Rebol Programming For The Absolute Beginner, by Nick Antonaccio

Contents:

1. От переводчика

Решив познакомиться с этим языком программирования столкнулся с тем, что очень мало рускоязычной информации по Rebol. Взявшись за чтение (с помощью Гугл-переводчика :)) этой обучалки - решил сразу сохранить русский вариант, который Вы сейчас читаете. Сильно текст не редактировал - лиж бы был понятен смысл. Испровлял восновном коментарии в примерах. Этот труд для меня уже окупился - я познакомился с основами Rebol-ом и плотнее начал им заниматься. По мере своего "чтения"(перевода) других документаций и примеров буду собирать всё в одну кучу, и куда нибудь выкладывать, чтобы другим, рускаязычным пользователям было проще освоить этот язык. Вас призываю последовать этому примеру :)

Этот документ посвящён второй версии Rebol`a. Сейчас уже есть "альфа" версия третьего Rebol`a с которой можно познакомиться тут

```
browse http://rebolsource.net
; (Кликне по серому полю, чтобы выполнить написанный там код)
```

Так же есть замечательный последователь - язык Red, который и похож командами и синтаксисом, и написан на Rebel`e, и способен создавать исполняемые файлы, но пока развивается и не имеет таких возможностей, как Rebol. Подробней тут

```
browse https://www.red-lang.org
```

Невзерая на разнообразие последователей - самой стабильной и полностью доделаной версией остаётся Rebol 2, поэтому рекомендую начинать обучение именно с него.

2. Новичкам

Это руководство демонстрирует, насколько легко достичь реальных целей программирования с помощью гибкого и мощного языка Rebol. Текст направлен на то, чтобы научить обычных пользователей программировать компьютеры для выполнения полезных вещей без долгого и сложного обучения, необходимого для работы с другими языками программирования. Если вы опытный программист, обязательно ознакомьтесь с полными примерами приложений. Вы будете поражены компактным кодом Rebol и простым кросс-платформенным использованием.

Скачать пакет из 11 примеров (на английском) из этой обучалке можно тут:

```
browse http://musiclessonz.com/rebol_tutorial_examples.zip
; (Кликне по серому полю, чтобы выполнить написанный там код)
```

ZIP-файл содержит снимки экрана и исполняемые файлы, демонстрирующие, что может легко сделать начинающий программист Rebol, обучающийся неполный день, в течении нескольких недель.

Концепции, продемонстрированные в руководстве, помогут новым программистам понять другие методы и инструменты программирования, а также заложат основу для приобретения новичками необходимых навыков программирования. На самом деле даже новички смогут создавать полезные программы Rebol в течении первыех нескольких дней. И Rebol не ограничивается определенными типами приложений или операционных систем. Код, написанный на Rebol, может работать без изменений во множестве операционных системах и может использоваться для создания огромного количества пользовательских приложений с современной графикой, интерфейсами CGI, сетевыми функциями, подключением к базам данных и многим другим. Если

вы привыкли к другим средам разработки, вы обнаружите, что Rebol - это красиво разработанный язык, содержащийся в чрезвычайно маленьком и эффективном пакете. Обучение Rebol является продуктивным, приятным и заставляет задуматься как новичков, так и опытных программистов. Независимо от вашего уровня опыта, изучение Rebol в какой-то момент сэкономит вам время и нервы в программировании. Это фантастически универсальный маленький инструмент, которому вы найдете множество применений.

3. Перспективы для начинающих

Прежде чем глубоко погрузиться в механику языка, рассмотрим фундаментальную перспективу, которую полезно понимать:

Современные компьютеры, операционные системы и программы - все делают очень простые вещи ограниченным образом, с ограниченным набором типов данных:

1) Они позволяют пользователям вводить различные типы данных: текст, изображения, звуки, видео и т.д.

2) Они позволяют пользователям сохранять, извлекать, организовывать, совместно использовать/передавать, манипулировать, изменять, просматривать и иным образом работать с этими данными полезными способами.

Все, что можно сделать с помощью современного компьютера, в основном связано с манипулированием текстовыми и нетекстовыми данными (нетекстовые данные называются «двоичными» или «бинарными»(binary) данными). В текущем состоянии современных вычислений данные всех типов обычно вводятся, обрабатываются и возвращаются через графические пользовательские интерфейсы, такие как программные интерфейсы Windows, веб-формы, отображаемые в браузерах, и другие управляемые с клавиатуры/мыши «GUI». Данные сохраняются на локальных жестких дисках и устройствах хранения (компакт-диски, флэш-накопители и т.д.) и на удаленных веб-серверах ("облаках") и обычно передаются через локальные сети и Интернет-соединения. Знание того, как управлять этими вычислительными элементами, чтобы позволить пользователям манипулировать данными, является целью обучения программированию. Не имеет значения, интересуетесь ли вы написанием бизнес-приложений для работы с финансами и планированием (текстовые данные), программами для изменения веб-страниц или электронной почты в Интернете (текстовые и графические данные), программами для организации или воспроизведения музыки (двоичные данные). , программы для передачи файлов по сети (текстовые и/или двоичные данные), программы для трансляции видео и звука через Интернет (быстро передаваемые последовательные кадры двоичных данных), программы для управления роботизированным оборудованием (двоичные данные, отправляемые по подключенному кабелю или сети), программы для игр и т.д. все они требуют обучения тому, как получать, манипулировать и возвращать какие-то данные. Вы можете делать все это с помощью Rebol, и если вы сделаете это на одном языке, вам будет легче делать подобное на других, специализированных языках и инструментах программирования.

Rebol легко и интуитивно позволяет управлять и создавать пользовательскиме интерфейсы и разные типы данных. Это даёт возможность программистам быстро создавать графические интерфейсы для ввода и отображения всех распространенных типов данных. Он может легко манипулировать текстом, графикой и звуком полезными и удобными средствами и предоставляет простые методы для сохранения, извлечения и обмена данными на разных типах оборудования, сетей и Интернета. Это отличный способ начать учиться программировать.

4. Использование интерпритатора Rebol для общения с компьютером

Интерпретатор Rebol - это программа, которая запускается на вашем компьютере. Он переводит письменный текст, организованный в синтаксисе языка Rebol («исходный код»), в инструкции, понятные компьютеру. Одна из замечательных особенностей Rebol заключается в том, что это очень маленькая программа, содержащаяся в одном файле, которая работает практически на всех типах компьютеров и операционных систем (ПК с Windows, компьютер Macintosh, веб-сервер

Linux и т.д.), без каких-либо сложных процессов установки и перекомпелирования программ. Это позволяет вам разговаривать со всеми этими машинами на одном языке. Вы просто загружаете программу-интерпретатор Rebol, загружаете ей текстовый файл с кодом программы, и машина делает то, что вы хотите, с данными, которыми вы хотите. Всё это очень легко использовывается.

Чтобы получить бесплатный интерпретатор Rebol для Microsoft Windows, перейдите по ссылке

```
browse http://rebol.com/view-platforms.html
```

и скачайте файл view.exe для Windows, просто щелкните его мышью и сохраните на жесткий диск. Когда вы запускаете view.exe в первый раз, вы можете установить его, если хотите, но это не обязательно. Просто следуйте инструкциям на экране. После того, как интерпретатор Rebol загружен и запущен на вашем компьютере, щелкните значок «Console»(Консоль), и вы готовы начать вводить текст в программах Rebol. (Либо просто нажмите на кнопку "Интерпретатор Rebol" с лева, с верху окна этой обучающей программы.) Чтобы создать свою первую программу, введите следующую строку в интерпретатор Rebol и затем нажмите клавишу [Enter](ввод) на клавиатуре:

```
alert "Привет мир!"
```

Прежде чем идти дальше, попробуйте. Загрузите или запустите Rebol и введите приведенный выше код, чтобы увидеть, как он работает. Это просто и занимает всего секунду. Если вы хотите запустить Rebol в любой другой операционной системе, просто выберите, загрузите и запустите файл для соответствующей системы и компьютера. Он работает одинаково во всех операционных системах.

ПРИМЕЧАНИЕ. В этой электронной версии руководства вы можете просто щелкнуть примеры кода, чтобы их запустить. Интерпритатор уже встроен в приложение.

5. Изучение языка Rebol

Чтобы выучить любой язык программирования, вам нужно будет запомнить множество команд. Не поленитесь вручную вводить примеры из этого руководства, это поможет лучше понять и запомнить синтаксис языка. Избегайте простого копирования и вставки, если действительно хотите учиться. Вы научитесь программировать так же, как вы изучаете новый разговорный язык - погрузившись в слова и фразы этого языка (даже если сначала вы не понимаете, о чем говорится). Повторение(запись) предложений - это естественный способ запомнит и понять, как команды звучат и работают. Некоторые слова, фразы и шаблоны со временем становятся привычными, а синтаксис языка становится ясным и понятным.

Наблюдать за работай примеров програм так же необходимо. Это помогает формировать ассоциации между тем, что делает компьютер, и фрагментами кода, которые заставляют его это делать. Недостаточно просто прочитать код - главное - увидеть и затем понять, как он выполняется. Если у вас нет времени набирать код вручную, по крайней мере, вставьте его в интерпретатор или кликните указатилем мышки по нему.

Поскольку языки программирования требуют более специфического синтаксиса, чем разговорные языки, изучение точной грамматики более важно, чем разговорный язык. Простые примеры, подобные приведенным ниже, легко запомнить - точно так же, как простые фразы, запоминаемые на разговорных иностранных языках. Однако, чтобы научиться говорить более свободно, правильный порядок слов, символов и других элементов должен быть набран точно, гораздо более конкретней, чем слова разговорной речи. Интерпретатор недостаточно умен, чтобы «угадать», что вы собираетесь делать, и если вы ошиблись в грамматике (даже на один символ), он неправильно интерпретирует ваш код и сделает что-то другое, а не то, что вы хотите. Иногда вы получаете сообщение об ошибке, и интерпретатор сообщит вам, где вы ошиблись в

грамматике в своем коде. Иногда ваша программа просто не работает правильно. Знакомство с синтаксисом с самого начала поможет вам избежать этих проблем.

6. Первые примеры кода - создание окна графического интерфейса пользователя

Теперь немного кода! Компьютерные программы обычно используют графические интерфейсы для получения данных от пользователя и отображения данных пользователю. Графические интерфейсы пользователя работают интуитивно, и современные пользователи компьютеров знакомы с ними. Они содержат интерактивные кнопки, поля для ввода текста, меню, изображения и другие «виджеты», которые позволяют пользователю взаимодействовать с программои и соответственно компьютером. Программы Windows - это восновном программы с графическим интерфейсом. Веб-страницы также являются графическими интерфейсами. Пользователи нажимают кнопки с помощью указателя мышки, чтобы выполнять действия, выбирать настройки из меню, вводить текстовые данные в поля и т.д. большинство современных языков программирования включают некоторые средства для создания графических интерфейсов, которые можно использовать для взаимодействия с пользователем. Rebol упрощает создание графического интерфейса. Чтобы создать простое окно графического интерфейса, просто введите следующую строку в интерпретатор Rebol и нажмите [ENTER]. Обратите внимание на слова «view layout» в примерах ниже - вы будете использовать их каждый раз, когда будете создавать графический интерфейс в программе на Rebol:

```
view layout/size [] 400x300
```

Эта строка кода создает окно на 400 пикселей по горизонтали(вправо) и на 300 пикселей по вертекали(вниз) (пиксели - это точки на экране компьютера). Программа пока ничего не делает, но окно графического интерфейса создаёт, и его можно перемещать по экрану, сворачивать и закрывать, как и любую другую программу Windows. В других языках программирования создание такого окна может потребовать нескольких страниц кода и большого количества предварительных знаний. Rebol упрощает эту задачу.

Чтобы добавить кнопку в указанный выше графический интерфейс, введите следующий код. Обратите внимание, что в скобки добавлено слово «button» (кнопка):

```
view layout/size [button] 400x300
```

Теперь в графическом интерфейсе есть обычная синяя кнопка, которую можно кликнуть мышью. Чтобы добавить текст к кнопке, введите следующий код. Обратите внимание, что после кнопки был добавлен текст «Кликни»:

```
view layout/size [button "Кликни"] 400x300
```

Чтобы заставить кнопку что-то делать, введите следующий код, а затем щелкните кнопку графического интерфейса с помощью указателя мыши. Обратите внимание, что текст [alert "Привет мир!"] был добавлен после текста кнопки:

```
view layout/size [button "Кликни" [alert "Привет мир!"]] 400x300
```

Теперь, когда вы нажимаете кнопку в вашем графическом интерфейсе, компьютер отвечает, сообщением «Привет мир!» в новом окне. Чтобы программа делала что-то более интерактивное,

введите следующее, а затем еще раз нажмите кнопку. Обратите внимание на добавление «data: request-text»(дата: тест-запроса) в начале строки. Этот код запрашивает у пользователя некоторый текст и назначает его слову «data» (переменной data присваевается значение, введённое пользователем), чтобы на него можно было ссылаться и использовать в программе:

```
data: request-text view layout/size [
    button "Кликни" [alert data]] 400x300
```

Приведенный выше код разделен на две строки так, чтобы он соответствовал ширине экрана, но его можно ввести в интерпретаторе Rebol как одну строку. Эта одна строка - все, что нужно для создания программы, которая получает некоторые данные от пользователя, создает графический пользовательский интерфейс, ожидающий взаимодействия с пользователем, и что-то делает с входными данными (отображает их в небольшом диалоговом окне).

Теперь мы сохраним некоторые данные на жестком диске вашего компьютера. Введите следующий код и нажмите «Yes», если интерпретатор Rebol запросит у вас разрешение на запись на жесткий диск. Обратите внимание на добавление «write %data.txt data» в конец строки. Этот код записывает данные, полученные от пользователя и хронящиеся в переменной data, в файл на диске, который называется "data.txt".

```
data: request-text view layout/size [
    button "Кликни" [alert data]] 400x300 write %data.txt data
```

Если после закрытия графического интерфейса вы посмотрите директории, откуда запустили этот справочник (или интерпритатор), то вы увидите, что теперь существует текстовый файл (data.txt), содержащий текст, который вы ввели в программу.

С этой единственной строчкой кода у вас есть рабочая программа, которая действительно делает что-то полезное. Он получает, отображает и сохраняет некоторые данные от пользователя, используя знакомые взаимодействия с графическим интерфейсом. Вы можете настроить его для хранения телефонных номеров, имен пользователей/паролей или любой другой полезной информации. Это просто - а дальше все становится еще интереснее!

Rebol отлично справляется со многими типами общих данных, а не только с текстом. Вы можете легко отображать фотографии и другую графику в окне программы, воспроизводить звуки, отображать веб-страницы и т.д. и так же хорошо справляется с данными, передаваемыми по сети и через Интернет. Вот код, который загружает изображение с веб-сервера и отображает его в графическом интерфейсе - обратите внимание на слова «view layout»:

```
view layout [image load http://rebol.com/view/bay.jpg]
```

Слово "image"(изображение) сообщает интерпритатору о том, что нужно вывести изображение. Слово "load"(загрузка) говорит о необходимости загрузки изображения перед отображением.

Rebol позволяет применять к изображениям множество встроенных эффектов. Обратите внимание на код «effect [тип эффекта]» в следующих примерах:

```
view layout [image load http://rebol.com/view/bay.jpg effect [Grayscale]]
```

.

```
view layout [image load http://rebol.com/view/bay.jpg effect [Emboss]]
```

.

```
view layout [image load http://rebol.com/view/bay.jpg effect [Flip 1x1]]
```

Вот как вы можете прочитать тот же файл с веб-сервера и сохранить его на свой диск. Модификатор «/binary» используется при работе с двоичными (нетекстовыми) данными:

```
write/binary %bay.jpg read/binary http://rebol.com/view/bay.jpg
```

Теперь вы можете читать изображение прямо с жесткого диска и отображать его в окне программы. Просто введите код ниже. Снова обратите внимание на слова «view layout», и на этот раз между скобками указание, что файл загружается с локального диска:

```
view layout [image load %bay.jpg]
```

Здесь важно отметить, что вы также можете использовать это изображение в других графических приложениях на вашем компьютере, включая другие программы, написанные не на Rebol. Вы можете, например, открыть этот файл в программе для редактирования изображений или прикрепить его к электронному письму и отправить другу. Очень важно понимать, что данные могут взаимодействовать между языками, инструментами программирования, операционными системами и другими связанными доменами. По мере того, как вы узнаете больше о программировании, вы сможете найти специализированные инструменты, которые легко и эффективно помогают достигать определенных целей. Вы можете обмениваться данными между различными программами(инструментами), просто сохраняя их на общем носителе. Таким образом, все языки программирования и инструменты могут использоваться вместе для достижения поставленных целей любой сложности. Это ключевая концепция, которую следует учитывать при изучении программирования. Очень мало технологий действительно ново. Основа для большинства вычислительных приложений существует уже несколько десятилетий (жесткие диски для хранения файлов в структурах каталогов, оборудование для ввода и вывода текста, графики, аудио и других данных, сети для передачи данных между машинами и т.д.). Эти базовые компоненты не слишком сильно изменились. Они просто улучшились в скорости и емкости - а программные инструменты, используемые для работы с ними, эволюционировали, чтобы позволить программистам и пользователям выполнять задачи высокого уровня быстрее, проще и эфективней. Однако опытный программист, пишущий приложение для работы с большими объемами текстовых данных, совместно используемых через Интернет, поймет, что все данные по-прежнему находятся в файлах где-то на жестком диске, на сервере, компьюторе подключенному к сети, и что данные могут быть доступ к нему осуществляется различными старомодными средствами программирования - даже если он был создан и помещен в программу с использованием передовых технологий баз данных, отображен в ярком новом графическом интерфейсе и передан по широкополосным беспроводным соединениям. Это по-прежнему просто текстовые данные, сохраненные где-то в файле, и ими можно управлять практически на любом языке, который обеспечивает доступ к сетям и текстовым файлам! Имейте это в виду, поскольку ваше знакомство с различными инструментами программирования расширяется ...

7. Больше примеров

Ниже приведены еще несколько коротких примеров чтения, записи и обработки данных на жестком диске и в Интернете, а также взаимодействия с пользователем. Введите их в интерпретатор Rebol, чтобы немного лучше познакомиться с языком Rebol. Вы можете настроить

Rebol так, чтобы он открывался прямо с консоли. Щелкните меню «User»(Пользователь) на панели Rebol и снимите флажок «Open Desktop On Startup»(Отрвыать Рабочий сол при запуске). Это избавит вас от необходимости нажимать кнопку «Console»(Консоль) каждый раз при запуске Rebol.

Следующая строка отображает текущий день и время в интерпретаторе:

```
print now
```

Слово «print»(печатать) - отобразит текстовые данные прямо в интерпретаторе Rebol. Слово «now»(сейчас) обозначает текущею дату и время.

Следующая строка выполнит некоторые математические вычисления и отобразит результат:

```
print (10 + 12) / 2
```

Следующая комманда просит пользователя выбрать файл на жестком диске:

```
request-file
```

Приведенный ниже код позволяет пользователю выбрать цвет:

```
request-color
```

Следующий код задает пользователю вопрос типа "да-нет":

```
request "Тебе нравится?"
```

Вот неплохой способ запросить у пользователя дату:

```
request-date
```

В приведенном ниже коде запрашивается имя пользователя и пароль:

```
request-pass
```

Следующий код открывает веб-браузер вашего компьютера и отображает указанную веб-страницу:

```
browse http://rebol.com
```

Следующий код запускает встроенный в Rebol текстовый редактор и открывает файл test.txt

```
editor %test.txt
```

Обратите внимание на символ процента («%») в приведенном выше примере. В Rebol этот символ используется для обозначения всех меток файлов. Поскольку Rebol может использоваться в разных операционных системах и поскольку все эти операционные системы используют разные форматы синтаксиса для обозначения дисков, путей и т.д., Rebol использует универсальный формат:%/диск/путь/путь/.../имя_файла.расширение. Например, «%/c/windows/notepad.exe» относится к «C:\Windows\Notepad.exe» в Windows. Rebol преобразует этот синтаксис в соответствующий формат операционной системы, в которой он запущен, чтобы ваш код, написанный один раз, можно было использовать в любой операционной системе без изменений.

Следующая строка отправляет электронное письмо на адрес user@website.com с текстом «Привет. Как дела?». Попробуйте заменить имя пользователя и веб-сайт своим собственным адресом электронной почты (если вы загрузили и запустили Rebol, не устанавливая его, вам нужно будет запустить мастер настройки, чтобы сконфигурировать параметры почтовой учётной записи, для отправки электронной почты. Для этого введите «install»(установить) и следуя инструкциям заполняя соответствующие поля, выплните установку.)

```
send user@website.com "Привет. Как дела?"
```

Строка ниже отправляет пользователю веб-страницу:

```
send user@website.com read http://www.rebol.com
```

В приведенном ниже коде отображается содержимое почтового ящика пользователя:

```
print read pop://user:pass@website.com
```

Следующая строка загружает один бинарный файл на ftp сервер (или web сервер через ftp доступ):

```
write/binary ftp://user:pass@website.com read/binary %file
```

Следующее загружает весь каталог файлов на ftp/web-сервер пользователя:

```
foreach file load %./ [if not dir? file [write/binary join
    ftp://user:pass@website.com/ file read/binary file]]
```

Все это очень похоже на разговорный английский, не так ли? Вам просто нужно правильно набрать текст, и компьютер сделает то, что вы хотите. Чем больше вы изучаете язык, тем больше у вас будет возможностей заставить компьютер выполнять ваши приказы ... Легко, верно?

8. Сравним побыстрому

Чтобы быстро понять, насколько Rebol проще, чем другие языки, вот небольшой пример. Простейший код для создания основного окна графического интерфейса Rebol, который был представлен ранее:

```
view layout/size [] 400x300
```

Он работает на всех компьютерах одинаково.

Код для такого же простого примера представлен ниже на языке программирования «C++». Он делает то же самое, что и однострочник Rebol выше, за исключением того, что работает только в Microsoft Windows. Если вы хотите сделать то же самое на компьютере Macintosh, вам нужно написать совершенно другой код на C++. То же самое верно для Unix, Linux, Beos или любой другой операционной системы. Чтобы делать очень простые вещи, нужно выучить огромные фрагменты кода, и эти фрагменты кода различны для каждого типа компьютеров и операционных систем. Кроме того, вам, как правило, прийдётся потратить немало времени на изучение самых элементарных вещей о формате кодирования и основ о том, как компьютер думает, прежде чем вы даже начнете создовать такие примеры, как код ниже:

```
#include <windows.h>

/*  Declare Windows procedure  */
LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

/*  Make the class name into a global variable  */
char szClassName[ ] = "C_Example";

int WINAPI
WinMain (HINSTANCE hThisInstance,
         HINSTANCE hPrevInstance,
         LPSTR lpszArgument,
         int nFunsterStil)

{
    HWND hwnd;
    /* This is the handle for our window */
    MSG messages;
    /* Here messages to the application are saved */
    WNDCLASSEX wincl;
    /* Data structure for the windowclass */

    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    /* This function is called by windows */
    wincl.style = CS_DBLCLKS;
    /* Catch double-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;
    /* No menu */
```

```
    wincl.cbClsExtra = 0;
    /* No extra bytes after the window class */
    wincl.cbWndExtra = 0;
    /* structure or the window instance */
    /* Use Windows's default color as window background */
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    /* Register window class. If it fails quit the program */
    if (!RegisterClassEx (&wincl))
        return 0;

    /* The class is registered, let's create the program*/
    hwnd = CreateWindowEx (
           0,
            /* Extended possibilites for variation */
           szClassName,
            /* Classname */
           "C_Example",
            /* Title Text */
           WS_OVERLAPPEDWINDOW,
            /* default window */
           CW_USEDEFAULT,
            /* Windows decides the position */
           CW_USEDEFAULT,
            /* where the window ends up on the screen */
           400,
            /* The programs width */
           300,
            /* and height in pixels */
           HWND_DESKTOP,
            /* The window is a child-window to desktop */
           NULL,
            /* No menu */
           hThisInstance,
            /* Program Instance handler */
           NULL
            /* No Window Creation data */
           );

    /* Make the window visible on the screen */
    ShowWindow (hwnd, nFunsterStil);

    /* Run the message loop.
        It will run until GetMessage() returns 0 */
    while (GetMessage (&messages, NULL, 0, 0))
    {
        /* Translate virtual-key messages
            into character messages */
        TranslateMessage(&messages);
        /* Send message to WindowProcedure */
        DispatchMessage(&messages);
    }

    /* The program return-value is 0 -
        The value that PostQuitMessage() gave */
    return messages.wParam;
}
/*  This function is called by the Windows
        function DispatchMessage()  */

LRESULT CALLBACK
WindowProcedure (HWND hwnd, UINT message,
```

```
    WPARAM wParam, LPARAM lParam)
{
    switch (message)
    /* handle the messages */
    {
        case WM_DESTROY:
            PostQuitMessage (0);
                /* send a WM_QUIT to the message queue */
            break;
        default:
            /* for messages that we don't deal with */
            return DefWindowProc (hwnd, message,
                wParam, lParam);
    }

    return 0;
}
```

У-ух. Возвращяемся к Rebol... :)

9. Понимание переменных и функций

Все языки программирования используют две важные вещи: переменные и функции/команды. Функции - это команды, которые говорят компьютеру что-то сделать. Они похожи на глаголы в разговорной речи. Вы уже использовали некоторые функции в предыдущих примерах. Например, «print»(печатать) - это функция - ее можно рассматривать как глагол, обозначающий действие.

Переменные - это имена данных, которые нужно вводить, сохранять, обрабатывать, отображать и т.д. Они похожи на существительные/наречия в разговорной речи. Например, «now»(сейчас) можно рассматривать как переменную. Это наречие, относящееся к определённым данным (текущее время).

Функции и переменные должны использоваться в синтаксисе, определяемом языком программирования, который вы используете. Ни один из существующих компьютерных языков не позволяет вам говорить свободно, как в предложении естественного языка «покажите пользователю фотографии его детей». Вы должны писать команды более конкретным образом - используя синтаксис, который неизменно включает переменные и функции. Как программист, чтобы компьютер "показывал пользователю фотографию", вы должны:

1) присвоить имя файла с изображением переменной (существительное)

2) используйте функцию (глагол) для отображения изображения, на которое ссылается переменная из первого пункта

3) назначьте действие функции кнопке или другому графическому виджету в графическом интерфейсе (который сам создается путем размещения переменных и функций в определенном порядке)

Rebol делает многие вещи настолько «высокоуровневыми», насколько и можно ожидать - максимально приближенными к общению с компьютером на человеческом английском, насколько это возможно в настоящее время, но для этого по-прежнему требуется особый синтаксис и структура. Все современные компьютерные языки пытаются это сделать, но каждый по своему. Если вы хотите научиться программировать на любом языке, вам нужно научиться использовать переменные и функции в грамматике, которую определяет соответствующий язык программирования. Это основа всего программирования.

