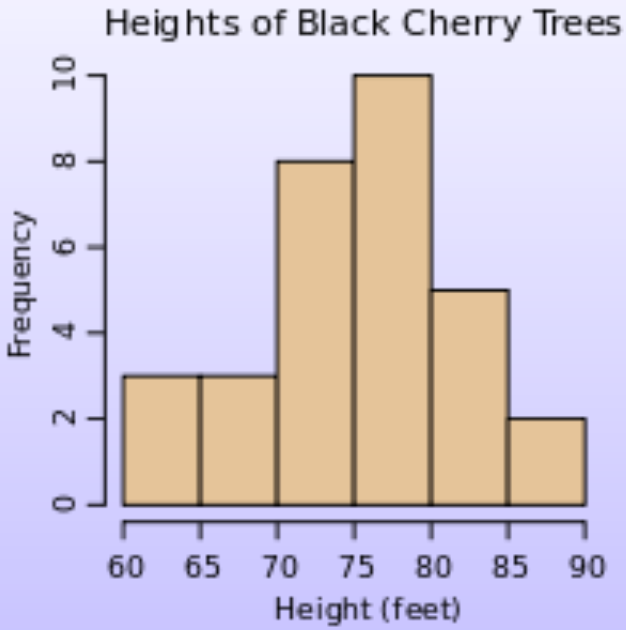
Lo scopo del progetto qCharts è lo sviluppo in C++/Qt di un sistema per creare, archiviare, modificare e visualizzare *charts*, ovvero diagrammi di dati.

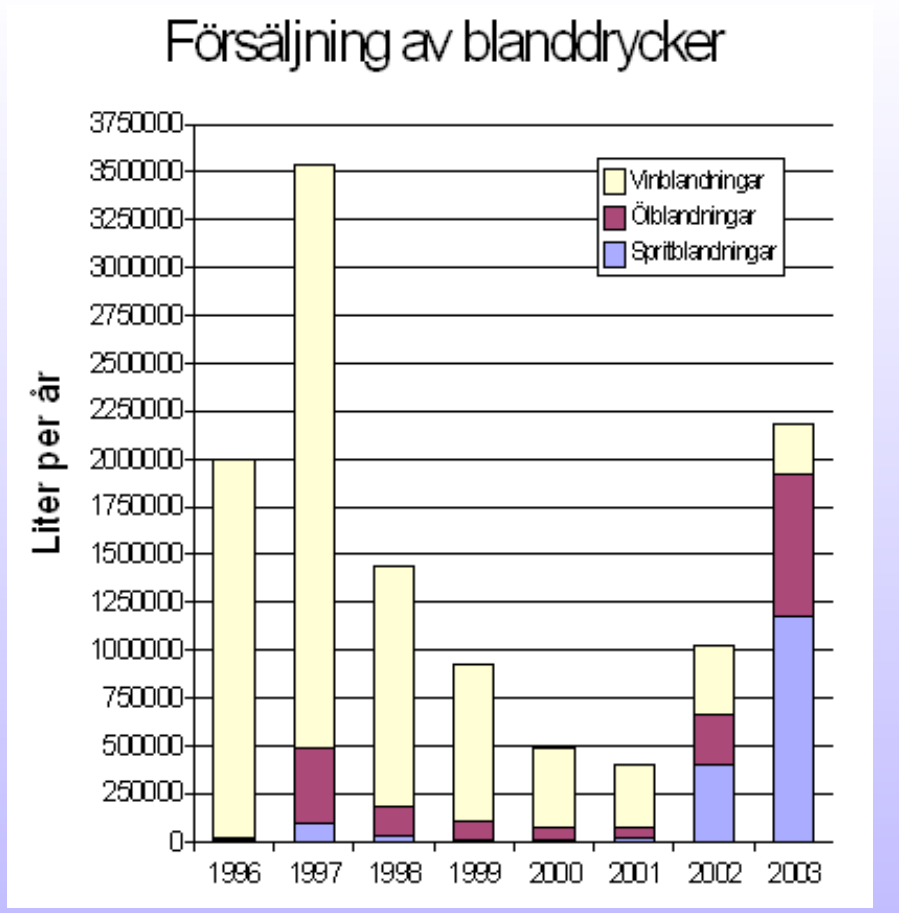
Un chart è una rappresentazione grafica di una collezione di dati tipicamente numerici, in cui i dati sono rappresentati mediante simboli: ad esempio, come rettangoli in un diagramma a barre (detto anche istogramma), linee in un diagramma a linee, settori di un cerchio in un diagramma a torta. Si rimanda alla voce Chart di Wikipedia/English per una descrizione dettagliata dei charts e per un ampio campionario delle tipologie più comuni di charts.

