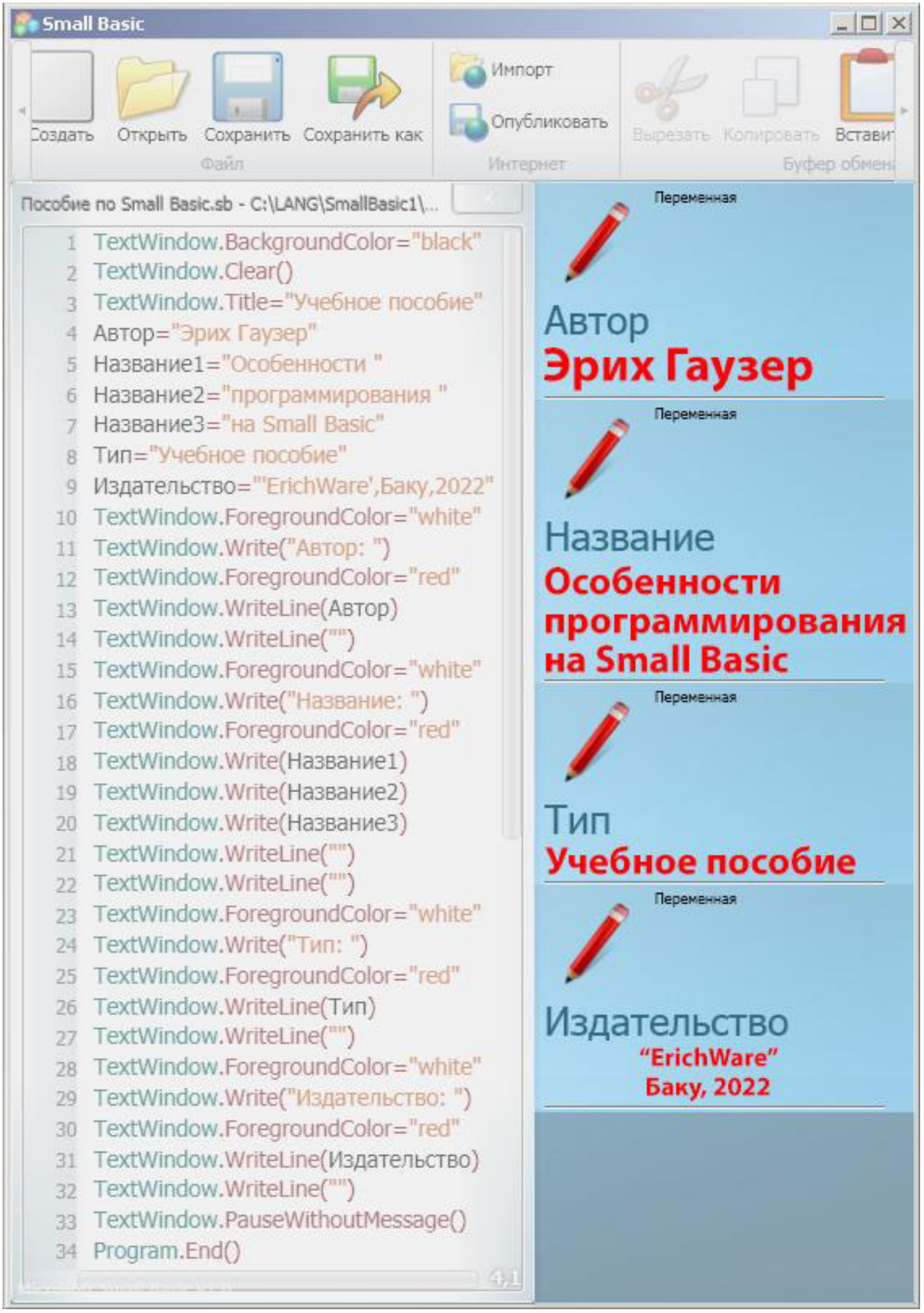

Small Basic
Создать
Открыть
Сохранить
Сохранить как
Файл
Импорт
Опубликовать
Интернет
Вырезать
Копировать
Встави
Буфер обмен
Пособие по Small Basic.sb - C:\LANG\SmallBasic1\...
1 TextWindow.BackgroundColor="black"
2 TextWindow.Clear()
3 TextWindow.Title="Учебное пособие"
4 Автор="Эрих Гаузер"
5 Название1="Особенности "
6 Название2="программирования "
7 Название3="на Small Basic"
8 Тип="Учебное пособие"
9 Издательство="'ErichWare',Баку,2022"
10 TextWindow.ForegroundColor="white"
11 TextWindow.Write("Автор: ")
12 TextWindow.ForegroundColor="red"
13 TextWindow.WriteLine(Автор)
14 TextWindow.WriteLine("")
15 TextWindow.ForegroundColor="white"
16 TextWindow.Write("Название: ")
17 TextWindow.ForegroundColor="red"
18 TextWindow.Write(Название1)
19 TextWindow.Write(Название2)
20 TextWindow.Write(Название3)
21 TextWindow.WriteLine("")
22 TextWindow.WriteLine("")
23 TextWindow.ForegroundColor="white"
24 TextWindow.Write("Тип: ")
25 TextWindow.ForegroundColor="red"
26 TextWindow.WriteLine(Тип)
27 TextWindow.WriteLine("")
28 TextWindow.ForegroundColor="white"
29 TextWindow.Write("Издательство: ")
30 TextWindow.ForegroundColor="red"
31 TextWindow.WriteLine(Издательство)
32 TextWindow.WriteLine("")
33 TextWindow.PauseWithoutMessage()
34 Program.End()
4:1
Переменная
Автор
Эрих Гаузер
Переменная
Название
Особенности программирования на Small Basic
Переменная
Тип
Учебное пособие
Переменная
Издательство
"ErichWare"
Баку, 2022

Автор – Гаузер Эрих Генрихович, программист из города Баку. Сайтов у меня много, структурированы они в некое подобие пирамиды. Вершина пирамиды – мой главный личный сайт hauserich.vgd.name. Приглашаю в гости!

Аннотация

Данная книга – не учебник по программированию как таковому. Это именно учебное пособие по конкретному языку программирования – Small Basic фирмы MicroSoft.

Почему я написал эту книгу и вообще, почему выбран этот язык – написано в самой книге. Надеюсь, она вам пригодится. Но напоминаю, что она не содержит базовых описаний и принципов программирования. Подразумевается, что использующий ее человек эти базовые знания имеет. Тем более что сейчас все это проходят в школе.

Книга на моем сайте доступна бесплатно, там же можно скачать архив со всеми приведенными тут примерами (в тексте книги это указано), а также архив с готовыми программами из «Приложения». Однако, вы можете сделать автору приятное, купив эту книгу (вместе со всеми исходными текстами упомянутых тут программ) за символическую плату. Просто для поддержки дальнейшего авторского творчества.

Итак, приступим!

Оглавление

Почему «Small Basic»? Необходимое предисловие

Когда я стал работать программистом, передо мной встал вопрос выбора языка создания программ. Было это в начале 90-х, во времена MS-DOS. К тому времени я умел писать программы для советских калькуляторов, что было полезно само по себе, но тут не помогло бы. Кроме того, в институте я научился писать программы на Фортране для ЭВМ серий ЕС и СМ. Но для IBM PC, на которых мне предстояло работать, Фортрана не было. Были Бейсик, Си и Паскаль, реализованные однотипными турбо-оболочками фирмы Borland.

Я выбрал Бейсик. Не столько за его схожесть с Фортраном, сколько за его простоту и логичность синтаксиса, отсутствие лишней «бюрократии», но главное – за наличие поистине огромных возможностей, встроенных прямо в язык, о которых ни Си, ни Паскаль не могли даже мечтать.

Через некоторое время я перешел на транслятор «QuickBasic 4» фирмы Microsoft. И на старом «tb1», и на новом «qb4» я (с применением ассемблера) сделал свою интерфейсную библиотеку, которая давала мне полный доступ к управлению мышью, портам клавиатуры, файловой системе и т.д. Я сделал свой набор динамических (векторных) шрифтов (чего не было в ДОС) и многое другое.

Но время MS-DOS уходило. К программированию под Windows я перешел довольно поздно: жаль было бросать наработанное за много лет, тем более что я освоил ДОСовский Бейсик, возможно, глубже и полнее, чем кто-либо еще (программа анализа SEM-изображений яркое тому подтверждение). Объектно-ориентированное программирование (ООП), принятое в новой ОС, показалось мне скучным, ресурсоемким и не слишком приятным. Но выхода не было!

Я уже давно пишу под Windows, и постепенно перевел почти все свои старые программы на эту платформу. Сначала писал на «Visual Basic 6», потом, опять же без удовольствия, вынужден был перейти на программирование в среде «Microsoft.NET». Однако все эти языки и оболочки требовали своей установки на компьютер, занимали уйму места и были достаточно сложны в использовании. А мне нужен был язык простой, компактный и работающий буквально с любой флешки. Зачем?

Ну, в компьютерной жизни достаточно часто возникают мелкие, но весьма нудные задачи, для решения которых проще написать программу, чем выполнять их вручную. Даже если это программа «на один раз». Сначала, еще во времена ДОС, я создал язык «Эллочка» («Ellochka») и написал к нему полноценный интерпретатор. Это было больше 20 лет назад, с тех пор я так и не нашел время перевести транслятор в среду Windows (хотя недавно начал этот очень долгий процесс). Но язык-то нужен!

Одно время я использовал «Visual Basic 3» - простой, удобный и неприхотливый. Но под Windows 7 он уже работал нестабильно, а в более новых ОС ни он сам, ни написанные на нем программы не запускаются вообще. Что делать? И я наткнулся на «Small Basic» - простой и обучающий язык от компании Microsoft. Разумеется, очень обрадовался: это же то, что я искал! Но… Но язык этот при всех своих плюсах имеет очень странные особенности, которым и посвящена данная книга. Надеюсь, она будет полезна всем начинающим программистам.

Глава 1. Общие характеристики

К сожалению, «Small Basic» (далее SB) не предназначен для создания реальных и полезных программ. Его смысл – в начальном обучении самому процессу программирования, знакомстве с базовыми понятиями ООП («объект», «свойство», «событие», «метод»), а также с циклами, условиями, процедурами и т.д.

Сама оболочка (я использую версию 1, потому что более новые не запускаются на старых ОС, да и на новых глючат) содержит текстовый редактор с подсветкой синтаксиса, всплывающие подсказки и минимальный набор операций с программой: открыть, сохранить, запустить…

Как ни удивительно, но обучающая(!) система не имеет никаких встроенных инструментов отладки программ. То есть найти и исправить ошибку – еще тот квест! Я такое помню по старому Фортрану на больших ЭВМ, где кроме промежуточной печати результатов никаких инструментов не было. Так что если программа ваша работает как-то «не так», единственный выход – это везде где есть подозрительное место вставлять в ее текст вывод значений переменных в виде привычных виндосовских сообщений или же в текстовое окно. Дикость, но…

Кроме того, нет никакой отдельной встроенной помощи, нет даже списка объектов! Хотя списки свойств, методов и событий на экран выводятся. Но сначала нужно набрать в редакторе имя объекта, списка которых, повторяю, нигде нет.