В Rebol переменные и функции называются «словам».

10. Слова Rebol

Если вы хотите присвоить некоторым данным метку, чтобы их можно было использовать в вашей программе, вы должны присвоить им название(слово). То же верно и для действий, выполняемых функциями. В Rebol встроено множество функциональных слов, которые представляют общие действия («print»(печать), «alert»(предупреждение), «request-date»(запрос-даты) и т.д.). Вы уже видели использование таких встроенных слов (функций) в предыдущих примерах. Чтобы создать свои собственные функции, вы берете ранее созданные функциональные слова и переменные, группируете их вместе в определенном порядке, который выполняет то, что вы хотите, и назначаете слово этой "программе". Затем вы можете обратиться к этой программе и/или данным, используя слово, которое вы ей присвоили.

Изучение функциональных слов, которые уже определены в языке программирования, является важной частью обучения. Эти слова являются существующим "словарным запасом языка", и для правильной работы они обычно ожидают, что за ними последуют другие слова в определенном порядке и формате. Слово «write»(запись), например, записывает данные на устройство хранения (жесткий диск, флэш-накопитель, веб-сервер и т.д.). Это функция, которая ожидает, что имя файла будет записано следом за ним, а затем данные или имя переменной с данными для записи в этот файл. Это надо писать в следующем порядке:

```
write %text.txt "Какой то, записываемый текст"
```

Если вы нарушите порядок, то команда(функция) несработает:

```
write "Какой то, записываемый текст" %text.txt ; НЕПРАВЕЛЬНО!
```

Если вы введете его в неверном порядке, то Rebol не поймет синтаксис, и вы получите сообщение об ошибке.

11. Слова и грамматика GUI - немного подробней.

Как вы видели в предыдущих примерах, слова "view layout", за которыми следуют две скобки ("[]"), могут использоваться для отображения графического пользовательского интерфейса (GUI - Graphic User Interface) в Rebol. Вы можете поместить в скобки элементы, которые хотите видеть в графическом окне программы. Rebol содержит слова, отображающие все основные графические элементы, используемые в графических интерфейсах. Попробуйте ввести следующие примеры кода:

```
view layout [button]
```

.

```
view layout [field]
```

.

```
view layout [text "В Rebol действительно довольно легко программировать"]
```

```
view layout [text-list]
```

.

```
view layout [
    button
    field
    text "В Rebol действительно довольно легко программировать"
    text-list
    check
]
```

Обратите внимание, что слова могут быть разделены «пробелом» внутри скобок. Лишние пробелы, переводы строк, табуляции и другие пустые символы интерпретатором игнорируются. Табуляцию традиционно используются для визуального групирования строк, но она не обязательна.

Дополнительные характеристики графических элементов могут быть включены непосредственно после каждого из их соответствующих слов. Такие модификаторы в Rebol называются «параметрами»(facets/фасеты), и они позволяют настроить все характеристики каждого типа графического виджета (размер, цвет, отображаемый текст) и т. д. Попробуйте ввести код ниже - это то же самое, что и приведенный выше код, с некоторыми дополнительными параметрами:

```
view layout [
    button red "Кликни"
    field "Введите какой нибудь текст"
    text "В Rebol действительно довольно легко программировать" purple
    text-list 400x300 "Линия раз" "линия два" "другая линия"
    check yellow
]
```

11.1 Действия

ВАЖНО: Если вы хотите, чтобы графический элемент выполнял какието действия, то просто поместите слово (слова) действия в скобки после него. Когда элемент графического интерфейса щелкается мышью или активируется иным образом, действия в собках будут выполнены. Введите следующий код, чтобы увидеть, как это работает:

```
view layout [button [alert "Вы нажали кнопку."] ]
```

.

```
view layout [button red "Кликни" [alert "Вы кликнули!"]]
```

.

```
view layout [
    text "Некоторые примеры действий. Попробуйте каждый:"
```

```
    button red "Кликни" [alert "Вы нажали на кнопку."]
    field 400 "Введите любой текст и нажмите ввод(Enter) на клавиатуре."
        [alert value]
    text-list 400x300 "Выберете эту линию" "потом эту" "теперь эту"
        [alert value]
    check yellow [alert "Вы кликнули по жёлтолму квадратику."]
    button "Выход" [quit]
]
```

Обратите внимание, как слово "value"(значение) относится к выбранному элементу в текстовом списке и к тексту, содержащемуся в поле ввода.

Альтернативные действия (т.е. те, которые запускаются щелчком правой кнопкой мыши) могут быть включены для любого графического виджета. Просто заключите их во вторм блоке в скобках:

```
view layout [button [alert "Левая  кнопка"] [alert "Правая кнопка"]]
```

Вот некоторые другие элементы графического интерфейса, используемые в языке Rebol:

```
view layout [
    backcolor white
    h1 "Ещё примеры графических элементов:"
    box red 500x2
    bar: progress
    slider 200x16 [bar/data: value show bar]
    area "Пишите тут"
    drop-down
    across
    toggle "Кликни" "Тут" [print value]
    rotary "Кликни" "Ещё" "...и ещё" [print value]
    choice "Выберете" "Элемент 1" "Элемент 2" "Элемент 3" [print value]
    radio radio radio
    led
    arrow
    return
    text "Нормальный"
    text "Жирный" bold
    text "Наклонный" italic
    text "Подчёркнутый" underline
    text "Жирный, наклонный и подчёркнутый" bold italic underline
    text "Текст с засечками" font-name font-serif
    text "Текст с интервалами" font [space: 5x0]
    return
    h1 "Заголовок 1"
    h2 "Заголовок 2"
    h3 "Заголовок 3"
    h4 "Заголовок 4"
    tt "Текст пишущей машинки"
    code "Текст кода"
    below
    text "Большой" font-size 32
    title "Центрированный заголовок" 200
    across
    vtext "Нормальный"
    vtext "Жирный" bold
    vtext "Наклонный" italic
```

```
    vtext "Подчёркнутый" underline
    vtext "Жирный, наклонный и подчёркнутый" bold italic underline
    vtext "Текст с засечками" font-name font-serif
    vtext "Текст с интервалами" font [space: 5x0]
    return
    vh1 "Видео заголовок 1"
    vh2 "Видео заголовок 2"
    vh3 "Видео заголовок 3"
    vh4 "Видео заголовок 3"
    label "Метка"
    below
    vtext "Большой" font-size 32
    banner "Баннер" 200
]
```

Приведенные выше примеры демонстрируют, как Rebol создает графические интерфейсы пользователя, использованные для ввода и отображения данных различными способами. Их можно настроить с помощью параметров, и они могут выполнять нужные действия. Это большая часть типичной современной компьютерной программы! Для получения дополнительной информации о возможнастях графического интерфейса см:

```
browse http://rebol.com/docs/easy-vid.html
```

и

```
browse http://rebol.com/docs/view-guide.html
```

Чтобы сделать ваши графические интерфейсы более полными, интересными и полезными, вам нужно больше узнать о том, какое разнообразие их представленно и как можно заставить интерпритатор манипулировать данными...

12. Создание собственных слов

Как и разговорные языки, языки программирования пластичны и выразительны. Никогда не бывает только одного способа написать какуюто программу. Вам порой нужно придумать и использовать свои собственные слова, и вам нужно организовать их так, чтобы они имели значение и функциональность, которые вам необходимы. Как и в разговорной речи, вы должны думать о том, что вы пытаетесь сказать. В Rebol у вас есть дополнительное преимущество, заключающееся в возможности создавать слова на собственном языке, чтобы выражать действия и маркировать данные.

Слова создаются и присваиваются переменным и функциям в Rebol с помощью символа двоеточия (":"). Вы можете использовать любое слово для обозначения любых конкретных данных или действий. Например, если вы хотите использовать слово «picture» для обозначения файла изображения в Интернете, вы можете сделать следующее:

```
picture: load http://rebol.com/view/bay.jpg

; Вышеупомянутая строка создает переменную (метку), которую можно
использовать
; в вашей программе в любом месте, где вы хотите прочитать и использовать
файл,
```

```
; расположенный по адресу http://rebol.com/view/bay.jpg. Встроенное слово
Rebol
; «load» (загрузка) выполняет фактическую работу по переходу на адрес веб-
сайта
; и сбору данных изображения.
; Теперь вы можете использовать слово «picture» для обозначения полученного
; изображения. Отобразите его в графическом интерфейсе, используя следующий
код:

view layout [image picture]
```

Слова «view»(вид), «layout»(макет) и «image»(изображение) встроены в Rebol, и слово «picture»(картинка) теперь так же понятно интерпритатору, потому что оно было определено.

Поскольку слово "picture" _переменная_, вы также можете переопределить и _изменить_ данные, на которые оно ссылается:

```
picture: load http://rebol.com/view/demos/palms.jpg

; Теперь, когда вы используете слово «picture» в своей программе, оно
относится
; к другой картинке, к другому файлу в другом месте в Интернета. Написание
того
; же кода, что выше теперь отображает другую фотографию:

view layout [image picture]
```

Вы также можете сделать так, чтобы слово "picture" относилось к файлу на вашем жестком диске или в любом другом месте, где вы хотите:

```
picture: load %bay.jpg
```

Вот еще несколько примеров создания и использования переменных слов. Введите их в интерпретатор Rebol, чтобы увидеть, как они работают, и понять, как назначенные слова могут заменять любые данные. ПРИМЕЧАНИЕ: все, что стоит после точки с запятой в коде Rebol, игнорируется интерпретатором. Он используется для включения в код удобочитаемых комментариев. Вам не нужно вводить какие-либо комментарии в интерпретатор:

```
acolor: "blue"

alert acolor ;предупреждает вас с помощью диалогового окна, отображающего
текст "blue"

print acolor ;печатает слово "blue" в консоли интерпретатора Rebol
```

.

```
anumber: 12

print anumber ;печататет цифру 12 в консоли
```

```
computation: (10 + 12) / 2

print computation ;печатате ответ расчёта строки записанной в переменной
computation
```

.

```
filename: request-file

print filename ;печатает путь и имя выбранного файла
```

.

```
chosen-color: request-color

print chosen-color
```

.

```
answer: request "Тема понятна?"

print answer
```

.

```
pick-a-date: request-date

print pick-a-date
```

.

```
userpass: request-pass

print userpass
```

.

```
webpage: http://rebol.com

browse webpage
```

.

```
file: %./test.txt

editor file
; запустит встроеный в Rebol текстовый редактор и поместит в него
; содержимое файла, если он существует
```

.

```
email-address: user@webpage.com

message:  "Привет! Как дела?"

send email-address message
```

Запомните главное о двоеточии (":"). Это говорит Rebol создать новое слово переменной и присвоить ему какое-то значение, которое будет использоваться позже..

P.S. Кстати, Вы можете использовать русские слова. Например следующий код полностью коректен:

```
мост: load http://rebol.com/view/bay.jpg
view layout [image мост]
```

Но, т.к. возможно с вашей программой будут работать не рускоговорящие пользователи, то рекомендуется всётаки использовать только английские слова в названиях. Хотя Rebol позволяет "руссифицировать" и "встроенные слова". Например можно выполнить следующий код:

```
напечатать: 'print
напечатать 123 + 654
```

Тут слово "напечатать" заменится автоматически на "print" и интерпретатором исполнится комманда

```
print 123 + 654
```

13. Блоки

В Rebol, вы можете использовать слова для представления нескольких частей данных, сгруппированных вместе. Просто заключите данные в скобки. Введите код ниже, чтобы увидеть, как это работает:

```
somecolors: ["red" "yellow" "blue" "black"]
; теперь "somecolors" содержит блок со словами
print somecolors
```

Можно через слеш ("/") указать конкретный элемент блока, например второй:

```
print somecolors/2
```

Можно узнать размер блока используя команду "length?":

```
length? somecolors
```

Так же можно узнать и размер конкретного элемента:

```
length? somecolors/2
```

В Rebol блоки очень важны. Фактически, это основная организационная единица Rebol и основная структура, в которой хранятся данные. Это очень важно - помните, что программирование - это ввод, хранение, извлечение и другие манипуляции с данными. В Rebol ЛЮБОЙ тип данных может быть назначен слову, а блоки могут содержать любую комбинацию слов и необработанных данных. Вы уже использовали блоки - ранее были вставлены скобки для отображения элементов графического интерфейса и выполнения действий с ними. Любая группа слов в квадратных скобках образует блок, и вы можете назначить слово для ссылки на этот блок. Слова, относящиеся к сложным блокам данных, можно также сгруппировать в другие блоки, просто заключив слова в квадратные скобки. Это упрощает работу с очень сложными и полезными структурами данных в Rebol.

Введите приведенный ниже код, чтобы увидеть, как можно построить и представить весь макет графического интерфейса с помощью одного слова.:

```
gui-layout: [button field text-list]

; "gui-layout" теперь содержит весь код. Отобразим с помощью "view layout":

view layout gui-layout
```

Блоки данных могут быть распределены по нескольким строкам и могут быть разделены пробелами и табуляциями, для лучшей читаемости. Просто заключите элементы в скобки:

```
gui-layout2: [
    button red "Кликни"
    field "Введи чтонибудь."
    text "Rebol - очень простой." purple
    text-list 400x300 "Линия 1" "линия 2" "другая линия"
    check yellow
]

; Теперь отобразим всё в GUI:

view layout gui-layout2
```

Блоки с большими объёмами данных могут быть сохранены в одно слово!

ПРИМЕЧАНИЕ. Стандартной практикой является создание отступа для составных блоков с последовательными позициями табуляции. Начальные и конечные скобки обычно размещаются

на одном уровне отступа. Это обычное явление для большинства языков программирования, поскольку упрощает чтение сложного кода. Например, составной блок ниже:

```
[blue red green [1 2 4 [jan feb march [monday tuesday wednesday]]]]
```

может быть записан более понятно:

```
[blue red green
    [1 2 4
        [jan feb march
            [monday tuesday wednesday]
        ]
    ]
]
```

Отступы не требуются, но они очень полезны при работе с более сложными структурами программирования.

Вот простой пример таблицы данных, содержащей информацию о расписании в блоке:

```
schedule: [
    ["John Smith" "Monday" "3:00 pm"]
    ["Dave Jones" "Tuesday" "11:00 am"]
    ["Janet Duffy" "Wednesday" "4:45 pm"]
]

; Вы можете отобразить вышеуказанный блок в графическом интерфейсе,
; используя встроенный в Rebol слово «list». Обратите внимание на
; код «data schedule» в последней строке ниже - он вставляет весь
; блок расписания, определенный выше, как данные, которые будут отображаться
; в виде списка (строка «across text 150 text 150 text 100» - это просто
; параметры(синтаксис), ожидаемый командой "list" для определения его
; графического оформления):

view layout [
    vh2 "This Week's Appointments:"
    list 600x400 [
        across text 150 text 150 text 100
    ] data schedule
]
```

Вот и начало графического приложения с "базой данных" в нескольких строках кода ... Совсем не сложно!

Многие встроенные в Rebol слова помогают управлять данными, хранящимися в блоках. Введите следующий код, чтобы увидеть, как работает сортировка (слово "sort"):

```
somecolors: ["red" "yellow" "blue" "black"]

sortedcolors: sort somecolors

; «sort» - это встроенное функциональное слово, которое по алфавиту
; (/ по порядку) сортирует элементы данного блока. Строка выше создает
```

```
; новую переменную «sortedcolors» и присваевает ей отсортированные
; значениея слов, содержащехся в «somecolors».

print sortedcolors
; Этот код выводит отсортированный список из переменной sortedcolors.

print first sortedcolors
; "first" - еще одно встроенное слово.
; Он выбирает первый элемент в данном блоке.

print find somecolors "red"
; "find" это встроенное слово, которое ищет данные в блоке.
```

Вы можете легко сохранять блоки данных на жесткий диск, читать их с веб-сервера и выполнять с ними любые другие файловые операции.

```
somecolors: ["red" "yellow" "blue" "black"]

write %colors.txt somecolors
; записывает весь блок текста, представленный "somecolors"
; в текстовый файл с именем colors.txt
```

Вот интересный пример, демонстрирующий, как Rebol может легко смешивать типы данных в блоке.:

```
somecolors: ["red" "yellow" "blue" "black"]
sortedcolors: sort somecolors

an-image: load http://rebol.com/view/bay.jpg
; загружаем из интернета картинку и кладём её в
; переменную "an-image".

append sortedcolors an-image
; "append" добавляет загруженное изображение в конец блока данных
; в настоящее время содержит простые текстовые слова, определенные
; выше. Теперь блок содержит как текстовые, так и двоичные данные
; изображения, все закреплено за одним словом - в Реболе не проблема!

; Теперь вы можете выбирать элементы из этого нового блока:
print first sortedcolors
; перчатает первый элемент - текст "black".

view/new layout [image fifth sortedcolors] do-events
; отображает пятый элемент в блоке данных -
; изображение, загруженное выше, - в GUI.

save %/c/sortedcolors.r sortedcolors
```

Вот еще несколько обозначений, с которыми следует ознакомиться при работе с последовательными данными в блоках (этот тип данных в Rebol называется «series»(сериями)):

```
sortedcolors: reduce load %/c/sortedcolors.r

view layout [image sortedcolors/5]
```

```
; "sortedcolors/5" это еще один способ обозначить пятый элемент
; в блоке данных.
```

В следующем примере приведенная выше нотация полезна:

```
sortedcolors: reduce load %sortedcolors.r

length-of-block: length? sortedcolors
; встроенное слово "length?" возвращает количество элементов
; в блоке (в данном случае 5).
view layout compose [image sortedcolors/(length-of-block)]
```

Слово «compose»(составить) позволяет оценивать и вставлять переменные в круглых скобках, как если бы они были явно введены в код программы. В приведенном выше примере код читается так, как будто sortedcolors / 5 был введен вручную.

В следующих примерах показаны дополнительные слова, используемые для обхода последовательной серии данных внутри блоков.:

```
somecolors: ["red" "yellow" "blue" "black"]
sortedcolors: sort somecolors

print insert sortedcolors "mauve"
; вставляем слово "mauve" в начало блока sortedcolors.
print sortedcolors

print remove sortedcolors
; удаляем первое словл в блоке.

print head sortedcolors
; устанавливаем маркер положения в начале блока данных

print next sortedcolors
; устанавливаем маркер положения к следующиму элементу в блоке

print last sortedcolors
; устанавливаем маркер положения к последнему элементу в блоке

print back sortedcolors
; устанавливаем маркер положения на предыдущем элементе в блоке

print tail sortedcolors
; устанавливает маркер позиции после последнего элемента в блоке
```

Тот факт, что вы можете смешивать вместе все типы данных в блоке, ссылаться на части блоков по имени и получать доступ / изменять данные в них с помощью встроенных функций, очень мощный и полезный инструмент. Блоки и переменные, слова назначенные блокам, помогают вам хранить, манипулировать и ссылаться на все данные, с которыми вы будете иметь дело в своих программах. Это похоже на присвоение существительных группам людей, мест и вещей в разговорной речи. Внутри этих групп небольшие группы и отдельные элементы могут быть названы, упорядочены и организованы иным образом-так же, как сложные приложения баз данных позволяют хранить и работать со всеми типами информации. Возможности обращения с данными таким образом безграничны.

14. Функциональные слова

Легко определить свои собственные функциональные слова (действия), если вы знаете часть встроенного словаря Rebol. Создание новых функций сравнимо с созданием собственных глагольных слов в разговорной речи. Только будьте осторожны, чтобы случайно не использовать слова, которые уже определены в языке Rebol или в вашей текущей программе. Это изменило бы значение существующего слова. Например, вы можете случайно изменить значение слова «write»(запись) на изображение на жестком диске, набрав следующее:

```
write: read %bay.jpg
; *** НЕ НАБИРАЙТЕ и НЕ ЗАПУСКАЙТЕ !!!  - это изменит значение слова
; «write» в интерпретаторе Rebol (только для текущего сеанса).
; Это пример того, чего нельзя делать. ***
; Если набрали, то закрывайте интерпретатро и/или эту программу
; и открывайте её заново.
```

Вы можете защитить все встроенные слова Rebol, набрав «protect-system». Это предупредит вас об ошибке и запретит любую попытку переопределить родные слова Rebol. Вы все равно должны быть осторожны, чтобы случайно не переопределить слова, которые вы создали. Определения слов действуют только в текущем сеансе, поэтому, если вы напутаете, все, что вам нужно сделать, это перезапустить интерпретатор Rebol.

Вот действительно важная концепция: (барабанная дробь) ... В Rebol вы можете использовать отдельные слова для представления целых блоков действий (т.е. совокупностей функциональных слов, сгруппированных вместе). Фактически, подобные блоки формируют первичные участки кода, из которых состоят программы на языке Rebol. Вот пример нескольких функциональных слов, сгруппированных вместе в блок (т.е. заключенных в скобки), и которым было присвоено новое функциональное слово:

```
some-actions: [
    alert "Это одно действие."
    print "Это другое."
    write %anotheraction.txt "А это третье."
]

; Приведенный выше код создал своего рода суперглагол, который
; относится к нескольким действиям. Вы можете выполнять действия,
; содержащиеся в любом блоке, используя команду «do». Чтобы
; выполнить все действия в приведенном выше блоке кода, просто введите:

do some-actions
```

Вы также можете включить слово «does»(делать) прямо в определение - это заставит команды (функциональные слова), содержащиеся в блоке, выполняться автоматически каждый раз, когда данное слово используется в Rebol:

```
more-actions: does [
    alert "4"
    alert "5"
    alert "6"
]

; Фактически, включив команду «does» прямо в определение слова, вы
; только что создали новую функцию (или «подпрограмму»), которую можно
```

```
; использовать как любое другое встроенное слово в Rebol! Теперь вы
; можете просто набрать это слово в интерпретаторе и он поймет, что вы
; имеете в виду. После того, как вы ввели приведенный выше код, попробуйте
; ввести слово «more-actions» в интерпретатор Rebol:
more-actions
```

Он живет своей собственной жизнью! Как и в случае с переменными словами, важно помнить о символе двоеточия - это символ, который фактически создает новое функциональное слово.

Вот пример небольшого полезного слова для очистки экрана командной строки в интерпретаторе Rebol:

```
cls: does [prin "^(1B)[J"]

; Это стандартный способ очистить консоль интерпретатора Rebol - набрать
; «prin "^(1B)[J"». Это довольно неудобно для ввода, и так же сложно
; для запоминания. Вместо этого мы можем назначить слово «cls» для выполнения
; действие - как в старых языках Basic. Теперь просто введите:

cls
```

и экран очищается - это намного проще.

Вот небольшая программа, которая создает новую коману "send-email". Она предоставляет пользователям простой интерфейс текстового запроса для отправки сообщения по электронной почте:

```
send-email: does [
    email-address: to-email request-text/title/default
        "Введите адрес электронной почты:" "user@webpage.com"
        ; строка выше создает новую переменную "email-address"
        ; "email-address" присваивается значение введённое пользователем
        ; при запросе программы
        ; дополнительные параметры "title"(заголовок) и "default"(значение по
умолчанию)
        ; настраивают информацию, отображаемую в текстовом запросе
    message: request-text
        ; строка выше создает новое слово переменной "message"(сообщение)
        ; и присваивает его запрошенному у пользователя тексту
    send email-address message
        ; указанная выше строка отправляет введённое пользователем
        ; сообщение на адрес электронной почты, указанный ранее
    alert "Ваше сообщение было отправленно."
]

send-email  ; Выполнить введённую выше процедуру
send-email  ; выполнить ещё раз, возможно с новым сообщением и/или
            ; новому адресату
```

Вышеупомянутый процесс ОЧЕНЬ важен. Это основа того, как вы выполняете более сложные и полезные действия в Rebol. Блоки действий и блоки данных составляют основу всего, что вы программируете в Rebol. А выполнение действий с данными - это то, что вы будете делать для выполнения практически всех задач программирования. В Rebol вы просто группируете данные вместе в блоки и назначаете им имя. Вы также группируете функции вместе в блоки для выполнения действий с данными и назначаете имя для этих действий. Вы даже можете

объединить большие группы данных и функций в блоки, которым могут быть назначены уникальные команды, которые выполнят поставленные задачи: хранить и управлять данными с помощью одного определяемого вами слова! Это позволяет вам создать свой собственный уникальный язык в любой программе, которую вы пишете, используя слова, которые вы определяете. Вы можете создать, например, свой собственный язык, на котором будет читаться что-то вроде «электронное письмо в полдень с главной страницей новостей с yahoo.com». Эти слова не встроены в Rebol, но им можно присвоить соответствующие действия и значения данных, чтобы сделать это предложение полностью функциональным фрагментом кода, понятным компьютеру! (Это все равно должно быть синтаксически правильным и точным). Создание значений слов таким образом называется построением «диалекта» в Rebol, и это один из отличий Rebol от других языков. Возможно, вы могли бы написать диалект для программирования роботизированных устройств, который гласит: «Пропылесосьте пол. Двигайтесь по концентрическим кругам по периметру». Вы могли бы написать на диалекте для врачей, который гласит: «Напечатайте рецепт для Джона Смита» (задумайтесь на минутку о возможностях такого типа естественного языка и гибкости ...). Перевод существующего языка Rebol на любой другой язык так же прост, как присвоение встроенных слов функций / данных словам на желаемом языке (например, перевод функции печати на французский, итальянский, испанский и русский языки так же просто, как набрать «imprime: stampa: impresion: печать: 'print "). Другие языки ориентированы на разные способы группировки и управления функциями и переменными. Как правило, другие языки используют более загадочные способы работы. Например, блоки, содержащие как полностью инкапсулированные функции, так и переменные, называются «объектами». Вы узнаете больше об "объектно-ориентированном" программировании, когда будете углубленно изучать Rebol и другие языки ...

