UEMF
الجامعة الأورومتوسطية بفاس
EUROMED UNIVERSITY OF FES
UNIVERSITÉ EUROMED DE FÈS

EIDIA
École d'Ingénierie Digitale et d'Intelligence Artificielle

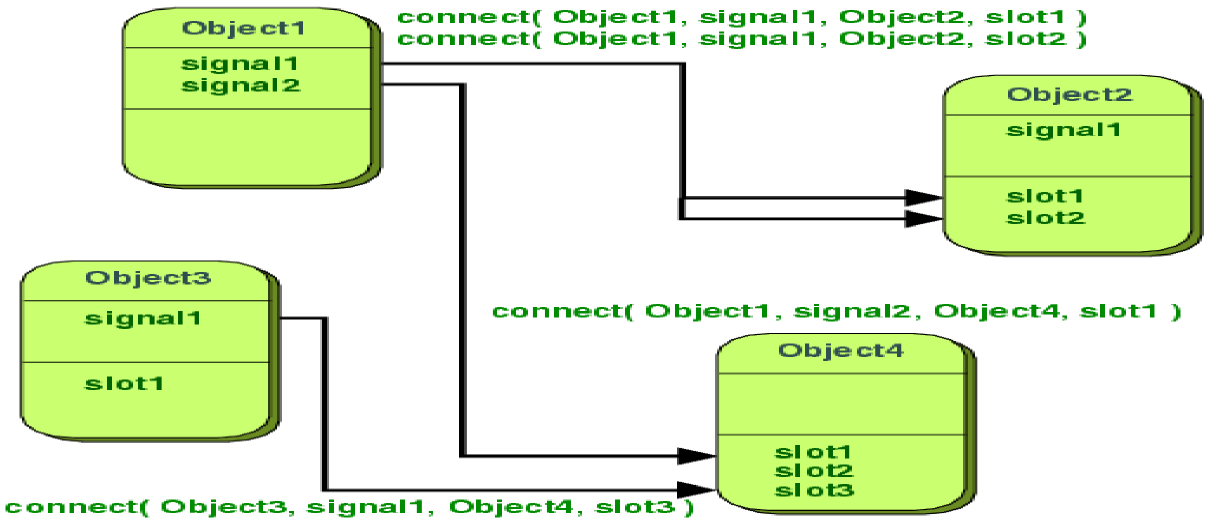
Signals and Slots

Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. Signals and slots are made possible by Qt's meta-object system.

Introduction

In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another. For example, if a user clicks a Close button, we probably want the window's close() function to be called.

Other toolkits achieve this kind of communication using callbacks. A callback is a pointer to a function, so if you want a processing function to notify you about some event you pass a pointer to another function (the callback) to the processing function. The processing function then calls the callback when appropriate. While successful frameworks using this method do exist, callbacks can be unintuitive and may suffer from problems in ensuring the type-correctness of callback arguments.



Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

Plan :

1.Traffic Light

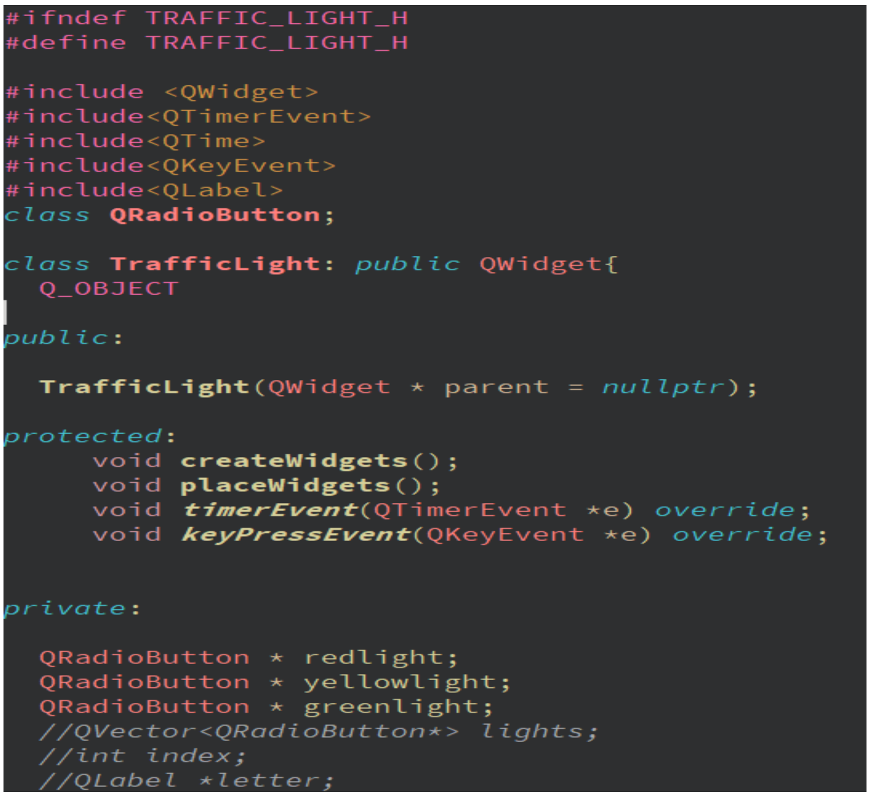
2.Calclator

3.Digital Clock

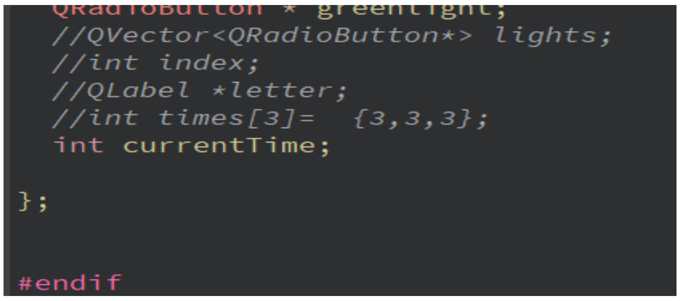
1.Traffic Light

Goal : is add some functions in order to change **each 3 sedonds** in the following order