Нет в языке и многого другого, без чего реальную программу написать сложно или вовсе невозможно. Ну, например (пояснения будут дальше в книге):

- нет работы с двоичными файлами;
- нет опроса клавиатуры в консоли;
- нельзя менять цвет фона текстбокса и кнопок;
- библиотеку нельзя поместить в системный каталог;
- всего два контрола: кнопка и текстовое поле;
- текст только в юникоде.

Да, для запуска любой написанной на SB программы, нужна библиотека под именем «SmallBasicLibrary.dll». Все бы ничего, ведь упомянутые выше системы «Visual Basic» всех версий тоже нуждаются в своих подобных файлах. Но разница в том, что для всех VB эти библиотеки либо входят в набор ОС, либо их легко раз и навсегда записать в системную папку Windows. На SB же данный файл придется переносить вместе с выполняемым exe-модулем. Недаром сама оболочка автоматически создает его в той же папке, куда помещает откомпилированную вами программу. И плодятся эти файлы несчетно…

Еще один минус – поддержка текстов только в юникоде. Иначе говоря, обычные текстовые файлы ни прочитать, ни создать вашей программой не получится. А если учесть, что язык вообще работает только с текстовыми файлами, огромный пласт задач из списка решаемых на SB автоматически исключается. И для меня, например, это критично! Так что при всей упомянутой выше радости, я так и не смог сделать этот язык своим «дежурным инструментом». Впрочем, книга не об этом.

Важнейшие примеры из книги можно скачать в виде архива на сайте автора.

Глава 2. Основные операторы

Любой язык программирования содержит в себе набор базовых элементов: условия, циклы, подпрограммы и, конечно, оператор присваивания. Реализация этих элементов может быть весьма разной, в том числе и очень богатой, широкой и т.д. Но на языке «Small Basic» эти базовые функции реализованы крайне минималистично.

Во-первых, в одной строке программы может располагаться только один оператор. Это сильно удлиняет листинг программы, но, конечно, упрощает создание транслятора. Я в своей «Эллочке» тоже ввел такое ограничение. Это не страшно, но помнить об этом нужно. Особенно при переводе программ с других версий Бейсика.

Во-вторых, на SB полностью отсутствует понятие типа переменной. Когда-то я выбрал Бейсик в силу его низкой бюрократичности (по сравнению с тем же Паскалем), но это все же перебор! Иначе говоря, что вы получите в результате таких действий?

```
A=1
B="2"
C=A+B
```

Вы получите результат «3», хотя по идее такой код ошибочен и не должен выполняться. Но отловить лишние кавычки (или отсутствие таковых) при отладке программы будет очень трудно.

Скажу больше! Если вы напишете в первой строке «A="1"», вы получите не ожидаемый результат в виде строки «12», а ту же самую цифру «3»! Чтобы получить «12» нужно не складывать переменные, а использовать специальный оператор работы с текстами:

```
A="1"
B="2"
C=Text.Append(A,B)
```

Причем в этом случае кавычки в переменных A и B можно вообще не указывать!

Так что иногда некоторая бюрократия бывает даже полезна, чтобы не было полной каши и путаницы…

Условный оператор имеет в принципе стандартный «блочный» формат:

```
If A>0 Then
  B=1
ElseIf A<0 Then
  B=-1
Else
  B=0
EndIf
```

Разумеется, блоки «**ElseIf**» и «**Else**» не являются обязательными, а блоков «**ElseIf**» может быть несколько. Но, к сожалению, простой однострочный вариант, присутствующий во всех языках («`If A>0 Then B=1`»), тут не работает. Это удлиняет листинг программы, но увы.

Что касается циклов, то обычный «цикл с параметром» на SB тоже имеет вполне привычный вид, но с двумя особенностями. Во-первых, вместо оператора «next» используется «**EndFor**» (что приводит к необходимости корректировки кода,

перенесенного из других версий), а во-вторых, нет оператора прерывания цикла, так что без не слишком «кошерного» «**Goto**» тут не обойтись.

```
For I=A To B Step C
  D=D+I
EndFor
```

Есть тут и цикл «While», но только в варианте с «предусловием»:

```
A = Clock.ElapsedMilliseconds
While (Clock.ElapsedMilliseconds - A < TIM)
EndWhile
```

Данный пример задерживает работу программы на заданное в переменной TIM число миллисекунд. Разумеется, внутри блока «While» могут содержаться любые операторы в любом количестве.

Условие (как и в операторе «If») может содержать логические операции «And» и «Or», но никаких других логических операций тут нет. Надо еще учитывать, что принятая в обычном Бейсике эквивалентность числовых значений переменных значениям логическим (если равно 0, то ложь, если не равно 0, то истина) здесь не работает. Во всяком случае, рисковать не стоит, и вместо привычного «If A Then» надо писать «If A<>0 Then».

Теперь о процедурах, ибо и тут есть свои особенности.

Во-первых, все переменные в программе на SB являются глобальными. То есть, в процедуру нельзя передать параметры, да и незачем это делать. Поэтому, если текст программы большой и переменных много, нужно четко помнить что где использовано, а лучше даже выработать систему имен для обозначения локальных переменных. Иначе потом замучаетесь ошибки искать.

Во-вторых, здесь нет понятия «функция», то есть создать можно только процедуру и результат ее работы нельзя присвоить какой-то переменной. Это не смертельно, хотя иногда неудобно.

В частности, есть математическое понятие «знак числа», обычно обозначаемое как функция «sgn». Такой встроенной функции на SB нет и надо написать ее самостоятельно. Собственно, вышеуказанный пример условного оператора фактически и дает текст этой функции. Его можно оформить в виде процедуры:

```
Sub sgn
  If A>0 Then
    B=1
  ElseIf A<0 Then
    B=-1
  Else
    B=0
  EndIf
EndSub
```

А потом придется писать такой код:

```
'вычисление переменной A
sgn()
'использование значения знака числа A в виде переменной B
```

И помнить, что в данной процедуре значение глобальной переменной B будет изменено.

О массивах и математике поговорим в Главе 8, а пока перейдем к элементам ООП, работе с которыми, собственно, и должен вас научить язык «Small Basic».

Глава 3. Объекты на SB

Для создания своих программ вы можете использовать следующие объекты:

Array *– позволяет работать с массивами;*
Clock *– часы;*
Controls *– работа с «контролами», их тут два: кнопки и текстовые поля;*
Desktop *– рабочий стол компьютера;*
Dictionary *– встроенный словарь;*
File *– файловая система;*
Flickr *– сервис хранения картинок в Интернете;*
GraphicsWindow *– графическое окно программы;*
ImageList *– список изображений;*
Math *– математические операции;*
Mouse *– управление мышью;*
Network *– сеть;*
Program *– сама выполняемая программа;*
Shapes *– графические фигуры;*
Sound *– звуки;*
Stack *– стек (в других языках не встречал, кроме ассемблера);*
Text *– работа с текстом;*
TextWindow *– консольное окно программы;*
Timer *– таймер;*
Turtle *– «черепашка» для рисования.*

Как видно, набор весьма скуден, а подчас и не совсем понятен.

Во-первых, на рабочем столе компьютера можно только менять фоновое изображение. Очень «важная», конечно, операция! Доступны также текущие ширина и высота рабочего стола, но об этом позже.

Во-вторых, я не знаю людей, которые используют «***Flickr***», а ничего иного не предусмотрено. Впрочем, возможно, вам это как раз необходимо?

В-третьих, с помощью «***ImageList***» можно загрузить в память готовое изображение с диска или Интернета, потом его можно вывести в графическое окно программы. Но как сохранить в виде файла картинки то, что вы долго в этом окне рисовали – большой вопрос.

В-четвертых, объект «***Network***» позволяет только загрузить файл из сети или получить содержимое веб-страницы. Особого смысла в этом не вижу.

Что касается «черепашки» (аналогичной применяемой в языке Logo), то она как раз могла бы быть весьма полезна в некоторых частных случаях, мне даже жаль, что ее нет в других версиях Бейсика, хотя ее аналог существовал в ДОСовских реализациях языка (и там я его использовал широко!).

Объект «***Program***» позволяет получить параметры командной строки, указанные при запуске программы, позволяет принудительно завершить программу, а также получить папку запуска и приостановить работу программы на заданное время. Очень мало возможностей… Впрочем, подробнее об этом объекте мы поговорим в Главе 10.

Глава 4. Текстовое окно

Представлено объектом «***TextWindow***», создающим обычное консольное окно системы. Что тут возможно?

Можно выводить текст любым цветом на любом фоне в любую строку и с любого столбца. Можно перемещать само окно по экрану компьютера. Можно задержать выполнение программы до нажатия клавиши пользователем. Можно прочитать строку, введенную пользователем, причем ждать можно как символьную строку, так и чисто числовую.

А вот чего нельзя – так это получить событие по нажатию пользователем произвольной клавиши, то есть, нет опроса клавиатуры! Причем сам метод для этого есть, называется он «***ReadKey***», но он не работает – транслятор его ругает и считает несуществующим. Почему – непонятно.

К сожалению, отсутствие такой функции фактически сводит на нет возможность создания интерактивного консольного приложения на SB. Когда я в ходе изучения этого языка дополнял свой список исходников программами на SB, я не смог перевести на этот язык две свои очень полезные программы. Забегая вперед, скажу, что и в графическом окне данная функция реализована из рук вон плохо. Но об этом позже.

Практическое применение консольного окна сводится (как и советует сама Microsoft) в основном к отладке программ, распечатке промежуточных результатов и прочим вспомогательным операциям. Тут речи нет о написании программ, хоть отдаленно напоминающих, например, знаменитый FAR. Нет, я понимаю, что SB вообще не для этого создан, но зачем же так явно ограничивать возможности без всяких на то оснований? Ведь не сложно было бы сделать…

Тем не менее, опять же, я прекрасно понимаю, что консольные программы обычные «юзеры» вообще не любят и работать с ними не умеют. Консоль – это для специалиста. Так что начинающим программистам она тоже не так уж и нужна, как и большинству потребителей софта.

Среди написанных мной за долгие годы бесплатных программ есть, конечно же, и консольные приложения. Часть из них я все же перевел и под SB, некоторые примеры мы рассмотрим в данной книге.

Необходимо учитывать, что текстовое окно можно скрыть специальной командой, программа при этом будет продолжать свою работу, но вы не сможете получить к ней доступ, если сам код программы не предусматривает в дальнейшем показ этого окна. Эта же особенность относится и к графическому окну.

