Programming Widget Layout TP

07/11/2021

Créé par : EL HARRARI Anas

École d'Ingénierie Digitale et d'Intelligence Artificielle

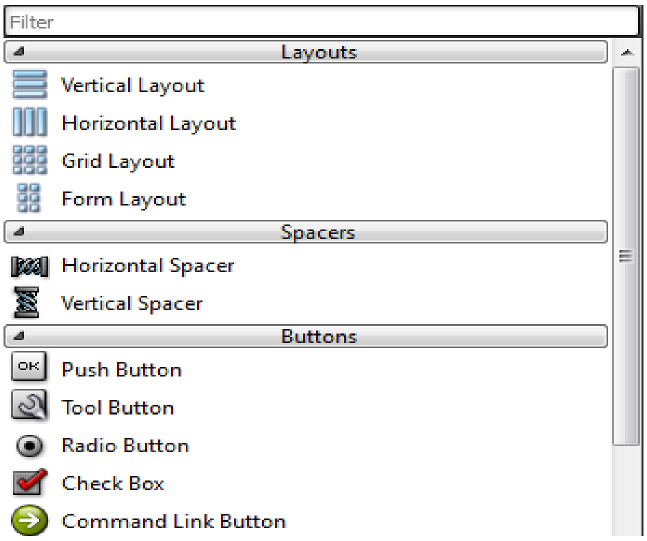
Programming Widget Layout

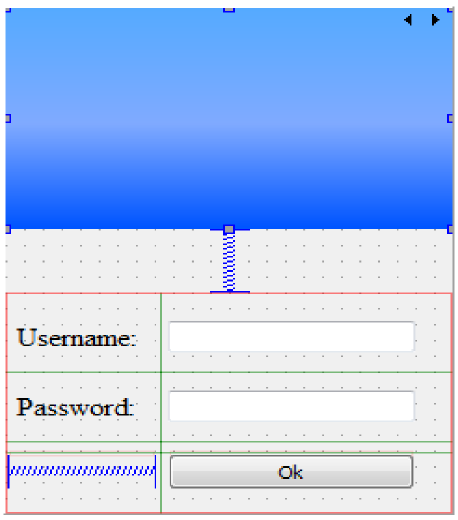
Motivation

A page layout program is the assembly area where all the parts of a project are put together. Page layout is the part of graphic design that deals with the arrangement of visual elements on a page. It establishes the overall appearance, relative importance, and relationships between the graphic elements to achieve a smooth flow of information and eye movement for maximum effectiveness or impact.

The Qt layout system provides a simple and powerful way of automatically arranging child widgets within a widget to ensure that they make good use of the available space. In other words, as it said in the Qt tutorial introduction: Qt includes a set of layout management classes that are used to describe how widgets are laid out in an application's user interface. These layouts automatically position and resize widgets when the amount of space available for them changes, ensuring that they are consistently arranged and that the user interface as a whole remains usable.

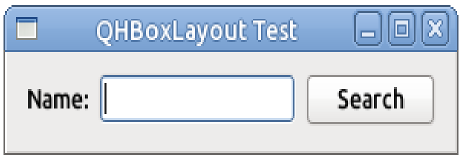
<<In this session, we will be trying to explore the powerful and the strong classes of QT. In addition, we will try to understand the mechanism of Layout working and discover different types of Layouts. >>





Experimenting with QHBOXLayout

In this task, we are going to modify the following code in order to display the window below:

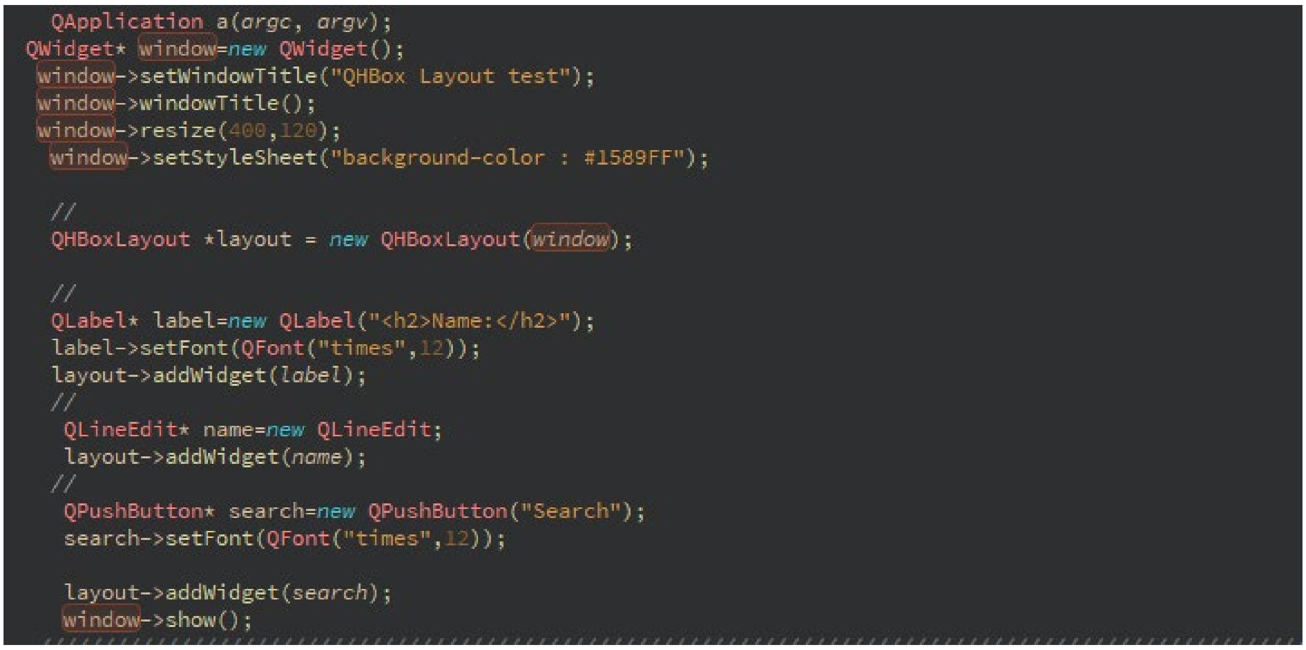


```
int main(int argc, char* argv[])
{
  QApplication app(argc, argv);

  QWidget* window = new QWidget();
  window->setWindowTitle("QHBoxLayout Test");

  window->show();

  return app.exec();
}
```

The purpose of this work is to practice with the QT layout's classes. The first type of these layouts is "QHBoxLayout". Here, we are showing the modified code that we will discuss later:

For a simplification coding, we are writing all the code in this work inside the "main". We remind that you would find the Qt project with the zip file.

First, it would be all clear that we should some includes (#include QWidget, #include

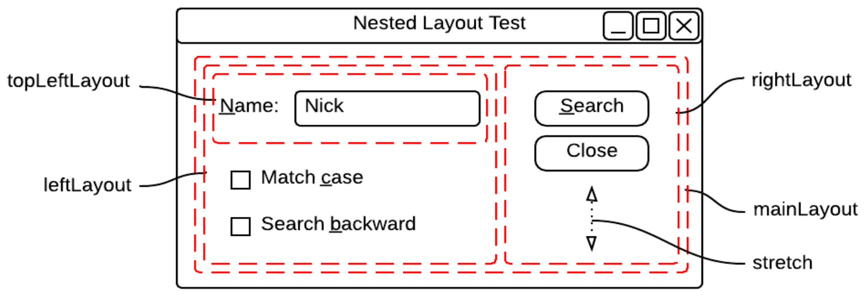


```
  QApplication a(argc, argv);
QWidget* window=new QWidget();
 window->setWindowTitle("QHBox Layout test");
 window->windowTitle();
 window->resize(400,120);
  window->setStyleSheet("background-color : #1589FF");

  //
  QHBoxLayout *layout = new QHBoxLayout(window);

  //
  QLabel* label=new QLabel("<h2>Name:</h2>");
  label->setFont(QFont("times",12));
  layout->addWidget(label);
  //
   QLineEdit* name=new QLineEdit;
   layout->addWidget(name);
  //
   QPushButton* search=new QPushButton("Search");
   search->setFont(QFont("times",12));

   layout->addWidget(search);
   window->show();
```

