Как я писал калькулятор или мои опыты с QtE5

После того, как Геннадий Владимирович опубликовал у нас статью, я был весьма заинтригован этой графической библиотекой, однако, все равно несколько колебался перед ее использованием… Проведя пару экспериментов, а также воспроизведя ряд примеров из статьи про QtE, я решил все-таки попробовать написать что-нибудь самостоятельно просто для того, чтобы заставить себя перейти на новый тулкит, а заодно пощупать его возможности.

Так мне пришла в голову мысль повторить «подвиг», написав калькулятор, но уже на D и Qt5!

Начну с того, что я вообще незнаком с Qt и его концепциями, хоть и когда-то пытался познакомится с ним. Однако, я уверен, что незнание Qt5 не помеха, поскольку во Всемирной Паутине есть куча примеров по этому замечательному тулкиту, а также есть вполне исчерпывающая документация по API, доступная вот здесь.

Первым делом, убеждаемся, что libQtE5Widgets64.so (или QtE5Widgets64.dll) в наличии (предполагается, что у вас, также как и у меня 64-разрядная операционная система) и если это не так, то скачиваем и собираем эти библиотеки.

Мне повезло, в том, что у меня оказалась абсолютно чистая Kubuntu 16.04 и один из наших гостей описал подробную процедуру по сборке QtE5 на 64-разрядном Linux, которой я воспользовался. Помимо этого, перед тем, как выполнить компиляцию пришлось установить кое-что из дополнительного программного обеспечения с помощью команды:

```
1. sudo apt-get install qt5-default qtcreator cmake
```

Далее, пришлось немного помучиться с компиляцией проекта в QtCreator, которая проходит по инструкции Виталия Колыванова, а вот установку скомпилированной libQtE5Widgets64.so.1.0.0 я провел несколько иначе: сначала я ее переименовал в libQtE5Widgets64.so, а затем просто скопировал в папку /lib системы — как это ни странно, но этого действия оказалось чуть более чем достаточно.

Теперь, все что нам нужно — это файл qte5.d из стандартной поставки QtE5 и чистый файл app.d.

Два этих файла мы объединим в один простой проект в среде Monodevelop с MonoD, воспользовавшись опцией создания пустого проекта (как это сделать, я уже описывал в одной из статей) и получим тем самым возможность автоматического дополнения кода и удобный список всех функций, который на настоящий момент содержаться в QtE.

На этом подготовка к разработке калькулятора закончена и мы плавно переходим к процессу его создания.

Для начала определимся какие элементы интерфейса нам потребуется, чтобы воплотить простейший калькулятор, который будет считать только в целых числах (в основном из-за чисто технического ограничения, однако, это для нас не так уж важно).

Вполне очевидно, что для калькулятора потребуется дисплей, роль которого сыграет элемент QLCDNumber; несколько кнопок под числа и операции, которые можно реализовать с помощью элемента управления QPushButton; а также потребуется некоторое количество так называемых «сайзеров» или «выравнивателей» для размещения внутри них элементов: QVBoxLayout — для вертикального размещения элементов внутри себя и QHBoxLayout — для горизонтального размещения элементов.

Я очень надеюсь, что вы перед прочтением этой статьи хотя бы просто пробежались глазами по статье «QtE5 — изучаем D и Qt5 в комфортной графической среде», поскольку дизайн приложения в QtE5 очень сложен (по крайней мере, мне так до сих пор кажется), а объяснить его в нескольких словах я вряд ли смогу и кроме того, дальше идет довольно таки сложный код размещения элементов…

Помимо самих элементов интерфейса, нам потребуется такой элемент, внутри которого мы могли бы с уверенностью размещать нужные элементы управления, такие как кнопки и др. К счастью, в Qt5 есть такой элемент и называется он QWidget, и именно от этого элемента нам и нужно осуществить наследование для того, чтобы создать окно со своими собственными элементами GUI:

```
1. alias WindowType = QtE.WindowType;
2. enum WHITE = "background : white;";
3. class MainForm : QWidget
4. {
5.    this(QWidget parent, WindowType windowType)
6.    {
7.      super(parent, windowType);
8.      resize(300, 400);
9.      setWindowTitle("QtE Calculator");
10.     setStyleSheet(WHITE);
11.      // ...
12.   }
13. }
```

Примерно такую заготовку мы будем использовать в нашем калькуляторе.

