ForexClub 10лет

Язык программирования **Rulang**

Яценко Ю.И.
Кияница А.С., Кобзарь Н.П.

СОДЕРЖАНИЕ

Язык программирования РУЛАНГ

Руководство пользователя для начинающих

Урок 1
Программировать просто!

Рано или поздно у любого трейдера после того, как он освоил многие разные торговые методики и приемы, частенько возникает желание сделать или придумать что-нибудь свое. Программа РУМУС2 – это очень хорошая программа технического анализа, и она позволяет осуществить многое из тех идей, которые приходят в голову трейдера. Например, у нее достаточно много самых разных индикаторов на любой вкус. Но, в один прекрасный момент, трейдер узнает, что есть где-то супер-чудо индикатор, а в РУМУС2 его нет... Что тогда делать?

Очень просто. Оказывается, РУМУС2 позволяет пополнять набор своих индикаторов. Для этого нужно только дать ему очень четкое описание того или иного индикатора в виде текстового файла. ,Не важно где Вы взяли этот текст: нашли в книге или на форуме, взяли у знакомых, или же, в конце-концов, придумали сами. Важно только одно: это описание должно быть очень четким и понятным компьютеру – ведь это именно он будет строить на экране Ваш индикатор.

Стоп! Что же это получается? Оказывается компьютер умеет читать какие-то тексты, да еще и прочтя их, понимает что там написано, и, мало того, после этого рисует индикатор на экране? Да, это именно так! Только с одной маленькой оговоркой. Компьютер действительно понимает текст индикатора, но только при одном очень важном условии – он должен быть написан на специальном языке, который понимает компьютер. Этот язык называется РУЛАНГ (или RuLang – Rumus Language). Говоря другими словами, Вам нужно запрограммировать свой индикатор, написать его текст на языке РУЛАНГ, и тогда Вы увидите его на экране.

Вот в этом месте многие трейдеры начинают пугаться. Ведь это же очень сложно: нужно уметь программировать, нужно быть программистом, нужно знать язык программирования. Наверное, это очень сложный язык (тут вспоминаются ужасы изучения какого-нибудь иностранного языка в школе). На самом деле все гораздо проще. Во-первых, в любом языке программирования очень мало слов. В частности, в РУЛАНГе их что-то чуть более двадцати. Неужели Вам трудно будет запомнить два десятка слов и их смысл (значение)? Во-вторых, эти слова очень просто связываются в определенные фразы, которые при некотором навыке Вы будете понимать, что называется, «с первого взгляда».

Чтобы не быть голословным, давайте рассмотрим простейший индикатор.

Допустим, что нам нужно узнать текущую волатильность рынка ЕВРО. Что такое волатильность? Это подвижность, «размашистость» движения цен. Измерять размер волатильности можно по-разному. Как вариант можно, например, брать за основу размер всей свечи, т.е. разницу ее «High» и «Low». Ну что ж, давайте для начала создадим индикатор, который будет вычислять эту разность, и показывать ее на экране.

Для этого запустим РУМУС2, и построим на нем дневной график ЕВРО (см. рис. 1.1). В верхней строке меню нажмите кнопку «Руланг». Затем в выпадающем списке выберите опцию «Список индикаторов» (она в этом списке единственная).

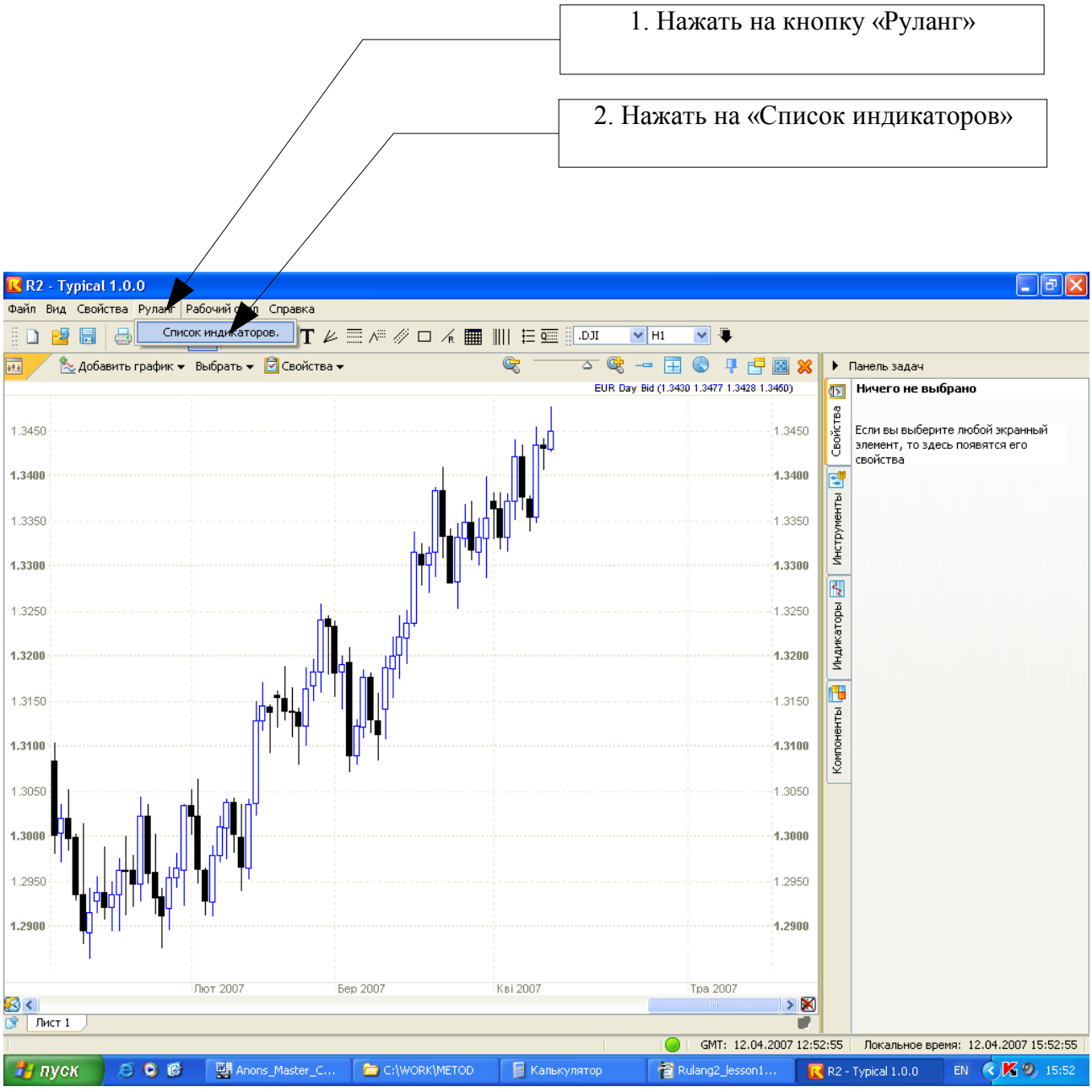

Рис. 1.1

В результате Вы увидите диалоговое окно со списком индикаторов, которые имеются в РУМУС2 на данный момент. См. рис. 1.2.

В этом окне чуть выше самого списка индикаторов находится поле «Фильтр». Оно нужно для того, чтобы быстрее найти необходимый нам индикатор. Например, если мы введем в этом поле букву «А», то в списке индикаторов останутся только те индикаторы, которые имеют в своем имени букву «А». А если, например, введем слог «AD», то тогда, соответственно в этом списке останутся только те индикаторы, которые в своем имени

имеют этот слог. Остальные индикаторы при этом никуда не денутся, просто мы видим этот список как бы «через фильтр». Этот фильтр позволяет резко сократить список индикаторов, среди которых найти нужный нам индикатор будет гораздо легче.

Чуть правее этого поля находится кнопка «Закрыть». При нажатии на нее произойдет то же самое, как если нажать на кнопку с крестиком в правом вернем углу этого окна. Т.е. окно со списком индикаторов будет закрыто.

Ниже этой кнопки находятся еще четыре, назначение которых вполне очевидно.

Кнопка «Новый» используется для того, чтобы создать новый индикатор и ввести его текст.

Кнопка «Правка» используется для того, чтобы исправить (отредактировать) текст уже существующего индикатора. Для этого сначала следует мышью выбрать нужный индикатор из списка, а затем нажать на эту кнопку.

Если Вы хотите навсегда избавиться от какого-нибудь индикатора, то вам следует воспользоваться кнопкой «Удалить». Сначала мышью выбираете нужный (вернее – НЕ нужный Вам) индикатор, а затем нажимаете на эту кнопку.

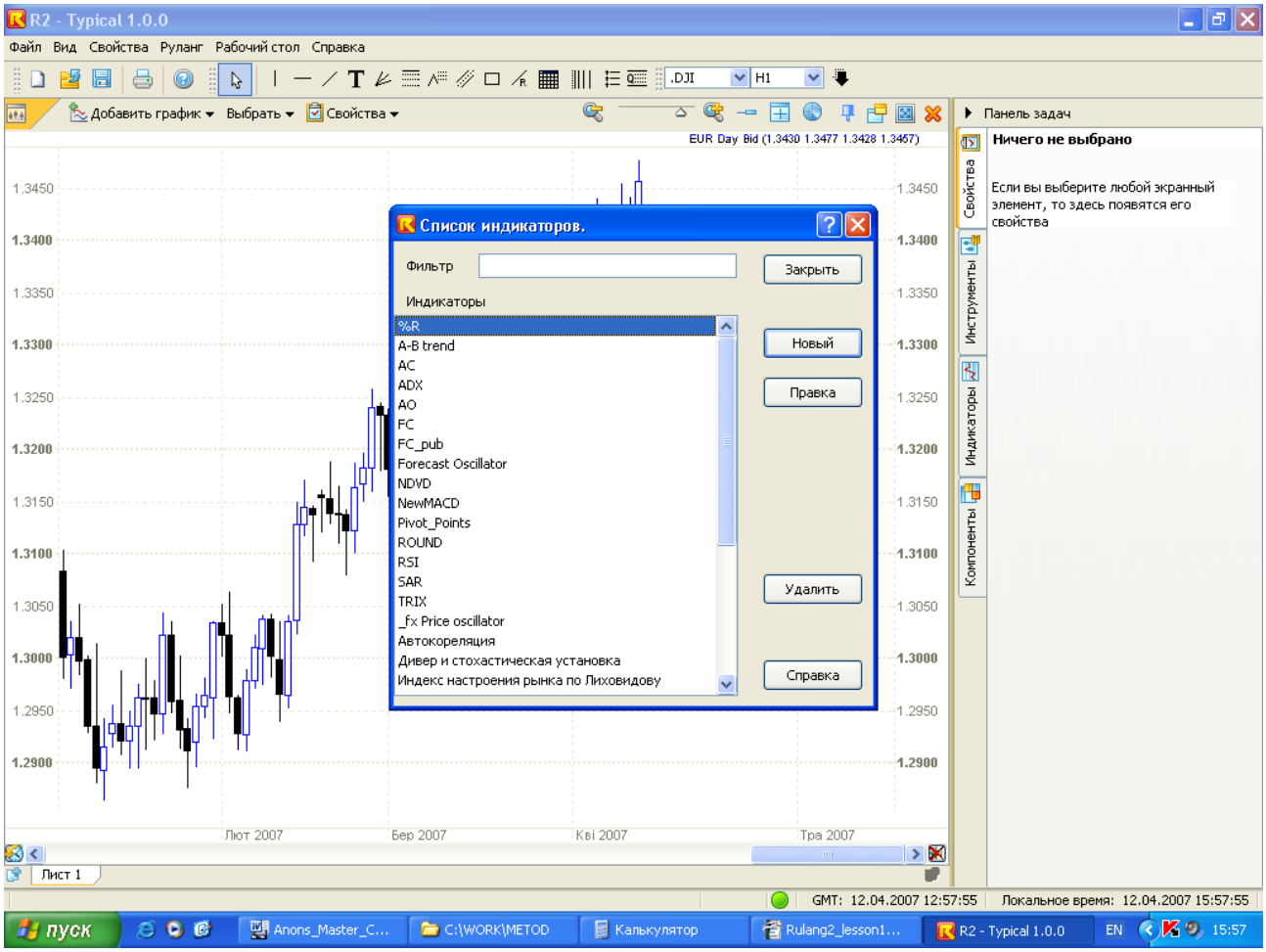

Рис. 1.2

Кнопка «Справка» позволит Вам вызвать справочную информацию по языку РУЛАНГ.

Поскольку нам нужно создать новый индикатор, то мы нажмем кнопку «Новый». В результате появится пустое (без текста индикатора) окно «Правка». См. рис. 1.3.

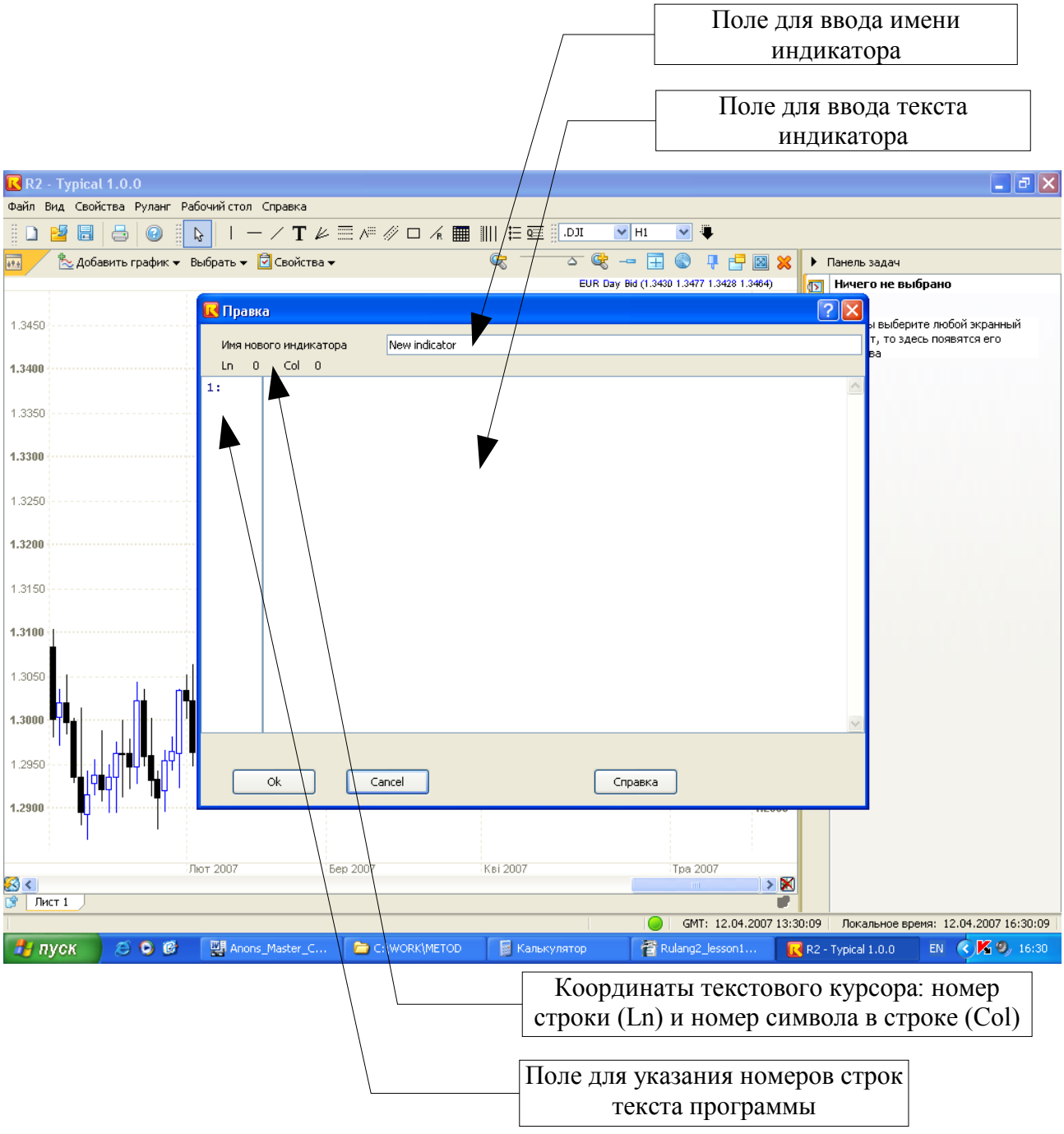

Рис. 1.3

В самом верху окна имеется поле для ввода имени нового индикатора. При создании нового индикатора в этом поле всегда по умолчанию указывается «New indicator». Ниже этого поля находится большое (почти на все окно) окно для ввода текста самого индикатора. Причем заметьте, строки текста автоматически нумеруются. Номера строк указываются слева от текста программы. Кроме того, при перемещении текстового курсора по тексту программы его координаты отображаются чуть выше поля текста программы. Поскольку сейчас текст программы отсутствует, то эти координаты равны [Ln 0; Col 0], т.е текстовый курсор установлен на нулевом символе (колонке) нулевой строки. А зачем нам нужны эти координаты? Дело в том, что зачастую (особенно при поиске ошибок в тексте) нам нужно будет находить определенное место в программе, о котором нам будет известно только, например, номер строки. Хорошо, если текст программы маленький, тогда это место можно будет быстро найти и «на глаз». А если у нас более ста строк текста? Тогда быстрее всего найти нужное место по известному номеру строки.

Итак, сейчас нам нужно сделать две вещи. Во-первых, придумать название индикатора, и, во-вторых, написать текст программы. Первую вещь сделать очень просто. Поскольку индикатор должен показывать волатильность, то мы его так и назовем «Volatil», и впишем это название (без кавычек) в нужное поле.

А вот какой текст нам надо ввести в программу, чтобы компьютер вычислял волатильность? Здесь, как это ни странно, тоже все очень просто. Поскольку волатильность – это разница High и Low, то мы так и запишем эту формулу в первую строку нашей программы:

```
H – L
```

Вот, собственно, говоря и все! Текст программы уже готов (см. рис. 1.4). Осталось только нажать на кнопку «Ok». Неужели это все? Да, это так! Смело нажимаем кнопку «Ok», и ... Что мы видим? (См. рис. 1.5).

Вот здесь трейдеру, который только начинает изучать программирование, становится очень плохо. Ну как же! Ведь нам поступило сообщение об ошибке! И что нам, бедным, делать? А Вы знаете, собственно говоря, ничего страшного не произошло. Самое главное – это не паниковать, а прочитать это сообщение об ошибке. В нашем конкретном случае ошибка просто элементарна: мы не поставили в конце первой (и единственной) строки точку с запятой. Ошибка эта исправляется ровно за три секунды. Кстати, пару слов об ошибках, которые выдает компьютер, когда он пытается прочесть текст программы. В подавляющем большинстве случаев они действительно элементарны, и бояться их не надо. Более того, даже очень опытные программисты при вводе текста своих программ делают ошибки, в том числе и эту. Да-да, именно так, иногда забывают поставить точку с запятой в конце строки (зачем она там нужна – мы разберем чуть позже). Хотя, если говорить по большому счету, то это даже не ошибка, а скорее описка. Так вот, большая часть таких ошибок, по сути, представляют из себя описки, которые очень быстро отыскиваются в тексте программы и также быстро исправляются. Так что паниковать и бояться просто не имеет никакого смысла.

Итак, давайте исправим нашу ошибку. Сначала надо убрать сообщение об ошибке, нажав на кнопку «Ok». Затем, собственно, ищем (а что ее искать – она же в первой строке) эту ошибку, и исправляем ее. Теперь первая строка должна выглядеть так:

```
H – L;
```

После этого снова нажимаем кнопку «Ok». Окно «Правка» исчезает. На экране остается окно «Список индикаторов», в котором уже должен появиться наш индикатор Volatil. Закроем это окно.

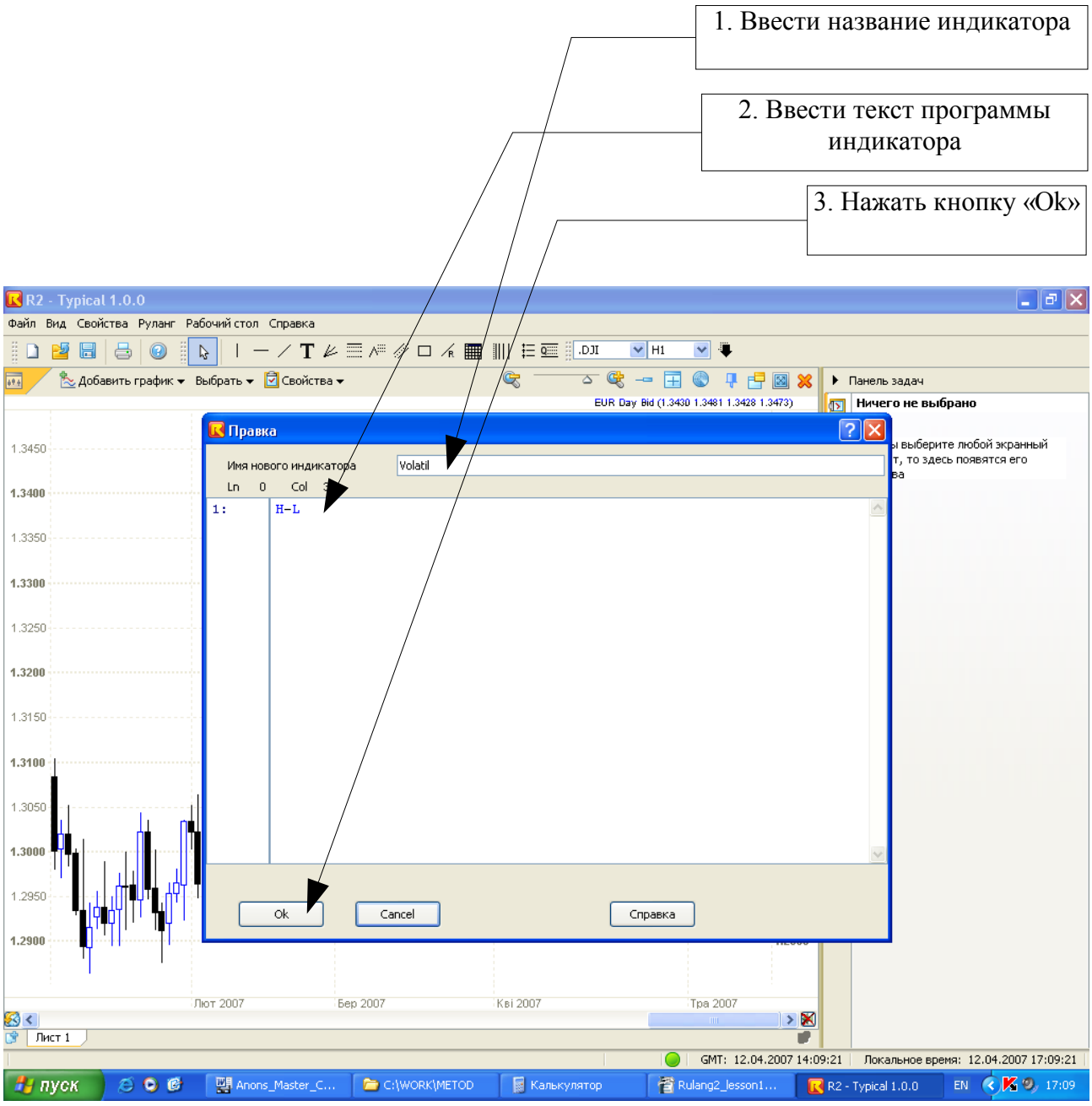

Рис. 1.4

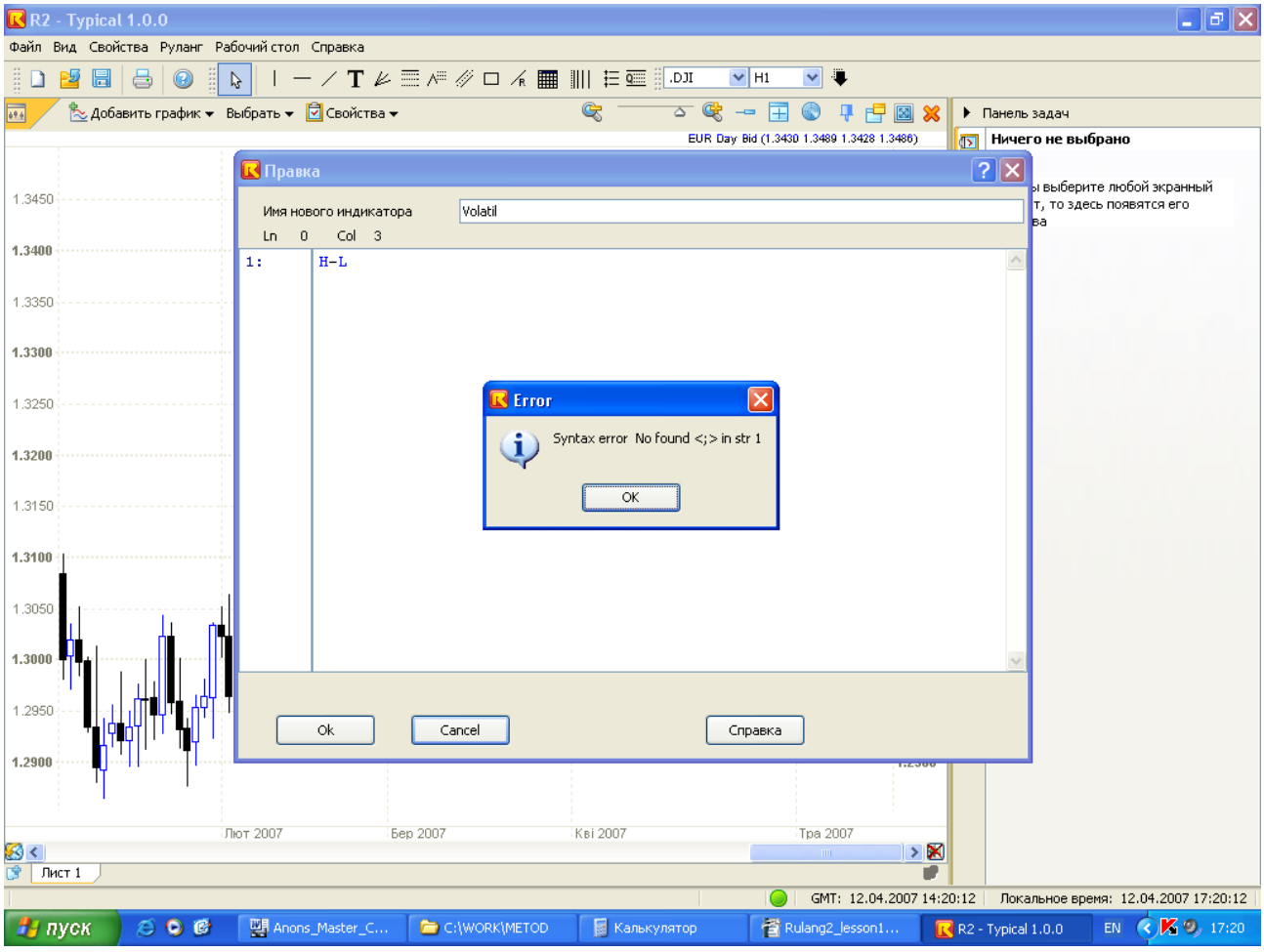

Рис. 1.5

Ну что ж, индикатор у нас уже есть, так что теперь его можно построить на графике. Строить его лучше всего в отдельной графической области, поэтому мы сначала ее создадим, а затем построим там наш индикатор. Должно получиться нечто, похожее на то, что изображено на рис. 1.6. Здесь видно, что этот индикатор очень сильно «дергается», но в то же время явно чувствуется, что у него должна быть некоторая средняя линия, возле которой он постоянно «ходит». Чтобы увидеть эту линию в явном виде, давайте построим по этому индикатору простую скользящую среднюю. Кстати, РУМУС2 может строить индикатор по индикатору, в том числе и по тому индикатору, который был создан пользователем. Давайте период усреднения этой скользящей средней сделаем побольше – где-то 260 (один год на дневном графике). В результате должно получиться примерно то, что изображено на рис. 1.7.