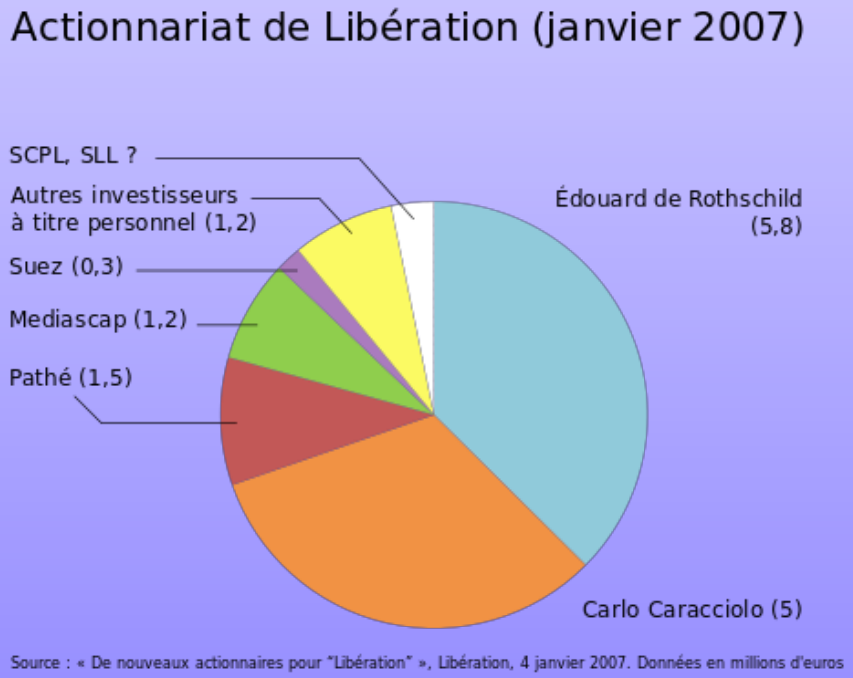
La specifica **dettagliata** di quali funzionalità il progetto qCharts debba fornire è lasciata in massima parte a libera scelta dello studente, che può trarre ispirazione dal web, dagli innumerevoli charting software esistenti, dai propri interessi e dalla fantasia. La specifica **minimale** delle funzionalità richieste a qCharts consiste in:

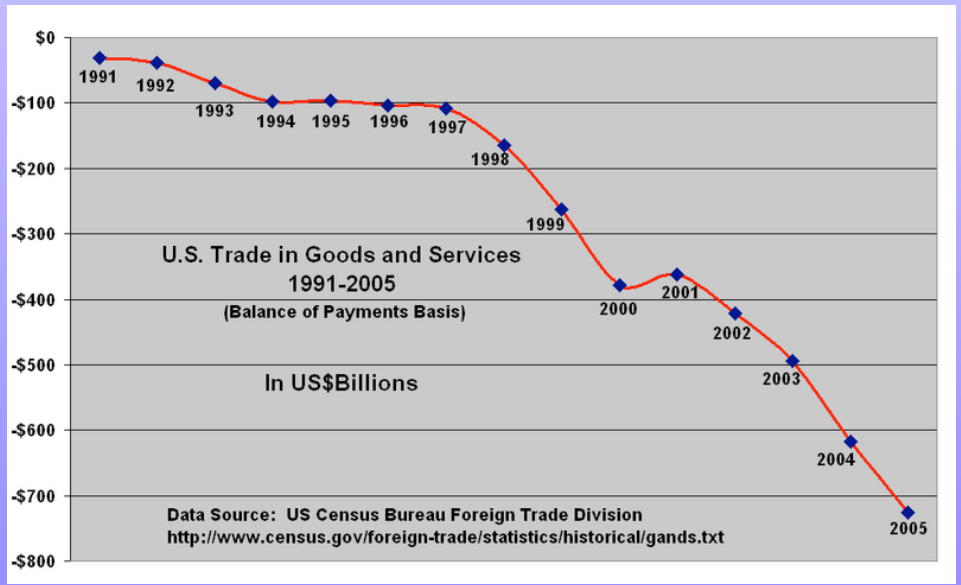
1. Creazione e modifica dei dati numerici di un chart
2. Archiviazione (e recupero) su file dei dati numerici di un chart
3. Visualizzazione grafica di un chart con possibilità di scegliere tra almeno tre tipologie diverse di chart