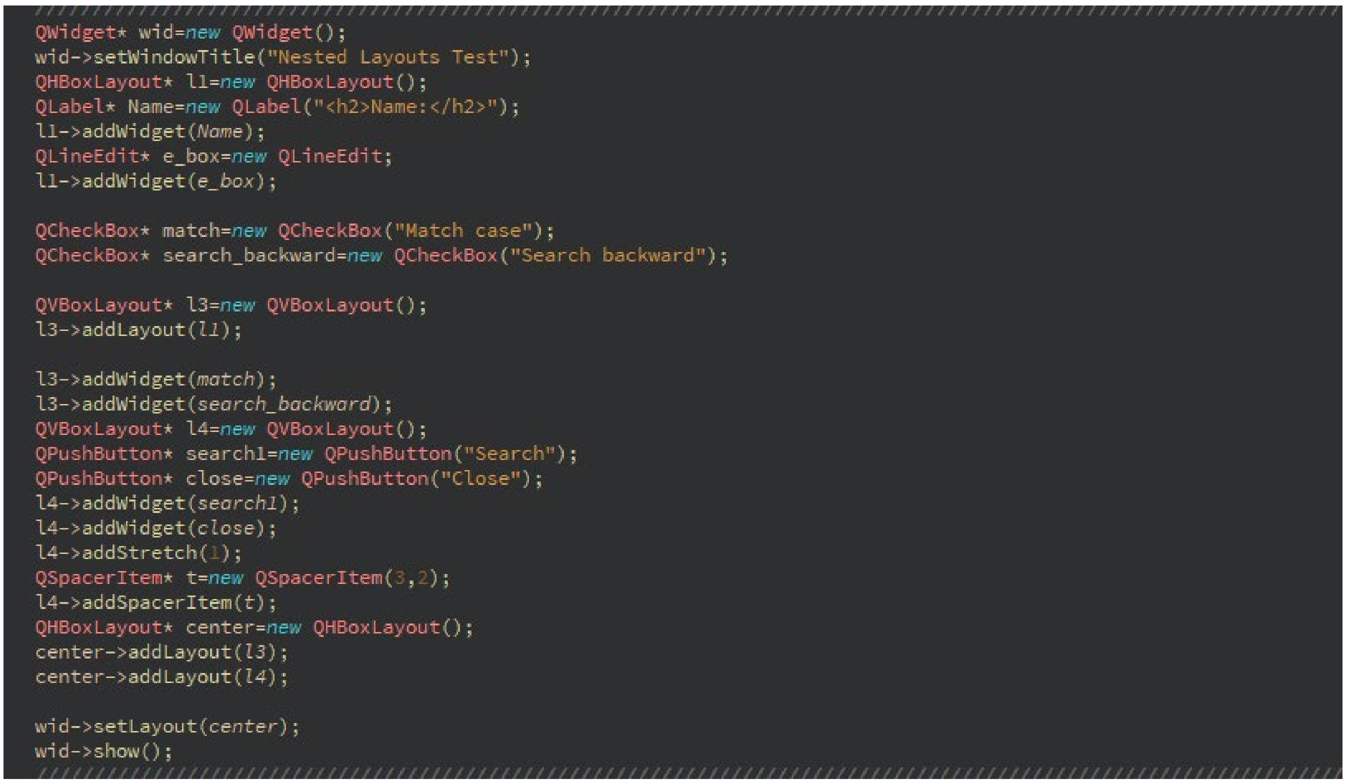
QHBoxLayout, #include QPushButton, etc.). In a first step, we are creating the components (line edits, buttons…). In our window, we have one LineEdit and a button after that, we are creating a QHBoxLayout. This type of layout will arrange the created components horizontally. We also have some additional properties like window->setStyleSheet where we change the color of the window.