15. Несколько способов создания функций в Rebol - передача переменных

В языке Rebol есть несколько встроенных слов, которые позволяют создавать более сложные функциональные слова. Для создания простых функций вы можете использовать команду «does»(делает), как описано выше. Но некоторые функции более сложные. Они выполняют работу с данными _переменных_. Например, следующая простая функция отображает квадратный корень из 4. Это все, что она может сделать.:

```
sqr-four:  does [print square-root 4]

; его функция аналогична той, что вы видели в предыдущем разделе.
; Слово «sqr-four» теперь присвоено действию «print square-root 4»(вывести
; квадратный корень 4) (слово «square-root»(квадратный корень) встроено в
Rebol).
; После ввода указанной выше строки в интерпретаторе Rebol введите:

sqr-four
```

Это выдаст вам ожидаемый результат 2.

А теперь скажите, что теперь вы хотите сделать что-то, используя числа, отличные от 4 ... Что, если вы хотите создать функцию, которая добавляет 4 к какому-либо другому числу, а затем вычисляет квадратный корень из этой суммы? Чтобы это работало, другой номер должен быть изменяемым и, следовательно, ему должно быть присвоено имя переменной. Затем это имя переменной можно «передать» функции. Встроенное слово «func» используется для создания функций, которым можно передавать изменяемые переменные. Синтаксис слова «func» предполагает, что за ним будут следовать два блока кода. Первый блок содержит имя (имена) передаваемых переменных. Второй блок содержит действия, которые необходимо предпринять. Вот как это выглядит:

```
func [имена принемаемых переменных] [
    действия, которые необходимо выполнить
]
```

В следующей строке создается функция, в которой принимается одна переменная имя которой внутри функции "anumber"(число). Всей этой функции присвоено слово/команда/название "sqr-var":

```
sqr-var: func [anumber] [print square-root (4 + anumber)]

; Теперь вы можете использовать команду «sqr-var», и интерпретатор
; Rebol знает, что делать с данными идущими за командой. Попробуйте следующий
код:

sqr-var 12 ; выведит "4", квадратный корень 12+4 (16)
sqr-var 96 ; выведит "10", квадратный корень 96+4 (100)
```

Вывод квадратного корня из суммы 4+некое_число может показаться вам не таким захватывающим, но он помог проиллюстрировать один из наиболее важных методов, используемых во всех современных языках программирования. Процесс передачи переменных параметров функциям является фундаментальной частью всего современного программирования. Это, пожалуй, самый распространенный элемент в современных языках, и понимание того, как это делается, необходимо. Ниже приведены еще несколько интересных примеров из реальной жизни..

следующая строка создает простую функцию для отображения изображений:

```
display: func [filename] [view layout [image load filename]]

; Он принимает имя файла изображения в качестве переданного параметра
; (%somefile.jpg, %somefile.gif, %somefile.png или %somefile.bmp), а затем
; создает окно для отображения изображения. Этот набор действий закреплен за
словом
; «display». Все просто, правда? Теперь вы можете использовать слово
«display» так:

image1: to-file request-file/title trim {
    Выберете файл с изображением у себя на диске:} ""
; запросили имя файла у пользователя

display image1
; отобразили изображение с использованием новой функции/команды

display http://rebol.com/view/bay.jpg
; отобразили изображение из интернета используя его адрес (url)

display %bay.jpg
; отображает изображение, которое было сохранено на
; жесткий диск ранее в этом руководстве
```

Как только функция/команда/слово «display» определено в ваших программах, вы можете использовать его, как если бы оно было встроенным в Rebol.

Вот пример, который запрашивает у пользователя два URL-адреса веб-сайтов, а затем открывает эти сайты в отдельных окнах браузера:

```
openwebsite: func [nameofwebsite] [browse nameofwebsite]
; Эта строка создаёт новую функцию/команду/слово "openwebsite", которая
```

```
; передаёт URL(адресс сайта) встроеной в Rebol команде "browse",
; которая открывает интернет браузер "по умолчанию".

website1: request-text/title "Введите адрес сайта (URL):"
; Эта строка создаёт новую переменную "website1" и кладёт туда то,
; что вернула встроенная в Rebol команда "request-text"(запрос текста).

website2: request-text/title "Введите адрес сайта (URL):"
; Запрашивает ещё текст икладёт его в переменную "website2"
openwebsite website1
; Эта строка вызывает ранее созданую функцию "openwebsite" передав
; ей как аргумент содержимое переменной "website1"

openwebsite website2
; Опять вызываем туже функцию, но теперь передаём содержимое переменной ж4;
website2
```

В приведённом примере слово «openwebsite» присвоено определению новой функции. «Website1» и «website2» - это имена для переменных, передаваемых этой функции. Ниже приведен вариант, в котором всему процессу присваивается одно слово:

```
display-website: does [
    openwebsite: func [nameofwebsite] [browse nameofwebsite]
    website: request-text/title "Введите адрес сайта (URL):"
    openwebsite website
]

; Теперь вы можете использовать одно слово «display-website» в своих
; программах для выполнения _всего_ этого блока кода.
display-website
```

Необходимо привыкнуть к приведенному выше синтаксису и образу мышления. Помните, что работа со разными типами данных - это основное, что вы будете делать. Передача данных по средствам переменных, является основным способом взимодействия внутри программы. Rebol упрощает работу с данными, используя встроенную поддержку наиболее распространенных типов данных. Ваша основная цель - понять, как вводить, обрабатывать и выводить эти данные. Использование функций и переменных, как описано выше, является фундаментальной частью этого процесса.

16. Условные операции

Программы часто должны принимать решения на основе ввода пользователя, состояний программы, содержимого переменных, данных и т.д. «Если пользователь выбирл ЭТО, то выполни ТО». «Если прошло определенное время, то сохрани данные на жесткий диск». Предоставление компьютером множества возможных действий на основании множества ожидаемых условий - это фундаментальный метод программирования, используемый на всех языках.

Математические операторы, такие как "=" "<" ">" "<>" (равно, меньше, больше, не равно), обычно используются для выполнения условных операций. Введите следующий код, чтобы увидеть, как работают операторы условий:

```
if now/time > 12:00 [alert "Уже после полудня."]
; now/time это разновидность или «доработка» встроенн
; функция «now»(сейчас), которая возвращает только текущее время.
```

Вот более сложный пример:

```
daily-calories: to-integer request-text/title {Сколько калорий вы потребили
сегодня?}
; получаем некоторую информацию от пользователя и записываем в
; переменную «daily-calories»(ежедневные калории)
; Встроенная функция "to-integer"(к целому числу) позволяет убедиться,
; что Rebol примет эту информацию только как целое число.
; Символы "{}" вокруг текста заголовка работают с
; так же, как кавычки, но позволяет тексту состоять из нескольких
; строк. Это очень важно.
if daily-calories > 2500 [alert "Вам сегодня более неследует есть."]
```

Встроенное слово Rebol "either"(либо/или) выбирает между двумя блоками функций для выполнения, на основе условия. Его синтаксис:

```
either {условие} [
    блок выполняемый если условие истина
][
    блок выполняемый если условие ложно
]
```

Вот небольшой пример:

```
either now/time > 8:00am [alert "Время вставать!"] [
    alert "Можно ещё поспать."]
```

Это вариант приведенного выше примера, который позволяет вам установить время пробуждения:

```
wake-up: to-time request-text/title "В котором часу нужно проснуться?"

either now/time > wake-up [alert "Время вставать!"] [
    alert "Можно ещё поспать"]
```

Встроенная команда Rebol «switch»(переключатель) позволяет выбирать между многочисленными блоками, выполняемыми для определённых значений. Его синтаксис:

```
switch/default {значение} [
{значение1} [блок выполняемый если "значение" = "значение1"]
{значение2} [блок выполняемый если "значение" = "значение2"]
{значение3} [блок выполняемый если "значение" = "значение3"]
; и т.д....
] [блок выполняемый если "значение" неравно не одному из выше перечисленных]
```

Вы можете сравнить любое количество вариантов с значением указанным в операторе "switch" и запустить блок кода для каждого совпадающего значения. Вот пример:

```
favorite-day:  request-text/title "Какой у тебя любимый день недели?"

switch/default favorite-day [
    "понедельник" [alert "Понедельник - день тяжёлый! Неделя только
началась..."]
    "вторник" [alert "Вторник и четверг вполне хороши..."]
    "среда" [alert "Среда «перевальная» - пол недели позади!"]
    "четверг" [alert "Вторник и четверг вполне хороши..."]
    "пятница" [alert "Пятница -показница! Ура!"]
    "суббота" [alert "Суббота - выходной :)"]
    "воскреенье" [alert "Воскреенье - выходной :)"]
] [alert "Вы не ввели название дня!"]
```

«switch» очень часто используется, потому что в программах порой требуются принять одно из множества решений, на основании какогото выбора.

Rebol включает в себя богатый набор слов и функциональных структур, которые помогут вам оценивать условия во всех типах и в разных ситуациях и со всевозможными типами данных. Понимание того, как это использовать - важная часть в изучении языка.

17. Зацикливание

Часто требуется, чтобы программы многократно выполнялись полностью или частично, например проверяя достижение какогото условия. Фактически, в большинстве больших приложений компьютер часто выполняет множество повторяющихся операций. Например, в программе напоминания требуется постоянно проверять время и дату, чтобы определить, пора ли сообщать пользователю о запланированном событии или нет. В других программах компьютеру может потребоваться многократное сканирование каких то данных или многократный запрос/ответ на ввод пользователя. Для решения данных задач существуют «циклические» структуры, предоставляющие способ систематического повторения действий.

Встроенное слово «forever»(всегда) создает простой повторяющийся цикл. Его синтаксис:

forever [блок действий для повторения]

Следующий пример кода создает простой таймер, который предупреждает пользователя по истечении одной минуты. Он использует бесконечный цикл для постоянной проверки времени:

```
alarm-time: now/time + 60
; присваеваем переменной значение времени, которое будет через 60 секунд
; т.е. текущее время + 60
forever [if now/time = alarm-time [alert "Прошло 60 секунд." break]]
```

Обратите внимание на слово «break»(перерыв) в приведенном выше примере. Он выходит из цикла, чтобы программа не продолжалась вечно, после отображения предупреждения.

Вот более интерактивная версия, в которой используется некоторая информация, введённая пользователем. Сердцем программы по-прежнему является "бесконечный" цикл в конце:

```
event-name: request-text/title "О чём вам напомнить?"
; запрашиваем предмет напоминания
seconds: to-integer request-text/title trim {
    Через сколько секунд напомнить?}
; запрашиваем количество секунд, через которое напомнить
```

```
alarm-time: now/time + seconds
; устанавливаем в переменную alarm-time время напоминания
alert join "Сейчас " [
    now/time ", а напоминание будет в " alarm-time "."
]
; выводим сообщение
forever [
    if now/time = alarm-time [
        alert join "Сейчас " [
            alarm-time ", и " seconds
            " секунд прошли. Время напомнить о " event-name
            ]
    break
    ]
]
; безконечный цикл forever постоянно сравнивает текущее время
; со временем сигнала и когда они совпадут - отображает
; предупреждение с текстом напоминания
```

Обратите внимание на слово «join»(присоединенять), используемое в нескольких строках выше. Его формат синтаксиса:

```
join {данные} [к блоку данных]
```

Используйте эту команду, чтобы объединяет вместе переменные, текст, блоки и другие биты данных, чтобы их можно было вывести распечатать вместе, отобразить и иным образом обработать для формирования как единая часть данных. Очень полезно! Вариант соединения - "rejoin". Он соберёт в один блок все отдельные элементы данных. Его синтаксис:

```
rejoin [элемент1 элемент2 элемент3 ...]
```

Теперь вернемся к циклам. Вот простой бесконечный цикл, который отображает/обновляет текущее время в графическом интерфейсе. Обратите внимание на отступ блока:

```
view layout [
    timer: field
    button "Пуск" [
        forever [
            set-face timer now/time
            wait 1
        ]
    ]
]
```

Вышеупомянутая программа содержит два виджета: текстовое поле, которому присвоена метка переменной «timer», и кнопка со словом «Пуск» на нем. Блок действий для кнопки содержит бесконечный цикл, который повторяет два действия повторно (до тех пор, пока пользователь не закроет окно): встроенное слово «set-face» устанавливает текст в поле «timer» значение текущего времяни, а команда «wait 1» ожидает 1 секунду, затем цикл повторяется.

Часто данные, обрабатываемые при каждом повторении цикла, должны изменяться. Как и большинство языков, Rebol включает в себя множество функций и программных структур, которые

позволяют вам перебирать блоки данных и выполнять операции, используя последовательно изменяемые значения. Распространенной структурой цикла во многих языках является структура «for». Она позволяет вам указать начальное значение, конечное значение, инкрементное значение и имя переменной для хранения текущего значения, так что вы можете циклически перебирать последовательно изменяющиеся значения контролируемым образом. Вот базовый синтаксис Rebol для цикла for:

for {имя переменной для хранения текущего значения} {начальное значение} {конечное значение} {шаг изменения значения} [блок кода для выполнения, который может использовать текущее значение переменной]

Вот несколько простых примеров. Обязательно введите их, чтобы увидеть, как они работают:

```
for counter 1 10 1 [print counter]
; переменная "counter" изменяется от 1 до 10 с шагом 1
; т.е. за каждый цикл к переменной прибовляется 1

for counter 10 1 -1 [print counter]
; переменная "counter" изменяется от 10 до 1 с шагом -1
; т.е. за каждый цикл от переменной отнимается 1

for counter 10 100 10 [print counter]
; переменная "counter" изменяется от 10 до 100 с шагом 10
; т.е. за каждый цикл к переменной прибовляется 10

for counter 1 5 .5 [print counter]
; переменная "counter" изменяется от 1 до 5 с шагом 0.5
; т.е. за каждый цикл к переменной прибовляется 0.5

for timer 8:00 9:00 0:05 [print timer]
; переменная "timer" содержит время изменяемое с 8:00
; до 9:00  с шагом в пять минут (0:05)
; т.е. за каждый цикл к переменной прибовляется пять минут
for dimes $0.00 $1.00 $0.10 [print dimes]
; переменная "dimes" содержит денежное значение и
; изменяется от нуля до одного доллара, с шагом десять центов

for date 1-dec-2005 25-jan-2006 8 [print date]
; переменная "date" содержит дату (число-месяц-год) которое
; меняется начиная с 1 декабря 2005 года, до 25 января 2006 года,
; с шагом в 8 дней
for alphabet #"a" #"z" 1 [prin alphabet]
; переменная "alphabet" содержит символ из таблицы ASCII
; начиная от "a" и заканчивая "z" с шагом в один символ
```

Обратите внимание, что Rebol легко перебирает различные типы данных, имея собственное понимание этих данных. Он может автоматически увеличивать даты, деньги, время и т. Д. На других языках реализовать это может быть непросто.

Также обратите внимание на использование слова «prin» в последнем примере. Он работает как «print», за исключением того, что он не переводит курсор на новую строчку (не вставляет в конце символ Enter), т.е. печатает каждый следующий выводимый символ рядом с предыдущим, а не на новой строке).

Вот пример цикла «for», который отображает именя первых пяти файлов в текущей папке на жестком диске:

```
files: read %.
```

```
; в переменную "files" помещается список файлов текушего директория
; точка (".") обазначает текущий директорий

for count 1 5 1 compose [print files/(count)]
; печатаем поочерёдно с 1 по 5 элемент в блоке "files"
```

В приведенном выше примере «files/1» - это первый элемент в списке файлов, «files/2» - второй и так далее. Обратите внимание на слово «compose», используемое в цикле for. В приведенном выше примере при первом прохождении цикла код читается так, как если бы [print files/1] были введены вручную и т.д.

«Foreach» (для каждого) - еще одна полезная вкоманда цикла, встроенная в Rebol. Она позволяет легко просматривать блок данных выбирая каждый элемент. Формат его синтаксиса следующий:

foreach {имя переменной, куда будет записываться каждый элемент блока} [блок данных] [блок команд и функций, которые должны выполняться для каждого элемента в блоке данных, при этом "имя переменной" будет содержать поочерёдно каждый элемент]

В приведенном ниже примере выводится имя каждого файла в текущем каталоге:

```
folder: read %.
; в переменную "folder" считывается перечень элементов (файлов/папок)
; текущего каталога

foreach file folder [print file]
; пока не переберуться все элементы переменной "folder",
; выполняется цикл в котором переменная "file" будет принемать
; значение всех элементов переменной "folder" поочереди
```

Следующая строка читает и печатает все сообщения по очереди из почтового ящик пользователя:

```
foreach mail read pop://user:pass@website.com [print mail]
```

«While» (до тех пор, пока) - другая полезная команда цикла, встречающаяся во многих языках программирования. Она проверяет условие и если оно "истина", то выполняет блок функций до тех пор, пока условие не станет "ложно" или цикл непрервётся. Синтаксис команды следующий:

```
while [условие] [блок функций, которые будут выполняться при выполнении
условия]
```

Простой пример:

```
x: 1  ; создали переменную и записали туда 1
while [x <= 5] [
    alert to-string x
    x: x + 1
]
```

По Русски код звучит так: "x изначально равно 1. До тех пор пока x меньше или равно 5 -

отображать x и добавлять 1 к его значению" (т.е. программа отображает счет от 1 до 5)

ПРИМЕЧАНИЕ. В Rebol код «x: x + 1» добавляет 1 к текущему значению x. Это одно из наиболее часто используемых выражений в программировании. Также обратите внимание на слово «to-string». Он преобразует числовое значение в «x» в текстовое («строковое») значение. Это необходимо, потому что слово «alert» отображает только строковые типы данных.

Вот еще несколько примеров цикла while:

```
while [not request "Завершить программу?"] [
    alert "Выберете YES для завершения."
]
```

В приведенном выше примере "not" меняет значение данных, полученных от пользователя (т.е. "yes"(да) становится "no"(нет) и наоборот).

```
alert "Пожалуйста выберете текущию дату" while [request-date <> now/date] [
    alert join "Выберете снгодняшнию дату. Сегодня " [now/date]]

while [request-pass <> ["secret" "password"]] [
    alert "Имя пользователя 'secret', а пароль 'password'"]
```

В приведенном ниже примере используется несколько циклов, чтобы выводить предупреждение о необходимости кормить кошку, каждые 6 часов с 8:00 до 20:00. Используем цикл for для расчёта времени получения увидомления, цикл while для постоянного сравнения расчётного времени с текущим, и цикл forever, чтобы делать одно и то же каждый день, непрерывно. Обратите внимание на отступ:

```
forever [
    for timer 8:00am 8:00pm 6:00 [
        while [now/time <= timer] [wait .5]
        ; ничего не делаем, пока не время кормить кошку
        ; Действие "подождите 0,5" дает пользователю
        ; шанс закрыть программу

        [alert join "Сейчас " now/time ".  Время кормить котика."]
        ; если цикл while завершился, то значит пора кормить кошку
        ; после предупреждения for пересчитает время и while
        ; запустится снова
    ]
]
```

18. Работа с более длинными примерами

Большинство программ значительно больше, чем примеры, показанные ранее. Ниже приведен пример более сложной программы. Он позволяет пользователю вводить Интернет-адрес веб-камеры и отображать потоковое видео с нее. Он отображает видеоизображения в окне с несколькими кнопками, которые используются для управления размером, расположением и действием на экране. Чтобы не вводить эту программу каждый раз при ее запуске, ее можно сохранить как текстовый файл. Когда вы сохраняете программу Rebol в текстовый файл, код должен начинаться со следующего фрагмента текста:

```
REBOL []
```

Этот текст сообщает интерпретатору Rebol, что файл содержит программу на языке Rebol. Он должен быть включен в начало любой программы на языке Rebol, сохранённой в файл. Вы можете включить в этот блок дополнительную информацию о программе, такую как название, версия, автор, описание и т.п., но это не обязательно.

Введите или скопируйте/вставьте исходный код ниже, в текстовый редактор, например Блокнот Windows. Вы также можете использовать встроенный текстовый редактор Rebol, набрав «editor %webcam.r». Сохраните его как файл с именем «webcam.r».

(Примечание: этот пример включает в себя несколько команд и диалектов Rebol, которые еще не были включены в это руководство. Цель примера - просто показать более объёмную и сложную программу, который можно через буфер обмена поместить в отдельный файл. Не переживайте, если вы не понимаете код.)

```
Rebol [Title: "Webcam Viewer"]
; try http://www.onedir.com/cam1.htm for some webcam links.
temp-url: "mms://www.onedir.com/cam1"
while [true]  [
    webcam-url: to-url request-text/title/default trim {
        Enter the web cam URL:} temp-url
    either attempt [webcam: load webcam-url]
        [break]
        [either request [trim {
            That webcam is not currently available.} trim {
            Try Again} "Выход"]
            [temp-url: to-string webcam-url]
            [quit]
    ]
]
resize-screen: func [size] [
    webcam/size: to-pair size
    window/size: (to-pair size) + 40x72
    show window
]
window: layout [
    across
    btn "Стоп" [webcam/rate: none show webcam]
    btn "Старт" [
        webcam/rate: 0
        webcam/image: load webcam-url
        show webcam
    ]
    rotary "320x240" "640x480" "160x120" [
        resize-screen to-pair value
    ]
    btn" Выход" [quit] return
    webcam: image load webcam-url  320x240
    with [
        rate: 0
        feel/engage: func [face action event][
            switch action [
            time [face/image: load webcam-url show face]
            ]
        ]
    ]
]
view center-face window
```

После того, как вы сохранили программу webcam.r, вы можете запустить ее удобным вам способом, например:

1) Если вы уже установили интерпретатор Rebol в Windows, просто найдите значок файла webcam.r в проводнике файлов и дважды щелкните/кликните его. Интерпретатор Rebol автоматически выполняет скрипт. По умолчанию во время первоначальной установки Rebol все файлы с расширением «.r» связаны с интерпретатором. Их можно щелкнуть и запустить, как если бы они были исполняемыми программами, как файлы «.exe». Это наиболее распространенный способ запуска скриптов Rebol, и он работает одинаково во всех основных операционных системах.

2) Введите "do %webcam.r" в интерпретатор Rebol.

3) Используйте такую программу, как XpackerX (http://www.marmaladefoo.com), или WinRAR (опции запуска в SFX), чтобы упаковать и распространить программу. XpackerX позволяет вам обернуть интерпретатор Rebol и программу webcam.r в один исполняемый файл, который имеет кликабельный значок, и автоматически запускает оба файла. Это позволяет вам создать исполняемую программу Windows в виде одного файла, которую можно распространять и запускать, как любое другое приложение. Просто щелкните по нему и бегите ...

4) Купить коммерческую версию Rebol "SDK", которая компилирует приложения.

ВАЖНО: Чтобы отключить запрос безопасности, который по умолчанию постоянно запрашивает разрешение на чтение/запись жесткого диска, введите «secure none» в интерпретаторе Rebol, а затем запустите программу с «do {filename} r». Выполнение "rebol.exe -s {filename}" делает то же самое. «-S» запускает интерпретатор Rebol без включения каких-либо функций безопасности, заставляя его вести себя как типичная программа Windows.

Возможность сохранять, запускать и распространять свои программы очень важна. Вам следует хорошо ознакомиться с приведенными выше вариантами.

19. Встраивание двоичных данных

Вам часто придется использовать изображения и другие данные, состоящие не из текста. Например, игры часто используют графические и звуковые файлы как часть своего интерфейса. Как вы видели ранее, у Rebol есть множество способов загрузить такие файлы. Вы можете прочитать его с жесткого диска, вы можете загрузить его из Интернета, вы можете прочитать его практически с любого другого локального или сетевого носителя и т.д. Однако при распространении ваших программ эти методы не всегда желательны. Вы же не хотите, чтобы ваша программа неработала без интернета, или чтобы пользователь загружал изображения и звуки каждый раз, когда он играет в игру. Возможной альтернативой является создание zip-файла или аналогичного пакета, который включает вашу программу и все ее вспомогательные файлы. Однако установка таких пакетов может быть сложной задачай для пользователя, и это ненужный шаг. Rebol предоставляет возможность для кодирования и включения внешних файлов в ваши программы. Чтобы увидеть, как это работает, используйте приведенный ниже код:

```
Rebol [Title: "Встраивание двоичных файлов"]

system/options/binary-base: 64
file: to-file request-file/only
data: read/binary file
editor data
```

Наберите эту программу в текстовом редакторе, сохраните ее как текстовый файл с названием «embed.r», а затем запустите с помощью любого из способов, описанных в предыдущем разделе (попробуйте ввести «do %embed.r» в консоле интерпритатора Rebol). При запуске программа

предложит вам выбрать файл, прочитает его, а затем отобразить его кодированное представление во встроенном редакторе. Вы можете скопировать и вставить этот текст в свою программу, присвоить ей имя переменной и использовать ее так, как если бы она была прочитана прямо с жесткого диска или другого носителя.

