Узнайте RFO Basic - самый простой способ создать Программы

Автор: Ник Antonaccio
Обновлено: (! РФО Базовая версия 1.71 или выше требуется для этого урока) 2-26-2013
Узнайте самый простой и продуктивной инструмент развития для Android.
Там совершенно нет простой способ создания полнофункциональных приложений для Android телефонов и устройства.
Посетите http://rfobasic.freeforums.org задавать вопросы о BPO Basic.
Перейти к http://laughton.com/basic/versions/index.html скачать RFO Basic.
См De Re Basic! для справочной документации API .
Проверьте easiestprogramminglanguage.com и learnrebol.com за связанных учебники углубленного рабочего стола программирования о REBOL языком,
и freeconsignmentsoftware.com для проекта в ней написано.

Содержание:

1. Введение RFO Basic!

Что РФО Основные? Почему его использовать?

- РФО Основные однозначно *простой* и *продуктивной* инструментом развития, которые могут быть использованы для создания мощных приложений для Android телефонов и устройств. Там не абсолютно никакой простой способ запрограммировать приложений для Android.
- РФО Основные работает полностью на вашем Android устройстве. Это крошечный, автономный, на устройстве программирования решение, которое *не требует никакого программного обеспечения, установленного на настольном* компьютере. Вы *можете* создать РФО Основные приложения на настольном компьютере, и если вы установите API Eclipse IDE и Android, вы сможете производить полноценные Android-файлы APK ("приложения", "Программы"), которые могут быть распределены в Android-магазин приложений. Для тех, кто хочет простой возможное решение развертывания, есть крошечная бесплатная программа для окон, которые автоматически создает APKs для вас. Никаких дополнительных знаний о среде Android не требуется.
- РФО Основные является мощным. Он обеспечивает доступ к **аппаратным, датчики, звук, графика, мультитач, файловая система, SQLite, сетевые розетки, FTP, HTTP, Bluetooth, HTML интерфейс, шифрование, SMS, телефон, электронная почта, текст-в-речь, распознавание голоса, GPS, математика, строковые функции, список функций** и т.д. В отличие от других ароматов BASIC, которые ограничиваются введением фундаментальные понятия программирования, РФО Основные богатый, современный, признакам заполнены язык. Несмотря на простоту в использовании, это не игрушка, а практический инструмент для создания полезного для Android программного обеспечения.
- РФО Основные 100% *бесплатно* и с открытым исходным кодом.
- РФО Основные поддерживается дружественной и знающих сообщества активных разработчиков во всем мире. Это регулярно улучшается, с обновления, как правило, происходящих несколько раз в месяц.
- РФО Основные имеет полезную *встроенный в помощь* для всех доступных функций и языковых конструкций, и несколько десятков полных примеров программ, доступных сразу непосредственно на Android устройства.
- РФО Основные является *до смешного просто в освоении и* использовании. Там нет глубоких понятия языка программирования вселить, нет никакого комплекса фоновых знаний, чтобы понять, не огромная IDE для рабочего стола, чтобы установить, нет громоздким Android SDK, чтобы узнать, нет языка JAVA хлама в мутной кода, не App Store утверждения не просить, или любой из типичные препятствия, возникающие при создании программ для мобильных устройств. Просто скачайте 1/2 Мэг РФО Basic приложение на свой телефон, ознакомиться с включенными программ справочных функций и примеров, и начать писать простую команду, основанную процессуальный кодекс в текстовом редакторе. Есть десятки кратких функций для управления данными, для управления оборудования, чтобы взаимодействовать с пользователями, и быстро завершить многие виды *полезной* работы. Переводчик / выполнения устанавливает менее чем за минуту, и вы

можете настроить настольный компьютер как редактирование кода машины с дополнительными пару минут.

- РФО Основные является *легко* достаточно для абсолютных новичков и средних пользователей компьютера, чтобы немедленно работать, но *мощный* достаточно для различных профессиональных применений. Вы можете рассматривать его как простой калькулятор личной или полезной приложение для вашего телефона, или, как полномасштабной среды разработки для создания приложений товарной. Вы можете написать быстрые скрипты на вашем устройстве для работы с файлами, просматривать и редактировать данные базы данных, текст процесса, загрузки веб данные, выполните FTP загрузки, делать фотографии, обрабатывать изображения, легко получить доступ к функции распознавания голоса и других особенностей Android и т.д., или вы можете использовать его для разработки графических игры или сетевым приложениям, вы можете продать на рынке - и все между ними. (Учтите, что РФО основным является GPL лицензии, поэтому, если вы отпустите любую коммерческую программу, которая включает в себя свой код, вы также должны освободить источника).
- Опытные разработчики и ИТ-специалисты найдут, что РФО Основные позволяет широкий спектр коммунальных скриптов, которые будут разработаны и развернуты в считанные минуты и часы, а не дни и недели, и с резко короткий кривой обучения.
- РФО Основные можно забрать сразу, кто это используется в любой форме традиционной Basic. Вся сообщество разработчиков и начинающих программистов могут использовать свои существующие навыки в этой знакомой и удобной среде, строить *полезные*мобильные приложения.

Если вы новичок в Basic или даже в программировании в целом, не волнуйтесь. РФО Основные является *чрезвычайно простым* подмножество других знакомых основных языках программирования. Вы можете прочитать всю API и справочной документации в течение одного дня. Этот учебник будет обеспечивать достаточно глубокое понимание, работая примеры кода, и тематические исследования, чтобы вы создания приложений всех типов, полностью по своему усмотрению.

2. Начало работы: Загрузка и установка BPO Basic, Hello World

РФО Основные переводчик приложение (программа), что работает на вашем Android устройстве. Это переводит текст написан в синтаксисе языка Основные BPO ("исходный код") в которой другие приложения пользователи могут запускать на своих устройствах. Чтобы получить бесплатный RFO Basic приложения, перейдите по ссылке:

http://www.laughton.com/basic/versions/index.html

Перейдите по ссылке выше в вашей *веб-браузере* телефона, скачать самую последнюю версию *Basic.apk* файла на Android устройства, и запустите установку (поочередно, вы можете выбрать для установки с помощью Google Play Store). После того, как вы получили РФО Основные установки, нажмите кнопку "Basic!" значок приложения, чтобы запустить его, нажмите кнопку Menu для Android на вашем телефоне, выберите "Clear" и введите следующий код в текстовое поле:

```
печатать "Привет, мир!"
```

Чтобы выполнить код, снова нажмите кнопку Menu для Android, и выберите "Выполнить".

Прежде чем идти дальше, дать ему попробовать. Скачать RFO Basic и введите код, чтобы увидеть, как это работает. Это чрезвычайно простой и буквально занимает всего несколько секунд, чтобы установить. Чтобы извлечь выгоду из этого урока, это важно, что вы *введите или вставьте каждый пример кода в BPO Базовая* переводчика, чтобы увидеть, что происходит.

3. Несколько Простые примеры

3.1 Крошечные Блокнот приложение

Чтобы использовать для работы с BPO Basic, и, чтобы увидеть, насколько простой код, попробуйте следующее:

Запустите РФО Basic приложение, нажмите кнопку Android меню, выберите "Clear" и введите следующий код в текстовый редактор области:

```
File.Exists б ", temp.txt"
если б = 0, то console.save "temp.txt"
grabfile S $ ", temp.txt"
text.input R $ S $
Печать P $
console.save "temp.txt"
```

Эта программа сначала проверяет, если текстовый файл "temp.txt" существует на Android устройства. Если файл "temp.txt" не существует, он создает новый пустой текстовый файл с именем "temp.txt". Затем он читает содержимое файла «temp.txt", и позволяет пользователю редактировать его. Наконец, она сохраняет отредактированного текста в файл "temp.txt". Это крошечная программа является реальной, полезной блокнот приложение, которое вы можете использовать, чтобы записать и сохранить важную текстовую информацию любого рода.

Чтобы сохранить программный код выше, нажмите кнопку Android меню, выберите "Сохранить" и введите имя файла (то есть, "edit.bas").Запустите программу, нажав кнопку Android меню и выберите "Выполнить". Вы можете загружать, редактировать и запускать эту сохраненную программу в любое время в BPO Basic, нажав на кнопку меню Android, и выбрав "Load".

Если у вас есть какие-либо проблемы, вы можете загрузить код, указанный выше от http://www.rfobasic.com/edit.bas. Вы можете попробовать рабочую версию этого приложения на http://www.rfobasic.com/edit.apk.

3.2 Тини Таймер приложение

Теперь создайте новый пустой файл .bas (! Basic -> Кнопка меню -> Очистить) и введите следующий код:

```
S = часы ()
печатать "Таймер началась в:"; s
делать
    INKEY $ K $
    если ((K $ <> "@") и (K $ <> "ключ 4")), то пусть делается = 1
не до готовности
e = часы ()
печать "ВРЕМЯ:"; (e - e) / 1000; "Секунд"
```

Сохраните и запустите программу. Это полезный секундомер приложение, точным, чтобы тысячных секунды. Если у Вас возникли проблемы с Например, вы можете загрузить и запустить заполненную код из http://www.rfobasic.com/timer.bas. Вы можете попробовать рабочую версию этого приложения на http://www.rfobasic.com/timer.apk.

Как вы можете видеть, полезные программы РФО Основные может быть *очень* коротким и простым для создания.

3.3 Примеры сайте

Есть ряд интересных примеров программ, созданных членами сообщества в http://laughton.com/basic/programs/. Вы можете узнать много нового о BPO Basic просто исследовать загружаемый код в этих приложениях. По умолчанию, все загруженные РФО основной

код программы должен быть сохранен в "/ / SDCard / RFO основного / источник" папку на вашем Android устройстве. Любые загруженные ресурсы, необходимые для запуска программ (изображения, файлы конфигурации и т.д.) должны быть помещены в "/ / SDCard / RFO основного / данных" папку. SQLite базы данных должны быть помещены в "/ SDCARD / РФО-основной / баз данных" папку. Более подробную информацию о коде, подаренной сообщества РФО Basic доступна на http://rfobasic.freeforums.org/shared-program-instructions-7-30-11-t125.html.

3.4 Редактирование и Workflow Советы

На многих небольших устройств Android, текстовый редактор, встроенный в BPO Basic не является самым удобным или продуктивной окружающей среды для редактирования кода. Это также часто полезно копировать / вставить код из файлов или веб-сайтов (например, учебник) этой с настольного компьютера, непосредственно в интерпретатор РФО Basic. Один из способов сделать это является использование Android Emulator, который работает на настольном компьютере. Youwave и Bluestacks два эмуляторы, которые работают с BPO Basic в Windows, и Mac OS. К сожалению, эмуляторы большие программы, которые используют много системных ресурсов, поэтому они могут работать медленно на всех, кроме самых мощных настольных компьютеров.

Более эффективный вариант для подключения Android устройство к настольному компьютеру с помощью кабеля USB с, и установить тип подключения к диск. При этом сделано, вы можете редактировать и сохранять программный код с помощью любого текстового редактора, а затем перезагрузить и запустить его в BPO Basic.

Беспроводной вариант заключается в использовании программы, такие как передачи Wi-Fi Файл, чтобы настроить сервер на вашем телефоне, а затем подключиться с помощью веб-браузера на настольном компьютере, передавать отредактированные файлы. Это простой способ получить код и обратно между Android и PC беспроводным, без создания каких-либо программное обеспечение на настольном компьютере.

Android File Explorer приложений, таких как свободного Explorer, ES File обеспечить еще один способ редактирования и Tranfer код. Загрузить отредактированные файлы кода на FTP- сервер или сохранить в общей сетевой папке (на рабочем столе ПК, щелкните правой кнопкой мыши любую папку и выберите "долю" с, разделить его на вашей локальной сети). Вы можете открывать файлы непосредственно с помощью Проводника ES File - это обеспечивает простое прямое загрузку и редактирование файлов на FTP-серверов и акций LAN. От редактора ES File, вы можете скопировать / вставить код в редакторе РФО Basic. Это может быть удобно, если несколько человек должны разделить / редактировать и тот же код.

Если у вас много кода редактирования непосредственно *на* вашем Android устройстве, это может быть полезно использовать программное обеспечение клавиатуры, такие как тип А. И., который поддерживает Undo / Redo, клавиш управления курсором и других полезных функций редактирования. Эта клавиатура может быть использована непосредственно в Основном редакторе кода BPO, но и другие редакторы, такие какJota и DroidEdit предоставлять аналогичные функции.

Инструмент предпочитают этого автора передачи Wi-Fi файла. Редактировать код с помощью своего любимого текстового редактора рабочего стола, сохранить и передать / SDCard / RFO основного / источник / папки на вашем Android с помощью Wi-Fi Transfer File, затем (повторно) открыть и запустить его с BPO Basic. (Будьте уверены, чтобы проверить "Заменить существующие файлы", когда с помощью этого процесса).Эта процедура проста, и позволяет копировать / вставить / редактировать / запустить туда и обратно между любым настольным ПК и Android устройств в течение нескольких секунд, без проводов, и без *какой-либо* конфигурации программного обеспечения по передаче рабочего стола (Wi-Fi File делает *не* требует общие папки, регистрационные credententials и т.д. - только браузер и подключение к сети).

Для того, чтобы выполнить многочисленные Редактировать / передачи / Run циклов, необходимых для написания и отладки любого типа кода, важно, чтобы стать совершенно как дома (и быстро), по крайней мере один из этих подпрограмм.

4. РФО основного языка Основы

4.1 Функции / команды

Как и в любом современном языке программирования, чтобы использовать RFO Basic, вы должны узнать, как использовать **"команды"**(попеременно называемые "функции" или "методы" на разных языках программирования). Команды являются слова, которые выполняют действия. Они, как правило, обрабатывать данные в некотором роде. Командные слова следуют данных **"параметров"** (также называется "аргументы"). В BPO Basic, несколько параметров, разделенных запятыми. Введите и выполните следующую команду в окне кода РФО Basic, чтобы увидеть, как это работает:

```
всплывающее "Привет, мир!", 0,0,1
```

Обратите внимание, как изменение параметров изменяет, как команда выполняет:

```
всплывающее окно "перешел, и короче продолжительность ...", 50,50,0
```

Команда "Печать" является очень важным. Вы можете распечатать один элемент, либо номер или текст:

```
печать "пять"
печать 5
```

Вы можете распечатать несколько элементов, разделенных запятыми:

```
печать "один", "два", "три"
```

Вы можете распечатать несколько элементов рядом друг с другом, используя точку с запятой:

```
печать "один"; "два"; "три"
```

Если вы в конечном команду печати с запятой, РФО Основные будет ждать следующей команды печати, на самом деле отображать любые данные:

```
Печать "в ожидании ...";
паузу 2000
печать "сделано"
```

Еще сразу полезна команда "console.save". Console.save сохраняет консольный вывод из BPO Basic (что-нибудь, что была отпечатана в текущей программе), в указанный файл:

```
печать "тест"
console.save "temp.txt"
```

Многие из команд, в BPO Basic следовать формат: category.action. Например, команда GPS "gps.latitude" получает текущую широту, "gps.altitude" получает текущую высоту. Аудио Команда "audio.play" начинается воспроизведение звука, и "audio.record" начинает запись звука и т.д.

Примечание: технически, есть разница между "командованием" и "функции" в BPO Basic (функции требуют скобки и всегда возвращает значение), но для целей этого урока мы будем использовать термин ***"функции", чтобы относятся ко всем словам действий, функций и команд.***

4.2 Комментарии

Комментарии маленькие читаемые заметки человека, добавленные в программы, которые полностью игнорируются интерпретатором языка компьютера. Их целью является, чтобы напомнить, что программисты конкретный кусок кода делает, и они предназначены только для чтения человеком. В BPO Basic, есть три типа комментариев:

```
печать "Привет"% весь текст после знака процента игнорируется

! Восклицательные точки заставить целые строки, которые будут игнорироваться

!!
Вы можете комментировать любое количество из нескольких строк, окружающих
им с двойными восклицательными знаками.
Ничто в этой программе не выполняется, за исключением первого
"печать" Привет "" код.
!!
```

4.3 Переменные

Переменные слова, которые программист приходит с чтобы *маркировать* и представляют сохраненные данные, так что данные могут быть переданы и использоваться позже в программе. Чтобы создать метку переменной, программисты выбрать любую комбинацию из букв и цифр, пока переменная начинается с буквы и не содержит никаких специальных символов. Текст, или *"строка"* переменных в BPO Basic, как правило, сопровождается символом "$". Слово сопровождается знака "=" устанавливает метку переменной:

```
человек $ = "Джон"
```

Наряду с функциями, переменные, пожалуй, наиболее фундаментальные понятия и распространенных во всем программировании, независимо от языка. После ввода кода выше была выполнена, этикетка "человек $" может быть использован в любом месте (без знака равенства), чтобы представить текст "Джон":

```
всплывающее человек $, 0,0,1
```

Значения переменных могут быть *изменены* в любой момент в программе, - это одна из главных причин для их использования:

```
человек $ = "Дэйв"
всплывающее человек $, 0,0,1
```

В BPO Basic, переменные не зависят от регистра:

```
всплывающее человек $, 0,0,1
всплывающее окно ЧЕЛОВЕК $, 0,0,1
всплывающее человека $, 0,0,1

! переводчику РФО Basic, три линии выше одинаковы
```

Переменные имеют чрезвычайно важное значение в каждом типе программирования, потому что они представляют собой сменные ("Variable") данные. Ввод данных пользователем, читать из файла, возвращенный с помощью датчика, и т.д., все это может быть представлено переменных. Вы можете хранить весь текст книги читать с веб-сервера в переменную с надписью "книга $". Вы можете прочитать имена файлов, содержащихся в папке на карте памяти SD и дать этого списка метку "файлы" $. Вы можете задать свой пользователю выбрать MP3 из файловой системы и присвоить ему значение переменной ярлык "песня" $. В любом из этих случаев, вы можете использовать varaiable этикетки вы дали данные для загрузки, сохранения, отображения, обработки и иным обрабатывать представленные данные, все с помощью одного переменной слово.

4.4 Функциональные Возвращаемые значения

Большинство функций производства **"Возвращение"** значения. Возвращаемые значения являются *выходные* данные получены, когда функция обрабатывает любые заданные входные данные. Данные, обработанные и возвращаемые функцией, как правило, хранятся в переменных:

```
опущенные нижние = $ (", эта функция преобразует текст в нижнем регистре.")
печать снижена
```

В некоторых случаях выход (возвратное значение) одной функции могут быть использованы непосредственно в качестве входа для другой функции:

```
печатать более низкие $ (", эта функция преобразует текст в нижнем
регистре.")

! Предыдущий строка содержит 2 функции: "Печать" и "нижний" $
! Выход нижнего $ функции используется в качестве входного
! параметр функции печати.
```

В BPO Basic, **возвращаемое значение (ы) функций (команд) наиболее часто определяется как *параметр (ы)* - помещен в список параметров, сразу же *после того, как* функции** слова. Следующая функция "grabfile" сохраняет текст прочитали из указанного файла, в произвольно выбранной переменной "с $":

```
печать "Hello World!" % Печатается текст
console.save "temp.txt"% это экономит печатный текст в файл
grabfile S $ ", temp.txt"% Об этом говорится в файл и сохраняет
                          % Возвращаемых данных в переменной S $
Принт S $% это печатает содержимое переменной
```

Возвращение значения выходного многочисленные функции в BPO Basic можно использовать в ваших программах, чтобы выполнить полезные цели. Функция "вход" позволяет запросить одну строку текста, или номер, от пользователя:

```
вход "Введите текст", с $ вход% текста пользователем хранится в переменной S
$
Принт S $
```

Обратите внимание, что следующая строка содержит возвращаемое значение "N", *без символа "$". N, следовательно, рассматривать как**количество*** переменной:

```
вход "Введите число", п
печать н
```

Функция "вход" имеет *дополнительный* параметр, который позволяет отображать текст по умолчанию в запрашивающей:

```
вход "Введите ваше имя", с $, "Джон"%, то "$" в S $ означает, что это текст
Принт S $
```

Функция "text.input" позволяет пользователю вводить, редактировать и сохранять большие объемы текста. Первый параметр является переменной строка, которая содержит окончательной редакции текста выход с помощью функции. Второй параметр имеет некоторые исходный текст появится в окне редактора:

```
text.input г $, "Редактировать этот текст"% пользователей отредактировал
текст хранится в R $
Печать P $
```

Вы можете назначить переменную прочитанного текста из файла (используя функцию grabfile выше), и использовать этот текст в качестве второго параметра в функции text.input. Это позволяет извлекать, редактировать и сохранять любой текстовый файл на вашем Android устройстве:

```
grabfile S $ ", temp.txt"% Переменная S $ присваивается прочитанного текста
                          % Из файла "temp.txt"
text.input R $ S $% Первоначальный текст в редакторе текст
                          % Сохраняется выше в S $ переменной. После
                          % Пользователь редактирует текст, возвращаемый,
отредактированный
                          % Текст хранится в переменной $ г
```

4.5 Разнообразие полезных примеров функциональных

Вот несколько примеров, которые демонстрируют, как выполнять полезные действия и, как обрабатывать данные в удобном виде, используя функции / команды. Попробуйте ввести или вставить каждый из них в BPO Базовая переводчика, чтобы увидеть, как они работают:

```
! Функция тон берет 2 параметра, высоту и длительность, и играет
! тон:

    тон 440, 2000
```

```
! Функция file.rename принимает 2 параметра, старые и новые имена файлов,
! и переименовывает старый файл с новым именем файла в файле Android
! Система:

    file.rename "oldname.txt", "newname.txt"

! Функция принимает clipboard.get 1 параметр, имя переменной присваивается
! к содержанию полученных из буфера обмена устройства:

    clipboard.get c $
    печать c $% это печатает данные, возвращаемые функцией выше

! Функция tts.init готовит для использования текста в гаджи функций
! Нет параметр не требуется:

    tts.init

! Функция tts.speak занимает 1 параметр, часть текста, чтобы говорить:

    tts.speak "Я жив"

! Этот код читает текущую текст в буфер обмена Android:

    clipboard.get c $
    tts.init
    tts.speak c $

! Audio.record.start занимает 1 параметр, имя файла для записи
! в:

    audio.record.start "record.3gp"

! Функция паузы занимает 1 параметр, количество времени, чтобы ждать (в
! миллисекунд):

    паузу 5000

! Функция audio.record.stop не принимает никаких параметров. Это просто
! останавливает запись ранее начала:

    audio.record.stop

! Функция audio.load принимает 2 параметра, название "указатель"
! Количество используется для обозначения определенного звукового файла, а
также имя из
! указанный звуковой файл:

    audio.load c, "record.3gp"

! Функция audio.play играет указанный файл на число указатель
! создан в предыдущей функции:

    audio.play c
    паузу 5000

! Функция часов () не принимает никаких параметров. Она возвращает количество
! миллисекунд с момента вашего устройства был включен:

    S = часы ()
```

```
! Это выводит данные, содержащиеся в числовой переменной, созданной выше:

    печать с

! Функция устройства занимает 1 параметр, имя переменной строкового
! магазин вернулись информации об устройстве:

    $ устройства сек

! Эта линия печатает данные, содержащиеся в переменной строку, созданную
! выше:

    Принт S $

! Gps.open функция не принимает никаких параметров. Это просто готовится к
! использование функции GPS:

    gps.open

! В gps.latitude и gps.longitude функции друг Принимать по 1 параметр,
! имя переменной для хранения возвращаемых значений широты и долготы:

    gps.latitude L1
    gps.longitude L2

! Функция gps.close не принимает никаких параметров. Это просто закрывает
! GPS-устройство открылся выше:

    gps.close

! Это печатает широты и долготы значения, хранящиеся в возвращении
! переменные, созданную выше:

    печатать "Ваш широту и долготу:"; L1; ","; L2

! Это присваивает текст к новой переменной строки:

    S $ = "три слепых мышей"

! Функция замены принимает 3 параметра, имя входа и возврата
! Переменная строка, в которой, чтобы найти и заменить текст, текст, чтобы
найти и
! заменить, и замещающий текст - все содержащиеся внутри скобок:

    заменить $ (с $, "слепой", "сексуальная")

! Это печатает заменить строку, возвращаемую функцией выше:

    Принт S $

! Ftp.open функция принимает 4 параметра, URL на FTP-сервере,
! порт, чтобы открыть, имя пользователя и пароль:

    ftp.open "ftp.site.com", 21 ", имя пользователя", "пароль"

! Функция ftp.put принимает 2 параметра, имя локального файла,
! переходом на удаленный FTP в сервере, а путь / имя файла для создания и
! отправить на сервер:

    ftp.put "temp.txt", "/public_html/temp.txt"
```

```
! Ftp.close функция не принимает параметр. Это просто закрывает
! в данный момент открыт FTP соединение:

    ftp.close

! Функция graburl принимает 2 параметра, имя переменной строку, в которой
! чтобы сохранить полученные данные с указанного URL, и URL-адрес, чтобы
прочитать:

    graburl W $ ", http://site.com/temp.txt"

! Это выводит полученные данные, считанные из URL и находится в
! переменная, созданная выше:

    печатать W $

! Функция is_in принимает 2 параметра, строку для поиска, и
! строка для поиска в. Она возвращает значение 1, если строка поиска
! находится в пределах 2 строки и 0, если не найдено:

    Q = is_in («слепые», «три слепых мышей")

! Эта строка печатает "Это там", если строка поиска находится выше
! (если возвращенное значение не равно 0):

    Если Q <> 0, то печать "Это там"

! Функция kb.show показывает на экранной клавиатуре Android. Это не
! принимать какие-либо параметры:

    kb.show

! Функция kb.hide скрывает экранную клавиатуру Android. Это не
! принимать какие-либо параметры:

    kb.hide

! Функция пл () занимает 1 числовой параметр, в скобках, и
! возвращает квадратный корень. Возвращаемое значение может быть присвоено
переменной
! имя:

    S = пл (16)
    печать с

! Эта линия делает ту же самую вещь, как и предыдущие 2 строки. Это просто
! печатает значение функции пл () (возвращаемое значение в SQR ()
! Функция используется в качестве параметра функции печати):

    печать пл (16)

! Функция array.load создает тип списка. Это займет 1 или более
! Параметры, имя переменной присвоить список, и несколько пунктов
! (в случае, ниже чисел), чтобы добавить в список:

    array.load p [], 0, 300, 200, 300, 200, 300

! Функция вибрации принимает 2 параметра, массив длительности, чтобы
приостановить
! и вибрировать, и ряд указать, будет ли этот шаблон вибрации
! следует повторить (-1 означает отсутствие повторения):
```

```
    вибрируют p [], -1

! Gr.camera.manualshoot занимает 1 параметр, "указатель" номер, используемый
для
! обратитесь к фото изображение, снятое:

    gr.camera.manualshoot p

! После выше функции, посмотрите на фото
! /sdcard/rfo-basic/data/image.jpg на вашем телефоне. Результаты сохраняются
! в этом временный файл (позже в учебнике, вы узнаете, как использовать
! возвращаемое значение указателя для отображения и манипулирования
изображение в
! графические макеты).
```

Наиболее важным первым шагом в изучении BPO Basic учится все встроенные команды / функции работают, что их синтаксис, и как они могут быть использованы *вместе, чтобы выполнять сложные операции и взаимодействие с данными.* Программировать телекамеры в целом, и на самом деле все вычисления, это все о обработки данных. Текст вводится пользователем, считывается из файла, или возвращается с помощью датчика аппаратных, например, потенциально может быть использована для определения, какая операция программа должна выполнить следующее, или это может потенциально содержать значение, которое подключено к математической формуле, или он может быть использован для перемещения графического изображения в заданной позиции на экране, или это может быть текст, который должен быть найдены и отсортированы по relavent содержания и т.д. Текст, изображения, звуки и т.д. сохраняются в файлах на Android устройство или FTP папки на веб-сервере могут быть вызваны и использованы в дальнейшем, редактировать, классифицировать, рассчитывается на, и т.д., а часть, например, учета программного обеспечения, инвентаризации, системы управления продаж и другие данные, коммуникации программное обеспечение, игры, и т.д. Числа, текст, бинарные данные аудио, изображения и параметры обработки изображений, параметры производительности видео, списки и связанные с упорядоченная информация о какой-либо реальной тему жизни, графических координат и 3D вычислений движения в играх, сохраненные настройки программы, и т.д. . - все эти вещи, и все остальное, что вычислительное устройство может взаимодействовать с, это *данные* из какой-то, и все, что данные обрабатываются функций.