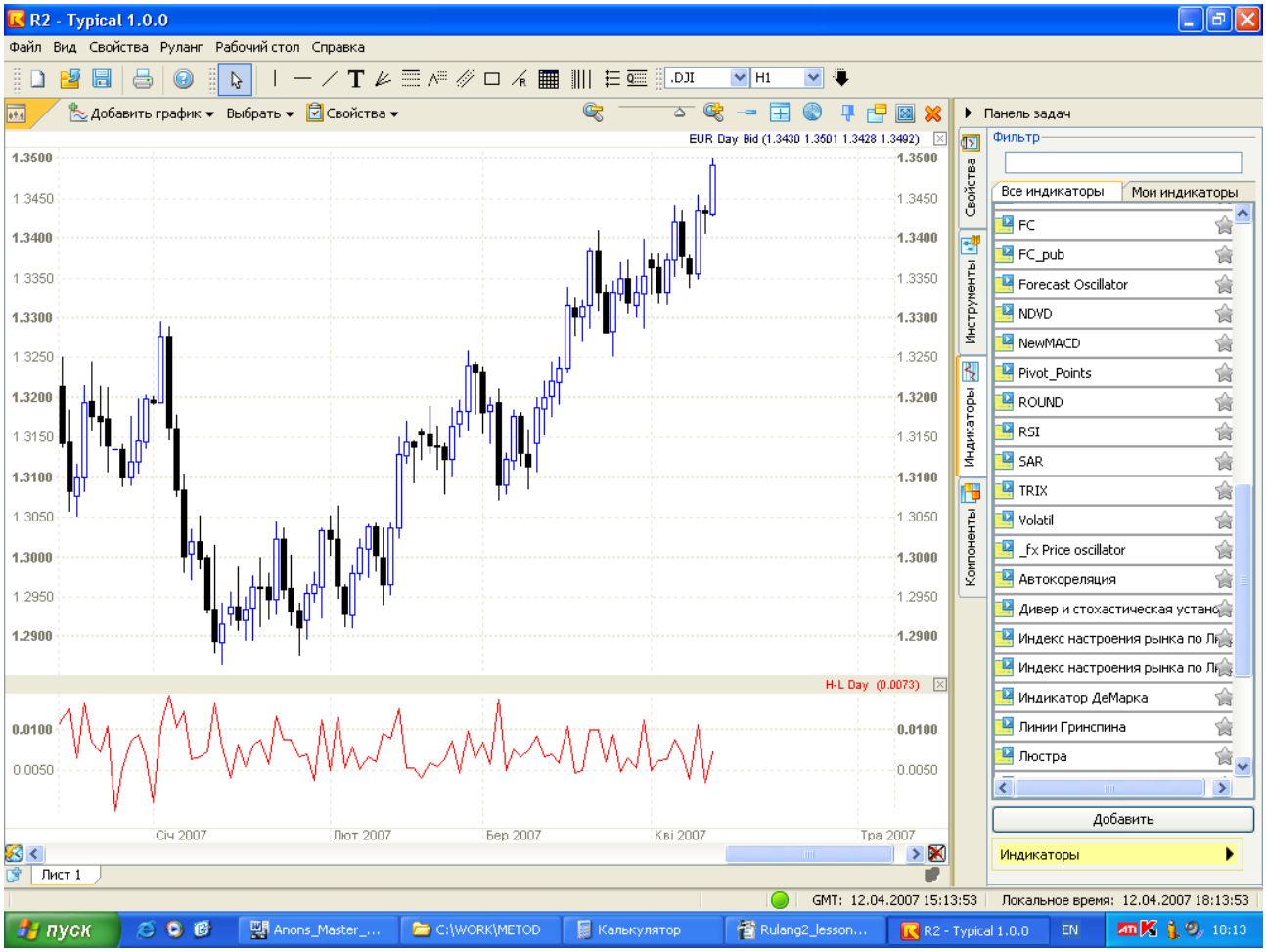

Рис. 1.6

Вот теперь из этого графика видно, что средняя волатильность за последний год по дневному графику ЕВРО составила примерно 90 пунктов. А если «прокрутить» график на год назад, то видно, что тогда волатильность составляла примерно 100 пунктов.

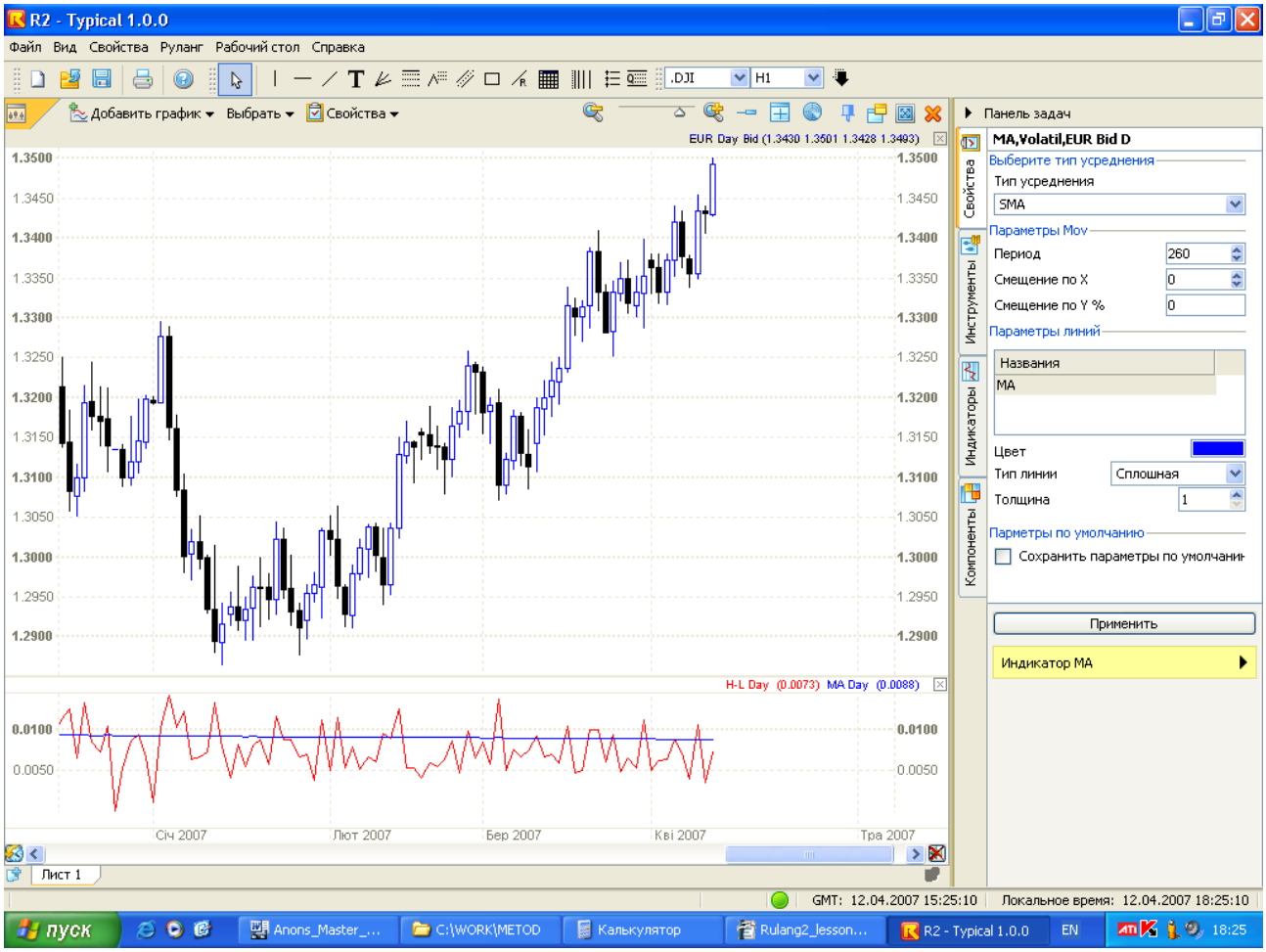

Рис. 1.7

Итак, как мы видим, программировать на самом деле не так уж и сложно. Иногда бывает достаточно обойтись буквально одной строкой текста, состоящей из нескольких символов. Все зависит от сложности того индикатора, который Вам нужен. В нашем случае, например, оказалось достаточно одной фразы:

H – L;

Урок 2

Язык программирования РУЛАНГ и его главные элементы: данные (переменные) и операторы

Теперь давайте немного подробнее поговорим о языке РУЛАНГ. Да, это действительно язык. Он нужен для того, чтобы сказать компьютеру о том, что нам нужно сделать. Как в любом языке, у него существуют определенные конструкции: слова, предложения, фразы, абзацы. Причем, как Вы знаете, эти конструкции языка находятся в иерархической взаимосвязи, т.е. абзацы строятся из предложений, предложения составляются из слов, слова состоят из символов и т.д. В любом языке программирования есть аналогичные иерархические конструкции, только называются они несколько иначе. Например, предложения – это операторы. Несколько операторов можно собрать вместе в «абзац». Такой «абзац» называется составным оператором.

Вы конечно же знаете, что в обычном языке в конце каждого предложения ставится некий знак препинания, а именно – точка. Так вот, в языке РУЛАНГ в конце каждого оператора также ставится определенный знак. Только это не точка, а точка с запятой (;). Можно сказать и так: точка с запятой – это один из знаков препинания, и он ставится в конце оператора. Если говорить точнее, то в языках программирования знаки препинания называются немного иначе – разделители.

Интересно, а почему в конце оператора («предложения») ставится точка с запятой, а не точка, как в обычных языках? Дело в том, что разделитель точка (.) также имеется в языках программирования, только он играет другую роль. Он используется для разделения друг от друга целой и дробной части числа. Ведь у нас в тексте программы могут оказаться и числа, в том числе и дробные. Так вот, чтобы эти два разделителя (разделитель целой и дробной части числа и разделитель операторов) не путались друг с другом, пришлось обозначать их разными символами. Символ точки (.) уже давно (еще до возникновения языков программирования) использовался для разделения целой и дробной частей числа друг от друга. Поэтому, чтобы не нарушать эту традицию, для разделителя операторов был выбран другой, очень похожий по смыслу знак – точка с запятой (;).

Как мы уже упоминали выше, существуют и более сложные конструкции: например, составные операторы (аналог «абзаца» в обычном языке). Интересно, а как выглядит составной оператор в тексте программы. Очень просто, вот так:

begin <оператор1>; <оператор2>; ...; <операторN> **end**;

Кстати, приведенная выше строка является обычной для любого учебника по программированию. На первый взгляд с непривычки может показаться, что тут написано что-то совсем непонятное. Но на самом деле здесь все просто. Надеемся, что такие записи не будут у Вас вызывать особых сложностей. Давайте для примера разберем эту запись более подробно.

<оператор1>; <оператор2>; ...; <операторN> - это понятно что. В символическом виде здесь один за другим перечислены все операторы, которые входят в составной оператор. И, разумеется, они отделяются друг от друга точкой с запятой.

begin и **end –** это два служебных слова. По сути это знаки препинания, их иногда так

и называют – операторные скобки. Т.е., можно сказать и так: операторы, входящие в составной оператор, заключаются в операторные скобки. Просто жуткая фраза, не правда ли? Если перевести ее на обычный язык, то получится примерно следующее: «Предложения», входящие в «абзац» заключаются в специальные символы: символ начала «абзаца» и символ конца «абзаца».

Заметьте, что когда в тексте программы встретится составной оператор, то он будет выполняться компьютером по одному оператору в том порядке, как они записаны внутри составного оператора: сначала первый, затем – второй, и, наконец, в самом конце – последний оператор. Впрочем, так может быть не всегда. Среди операторов могут оказаться более сложные операторы, которые могут повлиять на порядок их исполнения. В частности, это могут быть условные операторы, о которых мы будем говорить позже, на других уроках.

Сейчас Вы должны понять и запомнить, что операторы внутри составного оператора могут оказаться какие угодно, в том числе и составные. Таким образом, с помощью составного оператора можно создать какие угодно сложные конструкции, в том числе и вложенные друг в друга.

А теперь давайте немного поговорим о словах языка РУЛАНГ. Вы наверняка знаете, что в обычном языке слова – это не просто набор символов. Обычно у каждого слова имеется определенный смысл. Одни слова (существительные) обозначают вполне конкретные реальные предметы и вещи. Например, «стакан» или «вода». Другие слова (глаголы) могут означать действия, которые можно выполнить с некоторыми вещами. Например, «налить». Поскольку каждое слово имеет определенный смысл, то из них можно составить вполне осмысленные фразы. Например, из упомянутых слов можно составить фразу что-то вроде: «Вода налить стакан». Несмотря на немного неправильную грамматическую форму этого предложения его смысл, очевидно, будет понятен практически всем.

Так вот, в языке РУЛАНГ также есть свои слова, и среди них есть и «существительные», и «глаголы», и даже «прилагательные». Интересно, а если мы рассмотрим оператор из нашей программы Volatil:

```
H – L;
```

Где тут «существительные» и «глаголы»? Наверное, Вы уже догадались и сами. H и L – это «существительные», т.е. это те данные, которые участвуют в операции. А знак минус – это ни что иное, как «глагол». Он говорит о том, какие манипуляции надо провести над «существительными». Если говорить более строго, то слова в тексте программы обозначают либо данные (они же «существительные»), либо операции (они же «глаголы»), которые над этими данными выполняются.

И действительно, за словом «H» стоят определенные данные, а именно – цены High свечей графика. За словом «L» соответственно стоят данные о ценах Low свечей. Ну, а слово (знак) минус означает то действие (вычитание), которое нужно проделать над этими данными.

Кстати, в большинстве случаев почти 90% и более слов в тексте программы обозначают либо данные, либо действия над этими данными (т.е. либо «существительные», либо «глаголы»). Ведь по своей сути программа это ничто иное, как набор инструкций (или указаний) для компьютера о том, что нужно сделать с теми или иными данными. Вот поэтому, кстати, «предложения» в языке программирования и называются операторами. Потому, что каждое «предложение» (в большинстве случаев) - это и есть операция (или набор операций), которые надо выполнить над данными.

Переменные

Теперь подробнее поговорим о данных и операторах.

Данные в тексте программы «живут» в виде переменных. Кстати, «H» и «L» - это и есть переменные. Что они означают, Вы уже знаете. А какие еще есть переменные? Наверное, Вы уже догадались, что есть еще и такие переменные, как

«O» - Open;
«C» - Close;
«V» - Volume.

Причем в тексте программы каждая из этих переменных может быть записана в полном виде, либо в сокращенном (одной первой буквой слова). Так что запись

H – L; и High – Low;

означают абсолютно одно и то же самое.

И еще один нюанс. Регистр букв в слове не имеет значения. Т.е. запись HIGH, High, high и даже hIGh или higH означает абсолютно одно и то же. Еще говорят так: язык РУЛАНГ является регистро-независимым. Т.е. одно и то же слово, написанное в разных регистрах все равно будет считаться одним и тем же словом, а не разными словами.

Помимо собственных переменных (переменных данных H, L, O, C и V) программист может сделать и свои собственные переменные. Например, такая запись

Volatility = H – L;

означает, что компьютер вычислит разницу между High и Low, и то что получится поместит в переменную, которая называется Volatility. В дальнейшем эту переменную можно будет использовать в дальнейших выражениях и вычислениях.

А имя переменой Volatility не слишком ли длинное? Можете на этот счет не беспокоиться. Язык РУЛАНГ позволяет создавать имена переменных длиной до 20-ти символов. А какие еще существуют грамматические правила написания имен переменных? Их совсем немного. Давайте их все выпишем:

1. Длина имени должна быть не более 20-и символов.

2. В именах можно использовать только определенные символы (а именно: буквы латинского алфавита, буквы русского алфавита, арабские цифры и символ нижнего подчеркивания «_»). Это, в частности, означает, что нельзя в именах использовать пробелы. Если имя вашей переменной состоит из нескольких слов, то тогда между словами вместо пробелов ставьте символ нижнего подчеркивания. Например:

- неправильное имя: Point A
- правильное имя: Point_A

3. Символы, перечисленные в п.2 можно использовать в любом порядке и в любых сочетаниях (русские и латинские буквы и арабские цифры). Есть только одно ограничение: имя переменной НЕ должно начинаться с цифры. Например:

- правильное имя: ZX23N4U
- неправильное имя: 5ZX23N4U

4. Имя переменной не должно совпадать со служебными словами самого языка РУЛАНГ. Эти слова еще называют зарезервированными. Некоторые из этих слов Вы уже знаете: H, V, begin, end. Для того, чтобы узнать, какие еще существуют служебные слова, загляните в электронную справку программы РУМУС2, которая вызывается при нажатии на клавишу F1, и найдите там в разделе «Язык программирования РУЛАНГ» статью, которая так и называется «Зарезервированные слова».

Если хотя бы одно из этих правил будет нарушено, то появится сообщение об ошибке, и Вы не сможете нормально запустить программу своего индикатора до тех пор, пока эту ошибку не исправите.

По поводу имен переменных можно дать один хороший совет. Давайте переменным такие имена, которые были бы краткими, и в то же время отражали их смысл и суть. Вспомните пословицу: «Как Вы яхту назовете назовете, так она и поплывет». Если Вы назовете переменную A34, то потом будете долго думать и вспоминать: а зачем она нужна, и что она тут делает? А если Вы дадите имя четкое и понятное, то и мучиться такими вопросами не будете. Например, если Ваш индикатор генерирует сигналы на вход в рынок, то вполне разумно и логично было бы назвать переменные, содержащие такие сигналы, как Buy и Sell.

Массивы

Говоря о переменных, следует упомянуть о такой их разновидности, как массив. По сути массив – это набор переменных, у которых одно и то же имя. Т.е., если в обычной переменной можно держать только одно число, то в массиве можно держать много чисел под одним и тем же именем. Интересно, если имя будет одно и то же для многих чисел, то как же тогда компьютер поймет, какое именно из этих чисел имеется в виду. Дело в том, что при обращении к какому-либо числу (элементу) массива, к его имени массива добавляется еще и индекс (т.е. номер) этого числа в массиве. Этот номер записывается в квадратных скобках. Например, запись

```
Cls[5]
```

означает, что речь идет о пятом элементе некоторого массива Cls.

Допустим, что мы в массиве Cls будем хранить 20 последних цен Close. В первом элементе массива будет храниться самая свежая (последняя) цена Close. Во втором элементе будет цена Close предыдущего бара или свечи, и так далее. В последнем, двадцатом элементе будет содержаться цена Close бара, который находится на 20 баров левее текущего бара. В таком случае для «загрузки» самого первого элемента массива мы в тексте программы должны записать следующее:

```
Cls[1] = C;
```

Остальные элементы этого массива загружаются аналогично. Как это делается, мы рассмотрим чуть позже – на следующем уроке, когда будем рассматривать функции.

Если же нам понадобится какой-нибудь элемент массива, то тогда его будет достаточно легко извлечь оттуда. Допустим, нам надо, чтобы в переменной «a» оказалось содержимое 15-го элемента массива Cls. Тогда мы в тексте программы запишем следующее:

a = Cls[15];

Прежде чем использовать какой-либо массив, его надо объявить в самом начале программы. Делается это с помощью специального оператора объявления переменных Variable. Для нашего массива Cls это объявление выглядело бы следующим образом:

Variable: array Cls[20];

Оператор составляется следующим образом. Сначала записывается служебное слово Variable с разделителем двоеточие (:), что означает начало оператора объявления переменных. Затем записывается имя переменной Cls, перед которой указывается ее тип: array, т.е. массив. И, наконец, в квадратных скобках указывается размер массива. В данном случае – это 20-ть элементов. В конце оператора, как и положено, должен стоять разделитель точка с запятой (;).

Причем в одном операторе Variable можно объявлять несколько переменных. Тогда они перечисляются через запятую (,) внутри оператора Variable. В нем могут быть объявлены как переменные-массивы, так и обычные переменные. В этом случае перед обычными переменными слово array не пишется. Рассмотрим пример:

Variable: array A1[3,5], B5, C7, array N3[27];

В этом операторе были объявлены 4 переменных. Причем две из них (B5 и C7) являются обычными, а две других – массивами. Массив N3 27-мь элементов. А вот массив A1 является двумерным. Это значит, что каждый элемент в нем имеет не один номер (индекс), а целых два индекса. Таким образом, этот массив имеет 15 элементов, которые можно представить в виде таблицы из 3-х строк и 5-ти столбцов. Первый индекс задает номер строки, а второй – номер столбца.

А зачем нам вообще объявлять переменные и массивы? Дело в том, что работа компьютера с обычной переменной или с массивом существенно отличаются друг от друга. Отличие это проявляется уже хотя бы в том, что для обычной переменной и для массива потребуется совершенно разное количество памяти. Поэтому, чтобы правильно выделить им память, компьютер еще до первого обращения к переменной должен четко знать, что эта конкретная переменная является обычной переменной или же она является массивом. А если это массив, то он содержит такое-то конкретное количество элементов. А откуда компьютер узнает эту информацию? Так вот из оператора Variable, который должен быть в самом начале программы, он это и узнает.

Правда, тут есть одно маленькое исключение. Обычные переменные (не массивы) можно не объявлять в этом операторе. По умолчанию компьютер считает, что если ему в тексте попалась неизвестная для него переменная, то она является обычной. Он тут же выделяет ей нужное количество памяти. Тем более, что это количество памяти для обычной переменной является стандартным и заранее известным.

Операторы

Итак, с данными и с переменными мы немного разобрались. Теперь перейдем к операторам. Операторов, оказывается, существует несколько разновидностей. Вот некоторые

из них:

- математические;
- относительные;
- логические;
- операторы присвоения и вывода.

Остальные операторы мы рассмотрим на других уроках. А сейчас пока рассмотрим эти. Математические операторы вам уже известны. Это:

- сложение – обозначается символом «плюс» (+);
- вычитание – обозначается символом «минус» (-);
- умножение – обозначается символом «звездочка» (*);
- деление – обозначается символом «наклонная черта» (/).

Из этих операторов можно составлять довольно сложные арифметические выражения. При этом порядок действий в этих выражениях будет общепринятый: сначала выполняются действия умножения и деления, а затем – сложения и вычитания. Для того, чтобы изменить порядок действий при вычислении арифметического выражения, в нем можно применять скобки. Например, результатом вычисления выражения:

1 + (2 – 3) * 4 / (5 - (6 + 7))

будет число 1.5

Относительные операторы сравнивают друг с другом две переменных (или два выражения). Существуют следующие относительные операторы:

- меньше (<);
- больше (>);
- равно (=);
- не равно (<>);
- меньше или равно (<=);
- больше или равно (>=).

В результате вычисления любого из этих операторов будет получено логическое число. Логическое число может принимать только одно из двух значений: либо ноль (Ложь, False), либо единица (Истина, True). Например, результатом вычисления выражения:

5 > 3

будет логическая единица (True). А вот противоположное ему выражение

5 < 3

будет иметь значение логического нуля (False).

Давайте, чтобы более наглядно увидеть работу относительных операторов, напишем еще один маленький индикатор в стиле «программирование в одну строку». Назовем этот индикатор Demo1 (это означает демонстрационный индикатор номер 1). В нем действительно будет только одна строка. Вот она:

Close > Open;

Создадим этот индикатор (надеемся, что это не вызовет у Вас особых затруднений), и построим его в отдельной графической области чуть выше графика волатильности. В результате должно получиться что-то, похожее на рис. 2.1.

Рис. 2.1

На рис 2.1 видна какая-то «зубчатая» линия индикатора Demo1. В тех местах, где величина Close свечи больше величины Open этой же свечи, индикатор равен единице. А для тех свечей, у которых наоборот, Close меньше Open, индикатор показывает ноль. Что это означает? Если Close больше Open, то свеча белая, в противном случае – черная. Тогда получается, что этот индикатор показывает цвет свечи. Если он равен единице, то тогда свеча белая, а если он равен нулю, то свеча – черная.