Вот ещё пример - загрузите изображение по ссылке ниже (вы можете использовать свой веб-браузер, чтобы перейти по URL-адресу) и затем сохранить изображение на свой жесткий диск. Или вы можете загрузить его с помощью Rebol, как показано ранее в руководстве:

```
browse http://musiclessonz.com/test.png
```

После того, как вы загрузили файл, используйте указанную выше программу, чтобы выбрать и преобразовать его в двоичные данные. У вас должна получиться следующая распечатка:

```
64#{
iVBORw0KGgoAAAANSUhEUgAAAFUAAABkCAIAAAB4sesFAAAAE3RFWHRTb2Z0d2Fy
ZQBSRUJPTC9WaWV3j9kWeAAAAU1JREFUeJztlzEOgzAQBHkaT7s2ryZUUZoYRz4t
e9xsSzTjEXIktqP3trqqn3766/+qjqjBb86nUyZP
lM7kidKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE
v/ToX+4Pt0ubiNKZPFE6Ux5q/O/436lkh6affvrpp38ZRT/99Ov6f4tPPqX8Ps
/meidCZPlM7kidKZPFE6kydKZ/JE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE
j/7l/nC7tIkonekLdXm9dafSmeinn376D/rpp5/vv1GqBkT37+FR/9yf7hd2kSU
zuSJ0pk8UTpTJ4onSmTxROlMniidyROlM3midCZPlM7kidKZPFE6kydKZ/JE6kyd
KZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydK
//jR33v09x79vUd/73XvfwNmVzlr+eOLgmAABJRU5ErkJggg==
}
```

Теперь скопируйте и вставьте эти данные в интерпретатор Rebol и присвойте ему метку переменной, примерно так:

```
picture: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAFUAAABkCAIAAAB4sesFAAAAE3RFWHRTb2Z0d2Fy
ZQBSRUJPTC9WaWV3j9kWeAAAAU1JREFUeJztlzEOgzAQBHkaT7s2ryZUUZoYRz4t
e9xsSzTjEXIktqP3trqqn3766/+qjqjBb86nUyZP
lM7kidKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZ
PFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydKZPFE6kydK
zuSJ0pk8UTpTJ4onSmTxROlMniidyROlM3midCZPlM7kidKZPFE6kydKZPFE6kydK
//jR33v09x79vUd/73XvfwNmVzlr+eOLgmAABJRU5ErkJggg==
}
```

Вы получите ответ, что изображение было определено и создано:

```
make image! [85x100 #{
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF...}
```

Теперь вы можете использовать переменную «picture»(картинка), как любые другие данные. Отобразить его, сохранить, перенесити между подключенными к сети компьютерами и т.д. Либо просто отобразить:

```
view layout [image picture]
```

Для примера - фото собаки, которое изобразиться на экране и сохраниться на диске как изображение в формате PNG (dog.png):

```
dog: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAFkAAACNBAMAAAAuisulAAAAMFBMVEUeEw6kh3OM
TRZDSE/RxbWFal1KLiCtkoakZSXW0cm9mnlCNzq5no5oSDTg29B0UkzBjRe5AAAA
CXBIWXMAAC4jAAAuIwF4pT92AAAQ80lEQVR4nG1Yf2gjZ3r+kiW6gjHJZHdOjuLU
W0GMKRgShG7/UIVqqu2eUtd3c9pvpe6uGJ32dnqca+32dnqca+glubczCBZVsQhGkX
9UR369zgEK ( ... вырезанный текст ... )
zIc9zjc8xzH9mT7w2DCCDGLc+tidX3q8/egxLwJDu5fPJkNcwL0/cK
NyJ5MYHCZs9XkdE/9H1C1ye555P11G43ybTpn/v3XrfwFzPoFQpggI
EQAAAABJRU5ErkJggg==
}

view layout [image dog]
save/png %dog.png dog
```

Размер приведённого выше кода можно значительно ументшить, используя встроенное в Rebol слово «compress»(сжать) Встроенные функции сжатия и распаковки Rebol работают только с текстовыми данными, поэтому предоставленная ранее программа для внедрения двоичных файлов должна быть доработанна следующим образом:

```
REBOL [Title: "Встраивание двоичных файлов"]

system/options/binary-base: 64
file: to-file request-file/only
if not file [quit]
uncompressed: read/binary file

compressed: compress to-string uncompressed
; В этой строке мы конвертируем двоичные(бинарные)
; данные в текст(to-string) и сжимаем(compress) их.

alert rejoin ["Не сжатый размер:  " length? uncompressed
    " байт.  Сжатый: " length? compressed " байт."]
editor compressed
```

ВАЖНО: Чтобы использовать сжатую версию данных выше, вам необходимо распокавать и пеобразовать их из текста в двоичный(бинарный) код. Для этого используйте следующий код:

```
to-binary decompress {compressed data}
```

Итак, используя указанные выше переменные, следующие две строки кода отображают одно и то же изображение:

```
view layout [image load uncompressed]
view layout [image load to-binary decompress compressed]
```

Вот полный пример, демонстрирующий разницу в размере между необработанными и сжатыми встроенными данными:

```
; Вот несжатое встроенное изображение:

image-uncompressed: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAP8AAAEsCAIAAACDt/KoAAAAE3RFWHRTb2Z0d2Fy
ZQBSRUJPTC9WaWV3j9kWeAAAB9FJREFUeJzt3bGRHEkQBMETDaKd5v8UmlmzIaoz
MGtZ4Q ( ... вырезанный текст ... )1l/drL+rWX9Wsv69de1q+9rF97Wb/2
sn7t9T/igalLLsvMjgAAAABJRU5ErkJggg==
}

; Вот сжатое встроенное изображение:

image-compressed: load to-binary decompress 64#{
eJzrDPBz5+WS4mJgYOD19HAJAtL/GRgYdTiYgKzm7Z9WACnhEteIkuD8tJLyxKJU
hiBXJ38f/bDM1PL+m2IVDAzsFz1dHEMq5ry9u3GijKcAy0Fh3kVzn/0XmRW5WXGV
sUF25EOmKwrSjrrF9v89o//u+cs/IS75763Tv7ZO/5qt//p63LX1e9fEV0fu/7ap
7m0qZRIJf+2DmGZoVER5MQiz+ntzJix6kKnJ6CNio6va0Nm0fCmLQeCHLVMY1Ljm
TRM64HLwMpGK/334Hf4n+vkn+1pr9md7jAVsYv+X8Z3Z+M/yscIX/j32H7sl/0j3
KK+of/CX8/X63sV1w51WqNj1763MjOS/xcccX8hzzFtXDwyXL9f/P19/f0vxz4f2
OucaHfmZDwID+P7Hso/5snw8m+qevH1030pG4kr8fhNC4f/34Z89ov+vHe4vAeut
SsdqX8T/OYUCv9iblr++f67R8pp9ukzLv8YHL39tL07o+3pekn1h/dDVBgzLU/d3
9te/Lki4cNgBmA6/lO+J/RPdzty8Rr5y94/tfOxsX6/r8xJK0/UW9vlH93/9oAzR
e09yKIUBVbT9/br/U/m7x6CU98VAAJS2ZPPF/197eEDhtfs9vX9rDzc6/v3qzUyo
nJA/dz76Y77tHw+w3gXlbEMpDKihza/+7/o/c3+DU54tDwsobR2/fXR/qYXBiV8T
t3eDEmpA/d9LDASK0y/tnz+H/Ynmt78E1vti7lAKA6pouxz/X7v+uR045ZFdRE6x
1q21pG7NiSzx1f5R40pvvdNn+oB1P4Onq5/LOqeEJgCemFy1KQgAAA==
}

view/new layout [image image-uncompressed]
view/new layout [image image-compressed]
do-events
```

Сжатая версия изображения точно такая же, но намного меньше, и соответственно лучше её включить в программу. Когда вы используете такое сжатие, просто не забудьте включить слова/команды «to-binary decompress» при загрузке встроенных данных.

Используя предоставленную ранее программу кодирования(встаивания) данных, вы можете преобразовать любой тип файла во встраиваемые текстовые данные. Изображения, звуки, видео и даже целые исполняемые программы могут быть включены в ваш код!

20. Модульное программирование и повторное использование кода

Rebol предоставляет фантастический набор простых в использовании инструментов программирования, встроенных прямо в язык. Как вы видели, встроенные слова/команды в Rebol могут выполнять полезные действия, которые могут быть основой простых программ. Вы даже можете использовать интерпретатор как мощную небольшую служебную программу для швейцарского армейского ножа, которая не требует слодного программирования (например, как многоплатформенный текстовый редактор, калькулятор, средство чтения/отправления электронной почты, средство просмотра/редактор изображений, командный интерфейс для копирования/вставки файлов, загрузчик ftp и т.д.). Однако для более крупных программ встроенные возможности - это просто строительные блоки. Чтобы создавать более сложные приложения, вам необходимо _создавать_ новые функции, комбинируя и используя встроеные слова, грамматические структуры и простые фрагменты кода. Элементы языка - это просто сырье,

которое можно соединить для достижения более глобальных и конкретных целей. С этой целью очень важной концепцией программирования является (барабанная дробь ...): повторное использование существующего кода.

Никогда не изобретайте колесо заново. Повторное использование фрагментов существующего кода необходимо, если вы хотите стать продуктивным программистом. Создание функций - это основной способ реализации повторного использования кода: как только вы создаете новое функциональное слово для выполнения заданного действия, вы можете скопировать его определение и использовать его снова и снова в своих программах. Это избавляет вас от необходимости заново изобретать эти действия каждый раз, когда они необходимы в ваших программах. Это также снижает вероятность появления ошибок - старый проверенный код с меньшей вероятностью будет содержать ошибки, если он уже был протестирован и использовался в различных ситуациях и программах ранее.

Как было показано ранее, встроенное слово «do» загружает и запускает код Rebol, сохраненный в текстовом файле. Вы можете использовать его для импорта существующих модулей кода, как если бы этот код был введен в вашей программе. Этот существующий код может содержать определения функций и переменных, новые программные структуры и даже полные программы любой длины и сложности. После того, как эти слова и определения были импортированы в вашу программу, вы можете использовать включенные функции и переменные, как если бы они были встроенными словами языка Rebol. Вам даже не обязательно знать, как они были созданы. Попробуйте ввести следующий пример кода и сохранить его в «play_sound.r».

```
Rebol [title: "play-sound"]

play-sound: func [sound-file] [
    wait 0
    ring: load sound-file
    sound-port: open sound://
    insert sound-port ring
    wait sound-port
    close sound-port
]
```

Теперь, когда вы хотите воспроизвести звук, вы можете включить код в свою программу:

```
do %play_sound.r
; И теперь используйте функцию "play-sound", как и любую другую встроенную
; функцию (синтаксис: "play-sound {sound-file}"):

play-sound %/C/WINDOWS/Media/chimes.wav
```

Легко, правда? Вам нужно всего лишь один раз ввести строку "do %play_sound.r" в вашей программе. После этого определяется слово "play-sound", и вы можете использовать его где угодно:

```
do %play_sound.r

alert "Вот звук:"
play-sound %/C/WINDOWS/Media/chimes.wav

alert "И ещё один звук:"
play-sound %/C/WINDOWS/Media/chord.wav

alert "Теперь вы можете выбрать .wav фаил с диска:"
play-sound to-file request-file/file %/C/WINDOWS/Media/tada.wav
```

Вся эта концепция становится намного более полезной с осознанием того, что (последняя барабанная дробь ...): вам не нужно все писать самостоятельно! Во всем мире существует сообщество разработчиков, работающих над созданием полезных фрагментов кода. Поиск и изучение возможностей модулей кода, функций и диалектов, созданных другими программистами, останется в центре вашего внимания на начальном этапе обучения и будет играть важную роль в ваших последующих разработках при приобретении опыта.

Многие автономные модули кода Rebol были созданы для расширения встроенных возможностей Rebol, их количество продолжает расти по мере развития языка. Изучение диалектов и частей программ, созданных другими, сделает вас более способным программистом. Существующие модули кода могут помочь вам с легкостью выполнять высокоуровневые, сложные и специфические задачи, так что вам не придется начинать с нуля в каждой программе.

Таким образом, обучение программированию во многом похоже на обучение использованию других типов технологий, существующих в нашем обществе. Например, чтобы позвонить другу по телефону, вам не нужно заново изобретать телефон и все его электронные компоненты. Вам не нужно производить какие-либо из этих предметов, и вам не нужно прокладывать километры кабеля. Вам просто нужно знать, как пользоваться существующей телефонной системой. Кроме того, вы можете использовать эту систему для более сложных операций, которые будут доступны вам на другом уровне. Владельцы бизнеса полагаются на телефон для связи с клиентами и сотрудниками, не заботясь о том, как работает система. Они просто используют его функционально как часть своих бизнес процесов более высокого уровня. Существующие модули кода могут работать для программистов аналогичным образом. Они делают доступными высокоуровневые функции, которые можно использовать для создания еще более высокоуровневых, специфических и сложных приложений. Вам просто нужно знать, где их искать и как ими пользоваться.

Чтобы использовать модули кода, необходимо сначала изучить язык, чтобы вы могли понимать код, написанный другими разработчиками. Возможно, вам придется изменить и расширить код, написанный другими, чтобы он более точно соответствовал вашим задачам. Однако в большинстве случаев вы можете просто научиться использовать функции/слова/команды в импортированном модуле или диалекте, выполнив команду «do {filename}», и вы начнете работать с полностью новым расширением языка. Вот пример:

Веб-сайт

```
browse http://www.dobeash.com/rebgui.html
```

предлагает несколько бесплатных модулей расширения для языка Rebol. Они например, предоставляют модуль под названием «RebGUI» который расширяет и без того мощный синтаксис графического интерфейса пользователя, встроенный в Rebol («view layout...»). Используя RebGUI, можно легко отображать графические виджеты, которые изначально отсутствуют в Rebol. RebGUI конструирует эти компоненты из встроенных исходных материалов Rebol и делает их повторно используемыми в ваших собственных программах. Чтобы использовать RebGUI, просто загрузите файлы с указанного выше веб-сайта и разархивируйте их, затем введите следующий код и сохраните его как C: \ rebgui_example.r. Вы можете запустить его любым из способов, описанных в предыдущем разделе («do %tour.r» и т. Д.):

```
REBOL []

do %rebgui.r

display "Сетка" [
    table #WH options [
        "День" left .5 "Время" left .3 "Имя" left .3
    ] data [
```

```
            Понедельник 9:00 "Петя" Вторник 9:30 "Вася" Среда 10:00 "Женя"
    ]
]

do-events
```

В приведенном выше коде используются некоторые новые команды, которые не являются частью языка Rebol. В частности, функции и переменные «display», «table», «#WH», «options» и «data» определены в файле rebgui.r, и они добавляют новые функции в язык Rebol. Используя эти слова, как определено в rebgui.r, приведенный выше код отображает данный блок данных [Понедельник 9:00 "Петя" Вторник 9:30 "Вася" Среда 10:00 "Женя"]) в графическом окне программы с изменяемым размером и автоматически сортируется, щелкая по заголовкам столбцов. Такой тип отображения и повндения является обычным требованием в современных программах, отображающих списки данных (приложения «базы данных»), поэтому добавленные команды - долгожданное дополнение к языку Rebol. RebGUI содержит широкий набор дополнительных функций, которые полезны при создании сложных графических интерфейсов. Чтобы использовать их, вы должны сначала понять базовый синтаксис Rebol, а затем научиться использовать языковые расширения RebGUI в этом синтаксисе. Как только вы это сделаете, вы можете просто включить файл «rebgui.r» в свои программы и использовать эти языковые расширения, как если бы они были частью языка. Ключ в том, чтобы понимать, что команды RebGUI созданы из исходных материалов на языке Rebol, и для их использования все, что вам нужно сделать, это импортировать rebgui.r с помощью команды «do».

ПРИМЕЧАНИЕ: rebgui.r упакован с несколькими дополнительными файлами, которые, в свою очередь, импортируются в код rebgui.r. Эти файлы содержат код «нижнего уровня», который фактически определяет новые слова и грамматику, составляющие сам RebGUI. Они содержат код, который должен оставаться вместе с rebgui.r. Так же, как указанная выше программа не будет полной и работоспособной без включенного файла rebgui.r. Когда вы начинаете создавать более длинные и сложные программы, ваш исходный код часто будет состоять из множества отдельных исходных файлов, связанных вместе, чтобы составить целую программу. Управление и запоминание того, какие включенные файлы требуются в ваших программах, становятся все более обязательными, когда вы создаете сложные приложения.

Вот несколько веб-ссылок, содержащих бесплатные модули, которые могут помочь вам выполнять полезные программные задачи в Rebol:

```
browse http://www.hmkdesign.dk/data/projects/list-
view/downloads/release/list-view.r
```

мощный виджет для отображения и управления форматированными данными в приложениях с графическим интерфейсом

```
browse http://www.dobeash.com/rebdb.html
```

модуль базы данных, который позволяет легко хранить и систематизировать большие объемы данных, используя язык баз данных «SQL». Также есть модуль проверки орфографии, который можно включить в ваши программы.

```
browse http://www.rebol.org/cgi-bin/cgiwrap/rebol/view-
script.r?script=rebzip.r
```

модуль для работы с архивами в формате ZIP

```
browse http://www.colellachiara.com/soft/Misc/pdf-maker.r
```

диалект для создания pdf файлов

```
browse http://softinnov.org/rebol/mysql.shtml
```

модуль для прямого управления базами данных mysql

```
browse http://www.rebol.org/cgi-bin/cgiwrap/rebol/view-script.r?script=menu-
system.r
```

диалект для создания разных типов меню

```
browse http://softinnov.org/rebol/uniserve.shtml
```

структура, помогающая создавать сетевые приложения по клиент-серверной технологии

```
browse http://www.rebol.net/demos/BF02D682713522AA/i-rebot.r
do http://www.rebol.net/demos/BF02D682713522AA/i-rebot.r
```

и

```
browse http://www.rebol.net/demos/BF02D682713522AA/objective.r
do http://www.rebol.net/demos/BF02D682713522AA/objective.r
```

и

```
browse http://www.rebol.net/demos/BF02D682713522AA/histogram.r
do http://www.rebol.net/demos/BF02D682713522AA/histogram.r
```

Эти примеры содержат модуль 3D-движка, полностью написанный на Rebol. Модуль позволяет легко добавлять объекты 3D-графики и управлять ими в приложениях Rebol.

```
browse
http://web.archive.org/web/20030411094732/www3.sympatico.ca/gavin.mckenzie/
```

библиотека парсенга/разбора XML

```
browse
https://web.archive.org/web/20100802063323/http://box.lebeda.ws/~hmm/rswf/rsw
f_latest.r
```

диалект для создания flash (SWF) файлов

```
browse http://www.rebol.net/docs/makedoc.html
```

конвертирует текстовые файлы в красиво отформатированные HTML-файлы

```
browse http://www.rebol.org
```

множества дополнительных модулей и полезных фрагментов кода. Первое, на что следует обратить внимание при поиске исходного кода для Rebol.

Куски кода не всегда предназначены для модульного использования, но их можно найти в опубликованных программах с открытым исходным кодом иосновываясь на них создавать свою программу. Поиск полезных участков кода в программах, выпущенных для общего доступа - жизненно важный способ сэкономить время и усилия при программировании, а также улучшить свои возможности как разработчика. Большая часть потенциальной работы в любой мыслимой области программирования уже наверника проделана разработчиками гдето в мире. Всегда проверяйте наличие необходимых функций в связанных программах, написанных другими программистами. Если примеры, которые вы найдете, хорошо прокомментированы и организованы, вы, вероятно, сможете повторно использовать часть кода с минимальными усилиями. (Особенно внимательно следите за полезными определениями функциональных слов).

Список Интернет-ресурсов, содержащих доступный исходный код, включен в конце этого руководства. Программы, доступные на этих сайтах, представляют собой золотую жилу существующего кода, необходимого для продуктивной разработки.

Использование внешних программ как «модулей»:

Помните, что данные часто совместимы между существующими программами. После того, как данные были сохранены на носителе, вы можете передать их другим инструментам и программам. Встроенное в Rebol слово «Call»(вызов) позволяет запускать другие программы на вашем компьютере. Он предоставляет множество вариантов для отправки параметров командной строки и обработки вывода этих программ. Используя «Call»(вызов), вы можете выполнять все встроенные команды «оболочки», включенные в операционную систему вашего компьютера (например, команды DOS и Unix). Вы даже можете встраивать и использовать готовые приложения целиком, чтобы управлять данными в ваших программах Rebol. В приведенном ниже примере открывается Блокнот Windows для редактирования текстового файла webcam.r, созданного ранее:

```
call "notepad.exe webcam.r"
```

Этот пример запустит праграмму Paint из состава Windows и откроет в ней ранее скаченную картинку:

```
call "mspaint.exe bay.jpg"
```

Вот пример, который встраивает исполняемую программу в код, записывает ее на жесткий диск, а затем запускает с функцией вызова:

```
program: load to-binary decompress 64#{
```

```
eJztF11sU2X03K4VqJsrkZJp6OzchhFJsx8qDB9od1fHdIO6ds7AgJX2jttyey/p
vWUjJuNnmNhMibzwaCSLi+EBE1ziGIkBGh0BSYTwwAMme9Dk4kgkgSiKcj3nu7es
QrKFhMUQOcn5+c7fd875+vXe27FJAg4AbIiGAQwWIwZMEbqTcmODN5xRdmRi6aoy
Z83YogngLlaNtV+s6kV7q9KelHeu9LYqQTXt7e/v97UqLcLuqKJIvriShnAIoJ0r
gXvPn+StlDAF5dyzHLwAdlw4TZ1Mm7oQvWDu7jKLslsxBc4KQ30bb9bMHF3F/D5j
MFAHEIbHD+cwb88s9riSEIjvK7EKogZs//bxAvQmYlqM5JsOUwHPWFgEAYDTvqTp
eYdy1Fn5Sh/O96h9nLrrDcD4IpQm7UOkWL/nt6MlqMvxrkl+GVWS7xqWalzDzqGz
9rbyD5ehpmnl+ezt3M/RSPe7Q9/ajeh5+9Ztm3vKh9xoM7SaimLUR18C2JKf+Kg2
APoJwzDOuiAF+hHU/pHXryObdLyP+y2kEhx7UaLfo0gq/RJa60/n88Ndrpz7FmqG
u5bk3L8zwdWXc0+jdOYXkn4lnYfW++/qOPLyDz7BfH3jTXVnplx949inhPvnSgw/
8RSIHM7P8PdSUYtxlxSkONE+o/u7EkNElMbpcuRKUhTjmLH/iHbDQQ7DHqL77zbh
oQxeRa9duBQHkRj+HnIdr7y/e178AvmmnHt5VQAmaNo59/EZ8QSJAY7EURJvMu2x
KipYj2CaEToYve2eYYiwl4rWY6jN8RWF5XtsuWSyhO7aJG8XXQFkNdWYIqIHK8nH
8FOSFJMoteEfZfQEo1SNCPCW2/BTjWK1uXkp9dDDegjrDqpkAUtiJhNp4ma3qUrx
MG6dqkyFMQ2ExQmaxgU2c/07D2ZJsCz3Q68Xh76Cvac2pZwi8jCO8rIZd4jielmc
uHxmsEMe1vMBZJf0YY8Pda95yH5p+tWrI86XMZbTE5a1gVlXFKyryeowp0Cy4Wf+
hdSrWGp26N008hW4XnS6/OBS7MnUVHoK0osoTV+22qF56c95qKdtZBzB66J/imSc
/Rmsg/KDdHFbA9O3RrZWByD/qPf1KTCwze3y2KCbn9vnP4ExoItiwr11zvncqq6+
oXGV//XVa5qCzXxL6M3ZfBfMZyFPBvywgD3FGDjLnGVl83o4T+HJAZ/PFxWTqrcj
GxerHljRqyL9sWXxqU2/nkHki1H4HDkvJeM7vZooeLdnNU2R10K34G1XdgveTmE7
vmv7fNDcFY1u3ABpNa5J6rZd9MouqGpjw6z1GLXn6vDxV/s9o1cYvcroNUanGP2J
UZ3RG4zeZPQ2o3cY/YtRqCdqZ3Qho6WMuhitYHQZ0pr6mRr21Zvv03VFuuMoX0Gd
VqT7BlupKFoXw8eo/8yynUR+HvEa4g3EPxEXYuwSxOWIaxADiGHEBKKGeADxCOIx
a1wXkE81zH/ut0OdG0LtjQ2+hCSBzLUKWoeSyErC+pickIQgfAmhgaSG319xPEvo
ioQ6Ld9D0CL04ddZQuknaxA4W1hRtXeySa0DXWM7BHjDFhHkhLUKYs2cJTcrA0H4
mmtXYgk+m1GVTBBOsVVbXJGDsNTWKexIqpqQ4aWYqgbps4LPCDFNMPcLYXQpldrC
g0bcVHcKcQ220DqyB4PTHYKWScZVgCGsw/LBEgHWsjYLZR2zRTMxWZUwfaFwOAot
SXVXTIuLM9V/ZeuSMw/UxW/s4KOF6W2GNjmp8Uo6rci8ImsZRVLxG+1hZWhgrlv6
/4F/ABcSIgQAEAAA
}
write/binary %program.exe program
call %program.exe
```

Скопируйте и вставьте весь этот блок кода в интерпретатор Rebol и запустите его. Он выполнит небольшую демонстрационную программу, которая была написана и создана языком, совершенно не связанным с Rebol.

Существует целый мир приложений с открытым исходным кодом, доступных для выполнения практически всех возможных задач высокого уровня. Многие из них бесплатны для использования и распространения даже в коммерческих приложениях (см. Http://sourceforge.net). Если вы можете взаимодействовать с ними в командной строке, их можно использовать для помощи в ваших собственных приложениях - даже если вы не знаете, как они были написаны. Это расширяет возможности языка, но также может ограничивать его кроссплатформенность. Например, если вы пишете программу, которая вызывает команды операционной системы DOS, эта программа не может использоваться в системе UNIX.

Всякий раз, когда вы используете исполняемый файл или код, созданный другим программистом, обязательно проверьте и соблюдайте условия лицензирования, по которым он распространяется.

21. Краткое резюме

В приведенном ниже списке приведены некоторые ключевые характеристики языка Rebol. Знание того, как использовать эти элементы, составляет фундаментальное понимание того, как работает Rebol:

1) Для начала, Rebol имеет сотни встроенных функциональных слов, которые выполняют общие задачи. Как и в других языках, за функциональными словами обычно следуют переданные параметры. Для достижения желаемой цели функции располагаются по порядку, одна за другой, чтобы выполнить данную задачу. Выражения оцениваются в порядке слева направо, окончание

строки не требуются, а пустые пробелы (пробелы, табуляции, новые строки и т.д.) могут быть вставлены по желанию, чтобы сделать код более читабельным. Текст после точки с запятой и до новой строки рассматривается как комментарий. Простое знание предопределенных функций на языке и возможность упорядочить их в удобном порядке позволит вам легко работать с Rebol.

2) Многие распространенные типы данных понимаются и обрабатываются интерпретатором Rebol. Числа, текстовые строки, денежные суммы, URL-адреса, двоичные представления изображений, звуков и т.д. распознаются автоматически. Доступ к сетевым ресурсам и интернет-протоколам (http-документы, ftp-каталоги, учетные записи электронной почты, DNS-сервисы и т.д.) можно получить напрямую. Данные любого типа можно записывать и читать практически с любого подключенного устройства или ресурса. Помните, что символ процента («%») используется для обозначения значений локального файла. Поскольку многие типы значений встроены в Rebol, просмотр данных и принятие условных решений на основе пользовательского ввода и содержимого данных является простым и гибким.

3) И код действия, и данные хранятся в «блоках», которые разграничиваются начальными и конечными "квадратными" скобками («[]»). Данные и код, содержащиеся в блоках, разделяются пробелом. Функции, содержащиеся в блоках, могут быть оценены (их действия выполнены) с помощью слова «do». Новые служебные слова также могут быть определены с помощью слов «does»(делать) и «func»(функция). Слово "func" позволяет вам передавать ваши собственные заданные параметры вновь определенным функциональным словам. Вы можете «do» (сделать) модуль кода, содержащийся в текстовом файле, если он содержит минимальный заголовок «Rebol []». Данные, содержащиеся в блоках, могут быть любого типа, и блоки могут автоматически обрабатываться как списки данных, называемые «сериями». Сериями можно управлять с помощью встроенных функций, которые позволяют искать, сортировать, упорядочивать и иным образом организовывать заблокированные данные.