- Traffic.h

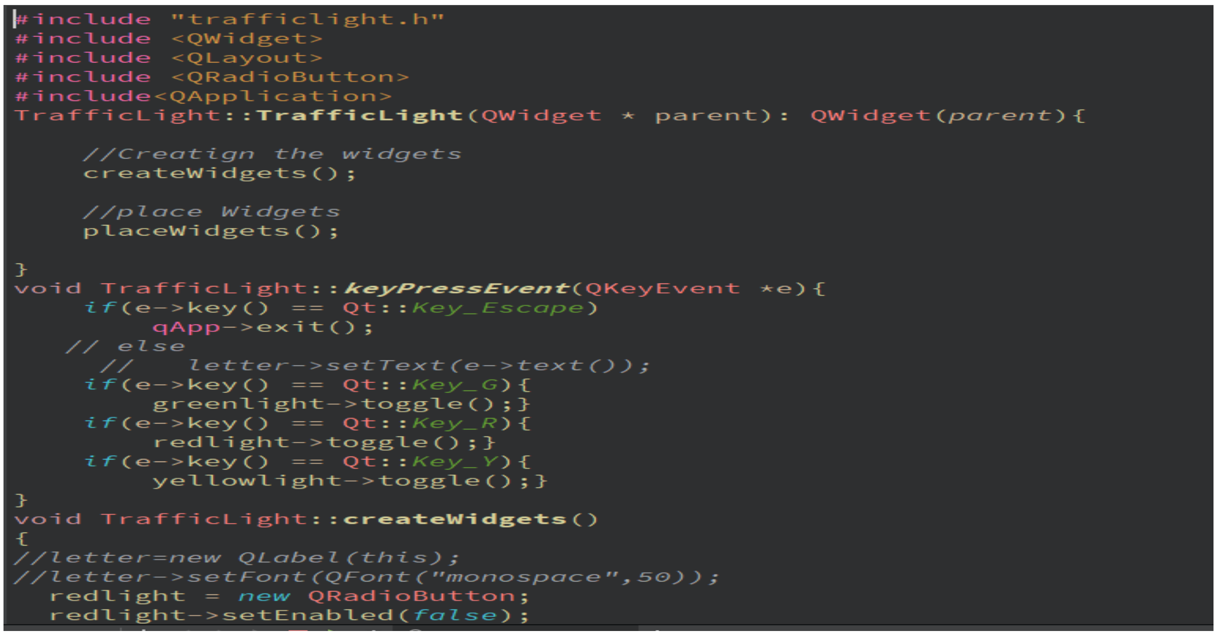


```
#ifndef TRAFFIC_LIGHT_H
#define TRAFFIC_LIGHT_H

#include <QWidget>
#include<QTimerEvent>
#include<QTime>
#include<QKeyEvent>
#include<QLabel>
class QRadioButton;

class TrafficLight: public QWidget{
  Q_OBJECT

public:

  TrafficLight(QWidget * parent = nullptr);

protected:
     void createWidgets();
     void placeWidgets();
     void timerEvent(QTimerEvent *e) override;
     void keyPressEvent(QKeyEvent *e) override;


private:

  QRadioButton * redlight;
  QRadioButton * yellowlight;
  QRadioButton * greenlight;
  //QVector<QRadioButton*> lights;
  //int index;
  //QLabel *letter;
```

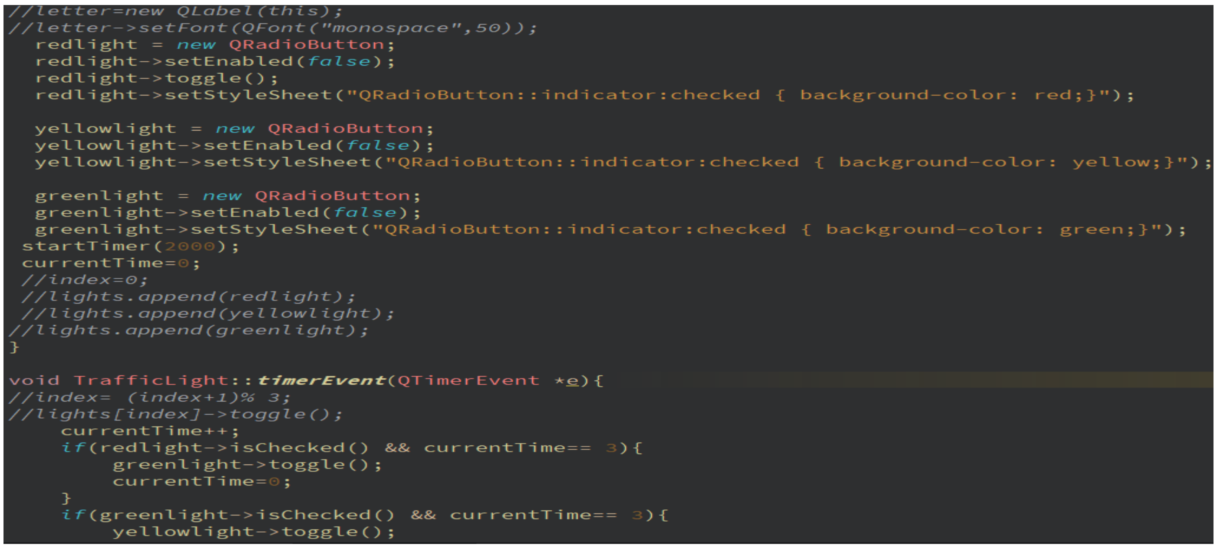


```
  QRadioButton * greenlight;
  //QVector<QRadioButton*> lights;
  //int index;
  //QLabel *letter;
  //int times[3]=  {3,3,3};
  int currentTime;

};


#endif
```