Кстати, те логические значения ноль и единица – это на самом деле самые обыкновенные ноль и единица, и их можно потом использовать в обычных арифметических выражениях. Например, тот единственный оператор в индикаторе Demo1 можно записать в несколько ином виде:

2 * (Close > Open) – 1;

То есть мы просто взяли прежнее выражение, домножили его на 2, и затем вычли единицу. Смысл этих манипуляций заключается в следующем. Теперь, когда свеча будет черной, индикатор будет показывать теперь не ноль, а минус единицу. В то же самое время, когда свеча будет белой, показания индикатора останутся прежними: плюс единица. Вы можете это проверить, подставив значение ноль или единица вместо выражения в скобках.

Давайте исправим текст программы индикатора, и убедимся в том, что все это действительно так. Кстати, чтобы быстро вызвать текст индикатора для редактирования, следует просто щелкнуть на кривой этого индикатора на графике, а затем в появившихся в панели задач свойствах этого индикатора нажать на кнопку «Редактор индикаторов». См. рис. 2.2. Впрочем, можно даже не нажимать на эту кнопку, а просто внести нужные изменения в окне, которое расположено чуть выше этой кнопки.

В результате таких исправлений индикатор изменяется не между нулем и единицей, а между минус единицей и плюс единицей, и это видно на графике на рис. 2.2.

Логические операторы уже работают только с логическими числами. Этих операторов всего две штуки:

- AND (логическое И);
- OR (логическое ИЛИ).

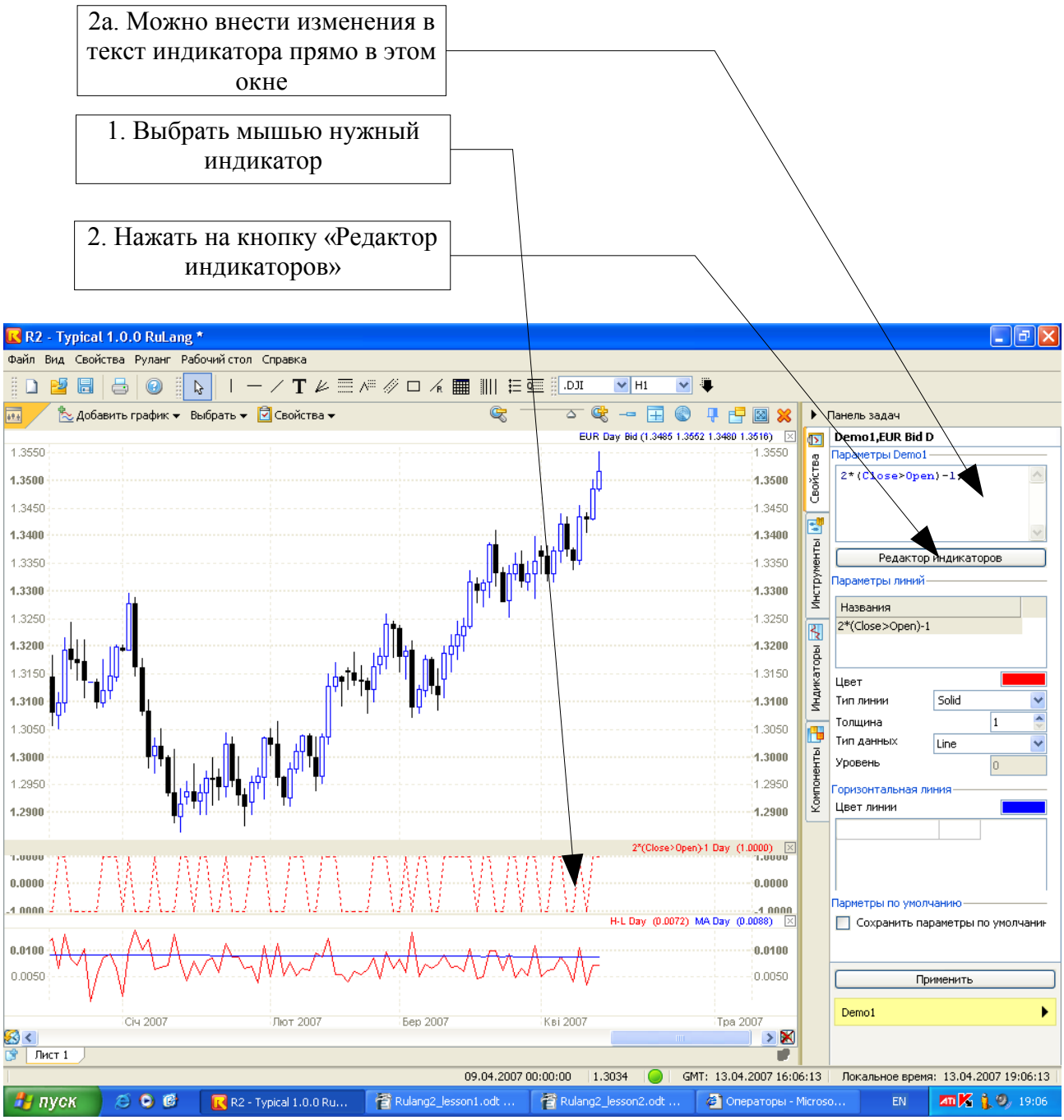

Рис. 2.2

Чтобы лучше понять работу этих операторов давайте рассмотрим конкретный пример. Допустим, у нас имеется такой оператор:

(C > O) AND (H > 1.3200);

Что он означает? Первое выражение в нем (которое стоит слева от слова AND) нам уже знакомо. Это индикатор цвета свечи. Он равен единице когда свеча белая, и равен нулю

когда свеча черная. Справа от слова AND находится другое выражение, смысл которого вполне очевиден. Оно будет истинным тогда, когда цена High свечи окажется выше уровня 1.3200. Так вот, все это выражение будет истинным только тогда будет истинным выражение И слева от слова AND, И справа от слова AND (одновременно). Оно будет равно единице только тогда, когда свеча будет белой и при этом одновременно ее High будет выше уровня 1.3200.

Давайте проверим это. Создаем индикатор по имени Demo2, в котором будет только одна строка с одним оператором, который мы только что разобрали. Полученный индикатор поместим в отдельную графическую область, которая будет располагаться выше области, в которой находится индикатор Demo1. Как Вы помните, этот индикатор показывает цвет свечи. Если Вы все сделали правильно, то у Вас должно получиться примерно то, что изображено на рис. 2.3.

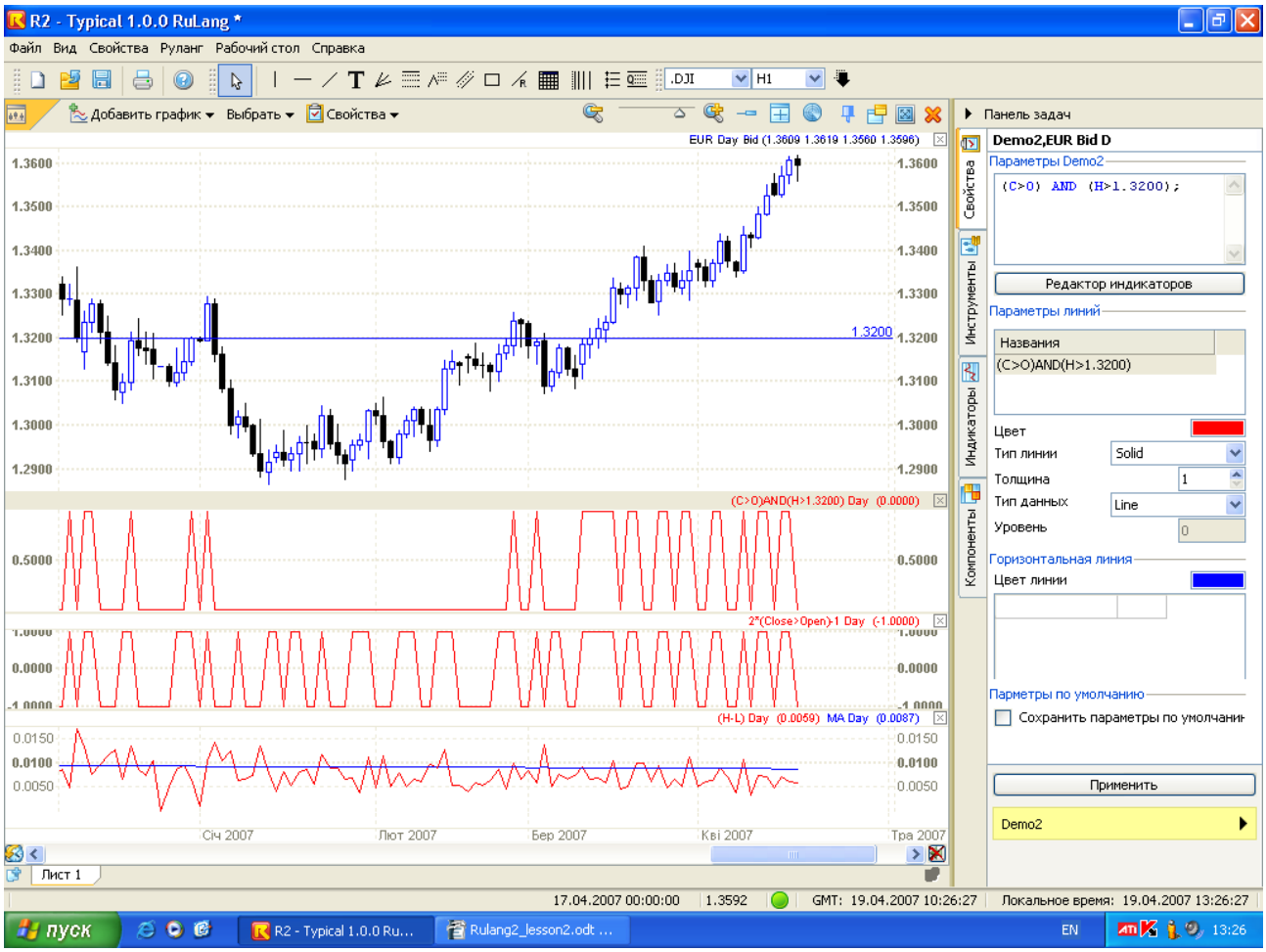

Рис. 2.3

На этом рисунке очень хорошо видно место, где цены были ниже уровня 1.3200. Как видите, на этом участке наш индикатор ничего не показывает (вернее, он показывает

логический ноль). А вот на других участках (там, где цены были выше уровня 1.3200), он показывает цвет свечи.

Давайте немного изменим этот индикатор, чтобы увидеть, как работает оператор OR. Перепишем его в следующем виде:

$$(C > O) \text{ OR } (H < 1.3200);$$

Здесь мы сделали два изменения. Заменили слово AND на слово OR, и во втором выражении знак «больше» заменили на знак «меньше». После этих изменений внешний вид картинки должен измениться. См. рис. 2.4.

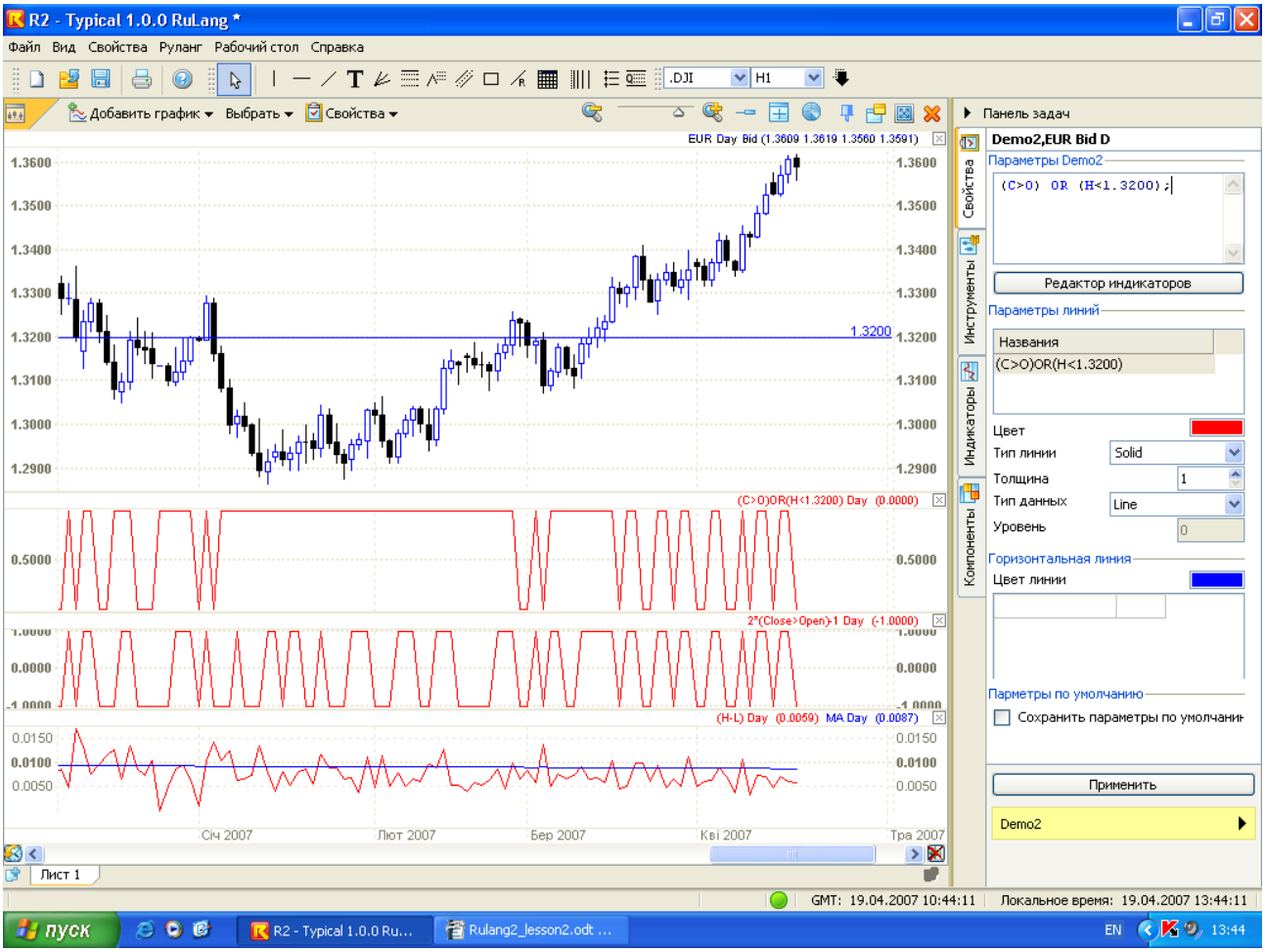

Рис. 2.4

Сравните этот рисунок с рис. 2.3. Теперь это выражение, которое содержится в индикаторе, будет равно логической единице в том случае, если хотя бы одно из выражений (ИЛИ то, что слева, ИЛИ то, что справа от слова OR) будет истинным. Т.е., говоря другими словами, это выражение будет истинным, когда либо свеча будет белой, либо эта же свеча будет ниже уровня 1.3200, либо же и в том и в другом случае одновременно. В результате на графике мы видим (в центре) то место, где цены были ниже уровня 1.3200. Там индикатор равен единице. В других же местах (там, где цены были выше уровня 1.3200) этот индикатор по-прежнему показывает цвет свечи.

Оператор присвоения Вам уже знаком. Просто мы никогда раньше не называли его так. Выглядит он, например, таким образом:

```
A = (H + L) / 2;
```

Знак равенства (=) это и есть оператор присвоения. Сначала вычисляется то выражение, которое находится справа от знака равенства, а затем то, что получилось присваивается той переменной, которая находится слева от него. Вот, пожалуй, и все, что можно сказать об этом операторе.

Оператор вывода, как это ни странно, тоже Вам знаком. Он внешне напоминает оператор присвоения и выглядит, например, таким образом:

```
(H + L) / 2;
```

Смысл его очень простой. Сначала вычисляется выражение, а затем то, что получилось выводится на экран в виде графика.

Урок 3

Функции

Что такое функция? По сути это тоже оператор, только записывается он несколько иначе, чем те операторы, которые мы рассматривали выше. Чтобы было понятно, о чем идет речь, давайте рассмотрим пару примеров функций.

Функция pow

Это функция возведения в степень. В тексте программы эта функция выглядит примерно так:

```
pow(X,3) ;
```

Это обычный способ записи функции. Сначала записывается имя функции (в данном случае – это pow), а затем за ним в скобках следует перечень аргументов. Аргументы перечисляются через запятую. По сути аргументы – это данные, которые передаются функции. Затем функция, получив эти данные, проводит над ними определенные вычисления, и то, что у нее после этого получилось, возвращает назад в программу в качестве результата. В данном конкретном примере функция pow – это функция возведения в степень. Она принимает аргументы X и 3, затем первый аргумент возводит в степень второго аргумента. Т.е. вычисляет величину X^3 («икс в кубе»). После этого полученный результат возвращается в программу. В дальнейшем этот результат можно использовать как угодно: например, присвоить его какой-нибудь переменной или же вывести на экран.

Функция sqrt

Это функция извлечения квадратного корня. В тексте программы эта функция выглядит примерно так:

```
sqrt(X) ;
```

Эта функция имеет единственный аргумент, из которого и производится извлечение квадратного корня. В данном приведенном примере вычисляется выражение $\sqrt{X}$.

Кстати, с помощью этих двух функций, pow и sqrt можно вычислить модуль (абсолютную величину) числа. Для этого в тексте программы можно, например, сделать такую запись:

```
B = sqrt(pow(A,2));
```

Это выражение будет вычисляться компьютером следующим образом. Сначала число A будет возведено в степень два (в квадрат). При этом каким бы ни было число A (положительным или отрицательным), оно станет положительным. Затем из полученного числа будет извлечен квадратный корень. При этом знак его не изменится. В результате всех этих вычислений будет получено то же самое число A, только теперь оно гарантированно будет положительным. Таким образом, по окончании вычисления всего этого выражения в переменной B будет находиться модуль (абсолютная величина) переменной A.

А какие еще бывают функции? В языке РУЛАНГ на данный момент существуют около 20-ти собственных (встроенных функций). Их условно можно разделить на три группы: математические функции, функции даты/времени и функции индикаторов. Давайте их рассмотрим немного подробнее.

Математические функции

К математическим функциям относятся следующие:
pow – функция возведения в степень;
sqrt – функция извлечения корня;
sum – функция вычисления суммы;
cum – функция вычисления кумулятивной суммы;
log – функция вычисления логарифма;
Ln – функция вычисления натурального логарифма;
hhv – функция вычисления максимального значения;
llv – функция вычисления минимального значения;
ref – функция ссылки;
cross – функция пересечения.

Функции pow и sqrt мы уже рассматривали, поэтому сразу перейдем к функции sum.

Функция sum подсчитывает сумму по заданной переменной за заданный период времени. Записывается она следующим образом (пример):

```
sum(C,20);
```

В этом примере подсчитывается сумма цен закрытия (Close) за последние 20-ть баров. В качестве первого аргумента функции указывается переменная (массив) сумма элементов которого будет подсчитываться, а в качестве второго аргумента указывается сколько элементов этого массива будет участвовать в вычислении этой суммы. Здесь не случайно переменная C названа массивом. Дело в том, что по сути любая переменная является массивом по времени. Это означает, например, что все та же переменная C имеет много разных значений для разных баров. Т.е., C – это временной массив и он содержит столько элементов, сколько имеется баров на графике. Кстати, и любая другая переменная, вычисленная в языке РУЛАНГ по сути также будет являться временным массивом.

Давайте подробнее рассмотрим работу этой функции на примере. Если мы вычислим величину

```
sum(C,20)/20;
```

то это будет не что иное как среднее арифметическое цен закрытия за последние 20 баров. А поскольку эта величина будет просчитана по всему временному массиву (т.е. по всем барам графика), то тогда такую величину можно с полным основанием назвать простой скользящей средней. Давайте этим воспользуемся, и немного доработаем наш индикатор Volatil. Сейчас он вычисляет только саму волатильность, но ее усреднение не производит. Из-за этого нам приходилось после построения этого индикатора проводить еще одно дополнительное

действие: строить по нему скользящую среднюю с периодом 260. Чтобы этого не делать, давайте «встроим» эту скользящую среднюю в наш индикатор.

Таким образом, его текст должен выглядеть, например, таким образом:

```
Vol = H – L;
Vol;
sum(Vol,260)/260;
```

Первый оператор вычисляет ту самую волатильность и результат запоминается в переменной Vol. Далее следующий оператор выводит переменную Vol на экран в виде графика. И, наконец, последний оператор вычисляет скользящую среднюю от волатильности, и результат вычислений также выводит на график.

Давайте внесем нужные изменения в индикатор Volatil, но только теперь сделаем это не так, как в прошлый раз (через вызов списка индикаторов), а по другому – с помощью компонента «Редактор индикаторов». Давайте рассмотрим этот процесс по шагам (см. рис. 3.1).

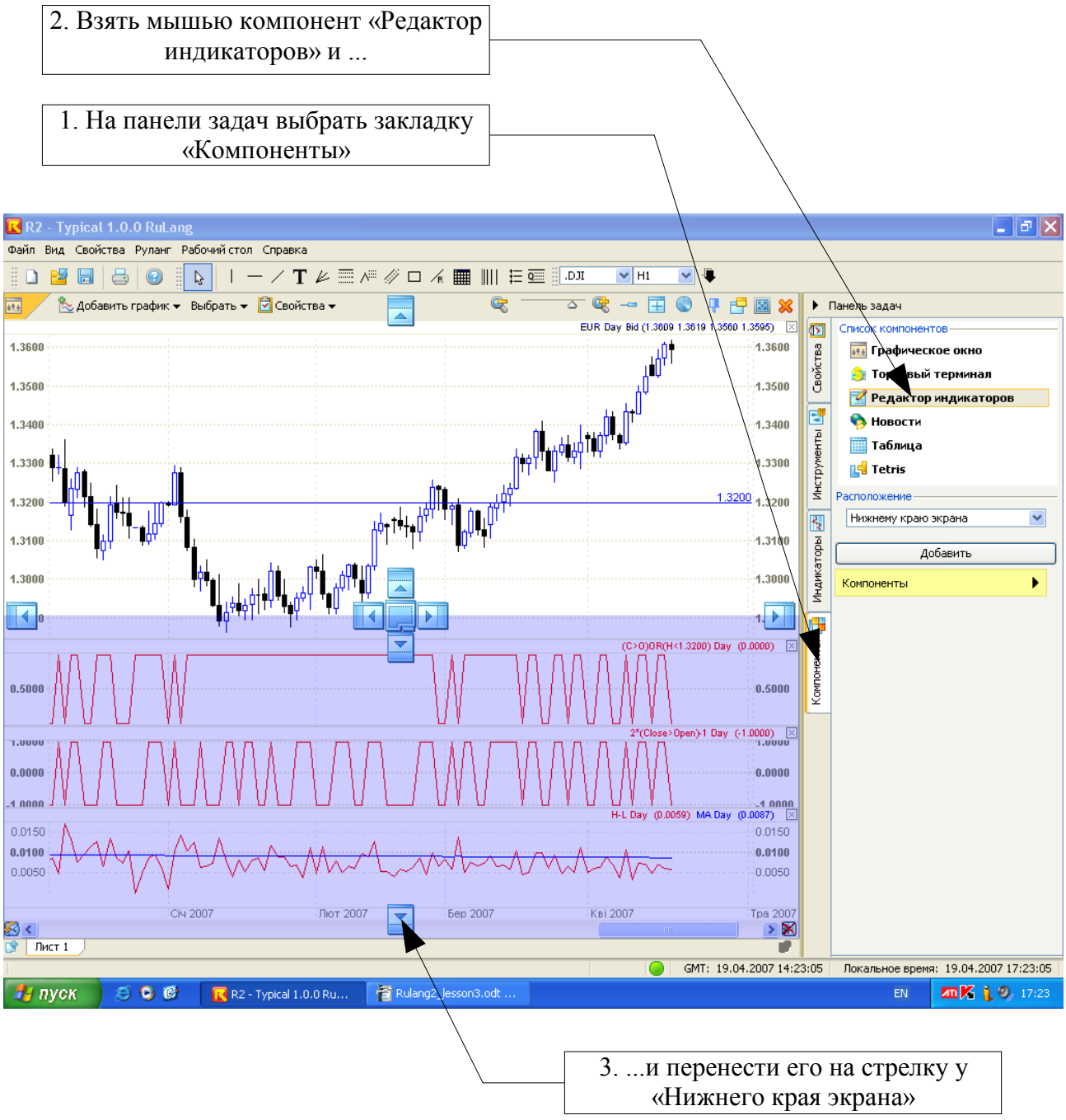

Рис. 3.1

Сначала на панели задач выбираем закладку «Компоненты». Затем берем мышью компонент «Редактор индикаторов» и переносим его на стрелку у «нижнего края экрана». В результате мы увидим примерно то, что показано на рис. 3.2.

