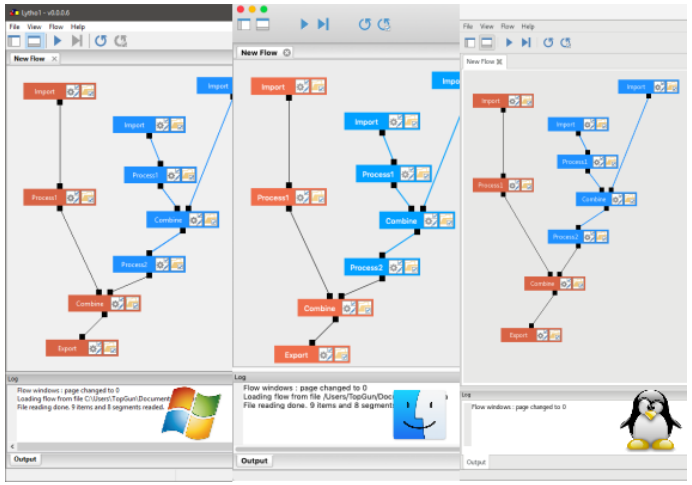
Национальный исследовательский университет «МИЭТ»

Институт интегральной электроники (группы ЭН-34-35, каф. ПКИМС)

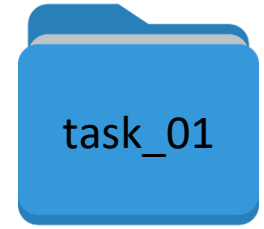
Кроссплатформенная разработка программного обеспечения



Лабораторная работа №6

Обработка текста

Работа с классом QString




```cpp
#include <QString>

int main() {
  QString str = "This is a TEXT";

  qDebug() << str;

  return 0;
}
```

```cpp
QString str = QString("%1 is a %2").arg("This").arg("TEXT");
```

```cpp
QString str = QString("%2 is a %1").arg("This").arg("TEXT");
```

Преобразования строк (1)

```
QString str = "This is a TEXT";

qDebug() << str.toLower();

qDebug() << str.toUpper();
```

```
qDebug() << str.isLower();

qDebug() << str.isUpper();
```

```
QString str = "x > y";

qDebug() << str.toHtmlEscaped();
```

```
QString str = "This is a TEXT";

std::string s  = str.toStdString();
std::wstring ws = str.toStdWString();
```

Преобразования строк (2)

```
QString str = "1024";

int N = str.toInt();
```

```
float f = str.toFloat();
double d = str.toDouble();
unsigned int i = str.toUInt();
```

```
QString str = QString::number(1);
QString str = QString::number(1.14);
```

```
QString str = "abc";
bool done = false;
qDebug() << str.toInt(&done);
```

```
QString str = "0xFA";
bool done = false;
qDebug() << str.toInt(&done, 16);
```

```
QString str = "11";
bool done = false;
qDebug() << str.toInt(&done, 10);

qDebug() << str.toInt(&done, 2);

qDebug() << str.toInt(&done, 19);
```

Основы регулярных выражений (1)

Классы символов:

`abcd` – литералы

`[abcd]` – один из символов
`[^abc]` – один не из символов

`\w` – один символ, одна буква, знак подчёркивания «_»
`\W` – один не символ, одна не буква и не знак подчёркивания «_»
`\s` – один пробельный символ ([\t\n\r\f\v])
`\S` – один не пробельный символ ([\t\n\r\f\v])
`\d` – одна цифра
`\D` – одна не цифра

`\b` – граница слова
`\B` – не граница слова

`.` – один любой символ

Основы регулярных выражений (2)

Якоря:

- `^` – начало строки
- `$` – конец строки

Итераторы:

- `?` – максимум одно повторение
- `+` – как минимум одно повторение
- `*` – любое число повтрений

Квантификаторы:

- `{n}` – ровно n повторений
- `{n,}` – как минимум n повторений
- `{, m}` – максимум m повторений
- `{n, m}` – не менее n, но не более m повторений

Обработка текста на основе регулярных выражений (1)

```
QString str = QString("ip address is : 192.168.254.17");

QRegExp re("\\d+\\.\\d+\\.\\d+\\.\\d+");
```

