Ministerul Educaţiei şi Tineretului al Republicii Moldova
Universitatea Tehnică a Moldovei

Facultatea Calculatoare Informatică şi Microelectronică
Catedra Automatica şi Tehnologii Informaţionale

Raport

MIDPS

Lucrarea de laborator Nr. 3

Tema: GUI Development

A efectuat: studentul grupei TI-141: **Buldumac Vasile**

A verificat: lector asistent: **Irina Cojanu**
lector superior: **Svetlana Cojocaru**

GUI Development

Conditii Necesare:

- IDEs: Visual Studio, QTCreator, Xcode, Code::Blocks
- Limbaje de programare: C/C++, C#, Objective C, Python, etc
- Technologii si Frameworks: Forms, wxWidgets, Win32 API, WinRT API, PyQt or others (depinde de IDE si limbajul de programare ales)
- Timp de lucru: 4-8 hours

Obiective:

- Realizeaza un simplu GUI Calculator
- Operatiile simple: +,-,*,/,putere,radical,InversareSemn(+/-),operatii cu numere zecimale.
- Divizare proiectului in doua module - Interfata grafica(Modul GUI) si Modulul de baza(Core Module).

Technical Prerequisites:

- Aplicatia trebuie sa fie divizata in doua module:
    - Core module - contine functionalitatile de baza
    - GUI module - include codul responsabil de crearea Interfetei Grafice si interactiunea ei cu elementele interfetei grafice si modulul de baza
- * Incearca sa realizezi programul tau sa fie cross platform (Compatibil cu diferite platforme: Windows,Linux,Mac).

Laboratory Requirements:

- *Basic Level* (nota 5 || 6):
    - Realizeaza un simplu GUI calculator care suporta functiile de baza: +, -, /, *.
- *Normal Level* (nota 7 || 8):
    - Realizeaza un simplu GUI calculator care suporta urmatoare functii: +, -, /, *, putere, radical, InversareSemn(+/-).
- *Advanced Level* (nota 9 || 10):
    - Realizeaza un simplu GUI calculator care suporta urmatoare functii: +, -, /, *, putere, radical, InversareSemn(+/-), operatii cu numere zecimale.
    - Divizare proiectului in doua module - Interfata grafica(Modul GUI) si Modulul de baza(Core Module).

IDE: PyCharm
OS: Ubuntu 14.04
Limbaj de programare: Python 2.7.6
Biblioteca folosită: PyQt4

PyQt este o extendie pentru limbajul de programare Python. Care ne permite lucru cu framework-ul grafic Qt, care este disponibil pe majoritatea sistememlor de operare populare (Linux, Windows, Mac).

Partea logică

Logica programului se îndeplinește în fișierul main.py care primește de la utilizator evenimentele de click pe butoane și afișarea rezultatlor.

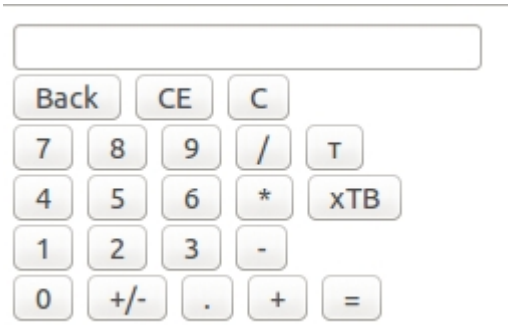
Partea View

Fișierul template.html răspunde de structura butoanelor în programm, valoarea butonului și culoarea font-ului butonului. Pentu acest fișier a fost elaborată o metodă aparte de parsing care va permite colectarea și convertarea datelor în date înțelese pentru PyQt.

```
def compile_template(self):
    content = open( 'template.html' ).readlines()
    self.structure = {}
    for line in content:
        line = line.strip()
        prepare = re.compile(r'<button type="(.+?)" name="(.+?)" color="(.+?)">(.+?)</button>')
        result = prepare.findall( line )
        if len( result ) > 0:
            values = result[0]
            self.structure.update({
                values[1]: [ values[0], values[2], values[3] ]
            })
```

