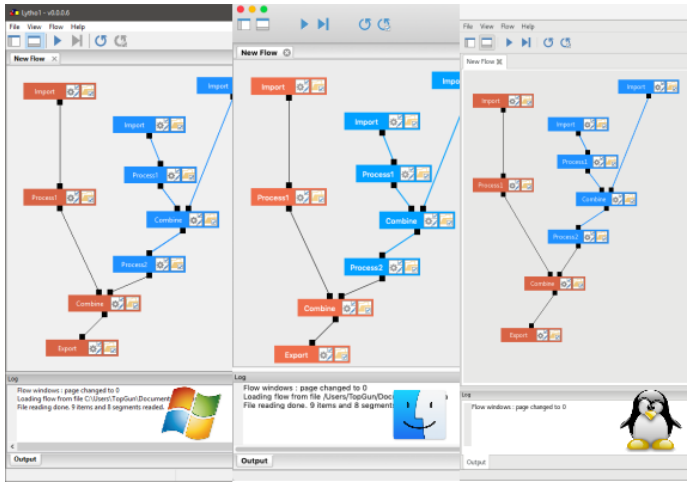
МИЭТ
Национальный исследовательский университет «МИЭТ»
Институт интегральной электроники (группы ЭН-34-35, каф. ПКИМС)

Кроссплатформенная разработка программного обеспечения



Лабораторная работа №7

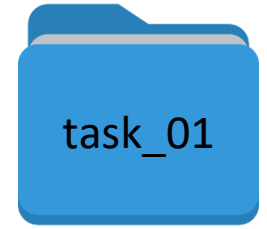
Продвинутый текстовый редактор

Подсветка неправильных конструкций (1)

```
ighlightingRule rule;
rule.format = keywordFormat;
for (const QString &pattern : keywordPatte
  rule.pattern = QRegularExpression(patter
  highlightingRules.append(rule);
}
```

орменная разработка программного обеспечения
ая работа 7. Продвинутый текстовый редактор

Посдсветка непра




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

enum QTextCharFormat::UnderlineStyle

This enum describes the different ways drawing underlined text.

Constant	Value	Description
QTextCharFormat::NoUnderline	0	Text is draw without any underlining decoration.
QTextCharFormat::SingleUnderline	1	A line is drawn using Qt::SolidLine.
QTextCharFormat::DashUnderline	2	Dashes are drawn using Qt::DashLine.
QTextCharFormat::DotLine	3	Dots are drawn using Qt::DotLine;
QTextCharFormat::DashDotLine	4	Dashes and dots are drawn using Qt::DashDotLine.
QTextCharFormat::DashDotDotLine	5	Underlines draw drawn using Qt::DashDotDotLine.
QTextCharFormat::WaveUnderline	6	The text is underlined using a wave shaped line.
QTextCharFormat::SpellCheckUnderline	7	The underline is drawn depending on the SpellCheckUnderlineStyle theme hint of QPlatformTheme. By default this is mapped to WaveUnderline, on macOS it is mapped to DotLine.

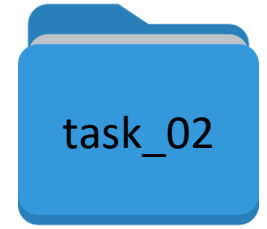
Подсветка неправильных конструкций (2)

```
rule.pattern = QRegularExpression("\\b\\d\\w+\\b");
rule.format.setForeground(Qt::black);
rule.format.setFontWeight(QFont::Normal);

rule.format.setUnderlineColor(Qt::red);
rule.format.setUnderlineStyle(QTextCharFormat::UnderlineStyle::WaveUnderline);

highlightingRules.append(rule);
```

Дополнительные виджеты в текстовом поле




```
44  void CodeEditor::paintLNArea(QPaintEvent *e) {
45      QPainter painter(lineNumbersArea);
46      painter.fillRect(e->rect(), QColor(Qt::gray).lighter(150));
47
48      QTextBlock block = firstVisibleBlock();
49      qint32 blockNumber = block.blockNumber();
50      qint32 top = (qint32)blockBoundingGeometry(block).translated(contentO
51      qint32 bot = top + (qint32)blockBoundingRect(block).height();
52
53      while (block.isValid() && top <= e->rect().bottom()){
54          if (block.isVisible() && bot >= e->rect().top()){
55              QString num = QString::number(blockNumber + 1);
56              painter.setPen(QColor(100, 100, 100));
57              painter.setFont(font());
58              painter.drawText(0, top, lineNumbersArea->width(), fontMetrics
59          }
60
```