Список по http://laughton.com/basic/help/De_Re_BASIC!.htm#_Toc317902038 показан формат для всех внутренних функций RFO BASIC, также называется это "API". Остальная часть документа на http://laughton.com/basic/help/De_Re_BASIC!.htm подробно объясняет, параметр (ы) на входе и выходе каждого из этих функций, а также содержит несколько десятков примеров программ, которые демонстрируют их использование.

Примерные программы из документа выше, также автоматически устанавливается на Android устройстве, при установке РФО Basic приложение.Вы можете запустить любой из них в BPO Basic, нажав на кнопку Android меню, затем Загрузить -> Sample_Programs (d). Полный список всех встроенных в РФО Основные функции также, нажав Меню -> Дополнительные -> Команды. Эти два ресурса вместе обеспечивают существенную документацию в течение всего BPO Basic языке, прямо на вашем Android устройстве.

4.6 Объединение

Конкатенация соединение вместе частей текста. В BPO Basic, вы можете объединить текст, используя символ "+":

```
MyString $ = "Привет" + "Мир" + "!"
печать MyString $
```

Сцепление особенно полезно при комбинировании статического текста с данными, хранящимися в переменных или возвращаемых функций. Вы будете использовать код, подобный следующему в *практически каждой программы вы* пишете:

```
Имя $ = вход "Что ваше имя?"
Сообщение $ = "Хорошо встретиться с вами" + имя + $ "!"
всплывающее сообщение $, 0, 0, 1
```

Поскольку вы не можете писать назвать Неизвестный пользователя в вашей программе, вы должны использовать переменную, и объединить, что текст с другом статического (неизменного) текста, когда вы хотите, чтобы обратиться к имени пользователя в вашей программе.

Для объединения текстовых строк на несколько строк, используйте символ тильды (~):

```
печать "весь этот текст от появляется" + ~
    "одна линия."
```

Функция "Время" устанавливает все переменные после возвращения в текущем году, месяц, день, час, минуты и секунды. Вот красиво отформатирована concatened строка, которая отображает текущую дату и время:

```
Время у $, $ м, d $, $ ч, п $, с $

MyTime $ = "Текущая дата" м $ + "-" д $ + "-" + у + ~ $
    ", И время" + Н $, ":", п $, ":" с $

всплывающее окно MyTime $, 0, 0, 1
```

Опять же, так как программист не может написать произвольный текущее время пользователя в код, переменные и сцепление должны использоваться для получения текущего времени, и обращаться к нему по динамически генерируемым текстом.

4.7 Некоторые основные принципы форматирования текста

Вы можете включать кавычки внутри строк, предварив их с обратной косой черты (\):

```
всплывающее "\" цитирует текст \ "", 0,0,1
```

Текст Многострочный использует "\ N" для представления возврат каретки:

```
линии = "Line 1 \ nLine 2 \ 3" nLine
печать строк
```

5. Условия

Условия используются для управления потоком программы ..., чтобы "решения" в вашей программе. Они являются одними из самых важных понятий во всех видах программ.

5.1 Если

Самое основное условное оценка ", если":

```
если (это выражение истинно), то
    сделать этот блок кода
ENDIF

! Скобки не требуется
```

Операторы Математические обычно используется для выполнения условных оценок: = <> <> (равно, меньше, чем, больше чем, не равно):

```
Время у $, $ м, d $, $ ч, п $, с $

ч = Val (ч $) функция% Val () преобразует текстовые данные в числа.
                      % Переменная "ч" теперь хранит это число.

Если H> 12, то%, если значение часа позже, чем 12 часов, то ...
    печать "Это после полудня."
ENDIF
```

Приведенный выше код можно записать следующим образом. Час текст преобразуется в число, в строке:

```
Время у $, $ м, d $, $ ч, п $, с $
если вал (ч $)> 12, то
    печать "Это после полудня."
ENDIF
```

5.2 Пробелы и отступы

РФО Основные *не требует концы строк* на концах строк кода, и вы можете вставить пустой белый пространства (вкладки, пробелы, и т.д.) по желанию в код. Это стандартная практика в практически все языки программирования, чтобы отступ блоков кода, которые выполняют группу смежных видов деятельности. Например, вы могли бы поставить несколько, если-то условия все в одной строке:

```
! Устанавливает некоторые числовые переменные (exp1 и EXP2 теперь оба
равняться числу 1):

exp1 = 1
exp2 = 1

! Выполните 2 отдельные оценки, если с помощью переменных, все на одной
строке:

если (exp1 = 1), то (если (exp2 = 1), то (печать "Да"))
```

Но это гораздо легче понять логику каждого соответствующего блока кода выше, если эти условные *блоки, каждый с отступом.* Следующий код делает ту же самую вещь как выше коде. Он проверяет второе условное **оценку, только если первая оценка (если переменная exp1 =**

1) верно. Так, *все* с отступом внутри этой внешней окружающей если-ENDIF Блок происходит только если первое условие оценивает к истине.Это гораздо легче следовать логике в этом отступом формате:

```
exp1 = 1
exp2 = 1

если (exp1 = 1)
    если (exp2 = 1)% только эти 3 линии
        печать "да", если запустить% выше
    ENDIF% оценка правда
ENDIF
```

ПРИМЕЧАНИЕ: если вы пишете весь, если-то условие 1 линии, то слово ", а затем" не требуется. Если вы отделить "если" оценка на своей собственной линии, то слово ", а затем" является *не* требуется. Обратите внимание, что нет ", то" ключевые слова в коде выше.

5.3 Если / остальное

Если / остальное выбирает между двумя блоками кода, чтобы оценить, в зависимости от того данное условие является истинным или ложным.Его синтаксис:

```
если (условие)
    выполнить этот код, если условие истинно
другой
    выполнить этот код, если условие ложно
ENDIF
```

Например:

```
Время у $, $ м, d $, $ ч, п $, с $
если вал (ч $)> 8
    печать "Пора вставать!"
другой
    распечатать "Вы можете сохранить на сон."
ENDIF
```

Эти типы оценок находятся в почти каждом типе компьютерной программы:

```
вход "Имя пользователя:", пользователь $

если пользователь $ = "VALIDUSER"
    печать "Добро пожаловать!"
другой
    распечатать "Вы не являетесь зарегистрированным пользователем."
ENDIF
```

5.4 Если / ElseIf / остальное

Вы можете выбрать между *несколькими оценок* любой сложности с использованием "Если / Elseif / Else" структуру. Если ни один из случаев не оценить, верно, вы можете использовать "остальное", чтобы запустить код по умолчанию:

```
вход "Что ваше имя?", имя $

если is_in ("a", имя $) <> 0
    печатать "Имя содержит букву" a ""
ElseIf is_in ("e", имя $) <> 0
    печатать "Имя содержит букву« e »"
ElseIf is_in ("я", имя $) <> 0
    печатать "Имя содержит букву" я "
ElseIf is_in ("O", имя $) <> 0
    печатать "Имя содержит букву" o ""
ElseIf is_in ("U", имя $) <> 0
    печатать "Имя содержит букву« U »«
другой
    печатать "Имя не содержит каких-либо гласные!"
ENDIF
```

5.5 Несколько Условия: "и", "или"

Вы можете проверить более одного состояния, чтобы быть правдой, используя "и" и "или" оценок. Вот пример, который получает имя пользователя и пароль от пользователя, проверяет, что данные с помощью "если" оценка и предупреждает пользователя, если *оба* ответы верны:

```
вход "Имя пользователя:", имя пользователя $
вход "Пароль:", $ пароль

если ((имя пользователя $ = "VALIDUSER") И (пароль $ = "validpass"))
    печать "Добро пожаловать!"
другой
    печать "Вы не зарегистрированы!"
ENDIF
```

Этот пример отвечает положительно, если пользователь выбирает *любой* из трех любимых цветов:

```
вход "Ваш любимый цвет?", цвет $

если ((цвет $ = "синий") или (цвет $ = "красный") или (цвет $ = "оранжевый")
    печатать "Это один из моих любимых цветов тоже!"
другой
    печать "Нам не нравится какой-либо из тех же цветов :("
ENDIF
```

5.6 Переключатель

Эта оценка позволяет сравнить, как много значений, как вы хотите от главного значения, и запустить выбранный блок кода для любого согласования стоимости с дополнительным блоком по умолчанию кода для выполнения, если ни одного совпадения не найдены:

```
вход "Какой ваш любимый день недели?", favoriteDay $

юго-западный начать favoriteDay $
sw.case "Понедельник"
    печать "Понедельника является худшим! Рабочая неделя начинается ..."
```

```
sw.break
sw.case "Вторник"
sw.case "Четверг"
    Печать "по вторникам и четвергам оба нормально, я думаю ..."
sw.break
sw.case "Среда"
    печать "горб день - неделя на полпути закончена!"
sw.break
sw.case "Пятница"
    печатать "Ура! TGIF!"
sw.break
sw.case "Суббота"
sw.case "Воскресенье"
    печатать "Конечно, выходные!"
sw.break
sw.default
    распечатать "Вы не ввести имя день!"
sw.end

! обратите внимание, что вы можете выполнять те же действия в течение 2 или
более условий
! путем размещения таких условий последовательно перед sw.break
! (как в вторник / четверг и суббота / воскресенье) условия выше
```

Условные конструкции вы видели здесь, поможет ваши программы выполняют соответствующие действия в BPO Basic, на основе ввода пользователя, содержание данных и других потенциально меняющихся ситуациях.

6. Больше Структуры Программа Flow

6.1 Gosub / Вернуться

Вы можете пометить любую часть вашей программы, перейти к той части кода, а затем вернуться к исходной точке прыжки, используя "Gosub" и команды "Возвращение". Используйте двоеточие (:), чтобы маркировать часть кода Перейти к:

```
печатать "Программа началась."
GoSub Whee
GoSub Whee
GoSub Whee
печать "Программа закончилась"
конец

Whee:
печатать "Whee!"
вернуть
```

Такое прыгать кода обычно считается depricated, и обычно заменяется функциональных структур, но они могут быть полезны для создания простых подпрограмм.

6.2 Пред

"Гото" также позволяет перейти к меченым части кода, но не вернуться. Это особенно полезно для перезапуска всю программу. Например, когда игра будет завершена, вы могли бы предоставить возможность перезапустить всю программу, просто называя начало программы с

"перезагрузки:". Следующая программа будет работать бесконечно, пока не остановился (с помощью клавиши возврата на вашем Android устройстве):

```
перезапуск:
печатать "Ahhhhh !!!"
перейти рестарт
```

6.3 Создание собственных функций

Вы можете создавать свои собственные функции для обработки данных. Используйте следующий формат:

```
fn.def <функции> (<параметр данных (ы), внутри скобок>)
    команды для обработки данных параметра
    fn.rtn <data_to_return>
fn.end
```

Вот пример:

```
fn.def среднем (x, y, r)
    cp = (x + y + r) / 3
    fn.rtn cp
fn.end

печать в среднем (4,7,2)
печать в среднем (24,56,14)

a = средний (83, 2, 71)
Распечатайте
```

Используйте функции во избежание дублирования кода, который выполняет те же действия (ы), или обрабатывать данные таким же образом, в нескольких местах в коде.

6.3.1 Глобальные и локальные переменные в функции

По умолчанию, значения, используемые внутри функции рассматриваются как *местные, что означает, что, если какие-либо переменные изменяются внутри функции, они не будут изменены в течение остальной части вашей программы:*

```
x = 10

fn.def changex (x)
    x = x + x
    печать "X =" + x
fn.end

changex (Q)% печатает «X = 20 '

печать "Вне зависимости, X еще:" + x% еще 10
```

Вы можете изменить это поведение по умолчанию, и указать, что любое значение параметра рассматривается как *глобальная* (менялся на протяжении остальной части вашей программы), с

помощью "&" символ на мировом параметра. Это упоминается как "прохождения переменную по ссылке":

```
x = 10

changex (& x)% внутри функции, x теперь 20, и это
                % Был изменен вне функции слишком

печать "Вне зависимости, X теперь изменено на:" + x
```

В большинстве случаев, чтобы получить значение вычисляется в определении функции, вы просто возвращать значение, вместо передачи глобальных переменных. Например, это намного проще просто вернуться расчетной стоимости от вышеупомянутой функции, и использовать возвращаемое значение в другом месте в вашей программе:

```
x = 10

fn.def addedval (x)
    y = x + x
fn.end

y = addedval (x)
печать y
```

Структуры данных: 7. Массивы, списки, расслоения и стеки

В BPO Basic, *несколько штук, сгруппированных элементов данных* хранятся в "массивов", "списки", "пучки", и "стеками". Эти структуры данных используются в основном для хранения *списка* данных. Списки данных чрезвычайно важны практически во всех типах приложений программирования.

7.1 Массивы

Массивы наиболее обычно используется для хранения информации списки, которые *не меняются в* длину. Чтобы создать массив в BPO Basic, используйте функцию "array.load". Параметры функции array.load это список строк *или* чисел, разделенных запятыми - эти два вида данных не могут быть смешаны в массивах (массив должен содержать либо все цифры, или все текстовые элементы). Первый пункт в списке параметров является переменная метка выбран для представления элементов в списке:

```
array.load НУМС [], 1,3,5,2 + 5,9
```

Вы можете выбрать элементы из массива, используя численное "индекс". Индексы в BPO Basic являются ", основанный", то есть количество элементов начинается с 1 (языки программирования многие нуля):

```
печать НУМС [1]% печатает 1 - первый элемент в массиве
печать НУМС [2]% печатает 3 - второй пункт
печать НУМС [4]% печатает 7 - четвертый пункт
```

Вы можете продолжить письменные данные на новые строк кода с помощью тильды (~) символ:

```
array.load дней $ [], "Понедельник", "Вторник" ~
    "Среда", "Четверг" ~
    "Пятница суббота воскресенье"
печать дней $ [1]
печать дней $ [6]
```

Массивы могут быть созданы только ("масштабно") один раз в течение программы. Если вам нужно перезарядить массив с различными значениями, сначала использовать "array.delete ArrayName $ []" функцию, а затем заново создать массив. Это хорошая идея, чтобы получить в привычку использовать array.delete каждый раз при создании массива:

```
array.delete в $ []
array.load в $ [], "ADF", "QWE"
```

РФО Основные содержит ряд функций для организации и управления данными в массивах. Попробуйте ввести или вставить все эти функции в ВРО Базовая переводчика, чтобы увидеть, как они работают:

```
array.reverse Nums []
печать НУМС []

array.shuffle Nums []
печать НУМС []

Array.sort НУМС []
печать НУМС []

Array.Copy НУМС [], newnums []
печать newnums []

array.length lngth, НУМС []
печать lngth

array.sum c, НУМС []
печать c

array.average A, Nums []
Распечатайте

array.min м, НУМС []
печать м

array.max x, НУМС []
Печать X

array.variance v, НУМС []
v печати

array.std_dev SD, Nums []
Печать SD

array.delete НУМС []
печать НУМС []
```

Многомерные массивы позволяют списков списков:

```
тусклый л, [1,2,3], [11,12,13], [21,22,23]
печать л [1] [1]
печать л [2] [1]
печать л [3] [2]
```

Многомерные массивы полезны для хранения матриц координат местоположения в графических приложениях и другие ситуации, в которых группы связанных списков Информация, нужно, чтобы быть помещен в большую одном списке.

ВАЖНО: размер массива не могут быть изменены, как только он будет создан. Если вам нужно добавить новый элемент в массив, за оригинальный размер массива, вы должны сначала сделать его копию, а затем удалить оригинал массив, а затем создать новый массив нужного размера, а затем заполнить ее данными из скопированный массив, а затем добавить новый элемент (ы) к нему. Массивы предназначены для хранения информации списки, которые не меняют, такие как список месяцев в году, в первую десятку высокие баллы в игре, местоположения заданного числа неизменных изображений в графическом макете, и т.д.

7.2 Списки

Списки работать так же, как массивы, но их *размер может быть изменен* по мере необходимости. Списки предназначены для хранения и управления данными пользователя, такими как товарно-материальных ценностей в магазине, имена друзей в базе данных контактов, список файлов в каталоге, и т.д. как массивы, списки могут содержать только один тип данных , Все данные в любом списке должны быть либо все строки (текст), или все числа. Если вам нужно хранить смешанные типы данных, хранить каждый элемент в виде строки, а затем использовать функцию Val () для преобразования числовых значений.

Списки создаются с помощью функции "list.create". Первый параметр говорит, будет ли список содержит текст ("строка") или номер детали.Второй параметр является метка для списка (числовое "указатель" переменной):

```
list.create c, myfriends
list.create n, mynumbers
```

К "list.add" функция добавляет элементы в списке. Первый параметр это метка (как поступил в функции list.create выше). Обратите внимание, что символ тильды (~) используется для продолжения элементы на дополнительных строк кода:

```
list.add myfriends, "Джон", "Боб", "Майк", "Джо" ~
    "Сью", "Билла", "Аманда", "Джим", "Джордж", "Тим"
```

РФО Основные имеет ряд функций, которые позволяют организовать и управлять данными в списках. Попробуйте эти примеры в интерпретаторе РФО Basic:

```
list.size myfriends, S
печать с

list.type myfriends, т $
принт футболка $

list.insert myfriends, 2, "Стив"
list.size myfriends, S
```

```
печать с

list.remove myfriends, 3
list.size myfriends, S
печать с

list.get myfriends, 3, с $
Принт S $

list.replace myfriends, 5, "Павел"

list.search myfriends, "Пол", с
печать с

list.add.array myfriends, дни $ []
list.size myfriends, S
печать с

list.create с, newfriends
list.add newfriends, "Тристан", "Питер"
list.add.list myfriends, newfriends
list.size myfriends, S
печать с

List.toArray myfriends, myfriends $ []
печать myfriends $ [7]

list.clear myfriends
list.size myfriends, S
печать с
```

Списки являются хранения данных рабочая лошадка BPO Basic. Вы будете использовать их регулярно, чтобы управлять всеми типами коллекций данных в ваших программах.

7.3 Связки

Связки позволяют сохранять фрагменты данных в виде списка, с соответствующим «ключ», или этикетке, так что вы можете искать связанное данные позже по имени. Вы могли бы, например, сохранить счет-фактура число в качестве ключа, а имена клиентов в качестве ассоциированного данных в связке. Затем можно сохранить все сведения счета клиента в отдельную структуру списка. Это может, например, обеспечить структуру данных для сохранения и просмотра контактной информации для любого счета сделки обрабатываются с помощью бизнеса. Связки могут содержать смешанные числовые и текстовые значения. Вот функции, которые вы можете использовать, чтобы создавать и управлять пучки:

```
bundle.create н

bundle.put п, "19327", "Фрэнк Томпсон"

bundle.get п, "19327", с $
Принт S $

bundle.keys п, MyKeys
list.get MyKeys, 1, к $
печать к $

bundle.type п, "19327", т $
принт футболка $
```

```
bundle.clear н
```

8. Петли и структурированных данных

"Loop" структуры обеспечивают способы методично повторять действия, управлять потоком программы, и автоматизировать длительные мероприятия обработки данных. Петли часто используются для *перехода по отдельным пунктам в списке* данных.

Не 8.1 Пока / повтора и DO / До

Функция ", а" неоднократно оценивает блок кода в то время как данное условие верно. В то время как петли в следующем формате:

```
в то время как (состояние)
    Блок функций будет выполняться, пока условие истинно
повторение
```

Этот пример подсчитывает до 5:

```
x = 1% создать начальное значение счетчика

в то время как x <= 5
    Печать X
    x = x + 1
повторение
```

В английском языке, что код гласит:

```
"x" вначале равен 1

В то время как x меньше или равно 5:
    отображать значение x
    Затем прибавить 1 к значению x
Повторение
```

"Приращение" значения счетчиков, как и в приведенном выше примере, является чрезвычайно распространенной техникой во всех типов программ. Вы увидите примеры этого регулярно в коде, когда вы хотите сохранить количество элементов.

Вы можете использовать в то время / Повтор создать навсегда повторяющиеся циклы. Просто используйте оценку, которая всегда верно, и использовать функцию «wr.break", чтобы закончить цикл. Обратите внимание, как время цикла и оценки Если отступом четко отделить логику:

```
печатать "Пожалуйста, подождите 5 секунд ...", 0,0,1
alarmtime = часы () + 5000

в то время как 1 = 1% это всегда верно
    Пауза 1000% ожидание 1 секунда
```

```
    если часы () = alarmtime
        печать "5 секунд прошло"% эти линии выполняются только, если
        w_r.break% времени часы = время срабатывания будильника
    ENDIF
повторение
```

Вот более интерактивная версия помощью некоторую информацию, предоставленную пользователем:

```
вход "Что вы хотите, чтобы напомнить о?", EventName $
входные "Секунд, чтобы ждать:", секунд
время окончания = часы () * 1000 + секунд
Время у $, $ м, d $, $ ч, п $, с $
MyTime $ = ч $ + ":" + N $ + ":" + S $
Печать «Сейчас« + MyTime + ", и вы будете предупреждены в"
    + Секунд + "секунд".
в то время как 1 = 1
    паузу 1000
    если часы () = время окончания
        печатать "Это теперь" + EndTime + "и" + + секунд
            ". Секунд прошло Пора:" + $ EventName
        w_r.break
    ENDIF
повторение
```

"У-До" петли не работают аналогично в то время повтора циклов, за исключением того, они гарантируют, что *по крайней мере один итерационный* из операторов внутри цикла выполняется:

```
я = 1
делать
    печать я
Пока я = 1
```

Этот тип цикла часто используется ждать нажатий клавиш от пользователя:

```
делать

    ! Эта функция получает текущий нажатие клавиши:

    INKEY $ K $

    ! Символ "@" символ используется, чтобы представлять не нажимается
клавиша:

    если ((K $ <> "@") и (K $ <> "ключ 4")), то пусть делается = 1

пока не сделано% при нажатии что-то другое, чем не ключ или ключ 4,
остановить
распечатать "Вы нажали '" + K $ + "ключ".
```

Запомните код, указанный выше. Вы будете использовать его, когда вы хотите, чтобы ждать одного символа от пользователя.

8.2 Перебор списки данных: ДЛЯ

«ЗА / Next" петли просто рассчитывать через ряд чисел:

```
для я = 0 до 6
    печать я
Следующий
```

Вы можете использовать условные тесты в пределах ДЛЯ петли, для выполнения различных операций на каждой пронумерованной пункта:

```
для I = 1 до 100
    если ((мод (1,3) = 0) и (мод (я, 5) = 0))% Мод проверяет остаток
        печать "FizzBuzz"% от операции деления
    ElseIf мод (я, 3) = 0
        печать "шипение"
    ElseIf мод (я, 5) = 0
        печать "кайф"
    другой
        печать я
    ENDIF
Затем я
```

Вы можете использовать последовательный подсчет способность для **циклов, чтобы забрать последовательных элементов из массива или *списка, используя пронумерованные*** индексы:

```
! Это создает новый список, обозначенный в переменной "месяцев":

list.create с, месяцев
list.add месяцев, "Январь", "Февраль", "Март", "Апрель", "Май", "Июнь" ~
    "Июль", "Август", "Сентябрь", "Октябрь", "ноябрь", "Декабрь"

! Это получает количество элементов в списке месяцев, и магазины, которые
! значение в переменной "размер"

list.size месяцев, размер

! Этот цикл считается от 1 до числа элементов в списке, и делает
! что-то с каждого пункта в списке:

для я = 1 размера

    ! Это выбирает пункт с номером индекса из списка и присваивает что
    ! Пункт переменной "$" м:

    list.get месяцев, я, м $

    ! Это печатает некоторую объединенных текст, используя строку выше:

    печать "Месяц" + Str $ (я) + ":" + т $

Затем я
```

Опция "Шаг" А для контура позволяет *пропустить* заданное число элементов, каждый раз через петлю:

```
для я = 0 до 12 шаг 3% перейти к каждой третьим номером
    печать я
Затем я
```

И считать в обратном:

```
для я = 5 до 1 шаг -1% запуске на 5, на конце 1, вычесть 1
    печатать я% каждый раз через петлю
Затем я
печатать "Доменная!"
```

"Шаг" особенно полезно, потому что он позволяет ***пропустить*** *элементы массива или* списка:

```
list.create c, месяцев
list.add месяцев, "Январь", "Февраль", "Март", "Апрель", "Май", "Июнь" ~
    "Июль", "Август", "Сентябрь", "Октябрь", "ноябрь", "Декабрь"

list.size месяцев, размер

! Граф от 1 до последнего пункта в списке, пропуская по 3:

для я = 1 размер шага 3
    list.get месяцев, я, м $% поднять каждые 3 пункта
    печать "Месяц" + Str $ (я) + ":" + т $% и распечатать его
Затем я
```

Этот метод особенно полезен при создании и использовании списков структурированных данных. Например, следующий список содержит три поля данных для каждого человека (имя, адрес и телефон), в последовательном порядке. Обратите особое внимание на то, как для / Следующая / Шаг цикл используется, чтобы выбрать блоки 3 последовательных блоков данных из списка:

```
list.create c, myusers
list.add myusers, "Джон Смит", "123 Тине пер. Forest Hills Нью-Джерси", "555-
1234" ~
    "Пол Томпсон", "234 Джорджтаун пл. Арахис роща А.Л.", "555-2345" ~
    "Джим Persee", "345 Соленья Пайк Orange Grove Флорида", "555-3456" ~
    "Джордж Джонс", "456 Topforge Суд Mountain Creek CO", "" ~
    "Тим Полсон", "", "555-5678"

печатать "Все пользователи:"
Распечатать ""

list.size myusers, размер

для я = 1 размер шага 3

  ! Выберите и распечатайте этот пункт на индекс текущего (раз в 3 поле, имя)

  list.get myusers, я, п $
  печать "Имя:" + п $
```

```
  ! Выберите и распечатайте этот пункт на следующей индекса (текущий индекс +
1, адрес)

  list.get myusers, я + 1, а $
  печать "Адрес:" + а $

  ! Выберите и распечатайте пункт 2 после текущего индекса (индекс + 2,
телефон)

  list.get myusers, я + 2, р $
  печать "Телефон:" + р $
  Распечатать ""

Затем я
```

Картина код выше *очень* полезно при создании приложений для управления данными. Маленькие списки данных, таких как один выше, может провести всевозможные полезные связанных блоков данных (телефонных книг, рецептов и т.д.). Вы можете группировать различные поля данных последовательно, и читать любой выбранный поле информации из любой записи, используя для / Next / Step петлю.