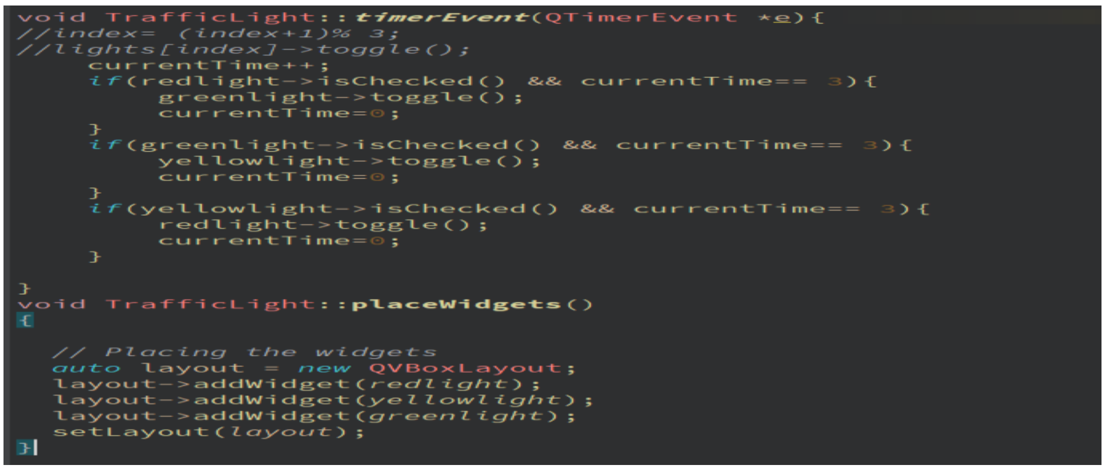
Traffic.cpp



```cpp
#include "trafficlight.h"
#include <QWidget>
#include <QLayout>
#include <QRadioButton>
#include<QApplication>
TrafficLight::TrafficLight(QWidget * parent): QWidget(parent){

    //Creatign the widgets
    createWidgets();

    //place Widgets
    placeWidgets();

}
void TrafficLight::keyPressEvent(QKeyEvent *e){
    if(e->key() == Qt::Key_Escape)
        qApp->exit();
   // else
     //    letter->setText(e->text());
    if(e->key() == Qt::Key_G){
        greenlight->toggle();}
    if(e->key() == Qt::Key_R){
        redlight->toggle();}
    if(e->key() == Qt::Key_Y){
        yellowlight->toggle();}
}
void TrafficLight::createWidgets()
{
//letter=new QLabel(this);
//letter->setFont(QFont("monospace",50));
  redlight = new QRadioButton;
  redlight->setEnabled(false);
```

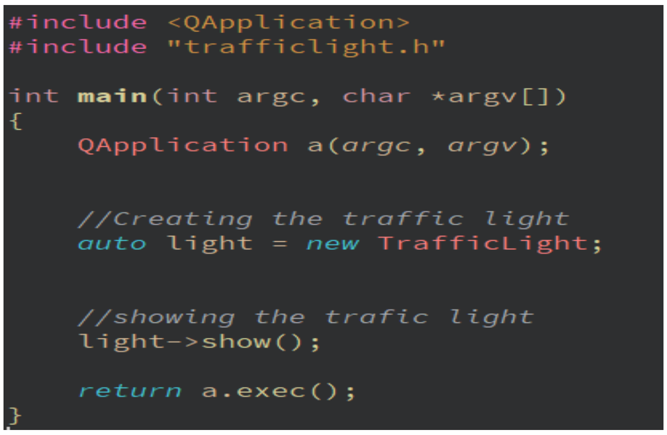


```cpp
//letter=new QLabel(this);
//letter->setFont(QFont("monospace",50));
  redlight = new QRadioButton;
  redlight->setEnabled(false);
  redlight->toggle();
  redlight->setStyleSheet("QRadioButton::indicator:checked { background-color: red;}");

  yellowlight = new QRadioButton;
  yellowlight->setEnabled(false);
  yellowlight->setStyleSheet("QRadioButton::indicator:checked { background-color: yellow;}");

  greenlight = new QRadioButton;
  greenlight->setEnabled(false);
  greenlight->setStyleSheet("QRadioButton::indicator:checked { background-color: green;}");
 startTimer(2000);
 currentTime=0;
 //index=0;
 //lights.append(redlight);
 //lights.append(yellowlight);
//lights.append(greenlight);
}

void TrafficLight::timerEvent(QTimerEvent *e){
//index= (index+1)% 3;
//lights[index]->toggle();
    currentTime++;
    if(redlight->isChecked() && currentTime== 3){
        greenlight->toggle();
        currentTime=0;
    }
    if(greenlight->isChecked() && currentTime== 3){
        yellowlight->toggle();
```



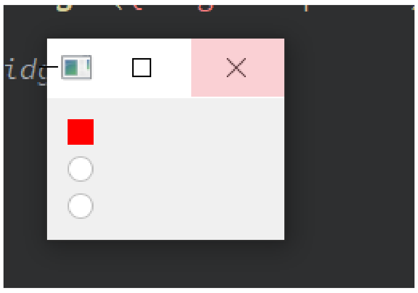
```cpp
void TrafficLight::timerEvent(QTimerEvent *e){
//index= (index+1)% 3;
//lights[index]->toggle();
    currentTime++;
    if(redlight->isChecked() && currentTime== 3){
        greenlight->toggle();
        currentTime=0;
    }
    if(greenlight->isChecked() && currentTime== 3){
        yellowlight->toggle();
        currentTime=0;
    }
    if(yellowlight->isChecked() && currentTime== 3){
        redlight->toggle();
        currentTime=0;
    }

}
void TrafficLight::placeWidgets()
{

  // Placing the widgets
  auto layout = new QVBoxLayout;
  layout->addWidget(redlight);
  layout->addWidget(yellowlight);
  layout->addWidget(greenlight);
  setLayout(layout);
}
```

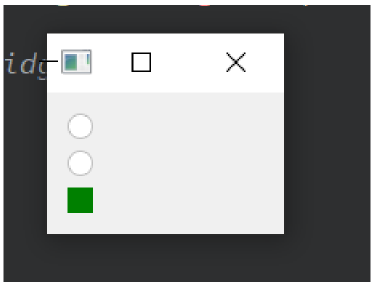
- Main

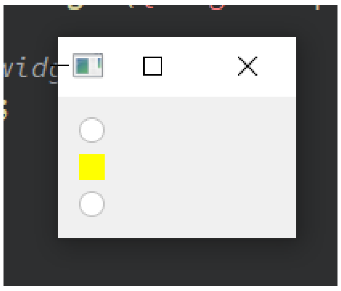


```
#include <QApplication>
#include "trafficlight.h"

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


    //Creating the traffic light
    auto light = new TrafficLight;


    //showing the trafic light
    light->show();

    return a.exec();
}
```