Что нужно для боковой панели (1)

```
class LineNumbersArea : public QWidget {
    Q_OBJECT
private:
    CodeEditor *parentEditor;
public:
    LineNumbersArea(CodeEditor *parent) : QWidget(parent), parentEditor(parent) {}
public:
    QSize sizeHint() const {
        return QSize(parentEditor->calcLNAreaWidth(), 0);
    }
private:
    void paintEvent(QPaintEvent *e) {
        parentEditor->paintLNArea(e);
    }
};
```

Что нужно для боковой панели (2)

```
class CodeEditor : public QPlainTextEdit {
...
private:
    LineNumbersArea *lineNumbersArea;
...
public:
    qint32 calcLNAreaWidth();
    void paintLNArea(QPaintEvent *e);
private:
    void resizeEvent(QResizeEvent* e);
private slots:
    void onUpdateLNAreaWidth(qint32 numberOfLines);
    void onUpdateLNArea(const QRect &, qint32);
    ...
};
```

Что нужно для боковой панели (3)

```
CodeEditor::CodeEditor(QWidget *parent) : QPlainTextEdit(parent),
                                          lineNumbersArea(new LineNumbersArea(this)) {
...
    connect(this,
            SIGNAL(blockCountChanged(qint32)),
            this,
            SLOT(onUpdateLNAreaWidth(qint32)));

    connect(this,
            SIGNAL(updateRequest(QRect, qint32)),
            this,
            SLOT(onUpdateLNArea(QRect, qint32)));

    connect(this,
            SIGNAL(cursorPositionChanged()),
            this,
            SLOT(highlightCurrentLine()));

    onUpdateLNAreaWidth(0);
}
```

Что нужно для боковой панели (4)

```
qint32 CodeEditor::calcLNAreaWidth() {
    qint32 digits2Show = 1;
    qint32 max = qMax(1, blockCount());
    while (max >= 10) {
        max /= 10;
        ++digits2Show;
    }
    return 3 + fontMetrics().horizontalAdvance(QLatin1Char('9')) * digits2Show;
}

void CodeEditor::resizeEvent(QResizeEvent *e) {
    QPlainTextEdit::resizeEvent(e);
    lineNumbersArea->setGeometry(QRect(contentsRect().left(), contentsRect().top(),
                                       calcLNAreaWidth(), contentsRect().height()));
}

void CodeEditor::onUpdateLNAreaWidth(qint32) {
    setViewportMargins(calcLNAreaWidth(), 0, 0, 0);
}
```

Что нужно для боковой панели (5)

```
void CodeEditor::paintLNArea(QPaintEvent *e) {
    QPainter painter(lineNumbersArea);
    painter.fillRect(e->rect(), QColor(Qt::gray).lighter(150));

    QTextBlock block = firstVisibleBlock();
    qint32 blockNumber = block.blockNumber();
    qint32 top = (qint32)blockBoundingGeometry(block).translated(contentOffset()).top();
    qint32 bot = top + (qint32)blockBoundingRect(block).height();

    while (block.isValid() && top <= e->rect().bottom()) {
        if (block.isVisible() && bot >= e->rect().top()) {
            QString num = QString::number(blockNumber + 1);
            painter.setPen(QColor(100, 100, 100));
            painter.setFont(font());
            painter.drawText(0, top, lineNumbersArea->width(),
                             fontMetrics().height(),     Qt::AlignRight, num);
        }
        block = block.next();
        top = bot;
        bot = top + (qint32)blockBoundingRect(block).height();
        ++blockNumber;
    }
}
```

Задача: подсветить номер текущей строки

```
void CodeEditor::paintLNArea(QPaintEvent *e) {
    ...


    while (block.isValid() && top <= e->rect().bottom()) {
        if (block.isVisible() && bot >= e->rect().top()) {
            QString num = QString::number(blockNumber + 1);
            painter.setPen(QColor(100, 100, 100));
```

```
if(blockNumber == textCursor().blockNumber())
    painter.setPen(QColor(0, 0, 255));
else
    painter.setPen(QColor(100, 200, 100));
```

Автоматическое дополнение кода


```
22      completer = new QCompleter(this);
23      completer->
24      QStringLis
25      words << "
26      QStringLis
27      completer-
28      completer-
29
30      connect(co
31  }
32
33  qint32 CodeEd
34      qint32 dig
35      qint32 max
36      while (max >= 10){
37          max /= 10;
```

complete
setModel
setPopup
setWidget
setCurrentRow
setFilterMode
setWrapAround
setModelSorting
setCompletionMode
setCompletionRole

void complete (*const QRect &rect = QRect()*)

```
"output" << "endmodule";
tringListModel(words, complet
mpleter::UnfilteredPopupCompl
ted(QString)), this, SLOT(onI
{
));
```

Что нужно для автоматического дополнения

```
#include <QCompleter>


class CodeEditor : public QPlainTextEdit {
    ...
    QCompleter *completer;
    ...
private:
    ...
    void keyPressEvent(QKeyEvent *event);
    QString textUnderCursor();

    ...
    void onInsertCompletion(const QString &completion);
```