Nested Layouts

In the second part, we will be discovering a quite interesting part of Layouts. It is what we called the “Nested Layouts”. The principal goal of this exercise is to learn how to analyze the construction of a form and then code it using Netsted layouts.Here; we are giving the following form:

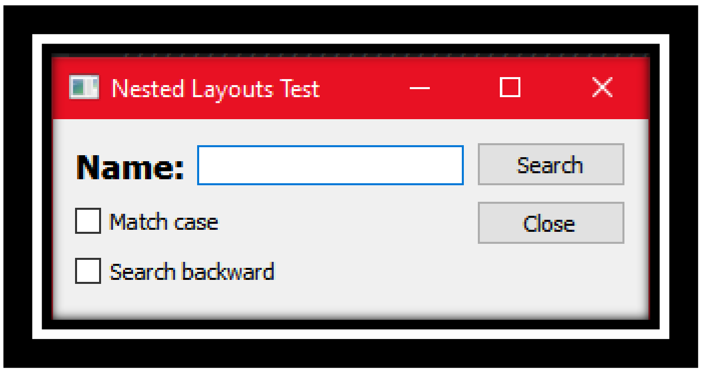


In this exercise, we are not supposed to code any functionality. Thus, we are just focusing on making the right code to display the form shown. As we can clearly see, we have five different layouts. The first one is topLeftLayout, this layout is responsible for arranging the label “name” and the line edit (it is a QHBoxLayout). The second one (a QVBoxLayout), leftLayout is arranging the two check boxes “Match case” and “Search backward”. As the last, one rightLayout is doing the same functionality with the two buttons. Finally, the main one, which is responsible of arranging all the layouts together. This time we are adding layouts to other ones. We also can see that we have something called stretch in order to put the two buttons at the top of the layout. Here, we are finding the code to display this window:



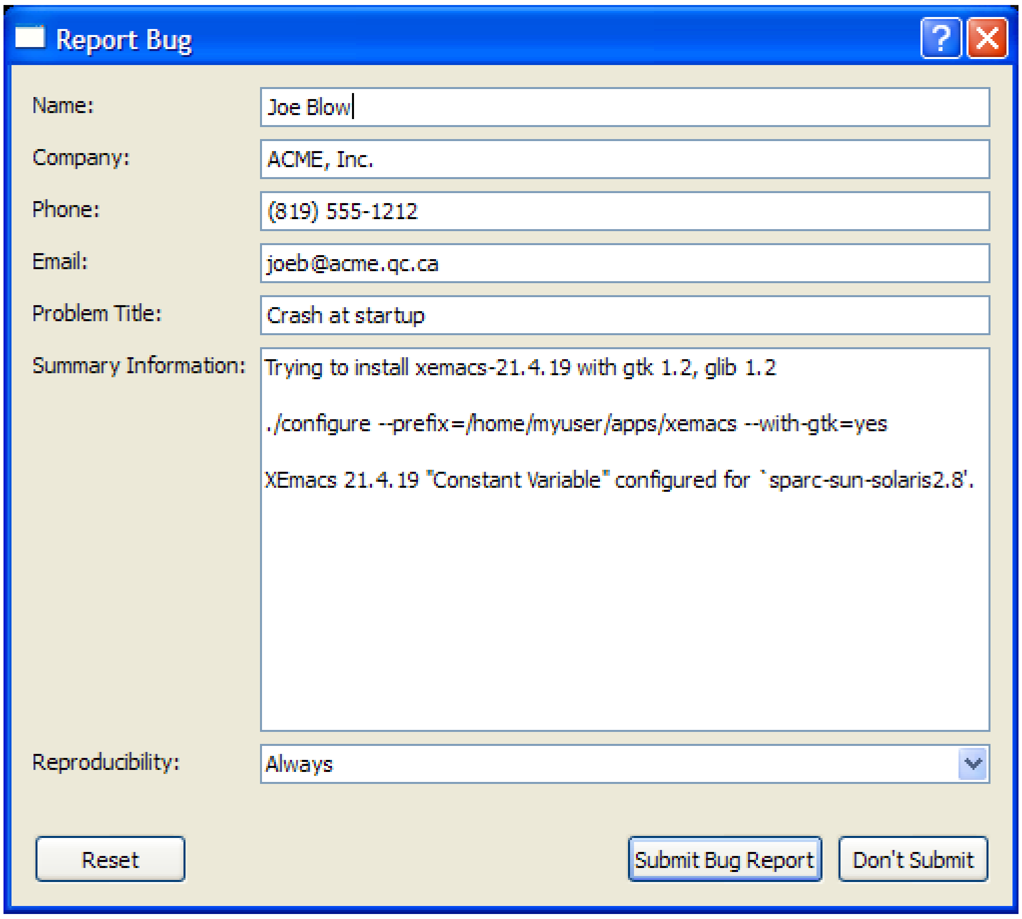
```
QWidget* wid=new QWidget();
wid->setWindowTitle("Nested Layouts Test");
QHBoxLayout* l1=new QHBoxLayout();
QLabel* Name=new QLabel("<h2>Name:</h2>");
l1->addWidget(Name);
QLineEdit* e_box=new QLineEdit;
l1->addWidget(e_box);

QCheckBox* match=new QCheckBox("Match case");
QCheckBox* search_backward=new QCheckBox("Search backward");

QVBoxLayout* l3=new QVBoxLayout();
l3->addLayout(l1);

l3->addWidget(match);
l3->addWidget(search_backward);
QVBoxLayout* l4=new QVBoxLayout();
QPushButton* search1=new QPushButton("Search");
QPushButton* close=new QPushButton("Close");
l4->addWidget(search1);
l4->addWidget(close);
l4->addStretch(1);
QSpacerItem* t=new QSpacerItem(3,2);
l4->addSpacerItem(t);
QHBoxLayout* center=new QHBoxLayout();
center->addLayout(l3);
center->addLayout(l4);

wid->setLayout(center);
wid->show();
```