- Final result



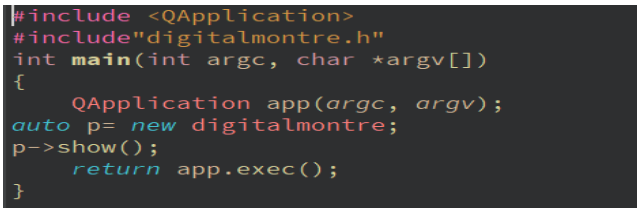




2.Digitalclock

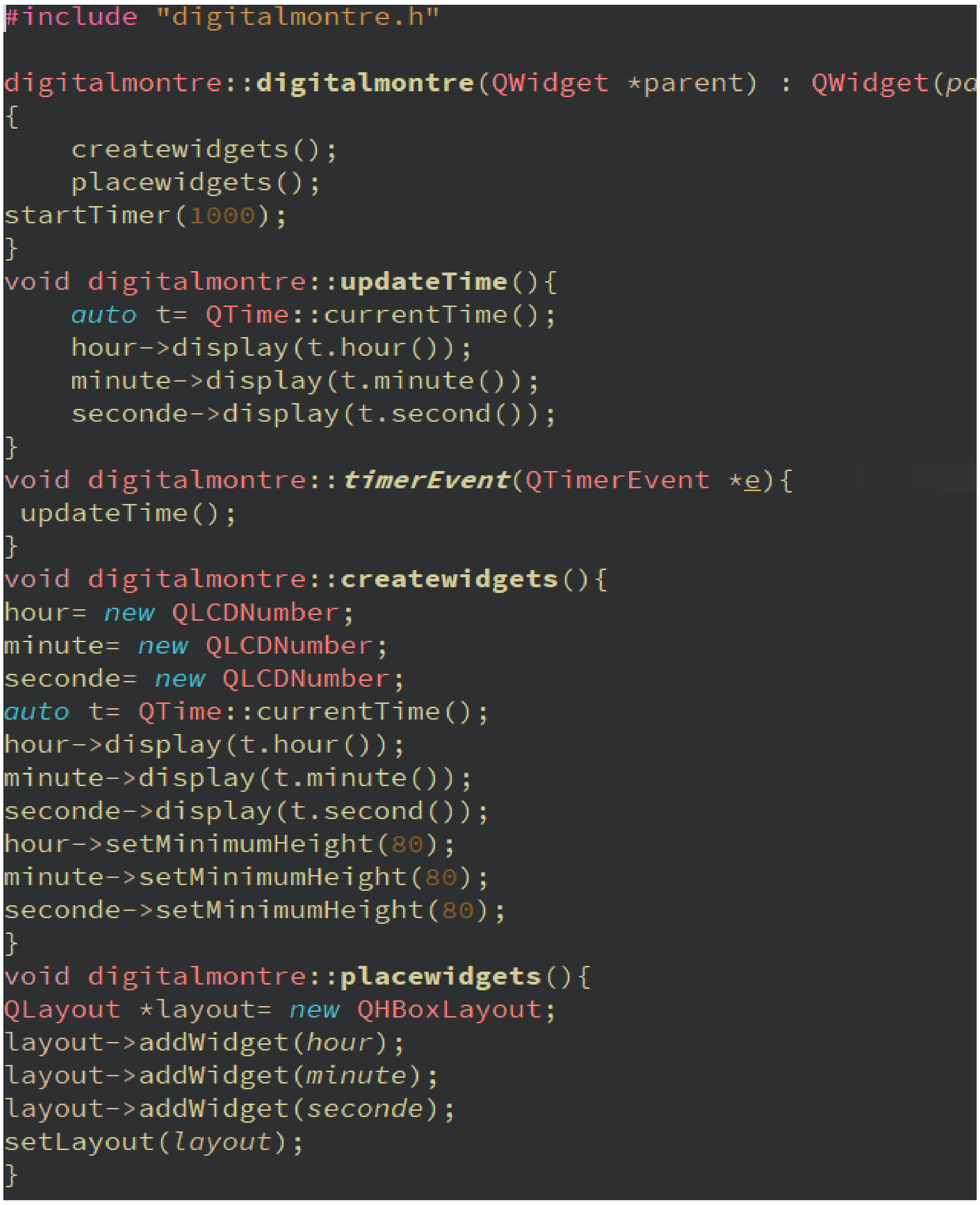
The Digital Clock example shows how to use QLCDNumber to display a number with LCD-like digits.

- Main



```cpp
#include <QApplication>
#include"digitalmontre.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
auto p= new digitalmontre;
p->show();
    return app.exec();
}
```

- Clock.cpp

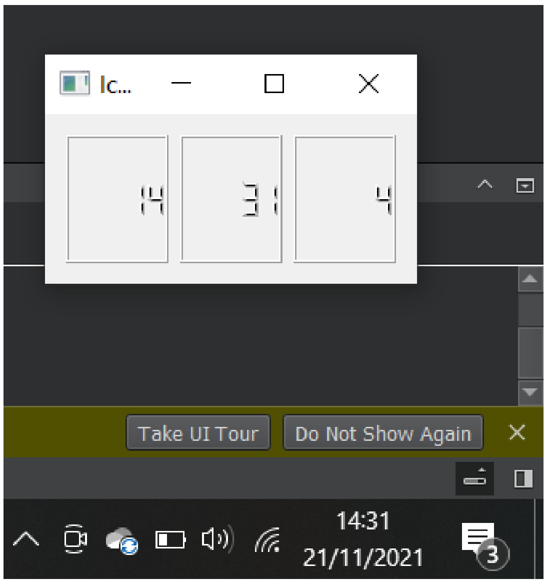


```cpp
#include "digitalmontre.h"

digitalmontre::digitalmontre(QWidget *parent) : QWidget(pa
{
    createwidgets();
    placewidgets();
startTimer(1000);
}
void digitalmontre::updateTime(){
    auto t= QTime::currentTime();
    hour->display(t.hour());
    minute->display(t.minute());
    seconde->display(t.second());
}
void digitalmontre::timerEvent(QTimerEvent *e){
 updateTime();
}
void digitalmontre::createwidgets(){
hour= new QLCDNumber;
minute= new QLCDNumber;
seconde= new QLCDNumber;
auto t= QTime::currentTime();
hour->display(t.hour());
minute->display(t.minute());
seconde->display(t.second());
hour->setMinimumHeight(80);
minute->setMinimumHeight(80);
seconde->setMinimumHeight(80);
}
void digitalmontre::placewidgets(){
QLayout *layout= new QHBoxLayout;
layout->addWidget(hour);
layout->addWidget(minute);
layout->addWidget(seconde);
setLayout(layout);
}
```