Вначале мы создаем псевдоним для QtE.WindowType чисто для упрощения кода и для простоты его прочтения, после чего определяем производный от QWidget класс MainForm. В классе MainForm определяем стандартный конструктор, принимающий два параметра — элемент, который является «родителем» (родителем в плане системы вызовов, а не наследования) и тип

окна. Внутри конструктора тоже не происходит ничего необычного: класс MainForm обращается к родительскому конструктору посредством super, затем выставляет свой собственный размер (300 пикселей в длину и 400 в высоту), а затем выводит в область заголовка некоторую надпись (в данном случае, этой надписью является надпись — «QtE Calculator»). А вот следующая инструкция, а именно инструкция setStyleSheet(WHITE) уже является необычной. Дело в том, что эта инструкция одинакова применима к большинству элементов графического интерфейса QtE и позволяет используя знания HTML и CSS задавать внешний вид некоторого элемента, в данном случае — обычного окна. Перечисление WHITE также является одним их элементов стратегии упрощения кода и задает обычную строковую константу, содержащую инструкцию по окраске фона в белый цвет.

Теперь, мы можем перейти ко внутреннему содержимому класса MainForm и для начала мы определим почти весь набор элементов интерфейса:

```
1. alias WindowType = QtE.WindowType;
2. enum WHITE = "background : white;";
3. class MainForm : QWidget
4. {
5.         QVBoxLayout verticalSizer, verticalSizer1, buttonGroup5;
6.         QHBoxLayout horizontalSizer, buttonGroup1, buttonGroup2, buttonGroup3,
   buttonGroup4;
7.         QPushButton button0, button1, button2, button3,
8.                 button4, button5, button6,
9.                 button7, button8, button9,
10.                sign, clear,
11.                add, subtract, multiply,divide, equal;
12.        this(QWidget parent, WindowType windowType)
13.        {
14.                super(parent, windowType);
15.                resize(300, 400);
16.                setWindowTitle("QtE Calculator");
17.                setStyleSheet(WHITE);
18.           // ...
19.        }
20. }
```

Почти весь (а это кнопки для чисел, операций и кнопки вычисления), потому что один из элементов интерфейса мы определим как глобальную переменную (да, да, я знаю, что это нехорошо, но иных вариантов не было), а именно — QLCDNumber:

```
1. QLCDNumber lcd;
```

После этого можно настроить элементы управления, согласно нашим нуждам, и начнем мы пожалуй, с LCD-дисплея, стилизовав его под дисплей обычных карманных калькуляторов:

```
1. lcd = new QLCDNumber(this);
2. lcd.setMode(QLCDNumber.Mode.Dec);
3. lcd.setStyleSheet("background : lightgreen; color : gray;");
4. lcd.display(0);
```

Этот код должен быть размещен непосредственно внутри конструктора класса MainForm, поскольку является кодом инициализации. Однако, требуется немного пояснить, что делает этот набор методов объекта lcd. Метод setMode устанавливает режим отображения значений на LCD-индикаторе и принимает в качестве аргумента константу (или перечисление, точно не скажу), которая размещена прямо в этом же классе. В данном случае, колнстанта QLCDNumber.Mode.Dec устанавливается десятичный режим отображения чисел, хотя ради интереса можно поставить что-нибудь более интригующее, к примеру. шестнадцатеоричный режим отображения (константа QLCDNumber.Mode.Hex). После установки правильного режима отображения, мы настраиваем стиль самого дисплея с помощью CSS (я ж говорил, что метод setStyleSheet будет нам часто попадаться!), после чего используем самый главный для нас метод — метод display, который позволяет вывести на lcd некоторое целое число.

К сожалению, именно из-за display наш калькулятор будет считать только в целых числах :(LCD-дисплей, не единственный элемент интерфейса, и это значит, что необходимо подумать о размещении элементов в окне приложения.

Дело в том, что элементы интерфейса в Qt5 можно разместить в окне приложения с помощью т.н. сайзеров (sizers) или, как их называет разработчик QtE5, выравнивателей.

Сайзеров в QtE5 два типа: QVBoxLayout — вертикальный и QHBoxLayout — горизонтальный выравниватель. В них мы и будем размещать кнопки, используя для этого методы, определенные для обоих типов сайзеров: addWidget, размеющающий элементы управления, и addlayout, размещающий уже сами сайзеры.