Moreover, this is the result:



Bug report Form

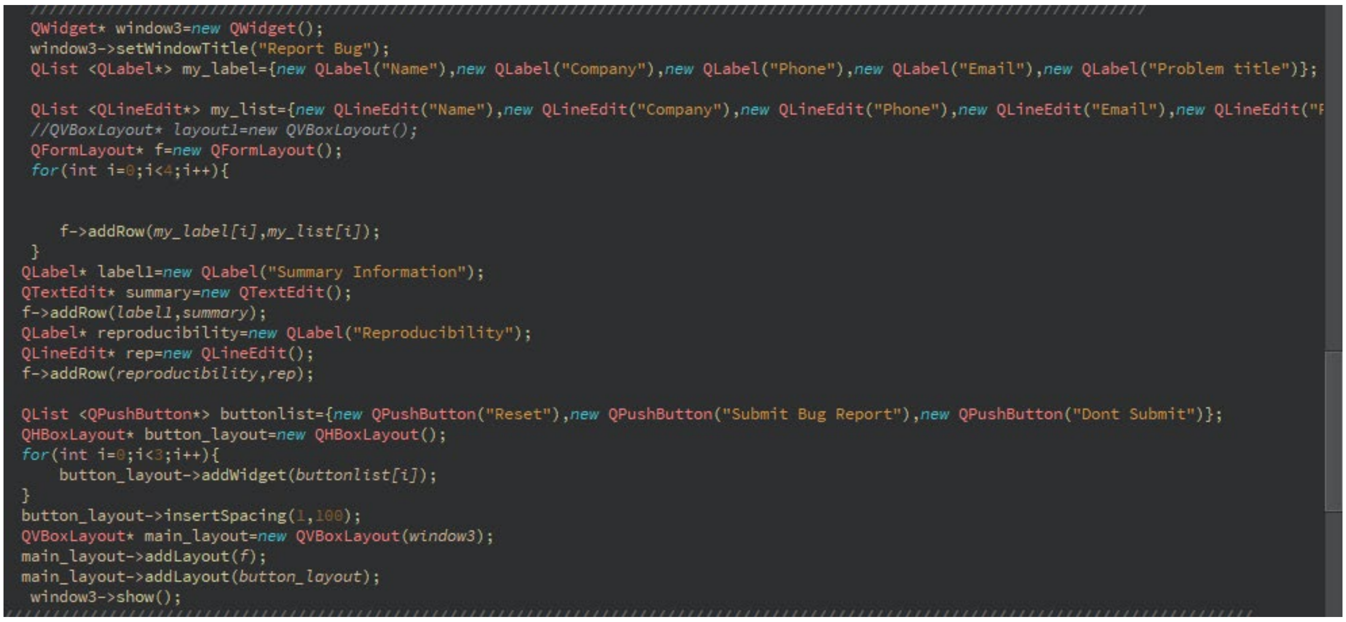
In this exercise, we are trying to write the appropriate code to display following window:



Name:	Joe Blow
Company:	ACME, Inc.
Phone:	(819) 555-1212
Email:	joeb@acme.qc.ca
Problem Title:	Crash at startup
Summary Information:	Trying to install xemacs-21.4.19 with gtk 1.2, glib 1.2 ./configure --prefix=/home/myuser/apps/xemacs --with-gtk=yes XEmacs 21.4.19 "Constant Variable" configured for `sparc-sun-solaris2.8'.
Reproducibility:	Always

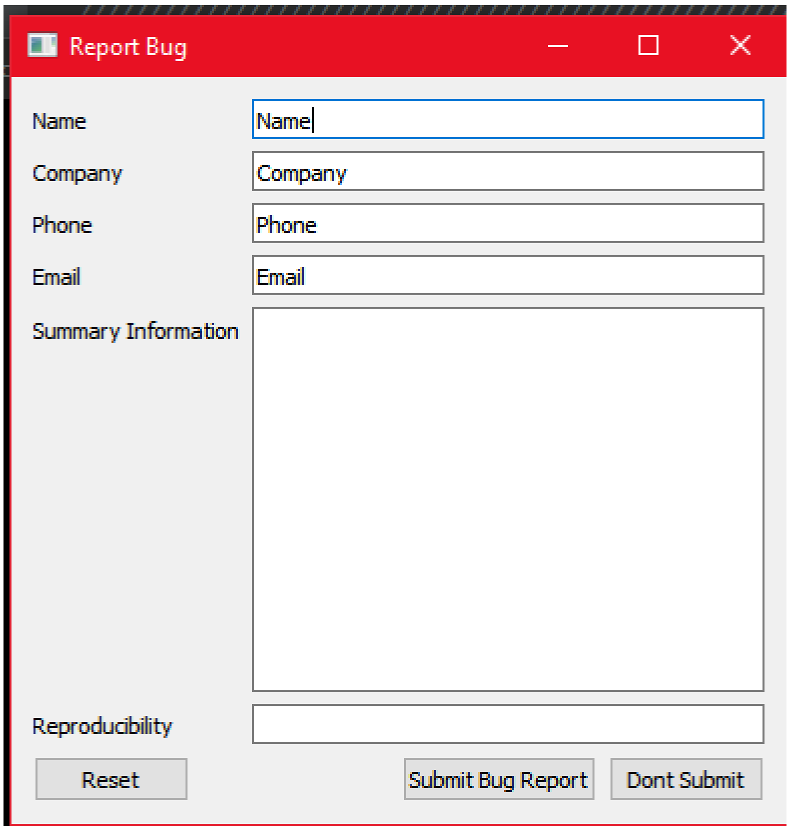
This window is a dialog to report a form. The components of this dialog are seven labels with 55

Five line edit, one text edit and three buttons. We can notice that it would be more efficient to create QLists for the different categories. We also have another kind of arrangement so that we can make each label before a text edit. We are using a QForm , QFormLayout is a convenient way to create two column form, where each row consists of an input field associated with a label, As a convention, the left column contains the label and the right column contains an input field. This is the code to display this window:



```
QWidget* window3=new QWidget();
window3->setWindowTitle("Report Bug");
QList <QLabel*> my_label={new QLabel("Name"),new QLabel("Company"),new QLabel("Phone"),new QLabel("Email"),new QLabel("Problem title")};

QList <QLineEdit*> my_list={new QLineEdit("Name"),new QLineEdit("Company"),new QLineEdit("Phone"),new QLineEdit("Email"),new QLineEdit("F
//QVBoxLayout* layout1=new QVBoxLayout();
QFormLayout* f=new QFormLayout();
for(int i=0;i<4;i++){


    f->addRow(my_label[i],my_list[i]);
}
QLabel* label1=new QLabel("Summary Information");
QTextEdit* summary=new QTextEdit();
f->addRow(label1,summary);
QLabel* reproducibility=new QLabel("Reproducibility");
QLineEdit* rep=new QLineEdit();
f->addRow(reproducibility,rep);

QList <QPushButton*> buttonlist={new QPushButton("Reset"),new QPushButton("Submit Bug Report"),new QPushButton("Dont Submit")};
QHBoxLayout* button_layout=new QHBoxLayout();
for(int i=0;i<3;i++){
    button_layout->addWidget(buttonlist[i]);
}
button_layout->insertSpacing(1,100);
QVBoxLayout* main_layout=new QVBoxLayout(window3);
main_layout->addLayout(f);
main_layout->addLayout(button_layout);
window3->show();
```