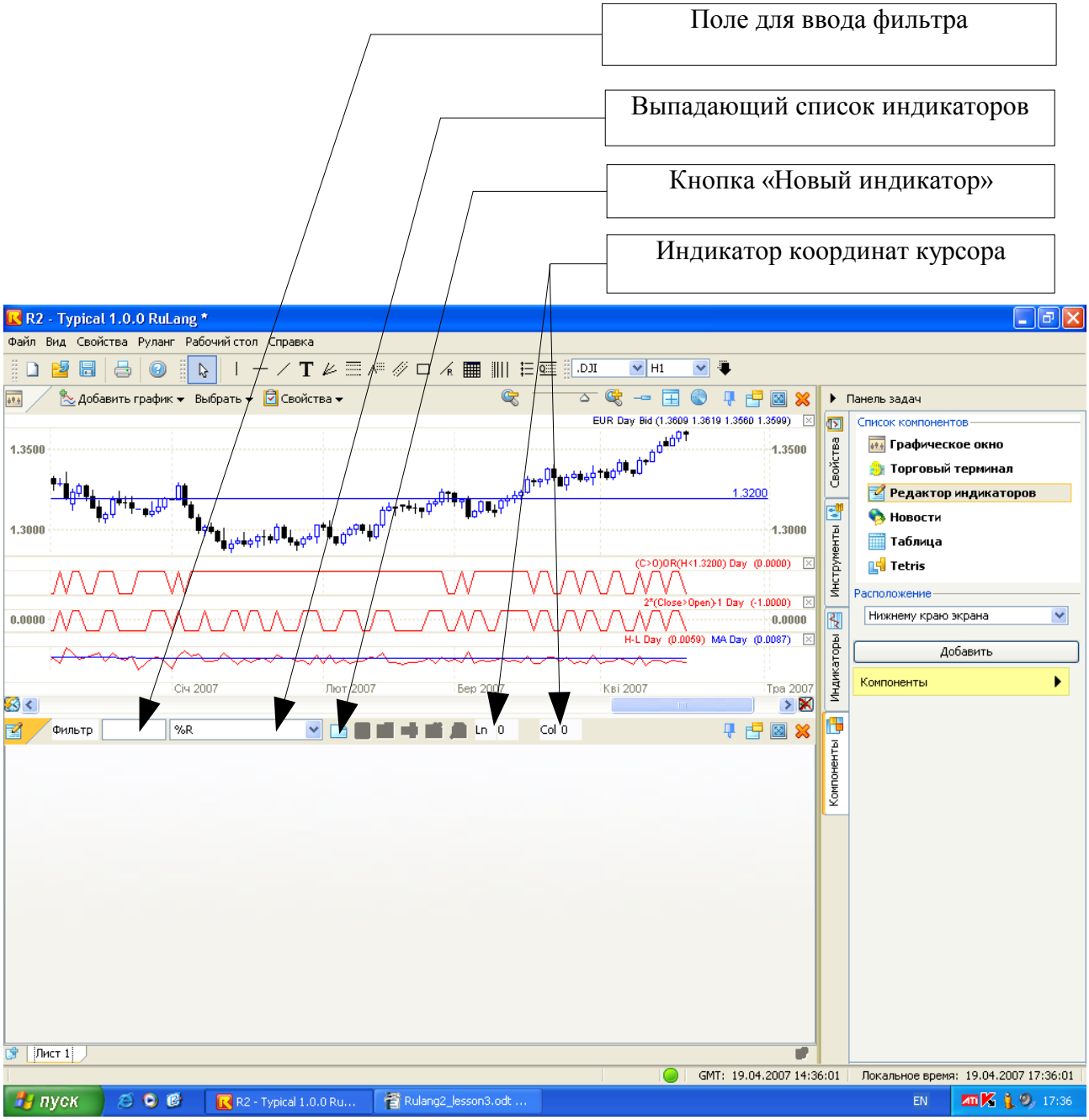
Поле для ввода фильтра
Выпадающий список индикаторов
Кнопка «Новый индикатор»
Индикатор координат курсора
R2 - Typical 1.0.0 RuLang *
Файл Вид Свойства Руланг Рабочий стол Справка
Добавить график
Выбрать
Свойства
Панель задач
Список компонентов
Графическое окно
Торговый терминал
Редактор индикаторов
Новости
Таблица
Tetris
Расположение
Нижнему краю экрана
Добавить
Компоненты
Свойства
Инструменты
Индикаторы
Компоненты
Фильтр
%R
Ln 0
Col 0
Лист 1
GMT: 19.04.2007 14:36:01
Локальное время: 19.04.2007 17:36:01
пуск
R2 - Typical 1.0.0 Ru...
Rulang2_lesson3.odt ...
EN
17:36

Рис. 3.2

Внизу экрана появилось окно, предназначенное специально для редактирования индикаторов. В его верхней строке имеются (слева направо):

- поле для ввода фильтра;
- выпадающий список индикаторов;
- кнопки редактирования;
- индикатор координат курсора.

Все эти элементы (кроме кнопок) точно такие же, как и в окне «Правка» (рис. 1.3) и «Список индикаторов» (рис. 1.2), мы их уже подробно рассматривали. А вот кнопки немного отличаются. Причем, поскольку в это окно не загружено ни одного индикатора, то в активном состоянии сейчас находится только одна кнопка создания нового индикатора. Давайте загрузим наш индикатор Volatil следующим образом (см. рис. 3.3).

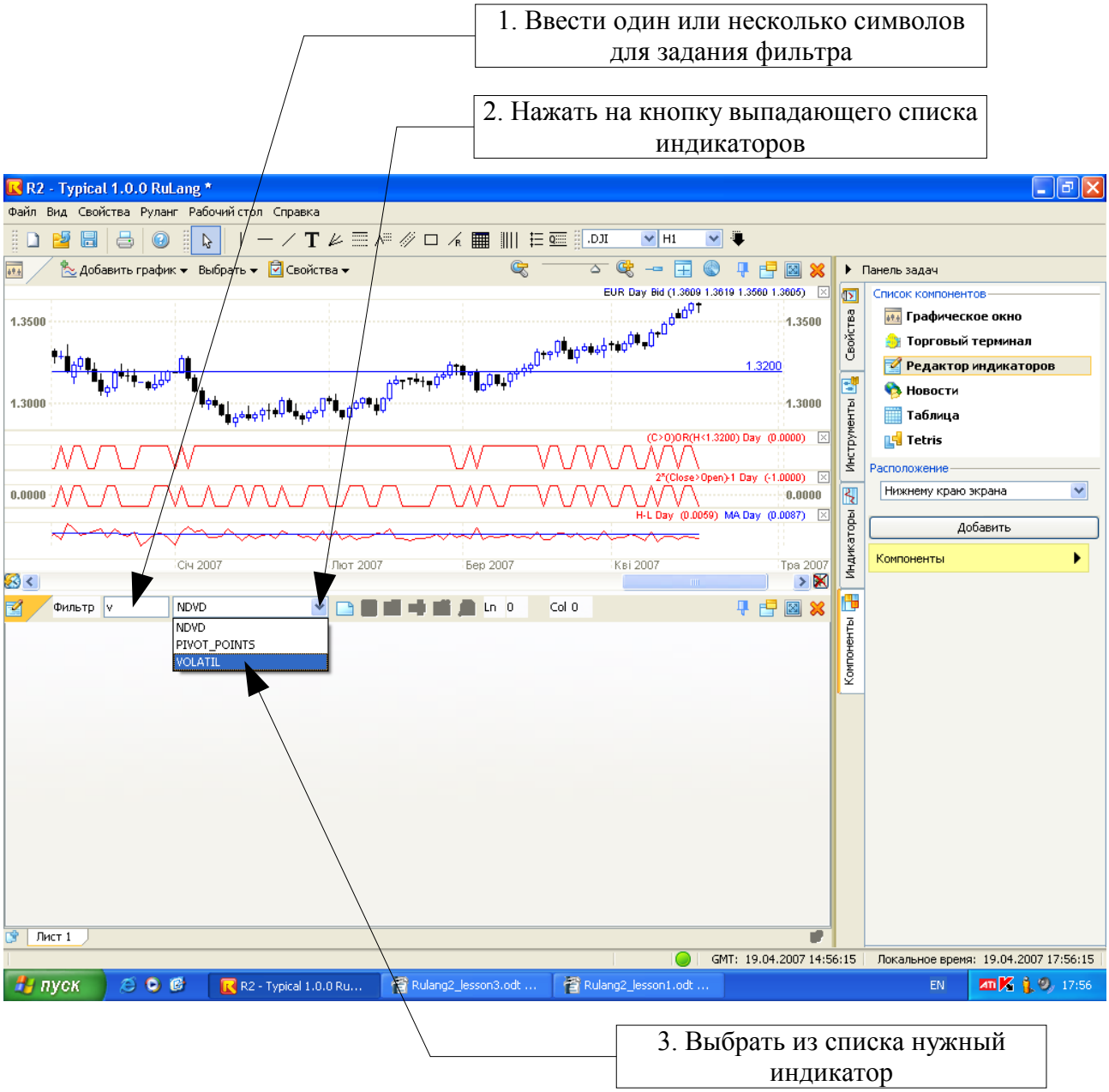

Рис. 3.3

Сначала можно (но это не обязательно) ввести в поле фильтра одну или несколько букв из имени того индикатора, который нам нужен. Поскольку название нашего индикатора Volatil, то мы можем ввести, например, его первый символ (v). Затем раскрываем выпадающий список индикаторов. Заметьте, что он сейчас очень маленький (только три индикатора), поскольку он сейчас просматривается через фильтр. В этом списке присутствуют только те индикаторы, которые имеют в своем имени букву «v». Есть в этом списке и наш индикатор Volatil. Выберем его. Если все сделано правильно, то Вы увидите примерно то, что изображено на рис. 3.4.

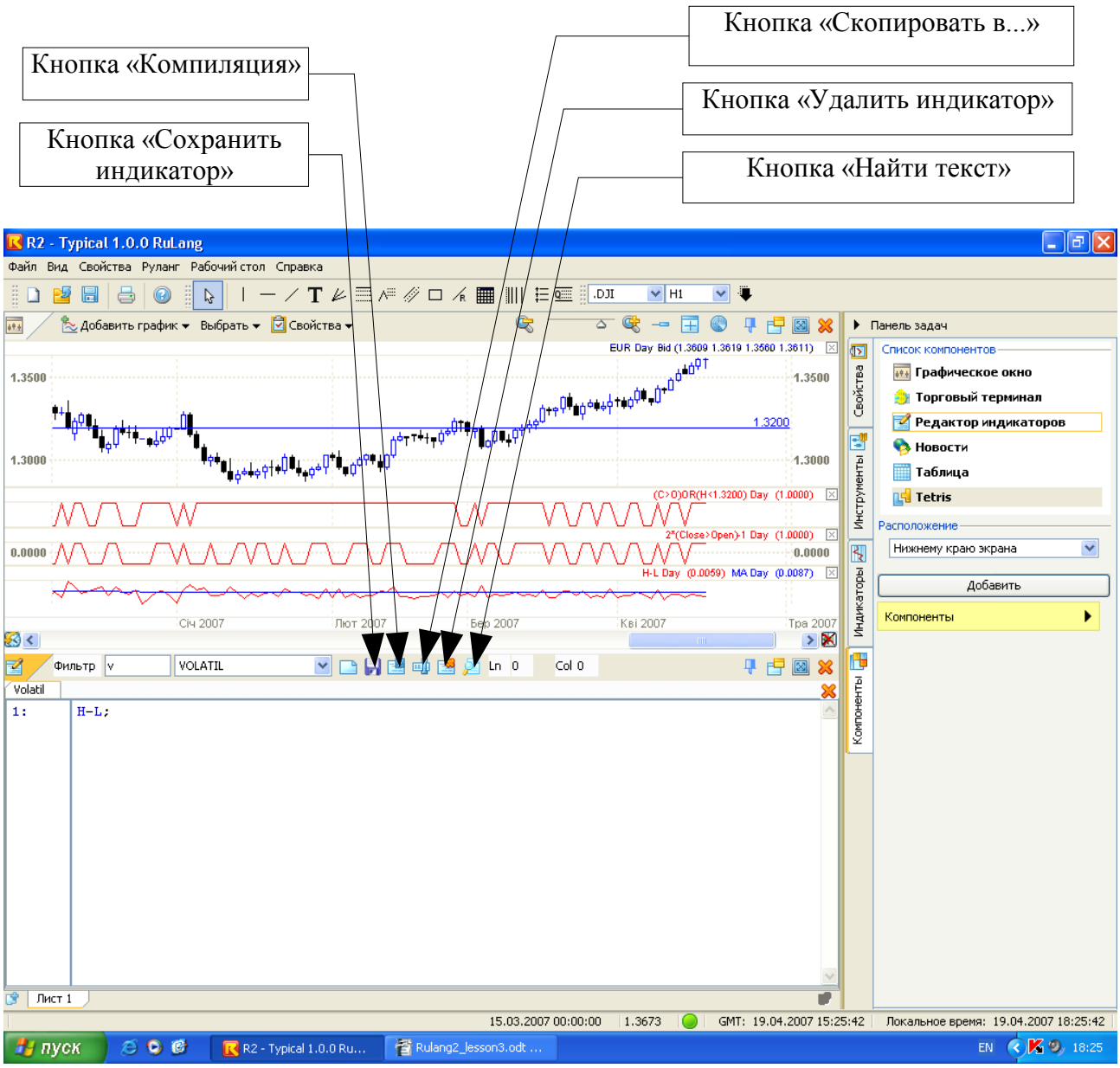

Рис. 3.4

Здесь видно, что текст индикатора Volatil загрузился в окно индикатора, и теперь его можно спокойно изменять. Но прежде чем это сделать, давайте сперва рассмотрим кнопки редактора, которые сейчас (см. рис. 3.4).

Кнопка «Сохранить индикатор» сохранит текст измененного индикатора на жестком диске. Однако при этом не будет производиться проверка правильности внесенных изменений с точки зрения грамматики и синтаксиса языка РУЛАНГ. При таком сохранении в тексте индикатора могут оказаться ошибки.

При нажатии на кнопку «Компиляция» текст индикатора не только будет сохранен на диске, он будет предварительно проверен на отсутствие ошибок. Причем индикатор не будет сохранен до тех пор, пока все ошибки не будут исправлены.

Если мы хотим создать очень похожий индикатор с другим именем, то тогда следует нажать на кнопку «Скопировать в ...». В результате мы будем иметь точную копию исходного индикатора, но с другим именем, которое мы укажем. Теперь мы спокойно можем менять его текст на свое усмотрение.

С кнопкой «Удалить индикатор» нужно быть очень осторожным. При нажатии на нее будет удален не только индикатор, но и файл с его текстом с диска.

Кнопка «Найти текст» используется для облегчения работы с текстом программы. Если, напрмер, текст программы достаточно большой, а нам нужно найти переменную «A5», причем ее имя может встречаться в программе не один раз, то согласитесь, что вручную этот поиск будет сделать не так уж и просто. А вот с помощью этой кнопки нужный текст находится очень быстро и легко, причем не в одном месте, а везде где он встречается по всему тексту программы.

Итак, теперь давайте изменим текст индикатора Volatil, чтобы он самостоятельно вычислял и показывал скользящую среднюю от волатильности, и нажмем на кнопку «Компиляция» (см. рис. 3.5). Тут же появится диалоговое окно с предупреждением о том, что наш индикатор Volatil используется (т.е. построен на экране), и задается вопрос о том, что мы хотим перестроить (перекомпилировать) наш индикатор. Если мы в этом уверены, то нажимаем на кнопку «Yes».

После проведенных манипуляций может показаться, что ничего не изменилось во внешнем виде нашего индикатора Volatil. Однако это не так. На самом деле на экране в одной и той же графической области построены два индиктора: Volatil и MA от него. Причем индикатор Volatil имеет имеет свою собственную линию скользящей средней, которая один к одному совпала с линией MA, и именно поэтому мы ее не видим. Если убрать индикатор MA (поскольку она нам теперь не нужен), то тогда на экране останется линия скользящей средней, которая встроена в индикатор Volatil.

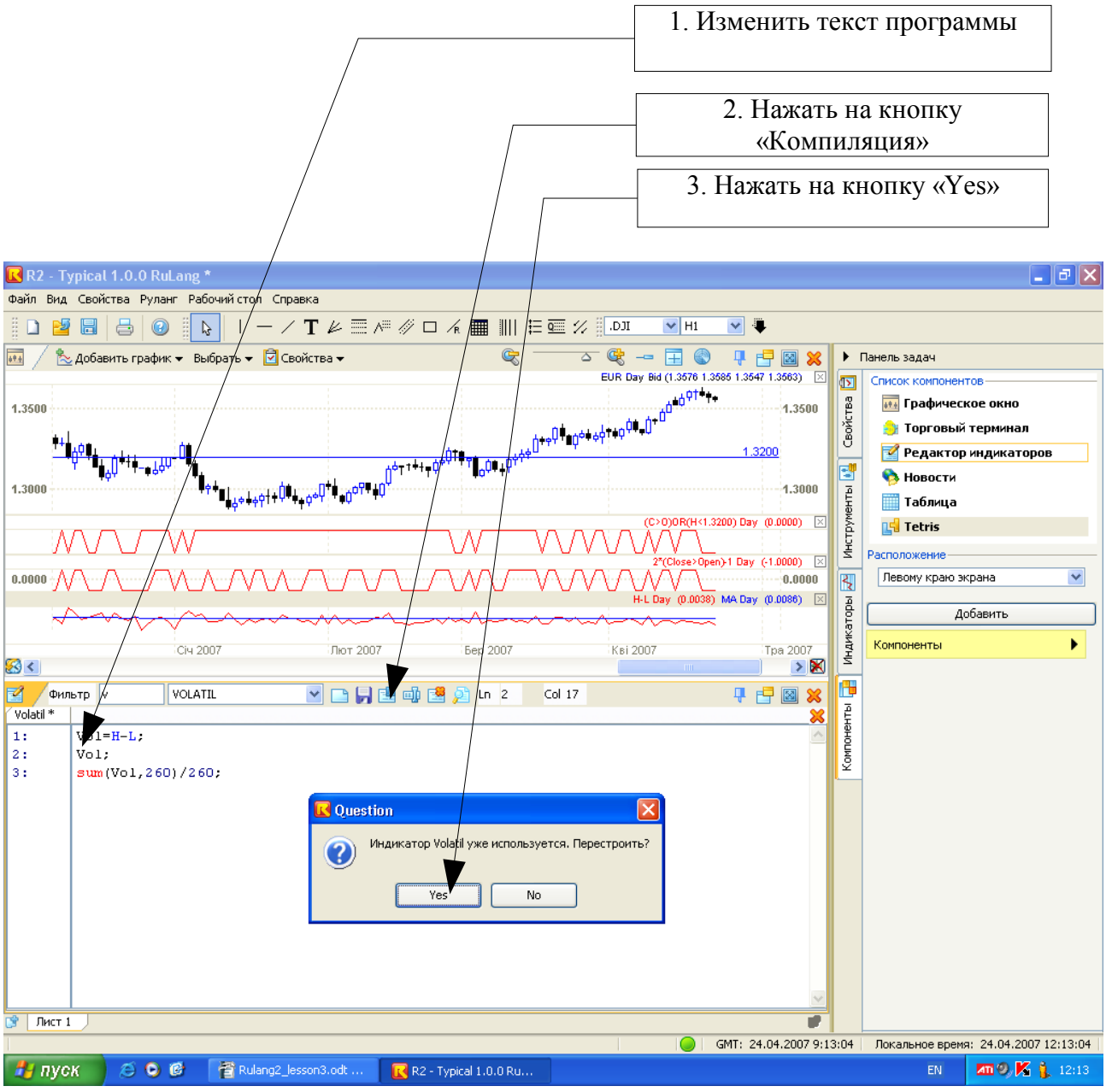

Рис. 3.5

Функция Ln – возвращает значение натурального логарифма.

Синтаксис : ln(A);
где A - аргумент, основание равно числу Эйлера
пример, Ln(close);

Функция Log – возвращает логарифм по указанному основанию.

Синтаксис : Log(A,X);
где A - аргумент, X – основание
пример, Log(close, 2);

Функция cum выполняет подсчет кумулятивной суммы.

Кумулятивная сумма – это сумма некоторой величины, которая подсчитывается нарастающим итогом от начала графика. Например, оператор

cum(1);

будет вычислять на каждом баре сумму с первого до текущего бара от того выражения, которое стоит в скобках. В скобках у нас стоит константа, равная единице, т.е. на каждом баре выражение в скобках будет равно единице. Кумулятивная сумма этой величины на, скажем, 126 баре будет равна ее сумме на этих всех 126 барах, т.е. в данном случае это будет сумма из 126-ти констант, каждая из которых равна единице. Как Вы уже догадались, выражение cum(1), просто-напросто равно номеру текущего бара: на 1-м баре оно будет равно единице, на втором – оно будет равно двум и т.д.

Интересно, а что нам покажет график, построенный с помощью оператора:

cum(C > O);

Если Вы помните, в скобках стоит выражение, которое равно единице в том случае, если свеча белая. А если свеча черная, то тогда это выражение равно нулю. Получается этот оператор подсчитывает количество белых свечей нарастающим итогом. На самом правом краю графика этот оператор покажет число белых свечей, которое имеется на всем графике данной валютной пары.

Давайте посмотрим, как будут выглядеть на графике кривые, построенные этими двумя операторами. Для этого воспользуемся нашей старой программой Demo1, и заменим ее текст на следующий:

```
cum(1);
cum(C > O);
```

Если Вы все сделали правильно и не забыли скомпилировать программу, то Вы увидете примерно картину, которая показана на рис. 3.6.

Из графика этих кривых видно, что график валюты имеет 530 свечей, и из них 263 свечи – белые. Почти половина свечей на графике белые. Соответственно, другая половина свечей должны быть черными.

Интересно, а всегда ли выполняется это соотношение «50 на 50» между белыми и черными свечами? Для того, чтобы ответить на данный вопрос давайте несколько модифицируем нашу программу, чтобы увидеть, например, отношение числа белых свечей к их общему количеству . Вот текст этой программы:

```
A = cum(1);
B = cum(C > O);
White_Ratio = B/A;
White_Ratio;
```

Надеемся, что текст этой программы Вам полностью понятен. Давайте его введем в текст программы Demo1 и скомпилируем ее. В результате получится примерно то, что изображено на рис. 3.7. Обратите внимание: мы сжали график валюты таким образом, чтобы он весь поместился на экране.

Как видно в левой части графика, где статистических данных было мало, это отношение было неустойчивым, хотя явно стремилось к величине 0.5. Ближе к концу графика, где в вычислении этого отношения используется достаточно много данных, видно, что оно примерно равно 0.5. Вообще-то это общеизвестный факт: белые и черные свечи следуют друг за другом в основном случайным образом, и их соотношение равно с среднем «50 на 50». Ну что ж, вот и мы экспериментально еще раз воочию убедились в этом.

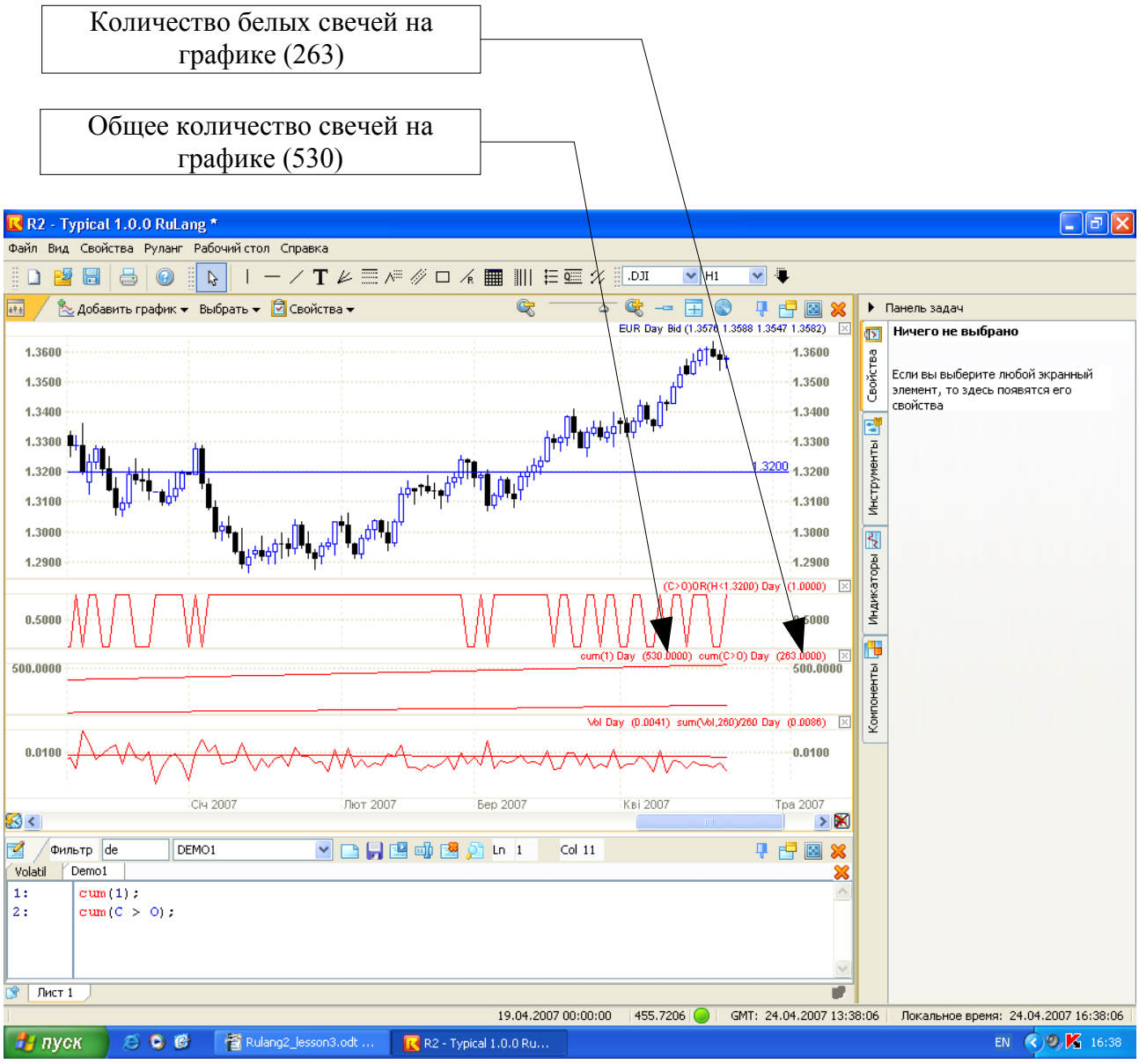

Рис. 3.6

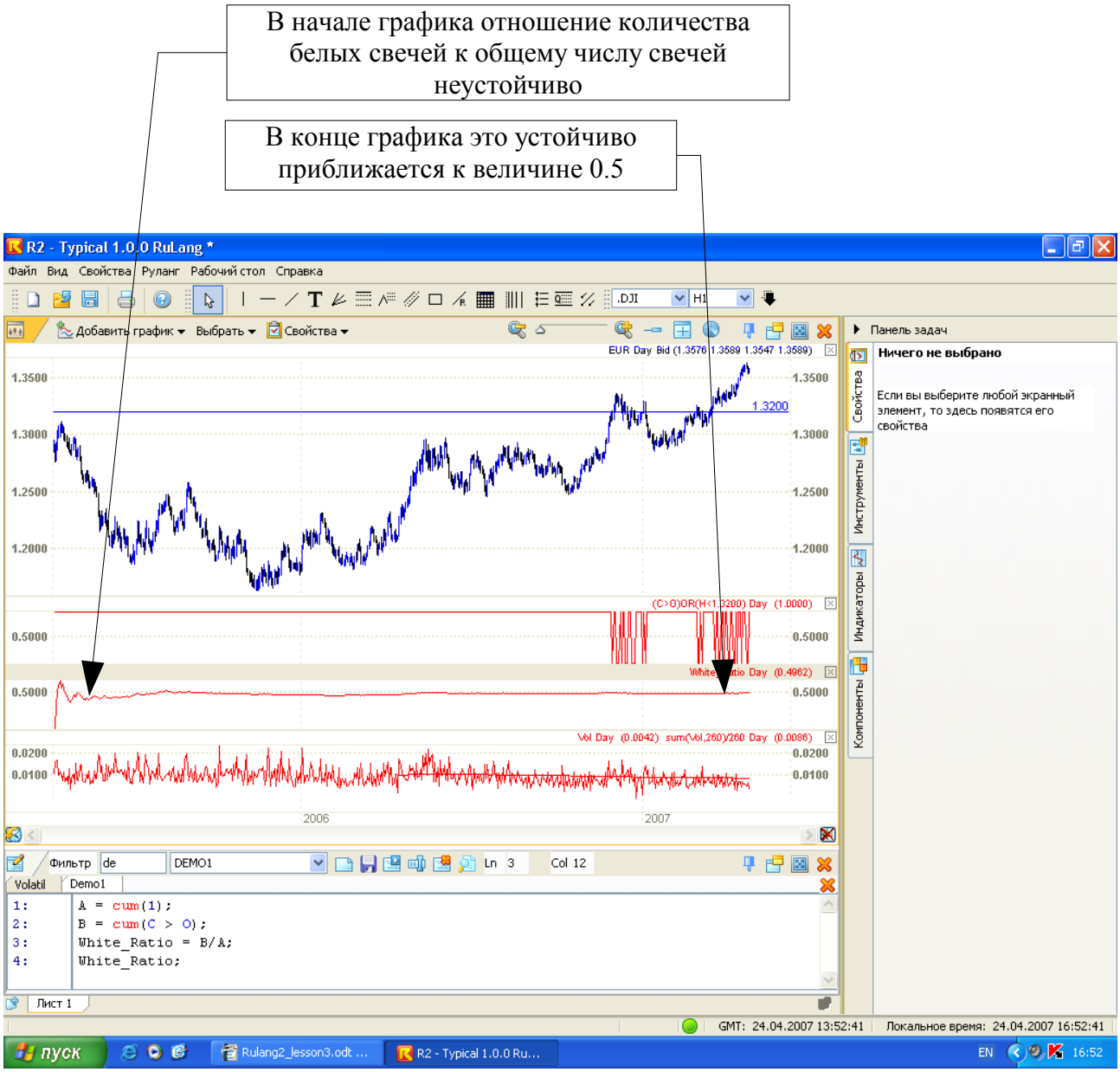

Рис. 3.7

Функции hhv и llv вычисляют максимальные и минимальные значения заданных выражений за заданное число баров. Например, рассмотрим программу, состоящую из двух операторов:

```
hhv(H,20);
llv(L,20);
```

Первый оператор вычисляет максимальное значение цены High за последние 20 баров. Аналогично, второй оператор определяет минимальное значение цены Low за те же 20

баров. По сути программа, состоящая из двух этих операторов, будет рисовать на экране так называемый PriceChannel (индикатор ценового канала) или сокращенно – PC. Давайте построим этот индикатор. Его текст у нас уже есть, а назовем мы его My_PC (мой ценовой канал).