Правда, я не понимаю, зачем начинающему программисту нужно скрывать окно работающей программы (разве что для игры в хакера), но просто имейте это в виду. И - да! Если вы работаете в оболочке SB, остановить работу скрытой программы можно средствами самой оболочки. Но если вы запустили готовый exe-файл и окно программы пропало, – остановить программу можно только средствами ОС через менеджер задач. Так что будьте осторожны!

В архиве примеров есть код, показанный на обложке книги. Там как раз для вывода используется текстовое окно.

А теперь поговорим о графическом окне – оно явно применяется чаще и шире!

Глава 5. Графическое окно

«***GraphicsWindow***» – это то самое окно, которое мы видим в каждой запускаемой программе. Мы привыкли к тому, что в этом окне есть множество разных элементов: меню, пиктограммки, текстовые поля, надписи, изображения – да много всего! В окне программы на SB мы всего этого не увидим. Да, можно писать текст с любого пикселя и любым цветом. Да, можно рисовать линии, эллипсы и прямоугольники. А с помощью объекта «***Shapes***» можно делать много чего полезного. Все так! Но если действовать «в лоб», возможности SB весьма невелики.

Итак, как только вы производите какое-либо действие с графическим окном, оно появляется на экране компьютера. Как и в случае текстового окна (см. Главу 4) графическое тоже можно скрыть. Но все подводные камни и риски остаются такими же. Будьте осторожны! Лично мне скрытие окна так и не понадобилось ни разу.

В отличие от средств конструктора «Visual Studio», на SB все элементы окна нужно создавать программно. Кнопки и текстовые поля (а больше ничего и нет) должны быть созданы, помещены куда нужно, установлены их размеры, цвета и т.д.

Важная особенность! Вы не можете явно менять цвет кнопок или текстовых полей. Фон в них вообще нельзя изменить, а цвет текста меняется изменением цвета кисти графического окна («**GraphicsWindow.BrushColor=**CVET»). На что влияет изменение цвета фона окна, я так и не понял. Мне вообще кажется, что транслятор SB содержит ошибки или пока полностью не реализованные «закладки». Ну да ладно.

Меню создать нельзя. Конечно, это ограничивает функциональность программы, но тут я согласен с разработчиками языка: меню – элемент сложный и для многих задач излишний, начинающий программист на стадии обучения вполне может обойтись без него.

Для графического окна, помимо цвета, можно задавать много других свойств: размеры и положение на экране, размер, имя и вид используемого шрифта, ширину кисти. Можно запретить изменение размеров окна работающей программы. А вот события окна обрабатывать не так-то просто…

Собственно говоря, событий может быть всего 6: нажатие и отпускание кнопок мыши и ее перемещение, нажатие и отпускание клавиш, а также ввод текста. И если с клавишами и мышью все достаточно понятно и логично, то с вводом текста возникают совсем не очевидные странности.

Событие «**GraphicsWindow.TextInput**» возникает при вводе текста в графическом окне. Но «просто в окне» текст обычно никто не вводит: на VB имеется т.н. «inputbox», который создается в виде всплывающего окошка по запросу программы и ждет ввода текста. Но такого «бокса» на SB нет. Что делать?

У меня есть программа для тестирования скорости работы компьютера. Я написал ее в разных версиях Бейсика (для ДОС в том числе) и дает она весьма любопытные результаты. Но сейчас речь не о скорости компьютера, а о тонкостях программирования на SB.

Задача в том, что по нажатию определенной кнопки нужно ввести в программу некое число повторений. Так вот, процедура, которая обрабатывает событие ввода текста, имеет в этой программе следующий вид (код упрощен для иллюстрации и работать не будет, поэтому в архиве его нет):

```
Controls.ButtonClicked=KNOKLIK
GraphicsWindow.TextInput=INTEXT

Sub KNOKLIK
  KNOP=Controls.LastClickedButton
  If KNOP=VVOD Then
    GraphicsWindow.DrawText(10,15,"Введите число:")
    FLAG=1
    SS=""
  EndIf
EndSub

Sub INTEXT
  If FLAG<>0 Then
    SIM=GraphicsWindow.LastText
    KOD=Text.GetCharacterCode(SIM)
    If KOD=13 Then
      FLAG=0
      If SS<>"" Then
        Controls.SetButtonCaption(VVOD,"Число "+SS)
      EndIf
    ElseIf KOD>47 And KOD<58 Then
      SS=Text.Append(SS,SIM)
      GraphicsWindow.DrawText(120,15,SS)
    EndIf
  EndIf
EndSub
```

Что мы видим? Любые нажатия клавиш воспринимаются программой как ввод текста. Но нам нужен этот ввод только после нажатия кнопки с именем «VVOD». Нажатие кнопки создает событие «***ButtonClicked***», которое обрабатывается процедурой «KNOKLIK». Если нажата нужная нам кнопка, на экран выводится текст «Введите число:» и устанавливается флаг восприятия ввода («FLAG»), а также создается пока пустая текстовая переменная («SS») для приема и хранения введенного значения.

После чего любые нажатия клавиш обрабатываются процедурой «INTEXT». Что видим в ней? Если введена цифра, то ее символ добавляется к строке «SS» и выводится на экран. Если символ не является цифрой, то он игнорируется. И все это происходит до тех пор, пока не будет нажата клавиша «Enter».

Когда нажата – ввод завершен, флаг восприятия снимается, а на кнопке появляется введенное число. После этого процедура «INTEXT» будет продолжать вызываться при любых нажатиях клавиш, но флаг не установлен, так что никакой обработки не будет.

Я не уверен, что предложенный тут (и в самой программе) вариант обработки ввода чисел является оптимальным. Можно ведь вместо события ввода текста обрабатывать событие нажатия клавиш «**GraphicsWindow.KeyDown**». Программирование вообще обычно подразумевает множество вариантов решения одной и той же задачи. Так что при желании можете поэкспериментировать.

Но в любом случае отсутствие инструмента для запроса ввода строки – серьезный недостаток языка. Второй недостаток из этой же области – отсутствие

привычного для нас запроса «Да? Нет? Отмена» и любых его аналогов. Да, можно создать окошко с сообщением вроде «Файл не найден!», но кроме кнопки «Ok» вы там ничего не увидите, и это окошко никакое значение в программу не вернет.

К сожалению, данный механизм обработки ввода создает и другую серьезную проблему, с которой я столкнулся при попытке перевести на SB свой текстовый мини-редактор. Там вообще было много проблем (в силу чего полученная программа намного примитивнее своих аналогов на других языках), но нас сейчас интересует конкретная.

Связана эта проблема с тем, что в языке отсутствует возможность передачи фокуса на конкретный объект. Суть вопроса в том, что в окне программы был большой «текстбокс» для собственно ввода и редактирования текста (это ведь программа - текстовый редактор). Я сначала написал блок обработки нажатий клавиш в окне для того, чтобы по нажатию клавиши «F3» запрашивать ввод имени файла для загрузки в редактор. Так, как это у меня сделано в иноязычных версиях редактора. Но, увы!

При запуске программы все работает, аналогично приведенному выше примеру запроса числа повторений. Но если кликнуть мышью на текстбокс, то все нажимаемые клавиши будут обрабатываться им самим (ввод и редактирование текста). Можно нажать «F3», в служебной строке редактора появится запрос на ввод имени файла (как выше описано появление запроса на ввод числа). Но само имя файла введено уже не будет! То есть, все буквенные клавиши будут по-прежнему обрабатываться текстбоксом, и нет способа перевести фокус на само графическое окно!

В результате мне пришлось полностью поменять логику и оформление данной функции… Не скажу, что результат выглядит дико, возможно, в чем-то он даже вполне логичен, но факт в том, что он принципиально иной! И далеко не в каждом случае вообще возможна подобная переделка… При желании вы можете скачать с моего сайта данную программу и все увидеть подробно (в «Приложении» есть вся нужная информация).

Работа с мышью на SB тоже имеет свои особенности. Например, свойство «**Mouse.MouseX**» возвращает горизонтальную позицию мыши не относительно окна программы, а относительно всего экрана! Первый раз такое встречаю… Соответственно, для того, чтобы узнать позицию мыши в окне, вам нужно учитывать текущие координаты самого окна на экране компьютера. Конечно же, это касается и вертикальной позиции мышиного курсора. Курсор мыши можно скрывать и показывать снова, но изменить его форму невозможно.

В качестве примера работы с мышью подходит моя программа игры в «Крестики-нолики», см. «Приложение» в конце книги.

Глава 6. Работа с графикой

Что касается вывода самой графики, то тут возможности стандартные: поставить пиксель, прочитать пиксель, нарисовать линию, эллипс, прямоугольник… В случае линии надо задать координаты ее концов, но в случае прямоугольника задаются координаты его начала (верхней левый угол), а также ширина и высота, хотя привычнее задавать координаты его конца (правый нижний угол). Для рисования закрашенного прямоугольника есть отдельный метод.

А вот в случае эллипса все не так удобно, хотя, возможно, это дело привычки. Дело в том, что в прежних версиях Бейсика было принято задавать координаты центра, задавать радиус и задавать аспект, т.е. соотношение высоты и ширины. На SB, как и для прямоугольника, задаются координата верхнего левого угла и ширина с высотой прямоугольника, в который вписывается эллипс. Вроде так даже удобнее: параметров меньше. Но с точки зрения восприятия мне все же прежний вариант кажется более логичным. Не говоря о том, что мне пришлось сильно попотеть, переписывая под эту логику свою программу-заставку в виде вращающихся колец (подробнее в «Приложении»).

Только в одной из написанных на SB программ я применил объект «***Shapes***». Объект действительно удобен и, например, если вы захотите написать программу игры в тетрис, этот объект вам явно пригодится.

Если указанные выше методы рисования просто выводят графику в окно программы, то данный метод позволяет работать с графическими объектами как с отдельными элементами. Эллипсы, прямоугольники, треугольники, линии, готовые загруженные фотографии и даже тексты можно перемещать по экрану, поворачивать, прятать и т.д.

В качестве примера подходит, опять же, указанная выше программа «Крестики-нолики», больше я этот объект нигде не использовал пока…

Вывод текста в графическое окно возможен в двух вариантах.

1. С помощью указанного выше объекта «***Shapes***», если требуется иметь этот текст в виде отдельного объекта: «TEK=**Shapes.AddText**("добавляемый текст")». Потом этот текст по его имени «TEK» можно вращать, переносить (в том числе с определенной скоростью с помощью «анимации») и т.д.

2. Простым рисованием, после чего уже никакие операции с текстом будут невозможны, это будет просто рисунок. Однако, что удобно, тут есть два метода: «**GraphicsWindow.DrawText**» и «**GraphicsWindow.DrawBoundText**». В чем разница? Первый просто выводит строку текста в заданную позицию. Второй позволяет задать максимальную ширину выводимого текста, и в полученной рамке текст будет автоматически переноситься на новую строку. У меня это использовано в программе подбора цветов «Palit».

Кстати, именно данная программа показывает большинство возможностей и особенностей SB в плане работы с графикой. В «Приложении» есть более подробная информация.