By executing this code, we display the following window:



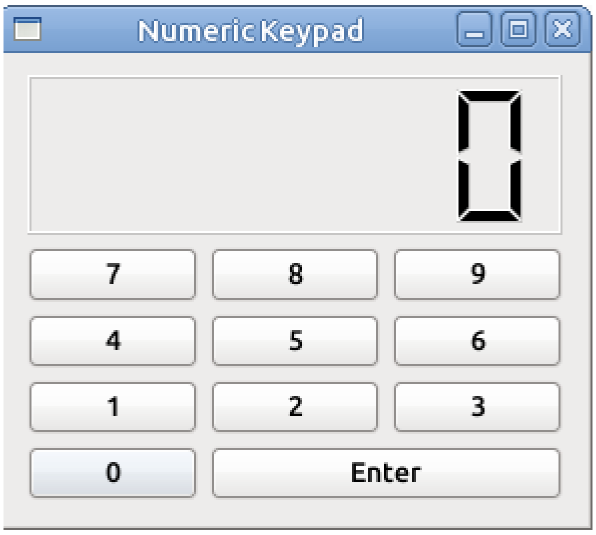
Grid Layout:

Overview

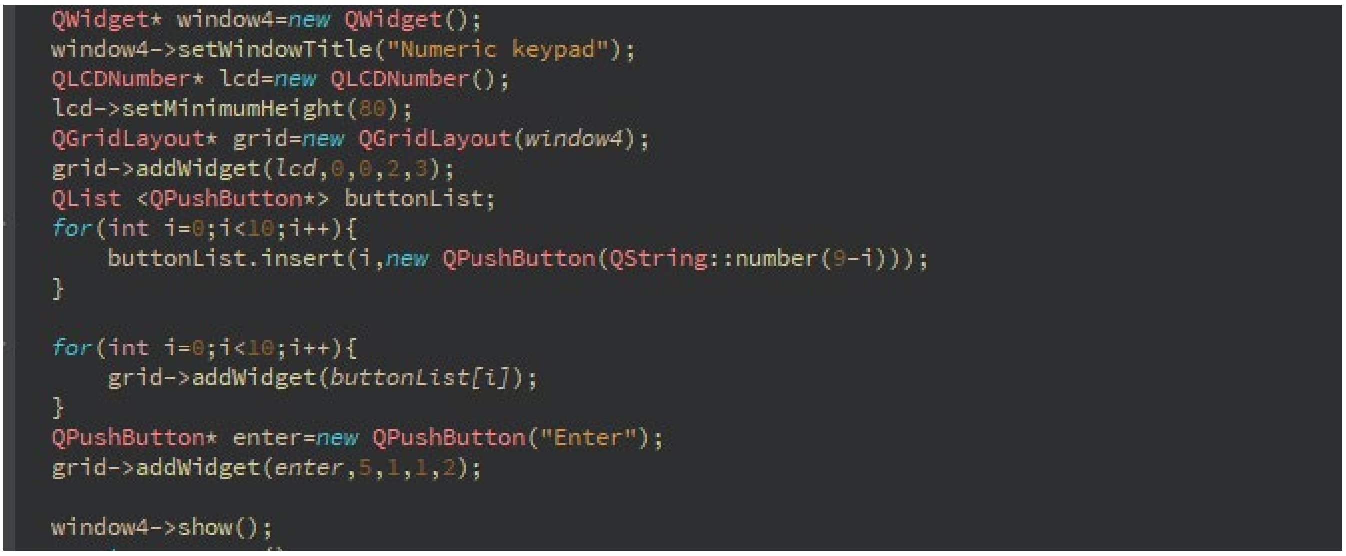
For our final exercise, we will visit an important layout that we missed in class. It is Grid Layouts. Grid Layout is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle. QGridLayout takes the space made available to it (by its parent layout or by the parentWidget ()), divides it up into rows and columns, and puts each widget it manages into the correct cell. This class offer many methods that facilitate the manipulation of this kind of forms.