- Clock.h

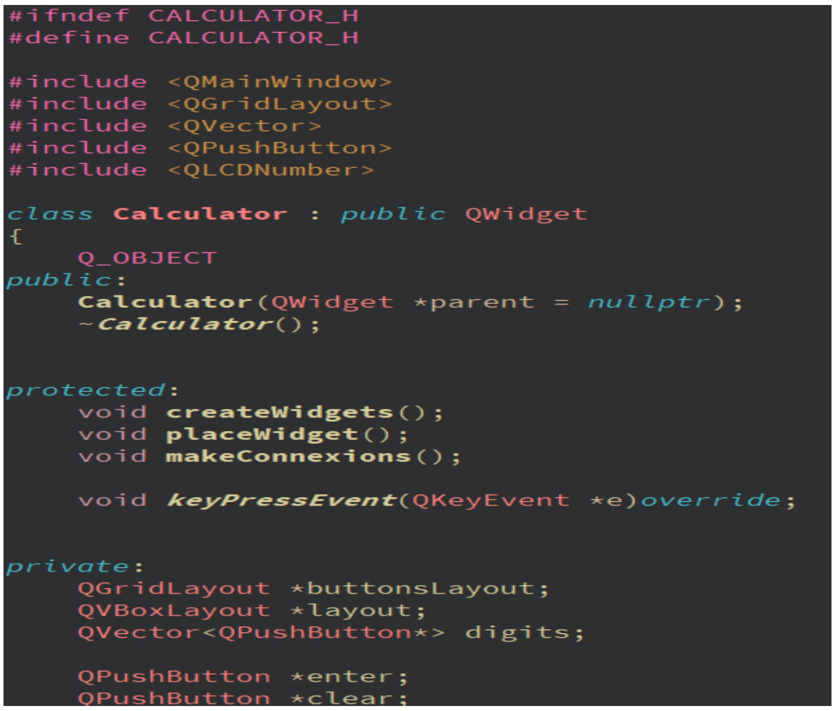
```
#ifndef DIGITALMONTRE_H
#define DIGITALMONTRE_H

#include <QWidget>
#include<QTimerEvent>
#include<QLabel>
#include<QTime>
#include<QLCDNumber>
#include<QHBoxLayout>
class digitalmontre : public QWidget
{
    Q_OBJECT
public:
    explicit digitalmontre(QWidget *parent = nullptr);

protected:
     void timerEvent(QTimerEvent *e)override;
void createwidgets();
void placewidgets();
void updateTime();
private:
QLabel *time;
QLCDNumber *hour;
QLCDNumber *minute;
QLCDNumber *seconde;

};

#endif // DIGITALMONTRE_H
```

- Final Resulat

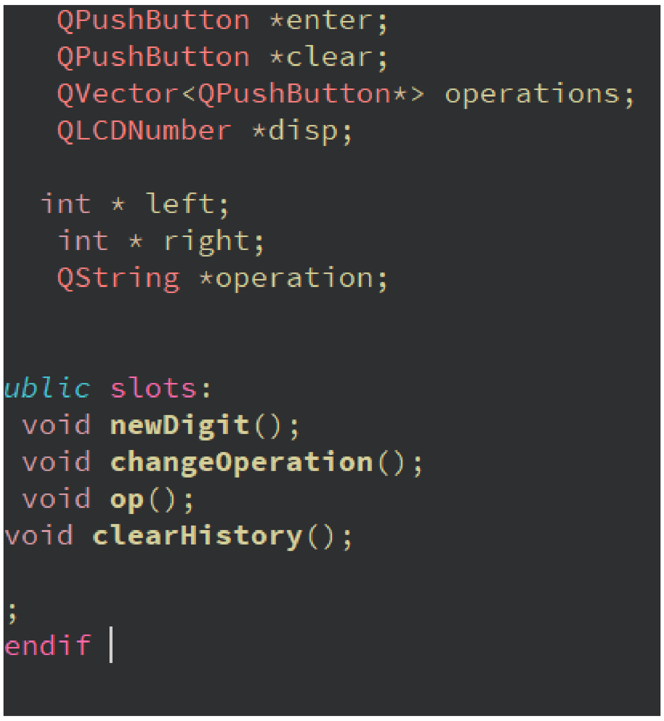


3.Calculator

- Calculator.h

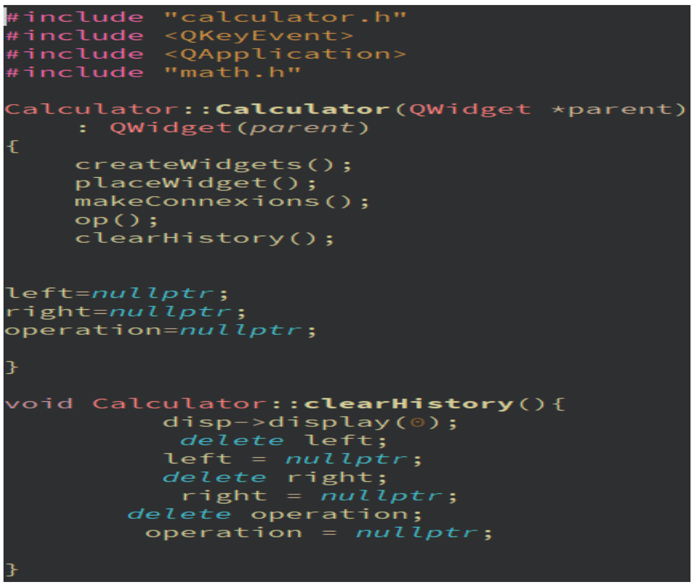


```
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include <QMainWindow>
#include <QGridLayout>
#include <QVector>
#include <QPushButton>
#include <QLCDNumber>

class Calculator : public QWidget
{
    Q_OBJECT
public:
    Calculator(QWidget *parent = nullptr);
    ~Calculator();


protected:
    void createWidgets();
    void placeWidget();
    void makeConnexions();

    void keyPressEvent(QKeyEvent *e)override;


private:
    QGridLayout *buttonsLayout;
    QVBoxLayout *layout;
    QVector<QPushButton*> digits;

    QPushButton *enter;
    QPushButton *clear;
```

