**Мини-учебник по MIDlet Pascal для чайников.**

Итак, это мое творение мини-учебника, по программе MIDlet Pascal, для чайников. Я уж постараюсь объяснить, как можно понятнее.

Что же такое MIDlet Pascal? Среда разработки игр и приложений для мобильных телефонов. Классно, не правда ли?

Коротко о программе: MIDlet Pascal – это среда разработки приложений для мобилы (как я и говорил). В программе используется язык программирования, схожий с языком Паскаль. Паскаль – язык программирования для компьютеров, разработанный Никлаусом Виртом (не помню в каком году). Тот, кто окончил какое-нибудь учебное учреждение, помнят, что проходили этот язык по информатике (может я и ошибаюсь). Язык MIDlet Pascal является обычным паскалем, но сильно урезанным, т.е. разработчики убрали некоторые операторы, которые были в обычном паскале. Не обязательно вдумываться в предыдущее предложение (операторы там всякие), мы разберем это все в течение учебного пособия. MIDlet Pascal переводит программу из языка Паскаль в язык Java. В конце концов, мы можем получить готовый мидлет (приложение, игру). Думаю, что многие знают, что игры для телефона пишут на другом языке, который называется Java или J2ME. Java, конечно превосходит по возможностям язык MIDlet Pascal, и многие бы хотели программировать на Java. Язык java сложен в изучении и поэтому, тот, кто не знаком ни с одним языком, вряд ли сможет понять язык Java. Для Java нужно качать не один мегабайт, то есть нужно опять же качать среду разработки, которая обойдется около ~120 мегабайт (JRE, JDK, Wireless Toolkit). MIDlet Pascal занимает гораздо меньший объем памяти (не более 2-х мегабайт). Взять программу MIDlet Pascal можно с нашего сайта: <http://blackwolf.h2m.ru> в разделе загруз-центр => Софт для компьютера (или что-то в этом роде), там есть русская и английская версия. Так же можно взят и с официального сайта программы: <http://www.midletpascal.ru> . На рисунке 1 показан общий вид программы.

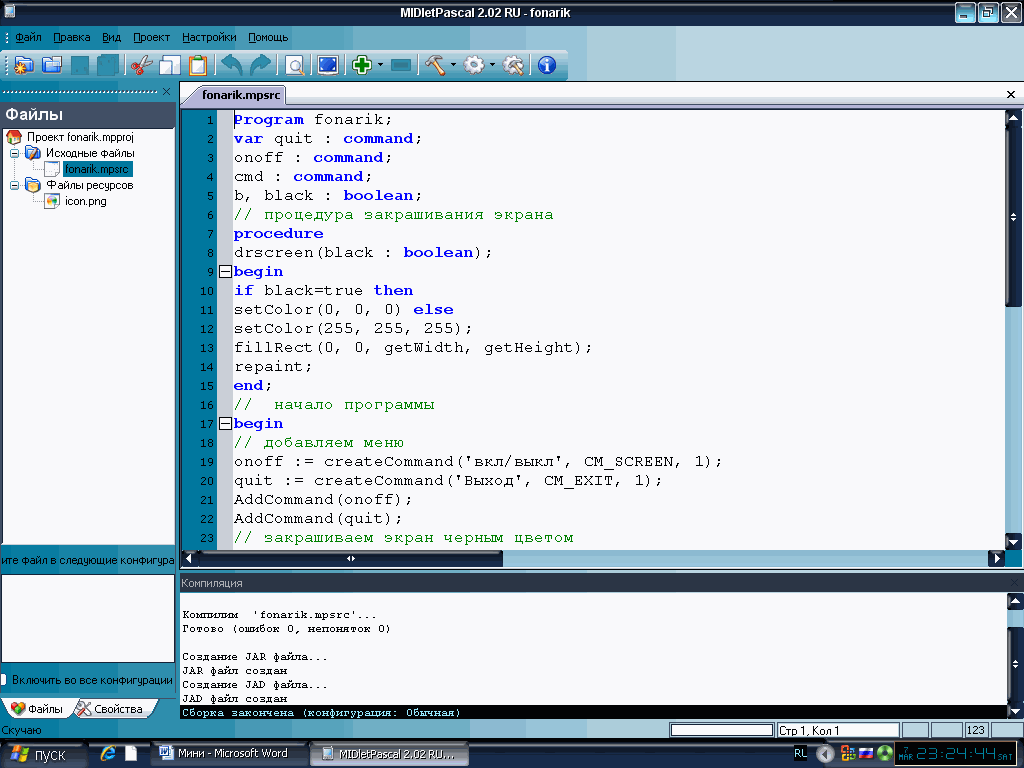


Рис. 1. Среда разработки игр и приложений MIDlet Pascal.

Вот такой вот симпатичный интерфейс имеет эта программа.

И чуть не забыл сказать, что нам еще потребуется для написания наших проектов. А понадобятся нам:

- MIDlet Pascal;

- Компьютер (без него никуда);

- Эмулятор мобильного телефона;

- Собственно телефон;

- USB-кабель для Вашего телефона;

Ну, вроде бы пока всё.

Ага! Остановимся на самом интересном! На эмуляторе мобильного телефона. Что же это такое? Это программа, которая может запускать Java приложения. Для чего он нам? Для отладки (проверке, тесте) созданного нами приложения или игры. А как же телефон? Я думаю, что неудобно все-таки каждый раз загружать приложение через USB-шнур, т.к. это займет достаточно много времени, поэтому лучше использовать эмулятор. И прогу эмулятор тоже можете взять с нашего сайта <http://blackwolf.h2m.ru> в том же разделе с названием MidpX.

На рисунке 2 показан вид эмулятора телефона.

На экране Вы видите запущенное приложение, правда, не вписанное во весь экран. Это одна из моих работ, которую я назвал J2ME WinMobile, то есть эмулятор компьютера на телефоне. В данный момент я ее разрабатываю, но это не важно.

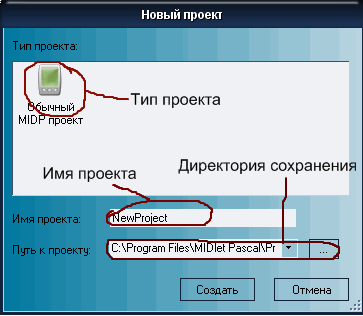


Если вы все скачали, то можете приступить к созданию Вашего проекта.

У меня стоит русская версия MIDlet Pascal, и поэтому я могу, где-то ошибаться на счет неправильного пункта меню (у кого английская версия).

Начнем с создания нашего проекта, т.е. зададим имя нашего мидлета, выберем путь для сохранения проекта.

Все это делается командой File => New Project. После всех этих трудоемких действий появится небольшое окошко (диалог), которое предложит Вам задать параметры проекта, а конкретно имя проекта, директорию для сохранения всей Вашей работы и тип проекта (он там один). Все это есть на рисунке 3.



Допустим, оставим все эти параметры по умолчанию и нажмем кнопку Создать (Create).

Рис. 2. Эмулятор мобильного

телефона MidpX

Перед нами откроется

рабочая область приложения (текстовый редактор), в котором будет написан многим непонятный код. Этот код и есть код программы на MIDlet Pascal. В общем, он должен быть примерно таким:

**Program** NewProject;

**Begin**

drawText(‘Hello, world!’, 5 ,5);

repaint;

delay(5000);

**end**.

Что это за код? Это код самой простейшей программы на MIDlet Pascal. При запуске этого приложения на эмуляторе или телефоне Вы увидите лишь обычную надпись “Hello, world!”, в переводе означает привет мир (все программисты начинают с этой фразы). Я думаю, что пока этот код мы разберем в следующей главе, так как многие его не поймут. Мой совет: лучше не копируйте примеры из этого учебника ибо получите кучу ошибок. Лучше переписывать код вручную, таким образом Вы быстрее научитесь.