Вы также можете использовать для петель *изменять* отдельные поля данных в списке. В следующем примере проверяется *каждый телефонный номер поля* в списке myusers (каждый третий элемент в списке, начиная с третьего пункта Если любое поле телефон пустой, то изменения, которые поле "000-000-0000".:

```
для я = 3 до размера шага 3
    list.get myusers, я, м $
    если т $ = ""
        list.replace myusers, я, "000-000-0000"
    ENDIF
Затем я
```

8.3 Выбор элементов из списка - "Выбор" Function

РФО Основные имеет встроенную функцию "Выбрать", чтобы позволить пользователям легко выбрать из пунктов в списке. Она занимает 3 параметров:

1) *возвращение* параметр, который держит *порядковый номер* в *пункте выбранный пользователем*

2) Переменная ярлык, который ссылается на список

3) текст сообщения для отображения пользователю. Текстовое сообщение отображается на верхней части экране выбора, а также быстро выводится во всплывающем окне. Если текст сообщения пустым (строка пустые кавычки ""), то нет всплывающее сообщение не происходит:

```
list.create c, месяцев
list.add месяцев, "Январь", "Февраль", "Март", "Апрель", "Май", "Июнь" ~
    "Июль", "Август", "Сентябрь", "Октябрь", "ноябрь", "Декабрь"

! Разрешить пользователю выбрать элемент из списка:

выберите SelectedItem, месяцы, ""
```

```
! Выберите из и распечатать выбранный элемент:

list.get месяцев, SelectedItem, м $
печать м $
```

Выбрать, функция также может быть использована для выбора элементов из массива:

```
array.load месяцев $ [], "Январь", "Февраль", "Март", "Апрель", "Май" ~
  "Июнь", "Июль", "Август", "Сентябрь", "Октябрь", "ноябрь", "Декабрь"

выберите SelectedItem, месяцев $ [], ""
печать месяцев $ [SelectedItem]
```

Существует дополнительный четвертый параметр, который вы можете предоставить выберите функцию, которая определяет, если пользователь выбрал пункт с коротким щелчком или длительного удержания. Эта опция может быть полезна для выполнения *различных* вещей с выбранными данными, в зависимости от того, как пользователь делает выбор физической:

```
array.load месяцев $ [], "Январь", "Февраль", "Март", "Апрель", "Май" ~
  "Июнь", "Июль", "Август", "Сентябрь", "Октябрь", "ноябрь", "Декабрь"
м $ = "Щелкните элемент, чтобы распечатать, нажмите и удерживайте для
удаления"
выберите SelectedItem, месяца $ [], м $, d
печать месяцев $ [SelectedItem]

! Окончательный необязательный параметр в функции выбора определяет, является
ли
! пользователь нажал короткие или длинные состоится на выбранном пункте.
! 0 = короткая 1 = долго

Если D = 1, то печать "Пункт будет удален"
```

8.4 Выбор записей из структурированных данных Списки

Вы уже видели, как использовать для / Следующий / шаг петлю, чтобы выбрать определенные поля элементов из списка структурированных данных (в следующем случае, каждые 3 пунктов из списка находятся имя, адрес и номер телефона):

```
list.create c, myusers
list.add myusers, "Джон Смит", "123 Тине пер. Forest Hills Нью-Джерси", "555-
1234" ~
    "Пол Томпсон", "234 Джорджтаун пл. Арахис роща А.Л.", "555-2345" ~
    "Джим Persee", "345 Соленья Пайк Orange Grove Флорида", "555-3456" ~
    "Джордж Джонс", "456 Topforge Суд Mountain Creek CO", "" ~
    "Тим Полсон", "", "555-5678"
list.size myusers, размер
для я = 1 размер шага 3
  list.get myusers, я, п $
  печать "Имя:" + п $
Затем я
```

Вы можете использовать эту технику, чтобы позволить пользователям выбирать из определенного *поля* в записи (т.е., в данном случае, либо имя, адрес или номер телефона пользователя,), а затем сделать *что-то полезное* с выбранными данными. В этом примере, пользователь может выбрать имя из данных пользователя, а затем имя, адрес и номер телефона всего за отображается, что пользователь:

```
list.create c, myusers
list.add myusers, "Джон Смит", "123 Тине пер. Forest Hills Нью-Джерси", "555-
1234" ~
    "Пол Томпсон", "234 Джорджтаун пл. Арахис роща А.Л.", "555-2345" ~
    "Джим Persee", "345 Соленья Пайк Orange Grove Флорида", "555-3456" ~
    "Джордж Джонс", "456 Topforge Суд Mountain Creek CO", "" ~
    "Тим Полсон", "", "555-5678"
list.size myusers, размер

! Создать новый список, содержащий только имена (используя цикл для
! выбрать каждый третий пункт в списке выше):

list.create c, списке имен
для я = 1 размер шага 3
  list.get myusers, я, п $
  list.add списке имен, п $
Затем я

! Добавьте секцию кода, так что мы можем перейти позже:

selectname:

! Разрешить пользователю выбрать имя из списка имен нового:

выберите S, списке имен, "Нажмите и удерживайте любую Имя для завершения
программы", г
Если D = 1, то в конечном

! Распечатать выбранную запись:

CLS% очистить экран
list.get myusers, S, N $% выбранный элемент (имя)
печать "Имя:"
печать N $
list.get myusers, S + 1, a $% следующий пункт, в порядке (адрес после имени)
печать "Адрес:"
напечатать $
list.get myusers, S + 2, p $% следующий пункт, в порядке (телефон после
обращения)
печатать "Телефон:"
печать p $

! Следующий цикл просто ждет, пока пользователь нажать клавишу,
! затем возвращается к этикетке выше:

делать
    INKEY $ K $
    если ((K $ <> "@") и (K $ <> "ключ 4")), то пусть делается = 1
не до готовности
перейти selectname
```

Познакомьтесь с выше пример хорошо. Эксперимент с ним и совершить его в памяти. Вы найдете общую концепцию и кодовую комбинацию, очень полезный во многих ситуациях. Вы можете использовать его для создания общих типов приложений поисковых данных: коллекции рецептов, адресные книги, информационные списки инвентаризации и т.д.

8.5 Сохранение списков структурированных данных в носителе - сериализации

Один из самых важных вещей, которые вычислительные устройства позволяют пользователям сделать, это сохранять и извлекать данные из локальных и удаленных средах хранения. Чтобы сохранить списки текстовых данных в РФО основной, полезной техникой является "сериализации" в информации, как длинный кусок текста, с каждого пункта в списке разделенных заданной текстовой характер. Чтобы сделать это, используйте цикл для цикла через каждый пункт в списке, и объединить каждый из пунктов вместе с указанным характера между ними каждого элемента в списке. Затем сохраните объединенных текст в файл:

```
list.create c, myusers
list.add myusers, "Джон Смит", "123 Тине пер. Forest Hills Нью-Джерси", "555-
1234" ~
    "Пол Томпсон", "234 Джорджтаун пл. Арахис роща А.Л.", "555-2345" ~
    "Джим Persee", "345 Соленья Пайк Orange Grove Флорида", "555-3456" ~
    "Джордж Джонс", "456 Topforge Суд Mountain Creek CO", "" ~
    "Тим Полсон", "", "555-5678"
list.size myusers, размер

! Соединить каждый элемент в "myusers" список вместе в один длинный
! Строка, с каждым элементом разделены 3 символов решетки (#):

сейв $ = ""% создать новую строковую переменную

для I = 1 до размера% проходного каждого элемента
    list.get myusers, я, м $

    ! Добавить 3 фунт символы после каждого пункта в списке, за исключением
последнего
    ! пункт:

    Если я = размер
        сейв $ = $ + сейв м $% конкатенация, если последний пункт
    другой
        сейв $ = $ + сейв м $ + "###"% объединять все другие пункт
    ENDIF
Затем я

! Все, что сцепляются текст существует в настоящее время в строке с надписью
"сейв $".
! Теперь просто написать эту строку в файл:

text.open ж, MYFILE, "myusers.txt"
text.writeln MYFILE, сейв $
text.close MYFILE
```

Чтобы извлечь данные и преобразовать его обратно в список структуре РФО Basic, выполните эту модель:

```
! Во-первых, прочитайте сохраненный текстовый файл:

text.open г, MYFILE, "myusers.txt"
text.readln MYFILE, serialdata $
text.close MYFILE

! Используйте функцию "Split", чтобы отделить каждый сохраненную запись от
! загруженный текст. Характер мы использовали, чтобы отделить эти элементы
были
```

```
! фунт символы (###). Функция Split будет отделить друг
! пункт на тех указанных символов фунт, и создать массив
! проведение каждого из разделенных элементов:

сплит myusers $ [], serialdata $ ", ###"

! Наконец, преобразования матричной структуры в списке. Чтобы сделать это,
! создать новый список и добавить элементы из массива, созданного выше,
! с помощью функции list.add.array (в основном это преобразует
! Массив в список, так что изменение размеров способность списке
! Структура может быть использован):

list.create c, myusers
list.add.array myusers $ []

! Теперь вы можете использовать этот список как обычно, цикл через него,
печать выбрано
! предметы, редактировать указанные элементы и т.д .:

list.size myusers, размер
для я = 1 размера
    list.get myusers, я, м $
    печать м $
Затем я
```

Сериализация данных гарантирует, что даже многострочного текста в списках отделена правильно и что каждая запись хранится в виде отдельного элемента в монолитном кусок текста. Просто убедитесь, что символ (ы) вы используете, чтобы отделить каждый элемент данных не будет не появляются в ваших пользовательских данных. Например, если ваши поля пользовательских данных выше, либо содержат 3 фунт символы в примере, функция Split бы incorrecly разделить сохраненный текст в этой точке, как отдельные записи. РФО Основные позволяет расщепление символ (ы) быть сколь угодно сложным. Вы могли бы, например, использовать "r% L3 # A @ q_x;!" как разбить строку, которая практически не имеет вероятность происходящих естественно в тексте пользователя.

ПРИМЕЧАНИЕ: Функция сплит самом деле принимает регулярное выражение "" как разбить строку. Избегайте использования одиночных символов () [] {} |. $ ^ +? чтобы отделить данные, потому что эти символы имеют особое значение при создании соответствие модели.Регулярные выражения являются мощным инструментом для поиска текстовых шаблонов. Поиск "регулярное выражение учебник" в Google, чтобы узнать, как они работают.

8.6 Отправка данных в последовательную форму на веб-сервер

Сериализация позволяет передавать списки данных с другими вычислительными платформами и средами хранения, таких как веб-серверы, а*одиноких текстовых* файлов. Вы можете использовать RFO Basic в FTP-команд, чтобы сохранить последовательную форму данные на удаленный сервер, например:

```
list.create c, myusers
list.add myusers, "Джон Смит", "123 Тине пер. Forest Hills Нью-Джерси", "555-
1234" ~
    "Пол Томпсон", "234 Джорджтаун пл. Арахис роща А.Л.", "555-2345" ~
    "Джим Persee", "345 Соленья Пайк Orange Grove Флорида", "555-3456" ~
    "Джордж Джонс", "456 Topforge Суд Mountain Creek CO", "" ~
    "Тим Полсон", "", "555-5678"
list.size myusers, размер
сейв $ = ""
для я = 1 размера
```

```
    list.get myusers, я, м $
    сейв $ = $ + сейв м $ + "###"
Затем я
text.open ж, MYFILE, "myusers.txt"
text.writeln MYFILE, сейв $
text.close MYFILE

! Приведенный выше код в точности так же, как в предыдущем примере.
! Теперь просто загрузите преобразованный текстовый файл на сервер:

ftp.open "ftp.site.com", 21 ", имя пользователя", "пароль"
ftp.put "myusers.txt", "/public_html/myusers.txt"
ftp.close
```

Следующий код будет читать данные из файла выше загруженного на веб-сервере, и импортировать его в структуру списка на локальном Android устройства. Это обеспечивает очень простой способ обмена полезные списки данных между пользователями, подключенными через Интернет. Вы можете использовать его для резервного копирования данных до местных, передача данных на различных устройствах, и т.д .:

```
graburl serialdata $, "http://site.com/myusers.txt"
сплит myusers $ [], serialdata $ ", ###"
list.create c, myusers
list.add.array myusers $ []

! Готово. Теперь что-то делать со списком:

list.size myusers, размер
для я = 1 размера
    list.get myusers, я, м $
    печать м $
Затем я
```

8.7 Обмен данными между различными языками программирования и сред

Любая полезная современный язык программирования обеспечит функции разделить символьные текстовые файлы с разделителями в списки данных. Учебник по http://re-bol.com/rebol.html#section-9.3, например, объясняет, как отделить такие данные, используя REBOL язык. Вы могли бы использовать следующий код, например, чтобы сделать использование загружаемых данных в приложении веб-сервера REBOL CGI. Этот код читает данные из загруженных в предыдущем примере, и печатает его в веб-браузере пользователя, строка за строкой:

```
myusers: разобрать читать myusers.txt% "#"
Eогеасп пункт myusers [
    Пункт печати
    печать "<br>"
]
```

Научиться пользоваться другие языки программирования для управления данными становится легче, когда вы видите, что каждый язык имеет свой собственный путь данных переменной маркировки, используя функции для выполнения действий, оценивая условия, циклы через список структур, читать и писать файл ввода / вывода и т.д. . Синтаксис обычно разные, но методология для достижения аналогичных целей обычно включает в себя те же самые типы логики и структуры, сопоставимые кодирования. Данные создавать и / или процесс в BPO Basic как правило, может быть передано и использовать в других средах очень легко, и визы наоборот.

8.8 Использование функции выбора для создания меню

Вы можете использовать функцию выбора, чтобы принять выбор пользователем относительно программного управления. Это обеспечивает очень простой и легкий в использовании система меню:

```
array.load выбор $ [], "Вариант 1", "Вариант 2", "Вариант 3"
выберите c, Выбор $ [] ","
если выбор $ [S] = "Вариант 1"
    ! Код для запуска по выбору 1:
    всплывающее "Вы выбрали вариант 1", 0,0,1
ElseIf выбор $ [S] = "Вариант 2"
    ! Код для запуска по выбору 2:
    всплывающее "Вы выбрали вариант 2", 0,0,1
другой
    ! Код для запуска за последний выбор:
    всплывающее "Вы выбрали вариант 3", 0,0,1
ENDIF
```

Вы можете создать подменю, просто работаете в другой выберите функцию, *внутри* ПЧ / Then / Else ответы главном меню:

```
array.load Главное меню $ [], "Вариант 1", "Вариант 2", "Вариант 3"
array.load submenu1 $ [], "к югу от Option1-1", "Sub-Option1-2"
array.load submenu2 $ [], "к югу от Option2-1", "Sub-Option2-2"

выберите S, Биржа труда $ [] ","
если Биржа труда $ [S] = "Вариант 1"
    выберите S1, submenu1 $ [], ""
    если submenu1 $ [S1] = "Sub-Option1-1"
        всплывающее "Вы выбрали вариант 1, суб-вариант 1", 0,0,1
    другой
        всплывающее "Вы выбрали вариант 1, суб-вариант 2", 0,0,1
    ENDIF
ElseIf Главное меню $ [S] = "Вариант 2"
    выберите c2, submenu2 $ [], ""
    если submenu2 $ [S2] = "Sub-Option2-1"
        всплывающее "Вы выбрали вариант 2, суб-вариант 1", 0,0,1
    другой
        всплывающее "Вы выбрали вариант 2, суб-вариант 2", 0,0,1
    ENDIF
другой
    всплывающее "Вы выбрали вариант 3", 0,0,1
ENDIF
```

Вы можете использовать функцию выбора для простых Да / Нет / Отмена операций:

```
array.load выбор $ [], "Да", "Нет", "Отменить"
выберите c, выбор $ [], "Вы хотите продолжить?"
если выборы $ [S] = "Да"
    всплывающее окно "Запуск ...", 0,0,1
другой
    всплывающее окно "Отмена ...", 0,0,1
ENDIF
```

Обучение использовать функцию "Выбрать", чтобы принять выбор меню, наряду с «входом» и «функций» text.input приобретать текст, введенный пользователями очень полезно. Вместе, эти инструменты обеспечивают удивительно мощные и простые способы для управления потоком программы и получения пользовательского ввода данных. GUI (графический) окна и виджеты *являются* возможно в BPO Basic, но в большинстве случаев, в простых и специализированных типов приложений, которые вы создадите в Android среде, вы не будете часто нужно ничего, более сложный, чем меню и ввода текста коробки , Обучение для обеспечения ввода данных и управлять интерактивность, используя эти простые инструменты пойдет *долгий путь* к делает вас способным РФО Basic кодер.

9. Полный пример приложения - Обучение, как все части подходят друг к другу

Примеры в этом разделе демонстрируют, как РФО Основные код воедино, чтобы создать полные программы. Код прокомментирован, чтобы обеспечить линия за линией объяснения о том, как работает каждый элемент. Загружаемые приложения исполняемые (.apk файл) скриншоты этих примеров можно на

```
http://rfobasic.com/examples (в стадии строительства)
```

9.1 Тест по математике

Эта программа помогает студентам проверить свои математические навыки сложения. Это позволяет пользователю выбрать верхние пределы для каждого из 2-х номеров в математике тестовых вопросов, затем неоднократно отображает случайные присоединения вопросы с Бутом слагаемых в пределах 0 и в диапазоне лимита. "Формат $" и "заменить $" функции используются для отображения аккуратно отформатированные вопросы и общее количество правильных и неправильных ответов в подсчитаны.

```
! Во-первых, использовать функцию "вход", чтобы запросить верхние предельные
цифры с.
! Пользователь. Сохраните ответ в переменных "L1" и "L2". Заметить, что
! нет "$" символы в этих переменных, так как они предназначены
! провести численные значения:

вход "Высокий предел первого номера", L1
вход "Верхний предел второго числа», L2

! Эти ближайшие два переменные используются для подсчета количества правильно
и
! неправильные ответы, введенные пользователем. Количество для обоих
первоначально
! устанавливается в ноль:

с = 0
я = 0

! Добавьте эту точку в программе, чтобы перейти позже:

Начало:

! Далее, генерировать случайные числа 2. Функция RND () генерирует количество
! между 0 и 1. Вот, умноженной на верхнем ряду предельного (вступил
! ранее пользователем), а затем округляется до ближайшего целого числа,
! с помощью функции Round (). Обратите внимание, что вся "RND () * L1"
! Выражение рассматривается в качестве аргумента для функции ROUND ().
```

```
! Результаты хранятся в переменных "N1" и "п2":

    N1 = круглый (RND () * L1)
    п2 = круглый (RND () * L2)

! Потому что выше округляемой цифра представляет собой десятичное число, мы
должны преобразовать
! его в строку, чтобы быть объединены и отображаются пользователю в виде
! вопроса. Мы могли бы использовать функцию ул $ (), но, что бы отобразить
! каждый номер с десятичной точки и заднюю 0 (т.е. "7.0"). Делать
! дисплей немного лучше, мы будем использовать формат $ () функцию для
отображения
! до 5 символов, без каких-либо задней десятичных знаков. Спасти
! Результат в переменной n1 $ (обратите внимание на использование символа
"$", потому что
! эта переменная содержит строку):

    N1 $ = формат $ ("#####", N1)

! Обрезать любые конечные пробелы из второго числа, используя заменить $ ()
! функция. Сохранить результат в переменной $ n2:

    п2 = $ заменить $ (формат $ ("#####", N2), "", "")

! Соединить число выше, в отформатированный вопрос, и использование
! функция входа, чтобы запросить ответ от пользователя. Сохранить
пользователя
! ответ в переменной "ответ".

    вход n1 $ + "+" + N2 + $ "=", ответ

! Если ответ пользователя является правильным (в общей сложности N1 + N2),
добавьте 1 к
! кол-во правильных ответов (хранящихся в числовую переменную "с"), и
! отображать позитивный сигнал для пользователя. Если ответ неправильный,
! неправильно переменная счетчик, который содержит номер неправильно
! Ответы ("я"), а также отображать соответствующее сообщение:

    если ответ = N1 + N2
        C = C + 1
        всплывающее "Правильно!", 0, 0, 0
    другой
        я = я + 1
        всплывающее "Неверно", 0, 0, 0
    ENDIF

! Подождите 2 секунды для пользователя, чтобы увидеть сообщение:

    паузу 2000

! Перейти обратно к "начать" этикетки и сделать процедуру вопрос все опять:

перейти начало

! Если в любой момент кнопку назад щелчке duing операции ввода, A.H.
! возникает ошибка, и программа автоматически переходит на метку отмечены
! "OnError":

OnError:

! Когда происходит событие OnError, отобразить сообщение, что объединит
```

```
! Общее количество правильных и неправильных ответов (можно использовать
! Формат функции здесь $ (), чтобы отобразить эти номера без задней
! десятичные символов, при желании):

    всплывающее окно, ул $ (с) + "правильно", ул + $ (я) + "неправильно".,
0,0,0
```

Вот и вся программа, без комментариев:

```
вход "Высокий предел первого номера", L1
вход "Верхний предел второго числа», L2
с = 0
я = 0
Начало:
    N1 = круглый (RND () * L1)
    п2 = круглый (RND () * L2)
    N1 $ = формат $ ("#####", N1)
    п2 = $ заменить $ (формат $ ("#####", N2), "", "")
    вход n1 $ + "+" + N2 + $ "=", ответ
    если ответ = N1 + N2
        С = С + 1
        всплывающее "Правильно!", 0, 0, 0
    другой
        я = я + 1
        всплывающее "Неверно", 0, 0, 0
    ENDIF
    паузу 2000
перейти начало
OnError:
    всплывающее окно, ул $ (с) + "правильно", ул + $ (я) + "неправильно".,
0,0,0
```

9.2 Текстовый редактор приложения

Вот полный текст редактор приложение, которое позволяет выбрать файл с вашего Android карты памяти, отредактировать его, а затем сохранить его обратно в файловой системе, когда закончите.

```
! В следующем переменная флаг будет использоваться позже, чтобы определить,
если пользователь
! уже вступила в начальную папку:

Флаг = 0

! Эта метка отмечает точку в код, который может быть прыгнул вернуться позже:
Начало:

! После того как программа запускается один раз, переменная флаг получает
набор в 1 и
! startfolder переменная $ получает выбран пользователем. Если это первый
! раз через цикл программы (переменная "флаг" еще равен 0), то
! Отправной умолчанию папка должна быть установлена:

если флаг = 0, то startfolder $ = "/ SDCard / RFO основного / данные /"

! Эта линия предлагает пользователю для папки с папки по умолчанию
отображается.
```

```
! Если startfolder $ переменная установлена на предыдущем цикле через
! Программа, которая значение используется в качестве значения по умолчанию:

вход "Папка", папка $, $ startfolder

! Сохранить выбранную папку в startfolder переменной $, и установить флаг
! Переменная 1 (чтобы указать, что отправной папки теперь были выбраны
! Пользователь):

startfolder $ = $ папка
Флаг = 1

! Помните, папки данных по умолчанию в BPO Basic является
! / SDCard / RFO основного / данные /
! Символы "../" используются, чтобы подняться на уровень выше. Так,
! Например, "../sdcard/rfo-basic/data/" = "/ SDCard / RFO основного /", и
! "../../../sdcard/rfo-basic/data/ то же самое, как верхнего уровня (корень)
! директории вашего Android файловой системы. Для того, чтобы сослаться на
папку
! выбирается пользователем, мы должны обратиться к этой папке по
относительных
! в папку RFO Basic по умолчанию. Эта линия объединяет строку
! должным образом относятся к относительного расположения выбранной папке:

rfofolder $ = "../../../" + папка $

! Эта линия удаляет массив "файлы $ []". Если массив не удаляется,
! следующая строка кода приведет к ошибке, когда петли программы
! вернуться к началу (помните, массивы не могут быть повторно размеры без
! удаления):

array.delete файлы $ []

! Эта линия читает список каталогов выбранной папке, и сохраняет
! что перечисление в переменной массива "файлы $ []":

file.dir rfofolder $, $ файлы []

! Эта линия сортирует массив по алфавиту:

Array.sort файлы $ []

! Эта линия позволяет пользователю выбрать имя файла из отсортированного
файла в
! Список:

выберите S, файлы $ [], ""

! Эта линия объединяет выбранную папку вместе с выбранным
! подать, чтобы создать полный путь к нужному файлу:

MYFILE $ = $ + rfofolder файлы $ [S]

! Эта линия читает содержимое файла и сохраняет текст в чтение
! строка переменной "$ неотредактированный":

grabfile неизмененном $, $ MyFile

! Эта линия позволяет пользователю редактировать текст для чтения, и
сохраняет отредактированный
! Содержание в строковой переменной "$ отредактировал":
```

```
text.input отредактировал $, неотредактированный $

! Эта линия открывает текстовый файл для записи, и создает указатель
! Переменная "Имя файла", чтобы обратиться к файлу:

text.open ж, имя файла, MyFile $

! Эта линия записывает содержимое из "редактировать $" переменную в файл:

text.writeln файла, редактировать $

! Эта линия закрывает файл:

text.close файла

! Эта линия предупреждает пользователя с коротким сообщением:

всплывающее окно "Сохраненные", 0,0,1

! Эта линия возвращается к этикетке в начале кода, и
! начинается весь процесс снова, чтобы выбрать и отредактировать другой файл:

перейти начало
```

Вот вся программа, без комментариев - это коротко и просто:

```
Флаг = 0
Начало:
    если флаг = 0, то startfolder $ = "/ SDCard / RFO основного / данные /"
    вход "Папка", папка $, $ startfolder
    startfolder $ = $ папка
    Флаг = 1
    rfofolder $ = "../../../" + папка $
    array.delete файлы $ []
    file.dir rfofolder $, $ файлы []
    Array.sort файлы $ []
    выберите S, файлы $ [], ""
    MYFILE $ = $ + rfofolder файлы $ [S]
    grabfile неизмененном $, $ MyFile
    text.input отредактировал $, неотредактированный $
    text.open ж, имя файла, MyFile $
    text.writeln файла, редактировать $
    text.close файла
    всплывающее окно "Сохраненные", 0,0,1
перейти начало
```

9.3 Веб-страница редактор

Эта программа представляет собой вариации на простой текстовый редактор, который выше позволяет редактировать HTML файлы (или любые другие текстовые файлы) на веб-сервере:

```
! Установите переменную флаг, чтобы определить, если пользователь вошел FTP
данные входа:

Флаг = 0
```

```
! Отметьте точку в коде, чтобы повторить вернуться позже:

Начало:

! Если это первый запуск программный цикл, установить некоторые умолчанию
! FTP значения:

если флаг = 0
    начальная страница $ = "ftp.site.com"
    startfolder $ = "/ public_html /"
    startusername $ = "пользователь"
    startpassword $ = "пройти"
ENDIF

! Запрос стоимость FTP, используя либо значения по умолчанию выше, или
значения
! введенный пользователем на предыдущей петли, с помощью кода:

вход "FTP-URL", на сайте $, начальная страница $
вход "Путь к файлу", папка $, $ startfolder
вход "Имя пользователя", имя пользователя $, $ startusername
вход "Пароль", пароль $, $ startpassword

! Установите переменную флаг, чтобы указать, что значения FTP были введены
! пользователь, и сохранить эти значения для последующего:

Флаг = 1
начальная страница $ = $ сайт
startfolder $ = $ папка
startusername $ = $ имени пользователя
startpassword $ = $ пароль

! Откройте соединение по FTP и загрузите имена файлов в
! выбранный каталог:

ftp.open сайт $ 21, $ имя пользователя, пароль $
ftp.dir список файлов

! Добавить выбор в список имен файлов, чтобы создать новый файл:

list.add список файлов, "Новый файл ..."

! Разрешить пользователю выбрать файл:

выберите INDX, список файлов, "Выберите файл:"
list.get список файлов, INDX, файл $

! Если пользователь выбрал "Новый файл ...", чтобы они могли ввести имя из
! Новый файл будет создан на сервере, а затем опорожнить содержимое из
! локальный файл температуры. Если пользователь выбрал имя существующего
файла,
! сохранить содержимое этого файла на локальный временный файл ("temp.txt"):

если файл $ = "Новый файл ..."
    вход "Имя файла:", файл $ ", file.html"
    Text.open ж, Временный файл, "temp.txt"
    Text.writeln временный файл ","
    Text.close временный файл
другой
    ftp.get файл $ ", temp.txt"
```

```
ENDIF

! Прочитать содержимое временного файла в переменную "неотредактированной":

grabfile неотредактированный $, "temp.txt"

! Разрешить пользователю редактировать текст выше, и сохранить
отредактированный текст в
! Переменная "$ отредактировал":

text.input отредактировал $, неотредактированный $

! Сохраните отредактированный текст временный файл:

text.open ж, имя файла, "temp.txt"
text.writeln файла, редактировать $
text.close файла

! Загрузка на содержание временного файла обратно на веб-сервер, и
! предупреждает пользователя, когда делается:

ftp.put "temp.txt", файл $
ftp.close
всплывающее окно "Сохраненные", 0,0,1

! Запустите программу снова весь:

перейти начало
```