Как реализуется автоматическое дополнение (1)

Этап 1. Создание автодополнителя и связывание его с виджетом

```
completer = new QCompleter(this);
completer->setWidget(this);
```

Этап 2. Наполнение автодополнителя словами для дополнения

```
QStringList words;
words << "module" << "input" << "output" << "endmodule";
QStringListModel *model = new QStringListModel(words, completer);
completer->setModel(model);

completer->setCompletionMode(QCompleter::UnfilteredPopupCompletion);
```

Этап 3. Связывание автодополнителя с сигналом для обработки вставки

```
connect(completer,
        SIGNAL(activated(QString)),
        this,
        SLOT(onInsertCompletion(QString)));
```

Как реализуется автоматическое дополнение (2)

```
void CodeEditor::keyPressEvent(QKeyEvent *e) {
    if ( completer->popup()->isVisible() ) {
        switch (e->key()) {
            case Qt::Key_Enter:
            case Qt::Key_Return:
            case Qt::Key_Escape:
            case Qt::Key_Tab:
                e->ignore();
                return;
        }
    }

QPlainTextEdit::keyPressEvent(e);

const QString completionPrefix = textUnderCursor();
```

Как реализуется автоматическое дополнение (3)

```
    if (completionPrefix != completer->completionPrefix()) {
        completer->setCompletionPrefix(completionPrefix);
        completer->popup()->setCurrentIndex(completer->completionModel()->index(0, 0));
    }

    QRect cRect = cursorRect();
    cRect.setWidth(
        completer->popup()->sizeHintForColumn(0) +
        completer->popup()->verticalScrollBar()->sizeHint().width());

    if (!e->text().isEmpty() && completionPrefix.length() > 2)
        completer->complete(cRect);
}
```

Запуск внешних приложений: класс Qprocess (1)

Класс QProcess **–** класс для работы со сторонними процессами: запуск, чтение вывода, остановка и пр.

Конструктор:

```
QProcess(QObject *parent = Q_NULLPTR)
```

Важные методы :

```
void setArguments(QStringList args)     //Устанавливает аргументы командной строки
void setProgram(QString prog)           //Устанавливает путь до программы
void setWorkingDirectory(Qstring dir)   //Устанавливает рабочую директорию процесса
void kill()                             //Моментально убивает процесс
void terminate()                        //Посылает программе запрос на завершение (может не завершиться)
int exitCode()                          //Возвращает код с которым завершился процесс
QStringList arguments()                 //Возвращает аргументы
Qstring program()                       //Возвращает путь до программы
QProcess::ProcessState state()          //Возвращает состояние процесса
bool waitForFinished(int msecs=30000)   //Блокирует работу пока процесс не завершен
bool waitForStarted(int msecs=30000)    //Блокирует пока процесс не начнет работать
bool canReadLine()                      //Вернет true если процесс полностью составил строку вывода
static QStringList systemEnvironment() //Возвращает системное окружение
```

Сигналы:

```
void finished(int exitCode, QProcess::ExitStatus status) //Процесс завершен
void readyReadStandartError()                            //Можно прочесть поток вывода ошибок
void readyReadStandartOutput()                           //Можно прочесть стандартный поток вывода
void started()                                           //Процесс запущен
void stateChanged(QProcess::ProcessState state)          //Состояние изменилось
```

Состояния процесса:

```
QProcess::NotRunning – Не запущен
QProcess::Starting   – Запускается
QProcess::Running    - Выполняется
```

Запуск внешних приложений: класс Qprocess (2)

Запуск программы:

```
void start(QString &program, QStringList &args, OpenMode mode = ReadWrite)
void start(QString &command, OpenMode mode = ReadWrite)
void start(OpenMode mode = ReadWrite)
bool startDetached(QString &command)
bool startDetached(QString &program, QStringList &args,
                   QString &workingDirectory = QString(),
                   qint64 *pid = Q_NULLPTR
                   )
static void execute(QString &program, QStringList &args)
static void execute(QString &command)
```

Аргументы методов:

```
QString program          - имя программы или путь до исполняемого файла
QStringList args         – список аргументов
QString command          – команда консоли, которая будет исполнена
QString workingDirectory – директория из-под которой будет запущена программа
qint64* pid              – идентификатор процесса
```

Инициализация процесса

```
void MainWindow::initProcess() {

    process = new QProcess(this);

    connect(process,
            SIGNAL(started()),
            this,
            SLOT(onProcessStarted()));

    connect(process,
            SIGNAL(finished(int, QProcess::ExitStatus)),
            this,
            SLOT(onProcessFinished(int, QProcess::ExitStatus)));
}
```