Кстати я ж не сказал об том, что, где и в каких папках у нас находится. Итак, если Вы догадались, как скомпилировать (собрать) приложение, в папке с установленным MIDlet Pascal, мы найдем папку с названием “Projects”, в этой папке должен находится Ваш проект “NewProject” (если вы его не сохраняли куда-нибудь в другое место). В папке “NewProject” находится еще четыре папки и один файл проекта:

- **bin** – здесь будут лежать мидлеты, после их сборки.

- **res** – здесь ресурсы вашего приложения (картинки, мелодии, текст и др.).

- **classes** – здесь будут лежать классы (рабочие модули) вашего приложения.

**- src** – а тут исходники (код) вашего проекта.

Если вам показался код выше сложным, то взгляните на код написанный на Java:

**import** javax.microedition.midlet.MIDlet;

**import** javax.microedition.lcdui.Display;

**import** javax.microedition.lcdui.Form;

**public** **class** NewProject **extends** MIDlet {

Form form;

Display display;

**public** startApp() {

display = Display.getDisplay(**this**);

form = **new** Form(“Форма”);

form.append(“Hello, world!”);

display.setCurrent(form);

}

**public** pauseApp() {}

**public** destroyApp(**Boolean** flag) {

notifyDestroyed()

}

}

А вот Вам код на java. Ну и как? Это практически тоже самое, только текст уже выводится в другой области, называемой формой (об этом позже).

**Глава 2. Hello, World!**

Как я и обещал мы сейчас попробуем разобраться с предыдущем примером, если Вы его забыли или не захотели смотреть на него, то вот он ниже:

**Program** NewProject;

**Begin**

drawText(‘Hello, world!’, 5 ,5);

repaint;

delay(5000);

**end**.

Начнем с первой строки. Итак, слово **program -** это слово обозначает имя программы или ее заголовок. Для чего нужно это слово? Да если так то оно здесь и ненужно вовсе, даже без него приложение будет отлично работать. Но если уж Вам охота узнать об этой строчке поподробнее, то я объясню, что, где что. Слово **program** это зарезервированное слово (зарезервированные слова выделяются синим цветом), которое объявляет программе заголовок. Напишу примерный синтаксис слова **program**.

**Program** имя\_программы;

Именем программы может быть любым, но не должно называться каким-нибудь другим зарезервированным словом, процедурой или функцией языка. Например:

**Program begin**;

Этот пример вызовет ошибку, поэтому лучше использовать такие имена, например: MyProgram, Calculator, Game и прочие. Обратите внимание на символ точка с запятой (;). Этот символ разделяет операторы друг от друга. В данном случае строка **Program NewProject**; является оператором.

Теперь перейдем к следующей строке **begin.** Это зарезервированное слово нужно для образования парных операторных скобок. Слово **begin** (начинать) используется в паре со словом **end** (закончить), таким образом, они образуют операторные скобки, внутрь которых помещается другие операторы, которые воедино надо выполнить. Синтаксис операторных скобок такой:

**Begin**

Операторы1;

Операторы2;

ОператорыN;

**End.**

В языке Java, вместо операторов begin и end существуют фигурные скобки: begin – это {, а end – это }. Так что на Java пример выше записывается так:

{

Операторы1;

Операторы2;

ОператорыN;

}

Кстати на MIDlet Pascal, код можно писать, не учитывая регистр букв, т.е. все равно как ты написал слово (с большой или маленькой буквы), например **Begin**, **begin** **BEGIN** или **BeGiN** – все это одно и тоже. Java в отличие от MP(MIDlet Pascal), к регистру чувствителен. Перейдем к следующей строке: *DrawText*(‘Hello, World!’, 5, 5);

Оператор *DrawText* рисует надпись Hello, World! на экран мобильного телефона. Как вы догадались, что надпись, которую нужно отобразить на экране указывается внутри одинарных кавычек (апостроф). Приведу пример использования оператора *DrawText*:

DrawText(‘Ваш\_текст’, координата\_X, координата\_Y);

Строка “Ваш\_текст” в операторе *DrawText* должна быть обязательно внутри апостроф, но не двойных кавычек:

DrawText(“Ваш\_текст”, координата\_X, координата\_Y);

Это будет являться кучей ошибок. Так что будьте внимательны.

Теперь узнаем что это за координата\_X и координата\_Y. Это координаты размещения надписи на экране в пикселях. Координата\_Х – это координата строки по горизонтали, а координата\_Y – это координата строки по вертикали. Таким образом, они образуют систему координат. Все координаты указываются в цифрах. На рисунке, 4 показано, как надпись будет располагаться на экране мобильного телефона (конечно без системы координат). Координаты по горизонтали идут слева на право, а

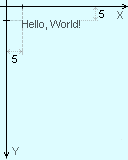


Рис.4. Отображение текста координаты по вертикали идут сверху вниз. Запомните это.

на экране мобильного теле- Теперь перейдем к следующей строке: Оператор *Repaint*. Этот

фона оператор обновляет экран и отрисовывает все графические элементы. Если мы не поставим этот оператор после оператора *DrawText,* то при запуске программы, мы увидим лишь белый экран. Объясню, почему это все так. Дело в том, что когда мы вызываем процедуру *DrawText*, текст который мы хотим отобразить рисуется в буфер экрана, а процедурой *Repaint* мы перерисовываем его на дисплей телефона. Старайтесь применять эту процедуру реже, так как эта процедура довольно таки расходует ресурсы телефона и нагружает его процессор. Например если вам нужно написать две строки *Hello, World!* А на другой *Привет* *мир!* То многие из вас бы написали код таким вот образом:

**Program** NewProject;

**Begin**

drawText(‘Hello, world!’, 5, 5);

repaint;

drawText(‘Привет мир!’, 5, 15);

repaint;

Delay(5000);

**End.**

Этот код не приведет к ошибке, но все же он неправильный, гораздо лучше было бы если мы запишем его так:

**Program** NewProject;

**Begin**

drawText(‘Hello, world!’, 5, 5);

drawText(‘Привет мир!’, 5, 15);

repaint;

Delay(5000);

**End.**

Пора переходить к следующей строчке. Оператор *Delay*(2000);. Этот оператор задерживает нашу программу на указанное количество миллисекунд. Для чего он нужен? Да если бы мы не поставили его, то в примере выше при запуске программы на эмуляторе или телефоне мы бы ничего не успели посмотреть, так как программа сразу бы при запуске закрылась. Поэтому мы и задерживали программу процедурой *Delay ,* чтобы мы хоть могли увидеть что происходит на экране мобильника. Синтаксис оператора *Delay* такой:

Delay(Время\_в\_милисекундах);

Последнюю строку я объяснять не буду так как я объяснил ее при рассмотрении ключевого слова **begin**.

Исходный код программы Hello, World! можете скачать [отсюда](http://blackwolf.h2m.ru/MP/src/HelloWorld.zip).

**Глава 3. Компиляция проекта и прочие настройки.**

Итак, я ж совсем ни слова не сказал о том как компилировать проект. Ладно, сейчас объясню. Чтобы скомпилировать ваш исходный код программы, нужно нажать в панели инструментов на кнопочку с изображением молотка или нажать на клавиатуре F7.



После этого через некоторое время появится окно с

оповещением об удачной компиляции или ошибке.