Вот и вся программа, без комментариев:

```
Флаг = 0
Начало:
    если флаг = 0
        начальная страница $ = "ftp.site.com"
        startfolder $ = "/ public_html /"
        startusername $ = "пользователь"
        startpassword $ = "пройти"
    ENDIF
    вход "FTP-URL", на сайте $, начальная страница $
    вход "Путь к файлу", папка $, $ startfolder
    вход "Имя пользователя", имя пользователя $, $ startusername
    вход "Пароль", пароль $, $ startpassword
    Флаг = 1
    начальная страница $ = $ сайт
    startfolder $ = $ папка
    startusername $ = $ имени пользователя
    startpassword $ = $ пароль
    ftp.open сайт $ 21, $ имя пользователя, пароль $
    ftp.dir список файлов
    list.add список файлов, "Новый файл ..."
    выберите INDX, список файлов, "Выберите файл:"
    list.get список файлов, INDX, файл $
    если файл $ = "Новый файл ..."
        вход "Имя файла:", файл $ ", file.html"
        Text.open ж, Временный файл, "temp.txt"
        Text.writeln временный файл ","
        Text.close временный файл
    другой
```

```
        ftp.get файл $ ", temp.txt"
    ENDIF
    grabfile неотредактированный $, "temp.txt"
    text.input отредактировал $, неотредактированный $
    text.open ж, имя файла, "temp.txt"
    text.writeln файла, редактировать $
    text.close файла
    ftp.put "temp.txt", файл $
    ftp.close
    всплывающее окно "Сохраненные", 0,0,1
перейти начало
```

9.4 Интернет Чат номер

Этот пример является чат приложение, которое позволяет коллективный группа пользователей отправлять мгновенные текстовые сообщения и обратно через Интернет. Чат "комнаты" создаются динамически создания, чтения, добавления и сохранения текстовых файлов с помощью FTP (использовать программу, вы должны иметь доступ к свободному FTP сервера:. FTP-адрес, имя пользователя и пароль Ничто не должно быть сконфигурирован на сервере).

```
! Во-первых, спросить пользователя для URL в качестве FTP-сервера, который
будет содержать текст
! беседы чата:

вход "FTP-URL", на сайте $, "ftp.site.com" URL% по умолчанию

! Далее, получить путь к папке / директории, который будет содержать текст
! файл, содержащий разговор:

вход "Папка", папка $, "./public_html/" папка по умолчанию%

! Далее, получить имя имя текстового файла, который будет содержать текст
чата:

вход "Файл", файл $ ", chat.txt" Файл% по умолчанию

! Далее, получить имя пользователя FTP:

вход "Имя пользователя", имя пользователя $, "пользователь"

! Получить пароль FTP:

вход "Пароль", пароль $, "пройти"

! Получить имя экрана пользователя:

вход "Ваш экран Имя", имя $, "Имя"

! Откройте соединение по FTP:

ftp.open сайт $ 21, $ имя пользователя, пароль $

! Изменить каталог на нужной папке:

ftp.cd папка $

! Эта часть кода проверяет, является ли файл $ текстовый файл уже существует
```

```
! на сервере. Во-первых, установить переменную флаг, чтобы указать, что по
умолчанию,
! это предполагается, что файл не существует:

= 0 существует

! Теперь получить список файлов в текущей папке FTP:

ftp.dir файлы

! Цикл через файлы в списке папок чтобы увидеть, если любой из существующих
! Имена файлов соответствуют имени файла, определенные выше. Если совпадение
найдено, изменение
! "существует" переменная флаг равным 1:

list.size файлы, S
для я = 1 сек
    list.get файлы, я, стр $
    если файл $ = $, то страница существует = 1
Затем я

! Если файл не существует (то есть, "существует" переменная флаг не был
! изменился с 0 на 1), а затем создать новый, пустой текстовый файл и
передачи
! ее на сервер:

если существует = 0
    console.save "temp.txt"% создавать новые пустой текстовый файл
    ftp.put "temp.txt", файл $% передача ее на сервер
    всплывающее окно "Новый файл создается", 0, 0, 0
ENDIF

! Теперь прочитайте текущее содержимое онлайн текстовый чат, и кроме того,
что
! Текст в файл "chat.txt" на локальном устройстве Android:

ftp.get файл $ ", chat.txt"

! Следующая добавить текст к загруженному текстового файла выше, указывая,
что
! пользователь вошел в чат ("_Name_ вошел в комнату»). Открытие
! текстовый файл с "a" вариант добавляет к существующему тексту в файле
! (в отличие от стирания, что уже есть). Использую конкатенацию, то
! текст, записанный в файле "chat.txt" является комбинированная значение тока
! Дата и время, имя пользователя, статический текст, и возврат каретки.
! Обратите внимание на использование символа подчеркивания, чтобы продолжить
конкатенацию
! на второй линии:

text.open a, chattext, "chat.txt"
Время у $, $ м, d $, $ ч, п $, с $
text.writeln chattext, м $ + "-" + D $ + "," + H $ + ":" + M $ + ":" + ~
    Имя $ + "вошел в комнату."
text.close chattext

! Теперь программа использует вечно повторяющуюся Do / While Loop постоянно
! ждать ввода пользователя, и сделать соответствующие для вещей с этого
входа.
! после "флаг $" переменная устанавливается в "продолжить". Эта переменная
будет
```

```
! использовать позже, чтобы определить, если пользователь хочет, чтобы
завершить программу:

Флаг $ = "продолжить"

делать

    ! Очистите экран:

    CLS

    ! Отображение приветствие и некоторые указания. Обратите внимание на
использование "\ N"
    ! печатать возврат каретки (новая линия), и символ подчеркивания
    ! (_), Чтобы продолжить конкатенацию на многочисленных линиях:

    печать "---------------------------------- \ п" + ~
        "Вы вошли в систему как:" + имя + $ "\ п" + ~
        "Тип" комнаты ", чтобы перейти в чатах. \ П" + ~
        "Тип" бросить ", чтобы завершить чат. \ П" + ~
        "Введите текст пустой (просто нажмите ОК) \ п" + ~
        "периодически обновлять экран. \ п" + ~
        "----------------------------------"

    ! Читайте обновляются сообщения, которые в настоящее время в "chat.txt"
текст
    ! подать на FTP-сервер, назначить переменную слово currentChat $ в
    ! что текст и распечатать содержимое:

    ftp.get файл $ ", chat.txt"
    grabfile currentChat $, "chat.txt"
    печать "Вот текущий текст чата на:" + $ файла
    печать currentChat $

    ! Пауза несколько секунд, чтобы позволить пользователю, чтобы прочитать
чат:

    паузу 5000

    ! Получить какой-то текст будет добавлен в чат, введенного пользователем.
    ! Сохранить введенный текст в строковой переменной "$ enteredText":

    вход "Вы говорите:", $ enteredText

    ! Оператор If / Elseif / остальное структура ниже используется для
проверки команд в
    ! текст, введенный пользователем. Если пользователь вводит "бросить",
пустой
    ! текст или какой-либо другой текст, который будет добавлен к разговору,
    ! Соответствующие действия происходят:

    если enteredText $ = "бросить"

        ! Если пользователь вводит "бросить", остановить навсегда петлю
(которая выйдет
        ! программа). Вы можете поочередно использовать "d_u.break"
        ! функционировать здесь:

        Флаг $ = "бросить"

    ElseIf enteredText $ <> ""
```

```
        ! Если пользователь вошел текст (ничего, кроме пустой текст),
        ! Соединить текст с именем пользователя и статический текст
        ! "Говорит:". Магазин, который объединяется текст в строковой
переменной
        ! "sentMessage $":

        sentMessage $ = $ + название "говорит:" + $ enteredText

        ! Теперь скачать текущий текст чата снова (чтобы убедиться, что вы
        ! работает с последнего обновления (ов)), добавьте сообщение
        ! выше него, а затем загрузить прилагаемой текст до FTP
        ! Сервер:

        ftp.get файл $ ", chat.txt"
        text.open a, chattext, "chat.txt"% помните, открытым для
присоединения
        text.writeln chattext, sentMessage $
        text.close chattext
        ftp.put "chat.txt", файл $

    ENDIF

    ! Единственный возможный вариант в этой точке, что пользователю
    ! не поступил ничего (пусто текст, ""). В этом случае, цикл просто
    ! повторяется, поэтому нынешний текст чата снова загружены, дисплей
    ! обновляется, пользовательский ввод просил, в IF / Then / Else условия
    ! оценивается снова, и т.д. Это продолжается, пока пользователь вводит
"не бросить".

до флага $ = "не бросить"

! Когда навсегда повторяя контур вышел, добавьте Последнее послание
! текст чата, закрыть FTP соедините, и завершить программу:

CLS
печатать "Прощай!"
ftp.get файл $ ", chat.txt"
text.open a, chattext, "chat.txt"% помните, открытым для присоединения
text.writeln chattext, ч $ + ":" + M $ + ":" + имя + $ ~
    " вышел из комнаты."
text.close chattext
ftp.put "chat.txt", файл $
ftp.close
паузу 1000
конец
```

Навсегда повторяя контур Структура используемая в данной программе является очень распространенным наброски для приложений, которые неоднократно принимают от пользователей ввода, обработки входящих данных и вывод результатов выходных данных. Это особенно полезно в играх, которые постоянно перемещаются списки графических элементов вокруг экрана (подробнее об этом позже в этом уроке).

Вот вся программа, без комментариев:

```
вход "FTP-URL", на сайте $, "ftp.site.com"
вход "Папка", папка $, "./public_html/"
вход "Файл", файл $ ", chat.txt"
```

```
вход "Имя пользователя", имя пользователя $, "пользователь"
вход "Пароль", пароль $, "пройти"
вход "Ваш экран Имя", имя $, "Имя"
ftp.open сайт $ 21, $ имя пользователя, пароль $
ftp.cd папка $
= 0 существует
ftp.dir файлы
list.size файлы, S
для я = 1 сек
    list.get файлы, я, стр $
    если файл $ = $, то страница существует = 1
Затем я
если существует = 0
    console.save "temp.txt"
    ftp.put "temp.txt", файл $
    всплывающее окно "Новый файл создается", 0, 0, 0
ENDIF
ftp.get файл $ ", chat.txt"
text.open a, chattext, "chat.txt"
Время у $, $ м, d $, $ ч, п $, с $
text.writeln chattext, м $ + "-" + D $ + "," + H $ + ":" + M $ + ":" + ~
    Имя $ + "вошел в комнату."
text.close chattext
Флаг $ = "продолжить"
делать
    CLS
    печать "---------------------------------- \ п" + ~
        "Вы вошли в систему как:" + имя + $ "\ п" + ~
        "Тип" комнаты ", чтобы перейти в чатах. \ П" + ~
        "Тип" бросить ", чтобы завершить чат. \ П" + ~
        "Введите текст пустой (просто нажмите ОК) \ п" + ~
        "периодически обновлять экран. \ п" + ~
        "----------------------------------"
    ftp.get файл $ ", chat.txt"
    grabfile currentChat $, "chat.txt"
    печать "Вот текущий текст чата на:" + $ файла
    печать currentChat $
    паузу 5000
    вход "Вы говорите:", $ enteredText
    если enteredText $ = "бросить"
        Флаг $ = "бросить"
    ElseIf enteredText $ <> ""
        sentMessage $ = $ + название "говорит:" + $ enteredText
        ftp.get файл $ ", chat.txt"
        text.open a, chattext, "chat.txt"% помните, открытым для
присоединения
        text.writeln chattext, sentMessage $
        text.close chattext
        ftp.put "chat.txt", файл $
    ENDIF
до флага $ = "не бросить"
CLS
печатать "Прощай!"
ftp.get файл $ ", chat.txt"
text.open a, chattext, "chat.txt"% помните, открытым для присоединения
text.writeln chattext, ч $ + ":" + M $ + ":" + имя + $ ~
    " вышел из комнаты."
text.close chattext
ftp.put "chat.txt", файл $
ftp.close
паузу 1000
```

```
конец
```

9.5 Blogger

Эта программа является еще одним примером FTP, что позволяет пользователям создавать и добавлять записи в интернет-страницы блога.Пользователь вводит название, текст блога, и ежедневное ссылку URL интересов. Программа определяет текущую дату и время. Все эти куски текста объединяются вместе, вместе с необходимой HTML код макета, и добавляется к загруженную копию предыдущего содержания блога.Этот код затем загружены обратно на веб-сервер пользователя, чтобы рассматривать публично. Будьте уверены, чтобы изменить URL переменные FTP и HTML, так они представляют актуальную информацию учетной записи пользователя.

```
! Храните FTP-адрес учетной записи, папки / файла, имя пользователя и пароль
! в переменных (для легкой конфигурации):

вход "FTP-URL", на сайте $, "ftpsite.com"
вход "Папка", папка $, "./public_html/"
вход "Супер", стр $, "blog.html"
вход "Имя пользователя", имя пользователя $, "пользователь"
вход "Пароль", пароль $, "пройти"
вход "URL HTTP", гиперссылка $, "http://site.com/blog.html"

! Эта часть кода проверяет, если страница $ файл уже существует
! на сервере:

ftp.open сайт $ 21, $ имя пользователя, пароль $
ftp.cd папка $
= 0 существует
ftp.dir файлы
list.size файлы, S
для я = 1 сек
    list.get файлы, я, файл $
    если файл $ = $, то страница существует = 1
Затем я

! Если файл не существует, создайте его:

если существует = 0
    console.save "temp.txt"
    ftp.put "temp.txt", стр $
    всплывающее окно "Новый файл создается", 0, 0, 0
ENDIF

! Получить титул блога, URL, и некоторые блоге текст от пользователя.
Магазин, что
! Текст в переменных blogtitle $ и $ blogtext:

вход "Название блога:", $ blogtitle, "Название"
вход "URL:", $ blogurl, "http://site.com"
text.input blogtext $, "Выезд сегодняшний ссылку!"

! Получить текущую дату и время:

Время у $, $ м, d $, $ ч, п $, с $

! Соединить текст, введенный выше, и дата / время, с HTML
! код, необходимый для отображения запись в блоге. Сохранить построенную HTML
```

```
! в переменной $ newblog. Обратите внимание на использование подчеркивания
! характер продолжить конкатенации на несколько строк:

newblog $ = "<h1>" + blogtitle $ + "</ h1>" + "Запись в блоге создан:" + ~
    м $ + "-" + D $ + "-" + у + $ "& NBSP; & NBSP;" + H $ + ":" + п $ + ~
    ":" + S $ + "на <br> <a href=\"" + blogurl$ + "\">" + blogurl $ + ~
    "</a> поиска по" + "<центр> <ширина таблицы = 80%> <TR> <TD> <PRE>
<STRONG>" + ~
    blogtext $ + "</ STRONG> </ PRE> </ TD> </ TR> </ TABLE> </ центр> <br>
<ч>"

! Скачать полный существующий HTML блог. Сохраните его в файл temp.txt:

ftp.get страница $, "temp.txt"

! Загрузите скачанный блоге текст в переменной $ previousblog:

grabfile previousblog $, "temp.txt"

! Написать объединенных новые и старые дневник текст в файл temp.txt:

text.open ж, имя файла, "temp.txt"
text.writeln файла, newblog $ + $ previousblog
text.close файла

! Загрузить обновленный блог файл на веб-сервере:

ftp.put "temp.txt", стр $
ftp.close

! Оповещение пользователя с сообщением, а затем открыть блог в браузере:

всплывающее окно "Сохраненные", 0, 0, 1
Просмотр гиперссылка $
```

Вот полная программа без комментариев:

```
вход "FTP-URL", на сайте $, "ftpsite.com"
вход "Папка", папка $, "./public_html/"
вход "Супер", стр $, "blog.html"
вход "Имя пользователя", имя пользователя $, "пользователь"
вход "Пароль", пароль $, "пройти"
вход "URL HTTP", гиперссылка $, "http://site.com/blog.html"
ftp.open сайт $ 21, $ имя пользователя, пароль $
ftp.cd папка $
= 0 существует
ftp.dir файлы
list.size файлы, S
для я = 1 сек
    list.get файлы, я, файл $
    если файл $ = $, то страница существует = 1
Затем я
если существует = 0
    console.save "temp.txt"
    ftp.put "temp.txt", стр $
    всплывающее окно "Новый файл создается", 0, 0, 0
ENDIF
вход "Название блога:", $ blogtitle, "Название"
вход "URL:", $ blogurl, "http://site.com"
```

```
text.input blogtext $, "Выезд сегодняшний ссылку!"
Время у $, $ м, d $, $ ч, п $, с $
newblog $ = "<h1>" + blogtitle $ + "</ h1>" + "Запись в блоге создан:" + ~
    м $ + "-" + D $ + "-" + у + $ "& NBSP; & NBSP;" + H $ + ":" + п $ + ~
    ":" + S $ + "на <br> <a href=\"" + blogurl$ + "\">" + blogurl $ + ~
    "</a> поиска по" + "<центр> <ширина таблицы = 80%> <TR> <TD> <PRE>
<STRONG>" + ~
    blogtext $ + "</ STRONG> </ PRE> </ TD> </ TR> </ TABLE> </ центр> <br>
<ч>"
ftp.get страница $, "temp.txt"
grabfile previousblog $, "temp.txt"
text.open ж, имя файла, "temp.txt"
text.writeln файла, newblog $ + $ previousblog
text.close файла
ftp.put "temp.txt", стр $
ftp.close
всплывающее окно "Сохраненные", 0, 0, 1
Просмотр гиперссылка $
```

9.6 Рецепт базы данных

Вот приложение базы данных рецепт, который позволяет создавать, редактировать, искать, хранить, извлекать и просматривать рецепты. Вы можете использовать эту программу в качестве контура для создания приложений, которые хранят какую-либо соответствующей информации.Вы будете использовать методы управления список, управление файлами, выбор меню, редактирование текста, сериализации, подпрограмм, и других, часто в своих программах РФО Basic, так изучать этот пример внимательно:

```
! Во-первых, проверьте, чтобы увидеть, если файл данных существует:

File.Exists б ", recipes.txt"

! Если файл не существует, создайте его:

если б = 0

! Создать новую структуру списка, а затем добавить некоторые названия по
умолчанию рецепт,
! ингредиенты и инструкции. Обратите внимание на использование символа тильды
! (~), Чтобы расширить содержание список по нескольким линиям. Текущий
! состояние list.add и array.load функций не позволяют строку
! конкатенации с помощью тильды в параметрах функции.
! Для того, чтобы добавить сцепленные строки в списке, кроме каскадный
! строки в переменной перед выполнением функции list.add (см
! http://rfobasic.freeforums.org/post6133.html?hilit=#p6133 для
! обсуждение этой темы):

    longstring $ = "Удалить блюдо из" + ~
    "упаковки \ nHeat в микроволновой печи \ nCool и едят вилкой"
    list.create с, newrecipes
    list.add newrecipes, "Овсянки", "Овсянки \ nWater \ nMaple Сироп" ~
    "Вскипятить воду \ Надда овсянка \ Надда кленовый сироп \ nCool и есть" ~
    "Замороженные Ужин", "замороженное блюдо \ nFork", longstring $

! Сериализация данных в одну строку, как показано ранее в
! руководство. Сериализованный данные объединяются с помощью строки
! separater "###", и сохраняется в переменной $ SAVEDATA:
```

```
    list.size newrecipes, размер
    сейв $ = ""
    для я = 1 размера
        list.get newrecipes, я, м $
        Если я = размер
            сейв $ = $ + сейв м $
        другой
            сейв $ = $ + сейв м $ + "###"
        ENDIF
    Затем я

! Сохранить сериализованных данных в "recipes.txt" текстовый файл, а затем
закрыть
! если условно структуры:

    text.open ж, MYFILE, "recipes.txt"
    text.writeln MYFILE, сейв $
    text.close MYFILE
ENDIF

! Загрузите сохраненный файл данных (новый, если только что создали выше, или
старый
! Один из них, если ранее сохраненные пользователем):

loadSavedRecipes:

    ! Читайте сериализованную строку:

    grabfile serialdata $, "recipes.txt"

    ! Создайте пустой массив:

    array.delete рецепт $ []

    ! Сплит строку на "###" символов:

    сплит рецепт $ [], serialdata $ ", ###"

    ! Преобразование массива в структуре списка:

    list.create c, рецепты
    list.add.array рецепты, рецепт $ []
! Создать отдельный список, содержащий только заголовки рецепт (каждый
! Третий пункт в списке):
list.size рецепты, размер
list.create c, названия
для я = 1 размер шага 3
  list.get рецепты, я, п $
  list.add названия, п $
Затем я

! Отображение название, ингредиенты и инструкции для одного рецепта,
выбранные
! пользователем:

viewRecipe:

! Добавить вариант в список названий рецепт, чтобы создать новый рецепт
! (только, если эта опция уже не было добавлено):

list.search названия, "Создать новый рецепт ...", c
```

```
если c = 0
    list.add названия, "Создать новый рецепт ..."
ENDIF

! Разрешить пользователю выбрать рецепт из списка названий:

выберите INDX, заголовки, "Выберите рецепт, чтобы просмотреть / редактировать
(удерживайте для выхода)", D

! Если пользователь длинных кликов, завершить программу:

Если D = 1, то в конечном

! Если пользователь выбирает, чтобы создать новый рецепт, прыгать в
соответствующую
! Подпрограмма:

list.get названия, INDX, RCP $
если RCP $ = "Создать новый рецепт ..."
    перейти createNewRecipe
ENDIF

! В противном случае, очистить экран и начать процедуру отображения рецепт:

CLS

! Обратите особое внимание на этой линии. Помните, что каждые 3 пунктов
! в структуре списка рецепта: название / ингредиенты / инструкции, так
! индекс выбранного заголовка будет первым каждый
! 3 шт. Если вы выбираете второй титул в списке название, это
! позиция в списке рецептов 2 * 3 - 2 (вторая группа из 3 предметов,
! 2 назад из индекса этого последнего пункта). Математически, которые могут
! быть представлены следующим образом:

S = INDX * 3 - 2

! Выберите из и распечатать выбранный заголовок из списка рецептов (найти на
! выше индекс):

list.get рецепты, S, N $
печать "НАЗВАНИЕ:"
печать N $ + "\ п"

! Выберите из и распечатать выбранные ingrediants из списка рецептов
! (найти на выше +1 индекса):

list.get рецепты, S + 1, $
распечатать »Состав:"
напечатать $ + "\ N"

! Выберите из и распечатать выбранные ingrediants из списка рецептов
! (найти на выше +2 индекса):

list.get рецепты, S + 2, p $
печать "Инструкции по приготовлению:"
печать p $
Распечатать ""

! Дайте Используйте опцию, чтобы нажать клавишу, чтобы продолжить (убедитесь,
! Экранная клавиатура показывает):
```

```
печать "(Нажмите любую клавишу для продолжения ...)"
kb.hide
паузу 1000
kb.toggle
паузу 1000
пусть делается = 0
делать
    INKEY $ K $
    если K $ <> "@", затем пусть делается = 1
не до готовности

! Когда процедура viewRecipe будет сделано, вернуться назад и сделать все это
снова:

перейти viewRecipe

! Вот подпрограмма, чтобы создать новый рецепт. Это в основном то же самое
! в качестве кода, используемого ранее для создания нового файла данных. Это
просто просит
! название, ингредиенты, и инструкции тексты от пользователя. Эти тексты
! присваиваются метки переменных, а затем добавил к существующему списку
рецептов
! состав. Список рецепт, то по частям в длинный объединяются
! строка и повторно сохранены в файле "recipes.txt". Когда эта процедура
является
! полная программа возвращается к подпрограмме loadSavedRecipes к
! начать снова:

createNewRecipe:
вход "Название:", название $ ","
text.input ингредиенты $ ", (тип ингредиенты здесь)"
Инструкции text.input $ ", (тип инструкции здесь)"
list.add рецепты, название $
list.add рецепты, ингредиенты $
list.add рецепты, инструкции $
list.size рецепты, размер
сейв $ = ""
для я = 1 размера
    list.get рецепты, я, м $
    Если я = размер
        сейв $ = $ + сейв м $
    другой
        сейв $ = $ + сейв м $ + "###"
    ENDIF
Затем я
text.open ж, MYFILE, "recipes.txt"
text.writeln MYFILE, сейв $
text.close MYFILE
перейти loadSavedRecipes
```

Вот код без комментариев:

```
File.Exists б ", recipes.txt"
если б = 0
    longstring $ = "Удалить блюдо из" + ~
    "упаковки \ nHeat в микроволновой печи \ nCool и едят вилкой"
    list.create c, newrecipes
    list.add newrecipes, "Овсянки", "Овсянки \ nWater \ nMaple Сироп" ~
    "Вскипятить воду \ Надда овсянка \ Надда кленовый сироп \ nCool и есть" ~
```

```
    "Замороженные Ужин", "замороженное блюдо \ nFork", longstring $
    list.size newrecipes, размер
    сейв $ = ""
    для я = 1 размера
        list.get newrecipes, я, м $
        Если я = размер
            сейв $ = $ + сейв м $
        другой
            сейв $ = $ + сейв м $ + "###"
        ENDIF
    Затем я
    text.open ж, MYFILE, "recipes.txt"
    text.writeln MYFILE, сейв $
    text.close MYFILE
ENDIF
loadSavedRecipes:
    grabfile serialdata $, "recipes.txt"
    array.delete рецепт $ []
    сплит рецепт $ [], serialdata $ ", ###"
    list.create c, рецепты
    list.add.array рецепты, рецепт $ []
    list.size рецепты, размер
    list.create c, названия
    для я = 1 размер шага 3
      list.get рецепты, я, п $
      list.add названия, п $
    Затем я
viewRecipe:
    list.search названия, "Создать новый рецепт ...", c
    если c = 0
        list.add названия, "Создать новый рецепт ..."
    ENDIF
    выберите INDX, заголовки, "Выберите рецепт, чтобы просмотреть /
редактировать", D
    Если D = 1, то в конечном
    list.get названия, INDX, RCP $
    если RCP $ = "Создать новый рецепт ..."
        перейти createNewRecipe
    ENDIF
    CLS
    S = INDX * 3 - 2
    list.get рецепты, S, N $
    печать "НАЗВАНИЕ:"
    печать N $ + "\ п"
    list.get рецепты, S + 1, $
    распечатать »Состав:"
    напечатать $ + "\ N"
    list.get рецепты, S + 2, р $
    печать "Инструкции по приготовлению:"
    печать р $
    Распечатать ""
    печать "(Нажмите любую клавишу для продолжения ...)"
    kb.hide
    паузу 1000
    kb.toggle
    паузу 1000
    пусть делается = 0
    делать
        INKEY $ K $
        если K $ <> "@", затем пусть делается = 1
    не до готовности
```

```
перейти viewRecipe
createNewRecipe:
    вход "Название:", название $ ","
    text.input ингредиенты $ ", (тип ингредиенты здесь)"
    Инструкции text.input $ ", (тип инструкции здесь)"
    list.add рецепты, название $
    list.add рецепты, ингредиенты $
    list.add рецепты, инструкции $
    list.size рецепты, размер
    сейв $ = ""
    для я = 1 размера
        list.get рецепты, я, м $
        Если я = размер
            сейв $ = $ + сейв м $
        другой
            сейв $ = $ + сейв м $ + "###"
        ENDIF
    Затем я
    text.open ж, MYFILE, "recipes.txt"
    text.writeln MYFILE, сейв $
    text.close MYFILE
перейти loadSavedRecipes
```

Это может показаться, как много кода для такой простой программы, но выше приложение демонстрирует некоторые основные закономерности абсолютно кода. Обучение думать в терминах переменных, список данных структур, петель и условных структур является основополагающим для достижения подавляющее большинство полезных целей программирования. Сохранение сериализованных данных в файл полезна, если вам нужно, чтобы поделиться с другими устройствами, загружать данные на веб-сервере, чтобы быть разобраны и введенные в базу данных, отображать его в Интернете, и т.д. В более простой способ, это приложение может быть легко преобразуется в магазин контактную информацию, розничной кадастровой информации, или любой другой вид личного или бизнес-информации, в основном путем переименования поля этикетки.Например, вместо "Название", "Ингредиенты", и "Способ приготовления", поля могут быть помечены "Элемент", "Размер", и "Описание", или "Имя", "Номер телефона (с), и "Адрес" и др Попробуйте изменить петли, индексные переменные, и список структурам немного, чтобы создать приложение, которое хранит более 3 поля. Далее, мы создадим GUI версию этой программы.

10. HTML / JavaScript ГПИ

Вы уже видели, как РФО Основные "вход" и "функции" text.input могут быть использованы для приобретения текст, введенный пользователями, функция "выберите" можно использовать для отображения и принять выбор меню, и "печать" может быть используется для вывода данных. Для простых типов коммунальные скриптов, которые РФО Основные идеально подходит для создания, эти функции ввода / вывода часто будут все, что когда-либо необходимости. Для создания более сложных экранов ввода данных, или строить более визуально привлекательных интерфейсов графического пользовательского ("GUI" S), РФО Основные позволяет использовать стандартный HTML и Javascript страницы для отображения кнопок, текстовых полей, выпадающих списках выбора, флажки, текстовые области, изображения и другие знакомые элементы формы. Вы можете использовать любой из распространенных библиотек Javascript, поддерживаемых Android-браузер (JQuery Mobile, Сенча Touch, jQTouch), чтобы обеспечить богатые графический интерфейс пользователя для взаимодействия РФО основных приложений.