Tasks

In this final exercise, we will be trying to display the following window:

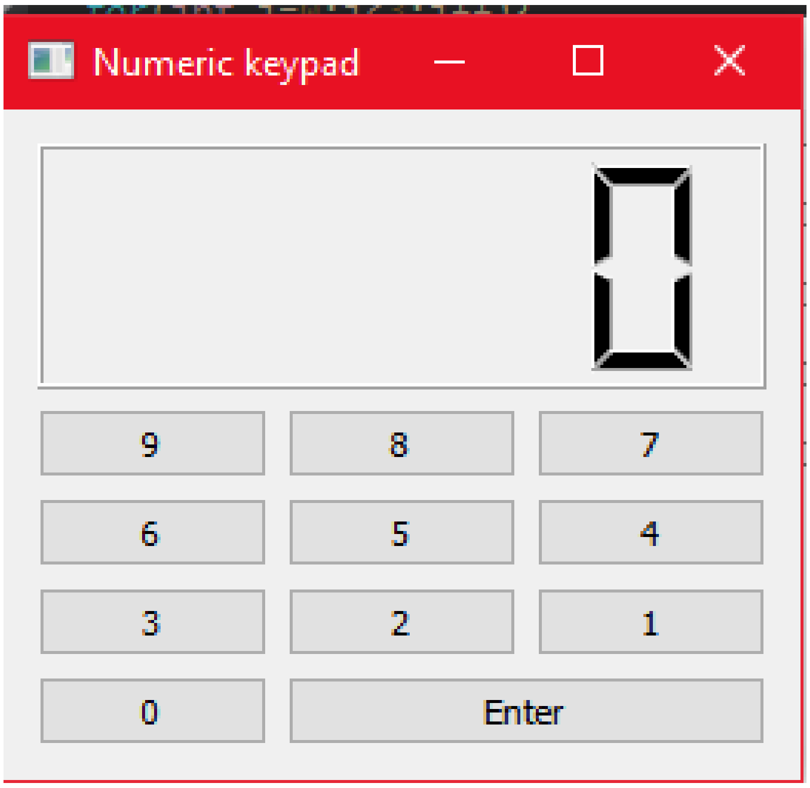


This window contains one component that shows the number which is called a LCD Number and it can be created from the class QLCDNumber. We can notice that again we have 10 similar buttons that can be created using a list. We also need to make the enter button larger than the other buttons.an other time that we do not have to code any functionality. We remind the following code is the modifying code to display a similar window:



```
QWidget* window4=new QWidget();
window4->setWindowTitle("Numeric keypad");
QLCDNumber* lcd=new QLCDNumber();
lcd->setMinimumHeight(80);
QGridLayout* grid=new QGridLayout(window4);
grid->addWidget(lcd,0,0,2,3);
QList <QPushButton*> buttonList;
for(int i=0;i<10;i++){
    buttonList.insert(i,new QPushButton(QString::number(9-i)));
}

for(int i=0;i<10;i++){
    grid->addWidget(buttonList[i]);
}
QPushButton* enter=new QPushButton("Enter");
grid->addWidget(enter,5,1,1,2);

window4->show();
```

And this is the result:



Conclusion:

In this session of work, we went through different types of Layouts and discovered the power and richness of the classes developed by QT that make the layout's manipulation easier. The layouts play an important role in any window. It is concerned as one of the essentials components of a window. The layout's classes offered large functionalities that are interesting to work with.[1]

[1] https://doc.qt.io/qt-5/layout.html

2 https://doc.qt.io/qt-5/qgridlayout.html

3 https://doc.qt.io/qt-5/

4 https://www.w3schools.com/css/css_grid.asp