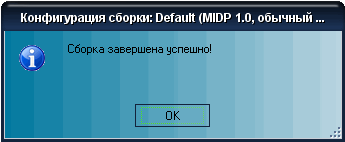
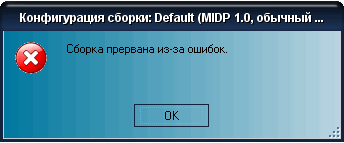


Рис. 7. Сборка завершилась успешно Рис. 8. Сборка завершилась неудачно

Вот на рисунках 7 и 8 изображены результаты компиляции. Если сборка завершилась с ошибками, то ниже в консоли можете глянуть, где ошибка.

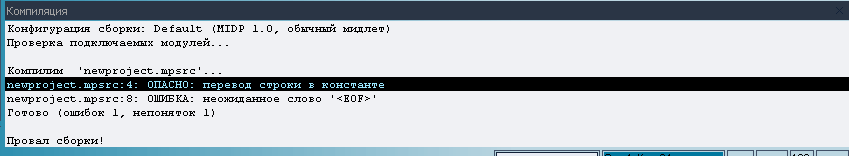


Рис. 9. Консоль сообщений о компиляции.

Так же в консоли можно два раза щелкнуть по сообщению об ошибке, тем самым выделится строка с ошибкой в редакторе кода.

Теперь о запуске скомпилированного приложения. Напомню, что скомпилированные приложения находятся в директории Вашего проекта в папке bin. Если сборка завершилась удачно, то в этой папке появятся два файла: архив с расширением \*.jar и еще один файл с расширением \*.jad. Jar – это и есть само приложение, которое можно загружать в телефон, а файл Jad – это дескриптор приложения и ссылка на Jar-файл. Файл jad нужен для установки приложения, т.к. некоторые не могут установить приложение через файл JAR (Телефоны Samsung, Philips, Sagem и др.). Файл Jar можно открыть любым архиватором, который поддерживает архивы ZIP (по сути, файл JAR – это и есть ZIP-архив).

**Глава 4. Переменные и константы.**

Теперь ознакомимся с такими важными частями языка, такие как переменные и константы. Начнем с констант.

Константа – это величина, которая не изменяется в течение выполнения программы. Вспомним, например число pi (π), или число Авогадро. Это обычные константы, которые не меняются, так же и в программе. Константы могут содержать числа и строки. Чтобы объявить константу – применяют зарезервированное слово const. прежде чем идти дальше, я предлагаю ознакомиться со структурой программы. Структура программы представлена ниже в виде кода:

**Program** заголовок\_программы;

**Const** список констант;

**Type** типы;

**Var** список переменных;

Подпрограммы функций и процедур;

**Begin**

Операторы;

**End.**

Итак, посмотрели структуру программы? Хорошо! Идем далее. Не следует пока забивать голову всем этим кодом. Эта структура показывает, в каком порядке должно, что-либо объявляться.

Вернемся к константам. Вспомните пример программы Hello, world! Вспомнили? Отлично! А теперь попробуем записать этот пример с использованием констант.

**Program** HelloWorld;

**Const**

str = ‘Hello, world!’;

**Begin**

DrawText(str, 5, 5);

Repaint;

Delay(5000);

**End.**

Программка несколько усложнилась. Для начала изучим синтаксис оператора **const**:

**Const** имя\_константы = значение;

В примере выше в качестве имени константы используется слово **str**. Можно использовать любое имя для константы, но оно должно быть на латинице, не должно начинаться с цифр, не должно содержать пробелов. Это самое главное.

Пример имен констант: val, chislo, loh, urod2 и тд. После присвоения имени константы какого-нибудь значения, его имя можно использовать в качестве его значения. Например если мы присвоим имени константы val число 5 то можно потом использовать имя val в качестве числа. Например:

Оригинальный пример: **A := 5 + 5;**

В этом примере используется переменная которая записывает в себя результат вычисления примера 5+5. То есть переменная получает значение 10. А вот как можно записать это используя константу:

**Const**

Val = 5;

**Var**

A:**integer**;

**Begin**

A:=val+val;

**End**.

Пока не обращайте внимание на слова var, integer о них чуть позже.

Вернемся к примеру:

**Program** HelloWorld;

**Const**

str = ‘Hello, world!’;

**Begin**

DrawText(str, 5, 5);

Repaint;

Delay(5000);

**End.**

Здесь процедура DrawText использует значение константы **str** в качестве строки. Так чтобы использовать в качестве параметра строку или число в функции или процедуре нужно поместить имя константы в параметр этой функции или процедуры. Константы в MIDlet Pascal могут быть одним из трех типов: целые числа, дробные числа и строки. Примеры записи констант всех трех типов:

**Const**

Chislo = 10;

RealChislo = 0.569;

Str = ‘Строка’;

Теперь поговорим о переменных. Переменные практически тоже самое что и константы. Отличаются они лишь тем, что могут изменять свои значения. Без переменных не было бы ни одного языка программирования. Для чего нужны переменные? Они нужны например для передачи значения другой подпрограмме. Например захотели вы создать калькулятор. Ну какой калькулятор без переменных? Числа ведь нужно во что-то записывать! Для того чтобы сложить два разных числа нужно объявить три переменные: для первого числа (пусть будет буква A), для второго числа (буква B) и результат (буква С). Таким образом для сложения числа получится вот такая формула: C:=A+B;

Вспомнили математику? Отлично! Пусть для переменной A мы ввели значение 6, для B – 5, а переменная С получает ответ, т.е. число 11. Думаю маленько понятно?

Итак, для того чтобы использовать переменную, ее нужно сначала объявить. Эту операцию мы делаем с помощью слова **var**, далее пишем ее имя (правила написания имени такие же, как и у констант) и указываем ее тип (число, строка и др.). Синтаксис объявления переменной:

**Var** имя\_переменной : **тип\_переменной**;

Например:

**Var** a: **integer**;

Если у Вас несколько переменных одного типа, то можно их записать по порядку, разделяя их запятой:

**Var** a, b, c : **integer**;

В MIDlet Pascal существуют 10 простых типов и 2 комплексных. Вот они все:

Простые:

* boolean
* char
* integer
* real
* string
* http
* resource
* image
* recordStore
* command

Комплексные:

* record
* array

И несколько не реализованных, один из которых тип file.

Далее буду объяснять типы этих данных. Начнем с первого типа boolean.

Тип boolean – этот тип называется так же булевым типом. Переменные типа boolean могут иметь только два значения true(истину) и false(ложь). Для чего же нужен этот тип? Ну например нужно сделать так - например переменная имеет значение true то включить в программе что-либо, например звук в плеере. То есть если переменная имеет значение true, то оставить звук включенным, если переменная имеет тип false, то звук выключить. Думаю что ясно. Все это рассмотрим на практике.

Тип char – этот тип называется символьным. Переменная, имеющая этот тип, может содержать в себе только один символ. С переменными этого типа можно делать такие операции, как конкатенация (сложение) +, сравнение >,<,<=,>=,=. Что же будет если мы сложим два символа? Ну например одной переменной присвоим значение ‘a’, а второй переменной – значение ‘b’, и если использовать процедуру drawText, то на экране мы увидим строку ab. Т.е. a и b сложились вместе. И заметьте что тип у строки ab будет уже string, так как переменная, имеющая тип char, может содержать в себе только один символ.

При операциях сравнения, переменная уже будет иметь тип boolean. Например: C:=’A’>’B’, здесь C получит значение false, так как A меньше (она стоит перед B), чем B.

Тип integer – это целочисленный тип, переменные которого могут иметь целое число или идентификатор объекта (текстового поля, шкалы и др.). Переменные этого типа могут иметь численное значение от -**2147483647** до **2147483647**. Нехило? Вот так вот! А вы думали, что это игрушки? Нее тут все серьезно. Для переменных этого типа доступны следующие операции:

* Операции сравнения: =, >, <, >=, <=
* Операции сложения/вычитания : +, -
* Операции деления/умножения: \*, “/”, div, mod

Операция “/” и div не имеет никакой разницы. Mod – возвращает остаток от целочисленного деления.