en.wikipedia.org/wiki/Chart









Interfaccia Grafica

Si richiede di sviluppare una GUI usando la libreria Qt che permetta all'utente di usufruire agevolmente delle funzionalità di qCharts mediante tale GUI. In particolare, la GUI dovrà permettere la visualizzazione delle varie tipologie di chart rese disponibili da qCharts. È possibile progettare alcune opzioni di qCharts che permettano di specializzare i chart per alcune situazioni di interesse. Alcuni semplici esempi di specializzazione (attuali in questo periodo) potrebbero essere: andamento spread/borse/monete, sondaggi di gradimento di partiti/personaggi politici, classifiche di eventi sportivi, market shares di prodotti come PC/cellulari/tablet, etc. (si faccia ricorso alla fantasia ed ai propri interessi).

Il campione di codice della GUI che accompagna e complementa il presente documento fornisce una intelaiatura minimale a fini esemplificativi per lo sviluppo della GUI. Come noto, la libreria Qt è dotata di una documentazione completa e precisa che sarà la principale guida di riferimento nello sviluppo della GUI, oltre ad offrire i tool QtCreator e QtDesigner. La libreria Qt offre una moltitudine di classi e metodi per lo sviluppo di GUI curate, dettagliate e user-friendly. Sarà apprezzato l'uso di funzionalità di Qt diverse da quelle minimali/standard.

2 Valutazione del Progetto

Un buon progetto dovrà essere sviluppato seguendo i principi fondamentali della programmazione orientata agli oggetti, anche per quanto concerne lo sviluppo dell'interfaccia grafica. La valutazione del progetto prenderà in considerazione i seguenti criteri:

1. **Correttezza**: il progetto deve compilare e funzionare correttamente, e raggiungere correttamente e pienamente gli scopi previsti.
2. **Orientazione agli oggetti**: (A) progettazione ad oggetti, (B) modularità (in particolare, massima separazione tra il codice logico del progetto ed il codice della GUI del progetto), (C) estensibilità e (D) qualità del codice sviluppato.
3. **Quantità e qualità**: quante e quali funzionalità il progetto rende disponibili, e la loro qualità.
4. **GUI**: utilizzo corretto della libreria Qt, qualità ed usabilità della GUI.

```
class Dati { // dati numerici interi generati casualmente
private:
  int size_;
  int* dati;
public:
  Dati(int n): size_(n), dati(new int[n]) {
    for(int i=0; i<size_; ++i) dati[i]= rand()%100; // [0,99]
  }
  int& operator[](int k) const {return dati[k];}
  int size() const {return size_;}
  ~Dati() {delete[] dati;}
};
```

```
class Valori { // dati normalizzati
private:
  int size_;
  int* v;
  int max_;
public:
  Valori(const Dati& d, int m): size_(d.size()), v(new int[size_]), max_(m) {
    int max=d[0];
    for(int i=1;i<size_;i++) if(max<d[i]) max = d[i];
    for(int i=0;i<size_;i++) v[i] = max_*d[i]/max;
  }
  int& operator[](int k) const {return v[k];}
  int max() const {return max_;}
  int size() const {return size_;}
  ~Valori() {delete[] v;}
};
```