10.1 HTML-Crash Course

Покрытие HTML и Javascript в глубине выходит за рамки данного руководства, но быстрый интенсивный курс для начинающих обеспечит достаточное понимание, чтобы быть очень полезным. Чтобы узнать больше, чем основной HTML, поиск Google для "Учебник HTML», «Мусор» учебник, и такие темы, как "мобильный рамках UI» - есть тысячи ресурсов для разработчиков с каждым уровнем опыта.

HTML является языком макет используется для форматирования текста и GUI элементов на всех веб-страниц. HTML не является языком программирования - это не может оценить условные выражения, он не имеет средство для управления телефоном аппаратных средств, таких как датчики, камеры и т.д., это *не может обрабатывать или манипулировать данными* в любом полезным способом, и т.д. Это просто формат разметки что позволяет формировать *внешний вид* текста, изображений и других предметов на просмотренных страниц в браузере. РФО Основные позволяет манипулировать данными и управления телефона оборудование, так что два являются идеальным дополнением друг с другом.

10.1.1 Метки

В HTML элементы на веб-странице заключены между "тегов" начальным и конечным:

```
<Начальный тег> Текст или графический элемент на веб-странице </ конечный тэг>
```

Есть теги для осуществления макет всячески. Выделение какой-то текст, например, окружают его в открытия и закрытия «сильные» теги:

```
<STRONG> некоторые полужирным текст </ STRONG>
```

Код выше появляется на веб-странице, как: **какой-то жирным** текстом.

Для курсивом текст, окружают его в <I> и <I /> теги:

```
<я> некоторые текст курсивом </ I>
```

Это, кажется, на веб-странице, как: *какой-то* курсивом.

Обратите внимание, что каждый

```
<открывающий тег>
```

в HTML код сопровождается соответствующий

```
</ закрытия тегов>
```

Закрывающий тег начинается с символа "/". Некоторые теги окружают все страницы, некоторые теги окружают части страницы, и они часто вложены друг в друга для создания более сложных конструкций.

Чтобы создать таблицу с тремя строками данных, выполните следующие действия:

```
<Таблица граница = 1>
    <TR> <TD> Первая строка </ TD> </ TR>
    <TR> <TD> Второй ряд </ TD> </ TR>
```

```
    <TR> <TD> третья строка </ TD> </ TR>
</ Table>
```

Приведенный выше код производит следующий результат:

Первая строка
Во-вторых ряд
Третий ряд

Обратите внимание, что вся таблица выше заключена в открытии и закрытии <TABLE> теги, а каждая строка заключена в открытии / закрытии <TR> (строки таблицы) и теги <TD> (табличные данные).

Минимальный формат для создания веб-страницы показан ниже. Обратите внимание, что название вложен между тегами "голова", а весь документ вложен внутри тегов "HTML". Содержание этой страницы видно пользователю окружен **"тела** тегов":

```
<HTML>
    <HEAD>
        <Title> Название страницы </ title>
    </ HEAD>
    <BODY>
        Связка текста и <я> HTML форматирования </ I> здесь идет ...
    </ BODY>
</ Html>
```

Если вы сохраните этот код в текстовый файл с именем "yourpage.html" в / SDCard / RFO основного / данные / на Android devive, и прибой подать: ///sdcard/rfo-basic/data/yourpage.html , вы увидите в вашем браузере страницу под названием "Заголовок страницы", с текстом "Связка текста и *HTML форматирования* здесь идет ...". Все веб-страницы работает таким образом - это обучающая программа, на самом деле просто HTML-документ, хранящийся на счете веб-сервера автора. Нажмите кнопку Просмотр -> Источник в вашем браузере, и вы увидите HTML-теги, которые были использованы для форматирования этого документа. Этот документ просто сохраняются на сервере, который является общедоступным в http://rfobasic.com URL.

10.2 Формы HTML - Основные GUI виджеты

В следующем HTML-пример содержит "форма" тег внутри стандартного HTML головы и тела макета. Внутри виде теги 2 ввода текста метки (полевые), многострочный текстовое поле тега, выбор выпадающий тег позволяет пользователю выбрать один день из 7 будние дни, 4 флажок теги, позволяющие пользователю выбрать любое количество фруктов и представить кнопку тег. В "метки", необходимых для производства возврат каретки (новые линии). Без них, каждый элемент будет течь слева направо по экрану, а затем обернуть вокруг, чтобы заполнить страницу браузера:

```
<HTML>
    <HEAD> <TITLE> Форма ввода данных </ название> </ HEAD>
    <тело>
        <форма действие = "форма">
            Имя: <br>
            <тип входного = "текст" имя = "имя"> Каталог <br>
            E-mail: <br>
            <тип входного = "текст" имя = "электронная почта"> Каталог <br>
            Сообщение: <br>
```

```
            <TextArea COLS = 25 NAME = "сообщение" строк = 5> Привет. </
TextArea> Каталог <br>
            Предпочтительный день: <br>
            <выберите имя = "день">
                <опция> понедельник <опция> вторник <опция> Среда
                <опция> Четверг <опция> Пятница <опция> суббота
                <опция> воскресенье </ вариант>
            </ выберите> Каталог <br>
            Любимый фрукт (ы): <br>
            <тип входного = значение "флажок" Имя "F1" = "рис" проверил> рис
<br>
            <тип входного = значение "флажок" Имя "F2" = "яблоко"> яблоко
<br>
            <тип входного = значение "флажок" Имя "F3" = "оранжевый">
оранжевый <br>
            <тип входного = "флажок" имя = значение "F4" = "банан"> банан
Каталог <br>
            <тип входного = "представить" имя = "представить" значение =
"представить">
        </ FORM>
    </ BODY>
</ HTML>
```

Скопируйте код, приведенный выше в пустой текстовый документ, сохраните его как "myform.html", а затем открыть этот документ с вашего браузера. Вы увидите следующее:

Имя:

Эл. адрес:

Сообщение:

Предпочтительный день:

Любимый фрукт (ы):

- [x] инжир
- [ ] яблоко
- [ ] оранжевый
- [ ] банан представить

Элементы форму выше достаточно, чтобы удовлетворить подавляющее большинство потребностей входных данных для простых программ. Для получения более подробной информации о тегах HTML форм, см http://www.w3schools.com/html/html_forms.asp.

10.3 Использование HTML-форм данных

Вы можете использовать данные, введенные в любом виде следующим образом (Использование этого примера, сохраните HTML форму выше, как /sdcard/rfo-basic/data/myform.html):

```
html.open
html.load.url "файл: ///sdcard/rfo-basic/data/myform.html"
делать
    html.get.datalink данные $
до $ данных <> ""
Данные $ = $ середине (данные $, 5)
печать "Форма данных:" + данные $
```

Все HTML-формы с помощью "&" характер сериализовать данные, представленные в один длинный каскадного строку. Вы можете использовать функцию "Split", чтобы создать массив, содержащий все элементы из представленных данных, путем разделения на каждой "&" характер:

```
сплит представлены $ [], данные $, "&"
Выберите X, представленный $ [], ""
```

Вот полный пример, что очищает возвращаемые значения немного больше, удаления URL вида и "?" характер, расщепление на «&» характер, и замена любой "+" (пробел) и "% 0D% 0A" (строки) символы, которые включены в последовательную строку данных:

```
html.open
html.load.url "файл: ///sdcard/rfo-basic/data/myform.html"
делать
    html.get.datalink данные $
до $ данных <> ""

! Получить индекс, где "?" символ найден в строке данных:

INDX = is_in ("?", данные $)

! Отрежьте все символы до и включая "?" персонаж:

Данные $ = $ середине (данные $ (INDX + 1))

! Сплит в & характер:

сплит представлены $ [], данные $, "&"

! Разрешить пользователю выбрать один из кусков раскол данных:

выберите выбран, представленный $ [], "Выберите поле:"
выбрали $ = $ представленные [выбрали]

! Сплит куски данных на "=" характер:

сплит выбрали $ [], выбранные $, "="

! Первый пункт из двух раскола выше это метка поля:

поле $ = $ выбранные [1]
```

```
! Во-вторых, значение:

Значение $ = $ выбранные [2]

! Космические и возвращение символов в тексте значение необходимо
декодировать:

Значение $ = $ заменить (значение $ ", +", "")
Значение $ = $ заменить (значение $ ",% 0D% 0A", "\ п")

! Распечатать поле и значение:

печать "поле =" + $ поле
"Значение =" печать + значение $
```

Вы можете использовать функцию замены $, чтобы удалить *любые другие коды* HTML, которые могут быть потенциально введенные пользователем в GUI HTML форм.

10.4 Динамически Создание GUI макеты

Функция "html.load.url" представил выше, могут читать HTML-файл из файловой системы, *или из* интернет-веб-сервер:

```
html.load.url "http://site.com/myform.html"
```

Если вам нужно создать графический интерфейс с динамически изменяющейся компоновки на основе пользовательского ввода или любой другой переменной состояния, используйте функцию "html.load.string". Это позволяет объединить в HTML строку, а затем отобразить его в качестве графического интерфейса:

```
вход "Имя:", имя $

FRM $ = "<действие формы = \" форма \ "> Имя: <br>" + ~
    "<тип входного = \" текст \ "Имя = \" "значение = \" Имя \ "+ ~
    назвать $ + "\"> поиска по "+ ~
    "<тип входного = \" представить \ "Имя = \" \ "представить значение = \"
представить \ ">" + ~
    "</ FORM>"
html.open
html.load.string FRM $
делать
    html.get.datalink данные $
до $ данных <> ""
Данные $ = $ середине (данные $, 5)
печать "Форма данных:" + $ Данные?
```

Техника выше, обеспечивает универсальный способ создания графических интерфейсов программно во время выполнения. Она также обеспечивает способ для хранения кода для макетов GUI непосредственно в коде программы, так, чтобы никакие дополнительные файлы HTML не должны быть распределены отдельно от сценария. Обратите внимание на использование "+ ~" символов конкатенировать многострочный строку, и использование "\" характер, чтобы включить кавычки в пределах цитируемого текста.

Еще одна полезная техника, чтобы прочитать exising HTML файл, а также использовать функцию "заменить $", чтобы динамически изменять макет по умолчанию. Например, следующий код считывает myform.html документ и изменяет цвет страницы:

```
grabfile FRM $, "myform.html"
! Замените тег тела с телом тега, содержащего цвет фона:
FRM $ = $ заменить (FRM $, "<тело>", "<тело BGCOLOR = \" # E6E6FA \ ">")
html.open
html.load.string FRM $
делать
    html.get.datalink данные $
до $ данных <> ""
Данные $ = $ середине (данные $, 5)
печать "Форма данных:" + данные $
```

Следующий код делает ту же самую вещь, но загружает HTML из каскадного строку, а не из внешнего файла:

```
FRM $ = "<голова> <название> Форма ввода данных </ название> </ HEAD>" + ~
"<тело>" + ~
    "<действие формы = \" форма \ ">" + ~
        "Имя: <br>" + ~
        "<тип входного = \" текст \ "Имя = \" Имя \ "> поиска по" + ~
        "E-mail: <br>" + ~
        "<тип входного = \" текст \ "Имя = \" по электронной почте \ ">
поиска по" + ~
        "Сообщение: <br>" + ~
        "<TextArea перевалы = 25 NAME = \" \ "сообщение строки = 5> Привет".
+ ~
        "</ TextArea> поиска по" + ~
        "Предпочтительный день: <br>" + ~
        "<имя выберите = \" день \ ">" + ~
            "<опция> понедельник <опция> вторник <опция> Среда" + ~
            "<опция> Четверг <опция> в пятницу <опция> суббота" + ~
            "<опция> воскресенье </ вариант>" + ~
        "</ выберите> поиска по" + ~
        "Любимый фрукт (ы): <br>" + ~
        "<тип входного = \" флажок \ "Имя = \" f1 \ "" + ~
        "Значение = \" рис \ "проверил> рис <br>" + ~
        "<тип входного = \" флажок \ "Имя = \" F2 \ "" + ~
        "Значение = \" яблоко \ "> яблоко <br>" + ~
        "<тип входного = \" флажок \ "Имя = \" F3 \ "" + ~
        "Значение = \" оранжевый \ "> оранжевый <br>" + ~
        "<тип входного = \" флажок \ "Имя = \" F4 \ "" + ~
        "Значение = \" банан \ "> банан поиска по" + ~
        "<тип входного = \" представить \ "Имя = \" \ "представить значение =
\" представить \ ">" + ~
    "</ FORM>" + ~
"</ BODY>"
FRM $ = $ заменить (FRM $, "<тело>", "<тело BGCOLOR = \" # E6E6FA \ ">")
html.open
html.load.string FRM $
делать
    html.get.datalink данные $
до $ данных <> ""
Данные $ = $ середине (данные $, 5)
печать "Форма данных:" + данные $
```

Использование заменить $ в этом смысле, это можно изменить *любой* предопределенный HTML в существующий макет GUI, добавьте данные по умолчанию для форм, или делать какие-либо другие изменения, все время выполнения, на основе данных, введенных пользователем, или любой другой динамический коэффициент (ы). Это очень мощный!

Узнать больше HTML, CSS, и Java-это полезно, если вы хотите создать впечатляюще сложные и / или ответные макеты, но элементы формы, которые вы видели до сих пор предоставляют все основные компоненты, необходимые для сбора данных от пользователей, таким образом, что знакомо и эффективным.

10.5 Некоторые более существенным HTML теги

Вот еще несколько примеров, демонстрирующих полезные теги HTML. Вставить каждый пример в отдельном файле кода, сохраните файлы с ".html" расширение, а затем открыть каждый файл в вашем веб-браузере, чтобы увидеть, как теги влияют макет и форматирование. Следующие примеры не улучшить вашу способность собирать пользовательские данные. Они просто демонстрируют корректировки свойств, таких как размер шрифта, пункт позиционирования, добавил графики, ссылок и т.д .:

```
<h1> Это некоторый текст заголовка </ h1>
<h3> Этот текст заголовка меньше </ h3>
В HTML, всех пространств, вкладок, новой строки и другие пробелы
сжаты (т.е. этот текст все на той же
Линия и переносится на длине экрана, и эти
пространства все сжимается до одного пространства. Если
Вы хотите, чтобы явно показывать определенное количество пробелов,
использовать эти символы: & NBSP; & NBSP; Для перевода строки (перевозка
возвращается), использовать "РБ" тег: Каталог <br>
Чтобы настроить размер и цвет текста, используйте "шрифт" тэг: Каталог <br>
<размер шрифта = 1 цвет = синий> Маленькая синий текст </ FONT> <br>
<размер шрифта = 6 цвет = красный> Большой красный текст </ FONT> Каталог
<br>
Для отображения ссылка на веб-сайте, использовать "а" тег: Каталог <br>
<a href="http://yahoo.com"> yahoo.com </a> Каталог <br>
Для отображения изображений, используйте "IMG" тег: Каталог <br>
<IMG SRC = "http://laughton.com/basic/logo.png"> Каталог <br>
Для того, чтобы ссылку на изображение к URL, использовать "IMG" и "а":
Каталог <br>
<a href="http://laughton.com/basic/" target=_blank>
    нажмите на изображение: <br>
    <IMG SRC = "http://laughton.com/basic/logo.png">
</a> Каталог <br>
<сильный> Этот текст полужирным. </ STRONG> Каталог <br>
<PRE>
    "Пред" тег отображает форматированный текст.
    Пробелы, вкладки, переводы строк и т.д., сохранены.
</ PRE>
Вы можете добавить горизонтальную разделительную линию с "ч" тега:
Каталог <br> <ширина ч = 80%> <br>
Ничего центре на страницу с "центром" тегу: Каталог <br>
<центр> По центру Текст </ центр> <br>
Таблицы используются для отображения строк и столбцов данных. Они
также могут быть использованы для выравнивания участков в HTML странице в
некоторых
области экрана (т.е., в меню макет баре отеля, заголовки,
Основными направлениями контент, и нижние колонтитулы): Каталог <br>
<центр>
    <таблица граница = 0 CELLPADDING = 10 ширина = 80%>
        <TR>
            <TD ширина = 33%> теги "TR" </ TD>
```

```
            <TD ширина = 33%> в </ TD>
            <TD ширина = 33%> Строки </ TD>
        </ TR>
        <TR>
            <TD ширина = 33%> теги "ТД" </ TD>
            <TD ширина = 33%> в </ TD>
            <TD ширина = 33%> колонки </ TD>
        </ TR>
        <TR>
            <TD VALIGN = верхняя Объединение колонок = 2 BGCOLOR = # CCCCCC>
                Вы можете установить "ТД" S, чтобы охватить как можно больше
столбцов, как
                необходимо, используя "Colspan" установку, и вы
                может использовать "VALIGN" свойство для установки, где
                В тексте клеток помещают. "BGColor" наборы
                Цвет ячейки фон.
            </ TD>
            <TD ширина = 20 = BGCOLOR # 000000>
                <цвет шрифта = белый размер = 2>
                    Вы можете установить размер свойства различны
                    части таблицы, используя либо процент
                    или пиксельные значения.
                </ FONT>
            </ TD>
        </ TR>
    </ стол>
</ центр>
Попробуйте изменить размер страницы, чтобы увидеть, как центрирование,
перенос текста и
стол макеты работать.
```

Следующие два примера показывают, как использовать таблицы для разметки элементов на странице:

```
<! DOCTYPE HTML PUBLIC "- // W3C // DTD HTML 4.0 Transitional // EN">
<HTML>
<HEAD>
  <Title> Общий HTML-таблиц с </ title>
  <META
        HTTP-экв = Content-Type содержание = "текст / HTML;
        кодировка = WINDOWS-1252 "
  >
</ HEAD>
<BODY BGColor = # CCCCCC>
    <Таблица выравнивание = центр фон = "" граница = 0
        CELLPADDING = 20 = 2 CELLSPACING высота = "95%" ширина = "85%"
    >
        <TR>
            <TD фон = "" BGColor = белый VALIGN = TOP>
                Ваше содержание страницы идет здесь.
            </ TD>
        </ TR>
    </ Table>
    <Таблица выравнивание = центр фон = "" граница = 0 = 2 CELLPADDING
        CELLSPACING = 2 ширина = "85%" высота = "5%"
    >
        <TR>
            <TD фон = "" CELLPADDING = 2 BGColor = # 000000 Высота = 5>
                <P ALIGN = центр>
```

```
                    <FONT COLOR = белый размер = 1>
                        Copyright © 2009 Yoursite.com.
                        Все права защищены.
                    </ FONT>
                </ P>
            </ TD>
        </ TR>
    </ Table>
</ BODY>
</ Html>


<! DOCTYPE HTML PUBLIC "- // W3C // DTD HTML 4.0 Transitional // EN">
<HTML>
<HEAD>
    <Title> Таблица основании макета страницы, Нет Меню </ title>
    <META HTTP-экв = Content-Type содержание = "текст / HTML;
         кодировка = WINDOWS-1252 "
    >
    <META
        HTTP-EQUIV = "Страница-Enter"
        СОДЕРЖАНИЕ = "RevealTrans (Продолжительность = 2, переход = 14)"
    >
    <СТИЛЬ ТИП = "текст / CSS">
        A: Hover {
            цвет синий;
            текст-отделка: подчеркнуть;
            фон: # F9EDED
        }
    </ STYLE>
</ HEAD>
<BODY BGColor = # 000000>
<Таблица выравнивание = центр фон = "" граница = 0 CELLPADDING = 2 = 2
CELLSPACING
    высота = "100%" ширина = "95%"
>
    <TR>
        <TD фон = "" BGColor = светло-серого VALIGN = TOP>
            <Таблица фон = "" граница = 0 CELLPADDING = 2 = 2 CELLSPACING
                высота = "100%" ширина = "100%"
            >
                <TR>
                    <TD фон = "" BGColor = черный Высота = 10>
                        <P ALIGN = центр>
                            <a href="./index.html">
                                <IMG ALIGN = базовый Alt = "заголовок
изображение"
                                    граница = 0 = 0 HSPACE SRC = "header.jpg"
                                >
                            </a>
                        </ P>
                    </ TD>
                </ TR>
                <TR>
                <TD фон = "" VALIGN = TOP>
                    <Таблица BGColor = # FFFFFF границы = 0 = 2 CELLPADDING
                        CELLSPACING = 2 высота = "100%" ширина = "100%"
                    >
                        <TR>
                            <TD BGColor = высота = 20 светло-серого>
                                <центр> </ центр>
```

```
                            </ TD>
                            <TD BGColor = высота = 10 светло-серого ширина =
"90%">
                                <центр>
                                    <a href="./Home.html">
                                        Главная
                                    </a>: Связаться
                                </ центр>
                            </ TD>
                            <TD BGColor = высота = 20 светло-серого>
                                <центр> </ центр>
                            </ TD>
                        </ TR>
                        <TR>
                            <TD фон = "" Colspan = 3 Высота = "100%">
                                <Таблица граница = 0 = 15 CELLPADDING
                                    CELLSPACING = 2 высота = "100%"
                                    ширина = "100%"
                                >
                                    <TR>
                                        <TD фон = "" VALIGN = TOP>
                                            Содержания страницы ЗДЕСЬ
                                        </ TD>
                                    </ TR>
                                </ Table>
                            </ TD>
                        </ TR>
                    </ Table>
                </ TD>
            </ TR>
            <TR>
                <TD фон = "" BGColor = # 000000 Высота = 5>
                    <P ALIGN = центр>
                        <FONT COLOR = белый размер = 1>
                            © 2010 веб-сайт.
                            Все права защищены.
                        </ FONT>
                    </ P>
                </ TD>
            </ TR>
        </ Table>
    </ TD>
</ TR>
</ Table>
</ BODY>
</ Html>
```

10.6 Отвечая на ссылки, кнопки Назад и ошибки

Следующий пример взят из руководства De Re Базовая Пола Лоутон. Он показывает, как реагировать на все возможные варианты HTML / Javascript UI:

```
html.open
html.load.url "файл: ///sdcard/rfo-basic/data/htmlDemo1.html"
xnextUserAction:
делать
    html.get.datalink данные $
до $ данных <> ""
```

```
тип $ = левый $ (данные $, 4)
Данные $ = $ середине (данные $, 5)
sw.begin типа $
    sw.case "BAK:"
        печать "НАЗАД ключ:" + данные $
        если данные $ = "1", то html.go.back еще конец
    sw.break
    sw.case "LNK:"
        печать "Гиперссылка выбран:" + данные $
        html.load.url данные $
    sw.break
    sw.case "ERR:"
        Печать "Ошибка:" + данные $
    sw.break
    sw.case "DAT:"
        печать "Данные пользователя:" + данные $
        Если оставить $ (данные $, 4) = "Выход"
            печать "Пользователь закончился демо."
            html.close
            конец
        ENDIF
        Если данные $ = "CTT"
            stt.results thelist
            list.get thelist, 1, ЬЬеТехЬ $
            ЬЬеТехЬ $ = "Вы сказал:" + $ ЬЬеТехЬ
            вставить $ = "JavaScript: текст (\" "+ $ ЬЬеТехЬ +" \ ")"
            печать вставки $
            html.load.url вставки $
        ENDIF
    sw.break
    sw.case "ЗА:"
        печать "Форма данных:" + данные $
        конец
    sw.break
    sw.case "DNL:"
        печать "Скачать:" + данные $
        array.delete P $ []
        сплит p $ [], данные $, "/"
        array.length л, p $ []
        п $ = p $ [л]
        html.load.string "<HTML> Начиная загрузку" Fn + $ + "</ HTML>"
        byte.open R, F, данных $
        паузу 2000
        html.go.back
        byte.copy e, п $
        byte.close
    sw.break
    sw.default
        печатать "неожиданный тип данных:" данные $
        конец
sw.end
```

10.7 Рецепт базы данных # 2 - Немного интерфейс хранения данных и Retrievel приложение

Программа ниже вариант сценария Рецепт предоставлен в предыдущем разделе. Для улучшения пользовательского ввода и отображения данных функциональность, длинная вереница HTML объединяется, чтобы сформировать интерфейс GUI. Заменить $ функция используется для динамического размещения данных в поля ввода текста и текстовой области, при необходимости

во время выполнения. Обратите внимание, что только измененный код из оригинальной программы являются строки, содержащие функции печати, входные и text.input:

```
! Вот макет по умолчанию графического интерфейса. Он содержит символы "TTTT",
! "IIII" и "CCCC", который будет заменен с живой название, ингредиента,
! и данные инструкции, как нужно:

GUI $ = "<действие формы = \" форма \ ">" + ~
    "Название: <br>" + ~
    "<тип входного = \" "NAME = \" "значение = \" текст \ название \ TTTT \
"> поиска по" + ~
    "Состав: <br>" + ~
    "<TextArea перевалы = 25 NAME = \" \ "ingred строк = 5> IIII </ TextArea>
поиска по" + ~
    "Инструкции: <br>" + ~
    "<TextArea перевалы = 25 NAME = \" \ "прибо строк = 5> CCCC </ TextArea>
поиска по" + ~
    "<тип входного = \" представить \ "Имя = \" представить \ "значение = \"
Готово \ ">" + ~
"</ FORM>"

! Эта функция используется для замены в формате символы, такие как пробелы и
! новые строки в виде текста, представленного:

fn.def pickvalue $ ($ представить)
    разделить представить $ [], представить $, "="
    Значение $ = $ представить [2]
    Значение $ = $ заменить (значение $ ", +", "")
    Значение $ = $ заменить (значение $ ",% 0D% 0A", "\ n")
    Значение $ = $ заменить (значение $ ", 27%", "" ")
    fn.rtn значение $
fn.end

! Основная программа начинается здесь:

File.Exists б ", recipes.txt"
если б = 0
    longstring $ = "Удалить блюдо из" + ~
    "упаковки \ nHeat в микроволновой печи \ nCool и едят вилкой"
    list.create c, newrecipes
    list.add newrecipes, "Овсянки", "Овсянки \ nWater \ nMaple Сироп" ~
    "Вскипятить воду \ Надда овсянка \ Надда кленовый сироп \ nCool и есть" ~
    "Замороженные Ужин", "замороженное блюдо \ nFork", longstring $
    list.size newrecipes, размер
    сейв $ = ""
    для я = 1 размера
        list.get newrecipes, я, м $
        Если я = размер
            сейв $ = $ + сейв м $
        другой
            сейв $ = $ + сейв м $ + "###"
        ENDIF
    Затем я
    text.open ж, MYFILE, "recipes.txt"
    text.writeln MYFILE, сейв $
    text.close MYFILE
ENDIF
loadSavedRecipes:
    grabfile serialdata $, "recipes.txt"
    array.delete рецепт $ []
    сплит рецепт $ [], serialdata $ ", ###"
```

```
    list.create c, рецепты
    list.add.array рецепты, рецепт $ []
    list.size рецепты, размер
    list.create c, названия
    для я = 1 размер шага 3
      list.get рецепты, я, п $
      list.add названия, п $
    Затем я
viewRecipe:
    list.search названия, "Создать новый рецепт ...", c
    если c = 0
        list.add названия, "Создать новый рецепт ..."
    ENDIF
    выберите INDX, заголовки, "Выберите рецепт, чтобы просмотреть /
редактировать", D
    Если D = 1, то в конечном
    list.get названия, INDX, RCP $
    если RCP $ = "Создать новый рецепт ..."
        перейти createNewRecipe
    ENDIF
    CLS
    S = INDX * 3 - 2
    list.get рецепты, S, T $
    list.get рецепты, S + 1, я $
    list.get рецепты, S + 2, в $
    gui2 $ = $ GUI
    gui2 $ = $ (заменить gui2 $ ", лл", т $)
    gui2 $ = $ заменить (gui2 $, "IIII", я $)
    gui2 $ = $ заменить (gui2 $, "CCCC", с $)
    html.open
    html.load.string gui2 $
    делать
        html.get.datalink данные $
    до $ данных <> ""
перейти viewRecipe
createNewRecipe:
    gui2 $ = $ GUI
    gui2 $ = $ заменить (gui2 $ ", лл", "")
    gui2 $ = $ заменить (gui2 $, "IIII", "")
    gui2 $ = $ заменить (gui2 $, "CCCC", "")
    html.open
    html.load.string gui2 $
    делать
        html.get.datalink данные $
    до $ данных <> ""
    INDX = is_in ("?", данные $)
    Данные $ = $ середине (данные $ (INDX + 1))
    array.delete представленные $ []
    сплит представлены $ [], данные $, "&"
    название $ = $ pickvalue (представленный $ [1])
    Ингредиенты $ = $ pickvalue (представленные $ [2])
    Инструкция $ = $ pickvalue (представленные $ [3])
    list.add рецепты, название $
    list.add рецепты, ингредиенты $
    list.add рецепты, инструкции $
    list.size рецепты, размер
    сейв $ = ""
    для я = 1 размера
        list.get рецепты, я, м $
        Если я = размер
            сейв $ = $ + сейв м $
```

```
        другой
            сейв $ = $ + сейв м $ + "###"
        ENDIF
    Затем я
    text.open ж, MYFILE, "recipes.txt"
    text.writeln MYFILE, сейв $
    text.close MYFILE
перейти loadSavedRecipes
```