Итак, создаем новый индикатор по имени My_PC, вводим в него текст, компилируем и строим его на графике EBPO. Если вы все сделали правильно, то у Вас должно получиться примерно то, что изображено на рис. 3.8. Если хотите, то для проверки можно наложить на график настоящий индикатор PriceChennal с периодом 20, и он должен идеально совпасть с нашим индикатором.

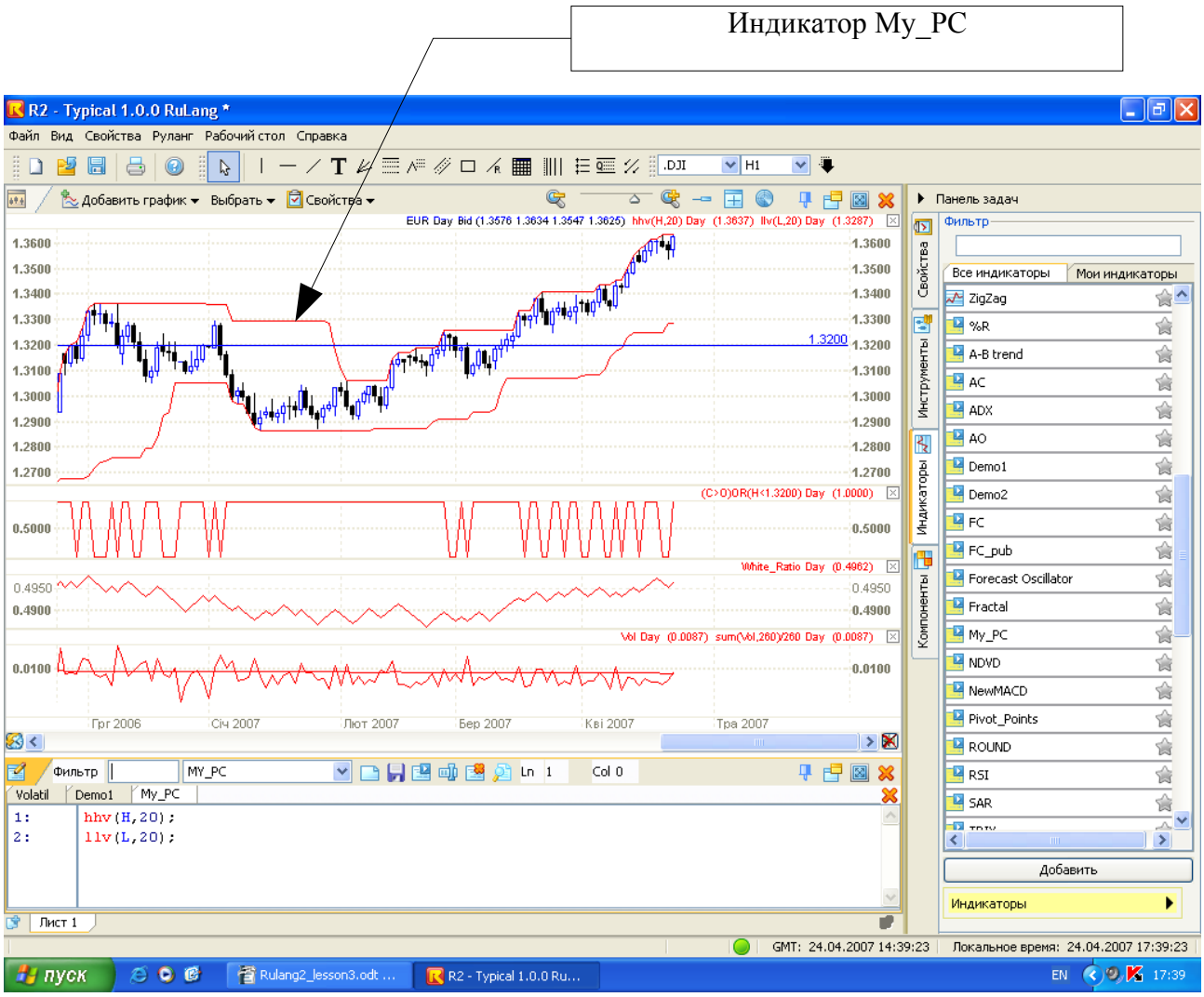

Рис. 3.8

Функция ref позволяет при вычислении выражения для текущего бара использовать данные другого бара, сдвинутого от текущего на заданное число баров. Например, оператор

ref(H, -20);

берет значение величины High того бара, который находится на 20-ть баров левее (ранее) текущего бара. Эта функция позволяет нам легко построить самим такой известный индикатор, как моментум (Momentum). Текст его программы состоит всего-навсего из одного оператора:

C – ref(C,-20);

Из сегодняшней цены Close вычитается та цена Close, которая была 20-ть баров назад. Давайте запишем этот оператор в текст программы индикатора, и назовем его My_Mom (Мой моментум). А после этого построим его в отдельной графической области, расположенной непосредственно под ценовым графиком, предварительно убрав из нее уже ненужный нам индикатор Demo2.

Если все сделано правильно, то у Вас должно получиться примерно то, что изображено на рис. 3.9. Вы можете проверить правильность этого индикатора, если наложите на график индикатор настоящего Momentum'a с периодом 20. Его кривая должна полностью совпасть с кривой нашего индикатора.

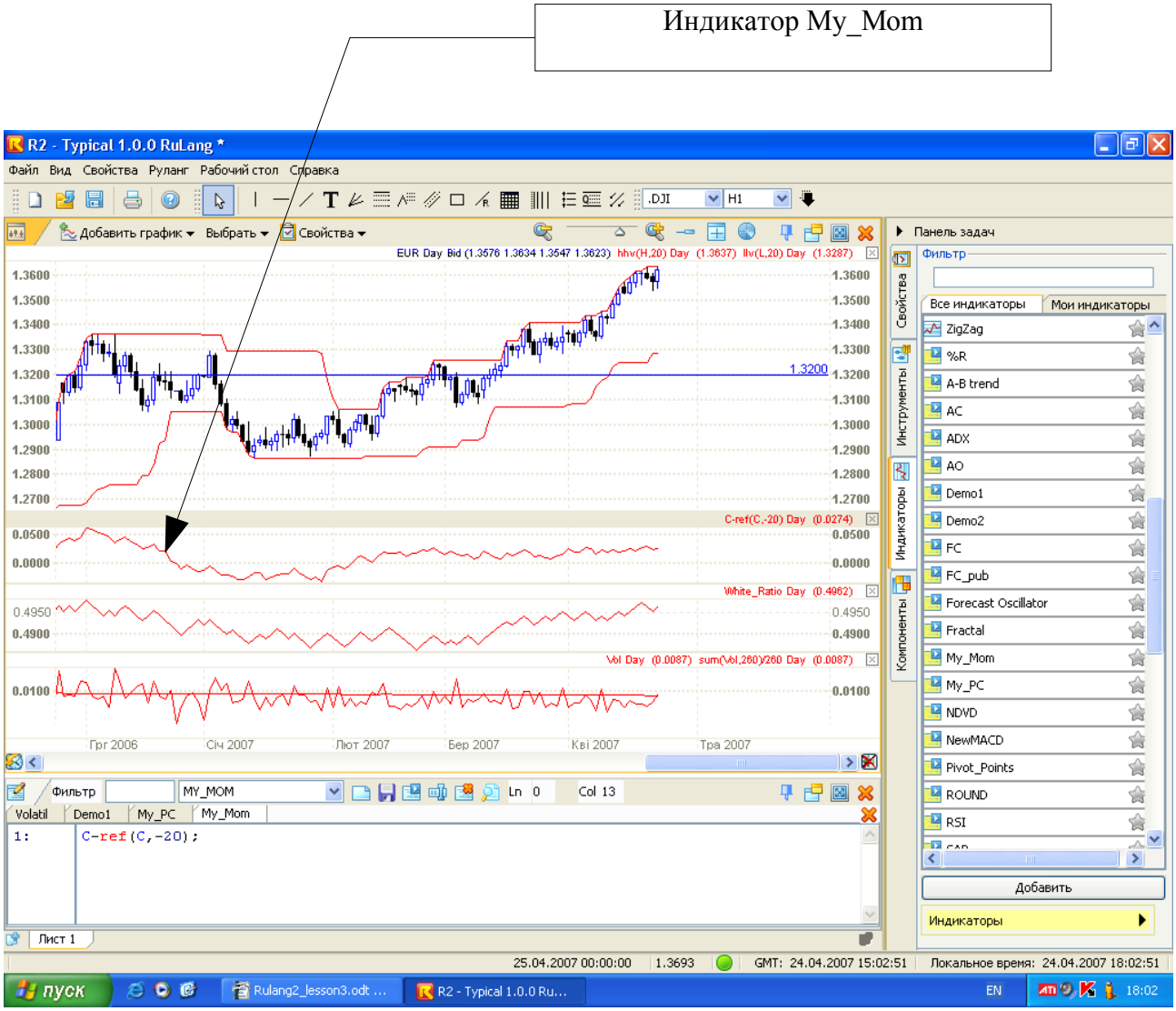

Рис. 3.9

С помощью функции ref мы можем ссылаться на данные со сдвигом на несколько периодов относительно текущего значения. В качестве данных могут быть как параметры свечи, так и значения какой-либо функции или переменной. Если мы хотим, что бы в качестве первого параметра использовать переменную, то обязательно необходимо при описании данной переменной указать ей тип $data. Например,

```
variable : f1($data);
f2=ref(f1,-1);
```

Функция cross показывает пересечение двух кривых. Записывается она в следующем виде:

```
cross(mov1,mov2);
```

Если линия mov1 пересекает линию mov2 снизу вверх, то функция cross будет равна логической единице, во всех остальных случаях она равна логическому нулю. По сути, функция cross сводится к следующему выражению:

```
(ref(mov1,-1) < ref(mov2,-1)) and (mov1 > mov2);
```

Первая кривая mov1 на предыдущем баре была ниже второй кривой mov2, а на следующем (текущем) баре она стала выше. Это как раз и означает, что первая линия пересекла вторую снизу вверх. Давайте посмотрим как будет выглядеть эта функция на графике. Для этого составим небольшой текст программы индикатора, который мы назовем My_Cross (Мое пересечение). Вот текст этой программы:

```
mov1=sum(C,20)/20;
mov2=sum(C,40)/40;
cross1=cross(mov1,mov2);
cross2=cross(mov2,mov1);
cross=cross1-cross2;
cross;
```

В первых двух строках этой программы вычисляются две скользящие средние mov1 и mov2. Первая с периодом 20, а вторая с периодом 40. В следующих двух строках вычисляются пересечения этих линий. Переменная cross1 будет принимать значение логической единицы, когда mov1 пересекает mov2 снизу вверх (это стандартный сигнал к покупке). Переменная cross2 будет принимать значение логической единицы, когда mov2 пересечет mov1 снизу вверх или же (что тоже самое) mov1 пересечет mov2 сверху вниз. Это считается стандартным сигналом к продаже. В следующей строке вычисляется общая переменная пересечения cross, которая построена следующим образом: когда возникает сигнал на покупку, она равна плюс единице; когда возникает сигнал на продажу, она равна минус единице; во всех остальных случаях эта переменная равна нулю. Добиться этого очень просто: достаточно из переменной cross1 вычесть переменную cross2. И, наконец, в последней строке вычисленная переменная cross выводится на экран.

Давайте введем текст этого индикатора, который мы назвали My_Cross, и построим

его в отдельной графической области: там, где сейчас находится индикатор My_Mom, предварительно удалив, разумеется, последний. Если Вы все сделали правильно, то должно получиться примерно то, что изображено на рис. 3.10. Обратите внимание, что здесь для наглядности в ценовой графической области нанесены графики двух простых скользящих средних с периодами 20 и 40, а индикатор My_PC и уровень 1.3200 удалены.

Как мы видим, этот индикатор действительно показывает сигналы на покупку и продажу при соответствующем пресечении скользящих средних с периодами 20 и 40. Кстати, как видно из рис 3.10, последний сигнал на покупку оказался очень прибыльным – цена резко пошла вверх.

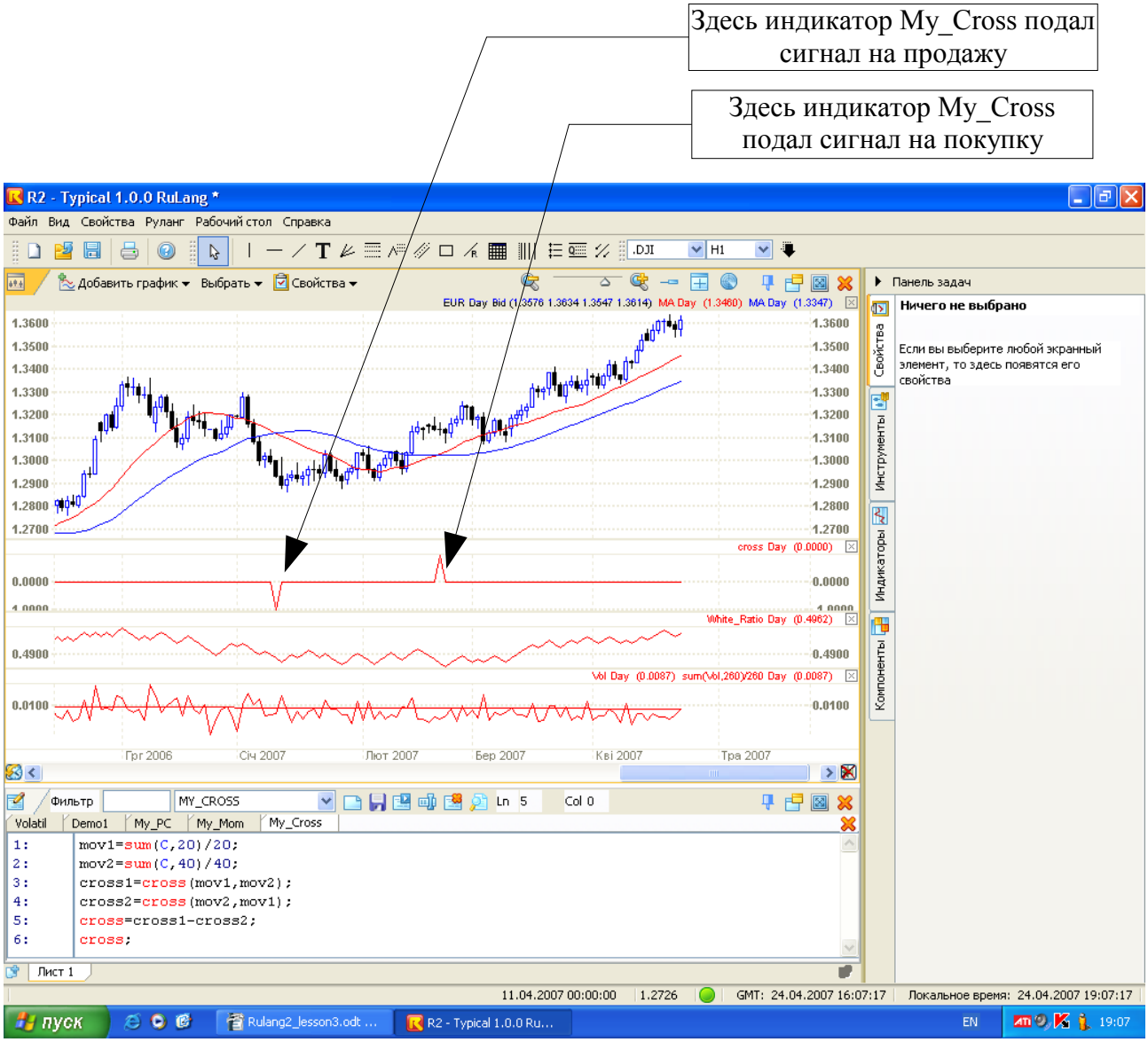

Рис. 3.10

Функции даты/времени

Эти функции возвращают параметры даты/времени текущего бара. Всего таких функций шесть:

- функция Year() возвращает номер года текущего бара;
- функция Month() возвращает номер месяца текущего бара (число от 1 до 12);
- функция DayOfMonth() возвращает номер дня текущего бара (число от 1 до 31);
- функция DayOfWeek() возвращает день недели (1 – понедельник, ..., 7- воскресенье);
- функция Hour() возвращает номер часа текущего бара (число от 0 до 23);
- функция Minute() возвращает номер минуты текущего часа (число от 0 до 59).

Например, если мы рассматриваем бар, у которого дата/время равны [12.04.2007 15:36], то тогда для него:

- функция Year() вернет значение 2007;
- функция Month() вернет значение 4 (апрель);
- функция DayOfMonth() вернет значение 12;
- функция DayOfWeek() вернет значение 4 (четверг);
- функция Hour() вернет значение 15;
- и, наконец, функция Minute() вернет значение 36.

Обратите внимание, эти функции не имеют аргументов, т.е в скобках ничего нет. При написании программы эти функции необходимо писать с пустыми скобками.

Функции индикаторов

Эти функции позволяют очень быстро вычислять значения тех или иных индикаторов и использовать их в дальнейших вычислениях в тексте программы Вашего индикатора. Например, если помните, мы в индикаторе Volatil вычисляли простую скользящую среднюю по формуле

```
sum(Vol,20)/20;
```

Интересно, а если бы нам нужно было вычислить не простую скользящую среднюю, а, скажем, экспоненциальную или взвешенную? Наверное, нам бы пришлось написать много формул, чтобы их вычислить. Да, это действительно можно сделать, но это было бы очень громоздко и трудоемко. К счастью, в РУЛАНГе имеется функция mov, с помощью которой можно очень быстро и просто вычислить значение скользящей средней любого типа. Например, ту формулу, о которой мы только что говорили, можно было бы записать по-другому с использованием функции mov следующим образом:

```
mov(Vol,20,s);
```

В качестве первого аргумента функции mov указывается та величина или выражение, от которого и будет вычисляться скользящая средняя. Это могут быть как переменные языка РУЛАНГ (O, H, L, C, V), так и переменные пользователя и, вообще, любые математические выражения.

Второй аргумент – это период, за который вычисляется скользящая средняя.

Третий аргумент указывает тип скользящей средней. Это текст из одного символа:

- s (простая);
- e (экспоненциальная);
- w (взвешенная);
- v (адаптивная).

Воспользовавшись такой возможностью, давайте в наших индикаторах Volatil и My_Cross исправим текст программы, используя обращение к функции mov.

В таком случае текст программы индикатора My_cross будет следующим:

```
mov1=mov(C,20,s);
mov2=mov(C,40,s);
cross1=cross(mov1,mov2);
cross2=cross(mov2,mov1);
cross=cross1-cross2;
cross;
```

А вот исправленный текст программы индикатора Volatil:

```
Vol=H-L;
Vol;
mov(Vol,260,s);
```

А какие еще функции индикаторов имеются в РУЛАНГе? Давайте их перечислим:

- mov (скользящие средние);
- bollinger1 и bollinger2 (линии коридора Боллинжера);
- momentum (индикатор моментум);
- rsi (индикатор RSI);
- stochastic_fast и stochastic_slow (стохастический осциллятор);
- CCI (индекс торгового канала);
- Zig (индикатор "Zig Zag")
- MACD_fast и MACD_slow (индикатор MACD)

Давайте рассмотрим их чуть подробнее.

Функция bollinger_1 вычисляет верхнюю границу канала, получаемого с помощью индикатора Bollinger Bands . Соответственно, функция **bollinger_2** вычисляет линию нижней границы канала. Например:

```
bollinger _1(close , 20, s);
```

bollinger _1(close , 20, s);

Как видите, аргументы этой функции точно такие же, как и у функции mov:

- первый аргумент – это переменная или выражение, от которой вычисляется скользящая средняя, на базе которой строится коридор Боллинжера;
- второй аргумент – это период усреднения;
- третий аргумент – это тип скользящей средней (s , e , w или v – такой же как для функции mov).

Следует так же обратить внимание, что по умолчанию значение «ширины канала» в данной реализации будет константой, равной 2,5. Изменять данный параметр, не вошедший в данную функцию как еще один аргумент, возможно, используя свойства стандартного индикатора Bollinger, который доступен в РУМУС2.

Функция momentum. Это общеизвестный индикатор, и хотя его очень просто вычислить с помощью функции ref, тем не менее имеется возможность воспользоваться для его вычисления стандартной встроенной функцией.

Пример использования:

momentum(C,20);

Первый аргумент функции – это переменная или выражение, от которой вычисляется моментум. Второй аргумент – период усреднения.

Функция rsi. Функция rsi вычисляет индикатор Relative Strength Index. Пример использования:

rsi(C,20);

В качестве параметров следует указать, по каким значениям вычисляется индикатор, а также его период усреднения. Такие параметры RSI как "Период усреднения RSI" и метод усреднения во встроенной функции rsi заданы по умолчанию и не меняются. Период усреднения RSI всегда равен 1 и метод усреднения RSI – simple.

Функции stochastic_fast и stochastic_slow вычисляют две линии стохастического осциллятора: соответственно медленный %K и медленный %D. Пример использования :

stochastic_fast(5,3);
stochastic_slow(5,3,3);

В функции stochastic _ fast используются только два параметра – период %K и период замедления %K . Для функции stochastic _ slow требуется еще третий параметр – период %D. Метод вычисления для встроенных функций выбрать невозможно, по умолчанию используется метод simple.

Функция cci вычисляет индикатор Commodity Channel Index. Пример использования:

cci (5);

В качестве параметра (аргумента) этого индикатора указывается период усреднения.

Функция Zig

Данная рассчитывает значения индикатора "Zig Zag".

Zig(DATA ARRAY, MINIMUM CHANGE, DIFF_METHOD);

Разберем аргументы данной функции. Аргумент DATA ARRAY - указывает на массив данных, для которых производятся вычисления. MINIMUM CHANGE – аргумент, содержащий значение минимальных изменений. DIFF_METHOD – это аргумент указывающий на метод, который используется при расчете значений индикатора "Zig Zag". Согласно теории Вы можете выбрать для расчета один из двух методов: процентный (PERCENT) и абсолютных значений (POINT).
Пример использования:

Zig(Close, 0. 5, percent);
Zig(Close, 0.0100, point);

Функции MACD_fast и MACD_slow вычисляют соответственно основную и сигнальную линии индикатора MACD. Пример использования:

macd_fast(12,26);
macd_slow(12,26,9);

Для вычисления функция MACD_fast в качестве аргументов следует указать короткий и длинный периоды скользящих средних. В качестве параметров (аргументов) для функции MACD_slow помимо короткого и длинного периода указывается также период усреднения для сигнальной линии. Метод вычисления для встроенной функции выбрать невозможно. По умолчанию сигнальная линия вычисляется как экспоненциальная.