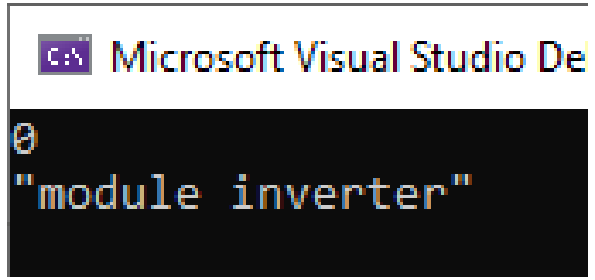
```
"\\d{,3}\\.\\d{,3}\\.\\d{,3}\\.\\d{,3}"

qDebug() << str.contains(re);
```

```
QString str = QString("ip address is : 192.168.254.17");

QRegExp re("\\d{,3}\\.\\d{,3}\\.\\d{,3}\\.\\d{,3}");

int pos = re.indexIn(str);

if(pos != -1)
    qDebug() << pos;
```

```
if (pos != -1) {
    qDebug() << pos;
    qDebug() << re.cap(0);
}
```

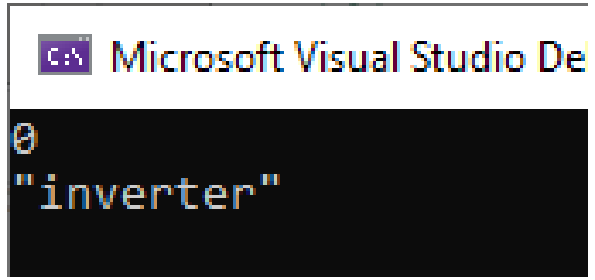
Обработка текста на основе регулярных выражений (2)

```
QString str = QString("module inverter (input x, output y);");

QRegExp re("module \\w+");

int pos = re.indexIn(str);


if (pos != -1) {
    qDebug() << pos;
    qDebug() << re.cap(0);
}
```



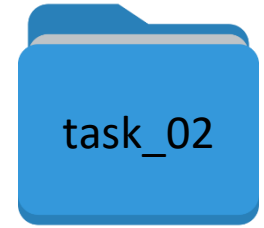
Обработка текста на основе регулярных выражений (2)

```
QString str = QString("module inverter (input x, output y);");

QRegExp re("module (\\w+)");

int pos = re.indexIn(str);


if (pos != -1) {
    qDebug() << pos;
    qDebug() << re.cap(1);
}
```




```
QRegExp re("^\\s*module\\s+(\\w+)");
```

Текстовый редактор с подсветкой синтаксиса (1)




```
#include <QTextEdit>

#include "Highlighter.hpp"

class MainWindow : public QMainWindow {
    Q_OBJECT
private:
    QTextEdit           *codeEditor;
    Highlighter         *syntaxHighlighter;
public:
    MainWindow(QWidget* parent);
   ~MainWindow();
private:
    void initCodeEditor();
private slots:
};
```

Текстовый редактор с подсветкой синтаксиса (2)

```
MainWindow::MainWindow(Qwidget *parent) : QMainWindow(parent) {
    setWindowTitle("KRPO_Lab_0x06");
    resize(800, 600);

    initCodeEditor();
}

void MainWindow::initCodeEditor() {
    codeEditor = new QTextEdit(this);
    setCentralWidget(codeEditor);

    QFont font;
    font.setFamily("Consolas");
    font.setFixedPitch(true);
    font.setPointSize(10);
    codeEditor->setFont(font);

    syntaxHighlighter = new Highlighter(codeEditor->document());
}
```

Текстовый редактор с подсветкой синтаксиса (3)

```
#include <QSyntaxHighlighter>
#include <QRegularExpression>

class Highlighter : public QSyntaxHighlighter {
    Q_OBJECT
public:
    Highlighter(QTextDocument *parent);
protected:
    void highlightBlock(const QString &text);
private:
    struct HighlightingRule {
        QRegularExpression    pattern;
        QTextCharFormat       format;
    };
    QVector<HighlightingRule> highlightingRules;
};
```

