Practical course report #3

Supervisor:

Dr. Anass Belcaid

[19-11-2021]

EIDIA

Authored by: -Yahia BENCHEDDI -Hamza Es-samlaly

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

Executive Summary

This project was a part of the first week assignment of a course on Human-Machine interface in C++ taught by Dr.Belcaid from department of computer science at UEMF. The goal of this project was to manipulate connections between different components of a given pre-coded calculator .

Program overview

Objectives

The goal of the program was to edit the code in the calculator zip file in order to create connections between numbers lcd display and the operations supported by the calculator.

The project was a bit easy specially with all the instructions given for the majority of the work, we faced some problems due to the crashing of the app with no error what so ever in the code but we managed to overcome the problem. Here’s the code that we added and edited:

In the .h file we declared some private members:

```
private:
    int * left;          //left operand
    int * right;         // right operand
    QString *operation;  // Pointer on the current operation
```

Then we declared the slots that we were gonna be using:

```
public slots:
    void changeOperation();
    void newDigit();
    void enterButtonclicked();
```

changeOperation() slot is to handle the click on opertations

newDigit() slot is to show the digits on the LCD display

enterButtonclicked() slot is to manipulate the enter button once the numbers and operations are entered.

Now in the .cpp file we coded the implementation of the slots declared in the header:

newDigit():

```
void Calculator::newDigit( )
{
    //getting the sender
    auto button = dynamic_cast<QPushButton*>(sender());

    //getting the value
    int value = button->text().toInt();

    //Check if we have an operation defined
    if(operation)
    {
        //check if we have a value or not
        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);
    }
}
```

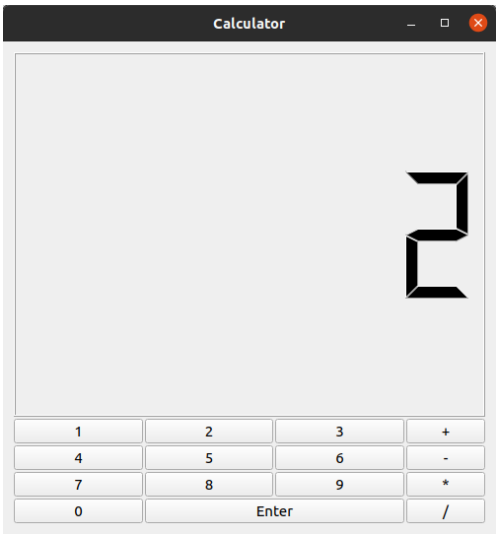
changeOperation():

```
void Calculator::changeOperation()
{
    //Getting the sender button
    auto button = dynamic_cast<QPushButton*>(sender());

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

    //Initiating the right button
    right = new int{0};

    //reseting the display
    disp->display(0);
}
```

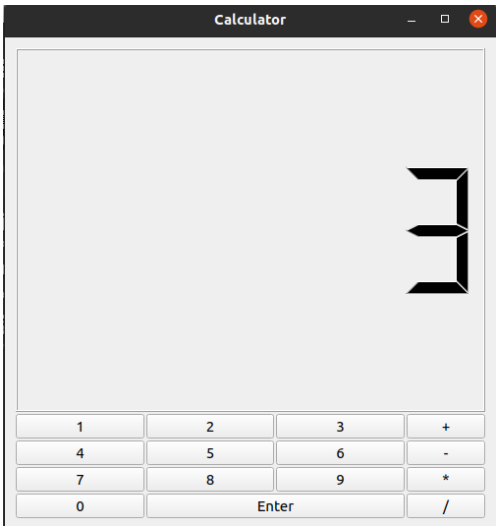
enterButtonclicked():

```
void Calculator::enterButtonclicked()
{
    if(operation == (QString) "+")
    {
        int value = *left + *right;
     disp->display(value);
    }

    if(operation == (QString) "-")
    {
        int value = *left - *right;
     disp->display(value);
    }

    if(operation == (QString) "*")
    {
        int value = *left * *right;
     disp->display(value);
    }

    if(operation == (QString) "/")
    {
        int value = *left / *right;
     disp->display(value);
    }

}
```