Создадим сами элементы GUI и настроим их свойства, после чего создадим сайзеры и поместим в них один за другим элементы (просто следите за кодом, иначе никак это и не объяснить :D). Само размещение будем проводить, используя замечательную конструкцию языка программирования D под названием with:

```
1. alias WindowType = QtE.WindowType;
2. enum WHITE = "background : white;";
3. class MainForm : QWidget
4. {
5.         QVBoxLayout verticalSizer, verticalSizer1, buttonGroup5;
6.         QHBoxLayout horizontalSizer, buttonGroup1, buttonGroup2, buttonGroup3,
```

```
buttonGroup4;
        QPushButton button0, button1, button2, button3,
                button4, button5, button6,
                button7, button8, button9,
                sign, clear,
                add, subtract, multiply,divide, equal;
        this(QWidget parent, WindowType windowType)
        {
                super(parent, windowType);
                resize(300, 400);
                setWindowTitle("QtE Calculator");
                setStyleSheet(WHITE);
                lcd = new QLCDNumber(this);
                lcd.setMode(QLCDNumber.Mode.Dec);
                lcd.setStyleSheet("background: lightgreen; color : gray;");
                lcd.display(0);
                verticalSizer = new QVBoxLayout;
                verticalSizer1 = new QVBoxLayout;
                horizontalSizer = new QHBoxLayout;
                with (buttonGroup1 = new QHBoxLayout)
                {
                        sign = new QPushButton("+/-", this);
                        button0 = new QPushButton("0", this);
                        clear = new QPushButton("C", this);
                        QAction action0 = new QAction(null, &onButton0, null);
                        connects(button0, "clicked()", action0, "Slot()");
                        QAction actionSign = new QAction(null, &onSignButton,
null);
                        connects(sign, "clicked()", actionSign, "Slot()");
                        QAction actionClear = new QAction(null,
&onClearButton, null);
                        connects(clear, "clicked()", actionClear, "Slot()");
                        addWidget(sign);
                        addWidget(button0);
                        addWidget(clear);
                }
                with (buttonGroup2 = new QHBoxLayout)
                {
                        button1 = new QPushButton("1", this);
                        button2 = new QPushButton("2", this);
                        button3 = new QPushButton("3", this);
                        QAction action1 = new QAction(null, &onButton1, null);
                        connects(button1, "clicked()", action1, "Slot()");
                        QAction action2 = new QAction(null, &onButton2, null);
```

```
48.                         connects(button2, "clicked()", action2, "Slot()");
49.                         QAction action3 = new QAction(null, &onButton3, null);
50.                         connects(button3, "clicked()", action3, "Slot()");
51.                         addWidget(button1);
52.                         addWidget(button2);
53.                         addWidget(button3);
54.                 }
55.                 with (buttonGroup3 = new QHBoxLayout)
56.                 {
57.                         button4 = new QPushButton("4", this);
58.                         button5 = new QPushButton("5", this);
59.                         button6 = new QPushButton("6", this);
60.                         QAction action4 = new QAction(null, &onButton4, null);
61.                         connects(button4, "clicked()", action4, "Slot()");
62.                         QAction action5 = new QAction(null, &onButton5, null);
63.                         connects(button5, "clicked()", action5, "Slot()");
64.                         QAction action6 = new QAction(null, &onButton6, null);
65.                         connects(button6, "clicked()", action6, "Slot()");
66.                         addWidget(button4);
67.                         addWidget(button5);
68.                         addWidget(button6);
69.                 }
70.                 with (buttonGroup4 = new QHBoxLayout)
71.                 {
72.                         button7 = new QPushButton("7", this);
73.                         button8 = new QPushButton("8", this);
74.                         button9 = new QPushButton("9", this);
75.                         QAction action7 = new QAction(null, &onButton7, null);
76.                         connects(button7, "clicked()", action7, "Slot()");
77.                         QAction action8 = new QAction(null, &onButton8, null);
78.                         connects(button8, "clicked()", action8, "Slot()");
79.                         QAction action9 = new QAction(null, &onButton9, null);
80.                         connects(button9, "clicked()", action9, "Slot()");
81.                         addWidget(button7);
82.                         addWidget(button8);
83.                         addWidget(button9);
84.                 }
85.                 with (buttonGroup5 = new QVBoxLayout)
86.                 {
87.                         add = new QPushButton("+", this);
88.                         subtract = new QPushButton("-", this);
89.                         multiply = new QPushButton("*", this);
90.                         divide = new QPushButton("/", this);
91.                         QAction actionAdd = new QAction(null, &onAddButton,
```

```
null);
                        connects(add, "clicked()", actionAdd, "Slot()");
                        QAction actionSubtract = new QAction(null,
&onSubtractButton, null);
                        connects(subtract, "clicked()", actionSubtract,
"Slot()");
                        QAction actionMultiply = new QAction(null,
&onMultiplyButton, null);
                        connects(multiply, "clicked()", actionMultiply,
"Slot()");
                        QAction actionDivide = new QAction(null,
&onDivideButton, null);
                        connects(divide, "clicked()", actionDivide, "Slot()");
                        addWidget(add);
                        addWidget(subtract);
                        addWidget(multiply);
                        addWidget(divide);
                }
                equal = new QPushButton("=", this);
                QAction actionEqual = new QAction(null, &onEqualButton, null);
                connects(equal, "clicked()", actionEqual, "Slot()");
                verticalSizer1
                        .addLayout(buttonGroup4)
                                .addLayout(buttonGroup3)
                                .addLayout(buttonGroup2)
                                .addLayout(buttonGroup1);
                horizontalSizer
                        .addLayout(verticalSizer1)
                                .addLayout(buttonGroup5);
                verticalSizer
                        .addWidget(lcd)
                                .addLayout(horizontalSizer)
                                .addWidget(equal);
                setLayout(verticalSizer);
        }
}
```