Template.html (View)

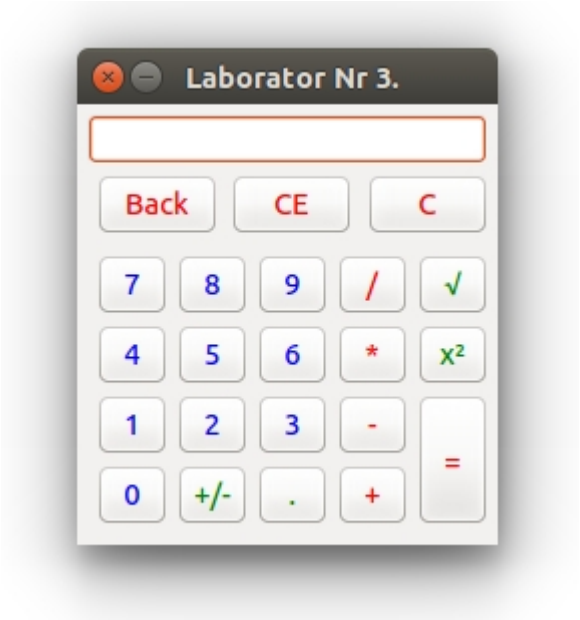
Fişierul template.html (sceletul programului).



Template.html – Source

```
<div>
    <input name="line">
</div>
<div>
    <button type="ops" name="back" color="red">Back</button>
    <button type="ops" name="ce" color="red">CE</button>
    <button type="ops" name="c" color="red">C</button>
</div>
<div>
    <button type="num" name="seven" color="blue">7</button>
    <button type="num" name="eight" color="blue">8</button>
    <button type="num" name="nine" color="blue">9</button>
    <button type="ops" name="div" color="red">/</button>
    <button type="ops" name="sqrt" color="green">√</button>
</div>
<div>
    <button type="num" name="four" color="blue">4</button>
    <button type="num" name="five" color="blue">5</button>
    <button type="num" name="six" color="blue">6</button>
    <button type="ops" name="mult" color="red">*</button>
    <button type="ops" name="squared" color="green">x²</button>
</div>
<div>
    <button type="num" name="one" color="blue">1</button>
    <button type="num" name="two" color="blue">2</button>
    <button type="num" name="three" color="blue">3</button>
    <button type="ops" name="minus" color="red">-</button>
</div>
<div>
    <button type="num" name="zero" color="blue">0</button>
    <button type="ops" name="switch" color="green">+/-</button>
    <button type="ops" name="point" color="green">.</button>
    <button type="ops" name="plus" color="red">+</button>
    <button type="ops" name="equal" color="red">=</button>
</div>
```