Глава 7. Цвета

Цвет в графических операциях – едва ли не основное понятие. Как известно, на компьютерах цвет формируется из трех компонентов: красного, зеленого и синего, обозначаемых буквами R, G и B. Система цветов так и называется кратко – «RGB». Каждый из компонентов имеет обычно 256 оттенков от полного отсутствия (значение 0) до максимального 255. Все прочие цветовые оттенки формируются как сочетания главных компонентов разной интенсивности. Всего это дает более 16 миллионов оттенков, что для человеческого глаза вполне достаточно.

К сожалению, способ хранения значений цветов и даже формат их визуального представления в разных языках отличается. Иногда цвет можно указывать в десятичной форме вида «(R,G,B)» (например, для желтого «(255,255,0)»), иногда принята 16-ричная формула, многим непонятная: «#FFFF00» для того же желтого. Впрочем, раз уж вы изучаете язык программирования, системы счисления должны знать.

Но иногда цвет хранится в виде целого числа: для желтого это будет число «65535», для белого – «16777215».

Все эти форматы используются в разных случаях, но очень часто нужны преобразования между ними. В частности, последний (числовой) формат применяется в других версиях Бейсика, а 16-ричный формат – при создании вебсайтов.

На языке SB, к сожалению, привычный программисту числовой формат не используется. И когда я переводил свои программы на SB, мне пришлось немало потрудиться. Например, в языке есть функция, по сути такая же как в других версиях Бейсика: «**CVET=GraphicsWindow.GetColorFromRGB(R,G,B)**». Но в том же VB она возвращает целое число, а в SB – 16-ричную текстовую форму. Соответственно, при переводе программ возникают сложности. Да, все они решаемы, но время тратить приходилось.

Самая активная из моих программ в этом плане – программа подбора цветов «Palit», которую я когда-то сделал еще под MS-DOS, а потом переписал под Windows. Сейчас, когда я взялся за перевод ее с VB6 на SB1, переделка потребовалась кардинальная, и не только в формате цветов. Я думаю, что эта программа вообще будет очень полезным примером для обучения работе с графикой. Подробности в «Приложении».

К сожалению, на SB нет никаких встроенных инструментов для работы с цветами в виде чисел и для каких-либо преобразований цветов. Я понимаю, что язык этот обучающий и для создания серьезных программ не предназначен. Мне, например, пришлось эту программу сильно упрощать, убрав из нее блок работы с движками (которых тут нет) и с форматом цветов «HSV», который там был помимо «RGB».

Максимум, что еще есть в этой сфере, так это функция, возвращающая случайный цвет: «**GraphicsWindow.GetRandomColor()**». Я ее не использовал, но для создания каких-то узоров она может и пригодиться…

Глава 8. Массивы и математика

Я объединил эти понятия в одну главу, поскольку именно в математических расчетах чаще всего используются массивы, а, кроме того, просто нет смысла делить этот материал на отдельные главы.

Итак, что тут необычного? На мой взгляд, сделаны массивы не слишком удобно: вместо привычного «бейсиковского» оформления «BUK(J1,J2,J3)» используется «сишный» вариант «BUK[J1][J2][J3]». Когда я переводил некоторые свои программы на SB, это сильно раздражало. Кроме того, для работы с массивами тут используется специальный объект «***Array***».

Есть и некоторые плюсы по сравнению с прежними версиями Бейсика. На SB, к сожалению, нет ни пользовательских типов данных (их тут вообще нет, как я писал ранее), ни структур. Но зато индексом массива может быть не только число, но и строка:

```
A["имя"][1]="Петя"
A["имя"][2]="Вася"
A["имя"][3]="Миша"
A["имя"][4]="Коля"
A["фам"][1]="Николаев"
A["фам"][2]="Михайлов"
A["фам"][3]="Петров"
A["фам"][4]="Васильев"
B=Array.GetItemCount(A["имя"])
For I=1 To B
  TextWindow.WriteLine(A["фам"][I]+" "+A["имя"][I])
EndFor
TextWindow.WriteLine("")
```

Для того чтобы вывести в текстовое окно список всех участников базы, нужно знать их количество. Сколько элементов в массиве «A»? Ответ неоднозначен. С одной стороны, всего элементов 8. С другой, массив двумерный и по разным измерениям число элементов разное. Если задать «**B=Array.GetItemCount(A)**», то подсчет будет вестись по первому индексу (что не очевидно) и значение «B» будет равно 2. Поэтому нужно использовать проверку по тому индексу, где число элементов максимально.

В данном случае это очевидно, но если нужно написать реальную базу данных… Ну, а вот если про массив «A» вообще ничего не известно? Что делать?

Добавьте после кода выше такой фрагмент:

```
B=Array.GetAllIndices(A)
C=Array.GetItemCount(B)
For I=1 To C
  TextWindow.WriteLine(B[I])
  BB=Array.GetAllIndices(A[B[I]])
  CC=Array.GetItemCount(BB)
  For J=1 To CC
    TextWindow.WriteLine(BB[J])
  EndFor
EndFor
```

Что мы получим (эти примеры есть в архиве на сайте)?

Мы получим значение первого индекса первого столбца («имя») и список значений его второго столбца (1, 2, 3, 4), потом значение второго индекса первого столбца («фам») и список значений его второго столбца (те же самые 1, 2, 3, 4).

Разумеется, в настоящей программе все это не будет выводиться на экран как здесь, а будет использовано в виде значений переменных. Но в целом замечу, что для многомерного массива сложной структуры обработка будет выглядеть весьма замысловато. Впрочем, базы данных вообще требуют особого подхода и часто для их создания применяются специальные языки.

Среди моих бесплатных программ есть и очень полезная (а в чем-то и уникальная) база контактов, но сами понимаете, написана она не на SB, да и распространяется с закрытыми исходниками. Так что как сделана - не покажу, но пользоваться можете свободно: писал я ее для себя, но доступна и полезна она всем.

К сожалению, заявленные в подсказках оболочки методы «***GetValue***», «***SetValue***» и «***RemoveValue***» на практике не работают: транслятор считает, что таковых не существует. Но есть и плюс: можно присваивать массивы целиком. То есть, написав оператор «NEWA=A», вы получите массив «NEWA», эквивалентный массиву «A» (в моей «Эллочке» это тоже присутствует).

Можно еще проверить, включает ли данный массив какой-либо конкретный индекс. Например, метод «**B=Array.ContainsIndex(A,"имя")**» поместит в переменную «B» значение «True», а метод «**B=Array.ContainsIndex(A,"отч")**» поместит значение «False».

Пожалуй, про массивы все. Смотрите также пример в Главе 16 и программы из «Приложения».

А теперь коротко о математических функциях на «Small Basic», ибо подробно о них и сказать нечего.

Встроенная в язык математика основана на объекте «***Math***» и выглядит достаточно внушительно: тут есть прямые и обратные тригонометрические функции, десятичный и натуральный логарифм, преобразования угловых мер, вычисление абсолютного значения числа и генерация псевдослучайных чисел, а также разные виды перевода дробного числа в целое. Можно получить значение числа Пи, вычислять максимальное и минимальное значение из пары чисел, можно получить остаток от деления двух чисел и значение квадратного корня из числа. Можно возводить любое число в любую степень, но нет встроенной функции для вычисления экспоненты! Загадка…

Функции, возвращающей знак числа, тоже нет. Так что придется использовать пример из Главы 2.

Что касается точности вычислений, то ярким примером слабости SB может служить моя программа вычисления основания натуральных логарифмов (см. «Приложение»). Формально все функции возвращают очень много знаков мантиссы, но точность вычисления низкая. Кроме того, можете поэкспериментировать с указанной моей программой в плане увеличения ее точности: получите сообщения об ошибках…

Еще один серьезный момент – отсутствие плавающей запятой. Да, вы не можете работать с числами больше чем примерно 10 в 29-ой степени. Ну и меньше чем 10 в минус 29-ой…

Глава 9. Работа со строками

Строки – важнейший элемент практически любой программы. И от того, насколько развита в языке поддержка работы со строками, во-многом зависит вообще спектр применения этого языка. В случае SB тут не все идеально, хотя и не плохо. Для работы с текстом имеется специальный объект «***Text***», к которому применимы следующие методы:

Append – сложение двух строк в одну;
ConvertToLowerCase – преобразование в строчные буквы;
ConvertToUpperCase – преобразование в прописные буквы;
EndsWith – проверка, является ли заданный текст концом строки;
GetCharacter – возврат символа по его юникоду;
GetCharacterCode – возврат юникода заданного символа;
GetIndexOf – возвращает позицию начала указанной подстроки;
GetLength – возвращает длину строки;
GetSubText – возвращает подстроку заданной длины с заданной позиции;
GetSubTextToEnd – возвращает всю строку, начиная с данной позиции;
IsSubText – проверяет, входит ли заданный текст в строку;
StartsWith – проверяет, является ли заданный текст началом строки.

Первый метод мы видели в Главе 2, когда пытались складывать строковые переменные. Да, вместо привычного простого «+» на SB надо выполнять «***Append***». Второй и третий метод очевидны: строка переводится в строчные или прописные буквы. Методы 4 и 12 аналогичны друг другу и тоже вполне понятны.

Да в общем тут понятно все! Но есть тонкости с кодами. Надо помнить, что «SmallBasic» работает только с юникодами, точнее, с кодировкой «UTF-8». Поэтому для кодов 0-127 метод «***GetCharacter***» вернет обычные ASCII-символы, для кодов 128-255 будут выданы стандартные символы в кодировке по-умолчанию, а вовсе не в той, что используется вашей системой (для русского языка – «Win-1251»).

Соответственно, если вы хотите получить русские символы, используйте диапазон юникода (для букв А-Я коды 1040-1071 и т.д.).

Еще одна тонкость со строками напрямую не связана, но тоже касается символов и их кодов. Если вы переходите на SB с более развитых языков (со мной вот именно так и было), то коды нажимаемых клавиш примерно знаете. Но на SB все будет иначе!

Еще много лет назад я написал программу, которая показывает все коды нажимаемых клавиш. Для DOS и Windows коды были немного разные, но, хотя «SmallBasic» работает тоже под Windows, у него система кодов почему-то несколько своеобразна. Поэтому я прежде всего сделал версию этой программы на SB, в «Приложении» об этом сказано более подробно, а код программы есть на моем сайте. Там же можно скачать версии этой программы на «нормальных» языках и сравнить результаты. Зачем убрана совместимость – понять не могу! Но знать об этом нужно.

Остальные методы особых вопросов и сложностей у меня не вызвали. И для основных текстовых операций предложенного набора вполне хватает.

Глава 10. Работа с файлами

Наконец-то мы добрались до файлов! Что ни говори, а любая работа на компьютере начинается и заканчивается файловыми операциями. Какие возможности у вас будут при программировании на SB с использованием объекта «***File***» (ниже будут подробные комментарии)?