Помимо определения свойств кнопок, мы также осуществим их «привязку» к обработчикам событий с помощью объекта QAction и функции connects, связывающей слоты и сигналы самих событий (они определены внутри самой Qt5, а также в файле qte5, созданном Геннадием Владимировичем Моховым). Подробнее об этом, вы можете прочесть в предыдущих статьях про QtE5.

Кроме этого, после размещения элементов и сайзеров (делаем это последовательно!), мы размещаем последний сайзер прямо в главном окне с помощью метода setLayout класса MainForm. По приведенному коду, видно, что размещение и настройка элементов очень просты, как в общем, и их создание — в ходе создания, мы просто передаем нужному объекту указатель на главный элемент (parent, в нашем случае, это форма MainForm) и некоторый параметр-описатель, которым может быть текст кнопки или какой-либо другой адекватный описатель.

Однако, приведенный код еще не является завершенным, поскольку мы еще не описали логику калькулятора и события, связывающие уже размещенные на форме кнопки и LCD-дисплей. Определим несколько глобальных переменных, которые будут накапливать цифры аргументов операций сложения, вычитания, умножения и деления, а также определим строку, хранящую результат операции (необходимо для реализации некоторого поведения обычного карманного калькулятора) и числовую переменную, которая сохранит итог операции, пригодный для отображения на экране:

```
1. int result;
2. string resultRegister, numberRegister;
3. string operationSign;
```

Теперь опишем реакцию цифровых кнопок и кнопки смены знака числа с помощью событий, определенных следующим образом:

```
1. extern(C)
2. {
3.         void updateLCD(string number)
4.         {
5.                 numberRegister ~= number;
6.                 lcd.display(to!int(numberRegister));
7.                 lcd.update;
8.         }
9.         void setOperationSign(string sign)
10.        {
11.                resultRegister = numberRegister;
12.                operationSign = sign;
13.                numberRegister = "";
14.                result = 0;
15.        }
16.        void onButton0(void* button)
17.        {
18.                updateLCD("0");
19.        }
20.        void onButton1(void* button)
21.        {
```

```
22.                 updateLCD("1");
23.         }
24.         void onButton2(void* button)
25.         {
26.                 updateLCD("2");
27.         }
28.         void onButton3(void* button)
29.         {
30.                 updateLCD("3");
31.         }
32.         void onButton4(void* button)
33.         {
34.                 updateLCD("4");
35.         }
36.         void onButton5(void* button)
37.         {
38.                 updateLCD("5");
39.         }
40.         void onButton6(void* button)
41.         {
42.                 updateLCD("6");
43.         }
44.         void onButton7(void* button)
45.         {
46.                 updateLCD("7");
47.         }
48.         void onButton8(void* button)
49.         {
50.                 updateLCD("8");
51.         }
52.         void onButton9(void* button)
53.         {
54.                 updateLCD("9");
55.         }
56.         void onAddButton(void* button)
57.         {
58.                 setOperationSign("+");
59.         }
60.         void onSubtractButton(void* button)
61.         {
62.                 setOperationSign("-");
63.         }
64.         void onMultiplyButton(void* button)
65.         {
```

```
66.                 setOperationSign("*");
67.         }
68.         void onDivideButton(void* button)
69.         {
70.                 setOperationSign("/");
71.         }
72.         void onClearButton(void* button)
73.         {
74.                 numberRegister = "0";
75.                 lcd.display(0);
76.                 lcd.update;
77.         }
78.         void onSignButton(void* button)
79.         {
80.                 numberRegister = "-" ~ numberRegister;
81.                 lcd.display(to!int(numberRegister));
82.                 lcd.update;
83.         }
84. }
```