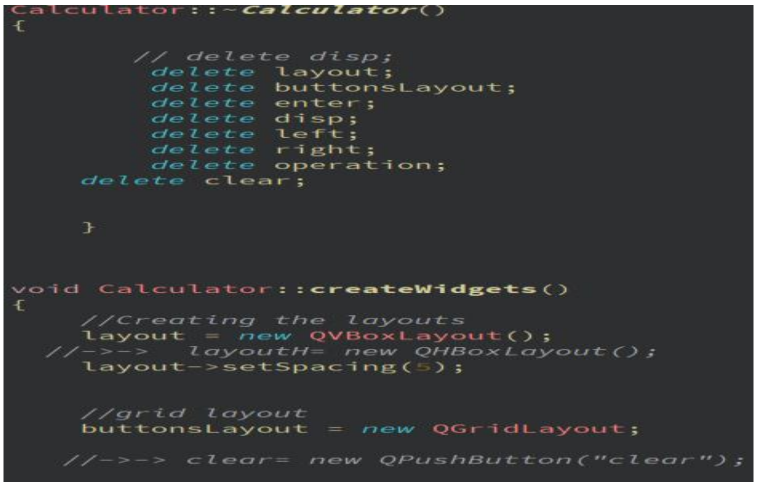


```
   QPushButton *enter;
   QPushButton *clear;
   QVector<QPushButton*> operations;
   QLCDNumber *disp;

  int * left;
   int * right;
   QString *operation;


ublic slots:
 void newDigit();
 void changeOperation();
 void op();
void clearHistory();

;
endif
```

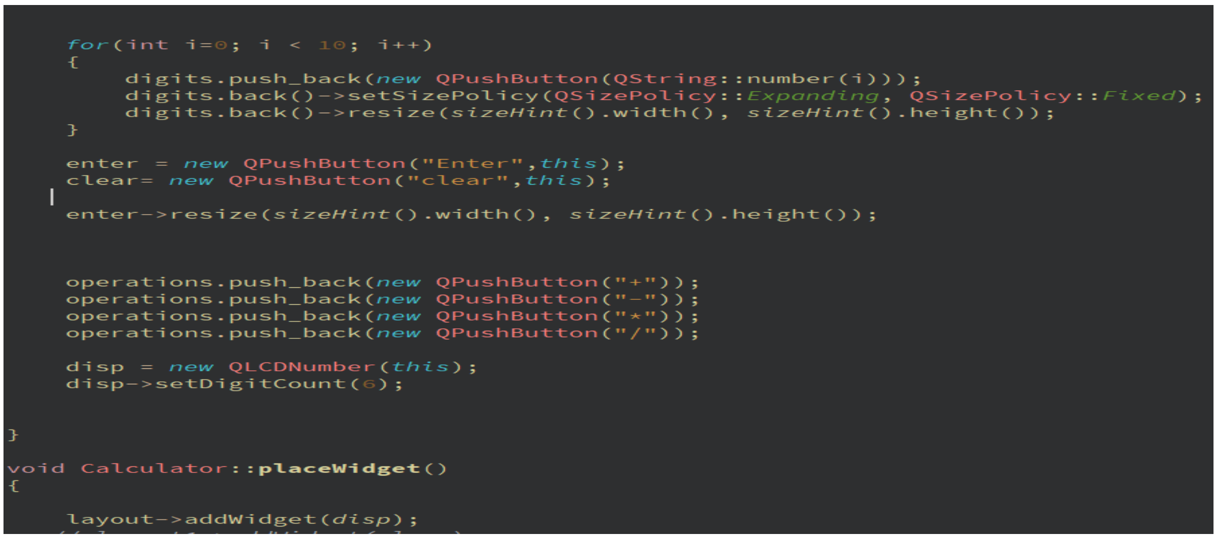
Calculator.cpp



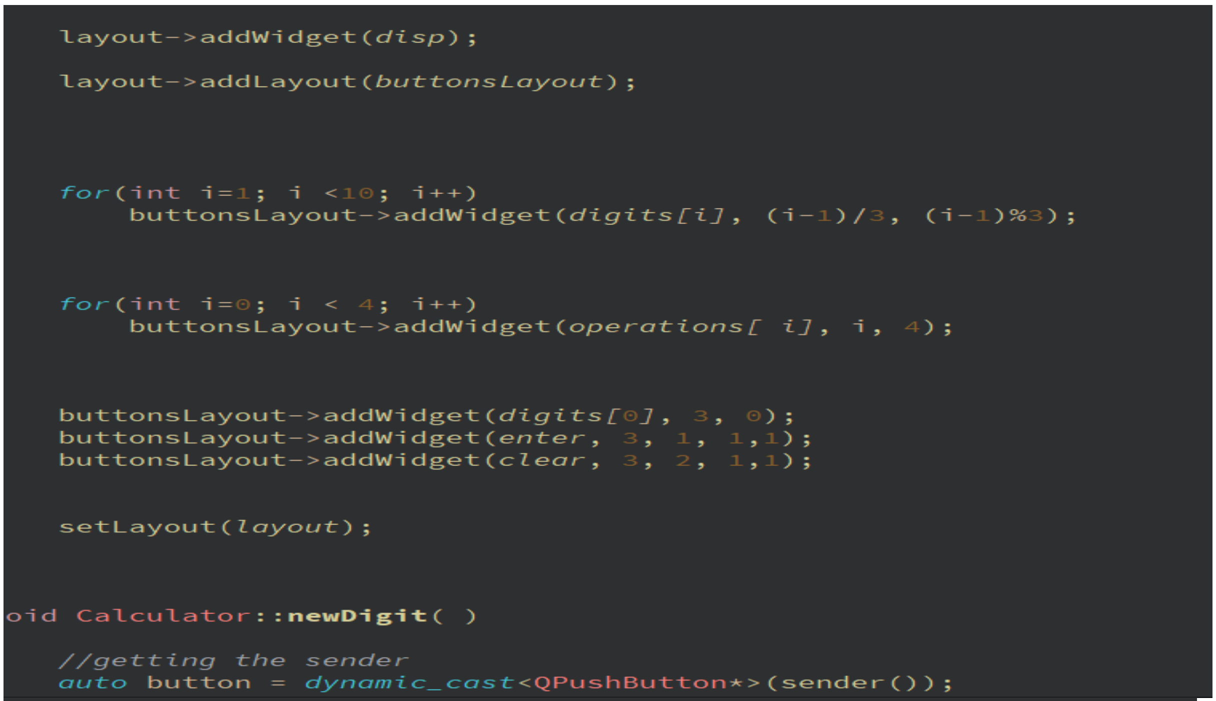
```cpp
#include "calculator.h"
#include <QKeyEvent>
#include <QApplication>
#include "math.h"

Calculator::Calculator(QWidget *parent)
    : QWidget(parent)
{
    createWidgets();
    placeWidget();
    makeConnexions();
    op();
    clearHistory();


left=nullptr;
right=nullptr;
operation=nullptr;

}

void Calculator::clearHistory(){
         disp->display(0);
          delete left;
         left = nullptr;
         delete right;
          right = nullptr;
       delete operation;
        operation = nullptr;

}
```