4) Синтаксис «view layout [блок]» используется для создания основных макетов графического интерфейса (окна программы). Добавьте слова в блок для создания графических виджетов: «button»(кнопка), «field»(поле), «text-list»(текстовый список) и т.д. добавьте цвет, положение, интервал и другие параметры/слова после каждого слова виджета. Добавляйте блоки действий после каждого виджета для выполнения функций, когда виджет активирован (то есть, когда виджет щелкают мышью, нажимают клавишу ввода и т.д.). Обратитесь к элементам в макете графического интерфейса пользователя, используя уточнения пути (то есть «face/offset»(грань/смещение) относится к положению выбранного виджета). Эти простые рекомендации можно использовать для создания полезных графических интерфейсов пользователя и облехчение ввода и вывода данных.

5) Любым данным или коду можно присвоить название/словесную метку. Метки слов присваиваются значениям, переменным, функциям, выражениям и блокам любого типа с помощью символа двоеточия (":"). После назначения переменные слова могут использоваться для представления всех функций и данных, содержащихся в данном выражении, блоке и т.д. просто поставьте двоеточие в конце слова, и после этого оно будет представлять все следующие действия и/или данные. Это составляет большую часть языковой структуры Rebol и является основой его гибких способностей к диалектированию естественного языка.

22. 8 Завершённых программы для изучения

В следующих программах используется ряд методов, описанных в руководстве. Они демонстрируют, как части кода могут быть собраны вместе для создания полноценных приложений, и дают некоторое видиние относительно того, как базовые "строительные блоки" языка могут быть собраны для достижения поставленых целей. В комментариях есть подробные построчные объяснения каждой программы (удивительно, насколько короткие эти программы без комментариев!). Изучение этих примеров, пожалуй, самая ценная часть урока.

Загружаемый пакет всех демонстрационных программ(на английском) доступен по адресу:

```
browse http://musiclessonz.com/rebol_tutorial_examples.zip
```

ZIP-файл содержит снимки экрана, отдельный исходный код и упакованные исполняемые файлы (файлы с расширением .exe) каждого примера. Также включены XML-файлы XpackerX, используемые для создания каждого исполняемого файла.

ПРИМЕЧАНИЕ. Чтобы уместить примеры кода по ширине веб-страницы, были введены определенные параметры форматирования. Длинные строки кода были сокращены, поэтому они не обрезаются при печати. Встроенное слово «trim»(обрезка) использовалось для очистки длинных строк текста, написанных на отдельных строках.

```
mystring: trim {
    Это строка}
```

тоже самое, что:

```
mystring: "Это строка"
```

«Join»(присоединение) также использовалось для объединения длинных частей текста:

```
join "Это " [
    "какой то "
    "текст "
    "на несколько строк."
]
```

тоже самое, что:

```
"Это какой то текст на несколько строк."
```

Блоки также были записаны в несколько строк, где это необходимо:

```
myblock: [Это
блок
]
```

тоже самое, что:

```
myblock: [Это блок]
```

Чтобы запустить примеры программ в этом разделе, вы можете использовать любой из методов, описанных ранее: кликнув по серому фону, введите или скопируйте/вставьте код в интерпретатор Rebol, сохраните в виде текстового файла и запустите ее, используя "do {filename}", упакуйте его с помощью XpackerX и т. д.

23. Маленький почтовый клиент

Первый пример - это законченный графический почтовый клиент, который можно использовать для чтения и отправки E-mail сообщений по электронной почте. Код сильно комментирован, чтобы как можно подробней объяснить как работает каждый элемент:

```
Rebol [Title: "Маленький E-mail клиент"]   ; (обязательный заголовок)

view layout [
        ; Первая строка создаёт графическое окно программы
    h1 "Отправить E-mail:"
        ; выводим заголовок(текстовую метку) в окне
    address: field "recipient@website.com"
        ; создаётся текстовое поле с текстом внутри
        ; "recipient@website.com", в котором предлагается
        ; ввести адрес получателя письма, который будет сохранён в
        ; переменной "address"
    subject: field "Тема"
        ; Ещё одно текстовое поле для ввода темы сообщения
        ; которая сохранится в переменной "subject"
    body: area "Текст"
        ; создаётся большое, многострочное поле для ввода
        ; текста сообщения, который будет сохранён
        ; в переменной "body"
    btn "Отправить" [
            ; Кнопка со соловом "Отправить". Команды в нутри этого
            ; блока будут выполняться при нажатии/клики на эту кнопку
        send/subject to-email address/text body/text subject/text
            ; Эта строка выполняет большую часть работы. Она использует
            ; встроенное слово Rebol "send"(отправить) для отправки
            ; электронного письма. Функция send с уточнением "/subject"
            ; принимает три параметра. Текст, содержащийся в каждом поле,
            ; отмеченном выше; (записанно как "address/text", "body/text"
            ; и "subject/text"). Встроенная функция "to-email"(на
            ; электронную почту) гарантирует, что текст адреса
            ; рассматривается как адресс электронной почты.
        alert "Письмо отправленно."
            ; окно предупреждения(alert) с сообщением, что письмо
            ; отправленно
    ]
    h1 "Прочитать E-mail:"
        ; Другая текстовая метка
    mailbox: field "pop://user:pass@website.com"
        ; Другое текстовое поле где пользователь вводит данные для
        ; доступа к ящику, чтобы прочитать письма
    btn "Читать почту" [
            ; Дополнительная кнопка, на этот раз с действием
            ; блок, который читает сообщения из указанного почтового ящика.
            ; Достаточно одной строчки:
        editor read to-url mailbox/text
            ; Встроенная функция "to-url" гарантирует, что
            ; текст в поле почтового ящика рассматривается как URL-адрес.
            ; Содержимое почтового ящика читается и отображается
            ; во встроенном редакторе Rebol.
    ]
]
```

Эта же программа, но без коментариев - намного меньше по размеру:

```
Rebol [Title: "Маленький E-mail клиент"]

view layout [
    h1 "Отправить E-mail:"
    address: field "recipient@website.com"
    subject: field "Тема"
    body: area "Текст"
    btn "Отправит" [
        send/subject to-email address/text body/text subject/text
        alert "Письмо отправленно."
    ]
    h1 "Прочитать E-mail:"
    mailbox: field "pop://user:pass@website.com"
    btn "Читать почту" [
        editor read to-url mailbox/text
    ]
]
```

24. FTP Чат

Второй пример - это простое чат приложение, которое позволяет пользователям отправлять мгновенные текстовые сообщения туда и обратно через Интернет. Он включает защищенный паролем доступ для администраторов для удаления содержимого чата. Это также позволяет пользователям приостанавливать действие на мгновение и требует имени пользователя/пароля для продолжения ["secret" "password"]. «Комнаты» чата создаются путем динамического создания, чтения, добавления и сохранения текстовых файлов через ftp (для использования программы вам потребуется доступ к ftp-серверу: ftp-адрес, имя пользователя и пароль).

```
Rebol [title: "FTP Чат"]  ; обязательный заголовок "Rebol []"

webserver: to-url request-text/title/default trim {
    Адресс FTP сервера:} {ftp://user:pass@website.com/chat.txt}
    ; получить URL-адрес текстового файла ftp-сервера
    ; для использования в чате. Имя пользователя ftp, пароль,
    ; домен и имя файла должны вводится в указанном формате.
name: request-text/title "Введите своё имя:"
    ; запрашиваем имя пользователя

cls: does [prin "^(1B)[J"]
    ; команда "cls" теперь очищает экран (уже было в учебнике).

write/append webserver join now [
    ": " name " вошол в комнату." newline
]
    ; Строка выше записывает текст на ftp-сервер.
    ; параметр "/append"(добавить) добавляет запись к существующим
    ; в текстовом файле на сервере (т.е. без удаления того,
    ; что там уже есть). Используя "join", текст
    ; записанный на веб-сервере - это совокупное значение
    ; {имя пользователя}, статический текст, текущий
    ; дата и время, а также ентер/возврат каретки/переход на новую строку.
forever [
    current-chat: read webserver
    ; читаем сообщения, которые в настоящее время находятся на
    ; сервере, и кладём их в переменную "current-chat" (текущий-чат)

    cls ; очистите экран, используя команду, определенную ранее
    print join "-------------------------------------------------" [
        newline {Вы вошли как: } name newline
```

```
        {Наберите "room" для переключения комнаты.} newline
        {Наберите "lock" для остановки/блокировки чата.} newline
        {Наберите "quit" для завершения чата.} newline
        {Наберите "clear" для удаления текущего чата.} newline
        {Нажмите [ENTER] для переодического обновления экрана.} newline
        "-------------------------------------------------" newline]
    ; отображает приветствие и некоторые инструкции

    print join "Вот текущий текст чата на: " [webserver newline]
    print current-chat

    sent-message: copy join name [
        " сказал: " entered-text: ask "Вы сказали:  "
    ]
    ; получите текст для отправки, затем проверьте команды ниже
    ; («quit»(выйти), «clear»(очистить), «room»(комната),
    ; «lock»(заблокировать) и [ENTER])
    ; Встроенное слово "ask"(спросить) запрашивает у пользователь
    ; чтонибудь в консоле интерпритатора.
switch/default entered-text [
    "quit"  [break]
            ; Если пользователь ввёл "quit"(выход),
            ; то прерываем цикл (break) и соответственно
            ; выходим из программы
    "clear" [
        if/else request-pass = ["secret" "password"] [
            write webserver ""] [alert trim {
            Вы должны знать и ввести логин и пароль
            администратора, чтобы очистить чат в этой комнате!}
        ]
    ]
            ; если пользователь ввёл команду "clear", то
            ; запрашиваем логин/пароль и если они ведены
            ; правельно, то очищаем чат
    "room"  [
        write/append webserver join now [
            ": " name " покинул комнату." newline]
        webserver: to-url request-text/title/default {New Web
        Server Address:} to-string webserver
        write/append webserver join now [
            ": " name " вошел в комнату." newline
        ]
    ]
            ; если пользователь ввел "room", то программа
            ; добавит в чат сообщение, о том, что пользователь починул
            ; чат и запросит новый адрес сервера и запустите код,
            ; который был представленный ранее в программе и используя
            ; только что введенную переменную "webserver",
            ; эффективно менять чат "комнаты"(room).
    "lock" [
        alert trim {Программа заблокируется через 5 секунд.
            Вам понадобятся логин и пароль, чтобы
продолжить.}
        pause-time: now/time + 5
        ; назначить переменную pause-time в которую записываем
        ; время на пять секунд больше текщего
        forever [if now/time = pause-time [
            ; wait 5 seconds
            while [request-pass <> ["secret" "password"]] [
                alert "Неверный пароль - смотрите в исходнике!"
                ]
            ; непродолжаем, пока пользователь не введёт правельный
```

```
            ; пароль.
            break
            ]
        ]
        ; выйти из цикла forever(навсегда) через 5 секунд
    ]
] [if entered-text <> "" [
        write/append webserver join sent-message [newline]]]
    ; случай по умолчанию: пока введенное сообщение не
    ; пусто ([Enter]), напишите сообщение на веб-сервер
    ; (добавить к текущему тексту)
]
; при выходе из цикла "forever" сделать следующее:
cls print "Goodbye!"
write/append webserver join now [
": " name " закрыл чат." newline]
wait 1
```

Основная часть программы выполняется в цикле «forever»(навсегда) и использует условный оператор «switch», чтобы решить, как реагировать на ввод пользователя. Это классическая структура, которую можно настроить для реагирования в разнообразных ситуациях, в которых компьютер постоянно ожидает и реагирует на действия пользователя.

P.S.Так, как ftp-сервер есть не у весх, то немного изменив программу можно организовать чат в локальной сети, используя сетевые ресурсы (Зашареные/общие папки (\\SERVER\share\filename.txt)), или даже на одном компьюторе, запустив несколько копий программы.

```
Rebol [title: "'Файловый' чат"]
chat-file: to-file request-text/title/default {Чат файл:} "chat.txt"
name: request-text/title "Введите своё имя:"
cls: does [prin "^(1B)[J"]
write/append chat-file join now [": " name " вошол в комнату." newline]
forever [
    current-chat: read chat-file
    cls
    print join "-------------------------------------------------" [
        newline {Вы вошли как: } name newline
        {Наберите "/чат" для переключения комнаты чата.} newline
        {Наберите "/блок" для остановки/блокировки программы.} newline
        {Наберите "/выход" для завершения чата.} newline
        {Наберите "/удалить" для удаления/очистки текущего чата.} newline
        {Нажмите [ENTER] для получения новых сообщений.} newline
        "-------------------------------------------------" newline]
    print join "Вот текущий текст чата на: " [chat-file newline]
    print current-chat
    sent-message: copy join name [" сказал: " entered-text: ask "Вы сказали:
"]
    switch/default entered-text [
        "/выход"    [break]
        "/удалить" [
            if/else request-pass = ["secret" "password"] [
                write chat-file ""] [alert trim {
                Вы должны знать и ввести логин и пароль
                администратора, чтобы очистить чат в этой комнате!}
            ]
        ]
        "/чат"  [
            write/append chat-file join now [": " name " покинул комнату."
```

```
newline]
            chat-file: to-file request-text/title/default {Чат файл:} to-
string chat-file
            write/append chat-file join now [": " name " вошел в комнату."
newline]
        ]
        "/блокировка" [
            alert trim {Программа заблокируется через 5 секунд.
                Вам понадобятся логин и пароль, чтобы продолжить.}
            pause-time: now/time + 5
            forever [if now/time = pause-time [
                while [request-pass <> ["secret" "password"]] [
                    alert "Неверный пароль - смотрите в исходнике!"
                    ]
                break
                ]
            ]
        ]
    ] [if entered-text <> "" [write/append chat-file join sent-message
[newline]]]]
cls print "Досвидания!"
write/append chat-file join now [": " name " закрыл чат." newline]
wait 1
```

25. Графические эффекты

Следующая программа имеет графический интерфейс, загружает и отображает изображение из Интернета, и позволяет накладывать на изображения различные эффекты и после сохранить его на диске. В смеси есть несколько подпрограмм, которые получают данные и предупреждают пользователя текстовой информацией.

```
REBOL [Title: ""]
; заголовок обязателен, даже если его блок пуст

effect-types: ["Invert" "Grayscale" "Emboss" "Blur" "Sharpen"
                "Flip 1x1" "Rotate 90" "Tint 83" "Contrast 66"
                "Luma 150" "None"]
; создаём список эфектов в переменной "effect-types"

image-url: to-url request-text/title/default {
    Введите адресс (url) изображения:} trim {
    http://rebol.com/view/demos/palms.jpg}
; спашиваем у пользователя адресс изображения (в поле будет адрес
; "по умолчанию"). Записываем его в переменную "new-image"

gui: [
    ; Следующий код отображает меню программы, используя
    ; виджет кнопки "choice"(выбор) (кнопка с выбором из меню
    ; встроен в Rebol). Кнопка 160 точек, расположена горизонтально
    ; и помещается в крайнем верхнем левом углу окна (0x0)
    ; с использованием встроенного слова «at».
    ; Блок действий для кнопки содержит различные функции, которые
    ; будут выполняться, в зависимости от выбранного пункта меню
    ; (с использованием условных оценок "if"(если). Это может было
    ; реализовать с использованием оператора "switch"(переключить)
    ; и при этом потребовалось бы меньше кода, но было сделанно
    ; умышленно, чтобы продеманстрировать, что всегда есть альтернативные
    ; методы реализации, приближенные к разговорному языку. Попробуете
    ; переписать программу заменив "if" на "switch"? :).
```

```
    across
        ; горизонтальное расположение графических элементов интерфейса,
        ; чтобы они отображались рядом друг с другом в окне
        ; программы (по умолчанию в Rebol - выравнивание элементов
        ; вертикально, т.е. распалогает их друг под другом).
    space -1
        ; уменьшаем размер интервала между элементами, чтобы они
        ; были ближе друг к другу
    at 20x2 choice 160 tan trim {
        Сохранить изображение} "Открыть сохранённое" "Загрузить новое" trim
{-------------} "Выход"
        [
            if value = "Сохранить изображение" [
                filename: to-file request-file/title/file/save trim {
                    Сохранить файл как:} "Сохранить" %effectedimage.png
                        ; запрашиваем имя файла для сохранения,
                        ; по умолчанию "effectedimage.png"
                save/png filename to-image picture
                        ; сохраняем изображение на диск
            ]
            if value = "Открыть сохранённое" [
                view-filename: to-file request-file/title/file trim {
                    Просмотр файла:} "Открыть" filename
                view/new center-face layout [image load view-filename]
                ; читаем файл с диска
                ; и отображаем его в новом окне
            ]
            if value = "Загрузить новое" [
                new-image: load to-url request-text/title/default trim {
                    Введите новы адрес(url) изображения:} trim {
                    http://www.rebol.com/view/bay.jpg}
                        ; запрашиваем новый адрес изображения
                        ; и кладём его в переменную "new-image"
                picture/image: new-image
                        ; заменяем старое излбражение новым
                show picture ; обновляем окно программы
            ]
            if value = "-------------" [] ; нечего неделаем
            if value = "Выход" [
                quit ; выход из программы
            ]
        ]

    choice tan "Справка" "О программе"
        [alert "Image Effector - Copyright 2005, Nick Antonaccio"]
            ; Пример вывода небольшой справки

    below
        ; вертикально расположение элементов в окне
        ; противоположность "across" - гаризонтального расположения
    space 5
        ; растояние между элементами сделаем побольше
    pad 2
        ; положили(put) 2 точки перед следующим эмементом в окне
    box 550x1 white
        ; рисует линию шириной 550 точек и высотой в 1 точку
        ; (просто красивый разделитель меню и основной области окна,
        ; в виде белой линии)
    pad 10
        ; добавили ещё пустого места
    vh1 "Кликните по эфекту в списке справа:"
```

```
        ; Большой заголовок сверху
    return
        ; следующая сторка
    across
        ; снова указываем на горизонтальное расположение
        ; графических элементов в окне
    picture: image load image-url
        ; загружаем изображение, введенное в начале программы,
        ; отображаем его и присваеваем ему ярлык

    text-list data effect-types [
        current-effect: to-string value
        picture/effect: to-block form current-effect
        show picture
    ]
        ; Приведенный выше код создает виджет графического интерфейса
        ; "text-list"(текстовый списк) и назначает действия для этого
        ; блока, которое выполняется/запускается каждый раз, когда
        ; пользователь нажимает на список. Блок действий с отступом
        ; и каждое действие помещено в отдельную строку для удобства чтения.
        ; В первой строке переменной «current-effect»(текущий-эффект)
        ; присваевается значение которое выбрал пользователь в списке.
        ; который пользователь выбрал из списка. Вторая строка
        ; применяет этот эффект к изображению (слова «to-block»(для блока)
        ; и «form»(из) - необходимый синтаксис для применения эффектов).
        ; т.е. звучит запись как picture(картинка)/effect(эфект): to-block(
        ; для блока) form(из) current-effect(текущий(выбраный)-эфект)
        ; Третья строка отображает только что обработанное изображение.
        ; "show"(показать) очень важно. Его нужно использовать всякий раз,
        ; когда элемент графического интерфейса обновляется.
]

view/options center-face layout gui [no-title]
; отображаем только что построеный блок(gui)
; "/options"(опции) - дополнительные параметры для отображения окна
; "center-face" - отображает окно по центру экрана
; "[no-title]" - отображает окно без строки заголовка, поэтому его
; неполучится перемещать.
```

Попробуйте добавить в программу возможность загрузки изображений для обработке с диска. Затем применение нескольких эфектов к одному изображению и т.д.

26. Игра "Пятнашка"

Вот простой пример, который реализует класическую игру с скользящими пронумерованными плитками с графическим интерфейсом всего... в 8 строках кода. Доска содержит 15 подвижных "плиток" и одно пустое место. Цель игры - переставить плитки в обратном порядке. Плитку можно перемещать только в том случае, если возле неё пустое место, екда она может переместиться. Просто кликните(нажмите мышкой) по плитке, чтобы она сдвинулась на пустое место.

```
Rebol [Title: "Игра Пятнашка"]

gui: [
    origin 0x0 space 0x0 across
        ; Определяем некоторые основные параметры макета окна.
        ; "origin 0x0" строим макет с верхнего левом угла окна
        ; "space 0x0"(растояние между элементами) означает, что
        ; графические элементы (виджеты) будут расположены вплотную
```

```
        ; и расположатся горизонтально ("across") друг за другом
style tile button 60x60 [
    ; Слово "style"(стиль) позволяет вам переопределить
    ; внешний вид и характеристики и действия любого встроенного
    ; виджета(графического элемента) GUI. В следующем разделе
    ; создается новый вид кнопки, называемый "tile"(плитка),
    ; с блоком действий, который меняет местами текущую позицию
    ; кнопки с соседним пустым пространством. Это действие
    ; выполняется всякий раз, когда нажимается одна из кнопок.

        ; В строке ниже проверяется, есть ли пустое пространство
        ; (empty) вокруг нажатой кнопки. «Offset»(смещение)
        ; уточняет, что нас интересует позиция виджета(графического
        ; элемента). Слово «face»(лицо) используется для обозначения
        ; выбранного в данный момент виджет.
        ; "empty" (пустая) кнопка есть в макете (определяется
        ; ниже, в конце макета графического интерфейса).
        ; Ничего страшного, что пустая кнопка еще не определена,
        ; потому что этот код не выполняется, пока весь макет не
        ; построен и не "виден".

    if not find [0x60 60x0 0x-60 -60x0]
        face/offset - empty/offset [exit]

        ; На человечьем языке эта строчка читается так:
        ; «вычесть позицию пустого(empty) пространство от
        ; позиции "кликнутой" кнопки (позиции записывается как
        ; два числа, объединённых символом икса("x"), читается
        ; как "кордината_по_горизонтали"x"кордината по вертикали").
        ; Если разница кординат не 60 точек/пикселей в одним из
        ; четырёх направлениях, тогда ничего не делайть, а просто
        ; выйти(exit) из блока.» (60 пикселей - это размер
        ; кнопки-плитки "tile", определенный выше.)

    ; Следующие три строки меняют местами нажатую плитку с пустым
    ; местом("кнопкой").

        ; Сначала создаём переменную для хранения текущего
        ; положение нажатой кнопки:
temp: face/offset

    ; Затем заменяем кординаты текущей кнопки кординатоми
    ; пустого места, т.е. перемещаем кнопку на пустое место
face/offset: empty/offset

    ; Наконец в кординаты пустого места записываем сохранённые
    ; ранее кординаты нажатой кнопки:

empty/offset: temp
; всё, блок действий при нажатии на кнопку/плитку закончился
]

; Теперь построчно рисуем наши плитки(tile), в окне программы,
; по три на строку. Каждая кнопка/плитка уже содержит описанный
; выше код поведения при клике/нажатии:
tile "1"   tile "2"   tile "3"   tile "4" return
tile "5"   tile "6"   tile "7"   tile "8" return
tile "9"   tile "10"  tile "11"  tile "12" return
tile "13"  tile "14"  tile "15"
empty: tile 200.200.200 edge [size: 0]
]
```

```
; Отображаем весь блок GUI по центру экрана пользователя:

view center-face layout gui
```

Вот вся программа без комментариев, в более компактном формате и с буквами алфавита вместо цифр. Как быстро раставите буквы по алфавиту, с лева на право, и с верху вниз? :) :

```
Rebol [Title: "Игра Плитка"]
view center-face gui: layout [
    origin 0x0 space 0x0 across
    style p button 60x60 [
        if not find [0x60 60x0 0x-60 -60x0]
            face/offset - empty/offset [exit]
        temp: face/offset face/offset: empty/offset
            empty/offset: temp
    ]
    p "Я" p "Ю" p "Э" p "Щ" p "Ш" return
    p "Ч" p "Ц" p "Х" p "Ф" p "У" return
    p "Т" p "С" p "Р" p "П" p "О" return
    p "Н" p "М" p "Л" p "К" p "Й" return
    p "И" p "З" p "Ж" p "Е" p "Д" return
    p "Г" p "В" p "Б" p "А"
    empty: p 200.200.200 edge [size: 0]
]
```

Обязательно сделайте перевыв в програмировании и поиграйте в собственно ручно написанные игры :)