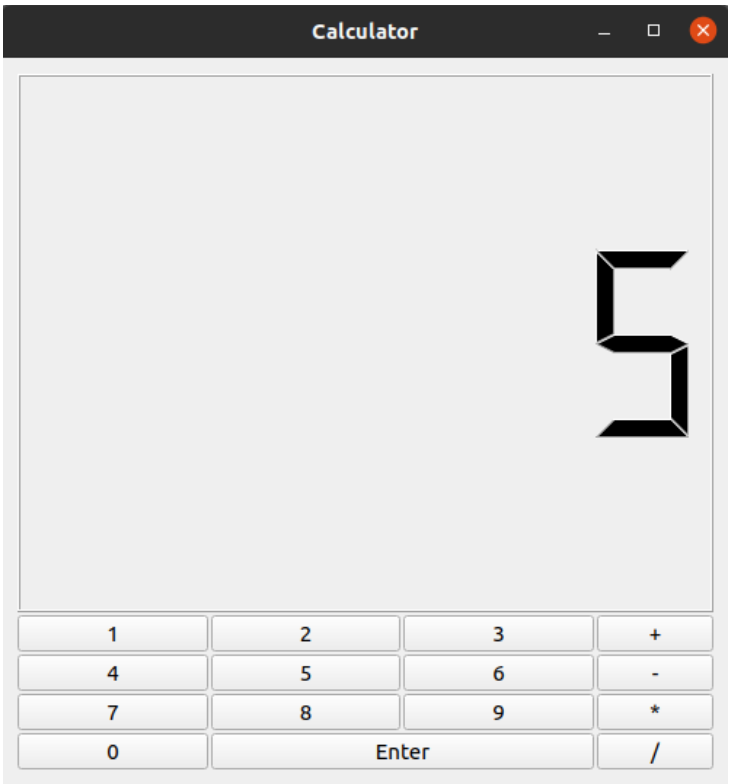
Here’s a simple operation of 2+3 and its result:



+



=



About the Digital Clock project we weren’t sure if we were to include it or not but here it is:

Here’s the header:

digitaclock.h:

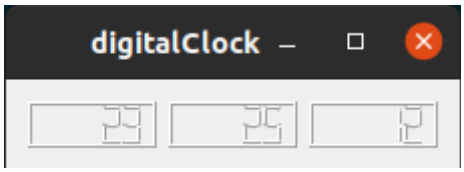
```
#ifndef TRAFFIC_LIGHT_H
#define TRAFFIC_LIGHT_H

#include <QWidget>
#include <QTimerEvent>
#include <QLCDNumber>
#include <QHBoxLayout>
#include <QTime>

class QRadioButton;

class DigitalClock: public QWidget{
  Q_OBJECT

public:

  DigitalClock(QWidget * parent = nullptr);

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

private:

   QLCDNumber *hour;
   QLCDNumber *minute;
   QLCDNumber *second;

};

#endif
```

Here’s the .cpp file:

digitalclock.cpp:

```
#include <digitalclock.h>
DigitalClock::DigitalClock(QWidget *parent) : QWidget(parent)
{
    createWidgets();
    placeWidgets();
    updateTime();
    startTimer(1000);
}

void DigitalClock::updateTime()
{
    auto T = QTime::currentTime();
        hour->display(T.hour());
        minute->display(T.minute());
        second->display(T.second());

}
void DigitalClock::createWidgets()
{
    hour = new QLCDNumber;
    minute = new QLCDNumber;
    second = new QLCDNumber;

}

void DigitalClock::placeWidgets()
{
    auto layout = new QHBoxLayout();
    setLayout(layout);
    layout->addWidget(hour);
    layout->addWidget(minute);
    layout->addWidget(second);

}

void DigitalClock::timerEvent(QTimerEvent *e)
{
    updateTime();
}
```

And here’s a look on the output of the program:



Excuse us for we have a graphical problem that we couldn’t find a solution for, here’s the error we keep on getting even though the program runs:

```
23:24:12: Starting /home/yahia/build-digitalClock-Desktop-Debug/digitalClock ...
Warning: QT_DEVICE_PIXEL_RATIO is deprecated. Instead use:
   QT_AUTO_SCREEN_SCALE_FACTOR to enable platform plugin controlled per-screen factors.
   QT_SCREEN_SCALE_FACTORS to set per-screen DPI.
   QT_SCALE_FACTOR to set the application global scale factor.
```

Hope you can help us solve this problem in the next session professor.

Conclusion

The practical course was very helpful and fun to code, thank you for your time professor and hope you find it interesting.