Вот та же самая программа снова некоторые полезные части кода разразился в функции и подпрограммы. Изоляция повторные разделы полезной кода, такие как это может помочь сделать ваш код короче, более читаемым, и более универсальными, так как ваши программы становятся все более сложными. Вы можете использовать вариации pickvalue $ (), $ displayGUI () и сериализации $ () функции в любой программе, которая делает использование HTML форм с графическим интерфейсом:

```
GUI $ = "<действие формы = \" форма \ ">" + ~
    "Название: <br>" + ~
    "<тип входного = \" "NAME = \" "значение = \" текст \ название \ TTTT \
"> поиска по" + ~
    "Состав: <br>" + ~
    "<TextArea перевалы = 25 NAME = \" \ "ingred строк = 5> IIII </ TextArea>
поиска по" + ~
    "Инструкции: <br>" + ~
    "<TextArea перевалы = 25 NAME = \" \ "прибо строк = 5> CCCC </ TextArea>
поиска по" + ~
    "<тип входного = \" представить \ "Имя = \" представить \ "значение = \"
Готово \ ">" + ~
"</ FORM>"
fn.def pickvalue $ ($ представить)
    разделить представить $ [], представить $, "="
    Значение $ = $ представить [2]
    Значение $ = $ заменить (значение $ ", +", "")
    Значение $ = $ заменить (значение $ ",% 0D% 0A", "\ п")
    Значение $ = $ заменить (значение $ ", 27%", "" ")
    fn.rtn значение $
fn.end
fn.def displayGUI $ (тт $, II $, $ куб.см, г $)
    gui2 $ = г $
    gui2 $ = $ заменить (gui2 $, "TTTT", тт $)
    gui2 $ = $ заменить (gui2 $, "IIII", II $)
    gui2 $ = $ заменить (gui2 $, "CCCC", cc $)
    html.open
    html.load.string gui2 $
    делать
        html.get.datalink данные $
    до $ данных <> ""
    fn.rtn данные $
fn.end
fn.def сериализовать $ (recipelist)
    list.size recipelist, размер
    сейв $ = ""
    для я = 1 размера
        list.get recipelist, я, м $
        Если я = размер
            сейв $ = $ + сейв м $
        другой
            сейв $ = $ + сейв м $ + "###"
        ENDIF
```

```
    Затем я
    text.open ж, MYFILE, "recipes.txt"
    text.writeln MYFILE, сейв $
    text.close MYFILE
fn.end
File.Exists б ", recipes.txt"
если б = 0
    longstring $ = "Удалить блюдо из" + ~
    "упаковки \ nHeat в микроволновой печи \ nCool и едят вилкой"
    list.create c, newrecipes
    list.add newrecipes, "Овсянки", "Овсянки \ nWater \ nMaple Сироп" ~
    "Вскипятить воду \ Надда овсянка \ Надда кленовый сироп \ nCool и есть" ~
    "Замороженные Ужин", "замороженное блюдо \ nFork", longstring $
    SSS $ = $ (сериализации newrecipes)
ENDIF
loadSavedRecipes:
    grabfile serialdata $, "recipes.txt"
    array.delete рецепт $ []
    сплит рецепт $ [], serialdata $ ", ###"
    list.create c, рецепты
    list.add.array рецепты, рецепт $ []
    list.size рецепты, размер
    list.create c, названия
    для я = 1 размер шага 3
      list.get рецепты, я, п $
      list.add названия, п $
    Затем я
viewRecipe:
    list.search названия, "Создать новый рецепт ...", c
    если c = 0, то list.add названия, "Создать новый рецепт ..."
    выберите INDX, заголовки, "Выберите рецепт, чтобы просмотреть /
редактировать", D
    Если D = 1, то в конечном
    list.get названия, INDX, RCP $
    если RCP $ = "Создать новый рецепт ...", а затем перейти createNewRecipe
    CLS
    S = INDX * 3 - 2
    list.get рецепты, S, T $
    list.get рецепты, S + 1, я $
    list.get рецепты, S + 2, в $
    Данные $ = $ displayGUI (т $, я $, C $, $ GUI)
перейти viewRecipe
createNewRecipe:
    Данные $ = $ displayGUI ("", "", "", $ GUI)
    INDX = is_in ("?", данные $)
    Данные $ = $ середине (данные $ (INDX + 1))
    array.delete представленные $ []
    сплит представлены $ [], данные $, "&"
    название $ = $ pickvalue (представленный $ [1])
    Ингредиенты $ = $ pickvalue (представленные $ [2])
    Инструкция $ = $ pickvalue (представленные $ [3])
    list.add рецепты, название $
    list.add рецепты, ингредиенты $
    list.add рецепты, инструкции $
    SSS $ = $ (сериализации рецепты)
перейти loadSavedRecipes
```

11. Графика

Настройка экрана 11.1

Графический экран создается в BPO Basic с помощью "gr.open альфа, красный, зеленый, синий {,} ShowStatusBar" функция (параметр альфа уровень прозрачности, другие цвета образуют цвет фона графического экрана):

```
gr.open 255, 255, 255, 255% твердого белом фоне
```

Чтобы установить цвет и заполнить шаблон, используемый для создания графики, использовать "gr.color альфа, красный, зеленый, синий, стиль" функцию (для стиля, шаблон заполнения 1 = ход (контур), 2 = заполните, 3 = strokeandfill) :

```
gr.color 255, 0, 0, 255, 0% 0 прозрачность синий контур
```

11.2 Рисование фигур

Ряд функций позволяют размещать различные графические фигуры на экране:

```
gr.circle Object_number, X, Y, радиус
gr.line Object_number, x1, y1, x2, y2
gr.oval Object_number, слева, сверху, справа, снизу
gr.poly Object_number, List_pointer {x, y}
gr.rect Object_number, слева, сверху, справа, снизу
gr.arc Object_number, слева, сверху, справа, снизу, start_angle,
    sweep_angle, fill_mode
```

Чтобы разместить 10x15 квадрат на 50 пикселей более и 40 пикселей вниз от левого угла экрана, маркированные "rct1":

```
gr.rect rct1, 50, 40, 60, 15% слева, сверху, справа, снизу и координаты
```

11.3 Изменение положения и других свойств

Чтобы изменить какое-либо имущество (расположение, размер и т.д.) любого графического объекта, используйте функцию gr.modify.Следующий код перемещает выше 10 пикселей прямоугольник вправо (влево координат с позиции 50 до 60, правильно координировать от 60 до 70):

```
gr.modify rct1, "влево", 60
gr.modify rct1, "право", 70
```

11.4 Предоставление Изменения

Изменения, внесенные в любой графической *не появляются на экране, пока вы не вызовете* ***gr.render*** функцию. Для того, чтобы увидеть какие-либо изменения, внесенные в графический объект, вы должны делать это каждый раз:

```
gr.render
```

11.5 Циклы Графический Модификации

Вы можете использовать петли для перемещения объектов, используя переменные для значений позиции:

```
для я = 50 до 100 шаг 1% Переместить на 50 пикселей, каждый раз через петлю
    gr.modify rct1, "влево", я
    gr.modify rct1, "право", (я + 10)
    gr.render
    пауза 1% регулировать скорость анимации здесь
Следующий
```

11.6 Закрытие графический экран

Закройте графический экран с помощью функции gr.close:

```
gr.close
```

11.7 Анимированные Прямоугольник

Вот полная программа, которая ставит вместе все выше, для получения анимированного прямоугольник, который перемещается по экрану 50 пикселей:

```
gr.open 255, 255, 255, 255
gr.color 255, 0, 0, 255, 0
gr.rect rct1, 50, 40, 60, 15%, помните: слева, сверху, справа, снизу
gr.render
для я = 50 100 шаге 1
    gr.modify rct1, "влево", я
    gr.modify rct1, "право", (я + 10)
    gr.render
    пауза 1
Следующий
паузу 1000
gr.close
```

11.8 Установка Ориентация

Вы можете установить ориентацию графического экрана с помощью функции gr.orientation (по умолчанию = 0, пейзаж = пейзаж, портрет 1 = -1 = определяется датчиком):

```
gr.orientation 1
```

11.9 Текст на экране графика

Вы можете разместить текст на графическом экране с помощью функции gr.text.draw:

```
gr.text.draw txt1, 10, 10, "Привет Графический мира!"
```

Вот программа, которая движется текст по экрану, и изменяет текст объявить это текущее положение:

```
gr.open 255, 255, 255, 255
gr.color 255, 0, 0, 255, 0
gr.orientation 1
gr.text.draw txt1, 1, 60, "Моя текущая позиция" x "1"
для я = 1 100 шаге 1
    gr.modify txt1, "x", я
    gr.modify txt1, "Текст", "Моя текущая позиция" x "является" + Str $ (я)
    gr.render
    пауза 1
Следующий
паузу 1000
gr.close
```

11.10 Загрузка изображений

Создать образ пространства с помощью функции gr.bitmap.create. Загрузите изображения из файла на вашем устройстве, используя функцию gr.bitmap.load. Дисплей загруженные изображения, используя функцию gr.bitmap.draw:

```
gr.bitmap.create img1, 50, 50% Параметры высоты и ширины
gr.bitmap.load img1, "cartman.jpg"% установлена с BPO Basic
gr.bitmap.draw Img1, IMG, 20, 30% параметры позиции
```

Вот программа, которая перемещает cartman.jpg изображение по экрану:

```
gr.open 255, 255, 255, 255
gr.color 255, 0, 0, 255, 0
gr.orientation 1
gr.bitmap.create img1, 50, 50
gr.bitmap.load IMG, "cartman.jpg"
gr.bitmap.draw img1, IMG, 20, 30
gr.render
для я = 20 200 шаге 5
    gr.modify img1, "x", я
    gr.render
    пауза 1
Следующий
паузу 1000
gr.close
```

11.11 Переключение между графикой и текстом консоли экранов

Вы можете переключаться между экраном графики и нормальной РФО основного текста консоли вывода, используя функцию gr.front:

```
gr.front 0% показывает текстовую консоль
печатать "Get Ready для стадии 2 ..."
паузу 2000
gr.front 1% показывает экран графики
```

```
gr.render
```

11.12 Сенсорный

Используйте функцию gr.touch чтобы определить, где пользователь коснулся экрана. Не используйте DO / до петли ждать оттенком:

```
делать
    gr.touch коснулся, X, Y
пока не коснулся
gr.front 0
распечатать "Вы коснулись экрана в координаты"; Икс; "Икс"; Y
конец
```

11.13 Простой раздвижных игра-головоломка

Следующая программа является классическим 8-головоломка, в которой у вас есть 8 подвижных плитки и 1 пустое пространство. Сенсорный цветной плитки, что вы хотите, чтобы двигаться в пустом пространстве. Этот код использует много графики функций, условные оценки и управления потоком структуры, которые вы видели до сих пор:

```
! Настройте экран:

gr.open 255, 255, 255, 255% белый фон
gr.orientation 1% Режим portraid

! Мы создадим 9 различных цветных плиток, каждая 100 пикселей по горизонтали,
положил
! в квадратной сетки. Первый ряд плитки 3 будут размещены на высоте
! 0-100, следующая строка на высоте 100-200 и последняя строка на высоте
! 200-300:

! 1-й плитка (rct1):

gr.color 255, 0, 0, 255, 1% ЦВЕТ (трансп, красный, зеленый, синий, заполните)
gr.rect rct1, 0, 0, 100, 100% ПОЛОЖЕНИЕ (слева, сверху, справа внизу)

! 2-й плитка (rct2):

gr.color 255, 0, 255, 0, 1% (изменение цвета для каждого нового окна ...)
gr.rect rct2, 100, 0, 200, 100% (изменить положение для каждого нового
окна ...)

! 3-й плитки (RCT3):

gr.color 255, 255, 0, 0, 1% цвет
gr.rect RCT3, 200, 0, 300, 100% создавать и позиция

! 4-я плитка:

gr.color 255, 255, 255, 0, 1% цвет
gr.rect rct4, 0, 100, 100, 200% положение

! 5-я плитка:
```

```
gr.color 255, 0, 255, 255, 1% цвет
gr.rect rct5, 100, 100, 200, 200% положение

! 6-я плитка:

gr.color 255, 128, 0, 0, 1% цвет
gr.rect rct6, 200, 100, 300, 200% позиция

! 7-я плитка:

gr.color 255, 0, 128, 0, 1% цвет
gr.rect rct7, 0, 200, 100, 300% позиция

! 8-я плитка:

gr.color 255, 0, 0, 0, 1% цвет
gr.rect rct8, 100, 200, 200, 300% позиция

! 9-плитка (этот представляет собой пустое пространство, так что цвет белый):

gr.color 255, 255, 255, 255, 1% цвет
gr.rect rct9, 200, 200, 300, 300% позиция

! Показать экран

gr.render

! Теперь начать бесконечный цикл постоянно получить пользовательский ввод, а
затем сделать
! что-то с этого входа:

getInput:

    ! Получить сенсорный событие:

    делать
        gr.touch коснулся, X, Y
    пока не коснулся

    ! Установите X и Y значения координат в углу коснулся
    ! Положение Плитка в (мы хотим иметь дело с конкретными значениями
позиции
    ! от коснулись плитки, а не с произвольной позиции, в которой
    ! палец пользователя коснулся, где-то в плитке):

    если x <100%, если X находится где-то между 0-100, установить X до 0
        x = 0
    ElseIf x <200% bewtween 101-200, положим X 100
        x = 100
    еще% bewtween 201-300, положим X 200
        x = 200
    ENDIF
    если y <100% Сделайте то же самое со всеми потенциальными значений Y
        y = 0
    ElseIf y <200
        y = 100
    другой
        y = 200
    ENDIF
```

```
    ! Теперь нам нужно, чтобы получить текущие позиции каждой плитки, чтобы
определить,
    ! которой был тронут:

    gr.get.position rct1, x1, y1
    gr.get.position rct2, x2, y2
    gr.get.position RCT3, x3, y3
    gr.get.position rct4, x4, y4
    gr.get.position rct5, x5, y5
    gr.get.position rct6, x6, Y6
    gr.get.position rct7, x7, y7
    gr.get.position rct8, x8, y8
    gr.get.position rct9, x9, y9

    ! Сравнение каждой позиции выше прикосновения углу
    ! координат и отмечены его как коснулся плитки:

    если (x = x1) и (Y = Y1), то tilenum = rct1
    если (x = x2) & (y = y2), то tilenum = rct2
    если (x = x3) и (y = y3), то tilenum = RCT3
    если (x = x4) и (y = y4), то tilenum = rct4
    если (x = x5) и (y = y5), то tilenum = rct5
    если (x = x6) и (y = Y6), то tilenum = rct6
    если (x = x7) и (y = Y7), то tilenum = rct7
    если (x = x8) и (y = y8), то tilenum = rct8
    если (x = x9) и (y = y9), то tilenum = rct9

    ! Получить позицию плитки # 9 ("пустой") пространстве:

    gr.get.position rct9, blankx, blanky

    ! Поменять положение коснулся кусок, и пустой кусок.
    ! Это эффективно перемещает коснулся кусок в пустое пространство:

    gr.modify tilenum, "влево", blankx% переместить коснулся кусок
    gr.modify tilenum, "сверху", blanky% к прежней позиции
    gr.modify tilenum, "право", blankx + 100% от чистого листа
    gr.modify tilenum, "снизу", blanky + 100
    gr.modify rct9, "влево", x% переместить черный кусок
    gr.modify rct9, "сверху", y% к прежней позиции
    gr.modify rct9, "право", x + 100% от прикосновения кусок
    gr.modify rct9, "снизу", y + 100
    gr.render

! Сделайте весь процесс снова и снова, до тех пор, пока загадка решена:

перейти getInput
```

Вот полная программа без комментариев:

```
gr.open 255, 255, 255, 255
gr.orientation 1
gr.color 255, 0, 0, 255, 1
gr.rect rct1, 0, 0, 100, 100
gr.color 255, 0, 255, 0, 1
gr.rect rct2, 100, 0, 200, 100
gr.color 255, 255, 0, 0, 1
gr.rect RCT3, 200, 0, 300, 100
gr.color 255, 255, 255, 0, 1
```

```
gr.rect rct4, 0, 100, 100, 200
gr.color 255, 0, 255, 255, 1
gr.rect rct5, 100, 100, 200, 200
gr.color 255, 128, 0, 0, 1
gr.rect rct6, 200, 100, 300, 200
gr.color 255, 0, 128, 0, 1
gr.rect rct7, 0, 200, 100, 300
gr.color 255, 0, 0, 0, 1
gr.rect rct8, 100, 200, 200, 300
gr.color 255, 255, 255, 255, 1
gr.rect rct9, 200, 200, 300, 300
gr.render
getInput:
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    если x <100
        x = 0
    ElseIf x <200
        x = 100
    другой
        x = 200
    ENDIF
    если y <100
        y = 0
    ElseIf y <200
        y = 100
    другой
        y = 200
    ENDIF
    gr.get.position rct1, x1, y1
    gr.get.position rct2, x2, y2
    gr.get.position RCT3, x3, y3
    gr.get.position rct4, x4, y4
    gr.get.position rct5, x5, y5
    gr.get.position rct6, x6, Y6
    gr.get.position rct7, x7, y7
    gr.get.position rct8, x8, y8
    gr.get.position rct9, x9, y9
    если (x = x1) и (Y = Y1), то tilenum = rct1
    если (x = x2) & (y = y2), то tilenum = rct2
    если (x = x3) и (y = y3), то tilenum = RCT3
    если (x = x4) и (y = y4), то tilenum = rct4
    если (x = x5) и (y = y5), то tilenum = rct5
    если (x = x6) и (y = Y6), то tilenum = rct6
    если (x = x7) и (y = Y7), то tilenum = rct7
    если (x = x8) и (y = y8), то tilenum = rct8
    если (x = x9) и (y = y9), то tilenum = rct9
    gr.get.position rct9, blankx, blanky
    gr.modify tilenum, "влево", blankx
    gr.modify tilenum, "сверху", blanky
    gr.modify tilenum, "право", blankx + 100
    gr.modify tilenum, "снизу", blanky + 100
    gr.modify rct9, "влево", x
    gr.modify rct9, "сверху", y
    gr.modify rct9, "право", x + 100
    gr.modify rct9, "снизу", y + 100
    gr.render
перейти getInput
```

11.14 Добавление больше возможностей для раздвижных игра-головоломка

Предыдущий плитки игра дает хороший минимальный пример того, как реагировать на пользователя ощупь, и как двигаться графики в интерактивном режиме на экране. Для того, чтобы более играть игровой, несколько дополнительных функций имеют важное значение. Во-первых, отслеживание количества своп движется обеспечивает средства ведения счета:

```
оценка = 0
gr.open 255, 255, 255, 255
gr.orientation 1
gr.color 255, 0, 0, 255, 1
gr.rect rct1, 0, 0, 100, 100
gr.color 255, 0, 255, 0, 1
gr.rect rct2, 100, 0, 200, 100
gr.color 255, 255, 0, 0, 1
gr.rect RCT3, 200, 0, 300, 100
gr.color 255, 255, 255, 0, 1
gr.rect rct4, 0, 100, 100, 200
gr.color 255, 0, 255, 255, 1
gr.rect rct5, 100, 100, 200, 200
gr.color 255, 128, 0, 0, 1
gr.rect rct6, 200, 100, 300, 200
gr.color 255, 0, 128, 0, 1
gr.rect rct7, 0, 200, 100, 300
gr.color 255, 0, 0, 0, 1
gr.rect rct8, 100, 200, 200, 300
gr.color 255, 255, 255, 255, 1
gr.rect rct9, 200, 200, 300, 300
gr.color 255, 0, 0, 0, 1
gr.text.draw табло, 0, 350, "Переместить:" + Str $ (оценка)
gr.render
getInput:
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    если x <100
        x = 0
    ElseIf x <200
        x = 100
    другой
        x = 200
    ENDIF
    если y <100
        y = 0
    ElseIf y <200
        y = 100
    другой
        y = 200
    ENDIF
    gr.get.position rct1, x1, y1
    gr.get.position rct2, x2, y2
    gr.get.position RCT3, x3, y3
    gr.get.position rct4, x4, y4
    gr.get.position rct5, x5, y5
    gr.get.position rct6, x6, Y6
    gr.get.position rct7, x7, y7
    gr.get.position rct8, x8, y8
    gr.get.position rct9, x9, y9
    если (x = x1) и (Y = Y1), то tilenum = rct1
    если (x = x2) & (y = y2), то tilenum = rct2
    если (x = x3) и (y = y3), то tilenum = RCT3
    если (x = x4) и (y = y4), то tilenum = rct4
    если (x = x5) и (y = y5), то tilenum = rct5
```

```
    если (x = x6) и (y = Y6), то tilenum = rct6
    если (x = x7) и (y = Y7), то tilenum = rct7
    если (x = x8) и (y = y8), то tilenum = rct8
    если (x = x9) и (y = y9), то tilenum = rct9
    gr.get.position rct9, blankx, blanky
    gr.modify tilenum, "влево", blankx
    gr.modify tilenum, "сверху", blanky
    gr.modify tilenum, "право", blankx + 100
    gr.modify tilenum, "снизу", blanky + 100
    gr.modify rct9, "влево", x
    gr.modify rct9, "сверху", y
    gr.modify rct9, "право", x + 100
    gr.modify rct9, "снизу", y + 100
    оценка = оценка + 1
    gr.modify табло, "Текст", "Перемещение:" + Str $ (оценка)
    gr.render
перейти getInput
```

Вы, возможно, заметили, что выше игра позволяет незаконных ходов (т.е. части, которые не являются рядом с пустым пространством можно перемещать непосредственно в пустое пространство). Это следующая версия сравнивает позиции чистого листа и коснулся кусок, чтобы устранить эту возможность:

```
оценка = 0
gr.open 255, 255, 255, 255
gr.orientation 1
gr.color 255, 0, 0, 255, 1
gr.rect rct1, 0, 0, 100, 100
gr.color 255, 0, 255, 0, 1
gr.rect rct2, 100, 0, 200, 100
gr.color 255, 255, 0, 0, 1
gr.rect RCT3, 200, 0, 300, 100
gr.color 255, 255, 255, 0, 1
gr.rect rct4, 0, 100, 100, 200
gr.color 255, 0, 255, 255, 1
gr.rect rct5, 100, 100, 200, 200
gr.color 255, 128, 0, 0, 1
gr.rect rct6, 200, 100, 300, 200
gr.color 255, 0, 128, 0, 1
gr.rect rct7, 0, 200, 100, 300
gr.color 255, 0, 0, 0, 1
gr.rect rct8, 100, 200, 200, 300
gr.color 255, 255, 255, 255, 1
gr.rect rct9, 200, 200, 300, 300
gr.color 255, 0, 0, 0, 1
gr.text.draw табло, 0, 350, "Переместить:" + Str $ (оценка)
gr.render
getInput:
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    если x <100
        x = 0
    ElseIf x <200
        x = 100
    другой
        x = 200
    ENDIF
    если y <100
```

```
        y = 0
    ElseIf y <200
        y = 100
    другой
        y = 200
    ENDIF
    gr.get.position rct1, x1, y1
    gr.get.position rct2, x2, y2
    gr.get.position RCT3, x3, y3
    gr.get.position rct4, x4, y4
    gr.get.position rct5, x5, y5
    gr.get.position rct6, x6, Y6
    gr.get.position rct7, x7, y7
    gr.get.position rct8, x8, y8
    gr.get.position rct9, x9, y9
    если (x = x1) и (Y = Y1), то tilenum = rct1
    если (x = x2) & (y = y2), то tilenum = rct2
    если (x = x3) и (y = y3), то tilenum = RCT3
    если (x = x4) и (y = y4), то tilenum = rct4
    если (x = x5) и (y = y5), то tilenum = rct5
    если (x = x6) и (y = Y6), то tilenum = rct6
    если (x = x7) и (y = Y7), то tilenum = rct7
    если (x = x8) и (y = y8), то tilenum = rct8
    если (x = x9) и (y = y9), то tilenum = rct9
    gr.get.position rct9, blankx, blanky
    diffx = абс (x - blankx)
    diffy = абс (y - blanky)
    если ((diffx + diffy) <101)
        gr.modify tilenum, "влево", blankx
        gr.modify tilenum, "сверху", blanky
        gr.modify tilenum, "право", blankx + 100
        gr.modify tilenum, "снизу", blanky + 100
        gr.modify rct9, "влево", x
        gr.modify rct9, "сверху", y
        gr.modify rct9, "право", x + 100
        gr.modify rct9, "снизу", y + 100
        оценка = оценка + 1
        gr.modify табло, "Текст", "Перемещение:" + Str $ (оценка)
        gr.render
    ENDIF
перейти getInput
```

Наконец, добавив некоторый текст каждой плитки делает его легче разъяснить порядок, в котором плитки следует изменить. В следующем примере, число добавляются в текстовом поле, размещены в центре каждой плитки. Соответствующие текстовые объекты перемещаются, когда связанные с ними плитка изображения перемещаются:

```
оценка = 0
gr.open 255, 255, 255, 255
gr.orientation 1
gr.color 255, 0, 0, 255, 1
gr.rect rct1, 0, 0, 100, 100
gr.color 255, 0, 255, 0, 1
gr.rect rct2, 100, 0, 200, 100
gr.color 255, 255, 0, 0, 1
gr.rect RCT3, 200, 0, 300, 100
gr.color 255, 255, 255, 0, 1
gr.rect rct4, 0, 100, 100, 200
gr.color 255, 0, 255, 255, 1
```

```
gr.rect rct5, 100, 100, 200, 200
gr.color 255, 128, 0, 0, 1
gr.rect rct6, 200, 100, 300, 200
gr.color 255, 0, 128, 0, 1
gr.rect rct7, 0, 200, 100, 300
gr.color 255, 0, 0, 0, 1
gr.rect rct8, 100, 200, 200, 300
gr.color 255, 255, 255, 255, 1
gr.rect rct9, 200, 200, 300, 300
gr.color 255, 0, 0, 0, 1
gr.text.draw табло, 0, 350, "Переместить:" + Str $ (оценка)
gr.color 255, 255, 255, 255, 1
gr.text.draw текст1, 50, 50, "1"
gr.text.draw текст2, 150, 50, "2"
gr.text.draw text3, 250, 50, "3"
gr.text.draw text4, 50, 150, "4"
gr.text.draw text5, 150, 150, "5"
gr.text.draw text6, 250, 150, "6"
gr.text.draw text7, 50, 250, "7"
gr.text.draw text8, 150, 250, "8"
gr.text.draw text9, 250, 250, "9"
gr.render
getInput:
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    если x <100
        x = 0
    ElseIf x <200
        x = 100
    другой
        x = 200
    ENDIF
    если y <100
        y = 0
    ElseIf y <200
        y = 100
    другой
        y = 200
    ENDIF
    gr.get.position rct1, x1, y1
    gr.get.position rct2, x2, y2
    gr.get.position RCT3, x3, y3
    gr.get.position rct4, x4, y4
    gr.get.position rct5, x5, y5
    gr.get.position rct6, x6, Y6
    gr.get.position rct7, x7, y7
    gr.get.position rct8, x8, y8
    gr.get.position rct9, x9, y9
    если (x = x1) и (Y = Y1)
        tilenum = rct1
        textnum = текст1
    ENDIF
    если (x = x2) & (y = y2)
        tilenum = rct2
        textnum = текст2
    ENDIF
    если (x = x3) и (y = y3)
        tilenum = RCT3
        textnum = text3
    ENDIF
```

```
    если (x = x4) и (Y = y4)
        tilenum = rct4
        textnum = text4
    ENDIF
    если (x = x5) и (y = y5)
        tilenum = rct5
        textnum = text5
    ENDIF
    если (x = x6) и (y = Y6)
        tilenum = rct6
        textnum = text6
    ENDIF
    если (x = x7) и (y = Y7)
        tilenum = rct7
        textnum = text7
    ENDIF
    если (x = x8) и (y = y8)
        tilenum = rct8
        textnum = text8
    ENDIF
    если (x = x9) и (y = y9)
        tilenum = rct9
        textnum = text9
    ENDIF
    gr.get.position rct9, blankx, blanky
    diffx = абс (x - blankx)
    diffy = абс (y - blanky)
    если ((diffx + diffy) <101)
        gr.modify tilenum, "влево", blankx
        gr.modify tilenum, "сверху", blanky
        gr.modify tilenum, "право", blankx + 100
        gr.modify tilenum, "снизу", blanky + 100
        gr.modify textnum, "x", blankx + 50
        gr.modify textnum, "y", blanky + 50
        gr.modify rct9, "влево", x
        gr.modify rct9, "сверху", y
        gr.modify rct9, "право", x + 100
        gr.modify rct9, "снизу", y + 100
        оценка = оценка + 1
        gr.modify табло, "Текст", "Перемещение:" + Str $ (оценка)
        gr.render
    ENDIF
перейти getInput
```

11.15 Масштабирование графика по размеру различным разрешением экрана

РФО Основные включает в себя простой функцию масштабирования графики правильно поместиться на экранах любого разрешения.Следующий код, взятый непосредственно из Де Re Основном руководстве демонстрирует, как масштабировать ваши экраны. Все, что вам нужно сделать, это задать начальные значения (2 di_height и di_width), и вся графических приложений не будет масштабироваться по отношению к тем заданного размера, независимо от того, насколько большой или как маленький размеры экрана являются:

```
di_height = 480
di_width = 800
gr.open 255, 255, 255, 255
gr.orientation 0
gr.screen actual_w, actual_h
```

```
scale_width = actual_w / di_width
scale_height = actual_h / di_height
gr.scale scale_width, scale_height
```

12. Более Графические программы

12.1 Поймайте игры

Поймайте это простая игра, в которой объекты красный круг упасть с неба. Падающие круги должны быть пойманы до удара о землю. Игрок изображение представляется в синей коробке на нижней части экрана. Пользователь затрагивает левые и правые части экрана, чтобы переместить игрока кусок из стороны в сторону. Одно очко добавляется каждый раз, когда падает красный круг, выловленную (сталкивается с) в поле голубой игроков. 1 балл вычитается каждый раз, когда падает красный круг делает его в нижней части экрана, не будучи пойманы. Игра скорость постепенно увеличивается со временем. Нажмите кнопку Назад, в любое время, чтобы закончить игру. Игра игра продолжается только пока пользователь касается экрана. Целью является заканчивается, когда вы думаете, у вас есть лучший возможный балл.