Тип real – этот тип представляет собой значения дробных чисел. Операции с дробными числами медленны, так что вы должны понимать, что приложения со сложной математикой, могут выполняться довольно таки долго. MIDlet Pascal предоставляет около 20 функций для работы с реальными числами и предопределенную константу pi (π). Операции с этими переменными схожие с переменными типа integer, за исключением операций mod и div.

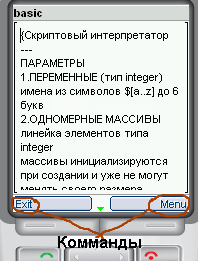
Тип string – этот тип служит для представления текста и содержит последовательности символов. Следующие операции доступны с переменными с этим типом:

* Операции сравнения: >, <, >=, <=, = (сравнение чувствительно к регистру)
* Операции сложения: + (вторым операндом может быть целое число, строка символ или логический тип (true или false))

Чтобы добавить в строку “кавычки” (‘) вводите ее два раза подряд (‘’’’), в этом случае MIDlet Pascal проинтерпретирует его как признак окончания строки.

Тип image – этим типом представляются графические изображения. Для загрузки изображения из JAR файла применяйте функцию *LoadImage*.

Тип command – этот тип служит для отображения команд на дисплее телефона. На рисунке 10 изображен пример команд. Бывает что в программе более двух команд. В этом случае команды группируются в специальное меню, у которых называются они по разному (на телефонах Sony Ericsson в меню “Функции”, у телефонов Siemens в меню “Опции”). Какую отобразить команду а какую поместить в меню функции решает сам аппарат (телефон). Это так же завит от приоритета команды.



Тип recordStore – этот тип предназначен для реализации хранилищ данных. Это например нужно для того чтобы сохранить настройки вашей программы. Хранилище на мобильном телефоне аналогично файлам на компьютерах. Как и файлы хранилища идентифицируются по имени. Данные из Рис.10. Команды.

хранилищ можно как считывать, так и записывать.

Тип http – этот тип служит для открытия соединения с сетью Интернет. У HTTP соединения существует следующий жизненный цикл:

* открытие подключения к удалённому web-серверу
* установка метода запроса http
* добавление полей-заголовков запроса (необязательно)
* добавление данных в тело запроса (только в случае, если запрос имеет тип POST)
* отправка запроса и ожидание ответа от сервера
* получение желаемых полей-заголовков из ответа
* получение данных ответа
* закрытие соединения

Тип Resource – этот тип служит для загрузки из JAR архива текстовые, бинарные и прочие файлы. Для распоряжения ими существуют следующие функции:

* openResource
* resourceAvailable
* closeResource
* readLine
* readByte

Тип array – этот тип служит для создания массивов (переменных одного и того же типа). Про массивы мы поговорим тоже чуть позже.

Комплексный тип record – с помощью этого типа Вы можете создавать свои типы данных. Зачем мне это нужно? Например вы пишите игру, в которой вы управляете героем игры на экране. Для описания позиции героя, уровня жизни - применяют этот тип. Например:

**Type**

HeroType = **record**

PosX, PosY: **integer**;

Health: **integer**;

Вот мы кратко ознакомились с типами переменными. Теперь объясню, как переменной присвоит значение. Для присваивания значения переменной применяют сочетание символов двоеточия и знака равенства (:=). Например:

**Var**

A: **integer**;

**Begin**

A:=5;

**End**.

Теперь попробуйте написать тот же Hello World! но используя при этом переменные. Подскажу лишь то, что тип переменной – string.   
Ответ можете поглядеть [здесь](http://blackwolf.h2m.ru/MP/src/HelloWorld1.zip).

**Глава 5. Условия.**

Итак, теперь настало время переходить к условиям. Надеюсь что вы правильно меня поняли. Язык программирования без условий – это уже язык, за исключением языка BrainFuck. Какое не приличное слово!

Пример условия: «Если будет солнышко, то я пойду на улицу, иначе буду сидеть дома». Вот Вам простое условие. В MIDlet Pascal условия совершенно другие. Для условия применяется такая конструкция:

**If** … **then** … [**else** …]

Синтаксис оператора условия такой:

**If**  условие **then**

**Begin**

Операторы1;

**End**

[

**Else**

**Begin**

Операторы;

**End**;

]

Этот код читается так: «Если условие истинно (правдиво) то выполнить операторы1, иначе (если условие ложно) выполнить операторы2 ». Участок кода внутри квадратных скобок ([]) не обязателен.

Заметьте знакомую конструкцию:

**Begin**

Операторы1;

**End**;

В отличие от предыдущих, здесь после слова **end** стоит символ точка с запятой (;), а не точка (.). Дело в том, что точка ставится в самом конце программы после слова **end**, а если после слова **end** идут дальше операторы, то разделяем их символом точкой с запятой. Думаю что понятно вам.

Давай приведем пример оператора с условием:

**If** x>5 **then** drawText(‘Больше’,5,5);

**Else** drawText(‘меньше’,5,5);

Здесь я не стал ставить операторные скобки begin … end, потому что в условии стоит всего лишь один оператор, если стоят в условии несколько операторов, то в этом случае объедините их операторными скобками.

Для того чтобы посмотреть, как работает условный оператор мы напишем небольшую программку которая будет генерировать число от нуля до десяти и если это число будет больше 5, будет выводиться надпись “Больше 5”, а если меньше, то надпись “меньше 5”. Исходный код [здесь](http://blackwolf.h2m.ru/MP/src/if_else1.zip).

**Program** NewProject;

**Var**

n: **integer**;

**Begin**

n:=random(11);

**if** n>=5 **then** drawText(‘Больше 5’,2,2);

**else** drawText(‘Меньше 5’,2,2);

drawText(‘Число ’+IntegerToString(n),2,40);

repaint;

delay(5000);

**end**.

В этом примере мы встретили две новые функции: IntegerToString и random.

Функция IntegerToString служит для перевода числа в строку, так как невозможно отобразить целочисленную переменную или число процедурой DrawText или какой-либо еще. В параметрах этой функции (в скобках), нужно указывать число или целочисленную переменную, которую нужно перевести строку. Синтаксис функции:

IntegerToString(число\_или\_целочиленная\_перем.) :string;

Функция random служит для генерации случайного числа от 0 до n-1.

В скобках указывается число – это предел, т.е. для генерации числа между 0 и 10, нужно написать так;

N:=Random(11);

А почему мы параметр указали 11, а не 10? Дело в том что функция random генерирует число от 0 до максимального значения при этом от максимального значения вычитается 1, поэтому мы написали 11.

А теперь разберем и сам пример:

Начнем с переменной n: integer;. Эта переменная для функции Random, переменная будет получать случайное число. Далее идет сам условный оператор, который будет сравнивать число в переменной n с числом 5. Для того чтобы отобразилась строка “Больше 5”, число в переменной должно быть больше числа 5 (в этом случае условие будет истинно), а чтобы отобразилась строка “Меньше 5” – число в переменной n должно быть меньше пяти (в этом случае условие будет ложно и выполнятся операторы после слова Else). После условного оператора идет процедура DrawText, которая выводит слово “число ” и с помощью функции IntegerToString прибавляет сгенерированное число (в итоге на экране отображается слово “ число ” и сгенерированное функцией random, число). Далее это все отрисовывается и задерживается на пять секунд. И программа заканчивается. Кстати при следующем запуске мидлета будет сгенерировано совершенно другое число.