Урок 4

Параметры индикатора
и украшение текста программы

Параметры индикатора. Функция inparam

Давайте посмотрим на текст индикатора My_Cross (см. рис. 4.1). Все вроде бы ничего. И индикатор хороший, и точки пересечения скользящих средних показывает. Вот только у него есть один существенный недостаток: он работает только с фиксированными параметрами скользящих средних (20 и 40). А если нам надо изменить эти параметры? Можно, конечно же, исправить их в прямо в тексте программы, но это не очень правильно. Дело в том, что тогда придется постоянно держать в голове и помнить то место, где надо внести соответствующие изменения, чтобы поменять эти параметры. Да и вообще, согласитесь, что это не очень удобная программа, в которой нужно постоянно менять ее текст только для того, чтобы изменить ее некоторые параметры.

А есть ли другой способ изменить параметры программы (индикатора), не изменяя при этом ее текст? Да, есть. И это делается с помощью функции inparam.

Рис. 4.1

Эта функция имеет следующий вид (пример):

P1 = inparam(«Period1»,1,1000,20);

Если в тексте программы будет присутствовать эта функция, то тогда в окне свойств нашего индикатора появится параметр, который называется Period1, и который можно будет изменять, проставляя в поле ввода нужное значение этого параметра (см. рис. 4.2).

В качестве первого аргумента этой функции указывается текст, заключенный в кавычки. Это имя параметра, которое потом пользователь увидит в окне свойств нашего индикатора. Второй и третий аргументы указывают минимальное и максимальное значение параметра, которое может ввести пользователь. В качестве четвертого аргумента указывается значение нашего параметра по умолчанию.

В данном случае, например, пользователь не может вести число, которое будет больше 1000. Если он все-таки это сделает, то параметру все равно будет присвоено только максимально возможное значение (1000).

Наверное Вы уже догадались, как нужно изменить текст программы, чтобы наш индикатор My_Cross имел два параметра. Давайте назовем параметры нашего индикатора Period1 и Period2. Тогда текст программы будет выглядеть таким образом:

```
P1=inparam(«Period1»,1,1000,20);
P2=inparam(«Period2»,1,1000,40);
mov1=mov(C,P1,s);
mov2=mov(C,P2,s);
cross1=cross(mov1,mov2);
cross2=cross(mov2,mov1);
cross=cross1-cross2;
cross;
```

В первых двух строках указываются параметры индикатора Period1 и Period2. По умолчанию они принимают значения 20 и 40 соответственно. Значения параметров, введенных пользователем (или значения по умолчанию) присваиваются переменным P1 и P2. А затем уже эти переменные используются в третьей и четвертой строках программы для указания параметров быстрой и медленной скользящих средних. Если Вы внесете эти изменения в текст программы индикатора и выполните его компиляцию, то тогда Вы увидите, что в свойствах, нашего индикатора появились два параметра Period1 и Period2. Причем они имеют значения, заданные для них по умолчанию (20 и 40).

Вот теперь можно спокойно менять эти параметры, задавая любые их значения (в пределах указанных ограничений, разумеется) и при этом ничего не надо менять в тексте программы.

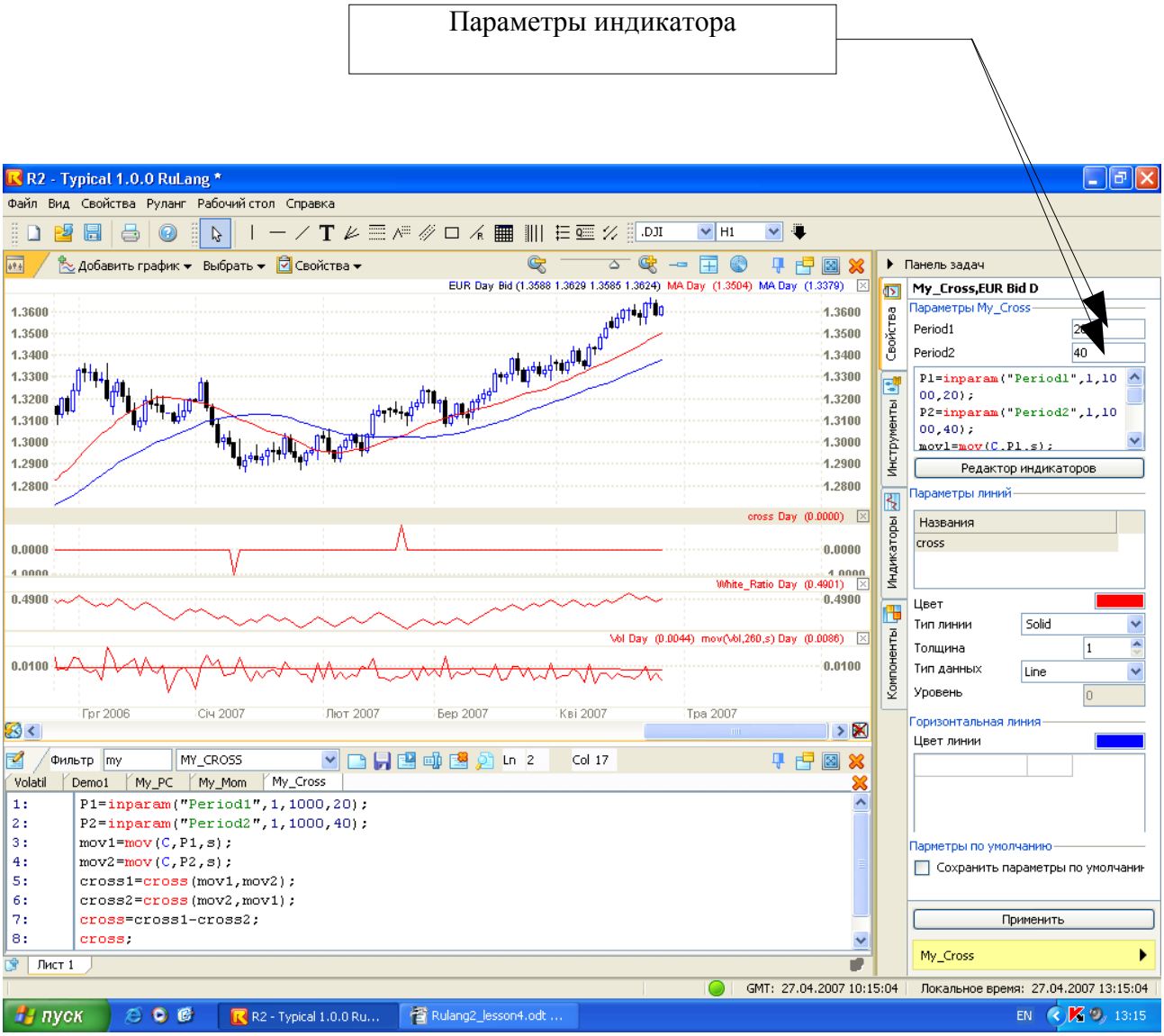

Рис. 4.2

Теперь аналогичные изменения можно внести в индикатор Volatil, задав ему единственный параметр Period. Его измененный текст будет выглядеть следующим образом:

```
P1=inparam(«Period»,1,1000,260);
Vol=H-L;
Vol;
mov(Vol,P1,s);
```

Как изменить тексты программ для двух индикаторов (My_Mom и My_PC) Вам, наверное, уже понятно. Теперь Вы это сможете сделать самостоятельно.

Свойства индикатора, созданного пользователем

Как Вы, конечно же, знаете – у каждого индикатора в РУМУС2 имеются некоторые свойства. Под свойствами понимаются такие вещи, как параметры индикатора, цвета и толщины его линий и т.д. У индикаторов, которые созданы Вами имеются аналогичные свойства. Давайте рассмотрим их несколько подробнее.

Для того, чтобы увидеть свойства Вашего индикатора (например, индикатора Volatil), его необходимо выбрать, щелкнув по нему кнопкой мыши. В результате его свойства тут же отобразятся на панели задач (см. рис. 4.3).

В верху заголовка свойств указывается имя этого индикатора и, далее, через запятую перечислены названия тех данных, по которым он был построен. Например: «Volatil,EUR Bid D» означает, что график Volatil построен по данным дневного графика евро.Запись «Volatil,EMA,EUR Bid D» означает, что индикатор Volatil был построен по графику экспоненциальной скользящей средней, которая в свою очередь была построена по данным дневного графика евро.

Ниже заголовка индикатора в разделе «Параметры» приведены:
- параметры индикатора;
- формула индикатора (текст программы в маленьком окне редактора индикаторов);
- кнопка вызова редактора индикаторов.

Вносить изменения можно прямо в приведенной формуле. Если вам делать это в окне с формулой неудобно, то можно вызвать редактор индикатора. Это делается с помощью кнопки, расположенной под формулой.

Следующий раздел свойств, который называется «Параметры линий» состоит из двух частей:
- список линий, которые имеются у индикатора;
- свойства этих линий.

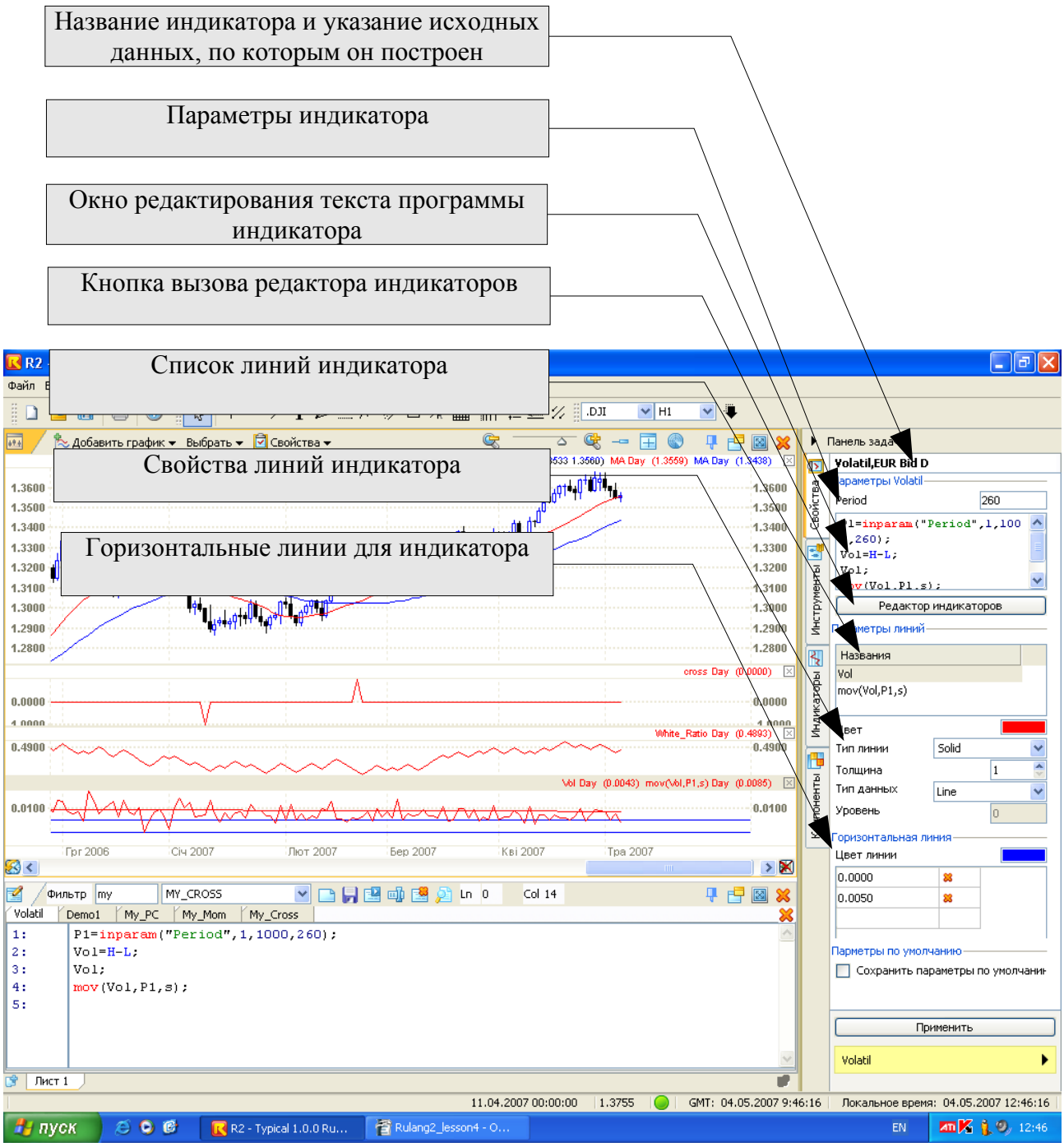

Рис. 4.3

Обратите внимание, что названия линий в этом списке точно такие же, как

упоминались в операторе вывода. Например, скользящая средняя от волатильности была выведена на индикацию с помощью следующего оператора вывода:

```
mov(Vol,P1,s);
```

В результате эта линия попала в список именно с таким именем (см. рис. 4.3). Ясно, что такое название линии индикатора не очень благозвучное и наглядное. Чтобы исправить это, нужно данную линию записать в тексте программы немного по-другому: сначала величину значения линии присвоить какой-нибудь переменной с более ясным и благозвучным названием (например, MA_Vol), а затем уже эту переменную вывести на экран с помощью оператора вывода. Тогда вместо той строки, которая показана выше в тексте программы индикатора, должно быть две строки:

```
MA_Vol=mov(Vol,P1,s);
MA_Vol;
```

Давайте внесем указанное изменение в текст программы индикатора Volatil и скомпилируем ее.

Для того, чтобы изменить свойства той или иной линии индикатора, необходимо сперва выбрать ее с помощью однократного щелчка мыши. Только тогда ее свойства (т.е. свойства именно этой линии!) появятся ниже списка линий, и только тогда их можно изменять. Для того, чтобы выполненные изменения вступили в силу необходимо нажать на кнопку «Применить».

Линия индикатора имеет следующие свойства:

- цвет линии;
- тип линии (невидимая, сплошная, штриховая, пунктирная, штрих-пунктирная);
- толщина линии;
- тип данных (линия или гистограмма).

Назначение и смысл всех этих свойств линий индикатора вполне очевидны и понятны.

В самом конце имеется раздел свойств «Горизонтальная линия». На любом индикаторе можно провести одну или несколько горизонтальных линий, и их свойства указываются в этом разделе. На рис. 4.3 на индикаторе волатильности проведены две линии: на уровне ноль и на уровне 50 пунктов. Цвет линий можно изменять. Этот цвет один и тот же на все горизонтальные линии. Список линий также можно изменять и добавлять по своему усмотрению. Чтобы удалить ненужную линию, следует нажать на крестик, который расположен в списке горизонтальных линий в правой колонке.

Оформление текста программы и комментарии в нем

Посмотрев на тексты программ, которые приведены выше, может создаться впечатление, что в одной строке должен быть только один оператор. На самом деле это не так. В одной строке может быть, например, один оператор, и, наоборот, один оператор может быть разделен на несколько строк.

Дело в том, что компьютер (вернее компилятор) при чтении и компиляции текста программы ориентируется только на разделители операторов (точка с запятой), или же на разделители слов (пробелы или переход на новую строку). Кстати, переход на следующую строку – это специальный символ в тексте программы, и он невидим на экране компьютера. Или вернее сказать так: он виден на экране в том смысле, что после него текст продолжается с новой строки. Так вот, и пробел, и переход на новую строку это всего лишь разделители слов языка РУЛАНГ. Причем идущие много раз подряд разделители (несколько пробелов подряд и/или несколько пустых строк подряд) все равно интерпретируются языком РУЛАНГ как один разделитель. Этим можно воспользоваться. Например, с помощью вставки в нужных местах определенного количества пробелов можно в оформить программу таким образом, чтобы текст в ней был выровнен определенным образом, сделаны отступы и т.п. Можно еще сделать выделения в тексте путем создания одной или нескольких пустых строк.
б) ожидание, пока индикатор выйдет из перекупленности, и потом покупка;
в) ожидание пока индикатор выйдет из перекупленности, и потом продажа. Можно можно оформить текст программы по своему усмотрению таким образом, чтобы он был наиболее удобен и понятен для чтения и восприятия человеком.

Разумеется, при этом надо соблюдать простые и вполне понятные правила. Нельзя, например, внутри названий функций или имен переменных ставить пробелы. Если это произойдет, то тогда имя функции будет рассматриваться компьютером как два раздельных слова, и, если они ему будут незнакомы, это может привести к сообщению об ошибке.

Повысить читабельность и понятность программы можно не только с помощью оформления в виде отступов, но и также использовать комментарии. Комментарий - это текст, который предназначен для человека, а компьютер его не воспринимает, а попросту игнорирует. Существует два вида комментариев: однострочный и многострочный.

Однострочный комментарий в тексте программы выглядит таким образом

```
// Это комментарий. Здесь можно писать все, что угодно.
```

Однострочный комментарий начинается с двух наклонных черточек (//). Между ними не должно быть пробелов. Как только компьютеру встретится такое сочетание знаков, он будет игнорировать весь текст, который идет после них, до самого конца строки. Начиная с новой строки компьютер опять будет пытаться прочесть текст программы как обычно.

Кстати, перед комментарием (в этой же строке) может располагаться текст программы (операторы). Это может, например, выглядеть так:

```
mov1 = mov(C,P1,w);  //  вычисляем взвешенную скользящую среднюю
```

В этой строке сначала идет текст программы, в котором выполняется вычисление скользящей средней, а затем идет комментарий, который будет восприниматься только человеком, а компьютер работать с ним не будет.

А если комментарий очень большой и длинный, и в одну строку явно не влазит? Тогда есть специальный многострочный комментарий. Он, например, может выглядеть следующим образом:

```
/* Это очень длинный-предлинный
комментарий, который тянется
```

на много строк */

Этот комментарий заключается в специальные комментаторские скобки. Открывающая скобка – это наклонная черта со звездочкой (/*), а закрывающая скобка – это звездочка с наклонной чертой (*/). Все, что находится между этими скобками, предназначено для глаз человека, а компьютер весь этот текст игнорирует. Этот тип комментария можно использовать для более объемных пояснений в тексте программы.

Оформление текста программы с использованием комментариев

Обычно текст программы, если он оформляется по всем правилам, должен содержать не только комментарии по отдельным операторам, но и пояснительный и служебный текст, относящийся ко всей программе в целом. В частности, совершенно не лишним было бы указать имя (название) индикатора, описание и назначение параметров индикатора, принцип его работы и т.п. Кроме того, можно (по желанию) также указать дополнительную информацию. Например, имя автора индикатора, его номер телефона, адрес электронной почты, знак авторского права и другие данные.

Давайте в качестве примера рассмотрим индикатор My_Cross, и дадим в нем подробные комментарии. Вот примерный текст вместе с комментариями.

```
/* My_Cross */

//------------------------//
//                        //
//  Пересечение средних   //
//                        //
//------------------------//

//--------------------------------------//
//                                      //
// Copyright © 2007, Petr P. Petrov     //
// All rights reserved                  //
// Organization: ForexClub              //
// Phone: +38-057-0000000               //
// http://www.fxclub.org                //
//                                      //
//--------------------------------------//

//-----------//
// Параметры //
//-----------//

P1=inparam("Period1",1,1000,20); // период быстрой скользящей средней
P2=inparam("Period2",1,1000,40); // период медленной скользящей средней

//--------------------//
// Скользящие средние //
//--------------------//

mov1=mov(C,P1,s); // быстрая СС
mov2=mov(C,P2,s); // медленная СС
```

```
//----------------------//
// Пересечения средних //
//----------------------//

cross1=cross(mov1,mov2); // быстрая СС пересекает медленную снизу вверх
cross2=cross(mov2,mov1); // медленная СС пересекает быструю сверху вниз
my_cross=cross1-cross2;

//-----------//
// Индикация //
//-----------//

my_cross;
```

Отключение части программы с помощью комментария

Иногда возникает ситуация, когда какой-то оператор в программе становится не нужным, но удалить его жалко из-за того, что в будущем он может понадобиться, и тогда его после удаления придется вводить заново. В таком случае следует временно «отключить» этот оператор. Это очень просто: нужно сделать так, чтобы этот оператор стал комментарием. У программистов, кстати, есть даже такой термин «закомментировать оператор». Например, строка

```
//mov1 = mov(C,P1,w);  //  вычисляем взвешенную скользящую среднюю
```

вычисляться не будет, но стоит только убрать знак комментария (//), который стоит в самом начале строки перед переменной mov1, то этот оператор «включится», и будет вычисляться.

Если же нужно отключить, скажем, сразу 20-ть строк программы, то тогда нет смысла ставить 20-ть знаков комментария (//) перед каждой из этих строк. Можно просто все эти строки заключить в скобки многострочного комментария (/* ... */). А чтобы опять их «включить» достаточно только убрать эти комментаторские скобки.

Урок 5

Создание собственных функций,
условный оператор и
оператор цикла

Собственные функции пользователя

Иногда может оказаться так, что нам понадобится очень простая функция, например, функция округления числа до кратного заданной точности, или нахождение наибольшего делителя. А этих функций в РУЛАНГЕ нет. Что тогда делать? Очень просто: язык РУЛАНГ позволяет создавать свои собственные функции. Это самые настоящие функции, в которые можно передавать аргументы. Функции, выполнив над этими аргументами определенные операции, возвращают вычисленное ими значение в программу. А во всех остальных отношениях функция – это самая обычная программа, написанная на языке РУЛАНГ.

Программу–функцию от обычной программы отличает только то, что для неё необходимо передать аргументы и получить от нее некоторое вычисленное значение. Следовательно, в языке РУЛАНГ должно быть два дополнительных оператора. Первый описывает аргументы, передаваемые функции, он называется inputs. А второй возвращает вычисленное значение, и он называется return.

Рассмотрим процесс создания собственной функции на примере функции ABS (вычисления абсолютной величины).

В языке РУЛАНГ для вычисления модуля числа существует встроенная функция Abs. Данная функция написана разработчиками языка РУЛАНГ и ее использование возвращает абсолютное значение числа (модуль числа). Синтаксис прост:

```
Abs(A);
```

Это может нам пригодиться, например, для получения размера тела свечи:

```
Abs(open - close);  // в результате получаем размер тела свечи.
```

Этот же результат можно получить и при помощи самостоятельно написанной функции. Текст программы функции ABS может быть следующим:

```
Inputs: X;
Y = sqrt(pow(X,2));
return Y;
```

В первой строке объявляются входные аргументы функции. В данном случае у нас только один аргумент: переменная X. Если бы нам нужно было передать несколько аргументов, то тогда их следовало перечислить через запятую.

Во второй строчке производятся необходимые вычисления: число X возводится в квадрат, а затем из него извлекается квадратный корень. И то, что в результате получилось, присваивается переменной Y. В результате таких манипуляций величина числа Y будет точно такой же, как и числа X, но только у числа Y теперь всегда будет положительный знак независимо от того, какой знак был у числа X.

И, наконец, в последней, третьей строке оператор return возвращает вычисленное значение в программу, которая обращалась к этой функции.

Итак, функция ABS у нас уже есть. А как на практике ей пользоваться? Для примера рассмотрим индикатор, который использует эту функцию. Пусть этот индикатор (назовем его Body) будет вычислять размер тела свечи (по модулю) и его среднее значение за заданный период.

Кстати, эта функция очень похожа на нашу функцию Volatil, но только она будет вычислять волатильность не всей свечи, а только ее тела. Итак текст программы Body выглядит примерно так:

```
P1=inparam("Period",1,1000,260);
Body=C-O;
Abs_Body=ABS(Body);
Abs_Body;
MA_Abs_Body=mov(Abs_Body,P1,s);
MA_Abs_Body;
```

Первая строка этой программы вполне очевидна: здесь задается параметр усреднения, который в дальнейшем может изменяться пользователем. Во второй строке вычисляется размер тела свечи (Body) как разница цен Close и Open. В следующей строке вычисляется абсолютная величина размера тела свечи (Abs_Body) с использованием той самой функции ABS, которую мы должны заранее написать и создать. Как видите, мы передаем этой функции величину Body, которая может оказаться как положительной, так и отрицательной. Но переменная Abs_Body уже всегда будет положительной.

Назначение оставшихся трех строк вполне очевидно. Здесь значение Abs_Body выводится на экран, затем это значение усредняется с помощью простой скользящей средней, и также выводится на экран.

Давайте построим полученный индикатор на экране. Строить его будем в дополнительной графической области там, где сейчас находится индикатор Demo1. Сам индикатор Demo1 мы, разумеется, удалим. Если все сделано правильно, то тогда на экране должно быть примерно то, что показано на рис. 5.1.

Как видно из этого рисунка, среднегодовая волатильность тела свечи (MA_Abs_Body) на правом краю краю графика составляет примерно 42 пункта, в то время как волатильность всей свечи (MA_Vol) примерно вдвое больше: 85 пунктов.