Front-end final



Source code: main.py

```
# -*- coding: utf-8 -*-
import sys
reload(sys)
sys.setdefaultencoding('utf8')
from PyQt4 import QtGui
from PyQt4.QtCore import Qt
from math import sqrt
import re
num = 0.0
newNum = 0.0
sumAll = 0.0
operator = ""
opVar = False
sumIt = 0
class Main(QtGui.QMainWindow):
    def __init__(self):
        QtGui.QMainWindow.__init__(self)
        self.initUI()
    def initUI(self):
        # Get buttons settings from HTML Template.
        self.compile_template()
        self.line = QtGui.QLineEdit(self)
        self.line.move(5,5)
        self.line.setReadOnly(True)
        self.line.setAlignment(Qt.AlignRight)
        self.line.resize(200,25)

        zero = QtGui.QPushButton( self.structure['zero'][2] ,self)
        zero.setStyleSheet("color: %s ;" % self.structure['zero'][1])
        zero.move(10,180)
        zero.resize(35,30)
```

```
one = QtGui.QPushButton( self.structure['one'][2] ,self)
one.setStyleSheet("color: %s ;" % self.structure['one'][1])
one.move(10,145)
one.resize(35,30)

two = QtGui.QPushButton( self.structure['two'][2] ,self)
two.setStyleSheet("color: %s ;" % self.structure['two'][1])
two.move(50,145)
two.resize(35,30)

three = QtGui.QPushButton( self.structure['three'][2] ,self)
three.setStyleSheet("color: %s ;" % self.structure['three'][1])
three.move(90,145)
three.resize(35,30)

four = QtGui.QPushButton( self.structure['four'][2] ,self)
four.setStyleSheet("color: %s ;" % self.structure['four'][1])
four.move(10,110)
four.resize(35,30)

five = QtGui.QPushButton( self.structure['five'][2] ,self)
five.setStyleSheet("color: %s ;" % self.structure['five'][1])
five.move(50,110)
five.resize(35,30)

six = QtGui.QPushButton( self.structure['six'][2] ,self)
six.setStyleSheet("color: %s ;" % self.structure['six'][1])
six.move(90,110)
six.resize(35,30)

seven = QtGui.QPushButton( self.structure['seven'][2] ,self)
seven.setStyleSheet("color: %s ;" % self.structure['seven'][1])
seven.move(10,75)
seven.resize(35,30)

eight = QtGui.QPushButton( self.structure['eight'][2] ,self)
eight.setStyleSheet("color: %s ;" % self.structure['eight'][1])
eight.move(50,75)
eight.resize(35,30)

nine = QtGui.QPushButton( self.structure['nine'][2] ,self)
nine.setStyleSheet("color: %s ;" % self.structure['nine'][1])
nine.move(90,75)
nine.resize(35,30)

switch = QtGui.QPushButton( self.structure['switch'][2] ,self)
switch.setStyleSheet("color: %s ;" % self.structure['switch'][1])
switch.move(50,180)
switch.resize(35,30)
switch.clicked.connect(self.Switch)

point = QtGui.QPushButton( self.structure['point'][2] ,self)
point.setStyleSheet("color: %s ;" % self.structure['point'][1])
point.move(90,180)
point.resize(35,30)
point.clicked.connect(self.pointClicked)
```

```
div = QtGui.QPushButton( self.structure['div'][2] ,self)
div.setStyleSheet("color: %s ;" % self.structure['div'][1])
div.move(130,75)
div.resize(35,30)

mult = QtGui.QPushButton( self.structure['mult'][2] ,self)
mult.setStyleSheet("color: %s ;" % self.structure['mult'][1])
mult.move(130,110)
mult.resize(35,30)

minus = QtGui.QPushButton( self.structure['minus'][2] ,self)
minus.setStyleSheet("color: %s ;" % self.structure['minus'][1])
minus.move(130,145)
minus.resize(35,30)

plus = QtGui.QPushButton( self.structure['plus'][2] ,self)
plus.setStyleSheet("color: %s ;" % self.structure['plus'][1])
plus.move(130,180)
plus.resize(35,30)

sqrt = QtGui.QPushButton( unicode( self.structure['sqrt'][2] ) ,self)
sqrt.setStyleSheet("color: %s ;" % self.structure['sqrt'][1])
sqrt.move(170,75)
sqrt.resize(35,30)
sqrt.clicked.connect(self.Sqrt)

squared = QtGui.QPushButton( unicode( self.structure['squared'][2] ) ,self)
squared.setStyleSheet("color: %s ;" % self.structure['squared'][1])
squared.move(170,110)
squared.resize(35,30)
squared.clicked.connect(self.Squared)

equal = QtGui.QPushButton( self.structure['equal'][2] ,self)
equal.setStyleSheet("color: %s ;" % self.structure['equal'][1])
equal.move(170,145)
equal.resize(35,65)
equal.clicked.connect(self.Equal)

c = QtGui.QPushButton( self.structure['c'][2] ,self)
c.setStyleSheet("color: %s ;" % self.structure['c'][1])
c.move(145,35)
c.resize(60,30)