Вы заметите, что большая часть кода и наброски этой программы похож на раздвижной плитка головоломка приложение. Функция gr_collision () используется для определения, если красный круг касаясь синюю коробку. Всякий раз, когда красный круг поймана или достигает нижней части экрана, то перемещаются в случайное координат выше верхней части экрана и снова упала.

```
оценка = 0
скорость = 4,0
gr.open 255, 255, 255, 255
gr.orientation 1
gr.color 255, 30, 30, 30, 1% игровая площадка
gr.rect область, 0, 300, 0, 300% игровая площадка
gr.color 255, 255, 0, 0, 1
gr.circle мяч, круглый (RND () * 300), 0, 10
gr.color 255, 0, 0, 255, 1
gr.rect проигрыватель, 150, 280, 200, 300
gr.text.draw табло, 0, 350, "Счет:" + Str $ (оценка)
gr.render
делать
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    gr.get.position мяч, x1, y1
    gr.get.position плеер, x2, y2
    если x <(x2 + 15)
        пусть направление = -10
    ElseIf x> (x2 + 35)
        пусть направление = 10
    другой
        пусть направление = 0
    ENDIF
    gr.modify мяч "y", y1 + круглая (скорость)
    gr.modify игрок, "левый", x2 + направление
    gr.modify игрок "право", x2 + 50 + направление
    gr.render
    если gr_collision (мяч, игрока)
        оценка = оценка + 1
        скорость = скорость + 0,2
        GoSub positionBall
    ENDIF
```

```
    если y1> 300
        оценка = оценка - 1
        GoSub positionBall
    ENDIF
    gr.render
    Пауза 5
до 1 = 2
positionBall:
    gr.modify табло, "Текст", "Результат:" + Str $ (оценка)
    gr.modify мяч "y", 0
    gr.modify мяч, "x", круглый (RND () * 300)
вернуть
OnBackKey:
    всплывающее "Счет:" + Str $ (оценка), 0,0,1
    конец
```

12.2 Калькулятор

Каждый программирование учебник включает в себя ав необходимую пример калькулятора. Вот один для BPO Basic:

```
! Начните с назначения некоторые начальные переменные, которые будут
использоваться позже:

PREVVAL = 0
Кюрваль = 0
displayFlag = 0

! Откройте экран графики и задать начальные параметры отображения:

gr.open 255248248248
gr.color 255,0,0,0,0
gr.orientation 1
gr.text.bold 1

! Нарисуйте прямоугольник на экране, чтобы использовать как номер дисплея:

gr.rect displayBox, 10,10,300,50

! Нарисуйте текст на экране, который будет отображать номера набранные
! пользователь, и результат любого расчета:

gr.text.draw DisplayText, 20,35, "0"

! Вся эта следующем разделе diplays кнопки на экране. Было бы
! Были немного более простым, чтобы просто привлечь каждый прямоугольник и
! текст вручную. Вместо этого, массив, содержащий текст и координаты
! каждая кнопка создается впервые, а затем цикл используется для
позиционирования
! каждый элемент на экране. Это позволяет легко репозиционирования и
изменение размеров
! для различных размеров экрана и других будущих изменений макета /
! Особенности:

! Этот массив содержит текст и координаты каждой кнопки:

array.load buttonData $ [], ~
    "1", "30", "60", "2", "90", "60", "3", "150", "60", ~
```

```
    "4", "30", "120", "5", "90", "120", "6", "150", "120", ~
    "7", "30", "180", "8", "90", "180", "9", "150", "180", ~
    "0", "30", "240", "+", "90", "240", "-", "150", "240", ~
    "*", "30", "300", "/", "90", "300", "=", "150", "300"

! Эта линия устанавливает переменную количества элементов в списке выше.
! Это позволяет более кнопок, которые будут добавлены позже, без
необходимости вручную
! редактировать переменные .:

array.length numButtons, buttonData $ []

! Создайте 2 новые пустые массивы, которые будут содержать ссылки на каждого
прямоугольника
! и текстовый элемент, который делает каждую кнопку графический:

тусклые rectNames [numButtons]
тусклые textNames [numbuttons]

! Установите несколько переменных, содержащих размер и горизонтальный /
вертикальный сдвиг
! все кнопки (эти значения могут легко рассчитывается как продукты
! Ширина и высота экрана):

BS = 20% размер кнопки
xshift = 65% в этом движется все кнопки в течение заданного # пикселей
yshift = 40% в этом движется все кнопки вниз данную # пикселей

! Этот цикл создает каждую кнопку, запустив хотя buttonData $ [] Массив
! выше, вытаскивая группы из 3 предметов из списка:

для I = 1 numButtons шагу 3

    ! Установите левый и верхние позиции для каждого последовательного текста
пункта:

    LPOS = Val (buttonData $ [я + 1]) + xshift
    OCT = Val (buttonData $ [я + 2]) + yshift

    ! Нарисуйте прямоугольник вокруг голубой местоположения каждого
текстового элемента.
    ! Обратите внимание, что позиции прямоугольника на основе текста
    ! позиции, переменного размера кнопка (BS), и десять дополнительных
пикселей
    ! смещение на верхней и правой сторон, чтобы вместить размер из
    ! Текст (это смещение может быть также рассчитана автоматически):

    gr.color 255,0,0,255,1
    gr.rect rectNames [я], LPOS-BS, OCT-BS-10, LPOS + BS + 10, + OCT BS

    ! Нарисуйте элемент текста в белом:

    gr.color 255,255,255,255,1
    gr.text.draw textNames [I], LPOS, OCT, buttonData $ [я]

Затем я

! Установите цвет обратно на черный и показать окно:

gr.color 255,0,0,0,0
gr.render
```

```
! Это бесконечный цикл ожидает ввода пользователя:

getInput:

    ! Подождите для ввода текста и сохранить координаты в переменных "x"
    ! и "y":

    делать
        gr.touch коснулся, X, Y
    пока не коснулся

    ! Это цикл проходит по массиву buttonData $ [] Каждый раз, когда
    ! пользователь прикасается к экрану, чтобы проверить, какая кнопка была
нажата,
    ! и реагирует с соответствующим действием (ов):

    для I = 1 numbuttons шагу 3

        ! Каждый раз подряд через петлю, координаты каждого
        ! текст кнопки оцениваются:

        LPOS = Val (buttonData $ [я + 1]) + xshift
        ОСТ = Val (buttonData $ [я + 2]) + yshift

        ! Расположение на ощупь по сравнению с расположением друг
        ! кнопку в списке:

        Если X> (LPOS-BS) и x <(LPOS + BS + 10) и ~
                Y> (ТПО-BS-10) и y <(ТПО) + BS

            ! Если коснулся СВЕРШИВШИМСЯ внутри прямоугольника, окружающего
            ! Нынешний текст кнопки, установите переменную Q $ к
соответствующим
            ! Текст кнопки:

            Q $ = $ buttonData [я]

            ! Цифровые кнопки ведут себя иначе, чем кнопками управления:

            Если Q $ = "1" | Q $ = "2" | Q $ = "3" | Q $ = "4" | Q $ = "5" |
~
                    Q $ = "6" | Q $ = "7" | Q $ = "8" | Q $ = "9" | Q $ = "0"

                ! Этот флаг используется для обновления экрана правильно
(хорошо
                ! Дисплей должен быть очищен, если в настоящее время общая
сумма
                ! показывая - этот флаг используется для отслеживания это
состояние):

                если displayFlag = 1
                    gr.modify DisplayText, "Текст", дисплей $
                    gr.render
                    displayFlag = 0
                ENDIF

                ! Если "0" было на экране, ясно и обновить
                ! Дисплей:

                если дисплей $ = "0"
```

```
                    дисплей $ = ""
                    gr.modify DisplayText, "Текст", дисплей $
                    gr.render
                ENDIF

                ! Добавьте текст выбранной кнопки, чтобы ток
                ! отобразить текст, и обновлять экран:

                дисплей $ = $ + дисплей buttonData $ [я]
                gr.modify DisplayText, "Текст", дисплей $
                gr.render

                ! Установите переменную "Кюрваль" провести численное значение
                ! текста отображается в данный момент на экране:

                Кюрваль = Val (дисплей $)

            ! Все ключи оператора выполнить то же самое. Они устанавливают
            ! переменная "оператор $" к работе ключевые выполняет,
            ! а потом запустить "setEval" подпрограмму:

            ElseIf Q $ = "+"
                Оператор $ = "+"
                GoSub setEval
            ElseIf Q $ = "-"
                Оператор $ = "-"
                GoSub setEval
            ElseIf Q $ = "*"
                Оператор $ = "*"
                GoSub setEval
            ElseIf Q $ = "/"
                Оператор $ = "/"
                GoSub setEval

            ! Равное ключ выполняет все следующие действия:

            ElseIf Q $ = "="

                ! Проверьте, если переменная displayFlag находится в
                ! соответствующее состояние (не отображается общая):

                если displayFlag = 0

                    ! Если пользователь пытается делить на ноль, всплывающее
сообщение об ошибке
                    ! сообщение и выход из текущего цикла:

                    если оператор $ = "/" & Кюрваль = 0.0
                        всплывающее "Деление на 0 не допускается.", 0,0,1
                        перейти getInput
                    ENDIF

                    ! Выполните соответствующую оценку по математике, на
основе
                    ! текущее состояние оператора переменной $
(устанавливается
                    ! выше, всякий раз, когда пользователь нажимает на
клавиши оператора):

                    если оператор $ = "+"
                        Кюрваль = Кюрваль + PREVVAL
```

```
                    ElseIf оператор $ = "-"
                        Кюрваль = PREVVAL - Кюрваль
                    ElseIf оператор $ = "*"
                        Кюрваль = Кюрваль * PREVVAL
                    ElseIf оператор $ = "/"
                        Кюрваль = PREVVAL / Кюрваль
                    ENDIF

                    ! Преобразование обновленное значение "Кюрваль" в строку,
а
                    ! обновления дисплея:

                    отобразить $ = $ ул (Кюрваль)
                    gr.modify DisplayText, "Текст", дисплей $
                    gr.render

                    ! Установите переменную displayFlag чтобы указать, что
                    ! Общая настоящее время отображается на экране:

                    displayFlag = 1

                ENDIF
            ENDIF
        ENDIF
    Затем я

! Теперь вернемся и ждать более удобного ввода, бесконечно:

перейти getInput

! Это подпрограмма выполняется, когда любой из клавиш оператора нажата:

setEval:
    ! Сохранить значение в настоящее время на строку в переменной "PREVVAL",
    ! которые будут использоваться позже:

    PREVVAL = Кюрваль

    ! Очистите отображения текста, а также обновлять экран:

    дисплей $ = ""
    gr.modify DisplayText, "Текст", дисплей $
    gr.render

вернуть

! Этот последний бит кода заканчивается программу изящно, когда пользователь
нажимает
! Андроид назад кнопка:

OnError:
    CLS
    конец
```

Вот и вся программа без комментариев:

```
PREVVAL = 0
Кюрваль = 0
displayFlag = 0
```

```
gr.open 255248248248
gr.color 255,0,0,0,0
gr.orientation 1
gr.text.bold 1

gr.rect displayBox, 10,10,300,50
gr.text.draw DisplayText, 20,35, "0"

array.load buttonData $ [], ~
    "1", "30", "60", "2", "90", "60", "3", "150", "60", ~
    "4", "30", "120", "5", "90", "120", "6", "150", "120", ~
    "7", "30", "180", "8", "90", "180", "9", "150", "180", ~
    "0", "30", "240", "+", "90", "240", "-", "150", "240", ~
    "*", "30", "300", "/", "90", "300", "=", "150", "300"
array.length numButtons, buttonData $ []
тусклые rectNames [numButtons]
тусклые textNames [numbuttons]
BS = 20% размер кнопки
xshift = 65% в этом движется все кнопки в течение заданного # пикселей
yshift = 40% в этом движется все кнопки вниз данную # пикселей
для I = 1 numButtons шагу 3
    LPOS = Val (buttonData $ [я + 1]) + xshift
    OCT = Val (buttonData $ [я + 2]) + yshift
    gr.color 255,0,0,255,1
    gr.rect rectNames [я], LPOS-BS, OCT-BS-10, LPOS + BS + 10, + OCT BS
    gr.color 255,255,255,255,1
    gr.text.draw textNames [I], LPOS, OCT, buttonData $ [я]
Затем я

gr.color 255,0,0,0,0
gr.render

getInput:
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    для I = 1 numbuttons шагу 3
        LPOS = Val (buttonData $ [я + 1]) + xshift
        OCT = Val (buttonData $ [я + 2]) + yshift
        Если X> (LPOS-BS) и x <(LPOS + BS + 10) и ~
                Y> (ТПО-BS-10) и y <(ТПО) + BS
            Q $ = $ buttonData [я]
            Если Q $ = "1" | Q $ = "2" | Q $ = "3" | Q $ = "4" | Q $ = "5" |
~
                    Q $ = "6" | Q $ = "7" | Q $ = "8" | Q $ = "9" | Q $ = "0"
                если displayFlag = 1
                    gr.modify DisplayText, "Текст", дисплей $
                    gr.render
                    displayFlag = 0
                ENDIF
                если дисплей $ = "0"
                    дисплей $ = ""
                    gr.modify DisplayText, "Текст", дисплей $
                    gr.render
                ENDIF
                дисплей $ = $ + дисплей buttonData $ [я]
                gr.modify DisplayText, "Текст", дисплей $
                gr.render
                Кюрваль = Val (дисплей $)
            ElseIf Q $ = "+"
```

```
                Оператор $ = "+"
                GoSub setEval
            ElseIf Q $ = "-"
                Оператор $ = "-"
                GoSub setEval
            ElseIf Q $ = "*"
                Оператор $ = "*"
                GoSub setEval
            ElseIf Q $ = "/"
                Оператор $ = "/"
                GoSub setEval
            ElseIf Q $ = "="
                если displayFlag = 0
                    если оператор $ = "/" & Кюрваль = 0.0
                        всплывающее "Деление на 0 не допускается.", 0,0,1
                        перейти getInput
                    ENDIF
                    если оператор $ = "+"
                        Кюрваль = Кюрваль + PREVVAL
                    ElseIf оператор $ = "-"
                        Кюрваль = PREVVAL - Кюрваль
                    ElseIf оператор $ = "*"
                        Кюрваль = Кюрваль * PREVVAL
                    ElseIf оператор $ = "/"
                        Кюрваль = PREVVAL / Кюрваль
                    ENDIF
                    отобразить $ = $ ул (Кюрваль)
                    gr.modify DisplayText, "Текст", дисплей $
                    gr.render
                    displayFlag = 1
                ENDIF
            ENDIF
        ENDIF
    Затем я
перейти getInput
setEval:
    PREVVAL = Кюрваль
    дисплей $ = ""
    gr.modify DisplayText, "Текст", дисплей $
    gr.render
вернуть
OnError:
    CLS
    конец
```

12.3 Змея

12.4 лыж

12.5 Простой стрелять-эм-до игры

12.6 гитарных аккордов Диаграмма чайник

12.7 Image Viewer С жестов

12.8 эскизов Image Maker

13. Строительство GUI интерфейсов с помощью графика

13.1 Основы - Нет HTML-Требуемые

Вы уже видели, как создавать интерфейсы GUI, используя HTML. В тех случаях, когда необходимо смешивать с ГПИ графика и анимация, или в ситуациях, когда вы не хотите использовать HTML / Javascript интерфейс, это может быть полезно, чтобы построить GUI интерфейсы с нуля, используя родные графические элементы. Следующий пример демонстрирует некоторые основные методы для создания интерактивных графических интерфейсов, используя РФО Основные команды:

```
! Этот пример позволит пользователю ввести 4 шт данных. Установите те
! переменные первоначально нуль:

Текст $ = ""
BigText $ = ""
Количество = 0
ListSelection $ = ""

! Откройте экран графики:

gr.open 255255255255
gr.color 255,0,0,0,0
gr.orientation 1

! Нарисуйте текстовую метку на экране и текстовую область, где введенный
текст будет
! отображаться. Подчеркните текстовую область. Эта область текста будет
принимать
! простой текст (вступил через функцию «входного»):

gr.text.draw t1,10,50 "текст:"
gr.text.draw f1,100,50 ","
gr.line L1, 100,52,300,52

! Нарисуйте еще один ярлык, текстовую область и подчеркнуть. Это будет
принимать
! длинный текст (вводится через функцию "text.input"):

gr.text.draw t2,10,100, "Большой текст:"
gr.text.draw f2,100,100 ","
gr.line L2, 100,102,300,102

! Другой лейбл, площадь, и подчеркивание. Это принимает число (введенные с
помощью
! функция "вход", с переменной типа входного установлен в номер,
! вместо строки):

gr.text.draw t3,10,150 "Номер:"
gr.text.draw f3,100,150 ","
gr.line L3, 100,152,300,152

! Другой лейбл, площадь, и подчеркивание. Это принимает данные, выбранные
! Пользователь из списка выбора (вступила с помощью функции "выбрать"):

gr.text.draw t4,10,200 "список Выберите:"
gr.text.draw f4,100,200 ","
gr.line L4, 100,202,300,202

! Добавить красный "Отправить" ссылку, которая пользователь щелкнет после
всех данных имеет
! были введены:

gr.color 255,255,0,0,0
```

```
gr.text.draw b1,100,270, «ВВЕСТИ"
gr.color 255,0,0,0,0

! Показать экран:

gr.render

! Начните бесконечный цикл, который принимает ввод данных пользователем:

GetText:

    ! Подождите пользователя прикоснуться к экрану. Назначьте коснулся
    ! координат переменных "x" и "y":

    делать
        gr.touch коснулся, X, Y
    пока не коснулся

    ! Скрыть экран графический мгновение (показать входные диалоги):

    gr.front 0

    ! Надлежащим образом реагировать в соответствии с положением, в котором
экран
    ! был тронут:

    если y <75

        ! "Текст" поле коснулся. Запустите функцию "вход" и обновить
        ! текстовое поле с пользователей вошли результаты:

        вход "Текст:" текст $, $ текст
        gr.modify f1, "Текст", текст $

    ElseIf y <125

        ! "Большой текст" поле коснулся. Запустите функцию "text.input" и
        ! обновить текстовое поле с введенным пользователем результатов:

        text.input BigText $, $ BigText
        gr.modify F2 ", текст", BigText $

    ElseIf y <175

        ! "Количество" поле коснулся. Запустите функцию "вход" и обновить
        ! текстовое поле с пользователей вошли результаты (в формате
необходимости):

        вход "Номер:", номер, номер
        gr.modify F3 ", текст", заменить $ (Str $ (номер), ".0", "")

    ElseIf y <225

        ! "Список Выберите" Поле коснулся. Запустите функцию «Избранное» и
        ! обновить текстовое поле с выбранным пользователем результатов:

        array.delete список $ []
        Список array.load $ [], "Красный", "Зеленый", "Голубой"
        выберите SelectedItem, список $ [] ", любимый цвет:"
        ListSelection $ = $ список [SelectedItem]
        gr.modify f4, "Текст", ListSelection $
```

```
    ElseIf y> 250 & Y <300

        ! "Отправить" ссылка коснулся. В программе, это, где вы будете
        ! procees данные, представленные пользователем. В этом примере,
        ! Введенные данные просто выводится на экран:

        gr.front 0
        печатать "Отправлено: \ п \ п"; Текст $, $ BigText, номер,
ListSelection $
        паузу 2000

    ENDIF

    ! Показать обновления графика экрана для отображения введенных данных:

    gr.front 1
    gr.render

! Продолжайте принимать ввод до тех пор, пока пользователь не представляет
данные или выходы:

перейти GetText

! Если пользователь выходит из программы с помощью кнопки назад, выйти
изящно:

OnError:
    CLS
    конец
```

Вот и вся программа без комментариев:

```
Текст $ = ""
BigText $ = ""
Количество = 0
ListSelection $ = ""

gr.open 255255255255
gr.color 255,0,0,0,0
gr.orientation 1

gr.text.draw t1,10,50 "текст:"
gr.text.draw f1,100,50 ","
gr.line L1, 100,52,300,52

gr.text.draw t2,10,100, "Большой текст:"
gr.text.draw f2,100,100 ","
gr.line L2, 100,102,300,102

gr.text.draw t3,10,150 "Номер:"
gr.text.draw f3,100,150 ","
gr.line L3, 100,152,300,152

gr.text.draw t4,10,200 "список Выберите:"
gr.text.draw f4,100,200 ","
gr.line L4, 100,202,300,202

gr.color 255,255,0,0,0
```

```
gr.text.draw b1,100,270, «ВВЕСТИ"
gr.color 255,0,0,0,0

gr.render

GetText:
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    gr.front 0
    если y <75
        вход "Текст:" текст $, $ текст
        gr.modify f1, "Текст", текст $
    ElseIf y <125
        text.input BigText $, $ BigText
        gr.modify F2 ", текст", BigText $
    ElseIf y <175
        вход "Номер:", номер, номер
        gr.modify F3 ", текст", заменить $ (Str $ (номер), ".0", "")
    ElseIf y <225
        array.delete список $ []
        Список array.load $ [], "Красный", "Зеленый", "Голубой"
        выберите SelectedItem, список $ [] ", любимый цвет:"
        ListSelection $ = $ список [SelectedItem]
        gr.modify f4, "Текст", ListSelection $
    ElseIf y> 250 & Y <300
        gr.front 0
        печатать "Отправлено: \ п \ п"; Текст $, $ BigText, номер,
ListSelection $
        паузу 2000
    ENDIF
    gr.front 1
    gr.render
перейти GetText
OnError:
    CLS
    конец
```

13.2 Типичные Форма графического интерфейса еще с несколькими прибамбасами

Этот пример показывает, как добавить прямоугольную кнопку, что изменяет размер, чтобы соответствовать текст кнопки, прямоугольные области для ввода текста, и по правому краю текстовых меток:

```
Текст $ = ""
BigText $ = ""
Количество = 0
ListSelection $ = ""

gr.open 255248248248
gr.color 255,0,0,0,0
gr.orientation 1

gr.text.align 3
gr.text.draw t1,75,50 "текст:"
gr.text.align 1
gr.text.draw f1,100,50 ","
gr.rect L1, 95,27,300,65
```

```
gr.text.align 3
gr.text.draw t2,75,100, "Большой текст:"
gr.text.align 1
gr.text.draw f2,100,100 ","
gr.rect L2, 95,77,300,115

gr.text.align 3
gr.text.draw t3,75,150 "Номер:"
gr.text.align 1
gr.text.draw f3,100,150 ","
gr.rect L3, 95,127,300,165

gr.text.align 3
gr.text.draw t4,75,200 "список Выберите:"
gr.text.align 1
gr.text.draw f4,100,200 ","
gr.rect L4, 95,177,300,215

gr.text.bold 1
btn1 $ = "ВВЕСТИ"
gr.color 255,0,0,0,1
gr.get.textbounds btn1 $, L, T, R, B
gr.rect btn1, L + 110-10, 270-10 T +, R + 110 + 10, B + 270 + 10% матч b1 пос
(110)
gr.text.bold 0
gr.color 255,255,255,255,0
gr.text.draw b1,110,270, btn1 $
gr.color 255,0,0,0,0

gr.render

GetText:
    делать
        gr.touch коснулся, X, Y
    пока не коснулся
    gr.front 0
    если y <75
        вход "Текст:" текст $, $ текст
        gr.modify f1, "Текст", текст $
    ElseIf y <125
        text.input BigText $, $ BigText
        gr.modify F2 ", текст", BigText $
    ElseIf y <175
        вход "Номер:", номер, номер
        gr.modify F3 ", текст", заменить $ (Str $ (номер), ".0", "")
    ElseIf y <225
        array.delete список $ []
        Список array.load $ [], "Красный", "Зеленый", "Голубой"
        выберите SelectedItem, список $ [] ", любимый цвет:"
        ListSelection $ = $ список [SelectedItem]
        gr.modify f4, "Текст", ListSelection $
    ElseIf y> 250 & Y <300
        gr.front 0
        печатать "Отправлено: \ п \ п"; Текст $, $ BigText, номер,
ListSelection $
        паузу 2000
    ENDIF
    gr.front 1
    gr.render
перейти GetText
```

```
OnError:
    CLS
    конец
```

? --- Рецепт базы данных # 3 (на этот раз с графическим GUI)

14. Transferrting данных через сетевые сокеты

Вычислительных устройств, соединенных вместе на одном проводной или беспроводной сети могут передавать данные и файлы непосредственно друг с другом через сеть (TCP и UDP) "Гнездо" соединений.