Текстовый редактор с подсветкой синтаксиса (4)

```
Highlighter::Highlighter(QTextDocument *parent) : QSyntaxHighlighter(parent) {
}

void Highlighter::highlightBlock(const QString &text) {

    for(int i = 0; i < highlightingRules.size(); ++i) {
        HighlightingRule &rule = highlightingRules[i];
        QRegularExpressionMatchIterator matchIterator = rule.pattern.globalMatch(text);

        while (matchIterator.hasNext()) {
            QRegularExpressionMatch match = matchIterator.next();
            setFormat(match.capturedStart(), match.capturedLength(), rule.format);
        }
    }

}
```

Написание правил (1)

```
Highlighter::Highlighter(QTextDocument *parent) : QSyntaxHighlighter(parent) {
    HighlightingRule rule;

    rule.pattern = QRegularExpression("\\bmodule\\b");

    rule.format.setForeground(Qt::blue);
    rule.format.setFontWeight(QFont::Bold);

    highlightingRules.append(rule);
}
```

```
struct HighlightingRule {
    QRegularExpression     pattern;
    QTextCharFormat        format;
};
```

```
QVector<HighlightingRule>
                    highlightingRules;
```

Написание правил (2)

```
const QString keywordPatterns[] = {
  QString("\\bchar\\b"),     QString("\\bclass\\b"),
  QString("\\bconst\\b"),    QString("\\bdouble\\b"),
  QString("\\benum\\b"),     QString("\\bexplicit\\b"),
  QString("\\bfriend\\b"),   QString("\\binline\\b"),
  QString("\\bint\\b"),      QString("\\bbool\\b"),
  ...
};
```

```
QTextCharFormat keywordFormat;
keywordFormat.setForeground(QColor(0, 0, 255));
keywordFormat.setFontWeight(QFont::Bold);

for (const QString &pattern : keywordPatterns) {
  rule.pattern = QRegularExpression(pattern);
  rule.format = keywordFormat;
  highlightingRules.append(rule);
}
```

Загрузка данных в текстовый редактор

```
void MainWindow::onMenuFileOpen() {

    QString fileName = QFileDialog::getOpenFileName(this);
    if (fileName.isEmpty())
        return;

    QFile file(fileName);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
        return;

    codeEditor->setPlainText(file.readAll());
    file.close();


   codeEditor->moveCursor(QTextCursor::Start);
   codeEditor->ensureCursorVisible();
}
```

Обработка событий редактора (1)

```
void MainWindow::initStatusBar() {
    statusBar = new QStatusBar(this);
    setStatusBar(statusBar);

    cursorPosLabel = new QLabel("0:0");
    statusBar->addPermanentWidget(cursorPosLabel);
}
```

Обработка событий редактора (2)

```
class MainWindow : public QMainWindow {
...
private slots:
    void onCursorPositionChanged();
```

```
void MainWindow::initCodeEditor() {
  codeEditor = new QTextEdit(this);
  setCentralWidget(codeEditor);

  connect(codeEditor,
          SIGNAL(cursorPositionChanged()),
          this,
          SLOT(onCursorPositionChanged()));
```

Обработка событий редактора (3)

```
void MainWindow::onCursorPositionChanged() {

    cursorPosLabel->setText(
        QString("%1:%2").arg(codeEditor->textCursor().blockNumber())
                        .arg(codeEditor->textCursor().positionInBlock()));

}
```

Обработка событий редактора (4)

```
class MainWindow : public QMainWindow {
...
private slots:
    void onTextChanged();
```

```
void MainWindow::initCodeEditor() {
  codeEditor = new QTextEdit(this);
  setCentralWidget(codeEditor);

  connect(codeEditor,
          SIGNAL(textChanged()),
          this,
          SLOT(onTextChanged()));
```

Обработка событий редактора (5)

```
void MainWindow::onTextChanged() {

    setWindowTitle("KRPO_Lab_0x06 *");

}
```

```
void MainWindow::onMenuFileOpen() {
    QString fileName = QFileDialog::getOpenFileName(this);
    ...
    setWindowTitle("KRPO_Lab_0x06");
}
```