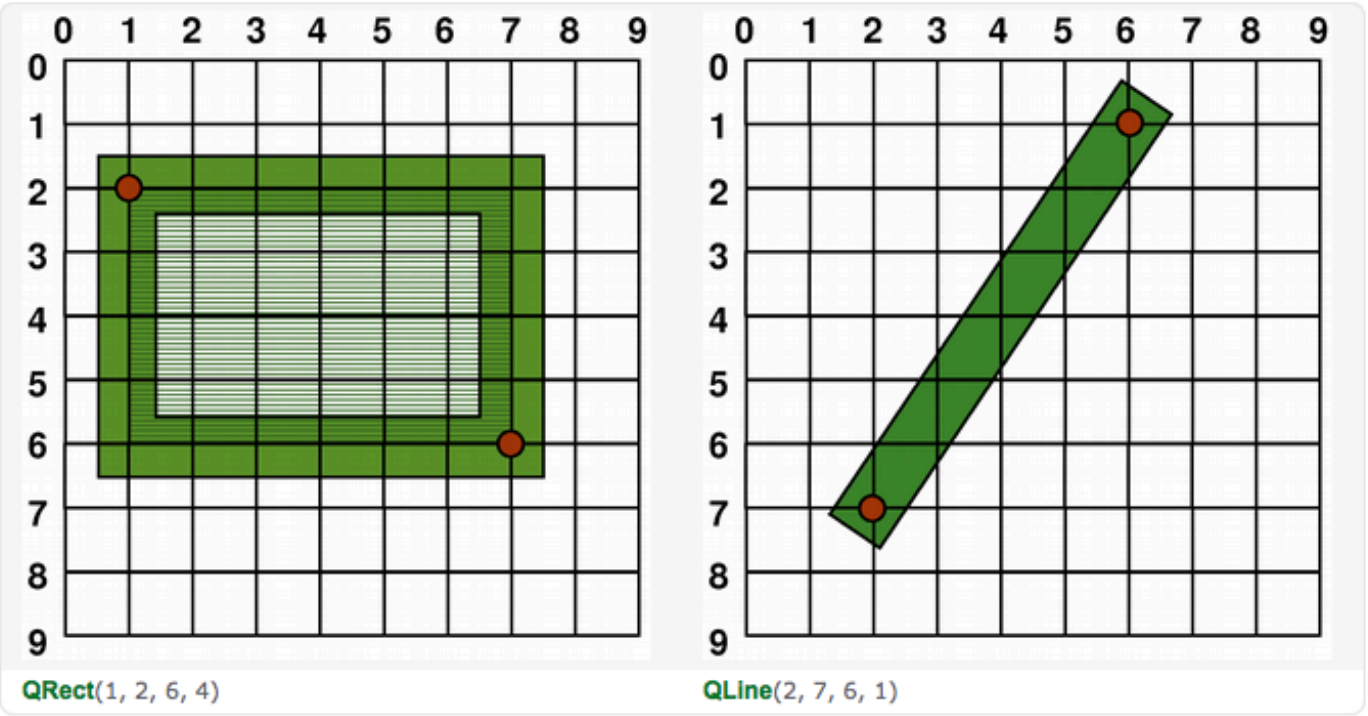
```
#include <QApplication>
#include "MyWidget.h"

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  MyWidget widget;
  widget.show();
  return app.exec();
}
```

```
// finestra Widget contenitore
class MyWidget : public QWidget {
 private:
  QPushButton* button;
  MyCanvas* canvas;
 public:
  MyWidget(QWidget* parent=0);
};
```