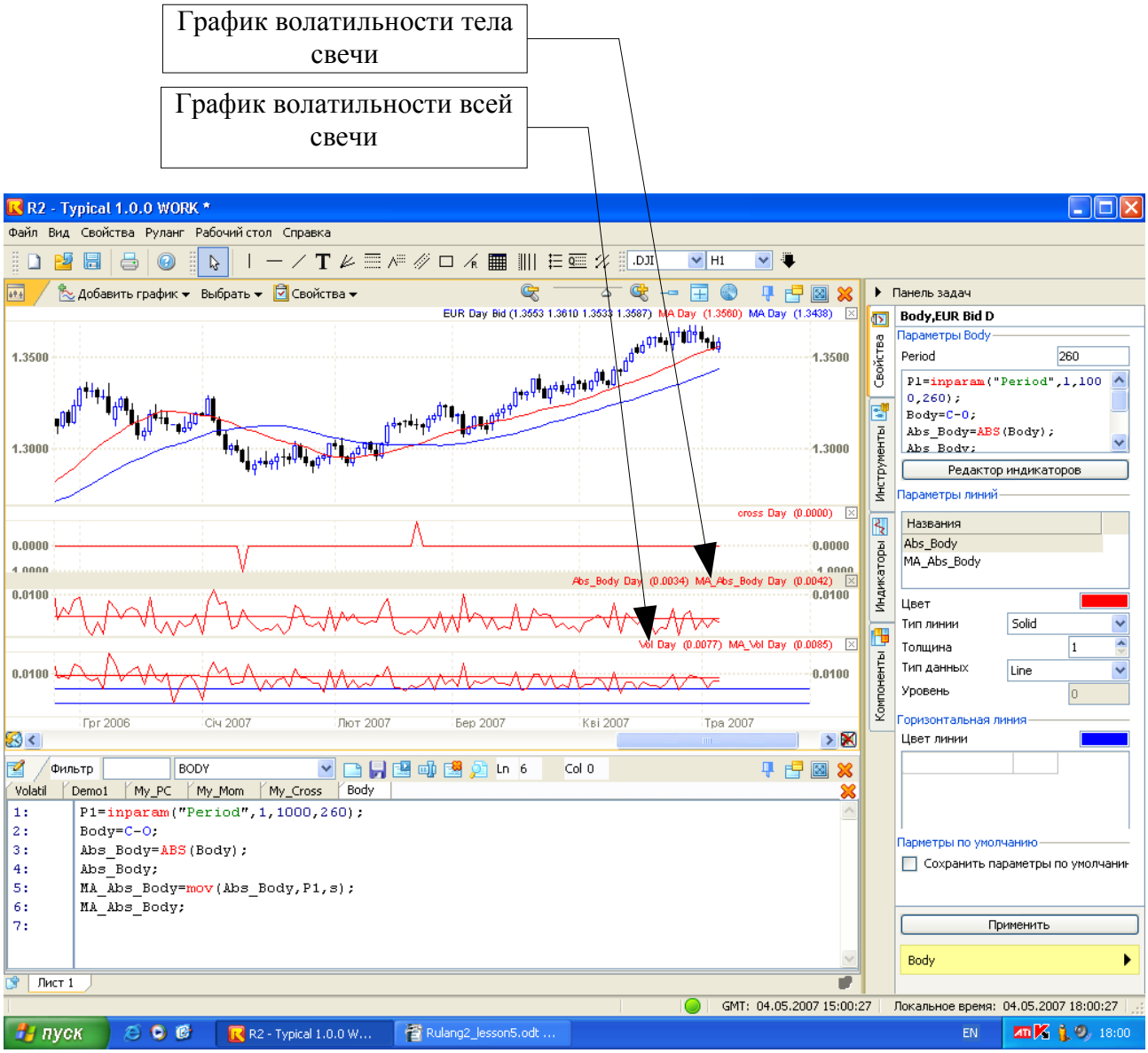

Рис. 5.1

Вложение функций. Условный оператор

Функции могут быть вложены друг в друга. Это, в частности, означает, что наш индикатор (функция) Body может быть вызван из другого индикатора. В свою очередь индикатор (функция) Body может вызывать другие индикаторы и функции. Мы знаем, что она вызывает функцию ABS. Разумеется, что для того, чтобы индикатор вызывался из другого индикатора, он должен быть оформлен как функция. Т.е. у него обязательно должны должны быть операторы inputs и return. При этом попутно следует решить еще пару проблем. Во-первых, хотелось бы чтобы наш индикатор, который будет использоваться в качестве

функции, можно было бы использовать и по прямому назначению, т.е. построить его на экране. Во-вторых, у нашего индикатора-функции может быть несколько линий (например, у индикатора Body имеется две линии Abs_Body и MA_Abs_Body). Так вот, хотелось бы, чтобы внешний индикатор мог бы воспользоваться любой из этих линий.

Чтобы удовлетворить всем этим требованиям, индикатор Body можно было бы модифицировать следующим образом.

```
1:   Inputs: Out(0), P1(260);
2:   // Out=0 ----> режим индикатора, выводит на экран Abs_Body и MA_Abs_Body
3:   // Out=1 ----> режим функции, которая возвращает Abs_Body
4:   // Out=2 ----> режим функции, которая возвращает MA_Abs_Body
5:   if Out=0 then P1=inparam("Period",1,1000,260);
6:   Body=C-O;
7:   Abs_Body=ABS(Body);
8:   Abs_Body;
9:   MA_Abs_Body=mov(Abs_Body,P1,s);
10:  MA_Abs_Body;
11:  if Out=1 then answer=Abs_Body;
12:  if Out=2 then answer=MA_Abs_Body;
13:  return answer;
```

В первой строке у нашей функции объявляются два аргумента: Out и P1. Что такое параметр P1 – это, наверное, вполне очевидно и понятно. Это период усреднения. Кстати, обратите внимание, что у аргументов можно определять значения по умолчанию. Они записываются в скобках. Если функция будет вызвана без аргументов, то тогда в качестве аргументов будут использоваться эти самые значения по умолчанию. В частности, параметр P1 по умолчанию является равным 260-ти.

А вот зачем нам понадобился еще один аргумент, который называется Out? Дело в том, что этот аргумент будет играть двойную роль. Во-первых, он будет определять режим работы нашей программы: когда Out=0, то программа работает в режиме обычного индикатора, т.е. выводит вычисленные линии индикатора на экран. Когда Out не равно нулю, тогда программа работает в режиме функции, и возвращает в вышестоящую программу некоторое значение.

Во-вторых, в режиме функции переменная Out определяет, какую именно величину необходимо вернуть в вышестоящую программу: если Out=1, то тогда возвращается величина самой волатильности (переменная Abs_Body), а если Out=2, то тогда возвращается средняя величина волатильности (MA_Abs_Body).

В строках 2, 3 и 4 в комментариях как раз описаны все эти режимы работы.

В 5-й строке использован так называемый условный оператор. Чуть позже мы о нем поговорим подробнее. А сейчас разберем, что он делает. В переводе на русский язык он звучит примерно так:

«Если (if) переменная Out=0 (т.е. режим работы у нас: вывод индикатора на экран), то (then) переменная P1 задается пользователем через оператор inparam».

Наверное, смысл этой фразы вполне очевиден: если программа работает в режиме вывода на экран, то тогда значение параметра усреднения будет задавать пользователь. А если программа работает в режиме функции (т.е. Out не равен нулю), то тогда оператор, который стоит после слова then, выполняться не будет. Следовательно, переменная P1 не может быть изменена пользователем, т.е. она останется такой, какая она была передана в

нашу функцию через оператор Inputs в первой строке программы.

Далее, строки с 6-й по 10-ю вполне понятны, и они не изменились по сравнению с предыдущей версией этой программы. А именно, здесь производится вычисление волатильности тела свечи (переменная Abs_Body) и ее скользящей средней (MA_Abs_Body), а также выполняется вывод на экран этих значений.

В строках 11 и 12 готовится ответ (переменная answer) для возврата его в вышестоящую программу. Здесь также используется все тот же условный оператор. Строка 11 звучит примерно так:

«Если (if) переменная Out=1 (т.е. вышестоящая программа запрашивала размер волатильности свечи), то (then) тогда нашим ответом (переменная answer) будет значение переменной Abs_Body». А строка 12 звучит примерно так:

«Если (if) переменная Out=2 (т.е. вышестоящая программа запрашивала усредненную величину волатильности), то (then) тогда нашим ответом (переменная answer) будет значение переменной MA_Abs_Body».

Ну и, наконец, в строке 13-й вычисленный таким образом наш ответ будет возвращен в вышестоящую программу с помощью оператора return.

Если мы соответствующим образом изменим текст программы Body и скомпилируем его, то увидим, что на экране этот индикатор останется абсолютно таким же, каким он показан на рис. 5.1. В принципе, так и должно быть. Ведь он сейчас работает в режиме индикатора, поскольку его переменная Out по умолчанию равна нулю.

Если же какая-нибудь вышестоящая программа будет обращаться к нашему индикатору Body, то это обращение могло бы выглядеть, например, таким образом:

```
Body1=Body(2,260);
```

В данном случае вышестоящая программа запрашивает усредненное значение волатильности (т.к. первый аргумент равен 2), причем период усреднения равен 260-ти. И затем полученное таким образом значение присваивает переменной Body1. Кстати, в данном случае к нашей функции можно было бы обратиться в более краткой форме:

```
Body1=Body(2);
```

Здесь второй аргумент опущен, следовательно он считается равным величине по умолчанию, т.е. 260-ти.

А теперь давайте несколько подробнее рассмотрим условный оператор.

Условный оператор if ... then ... else

В полном виде условный оператор выглядит так:

if <УСЛОВИЕ> then <ОПЕРАТОР1>; else <ОПЕРАТОР2>;

Выполняется он компьютером следующим образом. Сначала вычисляется УСЛОВИЕ

(это логическое выражение), которое стоит после слова if. Если это УСЛОВИЕ выполняется (т.е. оно равно логической единице), то тогда выполняется ОПЕРАТОР1, который стоит после слова then, а ОПЕРАТОР2, который стоит после слова else, не выполняется. В противоположном случае (когда УСЛОВИЕ не выполняется, т.е. оно равно логическому нулю) ОПЕРАТОР1 не будет выполняться, а ОПЕРАТОР2, наоборот, будет выполнен. В качестве примера можно рассмотреть вычисление модуля (абсолютной величины) переменной X с помощью условного оператора:

```
if X>0 then Y=X; else Y=-X;
```

Если перевести эту фразу на обычный язык, то она будет звучать примерно так. «Если (if) число X положительное (X>0), то (then) тогда число Y будет равно числу X без изменения знака. В противном случае (else), т.е. когда число X отрицательное, тогда его знак меняется на противоположный, и его значение присваивается числу Y». Таким образом число Y и будет являться модулем числа X, поскольку оно всегда будет положительным не зависимо от того, какой знак будет у числа X.

Оператор цикла while

Этот оператор позволяет выполнять одни и те же действия в программе («в цикле») до тех тех пор, пока условие цикла не будет выполнено.

Этот оператор в тексте программы в минимальной форме выглядит, например, вот так:

```
while (<УСЛОВИЕ_1>) <ОПЕРАТОР_1>;
```

В начале оператора пишется слово while (пока), вслед за которым в скобках указывается некоторое логическое выражение <УСЛОВИЕ_1>. Если это условие выполняется (т.е. его значение равно логической единице), то тогда выполняется один-единственный <ОПЕРАТОР_1>, который стоит после <УСЛОВИЯ_1>. В результате выполнения этого оператора <УСЛОВИЕ_1> может измениться. Если оно при этом все еще выполняется (т.е. равно логической единице), то тогда <ОПЕРАТОР_1> будет выполнен повторно. Если же это условие не выполняется (т.е. оказалось равно логическому нулю), то тогда работа оператора цикла на этом завершается, и далее будет выполняться последующий оператор программы, который стоит после <ОПЕРАТОРА_1>. Т.е. то же самое можно сказать и другими словами: <ОПЕРАТОР_1> будет выполняться до тех пор, пока (while) <УСЛОВИЕ_1> будет оставаться истинным. Как только <УСЛОВИЕ_1> станет ложным оператор цикла завершит свою работу. Каждое исполнение <ОПЕРАТОРА_1> в операторе цикла называется итерацией. Число итераций оператора цикла может быть самым разным: одна, две, сто или даже ни одной итерации. Последний случай будет иметь место, если <УСЛОВИЕ_1> уже было ложным еще до выполнения первой итерации оператора цикла.

Рассмотрим пример.

```
i=1;
```

```
while (i<=20) i=i+1;
```

Здесь в первой строке переменной i присваивается значение единица. Затем во второй строке оператор цикла в каждой итерации увеличивает значение этой переменной на единицу. После 20-й итерации переменная i станет равной 21. Тогда условие (i<=20) перестанет выполняться, и на этом выполнение оператора цикла прекратится. Заметьте, кстати, что если бы в первой строке программы переменной i было бы присвоено значение, большее 20-ти, то тогда оператор цикла не был бы выполнен ни разу, поскольку его условие (i<=20) не выполнялось бы еще до начала его первой итерации.

Может создаться впечатление, что от оператора цикла мало пользы, поскольку он может выполнять только простейшие действия, задаваемые одним-единственным оператором. Получается, что более сложные действия, состоящие из нескольких операторов, которые нужно повторять в цикле, ему недоступны. На самом деле это не так.

Да, действительно, у оператора цикла имеется только один исполняемый оператор. Но все дело в том, что это может быть какой угодно оператор, в том числе и составной. Напомним, что составной оператор, это такой оператор, который состоит из нескольких операторов, заключенных в логические скобки (слова begin и end). Таким образом, с помощью составного оператора оператор цикла может выполнять в цикле сколь угодно сложные действия, состоящие из многих обычных операторов.

При использовании составного оператора оператор цикла будет выглядеть примерно следующим образом:

```
while (<УСЛОВИЕ_1>) begin
        <ОПЕРАТОР_1>; <ОПЕРАТОР_2>; ... ; <ОПЕРАТОР_N>;
end;
```

Здесь если УСЛОВИЕ_1 выполняется, то тогда будут выполняться в цикле все операторы (от ОПЕРАТОР_1 до ОПЕРАТОР_N), которые входят в составной оператор. Как только УСЛОВИЕ_1 перестанет выполняться, оператор цикла завершит свою работу, и далее будут выполняться последующие операторы, которые идут в тексте программы после оператора цикла.

Рассмотрим пример. Мы его уже рассматривали в уроке 2, когда говорили о массивах. Давайте вспомним его. Допустим, что мы в массиве Cls будем хранить 20-ть последних цен Close. В первом элементе массива будет храниться самая свежая (предпоследняя) цена Close. Во втором элементе будет цена Close предыдущего бара или свечи, и так далее. В последнем, двадцатом элементе будет содержаться цена Close бара, который находится на 20-ть баров левее самого свежего (текущего) бара. В таком случае индикатор цикла, производящий «загрузку» цен Close в массив Cls, будет выглядеть примерно следующим образом.

```
i=0;
while (i<20) begin
        i=i+1; Cls[i]=ref(Close,-i);
end;
```

Выполнение этой программы происходит следующим образом. Сначала переменной i (она еще называется переменной-счетчиком, поскольку она подсчитывает номер текущей итерации в цикле) присваивается значение нуля. Далее начинает работу оператор цикла. Он проверяет условие (i<20), и поскольку оно является истинным, он выполняет составной оператор. Внутри этого составного оператора имеются два обычных оператора. Первый из них увеличивает счетчик итераций на единицу, теперь переменная i будет равна единице. Второй помещает в первый элемент массива Cls[1] значение цены Close предыдущего бара.

На этом первая итерация цикла заканчивается, и оператор цикла пытается выполнить вторую итерацию. Для этого он опять проверяет истинность условия (i<=20), и, поскольку оно выполняется (ведь переменная i сейчас, как Вы помните, равна 1), происходит повторное выполнение составного оператора. Однако на этот раз, поскольку переменная i увеличиться еще на единицу и будет равна двум, то тогда произойдет загрузка уже второго элемента массива Cls[2], а не первого, как это было в первой итерации. Причем в Cls[2] теперь попадет значение цены Close не предыдущего бара (как это было в первой итерации), а пред-предыдущего, т.е. того бара, который является вторым предыдущим относительно текущего бара.

После этого аналогичным образом будет выполнена третья итерация, затем – четвертая, пятая, и т.д. вплоть до 20-й итерации. В процессе 20-й итерации переменной i будет присвоено значение 20. Оператор цикла в очередной 21-й раз попытается выполнить следующую итерацию, однако на этот раз она не будет выполнена, поскольку условие цикла (i<20) теперь будет ложным. В результате работа оператора цикла на этом будет закончена, и в дальнейшем будет выполняться какой-то другой оператор, который в тексте программы стоит после нашего оператора цикла.

Досрочное прерывание цикла

Иногда возникает необходимость досрочно завершить текущую итерацию, а то и весь цикл в целом. В первом случае используется специальный оператор continue (прерывает текущую итерацию), во втором случае – оператор break (прерывает весь цикл). Обращаем Ваше внимание на то, что операторы continue и break имеют смысл только в операторе цикла. Если они встретятся где-нибудь в тексте программы вне оператора цикла, то это приведет к сообщению об ошибке. Итак, в случае использования операторов continue и break оператор цикла, например мог бы выглядеть примерно так.

```
while (<УСЛОВИЕ_1>) begin
        <ОПЕРАТОР_1>; <ОПЕРАТОР_2>; ... ; <ОПЕРАТОР_N>;
        if <УСЛОВИЕ_2> then continue;
        <ОПЕРАТОР_N+1>; <ОПЕРАТОР_N+2>; ... ; <ОПЕРАТОР_M>;
        if <УСЛОВИЕ_3> then break;
        <ОПЕРАТОР_M+1>; <ОПЕРАТОР_M+2>; ... ; <ОПЕРАТОР_P>;
end;
```

Заметьте, что до тех пор, пока УСЛОВИЕ_1 является истинным, операторы начиная с ОПЕРАТОР_1 и до ОПЕРАТОР_N выполняются всегда во всех итерациях цикла, независимо от того, выполняются ли УСЛОВИЕ_2 или УСЛОВИЕ_3 или нет.

Если оба условия УСЛОВИЕ_2 и УСЛОВИЕ_3 не выполняются, то тогда все остальные операторы (начиная с оператора ОПЕРАТОР_N+1 и заканчивая оператором

ОПЕРАТОР_P) будут выполняться в каждой итерации цикла.

Если УСЛОВИЕ_2 выполнится (т.е. окажется истинным), то тогда будет выполнен оператор continue. Это означает, что текущая итерация будет прервана, и операторы начиная ОПЕРАТОР_N+1 и заканчивая ОПЕРАТОР_P в ней выполняться не будут. А оператор цикла приступит к выполнению следующей итерации, начиная с ОПЕРАТОР_1.

Если же в процессе выполнения какой-либо итерации выполнится УСЛОВИЕ_3 (т.е. оно окажется истинным), то тогда выполнение оператора цикла прервется полностью. Т.е. операторы (начиная с ОПЕРАТОР_M+1 и заканчивая ОПЕРАТОР_P) выполняться не будут, новая итерация также выполняться не будет, а далее в действие вступит другой оператор, который по тексту программы находится после оператора цикла.

Например, допустим, что в том примере, который мы рассмотрели ранее, нам необходимо сделать так, чтобы в 11-й и 15-й элементы массива Cls цены Close не попадали. Этого можно добиться с помощью оператора continue, чтобы с его помощью прервать (а точнее – пропустить) итерации номер 11 и 15. Текст программы этого оператора цикла тогда мог бы выглядеть примерно таким образом.

```
1:      i=0;
2:      while (i<20) begin
3:              i=i+1;
4:              if i=11 or  i=15 then continue;
5:              Cls[i]=ref(Close,-i);
6:      end;
```

Работа этого оператора цикла происходит абсолютно аналогичным образом, как и в предыдущем примере. Только в итерациях 11 и 15 будет выполнено условие, которое приведено в строке 4 текста программы. В результате выполнения этого условия будет выполнен оператор continue, который прервет выполнение операций в строке 5 и вернется на начало цикла. Следовательно в итерациях 11 и 15 одноименные элементы в массиве Cls не будут заполнены соответствующими значениями цены Close.

Важное предупреждение при работе с операторами цикла

При работе с операторами циклов необходимо соблюдать определенную осторожность. Дело в том, что у этих операторов при неправильном обращении с ними может возникнуть ситуация, которая называется «зацикливанием». Рассмотрим конкретный пример. Допустим, у нас имеется текст следующей программы.

```
while (155 >= 70) begin
        X=Open; Y=Close;
end;
```

Эта программа – пример гарантированного «зацикливания». Дело в том, что условие цикла (155 >= 70) будет истинным всегда. Следовательно, оператор цикла, проверив его истинность, будет запускать одну итерацию за другой до бесконечности. Это и есть «зацикливание» - оператор цикла повторяется до бесконечности. Внешне на экране

компьютера это будет выглядеть как полное «замирание» программы: она пытается построить линию индикатора, но это ей никак не удается. Вывести программу из этого состояния «ступора» достаточно сложно, поэтому будьте очень внимательны, и не допускайте в тексте программы ошибок, которые могут привести к «зацикливанию».

Пример использования оператора цикла. Параллельные каналы

Параллельные каналы – это несколько скользящих средних, проведенных параллельно друг другу с некоторым шагом по вертикальной оси. Например, на рис. 5.2 показана система параллельных каналов, построенная из простых скользящих средних с периодом 65 и сдвинутых друг относительно друга с шагом 1% по вертикали. Построить их очень просто. Сначала строится центральная скользящая средняя (она на рис. 5.2 показана утолщенной линией). Ее период равен 65 и сдвиг по оси Y равен нулю. Затем параллельно ей чуть выше строится другая скользящая средняя. У нее период будет тот же самый (65), а сдвиг по оси игрек равен +1%. Следующая скользящая средняя проводится еще выше, т.е. ее сдвиг будет равен +2%. И так далее: у последующих линий сдвиг увеличивается на 1% по сравнению с предыдущими линиями.

Аналогично проводятся скользящие средние, которые расположены ниже центральной линии. Только теперь у них сдвиги будут отрицательными: минус 1%, минус 2% и т.д.

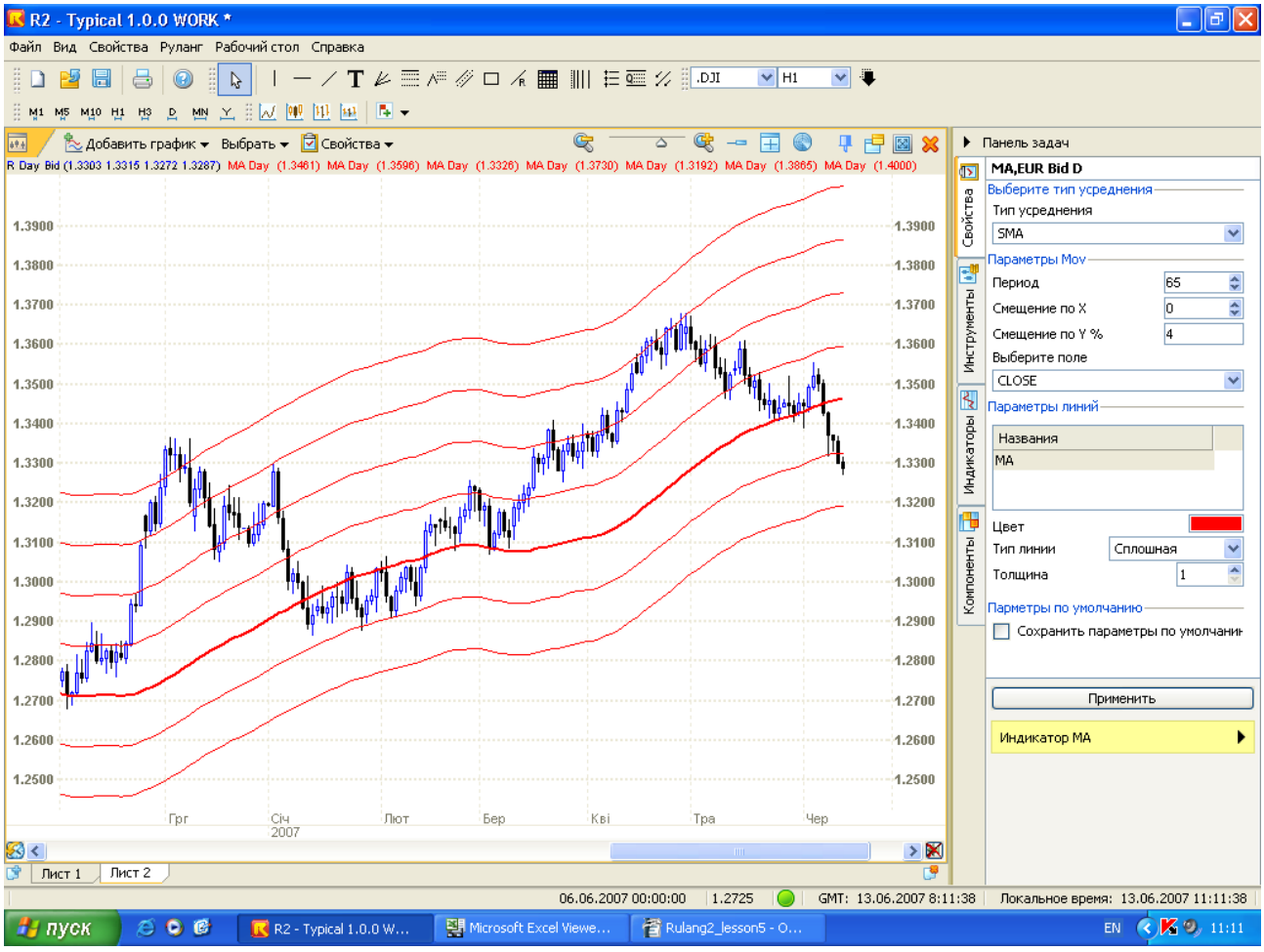

Рис. 5.2

В принципе в построении этих каналов нет ничего сложного, и поэтому его спокойно можно выполнять вручную, строя одну линию за другой. Однако, построив таких каналов, например, штук 5-6 на разных валютных парах, а в каждой системе каналов должно быть около 6-8 линий, Вы поймете, что несмотря на свою простоту эта процедура довольно-таки громоздка и утомительна. Ведь в нашем примере Вам бы пришлось провести порядка 30-50 линий! А это не так быстро и удобно, как может показаться с первого взгляда, ведь для каждой линии по отдельности нужно указать два параметра: период и сдвиг.

Чтобы упростить и ускорить процедуру построения таких каналов вполне разумно было бы создать индикатор, который строил такие каналы. Ему достаточно было бы указать только три параметра (период и сдвиг линий, а также количество этих линий). И он тут же построит все указанные линии. Как видие, процедура построения параллельных каналов ускоряется в несколько раз.

В этом индикаторе центральное место будет занимать оператор цикла, который будет вычислять значения для всех заданных линий параллельных каналов. Он будет работать с массивом, каждый из элементов которого и будет представлять из себя одну из линий

параллельных каналов.