Ниже я представлю несколько примеров условных операторов с разными типами данных:

* if a = true then drawText(‘Условие истинно’,5,5);
* if a = false then drawText(‘Условие ложно’,5,5);
* if 5 = 5 then drawText(‘Пять равно пять – это правда!’,5,5);
* if 5>9 then drawText(‘Неправильно! Пять меньше девяти.’,5,5);
* if ‘a’ = ‘A’ then drawText(‘Условие ложно ‘a’ – прописная, ‘A’- заглавная ’,5,5);
* if ‘a’ < ‘b’ then drawText(‘Условие истинно, ‘b’ – стоит дальше ‘a’, а значит ‘b’ больше чем ‘a’’,5,5);

Думаю что этих примеров достаточно.

Кстати для проверки условия, необязательно проверять его таким образом:

if a = true then drawText(‘Привет’,0,0);

Можно проверить и так:

if a then drawText(‘Привет’,0,0);

Здесь все будет точно так же при условии, что A будет истинно.

Бывают случаи, когда нужно проверить сразу несколько условий, например 5 больше 1 и 7 меньше 2. Для этого применяют такие служебные слова:

* AND (Логическое И)
* OR (Логическое ИЛИ)
* XOR (Исключающее логическое ИЛИ)
* NOT (Логическое НЕ)

Например нужно сделать условие чтобы сразу два условия были истинны:

if (x = true) and (y = true) then DrawText(‘Условия верны’, 0,0);

Это можно прочитать так: «Если x = true и y = true – все это верно, то “Условия верны”»

Если например будет y = false, то условие не будет выполнено, так как в данном условии должны быть два операнда равные true ((x = true) and (y = true)).

Если вместо AND поставить OR,то условие выполнится, если хотя бы одно выражение будет истинным:

if (x = true) or (y = false) then DrawText(‘Условия верны’, 0,0);

То есть если x = true будет истинным или y = false будет истинным, то условие выполнится. Условие выполнится, если хотя бы одно выражение будет истинным.

Операция XOR отличается от OR лишь тем, что выражения с XOR возвращают false.

Операция NOT (НЕ) всегда возвращает false. Например: «Не Буратино!». Заметили что идет отрицание? Вот такие вот дела!

С такими условиями мы ознакомимся на практике.

Напишем программу, с использованием операции OR, которая будет выводить текст на экран при нажатии клавиш 1 и 3 на телефоне.

**Program** NewProject;

**begin**

**repeat**

delay(100);

setColor(255,255,255);

fillRect(0,0,getWidth,getHeight);

**if** (getKeyPressed = KE\_KEY1)**or**(getKeyPressed = KE\_KEY3) **then** **begin**

setColor(0,0,0);

drawText('Привет!',5,20);

**end**;

repaint;

**until** getKeyClicked = KE\_KEY0; // Нажать 0 для выхода

**end**.

В этом примере мы встретили еще несколько незнакомых слов: setColor, fillRect, getKeyPressed, KE\_KEY1, KE\_KEY3, getKeyClicked и конструкцию repeat … until.

Начнем по порядку:

* **setColor** – эта процедура устанавливает цвет. Цвет задается в формате RGB. Процедура имеет три параметра, которые указываются целым числом. Например если вам нужно установить ярко-красный цвет – нужно написать так: setColor(255,0,0); чтобы указать белый цвет – пишем: setColor(255,255,255);. Цвета получаются в результате смешивания трех основных цветов: красного, зеленого и синего. В качестве одного из параметров указывается число в пределах 0…255.
* **fillRect** – эта процедура рисует закрашенный прямоугольник. В примере это было нужно для того, чтобы закрасить экран. Например, когда мы нажимали на 1 или на 3 (на телефоне) – появлялось слово “Привет”, при отпускании клавиши, надпись должна пропадать, но для этого нужно закрасить экран прямоугольником белого цвета. Процедура имеет параметры описывающие расположение прямоугольника, т.е. высота, ширина. Синтаксис: **fillRect(x, y, width, height);**

**x** – это начальная координата по горизонтали, **y** – это начальная координата по вертикали, **width** – конечная ширина, **height** – конечная высота. В примере мы использовали в качестве width слово getWidth, а в качестве height – слово getHeight. Функция getWidth – возвращает ширину дисплея телефона, getHeight – возвращает высоту. В качестве параметров процедуры fillRect мы ставили эти функции для того чтобы закрашивать экран полностью на всех моделях телефонов, на которых запущено приложение. Чтобы закрасить прямоугольник каким-либо цветом, используйте процедуру **setColor** перед процедурой **fillRect**

* **getKeyPressed** – эта функция предназначена для проверки нажатия клавиш. Когда мы нажимали на клавиши 1 и 3, проверялось условие

**if** (getKeyPressed = KE\_KEY1)**or**(getKeyPressed = KE\_KEY3) **then** **begin**

setColor(0,0,0);

drawText('Привет!',5,20);

**end**;

getKeyPressed = KE\_KEY1 – это значит что нажата клавиша 1, getKeyPressed = KE\_KEY3 – нажата клавиша 3.

То есть в этом условие было так: если нажать клавишу 1 или 3, то написать слово Привет!.

Для функции **getKeyPressed** существуют следующие константы клавиш телефонов:

* KE\_KEY0 – клавиша 0
* KE\_KEY1 – клавиша 1
* KE\_KEY2 – клавиша 2
* KE\_KEY3 – клавиша 3
* KE\_KEY4 – клавиша 4
* KE\_KEY5 – клавиша 5
* KE\_KEY6 – клавиша 6
* KE\_KEY7 – клавиша 7
* KE\_KEY8 – клавиша 8
* KE\_KEY9 – клавиша 9
* KE\_STAR – клавиша “Звездочка”
* KE\_POUND – клавиша “Решетка”
* **getKeyClicked** – эта функция отличается от функции **getKeyPressed** тем, что **getKeyClicked** для однократного нажатия, а **getKeyPressed** для удерживания нажатой клавиши.
* **Repeat** … **until** – эта конструкция образует цикл. Циклы повторяют код множество раз или определенное число раз. С циклами мы ознакомимся в следующей главе.

**Глава 5. Циклы.**

Итак, изучив условия, перейдем к следующему типу операторов – циклы.

Циклы тоже очень полезные операторы в MIDlet Pascal. Циклы используют для повторения выполнения операторов. В играх без циклов – никуда. Ведь надо каждый раз обновлять экран. С помощью циклов можно реализовать движение объектов, например, когда движется герой игры – все это происходит с помощью циклов. А вы что думали, что в сказку попали? Очнитесь! Вы в реальном мире!

Всего в MIDlet Pascal существует три оператора циклов:

* Цикл for … to … do
* Цикл while … do
* Цикл repeat … until

Начнем с цикла for … to … do. Этот оператор цикла называется счетчиком. Синтаксис цикла for … to …do:

**For** счетчик := начальное\_значение **to** конечное\_значение **do**

**Begin**

Операторы;

**End**;

Для цикла for нужно объявить переменную-счетчик. Приведем пример цикла:

**For** i:=0 **to** 10 **do begin**

x:= x + 2;

**End**;

В этом примере цикл выполнит код находящийся в операторных скобках begin … end десять раз (цикл сложит 10 раз выражение x:=x+2). Если, например x будет равен 0, то при выполнении цикла к переменной x будет прибавляться 2. Когда цикл окончится, x будет равен 20.