Работает это очень просто: мы будем прицеплять цифры к числу, отображенному в данный момент на калькуляторе с помощью оператора конкатенации строк. Вспомогательные функции updateLCD и setOperationSign помогут грамотно вывести на дисплей число-результат и корректно установить знак математической операции.

Помимо этого, функция setOperationSign и соответствующие операции смогут отследить выбранную пользователем математическую операцию (сохраняя знак операции в переменной operationSign) или отследить сброс текущего значения на дисплее.

Теперь осталось только опознать математическую операцию, выбранную пользователем, а также обновить значение на дисплее, например вот так:

```
1. void onEqualButton(void* button)
2.         {
3.                 switch (operationSign)
4.                 {
5.                         case "+":
6.                                 result = to!int(resultRegister) +
   to!int(numberRegister);
7.                                 numberRegister = to!string(result);
8.                                 break;
9.                         case "-":
10.                                result = to!int(resultRegister) -
```

```
    to!int(numberRegister);
                                numberRegister = to!string(result);
                                break;
                        case "*":
                                result = to!int(resultRegister) *
    to!int(numberRegister);
                                numberRegister = to!string(result);
                                break;
                        case "/":
                                result = to!int(resultRegister) /
    to!int(numberRegister);
                                numberRegister = to!string(result);
                                break;
                        default:
                                numberRegister = resultRegister;
                                break;
                }
                lcd.display(result);
                lcd.update;
        }
```

Соответственно, необходимо приведенный обработчик события разместить внутри блока extern(C), для того, чтобы Qt5 смогла правильно среагировать на нажатие кнопки «равно». В этом обработчике просто прооисходит перевод строкового накопителя цифр в целое число, а затем использование его в качестве второго аргумента в математической операции (первый аргумент, как вы уже поняли, это переменная result, уже содержащая первое введенное число), обозначение которой уже было помещено в operationSign с помощью соотвествующих кнопкам процедур обработки.

Следующим шагом будет размешение в файле следующей, немного переработанной функции main:

```
int main(string[] args)
{
        alias normalWindow = QtE.WindowType.Window;
        if (LoadQt(dll.QtE5Widgets, true))
        {
                return 1;
        }
        QApplication app = new QApplication(&Runtime.cArgs.argc,
    Runtime.cArgs.argv, 1);
        MainForm mainForm = new MainForm(null, normalWindow);
        mainForm
```

```
11.                 .show
12.                         .saveThis(&mainForm);
13.         return app.exec;
14. }
```

Внутри main мы создаем для удобства чтения кода псевдоним на тип окна QtE.WindowType.Window, а также производим проверку того, насколько успешно подгрузилась Qt5. Также мы создаем экземпляр класса QApplication (экземпляр приложения на Qt, входная точка в приложение), осуществляем захват переданных программе аргументов (в случае, если таковые последуют), после чего происходит создание экземпляра MainForm (первый параметр — родительский элемент, которого в данном случае нет, поскольку окно само по себе будет первым в иерархии элементов графического интерфейса; второй параметр — тот самый тип окна).
Затем, мы просто отображаем созданное окно и сохраняем на него указатель для Qt5, и просто запускаем.

Сохраняем в файл и компилируем, запускаем, получая вот такой результат:

Большое спасибо хочу сказать разработчику этой поистине великолепной привязки к Qt5, Мохову Геннадию Владимировичу, за его помощь при обучении работе с QtE5 и за очень ценные советы, данные в ходе написания этой статьи!

В следующей статье, я и автор QtE5, расскажем кое-что интересное и познавательное о QtE5, показав небольшой практический экзерсис в математической графике.