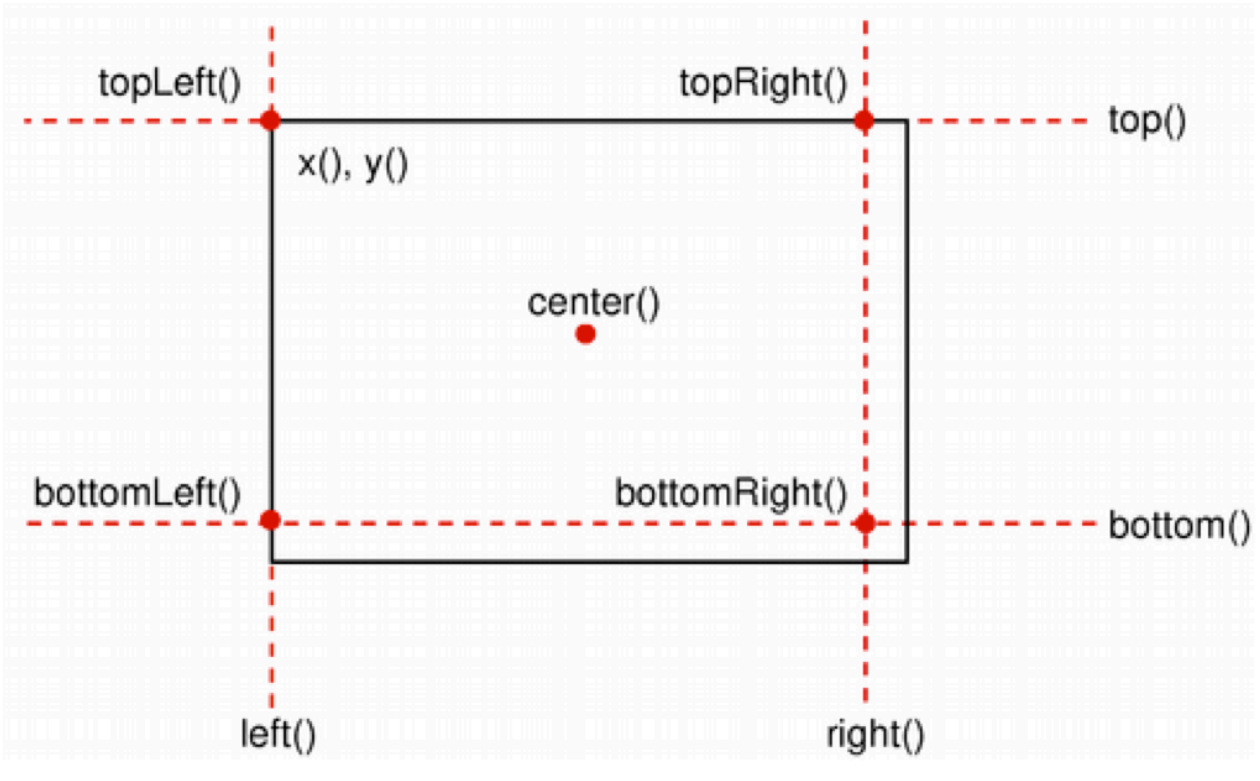
```
MyWidget::MyWidget(QWidget* parent) : QWidget(parent) {
  setFixedSize(QSize(420, 230));
  button = new QPushButton(tr("Draw"), this);
  button->setFont(QFont("Times", 18, QFont::Bold));
  canvas = new MyCanvas(this);
  QGridLayout* grid = new QGridLayout;
  grid->addWidget(button); grid->addWidget(canvas);
  setLayout(grid);
  QObject::connect(button,SIGNAL(clicked()),canvas,SLOT(draw()));
}
```