LastError – описание последней ошибки;
AppendContents – добавить содержимое в файл;
CopyFile – скопировать файл;
CreateDirectory – создать папку;
DeleteDirectory – удалить папку;
DeleteFile – удалить файл;
GetDirectories – получить список всех папок внутри указанной;
GetFiles – получить список всех файлов в заданной папке;
GetSettingsFilePath – возврат имени файла настроек;
GetTemporaryFilePath – возврат пути к временному файлу;
InsertLine – вставить строку в файл;
ReadContents – прочитать содержимое файла;
ReadLine – прочитать строку из файла;
WriteContents – записать в файл новое содержимое;
WriteLine – записать строку в файл.

Да, почти все эти методы и свойства требуют пояснений.

Прежде всего (я об этом уже говорил ранее) надо помнить, что SB работает только с текстовыми файлами! И не имеет значения, что там в файле находится на самом деле: язык и его методы любой обрабатываемый файл считают текстовым. Естественно, что такое ограничение лишает язык статуса универсального и вообще лишает его полноценной возможности применения. Да, я написал на нем больше десятка простых (но полезных!) программ, однако я не смог сделать его тем самым повседневным языком работы, о котором писал в предисловии к книге. А главное, зачем это ограничение? Ведь добавить в язык операции с двоичными файлами совсем не сложно…

Но что есть – то есть. А что именно есть? Есть, увы, серьезные ограничения даже для текстовых файлов.

Во-первых, все файлы воспринимаются только в кодировках юникод. В «Приложении» рассказано о моем текстовом редакторе, там прочтите подробнее. Во-вторых, нет метода для переименования файла. Собственно говоря, я с этой задачи и начал знакомство с SB. Мне нужно было переименовать сотню файлов с одним алгоритмом формирования имен на имена по другому алгоритму. Делать это вручную – нереально. А как? Логично написать программу, которая меняет одно имя на другое. Задачу я на SB решил, но как? Копировал файл со старым именем в файл с новым именем, а потом удалял старый. И на том спасибо, хотя, конечно, странновато…

В-третьих, метод «***GetFiles***» возвращает список всех файлов в заданной папке. А если нужны файлы только определенного типа? Увы, никакие фильтры и варианты задать нельзя. Что тоже сильно ограничивает применение языка.

Отдельно стоит остановиться на экзотических методах «***GetSettingsFilePath***» и «***GetTemporaryFilePath***». Да, сложные программы иногда пользуются файлами конфигурации. И по идее первый метод полезен для автоматического формирования имени этого файла для последующей работы с ним. Но, во-первых, формируемое имя не совсем стандартное (файл имеет необычное расширение «settings»), а во-вторых файл получится текстовый (с другими язык не работает), что для файла конфигурации далеко не всегда удобно. Да и не годится SB для столь сложных программ, чтобы требовался подобный файл… Но метод зачем-то сделали.

Второй метод возвращает имя автоматически созданного временного файла. Тут аналогично: такие файлы нужны сложным программам, да и то редко. И зачем это сделано на SB, мне непонятно.

Но самый главный момент – это обработка файловых ошибок. Многие из перечисленных выше методов возвращают определенное значение, равное либо «SUCCESS», либо «FAILED». По идее, если его проверить, то можно понять, успешно ли выполнена операция. Но если не успешно, то почему? Для этого служит свойство «***LastError***», распечатав которое, можно понять в чем проблема. Но…

Рассмотрим такой пример (с номерами строк для удобства, есть в архиве):

```
01 FIL="a:\vrem.dat"
02 A=File.ReadLine(FIL,2)
03 B=File.LastError
04 TextWindow.WriteLine(A)
05 TextWindow.WriteLine(B)
06 TextWindow.WriteLine("")
07 SS="бред сивой кобылы"
08 AA=File.AppendContents(FIL,SS)
09 If AA="FAILED" Then
10   BB=File.LastError
11   GraphicsWindow.ShowMessage(BB,"Ошибка")
12 EndIf
```

Ни диска «a», ни файла «vrem.dat» на компьютере нет. Что мы получим в результате работы программы?

В строке 02 происходит чтение второй строки из несуществующего файла. По идее, мы должны получить ошибку. Но метод возвращает только содержимое строки, без вариантов. Хорошо, пусть так! Строка, естественно, возвращается пустой. Но ведь явно была ошибка! В строке 03 мы должны получить сообщение типа «файл не найден». Но переменная «B» так же пуста, как и переменная «A»! Это показывают строки 04 и 05. Таким образом, ваша программа будет работать как ни в чем не бывало, а вы и знать не будете, что в данных ошибка!

Дальше (в строке 08) мы пытаемся добавить в этот же файл новое содержимое. И вот тут наконец-то ошибка видна! В строке 11 создается всплывающее сообщение с текстом «Не удалось найти часть пути “a:\vrem.dat”».

Что из всего этого следует? Следует то, что если метод возвращает флаг успешности, то можно и саму ошибку обнаружить, и причину ее узнать. А вот если метод флаг не возвращает (как в строке 02), то отловить ошибку будет весьма проблематично.

Кстати, на других версиях Бейсика (в том же VB) вместо текста ошибки возвращается ее номер, что позволяет программисту самому создавать диагностические сообщения для пользвателя программы, а не использовать готовые, как здесь. Понятно, что для обучающего языка и это сойдет, но то, что не

все методы вообще позволяют отловить факт ошибки – это крайне плохо, особенно как раз для начинающего программиста. Так что учтите и будьте внимательны!

Думаю, что здесь же стоит снова упомянуть объект «***Program***», обозначающий запущенную и выполняющуюся программу на SB. Как я уже писал ранее, этот объект имеет всего 2 свойства и три метода. Какие?

ArgumentCount – число параметров командной строки;
Directory – папка с программой;
Delay – задержка работы программы;
End – завершение работы программы;
GetArgument – получение списка параметров.

Очень многие программы нуждаются в параметрах командной строки при их вызове. На формат этой строки параметров нет никаких стандартов, поэтому в других версиях Бейсика вся строка передается в программу целиком, а дальше уже сама программа по заданному алгоритму смотрит, что ей там передали.

На SB применяется жесткий формат, в котором можно задавать различные параметры строки, разделяемые пробелами. Число таких «блоков» возвращает свойство «***ArgumentCount***», а сами параметры возвращаются методом «***GetArgument(K)***» где K – номер параметра. Отчасти такой формат упрощает работу с командной строкой, но зато требует жесткой ее структуры.

Свойство «***Directory***» возвращает папку, где расположен запущенный exe-файл программы, но имя самого этого файла получить невозможно! Да, оно редко нужно, но иногда это критичный фактор! Непонятно, почему было не сделать и свойство, например, «***FileName***» с таким значением?

Задержка работы программы нужна, наверно, только в целях отладки, потому что в иных ситуациях разумнее использовать таймер, часы и прочие встроенные в язык средства.

Метод «**Program.End()**» вызывает немедленное завершение работы программы, но нет никакого способа сообщить об этом факте самой программе! То есть, при закрытии программы этим оператором или крестиком в углу окна все явно не сохраненные ранее результаты работы будут просто потеряны. Даже если пользватель случайно нажал этот злополучный крестик. Опять же, все другие версии Бейсика предусматривают программный перехват завершения работы и последующее совершение любых действий вплоть до отмены завершения программы.

Я понимаю, что для обучения можно обойтись и без всего этого сервиса. Но не понимаю, зачем нужно искусственно ограничивать возможности, реализовать которые никакого труда не представляет. Впрочем, я допускаю, что перечисленные мной в этой книге недостатки языка критичны далеко не для всех авторов программ…

Глава 11. Работа с временем

Часы представлены объектом «***Clock***», а еще есть объект «***Timer***». Что они позволяют делать и зачем вообще нужны? Объект «***Clock***» не имеет методов, но имеет следующие свойства:

Date – системная дата в формате «dd/mm/yyyy»;
Day – текущее число месяца;
ElapsedMilliseconds – сколько миллисекунд прошло с 1900 года;
Hour – час;
Millisecond – миллисекунды;
Minute – минуты;
Month – текущий месяц (число);
Second – секунды;
Time – системное время в формате «hh:mm:ss»;
WeekDay – день недели в виде строки на языке вашей системы;
Year – текущий год (число).

Разумеется, все эти свойства можно только читать, изменить их значения невозможно.

Свойство «***ElapsedMilliseconds***» мы упоминали в Главе 2, демонстрируя работу цикла «While». Собственно говоря, никаких иных ему применений и не видно.

Но для работы с временем на SB есть еще один объект - «***Timer***», имеющий следующие возможности:

Interval – интервал срабатывания в миллисекундах (от 10 до 100000000);
Pause – приостановка работы таймера;
Resume – возобновление работы таймера;
Tick – событие срабатывания таймера.

В принципе, тут все просто: задаете нужный интервал, запускаете таймер в работу и через заданный интервал времени отрабатываете нужный код в процедуре, привязанной к событию «***Tick***».

Из тех моих программ, что написаны на SB и представлены на моем сайте, таймер я использовал в двух: это уже упомянутые ранее программа-заставка в виде вращающихся колец и программа для тестирования скорости работы компьютера. Подробнее эти программы описаны в «Приложении».

Разумеется, при работе с таймером (и не только на SB) надо избегать ситуации, когда установлен слишком маленький интервал, а процедура обработки события сложна и «неповоротлива». По идее во время обработки события работа таймера автоматически приостанавливается, но так ли это на самом деле, я не проверял. Если же он продолжает работать, то в указанном случае возможно наложение событий и в итоге аварийное завершение программы. Если же таймер отключается, то программа просто начнет «отставать», т.е. срабатывания таймера будут не через заданные (допустим) 20 мс, а через 30 или 40. Но, учитывая быстродействие современных компьютеров, такой конфуз, конечно, маловероятен. Но помнить о нем стоит!

Глава 12. Стек

Объект «***Stack***» имеет три метода: «***GetCount***», «***PopValue***» и «***PushValue***». Первый возвращает объем стека, второй извлекает из него значение. Третий – помещает значение в стек.

Что такое вообще «стек»? Это достаточно редкое понятие в жизни, но крайне необходимое в программировании. Даже если вы не пользуетесь стеком напрямую, вы должны понимать, что это такое. Например, я ни в одной из своих программ на SB этот объект не использовал, а на других версиях Бейсика его просто нет. Но когда я писал интерпретатор своего языка «Эллочка» (я упоминал его в Предисловии), мне пришлось создавать стек самостоятельно. Итак, зачем и что это такое?

Есть два основных понятия организации хранения данных: «стек» и «очередь». Что такое очередь, знают все: «первым встал – первым обслужили». В любом магазине, на АЗС и т.д. Стек организован иначе: «первым встал – обслужили последним». В быту это кажется несправедливым, но когда вы кладете в шкаф мытые тарелки, вы кладете их стопкой. И ту тарелку, которую вы положили последней, захотев кушать, вы возьмете первой. Потом вторую (жена поест). Потом вы их помоете и положите обратно. Потом возьмете снова.