c.clicked.connect(self.C)

ce = QtGui.QPushButton( self.structure['ce'][2] ,self)
ce.setStyleSheet("color: %s ;" % self.structure['ce'][1])
ce.move(77,35)
ce.resize(60,30)
ce.clicked.connect(self.CE)

back = QtGui.QPushButton( self.structure['back'][2] ,self)
back.setStyleSheet("color: %s ;" % self.structure['back'][1])
back.move(10,35)
back.resize(60,30)
back.clicked.connect(self.Back)

nums = [zero,one,two,three,four,five,six,seven,eight,nine]
ops = [back,c,ce,div,mult,minus,plus,equal]
rest = [switch,squared,sqrt,point]
```

```
        for i in nums:
            i.clicked.connect(self.Nums)

        for i in ops[3:7]:
            i.clicked.connect(self.Operator)

        self.setGeometry(300,300,210,220)
        self.setFixedSize(210,220)
        self.setWindowTitle("Laborator Nr 3.")
        self.setWindowIcon(QtGui.QIcon(""))
        self.show()

    def Nums(self):
        global num
        global newNum
        global opVar
        sender = self.sender()
        newNum = int(sender.text())
        setNum = str(newNum)

        if opVar == False:
            self.line.setText(self.line.text() + setNum)
        else:
            self.line.setText(setNum)
            opVar = False

    def pointClicked(self):
        global opVar
        if "." not in self.line.text():
            self.line.setText(self.line.text() + ".")

    def Switch(self):
        global num
        try:
            num = int(self.line.text())
        except:
            num = float(self.line.text())
        num = num - num * 2
        numStr = str(num)
        self.line.setText(numStr)

    def Operator(self):
        global num
        global opVar
        global operator
        global sumIt
        sumIt += 1
        if sumIt > 1:
            self.Equal()
        num = self.line.text()
        sender = self.sender()
        operator = sender.text()
        opVar = True
```

```
def Equal(self):
     global num
     global newNum
     global sumAll
     global operator
     global opVar
     global sumIt
     sumIt = 0
     newNum = self.line.text()
     print(num)
     print(newNum)
     print(operator)

     if operator == "+":
          sumAll = float(num) + float(newNum)
     elif operator == "-":
          sumAll = float(num) - float(newNum)
     elif operator == "/":
          sumAll = float(num) / float(newNum)
     elif operator == "*":
          sumAll = float(num) * float(newNum)
     print(sumAll)
     self.line.setText(str(sumAll))
     opVar = True

def Back(self):
     self.line.backspace()

def C(self):
     global newNum
     global sumAll
     global operator
     global num
     self.line.clear()
     num = 0.0
     newNum = 0.0
     sumAll = 0.0
     operator = ""

def CE(self):
     self.line.clear()

def Sqrt(self):
     global num
     num = float(self.line.text())
     n = sqrt(num)
     num = n
     self.line.setText(str(num))

def Squared(self):
     global num
     num = float(self.line.text())
     n = num ** 2
     num = n
     self.line.setText(str(n))
```

```
    def compile_template(self):
        content = open( 'template.html' ).readlines()
        self.structure = {}
        for line in content:
            line = line.strip()
            prepare = re.compile(r'<button type="(.+?)" name="(.+?)" color="(.+?)">(.
+?)</button>')
            result = prepare.findall( line )
            if len( result ) > 0:
                values = result[0]
                self.structure.update({
                    values[1]: [ values[0], values[2], values[3] ]
                })

def main():
    app = QtGui.QApplication(sys.argv)
    main = Main()
    main.show()
    sys.exit( app.exec_() )

if __name__ == "__main__":
    main()
```

Concluzie

În urma elaborării acestei lucrări de laborator am studiat documentația bibliotecii PyQt4. Am structurat programul într-o formă obiect-orientată pentru a permite manipularea mai rapidă a datelor. Am studiat cum are loc procesarea evenimentelor în PyQt. Am făcut o tangentă cu informația primită la cursurile WinAPI și studierea bibliotecii PyQt.

Bibliografie

http://zetcode.com/gui/pyqt4/
http://zetcode.com/ebooks/advancedpyqt4/