27. Создатель аппликатуры гитарных аккордов

Следующий пример - это программа, которая позволяет создавать, сохраняет и распечатывать аппликатуры (схемы) гитарных аккордов. Он демонстрирует некоторые более общие и полезные методы работы с файлами, данными и графическим интерфейсом:

```
Rebol [Title: "Создатель аппликатуры гитарных аккордов"]

; Загрузка(load) встроеных графических элементов (картинок):

fretboard: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAFUAAABkCAIAAAB4sesFAAAACXBIWXMAAAsTAAAL
EwEAmpwYAAAA2UlEQVR4nO3YQQqDQBAF0XTIwXtuNjfrLITs0rowGqbqbRWxEEL+
RFU9wJ53v8DN7Gezn81+NvvZXv3liLjmPX6n/4NL//72s9l/QGbWd5m53dbc8/kR
uv5RJ/QvzH42+9nsZ7OfOPzzX4fOfX62nfOPzzX62nfOPzzX4296qfVHo94/rM49y+b/Wz2n9P
Zj+b/WzuX/cvmfuz2fOfzX4296qfVHo94/rM49y+b/Wz2s9nP
/mWzn81+NvvZ7OfOPzXvm
/mWzn81+NvvZ7Gezn8396/4l2/n+y6N/f/vZ7Gezn81+tjenRWXD3TC8nAAAAABJ
RU5ErkJggg==
}

barimage: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAEoAAAAFCAIAAABtvO2fAAAACXBIWXMAAAsTAAAL
EwEAmpwYAAAAHElEQVR4nGNsaGhgGL6AaaAdQFsw6r2hDIa59wCf/AGKgzU3RwAA
AABJRU5ErkJggg==
}

dot: load 64#{
iVBORw0KGgoAAAANSUhEUgAAAAoAAAAKCAIAAAACUFjqAAAACXBIWXMAAAsTAAAL
```

```
EwEAmpwYAAAAFElEQVR4nGNsaGhgwA2Y8MiNYGkA22EBlPG3fjQAAAAASUVORK5C
YII=
}

; Дизайн GUI:

; Приведенная ниже процедура была скопирована из
; http://rebol.com/how-to/feel.html
; (почитайте на досуге документацию - очень познавательно)

movestyle: [
    engage: func [face action event] [
        if action = 'down [
            face/data: event/offset
            remove find face/parent-face/pane face
            append face/parent-face/pane face
        ]
        if find [over away] action [
            face/offset: face/offset + event/offset - face/data
        ]
        show face
    ]
]

; Эта функция добавляет "чувствительность к передвежению" к любому
; виджету/графическому элементу, делаея его подвижным в окне программы.
; Это очень полезно для разных графических приложений ...
; Если вы хотите создавать графические макеты, реагирующие на
; действия/движения пользователя, то изучите этот аспект
; Все о том, как заставить «чувствовать» элементы описанно в
; сылке(URL) выше. Почитайте для получения дополнительной информации.
gui: [
    backdrop white
        ; сделать фон(backdrop) белым(white)
    currentfretboard: image fretboard 255x300
        ; показать изображение грифа и изменить его размер
        ; (сохраненное изображение на самом деле 85x100 пикселей)
    currentbar: image barimage 240x15 feel movestyle
        ; Показать изображение панели, изменить его размер и сделайть
        ; подвижным. Обратите внимание на "feel movestyle"(стиль
        ; движения). Вот что позволяет реализовать перетаскивание.
    text "ИНСТРУКЦИЯ:" underline
    text "Переместите мышкой графические элементы на схему грифа."
    text "Перед печатью аппликатуры необходимо её сохранить."
    across
    text "Изменить размер грифа:"
    tab
        ; "tab" выравнивает следующий элемент графического интерфейса
        ; с заранее заданным расположением колоннок
    rotary "255x300" "170x200" "85x100" [
        currentfretboard/size: to-pair value show currentfretboard
        switch value [
            "255x300" [currentbar/size: 240x15 show currentbar]
            "170x200" [currentbar/size: 160x10 show currentbar]
            "85x100" [currentbar/size: 80x5 show currentbar]
        ]
    ]

        ; "Поворотная" кнопка выше позволяет выбрать размер для
        ; схемы грифа. В блоке действий кнопки изменяется размер
        ; картинки "схема гифа", а затем изменяется и изображение
        ; панели, в соответствии с выбранными размерами. Эти размеры
```

```
        ; выбранны так, чтобы сохранить пропорции изображения при
        ; маштабировании. После каждого изменения размера окно программы
        ; обновляется для того, чтобы эти изменения отобразились.
        ; Встроенное слово "show"(показать) обновляет окно. Это нужно
        ; делать всякий раз, когда виджет меняется. Помните об этом!
        ; Забыть "показыать" измененный элемент графического
        ; интерфейса - частая ошибка.

    return
    button "Сохранить" [
        filename: to-file request-file/save/file "1.png"
        save/png filename to-image currentfretboard
    ]

        ; Блок действий вышеуказанной кнопки запрашивает имя файла
        ; у пользователя, а затем сохраняет текущее изображение
        ; грифа(аппликатуру) в этом файле.

    tab

    button "Напечатать" [
        ; Блок действий этой кнопки создаёт HTML станицу, добавляет
        ; на неё изображения с диска (можно выбрать несколько файлов
        ; зажав клавишу Ctrl), открывает её в браузере и вызывает
        ; диалог печати.
filelist:
    sort request-file/title "Выберете изображения для печати:"
        ; Запросили у пользователя список файлов, отсортировали
        ; его и положили в переменную "filelist"
html: copy {<html><body onLoad="window.print()">}
    ; создаём блок, содержащий код HTML страницы и обзываем
    ; его как "html".
foreach file filelist [
    append html rejoin [
        {<img src="file:///} to-local-file file {">}
    ]
]
    ; Цикл foreach создает макет html, который содержит
    ; имя каждого из выбранных изображений.
append html [</body></html>]
    ; завершаем код html страницы. Теперь переменная "html"
    ; содержит полный html-документ, который будет
    ; записывается на жесткий диск и открывается в браузере
    ; по умолчанию. Код ниже выполняет эти действия.
write %chords.html trim/auto html
browse %chords.html
]
]

; Каждый из следующих циклов помещает 50 подвижных точек в
; графический интерфейс, все в одних и тех же местах.
; Ими обозначается зажатая на ладу струна. Эти команды
; создают три стопки точек, которые пользователь сможет
; перемещать по экрану и накладывать на изображение грифа.
; Есть три размера, чтобы гармонично гармонично смотрелось
; для трёх разных размеров схем грифа. Обратите внимание
; на "movestyle"(стиль движения) в конце каждой строки.
; это то, что даёт возможность элементу стать "перемещаемым".

loop 50 [append gui [at 275x50 image dot 30x30 feel movestyle]]
loop 50 [append gui [at 275x100 image dot 20x20 feel movestyle]]
loop 50 [append gui [at 275x140 image dot 10x10 feel movestyle]]
```

```
; Следующие циклы добавляют к ним дополнительные
; перетаскиваемые виджеты для обозначения задействованных
; или неиспользуемых струн. т.к. струн 6, то
; создаём по шесть элементов

loop 6 [append gui [at 273x165 text "X" bold feel movestyle]]
loop 6 [append gui [at 273x185 text "O" bold feel movestyle]]

view layout gui
```

28. База данных Listview

Этот пример представляет собой простую базу данных с графическим интерфейсом. Она позволяет вам хранить, искать, изменять, систематизировать и отображать данные удобным способом. Подобный тип программ часто встречается когда необходимо систематизировать какую либо информацию. Хотя в этом примере обрабатывается информация о расписании клиентов, демонстрируемые в нем методы могут быть напрямую применены к управлению информацией любого типа - запасами, списками клиентов/поставщиков, ведомостями заработной платы/налогов/отпусков/дежурств, коллекциями mp3, фотоальбомами, таблицами рекордов в играх и т.д.. Программа использует модуль listview, расположенный по адресу

```
browse http://www.hmkdesign.dk/rebol/list-view/list-view.r
```

который сжат и встроен в сценарий. Модуль listview выполняет всю основную работу по отображению, сортировке, фильтрации, изменению и манипулированию данными с помощью знакомого пользовательского интерфейса, который легко программировать. Документация доступна по адресу

```
browse http://www.hmkdesign.dk/rebol/list-view/list-view.html
```

Научиться пользоваться модулем listview абсолютно необходимо, если вы собираетесь заниматься программированием, связанным с обработкой и управлением информацией. Этому очень просто научиться - вы можете освоить основы за один присест и будете использовать их неоднократно.

Приведенная ниже программа создает базу данных по умолчанию, если она еще не существует, и делает резервную копию предыдущего файла с отметкой даты всякий раз, когда данные сохраняются. Есть дополнительная функция, которая блокирует кнопку закрытия графического интерфейса пользователя, чтобы предотвратить случайное завершение работы программы без сохранения данных.

```
Rebol [title: "Database"]

; Функция ниже следит за кнопкой закрытия графического интерфейса,
; чтобы уберег программу от случайного завершения работы. Код был
; взят из примера на http://www.rebolforces.com/view-faq.html

evt-close: func [face event] [
    either event/type = 'close [
        inform layout [
            across
            text "Сохранить изменения перед закрытием программы?"
```

```
            return
            Button "Сохранит" [
                ; при нажатии кнопки сохранения, данные резервной
                ; копии, файл создается автоматически:
                backup-file: to-file rejoin ["backup_" now/date]
                write backup-file read %database.db
                save %database.db theview/data quit
            ]
            Button "Нет" [quit]
            Button "ОТМЕНА" [hide-popup]
        ] none ] [
        event
    ]
]
insert-event-func :evt-close

; Код ниже - это модуль list-view.r в сжатом формате.
; Он распаковывается, а затем импортируется с помощью команды «do».
; Как и любой другой модуль, вам не нужно понимать, как это было
; запрограммирован (несжатый код - это просто родной Rebol).
; Вам просто нужно включить сжатый BLOB-объект и узнать, как
; использовать его ...

do decompress #{
789CCD3D6B73E3C871DFF52BE6369592F67629905A9F73A6B2A7B2EFECD8E557
2AE738A9 ( ... вырезанный текст ... ) 499F3EC92BF39B56E2B38FB875
96D241F19761DCC656986F7E969FFD1F598CCF767B840000
}

; Следующая строка проверяет, существует ли файл базы данных.
; Если нет, он создает его парой пустых блоков:

if not exists? %database.db [write %database.db {[][]}]

; Теперь файл с базой данных считывается в переменную database:

database: load %database.db

; Вот основа программы. Обязательно прочтите документацию, чтобы
; понимать, как работает виджет.

view center-face gui: layout [
    h3 {Чтобы ввести данные, дважды щелкните любую строку.
        Щелкните заголовки столбцов для сортировки:}
    theview: list-view 775x200 with [
        data-columns: [Учиник Учитель День Время Телефон
            Родитель Возраст Класс Перенос Примечание]
        data: copy database
        tri-state-sort: false
        editable?: true
    ]
    across
    button "добавить" [theview/insert-row]
    button "удалить" [
        if (to-string request-list "Вы уверены?"
                [yes no]) = "yes" [
            theview/remove-row
        ]
    ]
    button "поиск" [
        filter-text: request-text/title trim {
            Filter Text (leave blank to refresh all data):}
```

```
        theview/filter-string: filter-text
        theview/update
    ]
    button "Сохранить БД" [
        backup-file: to-file rejoin ["backup_" now/date]
        write backup-file read %database.db
        save %database.db theview/data
    ]
]
```

Вот урезанная версия, которая полностью функциональна (всего 18 строк кода!). Она запускает модуль просмотра списка из внешнего файла на жестком диске. Если модуля нет на жестком диске, он загружается непосредственно с веб-сервера (вместо того, чтобы встраивать его в код, как указано выше). Итак, если модуль просмотра списка не распространяется с этим сценарием, для его запуска требуется доступное подключение к Интернету:

```
Rebol []

if not exists? %list-view.r [write %list-view.r read
    http://www.hmkdesign.dk/data/projects/list-view/downloads/release/list-
view.r
]
do %list-view.r
if not exists? %database2.db [write %database2.db {[][]}]
database: load %database2.db

view center-face gui: layout [
    theview: list-view 775x200 with [
        data-columns: [Фамилия Имя Очество Телефон Адрес
            Должность Зарплата Отпуск Коментарий]
        data: copy database
        tri-state-sort: false
        editable?: true
    ]
    across
    button "добавить" [theview/insert-row]
    button "удалить" [theview/remove-row]
    button "фильтр" [
        filter-text: request-text/title trim {
            Filter Text (leave blank to refresh all data):}
        theview/filter-string: filter-text
        theview/update
    ]
    button "сохранить базу" [save %database.db theview/data]
]
```

В обеих программах, указанных выше, щелчок по заголовку столбца сортирует данные по выбранному столбцу, по возрастанию или по убыванию. Щелчок по ромбику в верхнем правом верхнем углу таблицы возвращает данные в их несортированный вид. Выбор строки данных с помощью мыши позволяет редактировать каждую ячейку прямо в списке. Поскольку возможно встроенное редактирование, дополнительные виджеты графического интерфейса пользователя не требуются для ввода/вывода данных. Например, поля ввода текста не требуются, потому что все данные вводятся и отображаются непосредственно в таблице. Это очень мощный и удобный инструмент, который полезен в самых разных ситуациях.

Все функции отображения и управления данными в приведенном выше примере активированы импортированным модулем представления списка. Это делает программирование баз данных в Rebol очень простым. Для получения дополнительной информации о работе со списками данных

с помощью необработанных инструментов Rebol см. Страницы ниже:

```
browse
https://web.archive.org/web/20040111082332/http://compkarori.com/vanilla/dis-
play/VID+list+style
```

Кроме того, не забудьте просмотреть приведенные ниже ссылки с примерами того, как организовать данные и управлять ими с помощью встроенных функций Rebol и некоторых полезных сторонних модулей:

```
browse http://www.rebol.net/cookbook/recipes/0012.html

browse http://www.dobeash.com/rebdb.html

browse http://softinnov.org/rebol/mysql.shtml
```

На странице rebDB есть больше ссылок на инструменты базы данных. Понимание и умение использовать такие инструменты баз данных - важная часть создания современных приложений всех типов, обрабатывающих большие объёмы данных.

29. Одноранговый мессенджер

Следующий пример позволяет двум пользователям подключаться напрямую через сетевой порт TCP/IP (пользователь выберает IP-адрес и номер порта) для обмена сообщениями. В отличие от FTP-чата, приведенного выше, в этом примере не требуется сторонний сервер для хранения или передачи данных. Текст пересылается напрямую между двумя компьютерами через установленное сетевое соединение с сокетом (в Интернете или в локальной сети). Приложение может действовать как клиент или как сервер, в зависимости от выбора пользователя. Для правильной работы через интернет серверный компьютер должен иметь открытый IP-адрес или открытый порт маршрутизатора/брандмауэра. Клиентский компьютер может быть расположен за маршрутизатором или брандмауэром без перенаправления входящих портов. Работу программы можно продемонстрировать на одном компьютере. Инструкции см. В справочной документации, включенной в код.

Базовый код в этом примере основан на примере книги рецептов Rebol по адресу

```
browse http://www.rebol.net/cookbook/recipes/0028.html
```

Посмотритеуказанную ниже страницу для подробного объяснения того, как Rebol открывает, подключается и отправляет данные через порты TCP/IP. Для получения информации о передаче двоичных файлов и данных непосредственно через одноранговое соединение:

```
browse http://www.rebol.net/cookbook/recipes/0058.html
```

Приведенные здесь примеры предоставляют простую модель для добавления возможностей совместного использования данных в ваших сетевыых приложениях. Методы, продемонстрированные в сетевых примерах Rebol, формируют основу для создания неродных протоколов и могут использоваться для отправки данных через последовательные порты, для управления внешними аппаратными устройствами и т.д.

Для получения базовой информации о том, как настроить маршрутизаторы и сетевые порты для

использования в этом примере, см.

```
browse http://portforward.com
```

Этот тип конфигурации выходит за рамки данного руководства, но он необходим для любого типа сетевого приложения одноранговой сети(точка-точка/peer-to-peer/p2p) - совместного использования файлов, многопользовательских онлайн-игр, совместного использования веб-камеры, обмена мгновенными сообщениями, веб-обслуживания и т.д.. Для более фундаментального понимания того, как работают сети, а также для информации о том, как настроить типичную сетевую настройку в MS Windows, см.

```
browse http://com-pute.com/FreeTutorials/Other/NetworkBasics.html
```

.

```
REBOL [Title: "Одноранговый мессенджер"]

connected: false
    ; Это «флаговая» переменная, используемая для обозначения того,
    ; что две машины уже подключены. Это помогает более изящно
    ;обрабатывать действия по подключению и отключению в прграмме.

; Приведенный ниже код перехватывает кнопку закрытия (просто вариант
; подпрограмма, использованная в предыдущем примере базы данных). Это
; гарантирует, что все открытые порты закрыты, и отправляет сообщение
; удаленному компьютеру, что соединение было прервано.
; Обратите внимание, что строки в сообщении об отключении отправляются
; в обратном порядке. Когда их принимает другая машина,
; они распечатываются по одному, каждая строка над пердыдущей - чтобы
; текст выглядел правильно при просмотре на другом компьютере.

insert-event-func closedown: func [face event] [
    either event/type = 'close [
        if connected [
            insert port trim {
                ************************************************
                И УСТАНОВИТЬ СВЯЗЬ ЗАНОГО.
                ВЫ ДОЛЖНЫ ПЕРЕЗАПУСТИТЬ ПРИЛОЖЕНИЕ
                ЧТОБЫ ПРОДОЛЖИТЬ ДРУГОЙ ЧАТ,
                УДАЛЕННАЯ СТОРОНА ОТКЛЮЧЕНА.
                ************************************************
            }
            close port
            if mode/text = "Режим сервера" [close listen]
        ]
        quit
    ] [event]
]

view/new center-face gui: layout [
    across
    at 5x2  ; этот код позиционирует следующие элементы в окне

; Текст ниже отображается как пункт меню в верхнем
; левом уголу графического интерфейса. При нажатии кнопки
; текст, содержащийся в области «display», сохраняется в
```

```
; выбранный пользователем файл.

    text bold "Сохранить чат" [
        filename: to-file request-file/title/file/save trim {
            Сохранит файл как:} "Сохранить" %chat.txt
        write filename display/text
    ]

    ; Текст ниже - это еще один пункрт меню. Он отображает
    ; IP-адрес пользователя при нажатии. Он опирается на
    ; общедоступный веб-сервер для поиска внешнего адреса
    ; (whatismyip.com). Команда "parse"(синтаксический анализ)
    ; используется для извлеките IP-адрес со страницы. Разбор
    ; принцепа её работы рассматриваются в отдельном разделе далее в
    ; учебнике.

    text bold "Lookup IP" [
        parse read https://whatismyip.com [
            thru <title> copy my-ip to </title> (
                alert to-string rejoin ["внешний: "
                    my-ip " внутрений: " read dns://]
            )
        ]
    ]

    ; Текст ниже - это третий пункт меню. Он отображает
    ; текст справки при нажатии.

    text bold "Помощь" [
        alert {
            Введите IP-адрес и номер порта в соответствующие поля.
            Если вы будете ожидать соединения, то с помощью поворотной
            кнопки выберите «Режим сервера» (для связи через Интернет
            вы должен иметь открытый IP-адрес и/или открытый порт
            способный принять входное соединение). Выберите "Режим
            клиента", если вы подключитесь к чужому чат-серверу (вы
            можете сделать это даже если вы находитесь за
            ненастроенным брандмауэром, роутер и т.д.). Нажмите
            «Подключиться», чтобы утановить соединение и начать чат.
            Чтобы протестировать приложение на одной машине, откройте
            два экземпляра чат программы, оставьте IP установленным на
            "localhost" на обоих. Установите один экземпляр как сервер,
            а другой - как клиент, затем щелкните «Подключиться».
            Вы можете редактировать текст чата прямо на области
            сообщений, и вы можете сохранить текст в локальный файл.
        }
    ]
    return

    ; Ниже представлены виджеты, используемые для ввода информации о
    ; подключении. Обратите внимание на ярлыки, присвоенные каждому
    ; элементу. Позже текст, содержащийся в этих виджетах, называется
    ; <метка>/text(ткст). Так же обратите внимание на блок действий
    ; для поворотной кнопки тоже. Каждый раз, когда вы её нажимаете,
    ; то либо скрываются, либо показываются другие виджеты. Когда в
    ; режим сервера, IP-адрес подключения не требуется -
    ; приложение просто ждет соединения на заданном порту. Скрытие
    ; поля IP-адреса избавляет пользователя от некоторой путаницы.
lab1: h3 "IP адрес:"  IP: field "localhost" 102
lab2: h3 "Порт:" portspec: field "9083" 50
mode: rotary 120 "Режим клиента" "Режим сервера" [
    either value = "Режим клиента" [
```

```
        show lab1 show IP
    ][
        hide lab1 hide IP
    ]
]

; Ниже находится кнопка подключения и большой блок действий.
; это делает большую часть работы. Когда кнопка нажата,
; то она скрывается, чтобы у пользователя не возникало соблазна
; снова нажать её и тем самым открыть порт (это вызовет ошибку).
; Потом, открывается TCP/IP порт - тип (сервер/клиент)
; определяется с использованием конструкции "either"(либо). Если
; ошибка происходит в любой из операций открытия порта,
; ошибка перехватывается и пользователь получаетс соответствующее
; продупреждение - это более изящно и информативно, чем позволить
; программе завершиться с ошибкой. Обратите внимание, что
; информация об IP адресе и порте берется из полей выше.
; Если выбран режим сервера (т.е. если кнопка «режим»
; выше отображает текст «Режим сервера»), то TCP-порты открыты в
; режиме прослушивания и ожидает подключения клиента. Если выбран
; клиентский режим, сделана попытка открыть прямое соединение с
; IP адресом и выбранным портом.
cnnct: button red "Соединиться" [
    hide cnnct
    either mode/text = "Режим клиента" [
        if error? try [
            port: open/direct/lines/no-wait to-url rejoin [
                "tcp://" IP/text ":" portspec/text]
        ][alert "Сервер неотвечает." return]
    ][
        if error? try [
            listen: open/direct/lines/no-wait to-url rejoin [
                "tcp://:" portspec/text]
            wait listen
            port: first listen
        ][alert "Сервер уже запущен." return]
    ]

    ; После открытия портов поле ввода текста подсвечивается, и
    ; для флага подключения("connected") установлено значение
    ; true(истина). Перевод фокуса в поле ввода текса сообщения
    ; хороший, но необязательный сигнализатор установки связи.

    focus entry
    connected: true

    ; Цикл "forever"(всегда) ниже постоянно ожидает данные в
    ; открытом сетевом соединении. Когда данные отправляются
    ; (вставляются) с другой стороны (другого ПК), они
    ; копируются и добавляются к текущему тексту в области
    ; отображения сообщений чата, и затем область отображения
    ; обновляется(show), чтобы показать новый текст.

    forever [
        wait port
        foreach msg any [copy port []] [
            display/text: rejoin [
                ">>>  "msg newline display/text]
        ]
        show display
    ]
]
```

```
; Ниже находятся область отображения сообщений и поля ввода текста.
; Уведомление ярлыки, присвоенные каждому. "return" просто положил
; каждый виджет на новой строке в графическом интерфейсе (потому
; что режим макета; установлен на "across"(горизонтарльно) выше.

return  display: area "" 537x500
return  entry: field 428  ; the numbers are pixel sizes

; Кнопка отправки ниже выполняет еще одну важную работу.
; Сначала она проверяет, установлено ли соединение.
; (используя флаг, установленный выше). Если да, он вставляет
; текст из поля ввода(entry) в открытый TCP/IP порт, который будет
; захвачен удаленной машиной - если соединение установлено,
; программа на другом конце ожидает чтения любых данных,
; вставленных в этот порт. После отправки данных через сетевое
; соединение, текст добавляется к локальной области текущего
; отображения сообщений в чате, и элемент в окне обновляется:
button "Отправить" [
    if connected [
        insert port entry/text focus entry
        display/text: rejoin [
            "<<<  " entry/text newline display/text]
        show display
    ]
]
]

show gui do-events  ;необходимо, т.к. "/new(новый)" объявлен выше.;
```

Чтобы использовать этот пример, пользователь-клиент должен знать IP-адрес серверной машины. На практике с этим можно справиться, просто отправив электронное письмо с текущим IP-адресом сервера, но это довольно громоздкое решение. В качестве упражнения попробуйте написать небольшое дополнение к коду, которое автоматически отправляет текущий IP-адрес сервера на постоянно доступный ftp-сервер при каждом запуске программы (функция получения текущих IP-адресов уже включена). Это должно занять всего одну строчку кода. Напишите еще одну строку для автоматической загрузки и вставки IP-адреса в текстовое поле при запуске в клиентском режиме, и вы получите полностью автоматизированный механизм подключения. Если вы хотите пофантазировать, вы можете настроить схему идентификаторов, привязанную к IP-адресам отдельных пользователей - все они обновляются автоматически и выбираются во время выполнения. Вы можете добавить функции для отслеживания того, кто находится в сети / офлайн в любой момент времени и т.д.

30. 3D Example

Этот последний пример представляет собой простую демонстрацию того, как использовать модуль Эндрю Хоадли r3D для добавления 3D-визуализаций в ваши программы. Более подробные и интересные примеры кода см. В примерах на

http://www.rebol.net/demos/download.html

.

```
REBOL [
    Title: "r3D Пример"
    Notes: {Simplified from:
        http://www.rebol.net/demos/BF02D682713522AA/i-rebot.r }
]
```

```
; Это запакованная библиотека r3D:

do to-string decompress 64#{
eJzdPGtT28iWn+Nf0cOXsTMYkGXznL1bBMzEtQSnjPMAipqSpTboRpa8kmwwv37P
Od0tdethO ( ... вырезанный текст ... ) OmG202NByXGg5Zl2jpQstvUJL
D/jpLcdA57DQcljAsoC/XWixM1606vsfGt6vyUFIAAA=
}

; Устанавливаем начальные значения некоторых переменных

base-rot: 150.0 ; значение по умолчанию для степени вращения
                ; основания перевод камеры и значения "Направление
                ; камеры"
cameraTransx: 300.0  cameraTransy: 300.0  cameraTransz: 300.0
cameraLookatx: 0.0  cameraLookaty: 0.0  cameraLookatz: 100.0

    ; Функция обновления ниже строит мир, настраивает камеру и
    ; рендерит мир, заполнив блок RenderTriangles со всеми
    ; необходимыми командами Triangle(Треугольник), pen(Перо) и
    ; fill-pen(Заливка) чтобы нарисовать мир (в обратном Z-порядке).
    ; "cube-model"(Куб-модель) переменная и все функции, такие как
    ; "r3d-scale", "r3d-rotatez", "r3d-translate" определены в сжатом
    ; r3D модуль. Чтобы понять, как пользоваться модулем r3D,
    ; обратите особое внимание новым переменным в этой функции.
    ; Здесь выполняются большая часть работы по созданию уникальных
    ; 3D объекты :
update: does [
    world: copy []  ; re-create the world
    ; МЕСТО База роботов
    base-modelWorld: r3d-scale 100.0 100.0 50.0
    base-object: reduce [ cube-model base-modelWorld navy ]
    append world reduce [ base-object ]
    ; МЕСТО Стенд для роботов
    stand-modelWorld: r3d-compose-m4 reduce [
        r3d-scale 35.0 35.0 150.0
        r3d-rotatez base-rot
        r3d-translate 0.0 0.0 52.0
    ]
    stand-object: reduce [ cube-model stand-modelWorld red ]
    append world reduce [ stand-object ]
        ; ДАЛЕЕ - НАСТРОЙКА КАМЕРЫ, ЧТОБЫ ПОСМОТРЕТЬ МИР
        ; создаются преобразования для камеры
    camera: r3d-position-object
        reduce [ cameratransx cameratransy cameratransz ]
        reduce [ cameraLookatx cameralookaty cameralookatz ]
        [ 0.0 0.0 1.0 ]
    ; Получаем матрицу проекции
    Projection: r3d-perspective 250.0
    RenderTriangles: render world camera Projection 400x360
]

RenderTriangles: copy []

; Ниже собирается графический интерфейс для отображения и управления
; трехмерным объектом. В основном это стандартные текстовые метки и
; несколько виджетов слайдеров (движки-рычажки). которые вызывают
; функцию обновления мира и матрицы проэкции, а затем обновляют
; графический интерфейс, используя "show"(показать). Функция обновления
; выполняет всю реальную работу:
out: layout [
    r3d-viewport: box 400x360 black effect [draw RenderTriangles]
```

```
    across
    style lab label 55 right yellow
    style lab2 label 40 right
    vh2 "Вращение верхнего ящика:"  rbslider: slider 60x16 [
        base-rot: (value * 300.0) update show r3d-viewport ]
    return
    lab "Позиция"
    lab2 "x" cpos_x: slider 60x16 [cameratransx:
        (value * 600 - 300.0) update show r3d-viewport]
    lab2 "y" cpos_y: slider 60x16 [cameratransy:
        (value * 600 - 300.0) update show r3d-viewport]
    lab2 "z" cpos_z: slider 60x16 [cameratransz:
        (value * 600) update show r3d-viewport]
    return
    lab "Направление камеры"
    lab2 "x" clook_x: slider 60x16 [cameraLookatx:
        (value * 400 - 200.0) update show r3d-viewport]
    lab2 "y" clook_y: slider 60x16 [cameraLookaty:
        (value * 400 - 200.0) update show r3d-viewport]
    lab2 "z" clook_z: slider 60x16 [cameraLookatz:
        (value * 200 ) update show r3d-viewport]
]

update
view out
```

31. Меню

Одна странность диалекта GUI Rebol заключается в том, что он не включает в себя собственный способ создания стандартных меню. Виджет кнопки «choice»(выбор) - это близкий аналог, который полезен в коротких приложениях, но он выглядит и работает не так, как ожидают пользователи (приведенная выше программа «Графические эфекты» демонстрирует, как кнопки выбора могут использоваться для обеспечения функциональности меню). Меню важны - они являются фундаментальной частью стандартного дизайна графического интерфейса пользователя, и пользователи склонны искать их интуитивно.