Чтобы показать на примере, как будет работать цикл, напишем небольшую программку, которая будет выводить цифры на экран от 0 до 100:

**Program** NewProject;

**Var**

i:**integer**;

**Begin**

**For** i:=0 **to** 100 **do** **begin** // начинаем счетчик

setColor(255,255,255); // белый цвет

fillRect(0,0,getWidth,getHeight);

setColor(0,0,0); // черный цвет

drawText(integerToString(i), 20, 20);

repaint;

delay(200); // задерживаем

**end**;

delay(2000);

**end**.

В этом примере код внутри цикла будет выполняться 100 раз и счетчик i будет увеличиваться на 1. Процедура DrawText будет использовать значение счетчика и выводить это значение на экран. С помощью задержки мы регулируем скорость выполнения цикла: чем меньше задержка, тем быстрее выполняется код внутри цикла. Последняя задержка, идущая после цикла, задерживает программу на две секунды.

Если вам нужно сделать обратный отсчет, то вместо **to** применяйте **downto**. Ниже приведен тот же код с использованием **downto**, который будет реализовывать обратный отсчет:

**Program** NewProject;

**Var**

i:**integer**;

**Begin**

**For** i:= 100 **downto** 0 **do** **begin** // начинаем счетчик

setColor(255,255,255); // белый цвет

fillRect(0,0,getWidth,getHeight);

setColor(0,0,0); // черный цвет

drawText(integerToString(i), 20, 20);

repaint;

delay(200); // задерживаем

**end**;

delay(2000);

**end**.

Объяснять я этот пример не буду, здесь практически так же как и в предыдущем примере.

Теперь ознакомимся с циклом while … do. Этот цикл называется циклом с предусловием. Он выполняется, если проверяемое условие истинно. Сначала проверяется условие, и если оно истинно, то цикл выполняется, затем опять проверяется условие и снова выполняется тело цикла. Графически это показано на рисунке 11. Цикл while … do является бесконечным циклом, т.е. он будет выполняться до тех пор, пока проверяемое условие не станет ложным или пока не прервется оператором break. Синтаксис оператора цикла с предусловием имеет вид:

Рис. 11. Цикл с предусловием while … do

условие

Тело цикла

**while** условие **do** **begin**

операторы;

**end;**

Условие – это проверяемое условие. Если оно истинно, то цикл выполняется. Если внутри цикла больше чем один оператор, то их нужно заключить в операторные скобки **begin … end**.

Напишем для демонстрации небольшую программку, которая будет выводить на экран число, которое будет возрастать, при нажатии клавиши 5, счет будет закончен и программа завершится.

**Program** NewProject;

**var**

a: **integer**;

**Begin**

a:=0;

**while** true **do** **begin**

setColor(0,0,0);

drawText(integerToString(a),5,5);

repaint;

a:=a+1;

setColor(255,255,255);

fillRect(0,0,getWidth,getHeight);

**if** getKeyClicked = KE\_KEY5 **then** **break**; // прерываем цикл

**end**;

**End**.

Начну объяснять пример с цикла. В условии цикла стоит булево значение true, в таком случае условие истинно и из-за этого запускается цикл, который будет выполнять код внутри цикла до бесконечности, пока вы не нажмете кнопку пять на клавиатуре своего телефона. При нажатии на клавишу пять – запускается оператор break и цикл прерывается.

Если цикл не прервать оператором break или не остановить его ложным условием, то произойдет зацикливание и приложение “зависнет”.

Теперь познакомимся с последним оператором цикла – repeat … until. Этот цикл так же называется циклом с постусловием, т.е. в этом цикле сначала выполняется код (тело цикла), а только потом проверяется условие.

Рис.12. Цикл с постусловием.

repeat … until

Тело цикла

условие

Хотел бы заметить, что для выполнения цикла с постусловием, условие должно иметь булево значение false. Если, например условие будет равно true, то тело цикла все равно выполнится, но только один раз. Если условие false, то тело цикла будет выполняться до тех пор, пока условие не станет true или не прервется оператором break. Иначе может так же случится зацикливание. На рисунке 12 графически изображен

цикл с постусловием.

Предыдущий пример можно было бы записать с циклом с постусловием таким образом:

**Program** NewProject;

**var**

a: **integer**;

**Begin**

a:=0;

**repeat**

setColor(0,0,0);

drawText(integerToString(a),5,5);

repaint;

a:=a+1;

setColor(255,255,255);

fillRect(0,0,getWidth,getHeight);

**if** getKeyClicked = KE\_KEY5 **then** **break**; // прерываем цикл

**until** false;

**End**.

Объяснять думаю, не придется.

**Глава 6. Массивы.**

Теперь поговорим о массивах. Массив – это упорядоченный набор переменных (элементов) одного и того же типа. Для чего нужны массивы? Допустим вам надо написать 200 переменных:

**var**

p1, p2 ,p3 , …, pN: **integer**;

Хотелось бы Вам так писать? Думаю, что нет. Гораздо лучше написать это все таким образом:

**var**

p:**array**[0..199] **of** **integer**;

Так гораздо лучше!

Массивы объявляются в разделе объявления переменных. Синтаксис:

*Имя массива*: **array**[*нижний индекс* .. *верхний индекс*] **of** Тип;

Нижний индекс и верхний индекс указывает размерность массива, т.е. количество элементов (переменных). Нижний индекс может начинаться от нуля.

Примеры массивов:

a:**array**[1..5] **of integer**;

Создает массив, состоящий из 5 целочисленных элементов

a, b:**array**[1..15] **of real**;

Создает 2 массива с именем a и b состоящих из 15 вещественных (дробных) элементов

a:**array**[0..19] **of** **string**;

Создает массив, состоящий из 20 элементов (нумерация с нуля).

Для обращения к элементу, как к обычной переменной осуществляется через индекс элемента массива. Например, есть массив, состоящий из пяти элементов

**var**

a: **array**[0..4] **of** **integer**;

второму элементу нужно присвоить значение числа два:

a[1]:=2;

пятому присвоить значение, состоящее из числа 9:

a[4]:=9;

То, что находится внутри квадратных скобок – это и есть индекс массива.

Давайте напишем пример заполнения массива случайными числами в диапазоне от 0 до 20:

**Program** Array\_example;

**Var**

A: **array**[1..10] **of** **integer**;

I, y: **integer**;

**Begin**

y:=0;

**For** I:=1 **to** 10 **do** **begin**

A[i]:=random(21);

drawText(IntegerToString(A[i]),5,y);

y:=y+13;

repaint;

**End**;

Delay(10000);

**End**.

Если запустить программу, то она выведет на экран 10 чисел. При каждом новом запуске программа выведет совсем другие числа. Получить случайное число помогает функция random (описывали ее выше).

Выражение **y:=y+13** служит для того, чтобы каждая выводимая строка числа была ниже на 13 пикселей. Как вы заметили, что в качестве индекса мы используем переменную цикла I, т.к. мы заполняем наш массив по порядку, а переменная I как раз изменяет свое значение в большую сторону на 1. Мы бы могли заполнить массив и таким образом:

A[1]:=random(21);

A[2]:=random(21);

A[3]:=random(21);

A[4]:=random(21);

A[5]:=random(21);

A[6]:=random(21);

A[7]:=random(21);

A[8]:=random(21);

A[9]:=random(21);

A[10]:=random(21);

Но вряд ли бы кто хотел бы писать, таким образом, думаю, что не к чему лишний код.

**Глава 7. Процедуры и Функции (Подпрограммы).**

Итак, теперь перейдем к частям языка таким как процедуры и функции. Что это такое? Это подпрограммы. Функция отличается от процедуры лишь тем, что она возвращает какое либо значение, процедура ничего не возвращает.

Если вы решили написать какую-нибудь игрушку, например какую-нибудь экономическую стратегию, то без процедур и функций вам не обойтись. Пускай одна функция будет отвечать за количество собранного урожая, другая будет отвечать за количество работников. Но до игр нам пока далековато.