В результате та тарелка, которая была положена в шкаф первой и лежит теперь в самом низу стопки, ждать своего часа может неделями. Впрочем, я не уверен, что она будет этим недовольна.

Зачем же стек в программировании? Во-первых, вы же знаете, что такое «процедура»? Это часть кода, имеющая свое имя и выполняемая по специальному вызову, когда нужно. Но то место программы, из которого она вызвана, будет помещено в стек. Потому что из этой процедуры может быть вызвана следующая, а из нее – третья. И каждый раз место вызова будт добавляться в стек, чтобы после выполнения процедуры программа знала, откуда продолжать вычисления.

Да, все это делается без вашего участия, автоматически. Но понимать это нужно, потому что нарушение порядка может вызвать необратимые последствия. При программировании на SB это вряд ли возможно, тут просто нет инструмента для нарушения порядка. Но вы же только начинаете программировать. А в других языках такие инструменты могут и присутствовать (как были на Бейсике для ДОС, создавая этим дополнительные возможности)…

Во-вторых, выше я сказал, что создавал стек сам. Зачем? Мой язык подразумевал и вычисление по формуле, в том числе и со скобками. Разумеется, я не буду тут подробно описывать работу транслятора того языка, я просто привожу пример: без стека там не обойтись. Но вы, если интересно, можете сами придумать и решить задачу с использованием стека. Я думаю (да и мой собственный опыт об этом говорит), что если человек увлечен программированием, то он всегда найдет себе задачу, а не найдет – так придумает. А если не увлечен – то стоит ли вообще заниматься программированием? Впрочем, заниматься в жизни вообще нужно тем, что интересно и приятно…

Глава 13. Звук

Работа со звуком производится с помощью объекта «***Sound***». К данному объекту применимы следующие методы:

Pause – приостановка воспроизведения;
Play – воспроизведение файла в формате «mp3», «wav» или «wma»;
PlayAndWait – воспроизвести и ждать окнчания (и возобновить с паузы);
PlayBellRing – воспроизвести звук звонка;
PlayBellRingAndWait – воспроизвести и ждать окончания;
PlayChime – воспроизвести звук колокола;
PlayChimeAndWait – воспроизвести и ждать окончания;
PlayChimes – воспроизвести звук колоколов;
PlayChimesAndWait – воспроизвести и ждать окончания;
PlayClick – воспроизвести звук клика мыши;
PlayClickAndWait – воспроизвести и ждать окончания;
PlayMusic – воспроизвести последовательность нот;
Stop – остановить воспроизведение файла.

Все методы, имеющие в своем составе «***AndWait***» томозят работу программы до тех пор, пока не будет закончено воспроизведение указанного звука. Методы без данного добавления к имени воспроизводят звук в фоновом режиме. Например, сделаем такой код (есть в архиве):

```
Sound.Play("C:\LANG\SmallBasic1\MOI\raznoe\primery\wiva.wma")
For I= 0 To 100000
  GraphicsWindow.PenColor=GraphicsWindow.GetRandomColor()
  X=math.GetRandomNumber(GraphicsWindow.Width)
  Y=math.GetRandomNumber(GraphicsWindow.Height)
  GraphicsWindow.DrawEllipse(X,Y,3,3)
EndFor
```

Будет играть музыка и одновременно будут рисоваться разноцветные кружки в графическом окне программы. Если же вместо «**Sound.Play**» использовать метод «**Sound.PlayAndWait**», то пока музыка не кончится, графическое окно даже не появится на экране компьютера.

И обратите внимание, что в операторе «**Sound.Play**» надо указывать полный путь к файлу! Даже если он находится в одной папке с программой. Загадка, но так.

Что касается метода «***PlayMusic***», то он как раз является преимуществом SB перед другими Бейсиками для Windows (VB любых версий) и делает то, что делал когда-то Бейсик для MS-DOS. Мне непонятно, почему на VB нет такой возможности!

Так вот, этот метод запускает процесс проигрывания заданной нотами мелодии. Мелодия задается на специальном языке «Music Markup Language», но сам язык мы тут, конечно, рассматривать не будем. Думаю, если он вам понадобится, вы найдете его описание. Но, конечно, с появлением современных звуковых форматов необходимость в этом языке практически пропала. Тем удивительнее, что его убрали из VB даже очень старых версий (где он как раз мог пригодиться), но затем добавили в SB, где его никто использовать не станет. Чудны дела твои, Microsoft!

В целом же работа со звуком на SB реализована вполне достаточно для обычного пользователя. Я редко пользуюсь звуком в программах, так что обошелся без него. Возможно потому не пользуюсь, что динамик у меня всегда отключен…

Глава 14. Черепашка

Объект «***Turtle***» тоже является уникальным для SB инструментом и на том же VB не используется. Хотя, как и указанный в Главе 13 инструмент игры по нотам, рисование фигур передвижением указателя было реализвано в Бейсике для MS-DOS и было там крайне полезно. И должен сказать, что в среде Windows мне этого инструмента тоже сильно не хватает иногда. Но увы! Я пишу на VB, где его нет, а на SB серьезную программу не напишешь. Вот такое противоречие!

«Черепашка» обладает следующими свойствами и методами:

Angle – текущий угол движения черепашки;
Speed – задать скорость движения черепашки (от 1 до 10);
X – горизонтальная координата черепашки;
Y – вертикальная координата черепашки;
Hide – спрятать черепашку (запретить операции);
Move – переместить черепашку на указанное расстояние;
MoveTo – переместить черепашку в заданную точку;
PenDown – включить перо черепашки;
PenUp – отключить перо черепашки;
Show – показать черепашку (разрешить операции);
Turn – повернуть черепашку на заданный угол;
TurnLeft – повернуть влево на 90 градусов;
TurnRight – повернуть вправо на 90 градусов.

Если вы пишете игровую программу и вам нужно управлять объектом игры, или если хотите создать красивый узор, то вполне можете сделать все это с помощью черепашки. Я не использовал ее в своих программах на SB, но вам никто не мешает это сделать!

Как видно, управлять черепашкой не сложно и возможности у нее большие. Управление цветом производится так же, как и для других графических операций.

Вот небольшой пример ее работы (есть в архиве).

```
GraphicsWindow.Title="Черепашка рисует звезду"
GraphicsWindow.PenColor="red"
Turtle.PenUp()
Turtle.MoveTo(GraphicsWindow.Width/2,GraphicsWindow.Height/2)
Turtle.PenDown()
Turtle.Speed=6
Turtle.Angle=18
For I=1 To 5
  Turtle.Move(100)
  Turtle.Turn(144)
EndFor
Turtle.Hide()
```

Мне кажется, тут все ясно без пояснений. В качестве упражнения попробуйте сами разобраться, что делает каждый оператор, попробуйте модифицировать программу, чтобы каждый луч звезды был нарисован другим цветом.

Но в целом, для решения практически задач «Черепашка», скорее всего, вам не понадобится…

Глава 15. Оставшееся

Последние объекты, которые мы должны рассмотреть, это «***Dictionary***», «***Flickr***», «***Desktop***» и «***Network***».

Объект «***Dictionary***» («Словарь») работает только с английским языком, посему ни мне, ни читателям этой книги он не понадобится. Да и вообще не совсем понятно, какой смысл программно получать из интернета толкование слов? Если кто-то захочет писать лингвистическую программу, он все равно будет это делать не на SB…

Аналогично, мне не ясно назначение объекта «***Flickr***». Объект возвращает интернет-адрес изображения по заданным тегам или «текущего», представленного на данном сервисе. Даже если кто-то пользуется данным сайтом, то зачем нужно в своей программе грузить оттуда фактически случайные картинки – не понимаю. Впрочем, если хотите – можете пользоваться, никаких «подводных камней» там нет.

Ну и третий малопонятный объект - «***Network***». Конечно, пользы в нем больше, чем в двух предыдущих. Он позволяет либо получить полное содержание заданной веб-страницы, либо скачать файл по заданному адресу. Все бы хорошо, но мне не удалось выполнить эти операции применительно к своим собственным сайтам. Ни содержимое страниц, ни файлы (разных типов) я так и не получил. То ли объект не работает, то ли он работает только с определенными сайтами – не знаю.

Поскольку (как я писал ранее) в SB фактически отсутствует нормальная диагностика ошибок, никаких сообщений от системы я не получил – просто возвращалась пустая строка. Так что ничего вразумительного по данному оператору сообщить не могу.

Кстати, аналогичная картина была и с упомянутыми тут другими странными объектами. И если сервис «***Flickr***», похоже, просто недоступен без авторизации на сайте, то что за словарь использует объект «***Dictionary***» вообще нигде не сказано.

Так что все эти три объекта опробовать на практике не получилось. Они не нужны, для меня вопрос был только в принципе. Но увы.

Что касается объекта «***Desktop***», то максимум что можно – это изменить фон десктопа. Но есть два полезных момента: вы можете получить ширину и высоту рабочего стола. Зачем это нужно? Ну, прежде всего это нужно для правильного позиционирования окна вашей программы. На «Visual Basic» можно изначально установить, где будет располагаться окно программы. На SB это невозможно, в силу чего при старте программа будет открыта «где попало». А хороший тон – открывать ее в центре экрана. Где центр? Тут и пригодятся размеры рабочего стола. Да и сам размер окна программы, возможно, придется скорректировать для маленького экрана нетбука. Так что это полезные свойства!

А все остальное мы рассмотрели. Что сказать? Язык «Small Basic» имеет, конечно, свои плюсы, особенно в плане обучения (для которого и предназначен). Но имеет и существенные минусы, в том числе как раз и в сфере обучения программировнию. Можно ли писать на нем что-то реально полезное, если никаких других языков вы не знаете, а задачу решить нужно?

Об этом расказано далее. Как говорится, «Не переключайтесь!».

Глава 16. Практический пример

Попробуем решить практическую задачу. Да, ее можно решить на «Экселе». Но, во-первых, я его плохо знаю и для собственных нужд предпочитаю писать свои программы, а во-вторых, данная книга – пособие по «Small Basic», а не по офисным приложениям. Так что давайте-ка решим задачу, создав соответствующую программу на SB.

Что за задача? Нужно построить гистограмму годовых изменений процентного состава некой мультивалютной корзины. Названия валют, годы и сами проценты будут храниться в текстовом файле в виде таблицы, значения в строках разделены символом табуляции.