Сетевые приложения, как правило, состоят из двух или более отдельных программ, каждое из которых работает на разных компьютерах. Любой компьютер, подключенный к сети или к Интернету присваивается определенный "IP-адрес", мечаниями в формате xxx.xxx.xxx.xxx. Цифры различны для каждой машины в сети, но большинство компьютеров, подключенных к маршрутизаторов дома, как правило, в местной IP диапазона "192.168.xxx.xxx". Вы можете получить IP-адрес локального компьютера с помощью следующей РФО Basic код:

(быть законченным...)

14.1 Порты

При записи сетевое приложение, необходимо выбрать конкретный номер порта, через который данные должны быть переданы. Потенциальные порты в диапазоне от 0 до 65535, но многие из этих чисел зарезервированы для конкретных типов приложений (почтовые программы используют порт 110, веб-серверы используют порт 80 по умолчанию, и т.д.). Чтобы избежать конфликтов с другими установленными сетевыми приложениями, то лучше выбрать номер порта между 49152 и 65535 для малых скриптов. Список номеров зарезервированных портов доступназдесь.

14.2 серверы и клиенты

Программы "Сервер" открыть выбранную сетевой порт и ждать, пока один или более программ "клиент", чтобы открыть тот же порт, а затем вставьте в него данные. *Порт открыт программой сервера упоминается в клиентской программе путем объединения IP-адрес компьютера, на котором работает сервер, вместе с выбранным номером порта, отделенных друг от друга символом двоеточия* (т.е. 192.168.1.2:55555~~HEAD=pobj).

Следующий простой набор скриптов демонстрирует, как использовать РФО Основные функции передать одну строку текста от клиента к серверной программе. Этот пример предназначен для запуска на одном компьютере, для демонстрации, так что слово "локальный" используется для обозначения IP-адрес сервера (что это стандартное соглашение используется для обозначения собственной локальной IP-адрес любого компьютера). Если вы хотите запустить это на двух разных компьютерах, подключенных через локальную сеть, вам нужно получить IP-адрес сервера (машины использовать код, указанный выше), и заменить слово "локальный" с этим числом.

Вот программа-сервер. Будьте уверены, чтобы запустить *его, прежде чем* начать клиента, или вы получите сообщение об ошибке:

(быть законченным...)

Вот клиента. Запустите его в *отдельный* экземпляр интерпретатора Basic РФО, после выше программа была запущена:

(быть законченным...)

14.3 Блокировка и без блокировки петли

Как правило, серверы будут постоянно ждать, пока данные появляются в порту, и неоднократно что-то делать с этими данными. Сценарии ниже продлить выше пример с вечно петель постоянно отправлять, получать и отображать сообщения, передаваемые от клиента (ов) на сервер. Этот тип цикла образует основу для большинства сверстников-равному и сетевых приложений клиент-сервер. Введите "конец" в клиентской программе ниже, чтобы бросить и клиент, и сервер.

Вот программа-сервер (запустить его в первую очередь):

(быть законченным...)

Вот программа клиента. Запустите его только *после того, как программа-сервер был* запущен, а в отдельном экземпляре интерпретатора РФО Basic (или на отдельном компьютере):

(быть законченным...)

Важно понимать, что серверы, такие как один выше может взаимодействовать независимо с более чем одним одновременного подключения клиента. "Подключение" определение ждет, пока новый клиент не связывает, и возвращает порт, представляющий, что первое соединение клиента. После того, как это происходит, "соединение" относится к порт, используемый для приема данных, передаваемых по уже подключенного клиента. Если вы хотите добавить больше одновременных клиентских подключений во время навсегда петлю, просто определить другой "первый сервер ожидания". Попробуйте запустить сервер ниже, а затем запустить два одновременных экземпляры выше клиента:

(быть законченным...)

Вот пример, который показывает, как отправить данные назад и вперед (в обоих направлениях), между клиентом и сервером. Опять же, работать обе программы в отдельных случаях интерпретатора Basic РФО, и быть уверенным, чтобы начать первый сервер:

(быть законченным...)

Ниже короткий сценарий сочетает в себе многие из методов показали, до сих пор. Он может действовать либо как сервер или клиент, и может отправлять сообщения (по одному), туда и обратно между сервером и клиентом:

(быть законченным...)

Следующий сценарий является полным сеть приложением мгновенное сообщение. В отличие от FTP-чат номер представлены ранее, текст в этом приложении направляется непосредственно между двумя компьютерами, через подключение сетевого сокета (пользователи FTP чата не подключаться непосредственно друг с другом - они просто читать и писать в файл на общедоступной *третья сторона* FTP-сервер):

(быть законченным...)

Вот еще более компактная версия (возможно кратчайший программа обмена мгновенными сообщениями вы всегда будете видеть!):

(быть законченным...)

И вот расширенная версия этого скрипта, что загрузит выбранное имя пользователя, WAN / LAN IP, и номера портов на FTP-сервере, так что, что информация может быть передана другим онлайн (что позволяет им найти и соединиться с вами) , Подключение в сервер загружает данные пользователя и запускает приложение в режиме сервера. После того, как это будет сделано, другие могут нажмите на кнопку "Серверы", чтобы получить и вручную введите информацию о соединении (IP-адрес и порт) для подключения в качестве клиента. Используя различные номера портов и

имена пользователей, несколько пользователей могут подключаться к другим нескольким пользователям, в любом месте в Интернете:

(быть законченным...)

14.4 Переадресация портов и виртуальные частные сети

Если вы хотите, чтобы запускать скрипты как они между компьютерами, подключенными к Интернету, широкополосные маршрутизаторы, вы, скорее всего, нужно научиться "вперед" порты от маршрутизатора к IP-адрес машины, выполняющей программу сервера. В большинстве ситуаций, когда маршрутизатор подключается локальный домой / бизнес-сеть к Интернету, только устройство маршрутизатор имеет IP-адрес, который виден в Интернете. Сами компьютеры все назначенные IP-адреса, которые *доступны только в пределах локальной* сети. Портовое экспедирование позволяет отправлять данные, поступающие на IP-адрес маршрутизатора (IP-который виден в Интернете), на единственный порт, к конкретному компьютеру внутри локальной сети. Полное обсуждение переадресации портов выходит за рамки данного руководства, но это легко узнать, как сделать - просто введите "перенаправление портов" в Google. Вы должны будете узнать, как направить порты на *вашей марки и модели* маршрутизатора.

С любой конфигурации клиент-сервер, только сервер машина должна иметь открытую IP-адрес или открытый маршрутизатор / порт брандмауэра.Клиентская машина может быть расположен за маршрутизатором или брандмауэром без каких-либо переданных входящих портов.

Другой вариант, который позволяет сетевым приложениям работать через маршрутизаторы является обеспечение "VPN". Такие приложения, какHamachi, Comodo, и OpenVPN позволяют подключать два отдельных сетей LAN через Интернет, и лечить все машины, как будто они соединены локально (подключение к любому компьютеру в сети VPN с использованием локального IP-адреса, такие как 192.168. 1.xxx). Программное обеспечение VPN, как правило, также добавляет слой безопасности на данных, отправленных туда и обратно между подключенных машин.Обратной стороной программного обеспечения VPN является то, что передача данных может быть медленнее, чем прямой связи с использованием перенаправления портов (данные проходит через сервер третьей стороны).

14.4.1 Передача файлов через двоичный TCP сетевые сокеты:

(быть законченным...)

14.5 Walkie Talkie (VOIP)

Следующая программа является рации Push-To-разговоры тип голос поверх IP приложения. Это чрезвычайно просто - это просто записи звука с микрофона в формат .wav файл, затем передает звуковой файл в другой IP (где же программа работает), для воспроизведения. Отправитель и получатель открыт в отдельных процессах, и оба работают в бесконечных петель, чтобы позволить и обратно постоянную связь.

(быть законченным...)

14.6 Передача файлов Desktop

Вот небольшое приложение для передачи любой выбранный файл из настольных ОС для Android телефона, по WIFI. Выбранный файл может быть любой тип файла: текст или двоичный: изображение, звук и т.д. На вашем Android запустить следующий код клиента в РФО основной:

```
вход "Сохранить Супер", имя файла $, "test.jpg"
входные "Подключение к IP", IP-адрес $, "192.168.1.136"
вход "Номер порта", порт 2345
socket.client.connect IP $, порт
```

```
печать "Connected"
maxclock = часы () + 30000
делать
    socket.client.read.ready флаг
    если часы ()> maxclock
        печать "Процесс истекло. Окончание."
        конец
    ENDIF
пока флаг
byte.open Вт, FW, имя файла $
socket.client.read.file FW
byte.close FW
socket.client.close
печать "Готово"
```

Загружаемый .apk файл выше приложение доступно на http://laughton.com/basic/programs/filetransfer.apk

На рабочем столе сервера написано в REBOL. Есть версии REBOL для Windows, Mac, Linux, BSD, Solaris, QNX, AIX, HP-UX, Windows CE, BeOS, Amiga, и т.д., так что вы можете использовать этот код для передачи файлов между кучей различных ОС и Android , Просто зайдите на http://rebol.com и скачать интерпретатор REBOL для вашей ОС (это только около 1/2 МЭГ), и вставьте следующий код в консоли (или сохранить его в файле .r):

```
Rebol []
Единственная просьба-файл /: выбранный файл-
номер-порта: запрос-текст / название / по умолчанию "Номер порта:" "+2345"
печать вернуться [
    "Сервер начал" (читай присоединиться DNS: // читаем DNS: //) ":" номер-
порта
]
если ошибка? пытаться [
    не порт: сначала дождаться открытия / двоичный / не ждать присоединиться
TCP: //: номер-порта
] [уволиться]
Файл: чтение / двоичный файл выбран,
вставить файл порта
рядом порт
печать "Done" Остановка
```

.exe-Окна доступна на http://laughton.com/basic/programs/file_transfer.exe

15. Базы данных SQLite

Один из самых мощных функций BPO Basic является его встроенная поддержка для SQLite. SQLite является небольшой, автономный систему реляционных баз данных, что позволяет хранить, извлекать, поиск, сортировать, сравнивать и иным образом манипулировать крупных магазинов данных, очень быстро и легко. SQLite используется в качестве основной системы хранения данных в устройствах Android, и опирается на почти каждый другой современной вычислительной платформы (PC, Mac, Linux, картинки и другие мобильные устройства, приложения браузера Webkit основе, веб-серверы и т.д.). Вы можете легко скопировать и передать весь SQLite базы данных информации, созданной на Android, и работать с ними на устройствах с разными операционными системами, и визы наоборот.

Имейте в виду, что программирование и использование вычислительных устройств в целом, в конечном счете, об управлении *данными, так научиться использовать SQLite принципиально полезно стать способным РФО Основные программист.*

15.1 Таблицы

В SQLite и других баз данных, данные хранятся в "таблицы". Таблицы состоят из столбцов соответствующей информации. А "Контакты" стол, например, может содержать имя, адрес, телефон и столбцы рождения. Каждая запись в базе данных может рассматриваться как строки, содержащей данные в каждой из этих областей столбцов:

```
Название Адрес Телефон рождения
---- ------- -------- --------
Джон Смит 123 Toleen Лейн 555-1234 1972-02-01
Пол Томпсон 234 Джорджтаун Место 555-2345 1972-02-01
Джим Persee 345 Портман Пайк 555-3456 1929-07-02
Джордж Джонс 456 Topforge суд 1989-12-23
Тим Полсон 555-5678 2001-05-16
```

15.2 SQL

Заявления "SQL" позволяют работать с данными, хранящимися в таблицах базы данных. Некоторые заявления SQL используются для создания, уничтожения, и заполнить колонки с данными:

```
CREATE TABLE имя_таблицы% создать новую таблицу информации
DROP TABLE имя_таблицы% удалить таблицу
INSERT INTO имя_таблицы значения (значение1, значение2, ...)% добавить данные
INSERT INTO Контакты
    ЦЕННОСТИ (Билли Пауэлл ',' 5 Binlow доктор »,« 555-6789 »,« 1968-04-19 »)
Вставьте в Контакты (имя, телефон)
    ЦЕННОСТИ ('Роберт Инграм', '555-7890')
```

Следующий код SQL создаст таблицу Контакты показано выше:

```
Вставьте в Контакты ЦЕННОСТЕЙ
    («Джон Доу», «1-стрит Лейн», «555-9876», «1967-10-10»),
    («Джон Смит», «123 Toleen Лейн», «555-1234», «1972-02-01»),
    ("Пол Томпсон", "234 Джорджтаун Пл. ',' 555-2345 ',' 1972-02-01 '),
    (Джим Persee ',' 345 Портман Пайк ',' 555-3456 ',' 1929-07-02 '),
    ('Джордж Джонс »,« 456 Topforge суд', '', '1989-12-23'),
    ('Тим Полсон', '', '555-5678', '2001-05-16')
```

Оператор SELECT оператор используется для извлечения информации из столбцов в данной таблице:

```
ВЫБОР column_name (ы) FROM table_name
SELECT * FROM Контакты
Выбрать Название, адрес Контакты ОТ
SELECT DISTINCT рождения из Контактов% возвращается нет дублированные записи
```

Для выполнения поиска, использовать ГДЕ. Приложите текст для поиска в кавычки и использовать следующие операторы:; =, <>,>, <,> =, <=, BETWEEN, LIKE (использование "%" для подстановки):

```
SELECT * FROM Контакты Где имя = 'Джон Смит "
SELECT * FROM Контакты где имя LIKE 'J%'% любое имя, начиная с "J"
SELECT * FROM Контакты откуда День рождения LIKE '% 72%' ИЛИ телефон LIKE '%
34'
SELECT * FROM Контакты
    ГДЕ НЕ рождения между "1900-01-01» И «2010-01-01»
```

В позволяет задать список данных соответствуют в колонке:

```
SELECT * FROM Контакты Где телефон В ("555-1234", "555-2345")
SELECT * FROM Контакты Заказать по результатам имя% сортировать в алфавитном
порядке
ВЫБОР имя, дата рождения из Контактов ORDER BY рождения, имя убыванию
```

Другие SQL заявления:

```
UPDATE SET Контакты адрес = '643 Pine Valley Rd.
    ГДЕ имя = 'Роберта Ингрэм'% изменять или добавлять к существующим данным
DELETE FROM Контакты Где имя = 'Джон Смит "
DELETE * FROM Контакты
ALTER TABLE% изменить структуру столбца таблицы
CREATE INDEX% создать ключ поиска
DROP INDEX% удалить поисковый ключ
```

Есть много учебников доступны в Интернете, чтобы помочь выучить язык SQL, и SQLite, в большей глубине. Взгляните наhttp://zetcode.com/databases/sqlitetutorial для получения дополнительной информации.

Чтобы открыть базу данных SQLite в, используйте следующий код. Имя файла может быть любым выбрать, и все остальные параметры являются необязательными. Если файл базы данных не существует, она будет создана:

```
дБ = SqlOpenDatabase ("myData.db", "1.0", "Мои данные")
Если DB = 0 Then MsgBox "Ошибка открытия базы данных"
```

Команды SQL должен быть отформатирован в массив. Вы можете определить свои собственные функции для обработки успешных и неудачных команд SQL. Функция dbSuccess ниже очень важно. Он демонстрирует, как использовать цикл для извлечь и отображения данных из команды SQL SELECT:

(быть законченным...)

Для выполнения команды SQL, используйте функцию SQL, с ID базы данных и массива командной SQL в качестве параметров:

(быть законченным...)

Вот полный пример, который открывает базу данных, создает новую "MyData" таблицу, вставляет данные в таблицу, а затем извлекает и выводит все содержимое таблицы:

(быть законченным...)

16. Отладка

РФО Основные имеет ряд функций отладки, которые можно использовать, чтобы помочь найти и исправить ошибки с кодом. Отладочные функции могут быть использованы для печати содержимого всех существующих скалярных переменных, массивов, списков и т.д. Вы можете разместить отладки функций на протяжении всего кода, а затем использовать debug.on и debug.off функции активации и деактивации все отладки деятельность однажды. В debug.show .___ функции особенно полезны. Функция debug.show.scalars показывает все существующие переменные и их значения, и ждет для взаимодействия с пользователем, прежде чем продолжить программу:

```
debug.on
x = 1
y = 2
debug.show.scalars
r = 3
debug.show.scalars
конец
```

Есть аналогичные debug.show.list, debug.show.array и другие функции, чтобы показать и посмотреть все возможные структуры данных в коде.

При написании графические приложения, функция gr.front в первую очередь должны быть использованы для переключения в режим консоли, то функции отладки можно запустить, то gr.front и gr.render функции должны быть запущены, чтобы вернуться к графическом режиме:

```
gr.front 0
debug.on
debug.show.scalars
gr.front 1
gr.render
```

17. Строки

А "строка" является просто набором символов. Взгляните на следующие примеры, чтобы увидеть, как это сделать несколько общих текстовых операций:

(быть законченным...)

18. Краткий обзор и резюме

В списке ниже приведены некоторые ключевые характеристики BPO Базовая языке. Зная, как положить эти элементы, чтобы использование является фундаментальное понимание того, как работает РФО Основные:

1. Для начала, РФО Основные имеет много встроенных функций, которые выполняют слова общих задач. Как и в других языках, служебные слова, как правило, с последующим параметров данных. Параметры помещаются сразу после функции слова и, как правило, через запятую. Для достижения желаемой цели, функции расположены последовательно один за другим. Значение, возвращенное одной функции часто используется в качестве аргумента входа к другой функции. Линейные терминаторы не требуются в любой момент, и все выражения вычисляются в порядке слева направо, а затем вертикально вниз через код. Вы можете завершить важную работу, просто зная, предопределенные функции в языке, а также организации их в полезный порядке.
2. Пусто (пробелы, табуляции, новой строки, и т.д.) могут быть вставлены по желанию, чтобы сделать код более читаемым. Тильда используется символ продолжать код на несколько

линий. Текст после знака процента и перед новой линии рассматривается как комментарий. Восклицательные точки комментировать целые линии. Двойные восклицательных знаков закомментировать несколько строк.

3. РФО Основные содержит богатый набор условных и сквозными структур, которые могут быть использованы для управления потоком программы и деятельность по обработке данных. Если во время / повтор, не делать / до, для и другие типичные структуры поддерживаются.
4. РФО Основные может увеличить, сравнивать и выполнять вычисления между элементами в списках, используя для петель. Данные любого типа можно записать и читать из файлов, чтобы SQLite базы данных, или отправлены и получены в / из интернет-серверов, либо FTP, по http.post, и прямых связей сокетов.
5. Любые данные могут быть назначены метку переменной. Знак равенства ("=") используется для назначения метки переменных слов и их значений. Многие функции включают в себя обратный переменную сразу после функции слова, в списке параметров функции. После назначения переменных слова могут быть использованы для представления всех данных и / или действий, содержащихся в данной строке, массив и т.д.
6. Несколько штук данных (списки) хранятся в массивах, списках и расслоений. Элементы могут быть добавлены или взял из списков по номерам индекса. Вы можете объединить списки в последовательную форму строк и сохраняется в файл, а также списки данных также могут быть сохранены в таблицах базы данных. Функция "раскол" может быть использован для преобразования строк обратно в списки.
7. Вы можете использовать графические экран не только отображать изображения и графического дизайна, но и представить полезные макеты GUI.

19. Построен в справке и документации

20. Дополнительные темы

(в процессе строительства)

21. реальном мире ПРИМЕРЫ - Обучение думать в кодексе

В этот момент, вы видели наиболее существенные биты BPO базовый синтаксис языка и API, но вы, вероятно, по-прежнему говорите себе "это здорово ... но, как я пишу полную программу, которая делает _____". Для материализуется любой рабочий программное обеспечение от воображаемой конструкции, это, очевидно, важно знать, какие языковые конструкции имеются построить куски программы, но "думать в коде" просто так о организации эти биты в более крупные структуры, зная, с чего начать, и, будучи в состоянии сломать процесс в управляемую, повторяемой рутины. Этот раздел предназначен, чтобы обеспечить некоторую общее понимание о том, как преобразовать концепции дизайна человека в BPO Basic кода, и как организовать свой рабочий процесс, чтобы подойти к любой уникальной ситуации. Тематические исследования представлены для понимания того, как конкретные жизненные ситуации в реальном были удовлетворены.

21.1 обобщенного подхода с использованием очертаний и псевдокод

Программное обеспечение не практически никогда не возвращается к жизни в какой-либо форме первоначально завершена. Это, как правило,*развивается* через несколько изменений, и часто развивается в направлениях первоначально непредвиденным. Там нет идеальный процесс для достижения окончательных проектов с нуля, но некоторые подходы, как правило, оказаться полезным. Имея план атаки то, что получает вы начали писать строки 1 из вашего кода, и это в конечном итоге обеспечивает то, что рабочий кусок программного обеспечения для устройств вашего пользователя. Вот обобщенная процедура, чтобы рассмотреть:

1. Начните с *детального определения того, что следует сделать* приложение, в человеческих терминах. Вы не получите в любом месте в процессе проектирования, пока вы не можете *описать* некоторую форму воображаемого окончательной программы. *Запишите* ваше объяснение и конкретизировать детали мнимой программы как можно больше. Включить как много деталь как возможно: как пользователь взаимодействует с ней, какие данные будут его принять, процесс, и вернуться, и т.д.

2. Определить список общих кода и структур данных, связанных с каждой из целей выше "человека описанной" программы. Подвести итоги каких-либо общих закономерностей кода, которые относятся к работе каждой воображаемой компонента программы. Подумайте о том, как пользователь будет получать данные в и из программы. Будет работать пользователь с списков выбора, полей ввода текста, и т.д.?Рассмотрим, как *данные, используемые в программе могут быть представлены в коде, организовано, и манипулировать.* Какие типы данных будут участвовать (текст типы, такие как строки, значения времени, или URL, двоичные типы, такие как изображения и звуки, и т.д.). Может программный код потенциально использовать переменные, список структур и функций, и т.д.? Будут ли эти данные храниться в локальных файлах, в удаленных файлов, в базе данных, или просто во временной памяти (переменные), и т.д.? Подумайте о том, как программа будет течь от одной операции к другой. Как куски данных должны быть отсортированы, сгруппированы и связаны друг с другом, какие типы условных и сквозными операций должны быть выполнены, какие типы повторяющихся функций должны быть изолированы и закреплены в функций, подпрограмм и т.д.? Рассмотрим все, что предназначено для *случиться* в воображаемой части программного обеспечения, и начать думать, "_this_, как я мог потенциально выполнить _that_, в коде ...".
3. *Начните писать код* контур. Это часто проще всего сделать это, рассказав взаимодействие с пользователем, а блок-схема операций может быть полезно тоже. Идея здесь, чтобы начать писать обобщенный код контейнер для рабочей программы. В этот момент, контур может быть заполнен простым естественном языке псевдокода, который *описывает, как фактический код может быть организован.*Начиная с контуром взаимодействия с пользователем особенно полезно, потому что это является отправной точкой на самом деле написать крупные структуры кода, и это заставляет вас заниматься, как программа будет обрабатывать входной, манипуляции, и вывод данных. Простые структуры, такие как "вариант меню (что это, когда нажата ...)", "список: (с наклейками и подсписков организованных как это ...)," функция: (который перебирает этого блока и сохраняет эти элементы другой переменной ...) "может быть конкретизирован позже с полным кодом.
4. Наконец, перейдем к замене псевдо код с фактическим рабочим кодом. Это не так сложно, как только вы закончили предыдущие шаги.Ссылка API может быть очень полезным на данном этапе. И как только вы действительно знакомы со всеми доступными конструкций в языке, все, что вам, скорее всего, нужно это иногда напоминает синтаксис из справки, RFO BASIC в. В конце концов, вы будете проходить через другие стадиях проектирования гораздо более интуитивно, и добраться до / через этот этап очень быстро.
5. В качестве последнего шага, отладки вашей рабочей кода и добавить / изменить функциональность, как вы проверить и использовать программу.

Основной план атаки всегда объяснить себе, что предназначены программа должна делать, с точки зрения человека, а затем продумать, как все необходимые структуры кода должна быть организована для достижения этой цели. Как воображаемая программа принимает форму, организовать свой рабочий процесс с использованием подхода сверху вниз: представить концепцию -> общий контур -> псевдокод описание / мыслительный процесс -> рабочий код -> готовый код.

Большинство код, написанный будет перетекать из одного пользовательского ввода, определение данных или функции внутреннего к другому.Начните отображение все то, что нужно, чтобы "произойдет" в программе, и данные, которые должны быть манипулировать по пути, для того, чтобы эти вещи происходят, от начала до конца. Процесс написания план может помочь, думая о том, как должна начаться программа, и то, что должно быть сделано до того, как пользователь начинает взаимодействовать с приложением. Подумайте о каких-либо данных или действий, которые должны быть определены до начала программы. Затем подумайте о том, что должно произойти, чтобы приспособить каждый возможный взаимодействие пользователь может выбирать. В некоторых случаях, например, все возможные действия могут происходить в результате выбора пользователем из списка опций меню. Это должно вызвать мысль о некоторых битов выбора структуры функции, и вы можете начать писать код схему, чтобы реализовать эти решения.

Независимо от задуманного интерфейс, думаю, все выборы пользователь может сделать в любой момент времени, а также обеспечить компонент пользовательского интерфейса, чтобы для этих выборов. Затем подумайте о всех операциях компьютер должен выполнить, чтобы реагировать на каждое выбору пользователя, и описать то, что должно произойти в коде.

Как вам решать каждую строку кода, используйте естественного языка псевдо-код, чтобы организовать свои мысли. Например, если представить, выбор меню делают что-то для вашего пользователя, вы не должны сразу написать РФО Basic код, что выбор меню работает.Первоначально, только написать *описание* того, что вы хотите сделать выбор. То же самое справедливо для функций и других участков кода.Как вы конкретизировать свой план, ***описать*** *языковые элементы и кодирования думал, что ты себе для выполнения различных действий или представляют различные структуры* данных. Дело писать псевдокод, чтобы сохранить четко сконцентрирована на общий дизайн программы, на каждом этапе процесса разработки. Делать это поможет вам не заблудиться в серцевины Подробнее о синтаксисе фактического кода. Это легко потерять из виду общую картину, когда вы попали в письменной форме каждой строки кода.

Как вам конвертировать псевдо код мысли синтаксиса языка, помните, что большинство действий в программе возникают в результате условных оценок (если это произойдет, это сделать ...), петли, или линейный поток от одного действия к другому. Если вы собираетесь выполнять определенные действия несколько раз или цикл через списки данных, вы, вероятно, нужно запустить через некоторые петли. Если вам нужно работать с изменяемыми данными, вам необходимо определить некоторые метки переменных, и вы, вероятно, нужно передать их функции для обработки данных. Подумайте в этих общих условиях в первую очередь. Создайте список данных и функций, которые требуются, и положить их в порядке, что делает структуру программы сборки и поток от одного определения, условия, петли, выбор меню, действия и т.д., чтобы в следующий.

21.2 Случай 1 - Планирование Учителя

21.3 Случай 2 - дней между двумя датами калькулятор

21.4 Случай 3 - Простой поиск

21.5 Случай 4 - калькулятор

21.6 Случай 5 - Резервное копирование Генератор Музыка ("Джем Инструмент")

21.7 Случай 6 - FTP-инструмент

21.8 Случай 7 - Опасность

21.9 Случай 8 - Тетрис

21.10 Дело 9 - Media Player

21.11 Дело 10 - Веб-сайт пауков приложение

(остальная часть этой секции находится в стадии строительства)

22. Другие скрипты

22.1 Количество Verbalizer

22.2 Бинго

22.3 Голос Сигнализация

(остальная часть этой секции находится в стадии строительства)

23. Обратная связь

Для обратной связи, ошибки отчетов, предложений и т.д., пожалуйста, напишите Ник на сайте:

```
назад (МОС Тод Пуск в эксплуатацию МПЦ-та cisab)
```

Чтобы нанять автора для обучения, для разработки программного обеспечения или веб-сайта приложения, или для общих консультаций / вычислительной поддержки, см http://com-pute.com или позвоните 267-352-3625.

Ключевые слова:

Android развития, создать программы, разработка программного обеспечения, научиться программировать телефон, как писать приложения, узнать мобильной программирование, легкий кодирование, как создавать приложения, научиться писать код, программирование учебник телефона, курс программирования, узнать о программировании, кодирование , самый простой способ создания приложений, простое программирование приложение, лучший язык программирования мобильного, простой язык программирования с мобильного, начать программирование мобильных приложений