Вот простой прототип, который можно включить в ваши программы для обеспечения дополнительных функций меню:

```
Rebol [Title: "Простой пример меню"]

view center-face gui: layout/size [

    at 100x100 H3 "Вы выбрали:"
    display: field

    ; Вот меню. Убедитесь, что он идет ПОСЛЕ другого кода графического
    ; интерфейса. Если вы поместите его перед другим кодом, при
    ; появлении меню скроет другие виджеты в графическом интерфейсе.
    ; Меню в основном просто виджет текстового списка, который
    ; изначально скрыт за экраном в позиции -200x-200. Когда элемент
    ; в списке меню нажат, запускается блок действий для текстового
    ; списка через структуру условного переключателя, чтобы решить,
    ; что делать для выбранного пункта. Сначала код для каждого
    ; варианта повторно скрывает меню, перемещая его за пределы экрана
    ; (в -200x-200 снова). Для использования в собственных программах
    ; вы можете поместите в список столько элементов, сколько хотите,
    ; и блоки действий для каждого элемента может выполнять любые
```

```
    ; команды и функции, какие захотите.
    ; Здесь каждая опция просто обновляет текст на «display» - поле
    ; ввода текста, созданного выше. Изменить, добавить в или удалите
    ; элементы "Первый пункт", "Второй пункт" и "Выход" в
    ; соответствии с тем, как вам нужно
origin 2x2 space 5x5 across
at -200x-200 file-menu: text-list "Первый пункт" "Второй пункт" "Выход" [
    switch value [
        "Первый пункт" [
            face/offset: -200x-200
            show file-menu
            ; PUT YOUR CODE HERE:
            set-face display "Файл / Первый пункт"
        ]
        "Второй пункт" [
            face/offset: -200x-200
            show file-menu
            ; PUT YOUR CODE HERE:
            set-face display "Файл / Второй пункт"
        ]
        "Выход" [quit]
    ]
]

; Первоначально меню отображается в виде некоторых вариантов
; текста в верхней части окна. При нажатии на меню "Файл" блок
; действий этого текстового виджета перемещает текстовый список
; созданый выше так, чтобы он отображался непосредственно под
; текстом "Файл" ("/offset"(/смещение) - это расположение
; выбранного в данный момент текстового виджета). Он исчезает, когда
; щелкнул снова - код проверяет, есть ли текстовый список
; находится под меню. Если это так, он перемещает
; список вне поля зрения.

at 2x2
text bold "Файл" [
    either (face/offset + 0x22) = file-menu/offset [
        file-menu/offset: -200x-200
        show file-menu
    ][
        file-menu/offset: (face/offset + 0x22)
        show file-menu
    ]
]

; Вот дополнительная опция меню верхнего уровня. Это обеспечивает
; только один выбор. Вместо открытия текстового списка
; виджет с несколькими параметрами, он просто гарантировано,
; закрывает(скрывает) другое меню, а затем запускается какой
; нибудь код.

text bold "Помощь" [
    file-menu/offset: -200x-200
    show file-menu
    ; Поместите свой код сдесь:
    set-face display "Помощь"
]
] 400x300
```

Если вам нужно полноразмерное меню со всеми наворотами, анимированными значками, подходящим внешним видом для различных операционных систем и всеми возможными

вариантами отображения, был создан модуль, который легко предоставляет эти возможности:

```
browse http://www.rebol.org/library/scripts/menu-system.r
```

Вот маленький пример, как его можно использовать:

```
do http://www.rebol.org/library/scripts/menu-system.r
menu-data: [edit: item "Menu" menu [item "Item1" item "Item2"]]
reb-style: [item style action [alert item/body/text]]

view center-face layout/size [
    at 2x2 menu-bar menu menu-data menu-style reb-style
] 400x500
```

А вот более полная демонстрация

```
do http://www.rebol.org/library/scripts/menu-system-demo.r
```

и её код

```
browse http://www.rebol.org/library/scripts/menu-system-demo.r
```

Она демонстрирует расширенные возможности модуля. Ниже приведен небольшой пример с объяснением наиболее важных функций. Модуль меню был сжат и встроен в код:

```
REBOL [Title: "Пример меню"]

; запакованый модуль меню.

do decompress   #{
789CED7DED761B3792E8EFEBA740343F24AD4D5352EC24C3331E1F59A213258E
E548B233191EDE7 ( ... вырезанный текст ... ) F0971E889FF22536399
125D064012EB8107FDFF0F1217899678240100
}

; Ниже приведен пример создания блока меню.
; Обратите внимание, что в следующих меню есть два
; блока. Меню "файл" с отступом и расширением
; через несколько строк. Все меню редактирования открыто
; одна линия. Обратите внимание, что вы можете размещать блоки действий
; после каждого пункта меню, чтобы выполняться всякий раз, когда
; выбран пункт меню - как и в случае с [print "Вы выбрал блок пункт 1"]
; ниже:
menu-data: [
    file: item "Файл"
        menu [
            new:    item "Пункт 1" [print "Вы выбрали пункт 1"]
            open:   item "Item 2" ; icons [1.png 2.png]
            ---
            recent: item "Загляни сюда..."
                menu [
```

```
                    item "Вы ВыЙгРаЛи ПрИз! :)!"
                    item "Попробуйте другую дверь"
                ]
            ---
            exit:   item <Ctrl-Q> "Выход"
          ]
    edit: item "Правка" menu [item "Копировать" item "Вставить"]
]

; Обратите внимание на блок действий в меню определение стиля ниже.
; Вы измените его, чтобы создать свои собственные функции для каждого
; возможного выбора меню.
; Большая часть, определяющая стиль, не является обязательной. Это
; разработан, чтобы выглядеть как родное меню Microsoft. В примере на
; http://www.rebol.org/library/scripts/menu-system-demo.r
; содержит еще много вариантов стилей и опций меню.
; Единственная часть, которая требуется в приведенном ниже примере, это
; блок действий в разделе «item style»(стиль элемента). Все
; остальное служит только для корректировки внешнего вида меню:
winxp-menu: layout-menu/style copy menu-data xp-style: [
    menu style edge [size: 1x1 color: 178.180.191 effect: none]
        color white
        spacing 2x2
        effect none
    item style
        font [name: "Tahoma" size: 11 colors: reduce [
            black black silver silver]]
        colors [none 187.183.199]
        effects none
        edge [size: 1x1 colors: reduce [none 178.180.191]
            effects: []]
        action [

            ; Изменив строки ниже в соответствии со своими
            ; потребностями. Вы можете задать блок действий для
            ; каждого элемента в структуре переключателя(switch),
            ; чтобы запустить свои собственные функции.
            ; "item/body/text"(элемент/тело/текст) относится к
            ; выбранному пункту меню. Это делает то же самое, что и
            ; включение блока кода для каждого элемента в
            ; определении меню выше (т.е. вы можете поместите
            ; блок [quit] после пункта "Выход" выше, и он будет
            ; работать точно так же, как "[print" Вы выбрали пункт 1 "]"
            ; блок после "нового" пункта выше).
switch/default item/body/text [
    "Выход" [quit]
    "Выйграть приз!" [alert "Вы выйграли!"]
    "Попробывать другую дверь" [alert "Плохой выбор :("]
] [print item/body/text]  ; действие по умолчанию
]
]

; Вот простая функция для перехвата события закрытия графического
; интерфейса. Это должно быть включено всякий раз, когда используется
; модуль меню, иначи часть приложения продолжит работу после того, как
; выключится.

evt-close: func [face event] [
either event/type = 'close [quit] [event]
]
insert-event-func :evt-close
```

```
; И вот наконец, пользовательский интерфейс:

window: layout/size [

; В строке ниже показано меню стиля winxp:

at 2x2 app-menu: menu-bar menu menu-data menu-style xp-style

: В слелующей строке меню отобразится при нажатии на кнопку

at 150x200 btn "Кнопка меню" [
show-menu/offset window winxp-menu
0x1 * face/size + face/offset - 1x0
]
] 400x500

view center-face window
```

32. CGI и веб-программирование в Rebol

Чтобы понять программирование CGI, полезно иметь небольшую перспективу. В массовых вычислениях есть три способа, которыми пользователи обычно взаимодействуют с программами для ввода/вывода данных. Вы уже видели примеры двух методов:

1) Оболочка: данные можно вводить/просматривать непосредственно в интерпретаторе Rebol. В коротких сценариях слова «ask»(спросить) и «print»(напечатать) могут использоваться для ввода и отображения данных. В этом формате потоком программы можно управлять, запрашивая у пользователя текстовые параметры и используя условные операции для реакции на ввод. Простые меню можно создать, распечатав списки вариантов выбора, которые должен выбрать пользователь:

```
Rebol []

forever [
    prin {^(1B)[J}
    print "Выберете из предложенных опций:"
    print newline
    print "1 - Вывести одно сообщение"
    print "2 - Вывести два сообщение"
    print "3 - Вывести три сообщение"
    print "4 - Выход"
    print newline
    answer: to-integer ask "Ваш выбор? "
    if answer = 4 [
        print newline
        ask "Досвидание! Нажмите любую клавишу." quit
    ]
    print ""
    loop answer [print "Rebol прекрасен!"]
    print ""
    ask "Нажмите Ентер[ENTER] для продолжения"
]
```

2) GUI: в графических програмах текст вводится и отображается в текстовых полях, областях и списках. Последовательность выполнения программы управляется реагированием на щелчки мыши по графическим кнопкам, спискам меню и другим виджетам:

```
Rebol []

view layout [
    text "Сколько раз вывести сообщение?"
    choice "1" "2" "3" [
        loop to-integer value [alert "Rebol прекрасен!"]
    ]
]
```

3) CGI: этот интерфейс позволяет вводить данные через веб-страницу, используя текстовые поля, области и раскрывающиеся списки в форме html. Данные, введенные в форму, отправляются и обрабатываются программой, которую вы создаете храните и запускаете на своем веб-сервере. Возвращенные данные отображаются в виде форматированного текста, таблиц и других элементов HTML, которые динамически создаются и выводятся вашей программой.

```
There are two parts:

1) Html page:

<HTML><HEAD><TITLE>Data Entry Form</TITLE></HEAD><BODY>
<FORM ACTION="http://yourwebserver/yourrebolscript.cgi">
How many times should the message be displayed? <BR>
<INPUT TYPE="TEXT" NAME="times" SIZE="25">
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">
</FORM>
</BODY></HTML>

2) CGI script:

#!/home/youruserpath/rebol -cs
REBOL []
prin "content-type: text/html"
print newline
print [<HTML><HEAD><TITLE>"Page title"</TITLE></HEAD><BODY>]
times: decode-cgi system/options/cgi/query-string
loop to-integer times/2 [print ["Rebol is great!" <BR>]]
print [</BODY></HTML>]
```

The first two options are typically used to create programs that run on a local computer. They're appropriate for programs that manipulate local user databases, games, audio/video applications, etc. The CGI option is useful when you want to create programs for the Internet.

All three methods above can be used to manipulate data stored online. You can, for example, write a program that uses a local shell or GUI interface to access and manipulate data stored on a web server. Multi-user business applications that share databases among employees at different locations, for example, are well suited to that type of client-server design. Each user gets a copy of the program, and everyone connects to a centralized data repository that exists at one location on a shared network.

To present data on the Internet, however, you don't always want to force users to download and install a local application. Contemporary computer users are familiar with custom database queries, web page searches, specialized email feedback forms, etc. that can be accessed directly on web pages. Rebol's built-in CGI interface enables this possibility, allowing the language to interact directly with information entered via web pages. Rebol's CGI ability allows you to write code that runs on a web server, receives data input from web pages, and returns data to the user's browser. In this way, html pages on your web site can form the complete user interface to programs you write. PHP, PERL and ASP are popular languages for such Internet programming, but if you know Rebol, you don't need to learn them. You can upload the Rebol interpreter to your web server and write cgi applications in the powerful and simple

Rebol language syntax (just choose the Rebol version for the operating system running on your server). With CGI, users can interact entirely via a familiar web page interface, without ever knowing or caring that a program exists behind the scenes. That's one of the most common types of computer application in contemporary use. And Rebol makes it easy to create those types of programs.

The detailed mechanics of CGI programming are slightly beyond the scope of this tutorial, but the following crash course will get you started:

Learning html is required. That's true if you intend to attempt any type of Internet programming. Html is the layout language used to format text and GUI elements on all web pages. Html is not really a programming language - it doesn't let you manipulate data. It's just a simple markup format that allows you to shape the visual appearance of pages viewed in a browser.

In html, items on a web page are enclosed between starting and ending "tags":

```
<STARTING TAG>Some item to be included on a web page</ENDING TAG>
```

There are tags to effect the layout in every possible way. To bold some text, for example, do the following:

```
<STRONG>some bolded text</STRONG>
```

To create a table with three rows of data, do the following:

```
<TABLE border=1>
<TR><TD>First Row</TD></TR>
<TR><TD>Second Row</TD></TR>
<TR><TD>Third Row</TD></TR>
</TABLE>
```

Notice that every

```
<opening tag>
```

in the code above is followed by a corresponding

```
</closing tag>
```

Some tags surround all of the page, some tags surround portions of the page, and they're often nested inside one another to create more complex designs.

A minimal format to create a web page is shown below. Notice that the title is nested between head tags, and the entire document is nested within html tags. The page content seen by the user is surrounded by body tags:

```
<HTML><HEAD><TITLE>Page title</TITLE></HEAD><BODY>
A bunch of text and html formatting goes here...
```

```
</BODY></HTML>
```

If you save the above code to a text file called "yourpage.html", upload it to a web server, and surf to http://yourwebserver.com/yourpage.html , you'll see in your browser a page entitled "Page title", with the text "A bunch of text and html formatting goes here...". All web pages work that way - in fact, this very tutorial is an html document. Click View -> Source in your browser, and you'll see the html tags that are used to format the document.

You can create a CGI program on your web site that outputs the same web page above. Here's the format:

```
#!/home/youruserpath/rebol -cs
REBOL []
prin "content-type: text/html"
print newline
print [<HTML><HEAD><TITLE>"Page title"</TITLE></HEAD><BODY>]
print ["A bunch of text and html formatting goes here..."]
print [</BODY></HTML>]
```

If you save the code above to a text file called "yourpage.cgi", upload it to a web server where you have Rebol installed in {youruserpath}, and surf to http://yourwebserver.com/yourpage.cgi , you'll see in your browser the exact same web page as in the previous example (to make it work, there will likely be some additional setup steps required to install Rebol - that's covered below). The first four lines in the code above are required to make your CGI scripts work properly. Memorize them by rote. The additional lines simply print out the html contained in the previous example. Simple, right?

That may seem like a lot of work to produce the same result as the previous example, but it enables the real power of CGI, which is to include dynamic, changeable information on your page. Say, for example, that you want to display the current time and date in the body of your web page. Html doesn't allow you do that, but the CGI program below does:

```
#!/home/youruserpath/rebol -cs
REBOL []
prin "content-type: text/html"
print newline
print [<HTML><HEAD><TITLE>"Page title"</TITLE></HEAD><BODY>]
print ["The current date and time is: " now]
print [</BODY></HTML>]
```

The fifth line above contains the built in Rebol word "now", which is evaluated and printed on the web page. In the same way, you can read and print data that's been saved to the hard drive on your web server, you can perform calculations and other data manipulations such as conditional evaluations and loops, you can access databases, send emails, create and manipulate images and sounds, and do anything else that the programming language is capable of. Output data can be formatted as a nice web page that your user sees online.

The example below is a short program that reads and sorts the contents of a text file called "users.txt" and displays it on your web page:

```
#!/home/youruserpath/rebol -cs
REBOL []
prin "content-type: text/html"
print newline
```

```
print [<HTML><HEAD><TITLE>"Page title"</TITLE></HEAD><BODY>]
if exists? %users.txt [data: sort read/lines %users.txt print data]
print [</BODY></HTML>]
```

If you want user data to be input into your CGI program, you need to create an html page with a "form". Html forms include text entry fields, dropdown selection boxes and other widgets that allow for data entry. The form should contain a single "action" that links to the web address of your CGI program. The form template below contains a text entry field and an action that points to http://yourwebserver/yourrebolscript.cgi .

```
<HTML><HEAD><TITLE>Data Entry Form</TITLE></HEAD><BODY>
<FORM ACTION="http://yourwebserver/yourrebolscript.cgi">
<INPUT TYPE="TEXT" NAME="username" SIZE="25">
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">
</FORM>
</BODY></HTML>
```

To do something useful with the data sent by the above html form, you'll need to write a CGI program. That program will be stored at http://yourwebserver/yourrebolscript.cgi . To extract the information submitted by the form, include the following line in your Rebol CGI program:

```
a-word: decode-cgi system/options/cgi/query-string
```

You can use the assigned word above to refer to all the data sent when the html form was submitted. Rebol automatically assigns words to the data input in the html form. If you type "Fred Thompson" as the name in the text entry field on the form above, Rebol will decode it as:

```
[username: "Fred Thompson"]
```

You can use that data in a CGI program as follows:

```
#!/home/youruserpath/rebol -cs
REBOL []
prin "content-type: text/html"
print newline
print [<HTML><HEAD><TITLE>"Page title"</TITLE></HEAD><BODY>]
username: decode-cgi system/options/cgi/query-string
print ["Hello " second username "!"]
print [</BODY></HTML>]
```

Just save that CGI code as http://yourwebserver/yourrebolscript.cgi , as referenced in the form action in the html page above. When you enter "Fred Thompson" in the html page, it sends the info to the CGI script, which displays a dynamically created web page saying "Hello Fred Thompson!".

Here's a cgi application that prints output which includes an html form layout. The action of that form points to the address of the CGI program itself (i.e., no additional html page is required to make this script work - it prints one out for its own use). The program outputs a simple page that displays the names of all its former visitors, lets you enter your name, and then calls itself when you click submit, which starts the process over again:

```
#!/home/youruserpath/rebol -cs
REBOL []
prin "content-type: text/html"
print newline
print [<HTML><HEAD><TITLE>"Page title"</TITLE></HEAD><BODY>]
print ["Here's a list of visitors who've signed this page:" ]
print [<BR>] ; prints a carriage return
either exists? %users.txt [
    newname: decode-cgi system/options/cgi/query-string
    if newname/2 <> none [
        write/append %users.txt join "<BR>" [newname/2]
    ]
    signatures: read %users.txt
    print signatures
] [write %users.txt ""]
print [<FORM ACTION="http://yourwebserver/yourrebolscript.cgi">]
print [<BR><HR><BR>"Please enter your name:"<BR>]
print ["Name: "<INPUT TYPE="TEXT" NAME="username" SIZE="25">]
print [<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">]
print [</FORM>]
print [</BODY></HTML>]
```

If you memorize the syntax of the previous two examples, you're well on your way to being able to program dynamic web applications with Rebol. The two-part template required to create CGI programs is the same regardless of how complex a program you create:

1) Design an html web page with a form that gets all the input required from the user. Point the form action to the url of your CGI script.

2) Design a Rebol CGI program to process the data input via the web form. The first three lines should be the same as the format shown in the template. Use "decode-cgi system/options/cgi/query-string" to automatically assign variable words to the data entered on the web form. To provide program output, print html formatted content to the user's browser using standard "print" statements. You can include an html form in the formatted output to call another CGI application.

Learning to use CGI comfortably requires a working knowledge of html. The required basics are very easy to learn. You can study enough html in one day to begin programming CGI applications in Rebol. Understanding how html forms work is the most important part. Try the search below for more information:

```
browse http://www.google.com/search?q=html+tutorial
```

Rebol CGI programming is covered in greater depth at the following links:

```
browse http://rebol.com/docs/cgi1.html

browse http://rebol.com/docs/cgi2.html

browse http://rebol.com/docs/cgi-bbs.html

browse http://www.rebol.net/cookbook/recipes/0045.html
```

The site below provides a free downloadable web server program written entirely in Rebol:

```
browse http://plain.at/vpavlu/plain-dev/r80v5/r80v5.html
```

It provides an alternative to CGI programming that lets you include Rebol code directly in your html pages, using a format similar to the PHP programming language. Using the above server, you can include Rebol code to be evaluated directly in your web pages using the following syntax:

```
<?rebol {Rebol code to be evaluated} ?>
```

For example:

```
<HTML>
<BODY>
The time is currently: <?rebol print now/time ?>.
</BODY>
</HTML>
```

That way, a CGI interface isn't even required. It's a convenient option.

33. Разбор

Встроенная функция «parse»(разбор/синтаксический анализ) - важная часть языка Rebol. Он используется для импорта и преобразования организованных фрагментов внешних данных в формат блока, который Rebol распознает изначально. Он также предоставляет средства анализа, поиска, сравнения, извлечения и обработки информации в неформатированных текстовых данных. Его возможности сопоставления с образцом аналогичны регулярным выражениям, найденным в Perl и других языках.

Основной формат синтаксического анализа:

parse

У parse есть несколько режимов использования. В простейшем режиме текст просто разделяется по общим разделителям и преобразуется в блок Rebol. Для этого просто укажите «none» в качестве правила сопоставления. Распространенными разделителями являются пробелы, запятые, табуляторы, точки с запятой и новые строки. Вот некоторые примеры:

```
text1: "яблоко апельсин груша"
print parsed-block1: parse text1 none

text2: "яблоко,апельсин,груша"
print  parsed-block2: parse text2 none

text3: "яблоко      апельсин                    груша"
print parsed-block3: parse text3 none

text4: "яблоко;апельсин;груша"
print parsed-block4: parse text4 none

text5: "яблоко,апельсин; груша"
print parsed-block5: parse text5 none

text6: {"яблоко","апельсин","груша"}
```

```
print parsed-block6: parse text6 none

text7: {
apple
orange
pear
}
print parsed-block7: parse text7 none
```

Все вышеперечисленные проанализированные блоки оцениваются как ["яблоко" "апельсин" "груша"]. Это полезно, потому что позволяет легко импортировать файлы данных, созданные другими программами. Вы можете, например, использовать правило «none» для импорта файлов CSV, созданных с помощью электронной таблицы или приложения базы данных. Данные автоматически преобразуются в собственные блоки данных Rebol для использования в ваших скриптах.

Если вам нужно разделить данные на основе какого-либо символа, отличного от общих разделителей, вы можете указать разделитель как правило. Просто заключите разделитель в кавычки:

```
text: "яблоко*апельсин*груша"
print parsed-block: parse text "*"

text: "яблоко&апельсин&груша"
print parsed-block: parse text "&"

text: "яблоко    &    апельсин&груша"
print parsed-block: parse text "&"
```

Вы также можете включить несколько смешанных символов для использования в качестве разделителей:

```
text: "яблоко&апельсин*груша"
print parsed-block: parse text "&*"

text: "яблоко&апельсин*груша"
print parsed-block: parse text "*&" ; порядок не имеет значения
```

Все вышеперечисленные проанализированные блоки также оцениваются как одно и то же ["яблоко" "апельсин" "груша"]. Использование режима синтаксического анализа «splitting»(разбиение) - отличный способ добавить отформатированные таблицы данных в ваши программы Rebol.