Начнем изучать процедуры. Простая процедура имеет такой синтаксис:

**Procedure** *имя процедуры* [ (параметры) ];

**Var** *переменные процедуры*;

**Begin**

*Операторы;*

**End**;

То, что находится внутри квадратных скобок – это необязательные параметры. Переменные объявленные в процедуре или функции называются локальными переменными. Такие переменные доступны только процедуре или функции в которой эта переменная объявлена. Переменные, которые были объявлены в разделе описаний переменных самой программы называются глобальными переменными и доступны полностью всей программе. Процедура или функция вызывается через их заданные имена. Например, процедура:

**Procedure** Example;

**Begin**

Операторы;

**End**;

Через ее имя, процедуру можно вызывать главной программой или другой программой. Например:

**Program** Ex\_pr;

**Procedure** Example;

**Begin**

Операторы;

**End**;

**Begin**

Example;

**End**.

Процедура или функция всегда должна стоять выше главной программы. Если одна процедура или функция вызывает вторую процедуру или функцию, то вторая процедура или функция должна стоять выше первой. Например:

**Procedure** b;

**Begin**

Операторы;

**End**;

**Procedure** a;

**Begin**

Операторы;

**End**;

**Begin**

a;

**End**.

Если например процедура a вызывает процедуру b, а та в свою очередь вызывает процедуру a, то необходимо создать опережающую процедуру. Т.е поверх всех этих процедур и функций вставить строчку **procedure** a; **forward**;

Например:

**Program** ex\_pr;

**Procedure** a; **forward**;

**Procedure** b;

**Begin**

Операторы;

**End**;

**Procedure** a;

**Begin**

Операторы;

**End**;

**Begin**

a;

**End**.

Таким образом, получается, что процедура стоит поверх всех остальных. Теперь перейдем к параметрам процедур.

Бывают случаи, когда нужно выполнить процедуру или функцию получив от главной программы или другой подпрограммы какие-нибудь данные. Рассмотрим процедуру, выполняющую функцию калькулятора:

**Procedure** Calculator(num1, num2: **integer**; op: **string**);

**Begin**

**if** op = ‘+’ **then** rez:=num1+num2; // сложить

**if** op = ‘-’ **then** rez:=num1-num2; // вычесть

**if** op = ‘/’ **then** rez:=num1/num2; // делить

**if** op = ‘\*’ **then** rez:=num1\*num2; // умножить

**End;**

Здесь процедура принимает три параметра: num1 (первое число), num2 (второе число), op (операция с числами). Если представить, что у нас будет три поля ввода, то первые два нам будут нужны для чисел, третье для ввода операции. Параметр op будет получать операцию вводимую пользователем в третье поле, и если строка, записанная в переменную совпадет с одной из строки (+, -, / или \*), то выполнится соответствующая операция над числами записанные в переменные num1 и num2. Переменная rez должна быть объявлена в самом начале программы. Эта переменная получает результат вычисления.

Такую процедуру можно записать и с помощью функции. Давайте немножко ознакомимся с функциями. Функция отличается от процедуры лишь тем, что она может возвращать какое-нибудь значение. Синтаксис функции следующий:

**Function** имя\_функции [необязательные\_параметры]:**возвращаемый\_тип**;

**Begin**

Тело функции (код);

**End;**

Тело функции – это код, который находится внутри функции. Возвращаемый тип – это тип данных которая возвращает функция (текст, число и пр.).

Для понимания запишем процедуру *Calculator* в виде функции:

**function** Calculator(num1, num1: **integer**; op: **string**): **integer**;

**begin**

**if** op = ‘+’ **then** Calculator:=num1+num2; // сложить

**if** op = ‘-’ **then** Calculator:=num1-num2; // вычесть

**if** op = ‘/’ **then** Calculator:=num1/num2; // делить

**if** op = ‘\*’ **then** Calculator:=num1\*num2; // умножить

**end;**

Заметьте, такая строка Calculator:=num1+num2 и идущие за ней дальше, в них слово Calculator – это уже не переменная, а та же самая функция. Получается что функция вызывает саму себя (с процедурами это тоже допустимо). Такие случаи называются *рекурсией*. В данном случае так функция получает значение при операциях над теми же переменными num1, num2. Возвращаемый тип должен строго соответствовать типу данных которая получает функция иначе вы получите ошибку о том что операнды должны иметь одинаковые типы данных. Операнды это переменные или числа над которыми введется какая либо математическая или логическая операция.

Получив знания о процедурах и функциях, мы теперь разработаем проект «калькулятор. Это будет наш первый полезный проект ☺.

Листинг программы «калькулятор»:

**Program** Calc;

**var**

n1,n2:**integer**; // переменные чисел

rez:**string**; // переменная для операции над операндами

**function** calculator(num1, num2:**integer**; op: **string**): **integer**;

**begin**

**if** op = '+' **then** calculator := num1 + num2; // сложить

**if** op = '-' **then** calculator := num1 - num2; // вычесть

**if** op = '/' **then** calculator := num1 / num2; // поделить

**if** op = '\*' **then** calculator := num1 \* num2; // умножить

**end**;

**procedure** start; // интерфейс программы

**var** num1f, num2f, opf : **integer**; //переменные для полей ввода чисел и операций

ok: **command**; // переменная для команды «решить»

**begin**

ShowForm; // показать форму на экран

ok:=createCommand('Решить',CM\_OK,1); // создать команду

num1f:=formAddTextField('Введите первое число','',20,TF\_NUMERIC); // текстовое поле для ввода чисел

num2f:=formAddTextField('Введите второе число','',20,TF\_NUMERIC);

opf:=formAddTextField('Операция','',1,TF\_ANY); // текстовое поле для ввода операций

addCommand(ok); // помещаем команду «решить» на экран

delay(500); // чтобы не зависало

**repeat** **until** getClickedCommand = ok; // ждем нажатия команды «решить»

n1:=StringToInteger(formGetText(num1f)); // извлекаем текст из текстовых полей и переводим в число

n2:=StringToInteger(formGetText(num2f));

rez:=formGetText(opf); // извлекаем операцию из поля ввода текста для операций

**if** rez = '#' **then** halt; // если операция будет соответствовать символу «#» то закрываем программу

ShowAlert('Решение','Ответ '+calculator(n1,n2,rez),LoadImage('/icon.png'),ALERT\_INFO); // показываем сообщение с результатом решения.

playAlertSound; // проигрываем звук

delay(3000); // показываем сообщение 3 секунды

clearForm; // и очищаем форму

start; // вызываем процедуру start заново, для дальнейших действий с программой

**end**;

**begin**

start; //вызываем интерфейс программы (процедура start)

**end**.

Итак, вот наш проект *Calculator*. На рис. 13 показан его вид на эмуляторе MidpX.

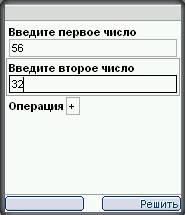


Рис. 13. Проект *«калькулятор»*

Наша программка имеет интерфейс, состоящий из трех текстовых полей и одной команды. Все это помещается на форму. Форма это контейнер, в который может помещаться такие элементы, как текстовые поля, шкалы, картинки и пр. подробнее об этом в следующей главе.

Теперь попробуем разобрать такую строку:

n1:=StringToInteger(formGetText(num1f));

Это строка состоит из двух функций: StringToInteger и formGetText.

Первая конвертирует строку в целое число, вторая функция извлекает строку из текстового поля с заданным идентификатором текстового поля. Таким образом, переменная получает число из текстового поля.