Вот этот файл:

```
year	AZN	USD	EUR	GBP	RUR	CHF	XAU	SUM
14	25	23	21	20	11	0	0	81
14.5	25	23	21	20	11	0	0	81
15	0	43	23	27	7	0	0	98
15.5	0	56	19	20	5	0	0	176
16	5	50	18	22	5	0	0	190
16.5	2	52	18	22	6	0	0	220
17	5	48	19	21	7	0	0	211
17.5	5	47	19	22	7	0	0	216
18	5	49	19	21	6	0	0	210
18.5	5	49	19	21	6	0	0	208
19	5	50	18	20	7	0	0	203
19.5	5	49	18	21	7	0	0	207
20	4	28	32	17	4	15	0	246
20.5	4	27	32	18	4	15	0	256
21	4	27	32	18	4	15	0	252
21.5	4	25	29	17	4	14	7	268
22	3	20	21	24	4	11	17	336
```

Как видно, некоторые значения равны нулю, то есть на гистограмме они видны не будут. Так же видно, что годы указаны с дробными частями, ибо значения вносились в таблицу раз в полгода. Но мы не будем писать дроби для лет, мы просто добавим 2000 к каждому году (для привычного представления) и уберем дробную часть. То есть, для почти всех лет на графике будут два значения. Последнее значение (SUM) – это эквивалентная сумма, показывающая общий размер корзины. Эти значения будут выведены в виде графика поверх гистограммы.

Теперь нам надо писать программу. Сначала я хотел сделать ее фиксированной для отображения данной конкретной таблицы. Но потом подумал, что это неправильный подход: надо сразу учиться делать программы универсальными. Сегодня одна таблица с данными будет, а завтра другая – и что, все переделывать? Конечно, нет. Да, это сложнее, но думаю, вы разберетесь.

Однако, окно мы сделаем фиксированного размера в расчете на маленький экран ноутбука. И менять его размер мы тоже запретим, чтобы не заморачиваться с перерисовкой содержимого. Кстати, SB все равно не позволяет отследить событие изменения размеров окна программы…

Итак, мы назовем наш исходный файл «grafdat.sb» и далее разберем его по частям подробно. Сразу скажу, что исходный код всей этой программы можно скачать на странице данной книги в составе архива с другими фрагментами, разобранными в предыдущих главах.

Вот собственно код самой программы:

```
GraphicsWindow.Title="Вывод графиков по табличным данным"
GraphicsWindow.Width=1020
GraphicsWindow.Height=720
GraphicsWindow.CanResize="False"
GraphicsWindow.Left=(Desktop.Width-GraphicsWindow.Width)/2
GraphicsWindow.Top=(Desktop.Height-GraphicsWindow.Height)/2
DEL = Text.GetCharacter(9)
FIL = "fin.dat" ' файл с исходными данными
X0 = 30 ' координата точки отсчета
Y0 = 640 ' координата точки отсчета
VYSOTA = 600 ' максимальная высота графика
WIRINA = GraphicsWindow.Width - 2*X0 ' максимальная ширина графика
COLSUM = GraphicsWindow.GetColorFromRGB(0 , 0 , 255) ' цвет суммы
COLLIN = "#000000" ' цвет шкалы
READDATA()
PODGOTOVKA()
GRAFIK()
```

Мы создаем окно размером 1020*720 пикселей и запрещаем менять эти размеры. Далее мы помещаем окно программы в центр экрана. А дальше мы указываем значения исходных переменных и вызываем подряд три процедуры. Что это за процедуры?

Первая процедура «READDATA» читает данные из файла и заполняет массивы:

```
Sub READDATA ' считывание данных из файла
  J=0
  While J>=0
    J=J+1
    STRO=File.ReadLine(FIL,J)
    If STRO="" Then
      Goto konec
    EndIf
    RAZBOR()
    DAT[J]=MAS
  EndWhile
konec:
  KOL=Array.GetItemCount(DAT[1])-1 ' количество валют +1
  GOD=J-1 ' количество лет +1
EndSub
```

Как видно, тут на самом деле используются две процедуры. Формально в этом не было никакой нужды, потому что процедура «RAZBOR» (показана ниже), предназначенная для анализа и разбора одной строки данных, вызывается в процедуре «READDATA» всего один раз (в цикле). И поэтому код разбора вполне можно было поместить внутрь процедуры чтения. Но я, когда начинал писать, еще не был уверен в «одноразовости» вызова, да и всегда пролезнее разбивать код на самостоятельные блоки.

Обратите внимание на строки после метки «konec»: из-за того, что в таблице содержатся не только значения валют, но и другие параметры, размеры массивов

не соответствуют ожидаемым. Можно, конечно, создать отдельные массивы на года, имена валют и суммы, но я этого делать не стал.

```
Sub RAZBOR ' разбор строки данных
  I=0
  While STRO<>""
    Q=Text.GetIndexOf(STRO,DEL)
    If Q=0 Then
      Q=Text.GetLength(STRO)+1
    EndIf
    ZNA=Text.GetSubText(STRO,1,Q-1)
    I=I+1
    MAS[I]=ZNA
    STRO=Text.GetSubTextToEnd(STRO,Q+1)
  EndWhile
EndSub
```

Что тут происходит? Каждая строка представляется в виде одномерного массива «MAS», который, в свою очередь, становится частью двумерного массива «DAT». Тут как раз используется возможность SB присваивать части массивов друг другу. В других языках пришлось бы писать отдельные циклы…

Теперь на основе полученных данных нужно рассчитать значения по горизонтальной и вертикальной шкале. Делает это процедура «PODGOTOVKA».

```
Sub PODGOTOVKA
  ' ----- вычисление ширины столбца одной валюты -----
  WIRKOL= Math.Floor(WIRINA/(GOD-1)/(KOL+1))
  ' ----- создание цветов для валют и вывод их имен -----
  For J=2 To KOL
    R=204*Math.Abs(Math.Sin(J*2))
    G=204*Math.Abs(Math.Sin(J*5))
    B=204*Math.Abs(Math.Sin(J*7))
    DAT[GOD+1][J]=GraphicsWindow.GetColorFromRGB(R , G , B)
  EndFor
  GraphicsWindow.BrushColor=COLLIN
  Y=Y0+48
  GraphicsWindow.DrawText(X0 , Y , "Цвета валют:")
  For J=2 To KOL
    X=X0+J*60
    GraphicsWindow.BrushColor=DAT[GOD+1][J]
    GraphicsWindow.DrawText(X , Y , DAT[1][J])
  EndFor
  MAX1=0 ' ----- вычисление экстемальных значений -----
  MAX2=0
  MIN2=100000
  For I=2 to GOD
    If MAX2<DAT[I][KOL+1] Then
      MAX2=DAT[I][KOL+1]
    EndIf
    If MIN2>DAT[I][KOL+1] Then
      MIN2=DAT[I][KOL+1]
    EndIf
    For J=2 To KOL
      If MAX1<DAT[I][J] Then
        MAX1=DAT[I][J]
      EndIf
    EndFor
  EndFor
```

```
  MAX1=Math.Ceiling(MAX1/10)*10 ' (округлено кратно 10)
  MAX2=Math.Ceiling(MAX2/10)*10 ' (округлено кратно 10)
  MIN2=Math.Floor(MIN2/10)*10 ' (округлено кратно 10)
  ' ----- вывод координатных осей и их градуировка -----
  XX=1.5*X0+WIRKOL*(KOL+1)*(GOD-1)
  YY=Y0-VYSOTA
  GraphicsWindow.BrushColor=COLLIN
  GraphicsWindow.DrawLine(X0 , Y0 , XX , Y0)
  GraphicsWindow.DrawLine(X0 , Y0 , X0 , YY)
  GraphicsWindow.DrawLine(XX , Y0 , XX , YY)
  DLI=MAX1/5 ' число отметок левой вертикальной шкалы (с шагом 5%)
  For I=1 To DLI ' вывод значений процентов по левой вертикальной оси
    GraphicsWindow.DrawText(X0-24 , Y0-VYSOTA/DLI*I , I*5)
  EndFor
  GraphicsWindow.DrawText(X0-24 , Y0-VYSOTA-20 , "Процент")
  For I=2 To GOD ' вывод значений лет по горизонтальной оси
    X=X0*1.5+WIRKOL*(KOL+1)*(I-2)
    S=2000+Math.Floor(DAT[I][1])
    GraphicsWindow.DrawText(X , Y0+12 , S)
  EndFor
  DLI=(MAX2-MIN2)/10 ' число отметок правой верт. шкалы (с шагом 10) - 1
  GraphicsWindow.BrushColor=COLSUM
  GraphicsWindow.PenColor=COLLIN
  For I=0 To DLI ' вывод значений сумм по правой вертикальной оси
    GraphicsWindow.DrawText(XX+8 , Y0-I/DLI*VYSOTA-6 , I*10+MIN2)
    GraphicsWindow.DrawLine(XX , Y0-I/DLI*VYSOTA , XX+5 , Y0-I/DLI*VYSOTA)
  EndFor
  GraphicsWindow.DrawText(XX , Y0-VYSOTA-25 , "Сумма")
EndSub
```

Как видно, она большая, и ее код никак не поместится на одну страницу.

Сначала на основе максимальной ширины графика и количества валют и лет из файла данных вычисляется ширина столбца для отображения одной валюты. А потом стоит блок подготовки цветов для вывода этих столбцов. И на нем надо остановиться подробнее.

Вначале я было решил выводить валюты случайными цветами, ибо это очень просто сделать. Но увы – каждый раз цвета были разными и каждый раз сочетания их были неудачными. Пришлось придумывать какой-то иной вариант.

Самый простой путь – это просто «в лоб» задать цвета для каждой валюты, мы ведь знаем, сколько их? Но я уже писал выше, что программа должна быть универсальной. А значит, цвета должны как-то вычисляться. Я перебрал разные варианты и то, что вы тут видите, это оптимальный. Точнее, мне просто надоело выискивать что-то лучше по принципу «лучшее – враг хорошего». Но вы можете сделать свой алгоритм подбора цветов!

Далее стоит блок масштабирования по вертикальной шкале. Мы не знаем по идее, какие в файле значения и просто с учетом максимальной высоты графика вычисляем масштаб по максимальным значениям в файле: отдельно для процентов и суммы. Проценты у нас будут выведены на левой вертикальной шкале, а суммы – на правой. И, естественно, своим отдельным цветом, который указан в блоке инициализации.

Максимальный процент мы округляем до ближайшего значения, кратного 10 (в большую сторону, понятно), и выводим шкалу с шагом 5%. Это грубовато, но для нас достаточно. Впрочем, вы можете выводить проценты и с другим шагом.

А вот для суммы все сложнее. Видно, что нулевой суммы нет, минимальное ее значение в таблице весьма велико. Если масштабировать с нуля, то график получится сжатым, грубым и некрасивым. Поэтому правую шкалу мы будем масштабировать от минимального до максимального значения. Что и как там округляется – разберите сами.

Ну а дальше, собственно, выводятся координатные оси и значения процентов и сумм по вертикали, а лет по горизонтали.