Обработка сигналов процесса

```
void MainWindow::onProcessStarted() {
    actTaskRun->setEnabled(false);
    codeEditor->appendPlainText("");
    codeEditor->appendPlainText("Processing...");
    codeEditor->appendPlainText("");
}
```

```
void MainWindow::onProcessFinished(int exitCode, QProcess::ExitStatus exitStatus) {
    codeEditor->appendPlainText("Process done!");
    actTaskRun->setEnabled(true);
}
```

Запускаем процесс

```
void MainWindow::onMenuTaskRun() {
    QStringList arguments; arguments << "5";
    process->start("D:\\test_Console.exe", arguments);
}
```

Инициализация процесса

```
void MainWindow::initProcess() {

    process = new QProcess(this);

    connect(process,
            SIGNAL(started()),
            this,
            SLOT(onProcessStarted()));

    ...

    connect(process,
            SIGNAL(readyReadStandardOutput()),
            this,
            SLOT(onReadStandardOutput()));

}
```

Обработка вывода процесса

```
void MainWindow::onReadStandardOutput() {
    codeEditor->appendPlainText(process->readAllStandardOutput());
}
```

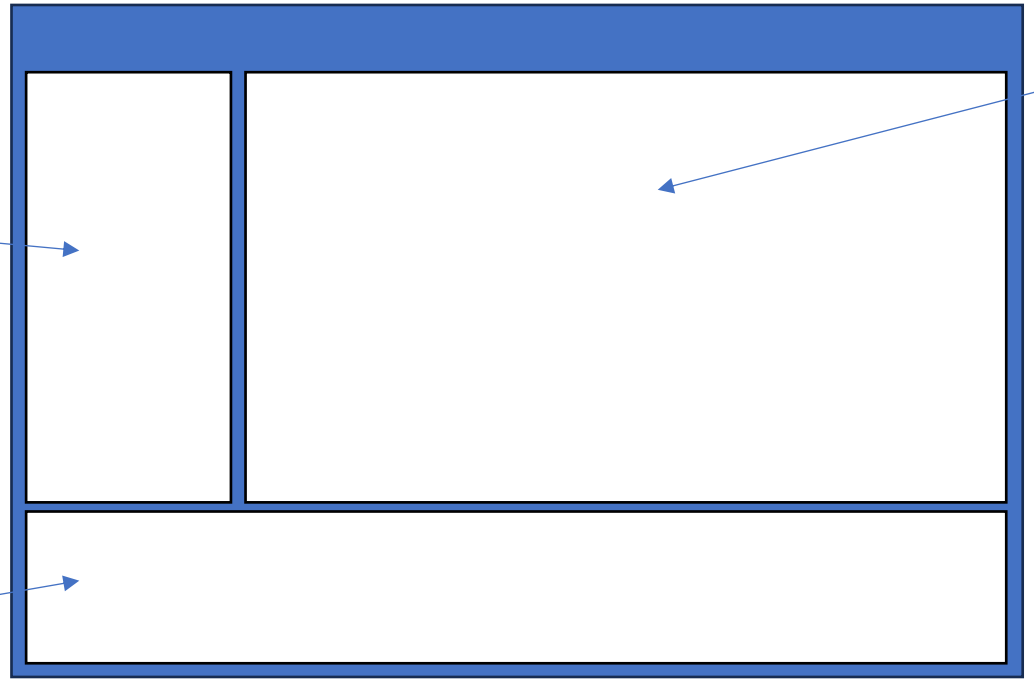
Задание

Мне нужен от вас текстовый редактор описаний SPICE с возможностью:

1. загрузки и редактирования нетлистов SPICE с подсветкой синтаксиса (как минимум: элементы, комментарии, директивы)
2. сохранения в двух вариантах («сохранить» и «сохранить как»)
3. запуска для отображаемого нетлиста симулятора (3 ЛР);
4. автоматический запуск для результатов моделирования графического постпроцессора (5 ЛР) в случае успешного моделирования.

Меню:

- File (New, Open, Save, Save As, Exit),
- Run(Netlist, Postprocessor)
- Help (About)



Древовидный список.
Сюда грузится перечень элементов из нетлиста (имена элементов)
Вид (по уровням):

```
<имя_файла> (без пути)
  +резисторы
        -R1
        -R2
  +конденсаторы
  ...
```

Текстовое поле.
Лог с возможностью добавления сообщений от идущих процессов (для ошибок и предупреждений - подсветка html)

Текстовое поле.
Сюда грузится нетлист, подсвечивается его синтаксис, этот код мы можем редактировать

Важно!
Если выбран пункт меню Run->Netlis, запускается автоматом симулятор, следом за ним - постпроцессор.

Если выбран пункт меню Run->Postprocessor, запускается отдельно только постпроцессор.