Строка

ShowAlert('Решение','Ответ '+calculator(n1,n2,rez),LoadImage('/icon.png'),ALERT\_INFO);

Выводит ответ на экран в виде сообщения. Эта строка так же обращается к нашей функции *calculator* с параметрами извлеченных из текстовых полей чисел и строки операции и получает результат вычисления в виде числа. Но чтобы ее отобразить мы поставили слово «Ответ» и прибавили функцию calculator. Таким образом, число перекодировалось в строку (вспомните операции со строками, что если к строке прибавить число, то это число так же станет строкой).

**Глава 8. Формы. Пользовательский Интерфейс**

**Высокого Уровня.**

Итак, настало время перейти к изучению высоко-уровнего интерфейса. В MIDlet Pascal существуют 2 вида интерфейса: высокоуровневый и низкоуровневый. К высокоуровневым относятся такие объекты:

* Form (форма) – это контейнер, который может содержать в себе текстовые поля, строки, изображения, поля выбора, шкалы и прочие элементы. Форма вызывается с помощью процедуры **ShowForm**.
* Alert (сообщение) – это область которая занимает весь экран и отображает некоторое сообщение. Например когда в вашем приложении пользователь выберет пункт о программе, то можно всю информацию о приложении можно отобразить с помощью alert.
* TextBox (поле ввода текста) – это поле которое занимает весь экран и предназначено для ввода текста.
* List (меню) – это меню выбора. Очень может пригодится при создании меню к игре или приложению (если вы конечно не предпочтете создать меню с помощью низкоуровнего интерфейса).

На рисунке 14 показаны все эти четыре объекта.

Рис.14. Элементы высоко-уровнего интерфейса

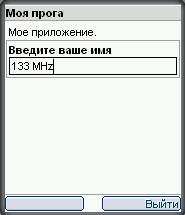
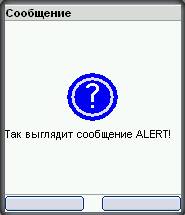
TextBox

(Поле ввода текста)

Alert (сообщение)

List (меню)

Form (форма)



Начнем с изучения **Form** (формы). Как говорилось выше, форма представляет собой контейнер в который помещаются ее элементы (текстовые области, поля выбора, строки и прочее).

Чтобы вызвать форму и показать ее на экран используют процедуру ShowForm.

Синтаксис: ShowForm;

Эта процедура не запрашивает ни каких входных параметров.

Пример вызова формы:

**program** myForm;

**begin**

ShowForm;

Delay(5000);

**end**.

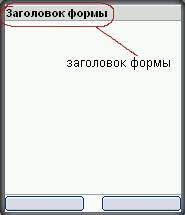
Этот маленький примерчик покажет пустую форму на экране. Для задания заголовка формы используйте процедуру:

* SetFormTitle(title: **string**)

title: string – это строка которая будет помещена в заголовок формы. На рисунке 15 показано как выглядит форма с заголовком на эмуляторе MidpX. Чтобы задать заголовок нужно вставить после строки ShowForm процедуру SetFormTitle(‘ваш текст’);

Рис.15. Пустая форма с

заголовком



Чтобы получить строку с название формы используйте функцию:

* GetFormTitle;

Эта функция вернет строку с заголовком текущей формы.

Для удаления заголовка формы используйте процедуру:

* RemoveFormTitle;

На форму также можно поместить бегущую строку, для этого применяется процедура:

* SetTicker(s: string);

C помощью этой процедуры в самом верху формы появится бегущая строка. Направление ее движения (влево или в право) зависит не от программы, а от самого телефона. На разных моделях телефонов элементы формы показываются по-разному.

**Функции добавления элементов на форму.**

Большинство элементов формы в MIDlet Pascal задаются с помощью функций. Например для того чтобы поместить строку на форму, нужно объявить переменную типа integer.

Пример:

**program** myForm;

**var** str: **integer**;

**begin**

ShowForm;

str:=formAddString(‘Строка на форме’);

Delay(5000);

**end.**

Как вы видите что переменная str используется для создания и добавления элемента «строка» на форму. Теперь ознакомимся с функциями и процедурами для создания элементов и добавления их в форму и функции с процедурами для работы с ними:

* formAddString(s: string) – эта функция добавляет на форму обычную строку.
* formAddTextField(title: string, value: string, maxSize: integer, const: integer) – функция добавляет область для ввода текста.
  + - title – строка заголовка элемента.
    - value – строка которая задает начальный текст в области для ввода текста. Если не надо то поставьте пустые апострофы (‘’).
    - maxSize – ограничение на количество вводимых символов (задается числом).
    - const – константа, задающая тип вводимой информации. Вот они все:
      * TF\_ANY – для ввода всех символов.
      * TF\_NUMERIC – для ввода только целых чисел
      * TF\_URL – для ввода адреса в интернете
      * TF\_EMAIL – для ввода e-mail адреса
      * TF\_NUMBERPHONE – для ввода телефонного номера
      * TF\_PASSWORD – для ввода пароля

Пример использования:

Id\_TF:=formAddTextField(‘Введите имя’, ‘’, 20, TF\_ANY);

Для работы с областью ввода текста (TextField) существуют одна процедура и одна функция:

* formSetText(ID, s) – процедура изменяет значение TextFieds.
  + - ID – идентификатор текстового поля (например Id\_TF).
    - s – строка которой заменяется значение TextField.

Пример использования:

Id\_TF:=formAddTextField(‘Введите имя’, ‘’, 20, TF\_ANY);

formSetText(Id\_TF, ‘133 MHz’);

* formGetText(ID) – функция возвращает текст из TextField с идентификатором ID текстовой области.
* formAddSpace – функция вставляет небольшой пропуск в форму, это например пригодится когда отделяешь элементы друг от друга.
* formAddImage(image) – функция помещает изображение на форму.
  + - Image – это загружаемая картинка, загруженная например с помощью функции LoadImage.

Пример использования:

ID\_img:=formAddImage(LoadImage(‘/img.png’));

* formAddGauge(title, isInteractiv, maxValue, initialValue) – функция помещает шкалу на форму.
  + - title – заголовок
    - isInteractiv – булево значение (true или false) устанавливает, можно ли изменять значение шкалы или нет. True – можно, false – нельзя.
    - maxValue – максимальное значение в единицах.
    - initialValue – начальное значение в единицах

Пример использования:

ID\_G:=formAddGauge(‘Шкала’, true, 100, 50);

Шкала содержит следующие процедуры и функции:

* formSetValue(ID, value) – процедура изменяет значение шкалы с идентификатором ID на число value.
* formGetValue(ID) – функция возвращает значение шкалы с идентификатором ID.
* formAddChoice(title, type) – функция помещает поле выбора на форму.
  + - title – заголовок.
    - type – тип выбора:
      * CF\_EXCLUSIVE – может быть выбран только один элемент (имеет вид переключателя)
      * CF\_MULTIPLE – может быть выбрано несколько элементов (имеет вид галочки).

Функции и процедуры для работы с полем выбора:

* choiceAppendString(ID, ItemString) – добавляет элемент (пункт) в поле выбора и возвращает внутри - групповой индекс элемента. Индексация начинается с нуля. ID – идентификатор, ItemString – название элемента (название пункта).
* choiceAppendStringImage(ID, ItemString, Image) – добавляет элемент (пункт) с изображением в поле выбора и возвращает внутри - групповой индекс элемента. Индексация начинается с нуля. ID – идентификатор, ItemString – название элемента (название пункта). Image – картинка.
* choiceIsSelected(ID, ItemIndex) – возвращает true если выбран элемент с индексом itemIndex. ID – идентификатор.
* choiceGetSelectedIndex() – возвращает индекс выбранного элемента или -1 если не выбран не один элемент. Индексация начинается с нуля