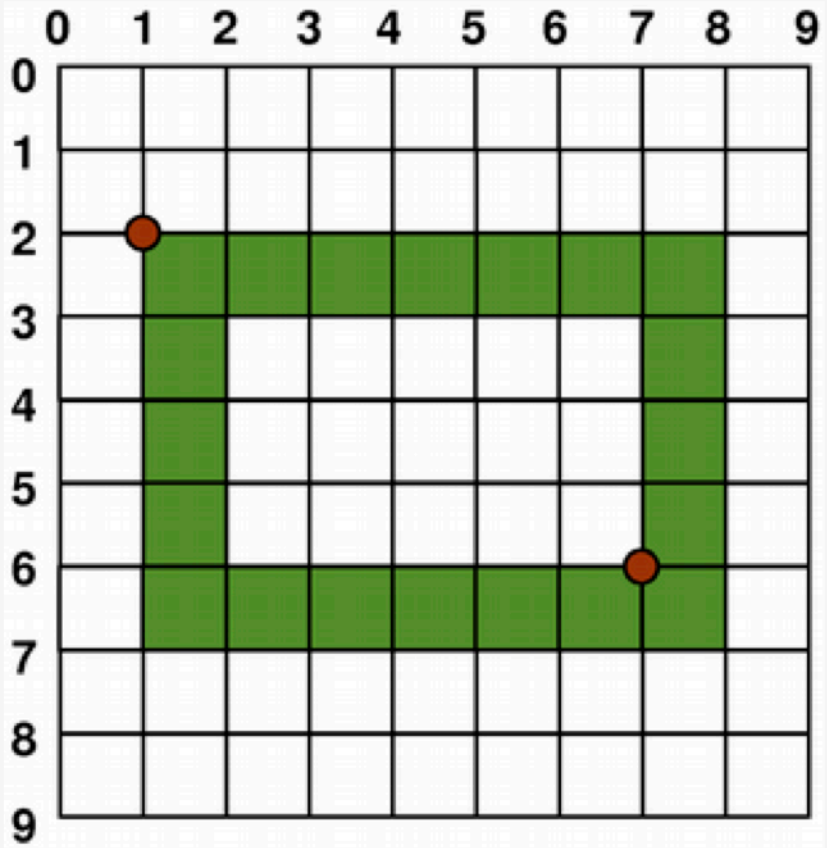
```
// Widget canvas su cui fare painting
class MyCanvas : public QWidget {
  Q_OBJECT
public:
  MyCanvas (QWidget* parent=0);
public slots:
  void draw();
protected:
  void paintEvent(QPaintEvent*);
};
```

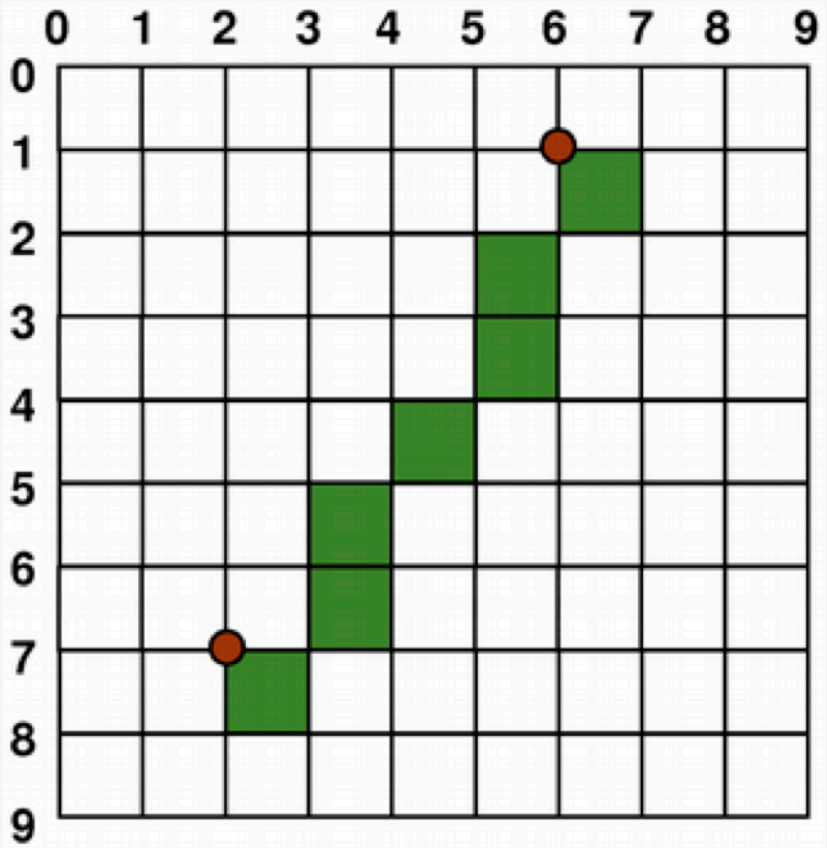
```
MyCanvas::MyCanvas (QWidget* parent) : QWidget(parent) {}

void MyCanvas::draw() {
  update();
}

void MyCanvas::paintEvent(QPaintEvent*) {
  QPainter p(this); // low-level painting object p
  Dati dati(10); // 10 dati random
  int max=100; // massimo dato normalizzata
  Valori val(dati,max); // dati normalizzati
  if(dati[0]%2 == 0) BarChart(p,val,dati);
  else LineChart(p,val,dati);
}
```