Разделив нижеследующий текст символами новой строки(возврата каретки/enter), вы столкнетесь с небольшой проблемой:

```
text: { First Name
        Last Name
        Street Address
        City, State, Zip}

endofline: to-string (probe newline)
probe parsed-block: parse text endofline
```

Пробелы включены в правило синтаксического анализа по умолчанию (синтаксический анализ автоматически разделяется на все пустое пространство), поэтому вы получаете блок данных, который более разбит, чем предполагалось:

```
["First" "Name" "Last" "Name" "Street" "Address" "City,"
    "State," "Zip"]
```

Вы можете использовать уточнение «/all», чтобы исключить пробелы из правила разделителя. Код ниже:

```
text: { First Name
        Last Name
        Street Address
        City, State, Zip}

endofline: to-string (probe newline)
parsed-block: parse/all text endofline

; преобразует заданный текст в следующий блок:

; [" First Name" "      Last Name" "      Street Address"
;    "      City, State, Zip"]

; Теперь вы можете обрезать лишнее пространство на каждой из строк:

foreach item parsed-block [trim item]
probe parsed-block
```

и вы получите следующий анализируемый блок, как и предполагалось:

```
["First Name" "Last Name" "Street Address" "City, State, Zip"]
```

Приведенный выше пример можно настроить для извлечения неформатированных строк информации из любого текстового файла в приложение Rebol. Но это еще не все, на что способен синтаксический анализ. Вы также можете использовать его, чтобы проверить, существуют ли какие-либо конкретные данные в данном блоке. Для этого укажите правило (шаблон соответствия) в качестве искомого элемента. Вот пример:

```
print parse ["apple"] ["apple"]

print parse ["apple" "orange"] ["apple" "orange"]
```

Обе строки выше оцениваются как истина, потому что они точно совпадают. ВАЖНО: По умолчанию, как только синтаксический анализ обнаруживает что-то, что не соответствует, все выражение оценивается как ложное, ДАЖЕ, если данное правило встречается в данных один или несколько раз. Например, ЛОЖНО следующее:

```
print parse ["apple" "orange"] ["apple"]
```

Но это просто поведение по умолчанию. Вы можете контролировать реакцию синтаксического анализа на несоответствующие элементы. Добавление следующих слов в правило вернет истину, если данное правило соответствует данным указанным образом:

1) "any"(любой) - правило соответствует данным ноль или более раз

2) "some"(несколько) - правило соответствует данным один или несколько раз

3) "opt"(выбрать) - правило соответствует данным ноль или один раз

4) "one"(один) - правило соответствует данным ровно один раз

5) одно число - правило сопоставляет данные заданное количество раз

6) два числа - правило сопоставляет данные несколько раз, включенных в диапазон между двумя целыми числами

Все следующие примеры верны:

```
print parse ["apple" "orange"] [any string!]
print parse ["apple" "orange"] [some string!]
print parse ["apple" "orange"] [1 2 string!]
```

Вы можете создать правила, которые включают несколько вариантов соответствия - просто разделите варианты знаком "|" и заключите их в квадратные скобки. Верно следующее:

```
print parse ["apple" "orange"] [any [string! | url! | number!]]
```

Вы можете инициировать действия, которые будут выполняться всякий раз, когда будет найдено правило. Просто заключите действия в круглые скобки:

```
print parse ["apple" "orange"] [any [string!
    (alert "Блок содержит строку.") | url! | number!]]
```

Вы можете пропускать данные, игнорируя фрагменты, пока не дойдете до заданного условия или не пропустите его. Слово «to»(до) игнорирует данные ДО тех пор, пока не будет найдено условие. Слово «thru»(через) игнорирует данные до тех пор, пока НЕ ПРОШЛО условие. Верно следующее:

```
print parse [234.1 $50 http://rebol.com "apple"] [thru string!]
```

Настоящая ценность сопоставления с образцом состоит в том, что вы можете организованно искать и извлекать данные из неформатированного текста. Слово «копия» используется для присвоения переменной совпадающим данным. Например, следующий код загружает необработанный html с домашней страницы Rebol, игнорирует все, кроме того, что находится между тегами заголовка(titl) html, и отображает этот текст:

```
parse read http://rebol.com [
    thru <title> copy parsed-text to </title> (alert parsed-text)
]
```

Вот полезный пример, который удаляет все комментарии из заданного скрипта Rebol (любая часть строки, начинающаяся с точки с запятой ";"). Этот код основан на сценарии по адресу http://www.rebol.org/library/scripts/uncomment.r. Сначала он запрашивает имя файла и присваивает содержимое этого файла переменному слову «code». Затем с помощью слов «to» и «thru» содержимое анализируется на предмет поиска любого текста, который начинается с символа «;» символ и заканчивается новой строкой. Блок действий правила синтаксического анализа использует встроенную функцию "remove / part" для удаления этого проанализированного текста из кода. Наконец, проанализированный код отправляется во встроенный текстовый редактор Rebol для просмотра, сохранения и т.д.:

```
code: read to-file request-file
parse/all code [any [
    to #";" begin: thru newline ending: (
        remove/part begin ((index? ending) - (index? begin))) :begin
    ]
]
editor code
```

Чтобы узнать больше о синтаксическом анализе, перейдите по следующим ссылкам:

```
browse http://www.codeconscious.com/rebol/parse-tutorial.html

browse http://www.rebol.com/docs/core23/rebolcore-15.html

browse
https://web.archive.org/web/20100417102240/http://www.rebolforces.com/zine/rz
ine-1-06.html
```

34. Ошибки

Ниже перечислены решения множества распространенных ошибок, с которыми вы столкнетесь при первом эксперименте с Rebol:

1) "** Syntax Error: Script is missing a REBOL header" («** Синтаксическая ошибка: в сценарии отсутствует заголовок REBOL») - всякий раз, когда вы «do»(выполняете) сценарий, сохраненный в виде файла, он должен содержать хотя бы минимально необходимый заголовок в верхней части кода. Просто включите следующий текст в начало скрипта:

```
Rebol []
```

2) "** Syntax Error: Missing] at end-of-script" («** Синтаксическая ошибка: отсутствует] в конце скрипта») - вы получите эту ошибку, если не поставите закрывающую скобку в конце блока. Вы увидите аналогичную ошибку для незакрытых строк. Приведенный ниже код выдаст вам ошибку, поскольку в конце блока отсутствует символ "]":

```
fruits: ["apple" "orange" "pear" "grape"
print fruits
```

Вместо этого должно быть:

```
fruits: ["apple" "orange" "pear" "grape"]
print fruits
```

Отступы у блоков помогают находить и устранять подобные ошибки.

3) "** Script Error: request expected str argument of type: string block object none" («** Ошибка сценария: запрос ожидаемого аргумента str типа: объект строкового блока нет») - этот тип ошибки возникает, когда вы пытаетесь передать неверный тип значения функции. Приведенный ниже код выдаст вам ошибку, потому что Rebol автоматически интерпретирует переменную веб-сайта как URL-адрес, а для функции «alert»(предупреждения) требуется строковое значение:

```
website: http://rebol.com
alert website
```

Приведенный ниже код решает проблему путем преобразования значения URL-адреса в строку(to-string) перед передачей его в функцию предупреждения.:

```
website: to-string http://rebol.com
alert website
```

4) "** Script Error: word has no value" («** Ошибка сценария: слово не имеет значения») - неправильное написание вызовет этот тип ошибки. Вы столкнетесь с этим каждый раз, когда попытаетесь использовать слово, которое не определено (либо изначально в интерпретаторе Rebol, либо вами в предыдущем коде):

```
wrod: "Hello world"
print word
```

5) ВАЖНО: вот особенность Rebol, которая не вызывает ошибки, но может вызывать запутанные результаты, особенно если вы знакомы с другими языками:

```
unexpected: [
    empty-variable: ""
    append empty-variable "*"
    print empty-variable
]

do unexpected
do unexpected
do unexpected
```

В строке:

```
empty-variable: ""
```

не переинициализирует переменную в пустое состояние. Вместо этого каждый раз, когда блок запускается, «empty-variable»(пустая переменная) содержит предыдущее значение. Чтобы вернуть переменную в пустое значение, как и предполагалось, используйте слово «copy»(копия) следующим образом:

```
unexpected: [
    empty-variable: copy ""
    append empty-variable "*"
    print empty-variable
]

do unexpected
do unexpected
do unexpected
```

6) Load(Загрузка)/Save(охранение), Read(чтение)/Write(запись), Mold(прессование), Reform(преобразование) и т.д. - еще одна путаница, с которой вы можете столкнуться изначально с Rebol, связана с различными словами, которые читают, записывают и форматируют данные. Например, при сохранении данных в файл на жестком диске вы можете использовать одно из слов «save»(сохранить) или «write»(записать). «Save»(Сохранить) используется для хранения данных в формате, более удобном для Rebol. «Write»(Запись) сохраняет данные в сырой, «не пеREBOLизированной» форме. «Load»(Загрузить) и «read»(прочитать) имеют схожие отношения. «Load»(Загрузка) считывает данные таким образом, чтобы их можно было автоматически понять и использовать в коде Rebol. «Read»(Чтение) открывает данные точно в том формате, в котором они сохранены, байт за байтом. Как правило, данные, которые являются «сохраненными»(save), также должны быть «загружены»(load), а данные, которые «записывают»(write), должны быть «прочитаны»(read). Для получения дополнительной информации см. Следующие статьи Rebol:

```
browse http://rebol.com/docs/words/wload.html

browse http://rebol.com/docs/words/wsave.html

browse http://rebol.com/docs/words/wread.html

browse http://rebol.com/docs/words/wwrite.html
```

Другие встроенные слова, такие как "mold"(прессование) и "reform"(реформатирование), помогут вам работать с текстом способами, которые либо более удобочитаемы для человека, либо в большей степени читаются интерпретатором Rebol. Для более полного объяснения см.

```
browse http://www.rebol.net/cookbook/recipes/0015.html
```

7) Порядок приоритета математических операций в Rebol ВСЕГДА идёт слева направо, независимо от типа выполняемых операций. Если вы хотите, чтобы сначала вычислялись определенные математические операторы, их следует заключить в круглые скобки или поставить первыми в выражении. Например, для интерпретатора Rebol:

```
2 + 4 * 6
```

тоже самое, что:

```
(2 + 4) * 6  ;сперва выполняется левая сторона...

== 6 * 6

== 36
```

Это противоречит другим общепринятым правилам математических операций. Например, во многих языках умножение обычно выполняется перед сложением. Итак, то же выражение:

```
2 + 4 * 6
```

будет выполнено как:

```
2 + (4 * 6)  ; сперва умножение, затем сложение

== 2 + 24

== 26
```

В Rebol порядок приоритета слева направо согласован, легко запоминается и сохраняется для всех математических выражений. Знаки конца строки даже не требуются, как в других языках (например, точка с запятой в языках типа C), поэтому следующий пример:

```
save %text.txt read http://rebol.com print sort read %text.txt
```

может быть написано чуть более понятно/интуитивно, например так:

```
save %text.txt (read http://rebol.com)
print (sort (read %text.txt))
```

Весь этот сценарий выполняется следующим образом: функция "save" принимает два параметра - сначала имя файла, затем данные для записи. Эти данные считываются с http://rebol.com (следующее выражение по порядку). Следующая на очереди функция print принимает один параметр. Эти данные представляют собой отсортированный текст, прочитанный из файла text.txt. Все просто оценивается слева направо. Привыкание к этому мыслительному процессу может стать источником некоторых первоначальных ошибок, если вы уже привыкли к другим языкам.

Перехват ошибок:

Есть несколько простых способов уберечь вашу программу от сбоя при возникновении ошибки. Слова "error?"(ошибка?) и «try»(попытка) вместе обеспечивают способ проверки и обработки ожидаемых ошибок. Например, если подключение к Интернету отсутствует, приведенный ниже код внезапно завершится с ошибкой:

```
html: read http://rebol.com
```

Приведенный ниже скорректированный код будет более корректно обрабатывать ошибку:

```
if error? try [html: read http://rebol.com] [
    alert "Недоступно."
]
```

Слово «attempt»(попытка) является альтернативой подпрограмме «error? try»(ошибка? попытка). В случае успеха он возвращает оцененное содержимое данного блока. В противном случае возвращается «none»(нет):

```
if not attempt [html: read http://rebol.com] [
    alert "Недоступно."
]
```

Чтобы уточнить, «error? Try [block]» оценивается как true(истина), если блок вызывает ошибку, а «try [block]» оценивается как false(лож), если блок вызывает ошибку.

Полное объяснение кодов ошибок Rebol см .:

```
browse http://www.rebol.com/docs/core23/rebolcore-17.html
```

35. 6 разных Rebol-ов

В этом руководстве рассматривается версия интерпретатора языка Rebol под названием Rebol/View. Это на самом деле только одна из нескольких доступных версий Rebol. Вот краткое описание различных версий:

1) View (вид) - бесплатно для загрузки и использования, он включает языковые конструкции, используемые для создания и управления графическими элементами. View поставляется со встроенным диалектом под названием «VID», который представляет собой сокращенный мини-язык, используемый для отображения общих виджетов GUI. Концепции просмотра и диалекта VID были интегрированы в этот документ. Слово «layout»(макет) в типичном дизайне графического интерфейса «view layout»(макет представление) фактически означает использование кода диалекта VID в прилагаемом блоке. Диалект VID используется внутри интерпретатора Rebol для анализа и преобразования простого кода VID в команды просмотра нижнего уровня, которые составляются с нуля рудиментарным механизмом отображения в Rebol. VID упрощает создание графического интерфейса пользователя без необходимости иметь дело с графикой на элементарном уровне. Но для точного управления всеми графическими операциями полный язык View доступен в Rebol/View и может быть смешан с кодом VID. View также имеет встроенный диалект «draw»(рисования), который используется для создания и изменения изображений на экране. Помимо графических эффектов, View имеет встроенный звук и доступ к функции «call»(вызов) для выполнения приложений командной строки операционной системы или сторонних программ. Последние официальные выпуски View можно скачать с

```
browse http://rebol.com/view-platforms.html
```

Старые версии с

```
browse http://rebol.com/platforms-view.html
```

2) Core (ядро) - текстовая версия языка, обеспечивающая базовые функции. Она меньше, чем View (примерно 1/2 размера файла), без расширений графического интерфейса, но по-прежнему полностью включен в сеть и может запускать все неграфические конструкции кода Rebol. Он предназначен для консольных и серверных приложений, таких как сценарии CGI, в которых средства графического интерфейса не нужны. Ядро также бесплатно и может быть загружено с

```
browse http://rebol.com/platforms.html
```

Старые версии с

```
browse http://rebol.com/platforms-core.html
```

3) View/Pro - Созданный для профессиональных разработчиков, он добавляет функции шифрования, доступ к Dll и многое другое. Лицензии Pro не бесплатны.

4) SDK - также предназначенный для профессионалов, он добавляет возможность создавать автономные исполняемые файлы из сценариев Rebol, а также доступ к реестру Windows и многое другое в View / Pro. Лицензии SDK не бесплатны.

5) Command - еще одно коммерческое решение, оно добавляет в View / Pro собственный доступ к общим системам баз данных, SSL, FastCGI и другим функциям.

6) Command/SDK - комбинирует возможномти SDK и Command.

Некоторые функции, предоставляемые версиями SDK и Command Rebol, были включены в модули, исправления и приложения, созданные сообществом пользователей Rebol. Например, доступ к базам данных mysql и postgre, доступ к dll и автономная упаковка исполняемых файлов могут управляться бесплатными сторонними творениями (параметры можно найти на сайте rebol.org). Поскольку эти решения не соответствуют официальным стандартам Rebol и Rebol Technologies не предлагает их поддержку, для критически важных работ рекомендуются коммерческие решения.

36. Распространение информации: 7 причин изучить и использовать Rebol

Rebol - необычный инструмент, но относительно молодой и малоизвестный. Чтобы стать более полезным и зрелым, ему нужна поддержка растущей базы пользователей. Новые модули, диалекты и полезные функции создаются активными кодировщиками, и это помогает сделать Rebol еще более мощным. Если вам нравятся преимущества, которые предлагает Rebol, найдите время и отправьте эту программу другу. Это поможет распространить «REBOLюцию» :)

1) По сравнению с другими средами разработки Rebol крошечный. В самой тяжелой несжатой форме интерпретатор Rebol составляет ~ 800к. Вы можете распространять весь сжатый язык, включая интерпретатор, встроенную документацию и другие функции, чем можно найти в других многомегабайтных средах разработки, в 200–800 КБ - в зависимости от выбранной вами версии.

2) Код, написанный на Rebol, может работать без изменений в более чем 40 операционных системах. Помимо предоставления отличного решения для разработки кроссплатформенных задач, Rebol также является полезным инструментом для повседневной работы. Он предоставляет последовательный и простой интерфейс, который поможет вам выполнять задачи в любой операционной системе (в качестве многоплатформенного файлового менеджера, текстового редактора, калькулятора, почтового клиента, ftp-клиента, программы чтения новостей, программы просмотра/редактирования изображений и т.д.).

3) Rebol/View и Rebol/Core бесплатны для коммерческого и некоммерческого использования.

4) Ребол легко освоить. Даже абсолютные новички могут научиться выполнять сложные и полезные задачи по программированию за недели. Обширное разнообразие встроенных типов данных, встроенное сетевое подключение, сверхпростое создание графического интерфейса, легкие функции обработки и анализа данных, а также согласованный интерфейс для всех типов

файлов, данных и функций делают его простым в использовании. Нет более простого языка для изучения, и последовательный стиль/синтаксис Rebol поддерживается на протяжении всего обучения.

5) Rebol не ограничивается определенными типами приложений. Вы можете использовать его для создания огромного количества программ с современной графикой, интерфейсов CGI для веб-программирования, сетевых функций, подключения к базам данных и многого другого. Вместо того, чтобы изучать и использовать различные языки и инструменты разработки для выполнения различных задач, вы можете сосредоточиться на улучшении своих навыков в одной простой парадигме.

6) Сообщество пользователей Rebol дружелюбное и знающее. Помощь по большинству проблем и вопросов обычно доступна в списке рассылки Rebol, и время ответа обычно очень быстрое (пользователи Rebol любят демонстрировать, насколько способным и элегантным является Rebol).

7) Язык Rebol может создавать диалекты, которые легко читаются людьми. Как и любой другой язык программирования, в нем используются переменные, определения функций, конструкции объектов и т.д., Но в нем есть некоторые необычные особенности, позволяющие реализовать расширения языка, которые легко реализовать даже неопытным программистам. Встроенные в Rebol диалекты демонстрируют, насколько доступными базовые синтаксические концепции Rebol могут выполнять сложные задачи даже для неподготовленных кодеров. По этой причине Rebol предлагает рядовым пользователям новые возможности для программирования своих компьютеров.

37. Что дальше?

К этому моменту вы познакомились с полезным набором концепций и методов, но вы только начали.

Если вы внимательно посмотрите на код, представленный в этом руководстве, поиграйте с ним и поймете полное значение того, как он работает, вы поймете некоторые из реальных возможностей, гибкости и простоты, с которыми Rebol позволяет вам хранить , извлекать, отображать и иным образом манипулировать данными в любой компьютерной среде. Используя встроенные функциональные слова(команды) в Rebol, вы можете создать и выполнять множество полезных программ без написания большого количества кода, и без сложного обучения. Вы можете создавать графические интерфейсы пользователя для ввода, сохранения, передачи и отображения данных. Вы можете читать, отправлять, изменять и отображать текст электронных писем и веб-страниц. Вы можете создавать, просматривать, изменять и манипулировать изображениями и звуками. Вы можете отображать, сортировать, фильтровать и организовывать блоки текстовых и двоичных данных, подобно базам данных.

Помните, что Rebol может работать практически на любом компьютере, веб-сервере и т.д. И может позволить вам так же легко работать со всеми типами данных: текст, изображения, звуки, денежные суммы, время и даты, веб-адреса и т.д. Он имеет встроенные функции для простой работы с сетями. Ваши программы на языке Rebol могут быть написаны по частям, которые запускаются на разных машинах и работают в тандеме, взаимодействуя через Интернет. Когда в языке нет слов или типов данных, чтобы делать именно то, что вы хотите, вы можете создавать свои собственные функциональные слова и блоки, которые выполняют действия по сбору данных, с которыми вы хотите работать. Как только вы поймете, как сохранять данные с помощью словарных переменных и блочных структур, и выполнять действия с помощью самодельных функций, вы будете готовы более глубоко погрузиться в синтаксис и мыслительные процессы, которые делают язык всемогущем.

Начните изучать и понимать логику и рабочий процесс, который происходит в более длинных программах. Выясните, с какими типами данных вы хотите, чтобы ваши программы работали. Выясните, как получить эти данные в ваших программах (ввод из графического интерфейса, чтение с локального жесткого диска или компакт-диска, загрузка с удаленного веб-сайта или почтового сервера, вывод из другой программы и сохранение на диск и т.д.). Определите структуры данных, которые могут представлять их в вашей программе (простые переменные,

простые списки блоков, составные блочные структуры, системы баз данных и т.д.). Решите, где вы хотите сохранить данные (на локальном диске, на файловом сервере в вашей локальной сети, на веб-сервере в Интернете и т.п.). Решите, какие действия вы хотите выполнить с этими данными (организация, сортировка, категоризация, редактирование, стирание, фильтрация, разделение, передача, отображение, прослушивание и т.п.), И используйте/создайте функции, которые выполняют эти действия. В Rebol все эти вещи легко понять и выполнить. Ознакомьтесь с общими подходами к управлению типичными ситуациями в программах, которые вы хотите написать. Ознакомьтесь со структурами данных, функциями, условными операциями, циклами и общим потоком кода, встречающимися в этих типах программ. Прочтите и запомните код программ, написанных другими. Изучите и запомните все встроенные в Rebol слова. Узнайте, как другие создают и используют новые слова для решения распространенных проблем программирования.

Поступая так, вы научитесь бегло говорить на языке Rebol. Учебник на

```
browse http://www.rebol.com/docs/rebol-tutorial-3109.pdf
```

дает хорошее понимание основных концепций. Это отличный документ для следующего чтения. Чтобы серьезно изучить Rebol, начните с чтения основного руководства пользователя Rebol:

```
browse http://rebol.com/docs/core23/rebolcore.html
```

Он охватывает все типы данных, встроенные функции слов и способы работы с данными, которые составляют язык Rebol/Core (но не графические расширения в View). Он также включает множество базовых примеров кода, которые вы можете использовать в своих программах для выполнения общих задач. Кроме того, обязательно держите справочник функций Rebol под рукой всякий раз, когда вы пишете код Rebol:

```
browse http://rebol.com/docs/dictionary.html
```

Он определяет все слова на языке Rebol и примеры правельного использования их синтаксиса. Это также полезно для перекрестных ссылок(cross-referencing) на функциональные слова, которые выполняют связанные действия на языке. Попутно читайте документы Rebol View и VID по адресу:

```
browse http://rebol.com/docs/easy-vid.html
```

.

```
browse http://rebol.com/docs/view-guide.html
```

.

```
browse http://rebol.com/docs/view-system.html
```

Эти документы объясняют, как писать графические пользовательские интерфейсы в Rebol. Как

только вы поймете грамматику и словарный запас языка, погрузитесь в кулинарную книгу Rebol:

```
browse http://www.rebol.net/cookbook/
```

Он содержит множество простых и полезных примеров кода, необходимого для создания реальных приложений. Когда вы все это прочитаете, закончите остальные документы на

```
browse http://rebol.com/docs.html
```

чтобы получить более конкретные примеры того, как программировать с использованием звука, сети, веб-сайтов, сжатия, безопасности и других широко используемых функций.

Помимо базовой документации, существует библиотека из сотен прокомментированных и одобренных исходных кодов программ Rebol, доступных по адресу

```
browse http://rebol.org
```

Это отличный способ погрузиться в чтение кода реальныз программ на языке Rebol, которыt существует и работают в реальном мире. Кроме того, вы можете использовать его для копирования и вставки примеров кода из других программ в качестве основы для программ, которые вы хотите написать (незабывайте указывать авторов). На том же сайте rebol.org есть также архив списков рассылки с возможностью поиска с почти 45 000 писем. Он содержит ответы на многочисленные вопросы программистов Rebol. Есть много других веб-сайтов, таких как

```
browse http://www.codeconscious.com/rebol/
```

и

```
browse https://web.archive.org/web/20071117142026/http://www.rebolforces.com/
```

которые помогают лучше понять и использовать язык. Большой список веб-страниц и статей, связанных с Rebol, см.

```
browse
https://web.archive.org/web/20160316151940/http://www.dmoz.org/Computers/Pro-
gramming/Languages/REBOL/
```

Наконец, интерпретатор Rebol сам по себе является фантастическим источником кода и справочного материала. Встроенное слово «help»(помощь) обеспечивает вывод необходимой информации о структуре синтаксиса для всех встроенных слов в Rebol, а слово «what»(что) перечисляет все встроенные слова. Вместе эти два слова предоставляют справочную информацию, достаточную для большинства случаев, когда вам понадобится помощь по синтаксису - без какого-либо внешнего руководства. Вы также можете использовать встроенное слово «source»(источник), чтобы получить фактический внутренний код Rebol, который определяет многие из функциональных слов среднего и высокого уровня в языке. Интерпретатор также предоставляет сообществу встроенный способ поделиться полезным кодом.

«desktop»(Рабочий стол) Rebol, который появляется по умолчанию при запуске интерпретатора view.exe, является шлюзом в мир «сайтов», которые разработчики используют для публикации кода для программ, которые они создали. Просмотр общедоступных сайтов, на которых разработчики делают доступным код, - отличный способ более глубоко изучить язык. Весь код из архива rebol.org и многое другое доступно на веб-сайтах (удивительно, насколько много позволяет этот маленький файл view.exe!)

Если вас вообще интересует программирование, вы найдете всю документацию и ресурсы Rebol интересными и немного затягивающими, потому что они показывают вам, как делать практически все, что вы когда-либо представляли себе как программист. Rebol - крошечный инструмент (по сравнению с размером файла других интерпретаторов и компиляторов), он очень мощный и полностью кроссплатформенный. Вы сможете программировать локальные компьютеры с Windows, а также веб-серверы Linux, Mac и другие типы компьютеров, используемых в самых разных средах. Вы сможете программировать бизнес-приложения, аудиовизуальные приложения, сетевые и Интернет-приложения, игры и т.д.. Кроме того, вы узнаете основы программирования, которые вы также перенесете на другие языки.

37.1 Заключительный момент

Rebol - это язык, который аккуратно «обертывает» наиболее распространенные функции, доступные в различных операционных системах. Он представляет собой единый простой формат, позволяющий одинаково разговаривать со всеми компьютерами. У него есть собственный способ говорить, отличный от многих других языков. Говоря техническим языком, эта грамматика и синтаксис называется api. Если вы продолжите заниматься программированием на других языках и в различных средах, вы столкнетесь с API других языков. В конце концов, как серьезный программист, вы, как правило, научитесь работать с необработанным api операционной системы, над которой работаете. Это базовый язык, на который фактически переводится большинство других языков. Поскольку операционной системе требуется быстрый доступ к аппаратному обеспечению компьютера, она написана на языке «нижнего уровня» - языке, который отформатирован так, чтобы думать больше как необработанные вычисления компьютера, а не как человеческая речь.

С Rebol вы можете делать самые типичные вещи, которые хотят делать программисты, но есть много функций в различных api операционной системы, которые не включены (например, доступ к веб-камере, ввод звука, низкоуровневое управление оборудованием и т.д.). Для этого будьте готовы изучить необработанный api операционной системы и языки, на которых он был написан. В Windows, Unix, Macintosh и других платформах это обычно означает изучение синтаксиса и структуры языков «C» и «C ++». Кроме того, очень важно изучить общие методы доступа к файлам с общим кодом, таким как .dll. После того, как вы изучите полный api Rebol, это хорошее направление для ваших исследований ...

Наслаждайтесь!

38. Обратная связь

Для обратной связи, отчетов об ошибках, предложений и т.д., Пожалуйста, свяжитесь со мной по электронной почте:

```
print reverse {moc tod znosselcisum ta lober}
```


Перевод на русский язык выполнил Google Translate. Его немного подправил Сергей, с замечаниями:

```
print reverse {ur.kb@pawsym}
```