Небольшое уточнение. У этого индикатора на самом деле будет 4 параметра, а не три, как мы говорили выше. Дело в том, что 4-й параметр необходим для того, чтобы указать индикатору, с какой линии следует строить параллельные каналы. Чтобы было понятно, о чем идет речь, давайте введем такое понятие, как номер линии. Давайте посмотрим на рис. 5.2. Там показано семь линий. Для того, чтобы нам (а также компьютеру) было понятно, о какой конкретной линии идет речь, было бы вполне естественно присвоить этим линиям какие-то имена (а еще лучше – номера). Эту нумерацию, можно было бы, например, организовать таким образом. Центральной линии (она на рис. 5.2 показана утолщенной) присваивается номер ноль. Тем линиям, которые находятся выше ее, присваиваются по мере удаления от центральной линии положительные номера (+1, +2, +3, и т.д.). Аналогично, те линии, которые находятся ниже центральной, получают отрицательные номера также последовательно по мере удаления от нее (минус 1, минус 2, и т.д.). Если внимательно посмотреть на рис. 5.2, то не трудно заметить, что там нарисованы линии начиная с номера «минус 2» по номер «+4» (т.е. всего семь линий, включая нулевую).

Так вот, четвертый параметр это и есть тот самый номер линии, с которой следует начинать строить параллельные каналы. Например, если бы мы захотели, чтобы наш индикатор построил систему линий, которая изображена на рис. 5.2, то мы должны были бы указать ему следующие параметры:

период = 65;
шаг = 1%;
число линий = 7;
номер первой линии = -2.

Индикатор, получив такой набор параметров, в первую очередь вычислит значение для линии номер «минус 2», а затем последовательно одно за другим вычислит значения для остальных шести линий (поскольку всего их семь). Причем, как Вы уже догадались, эти вычисления для расчета значений линий будут выполняться в цикле.

Здесь, правда есть небольшой нюанс. Отрицательные числа не очень хорошо применять для нумерации линий. Во-первых, многие люди отрицательные числа воспринимают намного хуже, чем положительные (чисто психологически). А во-вторых, отрицательные числа нельзя указывать в качестве номеров элементов массивов. Например, обращение к массиву Line[-2], приведет к сообщению об ошибке.

Поэтому мы поступим следующим образом: просто сдвинем нумерацию на величину 100. Теперь центральная линия будет иметь номер 100, линия, идущая чуть выше нее, будет иметь номер 101, а линия, идущая, чуть ниже ее, будет иметь номер 99. В частности, самая нижняя линия, изображенная на рис. 5.2, будет иметь номер 98 (т.е. к числу 100 прибавляется старый номер линии, в данном случае - «минус 2»).

Давайте назовем наш индикатор «P_chan», что означает «параллельные каналы», и посмотрим как выглядит текст его программы.

```
1:    /* P_chan */
2:    Period=inparam("Period",1,1000,65);
3:    Step=inparam("Step",0,10,1);
4:    AmountL=inparam("Amount of Lines",1,19,7);
5:    FirstL=inparam("Number of first Line",80,120,97);
```

```
 6:      Array: Line[19];
 7:      x0=mov(C,Period,s);
 8:      Stp=x0*Step/100;
 9:      i=1;
10:      while (i<=19) begin
11:              Line[i]=x0+(i-101+FirstL)*Stp;
12:              i=i+1;
13:      end;
14:      if AmountL>=1 then Line[1];
15:      if AmountL>=2 then Line[2];
16:      if AmountL>=3 then Line[3];
17:      if AmountL>=4 then Line[4];
18:      if AmountL>=5 then Line[5];
19:      if AmountL>=6 then Line[6];
20:      if AmountL>=7 then Line[7];
21:      if AmountL>=8 then Line[8];
22:      if AmountL>=9 then Line[9];
23:      if AmountL>=10 then Line[10];
24:      if AmountL>=11 then Line[11];
25:      if AmountL>=12 then Line[12];
26:      if AmountL>=13 then Line[13];
27:      if AmountL>=14 then Line[14];
28:      if AmountL>=15 then Line[15];
29:      if AmountL>=16 then Line[16];
30:      if AmountL>=17 then Line[17];
31:      if AmountL>=18 then Line[18];
32:      if AmountL>=19 then Line[19];
```

В первой строке в качестве комментария указано название нашего индикатора.

В строках со 2-й по 5-ю задаются параметры индикатора: период, шаг, количество линий и номер первой линии.

В 6-й строке описывается массив линий Line. Он имеет 19-ть элементов. Это означает, что наш индикатор может построить одновременно не более 19-ти линий.

В 7-й строке вычисляется значение для центральной линии, и оно запоминается в переменной x0.

В 8-й строке вычисляется величина шага (в пунктах), и это значение запоминается в переменной Stp.

В строках с 9-ю по 13-ю выполняется вычисление значений заданного количества линий, которые запоминаются в массиве Line. При их вычислении используются величины значений центральной линии (переменная x0), шага (переменная Stp) и номера первой линии (переменная FirstL).

И, наконец, в строках с 14-ю по 32-ю выполняется вывод на экран нужного числа линий.

Если построить этот индикатор с параметрами по умолчанию, то получится нечто, похожее на рис. 5.3.

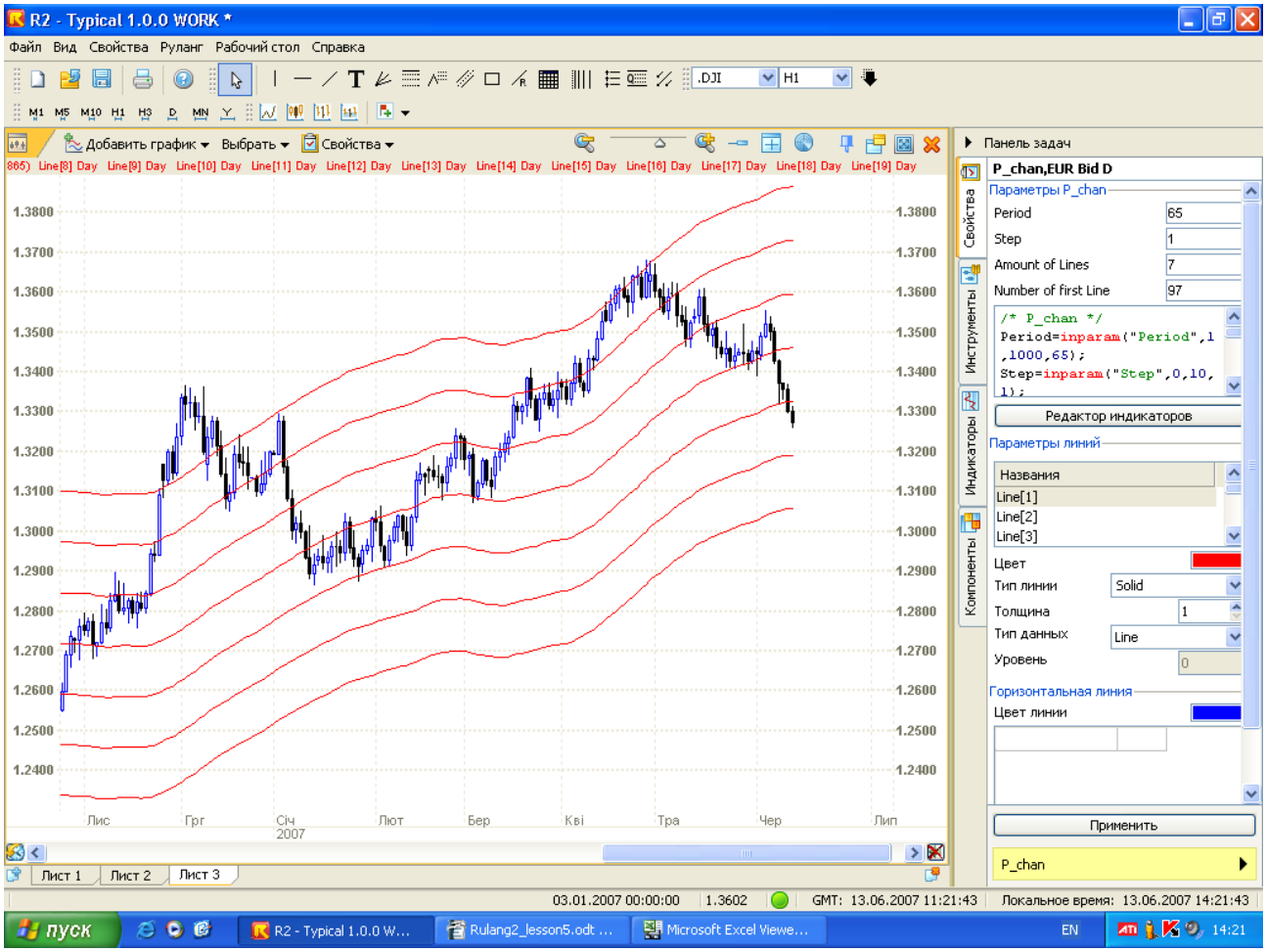

Рис. 5.3.

Очень похоже на то, что показано на рис. 5.2, где параллельные каналы были построены вручную. Чтобы добиться полного соответствия с рис. 5.2, нужно в параметрах нашего индикатора изменить номер первой линии с 97 на 98 (по старой нумерации – с «минус 3» на «минус 2»).

Заключение

Итак, вот мы с Вами и прошли, как говорится, «азы» программирования на языке РУЛАНГ. Как видите, он достаточно прост. Многие интересные вещи можно увидеть, написав буквально несколько строк программы. Но, не смотря на свою простоту, он позволяет сделать очень многие и очень серьезные вещи – даже провести автоматическое тестирование Вашей торговой системы и рассчитать ее кривую капитала.

Желаем Вам не останавливаться на достигнутом, а продолжать развивать свои навыки программирования, создавать свои собственные индикаторы, создавать и тестировать свои собственные торговые системы. Это в конечном итоге приведет к тому, что Вы станете не просто трейдером-ремесленником, который использует и копирует чужие (зачастую – очень устаревшие) торговые методы и идеи. Обычно трейдеры-ремесленники «живут» до «первого милиционера», то есть до первого серьезного изменения рынка. Вы же станете трейдером-интеллектуалом, который будет способен принять от рынка любой вызов, и при этом не просто «остаться на плаву», а достичь еще больших успехов даже при самых серьезных изменениях рыночных условий.

Приложение

Пример создания программы. Расчет кривой капитала торговой системы

Сейчас мы рассмотрим очень интересную программу. Она позволяет рассчитать кривую доходности (еще говорят – кривую капитала) конкретной торговой системы. Программы, которые могут рассчитывать такую кривую, еще называются тестерами, поскольку они, по сути, тестируют торговую систему. Причем сейчас мы не просто приведем текст этой программы, а разберем сам процесс ее создания.

Сначала рассмотрим торговую систему.

Допустим, стоит задача открываться по пересечению короткой и длинной средней. Закрываться будем либо по достижении ордера Take-profit (TP), либо по срабатыванию ордера Stop-loss (SL). Пока какой-то из ордеров не сработает, позицию держим открытой. Сразу комментарий: такая совсем простая система не есть самая эффективная. Рынок давно научился сдерживать захватнические порывы «механических трейдеров»… Но все равно попробуем написать тестер.

Итак, покупка совершается, когда короткая средняя пересекает длинную среднюю снизу вверх, продажа – когда пересечение сверху вниз. Сделки для простоты совершаем по цене close сигнальной свечи.

Что требуется: найти момент открытия позиции, затем вплоть до момента закрытия отслеживать положение цены относительно ордеров TP и SL. Если какой-то ордер достигнут – закрыть позицию. Если средние дали сигнал на разворот – закрыть позицию и открыть противоположную.

Начнем.

Пишем запрос на ввод параметров средних с клавиатуры. Допустим, мы хотим работать как с обычными средними, так и со сдвинутыми. Значит, потребуется всего четыре параметра: p1 и p2 – периоды короткой и длинной средней, s1 и s2 – периоды сдвига средней. Вот как выглядит запрос на языке РУЛАНГ:

(1) – так мы будем указывать части кода, чтобы на них ссылаться.

```
p1=inparam("Period 1",0,10000,5);

p2=inparam("Period 2",0,10000,10);

s1=inparam("Sdvig 1",0,10000,0);

s2=inparam("Sdvig 2",0,10000,0);
```

В данном случае сдвиги мы оставили нулевые. Захотим – сдвинем.

Рассчитаем наши средние с учетом сдвига, чтобы потом было короче их записывать.

Их текущие значения:

(2)

m1=ref(mov(c,p1,s),-s1);

m2=ref(mov(c,p2,s),-s2);

Их значения 1 шаг назад (будут нужны для отслеживания факта пересечения средних):

ms1=ref(m1,-1);

ms2=ref(m2,-1);

Теперь пишем блоки открытия позиций. Интересующее событие – пересечение средних. Действие при возникновении события, которое мы должны совершить, – определение уровня открытия позиции и уровней ордеров на закрытие позиции.

Запрос на размеры Take Profit и Stop Loss, а также спрэда (спрэд же разный может быть, и нам надо заранее его указать) записывается так:

(3)

tp=inparam("Take Profit",0,1,0.10);

st=inparam("Stop Loss",0,1,0.01);

sp=inparam("Spread",0,1,0.0005);

Если пересечение вверх, то производим покупку BUY и фиксируем цену открытия (popb = price open buy), уровень Take-profit (ptpb) и Stop-loss (pstb). Спрэд учтем в окончательном расчете результата – будем уменьшать на его размер прибыль и увеличивать убыток. А при определении цен и фактов открытий и закрытий просто будем ориентироваться на график цен Bid, поступающих в RUMUS. Итак, вот как определяются интересующие нас цены:

(4) – пункт позже будет еще чуть-чуть откорректирован

If m1>m2 and ms1<ms2 then

begin

popb=c; ptpb=c+tp; pstb=c-st;

end;

Операторы begin и end нужны для того, чтобы при выполнении (или при невыполнении) условия, следующего после if, одновременно были выполнены (! – или

невыполнены) несколько команд, заключенных внутрь «begin – end».

То же самое – и для продажи, только ищем пересечение средних вниз, а ордера отсчитываются в другую сторону:

If m1<m2 and ms1>ms2 then

begin

pops=c; ptps=c-tp; psts=c+st;

end;

Следующие шаги очень просты. Нужно делать шаг за шагом вправо по графику и проверять, сработали ли какие-то ордера, а также нет ли сигнала на разворот. Но чтобы это делать, нужно будет в самом начале текста нашего индикатора, который по сути будет кривой капитала, «объявить переменные». Такими переменными будут цены открытия и уровни ордеров, а также два новых параметра posb (position buy) и poss (position sell), которые будут равны 1 и -1 при открытых позициях, а нулю – при закрытых. Мы их назовем показателями открытых позиций, т.е. если posb=1, значит у нас открыта длинная позиция. Объявлять переменные нужно для того, чтобы потом можно было даже через десять-двадцать-сто шагов узнать, какие цены открытия, какие цены ордеров, какая позиция открыта – на покупку или на продажу.

(5) – встанет перед 1!

variable: popb($data);

variable: ptpb($data);

variable: pstb($data);

variable: pops($data);

variable: ptps($data);

variable: psts($data);

variable: posb($data);

variable: poss($data);

В принципе, можно было бы обойтись и без разделения цен на цены buy и цены sell, но так проще понять, поэтому оставим так. Вышеприведенную строку запишем в в самую шапку текста индикатора. Объявление этих переменных должно производиться до того, как они первый раз встречаются в тексте.

Еще одно замечание. Зафиксировав, что есть сигнал на открытие позиции вверх или вниз, мы в строчках (4) должны будем сразу указать, открыта ли нами позиция вверх или вниз. Поэтому формулы (4) станут такими:

(4) – откорректированы

```
posb=0; poss=0;

If m1>m2 and ms1<ms2 then

begin

popb=c; ptpb=c+tp; pstb=c-st;

posb=1;

end;

If m1<m2 and ms1>ms2 then

begin

pops=c; ptps=c-tp; psts=c+st;

poss=-1;

end;
```

Наконец-то теперь, делая шаг за шагом вправо, мы:

- «вспоминаем», какие были цены открытия и ордеров и каковы были показатели открытой позиции «вчера»;
- проверяем положение цены закрытия относительно ордеров;
- проверяем, нет ли сигнала на разворот (или на открытие, если позиций не было);
- открываем или закрываем позицию;
- рассчитываем доход.

Чтобы вспомнить цены прошлого шага, мы должны проверить, была ли «вчера» открыта позиция, а затем текущим ценам присвоить вчерашние значения. Само собой, если «вчера» показатель позиции posb или poss был равен нулю, то соответствующей позиции не было. Попутно заметим, что в момент закрытия позиции мы в будущем обязательно должны будем показателю соответствующей позиции присваивать значение 0. Итак, для покупки и для продажи процесс «вспоминания» и переприсваивания значений цен и показателя позиции будут выглядеть так:

(6)

```
if ref(posb,-1)=1 then begin
popb=ref(popb,-1);
ptpb=ref(ptpb,-1);
pstb=ref(pstb,-1);
posb=ref(posb,-1);
end;

if ref(poss,-1)=-1 then begin
pops=ref(pops,-1);
ptps=ref(ptps,-1);
psts=ref(psts,-1);
poss=ref(poss,-1);
end;
```

Приступаем к проверке ордеров Take Profit для длинной и короткой позиции. Суть проверки вот в чем: если открыта длинная позиция и есть пересечение цены High с уровнем ордера Take Profit, фиксируем результат сделки resb (result buy). Для короткой позиции поступаем аналогично. При расчете результата от прибыли отнимаем спрэд.

(7)

```
if posb=1 and ref(posb,-1)=1 and h>ptpb then
begin
resb=ptpb-popb-sp;
posb=0;
end;
```

if poss=-1 and ref(poss,-1)=-1 and l<ptps then

begin

ress=pops-ptps-sp;

poss=0;

end;

Обратите внимание, что для правильного расчета дохода прибыльные сделки должны быть плюсовыми по результату, поэтому для короткой позиции мы не от ptps отняли pops, а наоборот – от pops отняли ptps. Цена открытия короткой позиции выше цены закрытия. По факту закрытия позиции показателю позиции posb или poss присвоили нулевое значение (позиция закрыта, позиции нет).

Теперь пишем закрытие по ордеру Stop Loss. Результаты будут отрицательными числами. Отнятый также как в случае с прибылью спрэд будет фактически прибавлен к убытку, увеличив его абсолютное значение.

(8)

if posb=1 and ref(posb,-1)=1 and l<pstb then

begin

resb=pstb-popb-sp;

posb=0;

end;

if poss=-1 and ref(poss,-1)=-1 and h>psts then

begin

ress=pops-psts-sp;

poss=0;

end;

Теперь следует рассчитать результаты в том случае, когда ордера достигнуты не были, но подан сигнал на переворот. Условие, которое нужно искать для закрытия длинной позиции, такое: показатель длинной позиции был «вчера» и есть «сегодня» +1, а показатель отрицательной изменился с 0 на -1, то есть позиция вниз только-только открывается. Соответственно, при открытии позиции вниз мы фиксируем результаты длинной позиции и показатель длинной позиции в итоге приравниваем к нулю. Для смены направления с нисходящего на восходящее – зеркальная ситуация.

(9)

```
if posb=1 and ref(posb,-1)=1 and ref(poss,-1)=0 and poss=-1 then
begin
resb=c-popb-sp;
posb=0;
end;

if poss=-1 and ref(poss,-1)=-1 and ref(posb,-1)=0 and posb=1 then
begin
ress=pops-c-sp;
poss=0;
end;
```

Теперь у нас на руках все результаты по всем сделкам – как прибыльные, так и убыточные. Какова же общая прибыль отдельно по длинным, а отдельно – по коротким позициям? И какая – совокупная?

Ниже в пункте (10) – расчеты этих параметров. Есть такое понятие – «кумулятивная сумма». Кумулятивная сумма всех длинных позиций покажет общую прибыль resultBuy, а кумулятивная сумма всех коротких позиций – прибыль resultSell. Сложив эти две величины, мы на данный конкретный момент получим итоговую зафиксированную прибыль result, которую мы заработали, совершая сделки с применением средних как в длинную, так и в короткую сторону и закрываясь по ордерам и путем переворота.

Две последние строчки пункта (10) позволяют нам рассчитывать «текущую незафиксированную прибыль или убыток» - «curve» (кривая капитала в пунктах).

(10)

resultBuy=cum(resb);

resultSell=cum(ress);

result=resultBuy+resultSell;

if posb=1 then begin buy=result+c-popb; curve=buy; end; else curve=result;

if poss=-1 then begin sell=result+pops-c; curve=sell; end;

(11)

Теперь выведем результаты на экран, а также укажем линию нулевой прибыли. В параметрах индикатора линию curve надо сделать синей (вид обычный – линия), а линии buy и sell сделать зеленой и красной, отразив в виде гистограммы. В результате будет видно, где позиций не было открыто вообще, где работала длинная, а где – короткая. На том баре, где длинная позиция была открыта, впервые появится зеленая гистограмма, где длинная позиция закрыта, зеленая гистограмма buy будет равна нулю. На баре, где короткая открыта, появится красная гистограмма, а там, где короткая закрыта, красная станет равна нулю.

curve;

buy; sell;

0;

Вот и родилась наша кривая дохода в пунктах. Как видите, мы можем все – и даже нарисовать кривую капитала с использованием языка РУЛАНГ. Если есть четкое понимание, как бы вы хотели управлять объемами контрактов, то и это вполне можно было бы запрограммировать. Надеемся, разберетесь уже сами.

Итоговый текст индикатора, строящего кривую капитала (просто скопируйте его):

// !_AK_Cap_2Mov, 30.06.2006, Alexey Kiyanitsa, Forex Club

// Capital curve for 2 Moving Average Trading System

// Open when cross and close when order made or when cross

//

```
// (5)
variable: popb($data);
variable: ptpb($data);
variable: pstb($data);
variable: pops($data);
variable: ptps($data);
variable: psts($data);
variable: posb($data);
variable: poss($data);
// (1)
p1=inparam("Period 1",0,10000,5);
p2=inparam("Period 2",0,10000,10);
s1=inparam("Sdvig 1",0,10000,0);
s2=inparam("Sdvig 2",0,10000,0);
// (2)
m1=ref(mov(c,p1,s),-s1);
m2=ref(mov(c,p2,s),-s2);
ms1=ref(m1,-1);
ms2=ref(m2,-1);
// (3)
tp=inparam("Take Profit",0,1,0.10);
st=inparam("Stop Loss",0,1,0.01);
sp=inparam("Spread",0,1,0.0005);
```

```
// (4)
posb=0; poss=0;
If m1>m2 and ms1<ms2 then
begin
popb=c; ptpb=c+tp; pstb=c-st;
//if ref(poss,-1)=0 then posb=1; else posb=0;
posb=1;
end;
If m1<m2 and ms1>ms2 then
begin
pops=c; ptps=c-tp; psts=c+st;
//if ref(posb,-1)=0 then poss=-1; else poss=0;
poss=-1;
end;
// (6)
if ref(posb,-1)=1 then begin
popb=ref(popb,-1);
ptpb=ref(ptpb,-1);
pstb=ref(pstb,-1);
posb=ref(posb,-1);
end;
if ref(poss,-1)=-1 then begin
pops=ref(pops,-1);
ptps=ref(ptps,-1);
```

```
psts=ref(psts,-1);

poss=ref(poss,-1);

end;

// (7)

if posb=1 and ref(posb,-1)=1 and h>ptpb then

begin

resb=ptpb-popb-sp;

posb=0;

end;

if poss=-1 and ref(poss,-1)=-1 and l<ptps then

begin

ress=pops-ptps-sp;

poss=0;

end;

//(8)

if posb=1 and ref(posb,-1)=1 and l<pstb then

begin

resb=pstb-popb-sp;

posb=0;

end;

if poss=-1 and ref(poss,-1)=-1 and h>psts then

begin

ress=pops-psts-sp;
```

```
poss=0;
end;
//(9)
if posb=1 and ref(posb,-1)=1 and ref(poss,-1)=0 and poss=-1 then
begin
resb=c-popb-sp;
posb=0;
end;
if poss=-1 and ref(poss,-1)=-1 and ref(posb,-1)=0 and posb=1 then
begin
ress=pops-c-sp;
poss=0;
end;
//(10)
resultBuy=cum(resb);
resultSell=cum(ress);
result=resultBuy+resultSell;
if posb=1 then begin buy=result+c-popb; curve=buy; end; else curve=result;
if poss=-1 then begin sell=result+pops-c; curve=sell; end;
//(11)

curve;
buy; sell;
0;
```

Рис. 6.1

О графике, изображенном на рис. 6.1. Этот график дневных свечей отражает доход в пунктах для британского фунта GBP за период с 01.01.2005 по 30.06.2006 при работе со средними, имеющими периоды 5 и 10. Тэйк-профит – 0.1000, стоп-лосс – 0.0100. Закрытие по ордерам или по цене закрытия сигнальной свечи при подаче средними приказа о совершении сделки в обратном направлении. Явный минус этой простейшей системы – ее чувствительность к параметрам. Чуть изменишь – дохода нет. На других валютах тоже «не уха». Таким образом, систему мы не рекомендуем. Однако делаем полезный вывод: чтобы стать успешным трейдером, имеет смысл учиться, потому что все гениальное просто, но, как правило, оно не совсем уж примитивно.

ЛИТЕРАТУРА

1. Ахо А., Хопкрофт Дж., Ульман Дж. «Построение и анализ вычислительных алгоритмов». — М.: Мир, 1979
2. Бигеев Р. «Параллельные каналы». Казань, 2005.
3. Вильямс Л. «Долгосрочные секреты краткосрочной торговли», М.:Аналитика, 2001
4. Вирт Н. «Алгоритмы и структуры данных». - М.:Мир, 1989.
5. Кнут Д. «Искусство программирования для ЭВМ» т. 1. Основные алгоритмы. — С.-П.: Вильямс, 2000
6. Колби Роберт «Энциклопедия технических индикаторов рынка», М.:Альпина Бизнес Букс, 2004
7. Швагер Джек, «Технический анализ. Полный курс», М.:Альпина Бизнес Букс, 2005