Sistema di coordinate di Qt

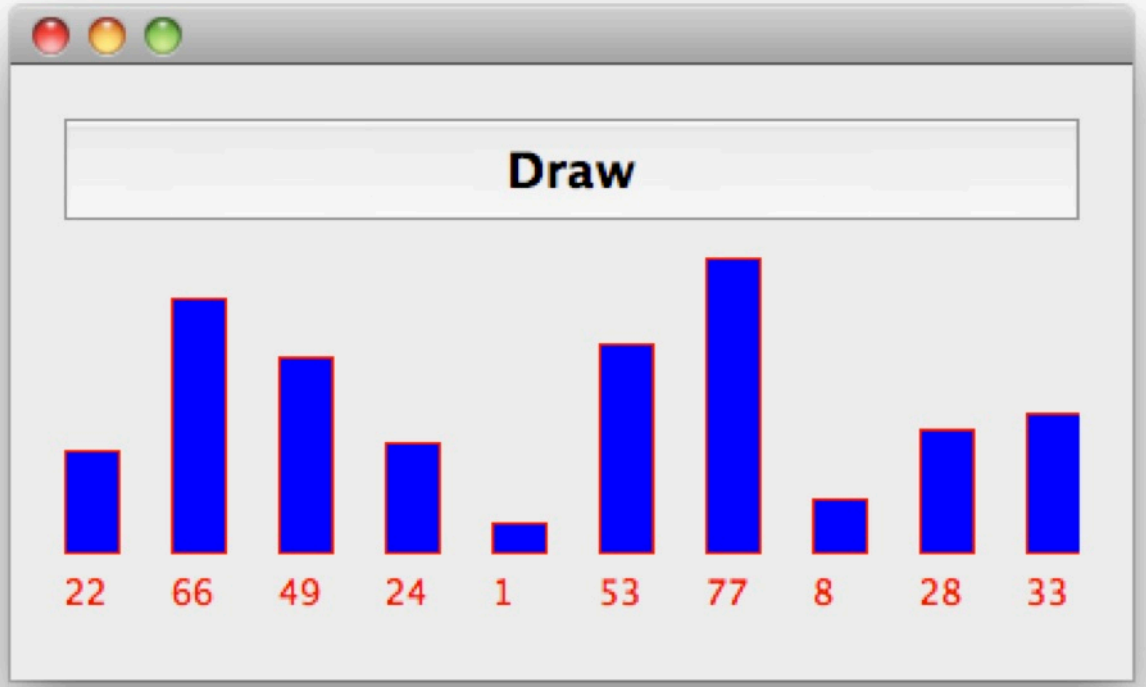



0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

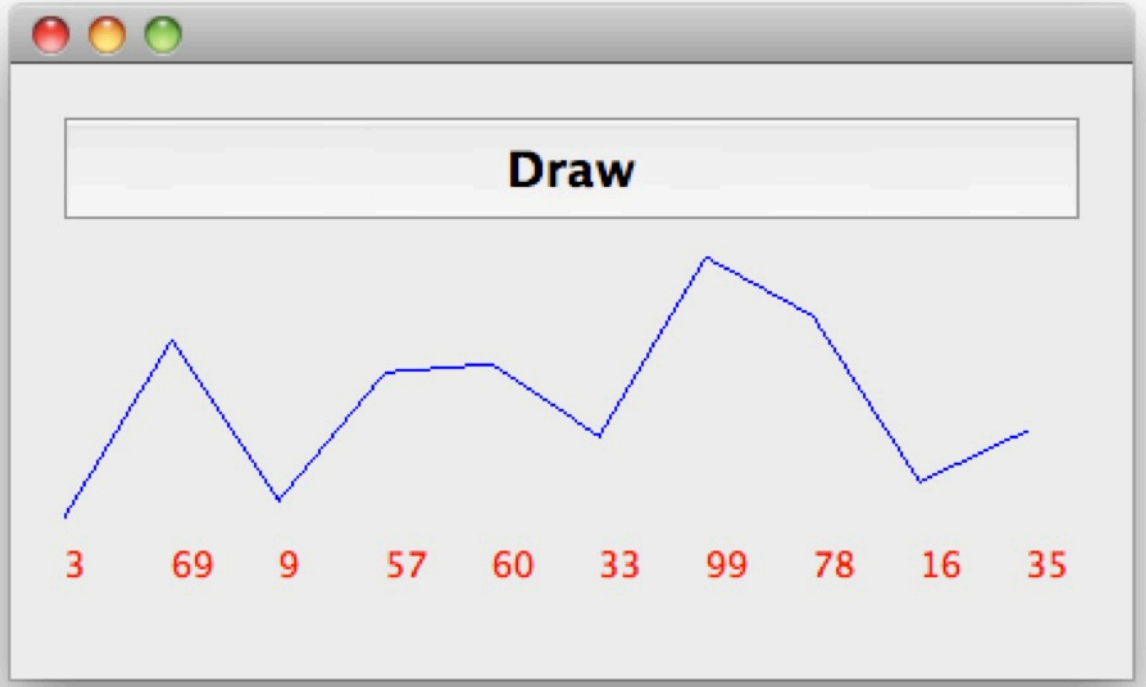

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

```
class Chart { // punti in coordinate Qt
protected:
  QPoint* points;
  int xGap;
public:
  Chart(const Valori& val, int xg=40):
    points(new QPoint[val.size()]), xGap(xg) {
    for(int i=0; i<val.size(); ++i)
      points[i] = QPoint(i*xGap,val.max()-val[i]);
  }
  virtual ~Chart() {delete[] points;}
};
```

```
class BarChart: public Chart { // diagramma a barre
private:
  int minHeight;
  int width;
  int textGap;
public:
  BarChart(QPainter& p, const Valori& val, const Dati& dati, int mh=10, int w=20, int tg=20):
    Chart(val), minHeight(mh), width(w), textGap(tg) {
    for(int i=0;i<val.size();i++) {
      QRect r(points[i],QSize(width,minHeight+val[i]));
      p.setPen(QColor(Qt::red)); p.drawRect(r);
      QRect r2(points[i]+QPoint(1,1),QSize(width-1,minHeight+val[i]-1));
      p.fillRect(r2, QBrush(QColor(Qt::blue),Qt::SolidPattern));
      p.setPen(QColor(Qt::red));
      p.drawText(points[i]+QPoint(0,minHeight+val[i]+textGap),QString(QString::number(dati[i])));
    }
  }
};
```


Draw
22
66
49
24
1
53
77
8
28
33

```
class LineChart: public Chart { // diagramma a linea
private:
  int textGap;
public:
  LineChart(QPainter& p, const Valori& val, const Dati& dati, int tg=20):
    Chart(val), textGap(tg) {
    int i = 0;
    for(;i<val.size()-1;i++) {
      p.setPen(QColor(Qt::blue));
      p.drawLine(points[i],points[i+1]);
      p.setPen(QColor(Qt::red));
      p.drawText(points[i]+QPoint(0,val[i]+textGap),QString(QString::number(dati[i])));
    }
    p.drawText(points[i]+QPoint(0,val[i]+textGap),QString(QString::number(dati[i])));
  }
};
```


Draw
3
69
9
57
60
33
99
78
16
35