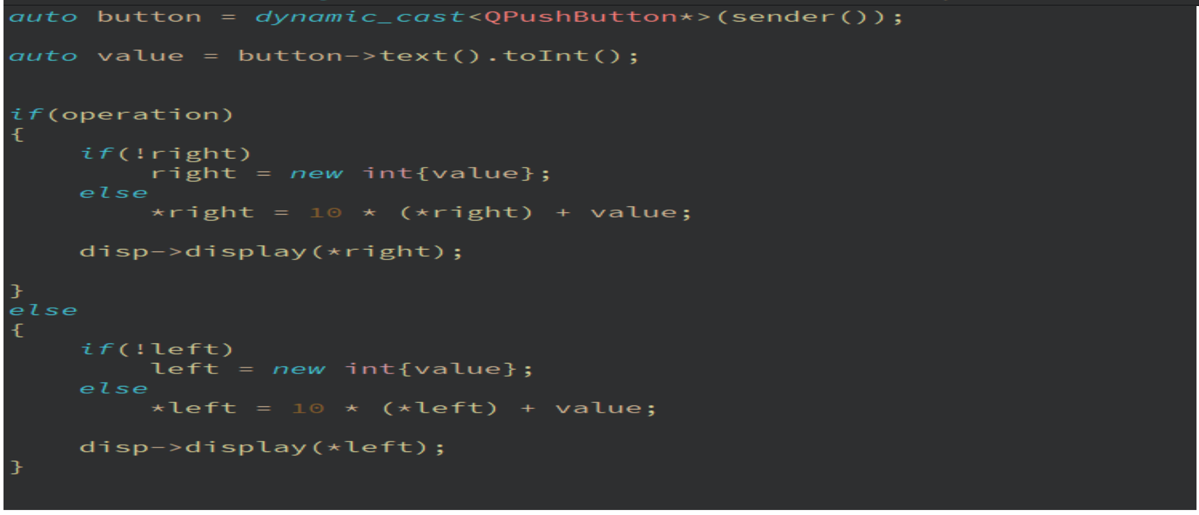
```cpp
Calculator::~Calculator()
{

       // delete disp;
        delete layout;
        delete buttonsLayout;
        delete enter;
        delete disp;
        delete left;
        delete right;
        delete operation;
    delete clear;


    }


void Calculator::createWidgets()
{
    //Creating the layouts
    layout = new QVBoxLayout();
  //->-->  layoutH= new QHBoxLayout();
    layout->setSpacing(5);


    //grid layout
    buttonsLayout = new QGridLayout;

   //->--> clear= new QPushButton("clear");
```



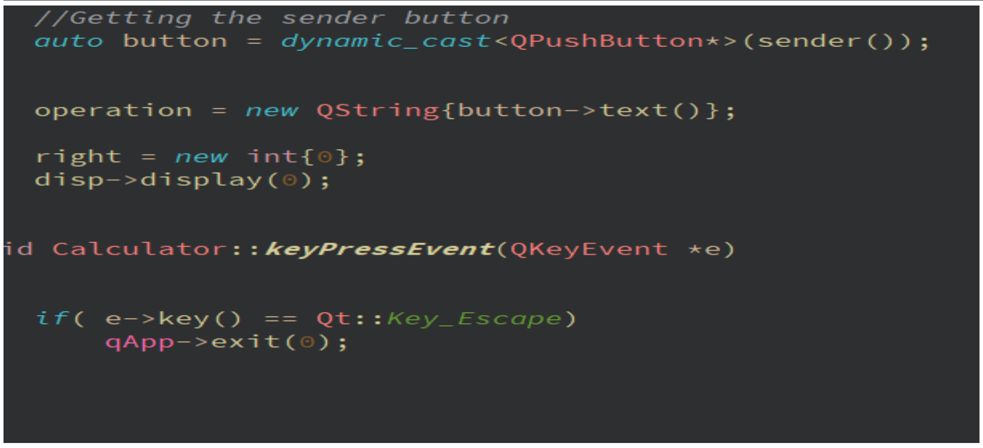
```cpp

    for(int i=0; i < 10; i++)
    {
        digits.push_back(new QPushButton(QString::number(i)));
        digits.back()->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
        digits.back()->resize(sizeHint().width(), sizeHint().height());
    }

    enter = new QPushButton("Enter",this);
    clear= new QPushButton("clear",this);

    enter->resize(sizeHint().width(), sizeHint().height());


    operations.push_back(new QPushButton("+"));
    operations.push_back(new QPushButton("-"));
    operations.push_back(new QPushButton("*"));
    operations.push_back(new QPushButton("/"));

    disp = new QLCDNumber(this);
    disp->setDigitCount(6);


}
void Calculator::placeWidget()
{

    layout->addWidget(disp);
```



```
   layout->addWidget(disp);

   layout->addLayout(buttonsLayout);



   for(int i=1; i <10; i++)
       buttonsLayout->addWidget(digits[i], (i-1)/3, (i-1)%3);



   for(int i=0; i < 4; i++)
       buttonsLayout->addWidget(operations[ i], i, 4);



   buttonsLayout->addWidget(digits[0], 3, 0);
   buttonsLayout->addWidget(enter, 3, 1, 1,1);
   buttonsLayout->addWidget(clear, 3, 2, 1,1);


   setLayout(layout);


oid Calculator::newDigit( )

   //getting the sender
   auto button = dynamic_cast<QPushButton*>(sender());
```

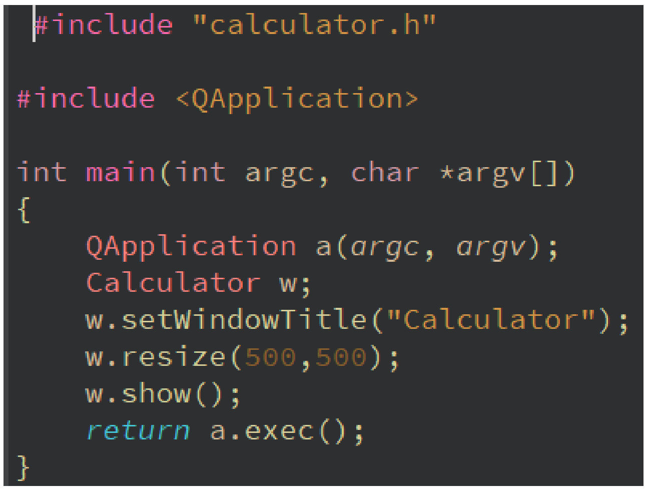


```
auto button = dynamic_cast<QPushButton*>(sender());

auto value = button->text().toInt();


if(operation)
{
    if(!right)
        right = new int{value};
    else
        *right = 10 * (*right) + value;

    disp->display(*right);

}
else
{
    if(!left)
        left = new int{value};
    else
        *left = 10 * (*left) + value;

    disp->display(*left);
}
```



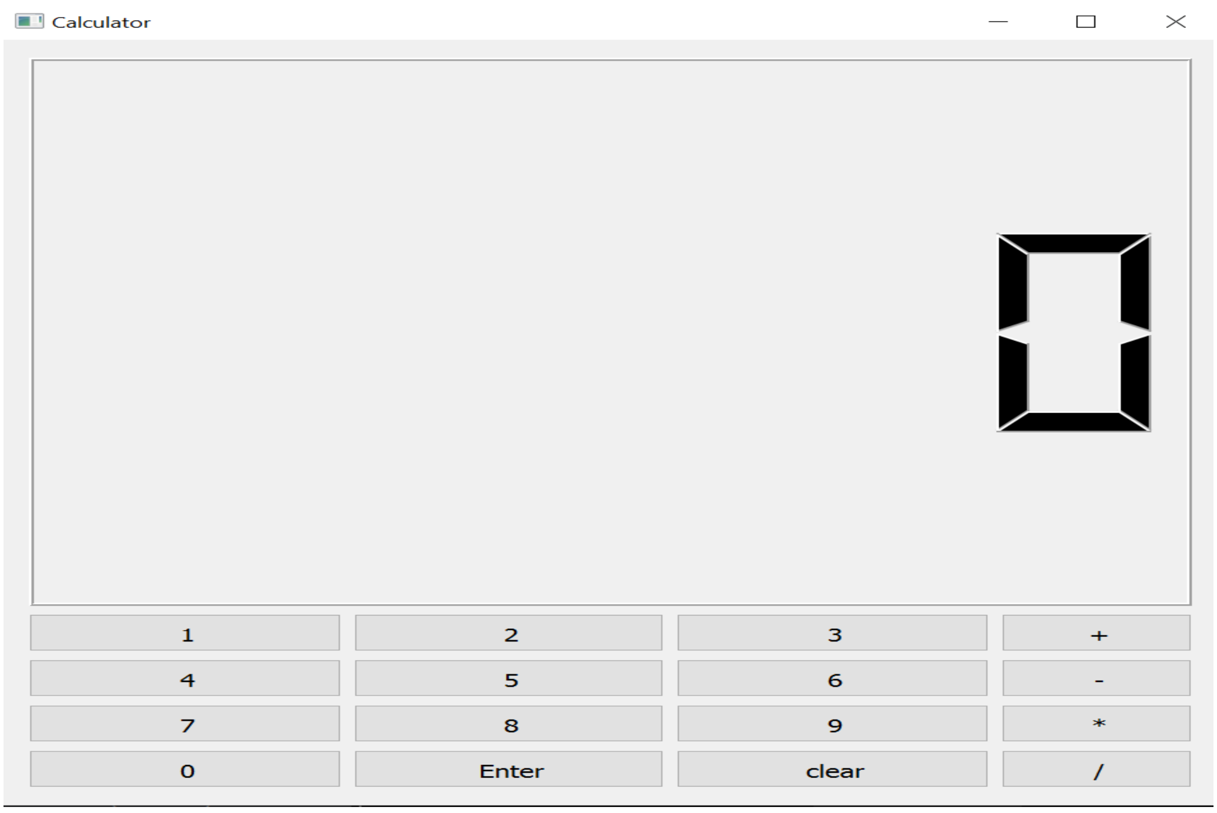
```
  //Getting the sender button
  auto button = dynamic_cast<QPushButton*>(sender());


  operation = new QString{button->text()};

  right = new int{0};
  disp->display(0);


id Calculator::keyPressEvent(QKeyEvent *e)


  if( e->key() == Qt::Key_Escape)
      qApp->exit(0);
```

- Main



```
#include "calculator.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Calculator w;
    w.setWindowTitle("Calculator");
    w.resize(500,500);
    w.show();
    return a.exec();
}
```

- Final Resulats



Made by Ghita Chraibi