А дальше вызывается процедура «GRAFIK», которая и выводит в окно программы нужную нам гистограмму.

```
Sub GRAFIK
  XP=X0
  YP=Y0
  For I=2 To GOD
    For J=2 To KOL
      GraphicsWindow.BrushColor=DAT[GOD+1][J]
      Y=DAT[I][J]/MAX1*VYSOTA
      X=X0+(I-2)*(KOL+1)*WIRKOL+J*WIRKOL
      GraphicsWindow.FillRectangle(X , Y0-Y , WIRKOL-1 , Y)
    EndFor
    GraphicsWindow.PenColor=COLSUM
    Y=(DAT[I][KOL+1]-MIN2)/(MAX2-MIN2)*VYSOTA
    X=X0*1.5+WIRKOL*(KOL+1)*(I-2)+WIRKOL*(KOL+1)/3
    If XP<>X0 then
      GraphicsWindow.DrawLine(XP , YP , X , Y0-Y)
    EndIf
    XP=X
    YP=Y0-Y
  EndFor
EndSub
```

Код этой процедуры достаточно прост, советую разобрать его самостоятельно. Единственный момент – мы график сумм чертим не с точки (0,0), а с первого года таблицы. Поэтому появился лишний «If». Заметили его? А сам график рисуется в виде отрезков, на каждом шаге цикла от конца прошлого отрезка. Получается не плавный график, а ломаная линия, но такой он и должен быть в нашем случае.

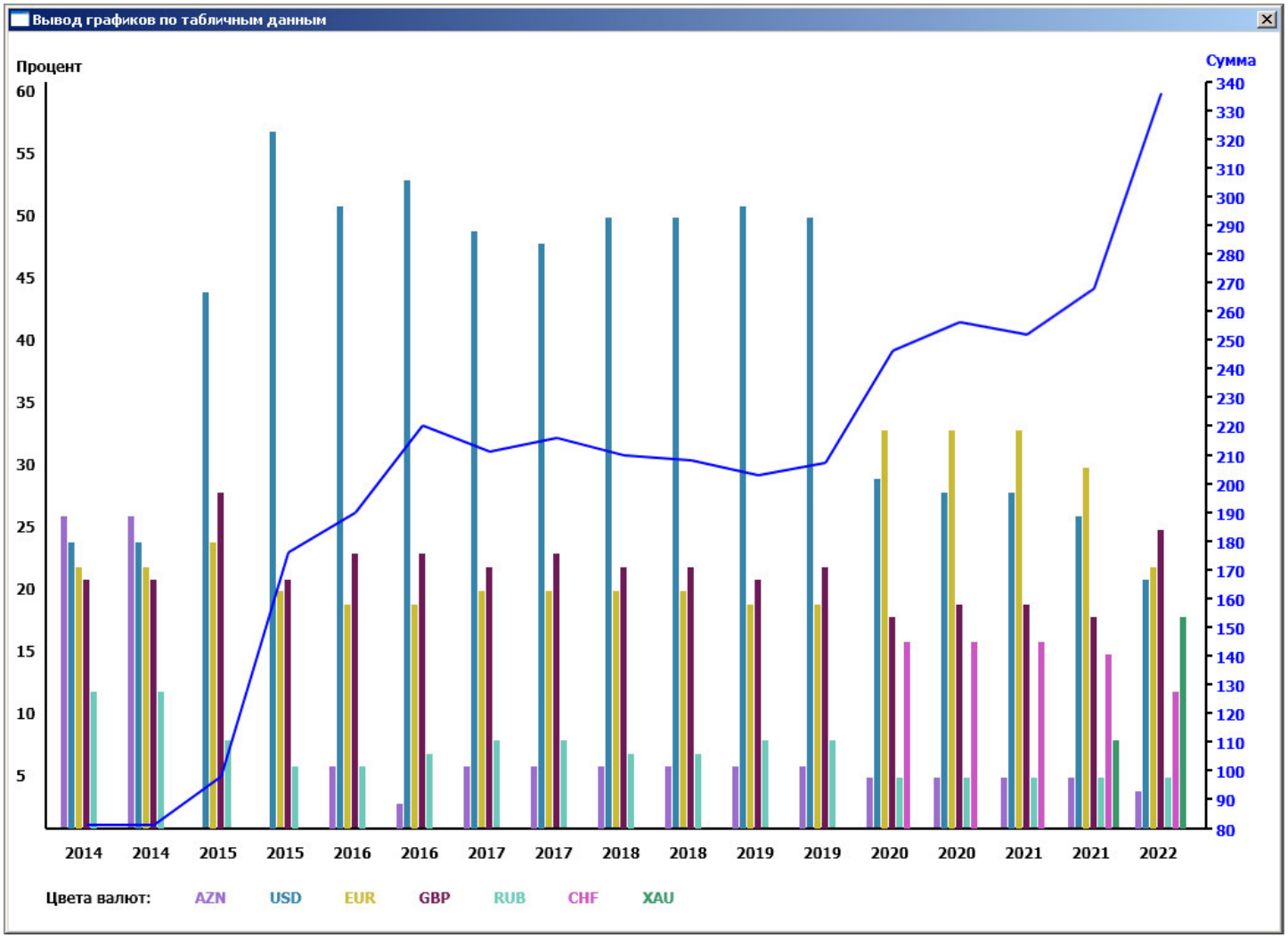
Обратите внимание на значения границ циклов в программе. Выше я писал, что мы не будем создавать отдельные массивы для разнотипных данных. Дело в том, что на SB массивы могут одновременно хранить данные разных типов. Точнее, в этом языке все данные на самом деле – текстовые. Поэтому в одном массиве «DAT» хранятся и названия валют, и годы, и проценты. Первая строка массива – это названия валют, а первый столбец – это годы. Поэтому все операции с самими процентами происходят со второй строки и второго столбца. На других языках было бы сделано иначе.

Что касается цветов, то мы в программе сделали все «в лоб»: устанавливаем цвет принудительно перед каждым его использованием. Конечно, мы группировали одноцветные операции там, где это было возможно, но в целом лучше использовать установку цвета лишний раз, чем потом искать причину его несоответствия задуманному.

Но вот – получили мы график в окне программы. Что дальше? Как его сохранить? Увы, никак. На SB нет механизма сохранения содержимого окна.

Единственное, что нам остается, это сделать банальный скрин экрана и обработать полученное изображение в любом графическом редакторе.

Наше изображение получилось размером 1030*748 пикселей. Там все видно нормально и само это изображение также входит в комплект вместе с текстом программы, файлом данных и т.д. А тут мы просто приложим его в качестве иллюстрации, но за счет масштабирования качество восприятия немного пострадает:



Конечно, цвета получились не идеальными, но в целом все достаточно понятно. Вы можете попытаться самостоятельно придумать иной алгоритм выбора цветов.

Мы не будем разбирать экономическую эффективность и степень оптимальности данной мультивалютной корзины в ее динамике за указанные 9 лет: это выходит за рамки тематики данной книги. Я просто выбрал именно этот пример из-за практической надобности: собирался писать статью на основе результата работы программы. В результате убил двух зайцев: получил материал для будущей статьи плюс разобрал хороший практический пример для этой книги.

В принципе, книга закончена. Думаю, вы получили немало полезной информации и теперь сможете писать программы самостоятельно. Ниже я кратко опишу другие свои программы, написанные на «Small Basic». Во-первых, они сами по себе могут быть вам полезны, а во-вторых, там тоже есть немало интересных моментов и приемов, а также иллюстраций к особенностям данного языка программирования.

Приложение. Примеры авторских программ

На моем компьютерном сайте есть раздел «Исходники», где уже давно представлены некоторые мои программы в виде исходных текстов. Начиная освоение «Small Basic» я, естественно, стал переводить на него именно программы из данного раздела. Раздел-то и предназначен для обучения, сравнения языков и т.д.

Что показал перевод? Довольно интересные результаты, а также он помог мне выявить многие особенности SB, коим и посвящена эта книга. Ниже я перечислю эти программы, а скачать их вы можете в виде готового архива на **странице данной книги**. Там же есть архив со всеми примерами из данной книги, хотя я советовал бы набирать их вручную (для тренировки). Они ведь очень маленькие…

Если же вы хотите скачать версии моих программ и на других языках (для сравнения с SB), то обратитесь к разделу «Исходники».

Не удивляйтесь странным (иногда) названиям этих программ: это сделано для единообразия, и лично мне было так удобно. Да и не в названиях суть…

Я советую самостоятельно изучить и разобрать все эти программы. Делайте копии под другим именем – и меняйте что хотите! В этом и заключается процесс обучения. Ну и, конечно, придумывайте свои задачи. Практика – это главное!

1. «codkey48»

Эта программа показывает коды нажимаемых и отпускаемых клавиш. Используются перехват соответствующих событий, вывод текста в графическое окно, а также получение кода символа. Программа короткая и простая для понимания.

2. «disk2nol»

Данная программа предназначена для необратимого удаления содержимого файлов и чистки свободного места на диске от остатков информации. К сожалению, в силу ограничений SB, возможности программы меньше, чем у ее аналогов на других версиях Бейсика.

Здесь используется только текстовое (консольное) окно, используются параметры командной строки, а также операции с файлами, папками и строками. Также проверяется наличие ошибок в ходе файловых операций.

Код программы довольно длинный из-за невозможности помещать в одну строку несколько однотипных операторов. Это затрудняет чтение и анализ программы, но что делать?

3. «editext5»

Это мой простейший текстовый редактор, о котором я рассказывал в Главе 5. К сожалению, на SB он действительно вышел крайне примитивным и вряд ли может иметь практическое применение (в отличие от аналогов на других языках и версиях Бейсика).

Сначала я вообще хотел написать его в консольном варианте, но это оказалось технически невозможно, пришлось использовать графическое окно, что по сути совершенно неправильно.

Поскольку программа вышла очень ограниченной по функционалу, ее код тоже достаточно мал и прост для понимания. В программе создаются объекты (текстовые поля и кнопки), происходит обработка связанных с ними событий, а также производятся чтение и запись информации в файлах.

4. «expone»

Назначение этой программы – вычисление основания натуральных логарифмов (числа «e») с максимально возможным количеством десятичных знаков после запятой. В указанном выше разделе «Исходники» есть коды этой программы на многих языках, что позволяет сравнить возможности и увидеть особенности. В частности, на SB количество знаков получается весьма небольшим.

Программа работает в консольном окне с вводом и выводом информации, а также использует математические функции, логику и файловые операции. Код программы небольшой, но достаточно сложный. Однако, советую разобраться!

5. «gomoku48»

А это – игра в «Крестики-нолики» на бесконечном поле, человеку надо поставить в ряд пять крестиков. Как ни странно, но вроде бы сложная игра по своему функционалу и оформлению практически не отличается от вариантов на более «продвинутых» языках.

Эту программу я тоже упоминал в Главе 5, поскольку она не только использует объект «***Shapes***», но и активно работает с мышью и нестандартно применяет работу с клавиатурой. Также программа широко использует многомерные массивы.

В целом код программы довольно велик и довольно сложен, ибо алгоритм игры этого требует. Напомню, что сам алгоритм игры не мой, я только внешнее оформление делал.

6. «hind48»

Данная программа является просто образцом работы с графикой. У нее два режима: вращающиеся кольца и разноцветные растущие круги. В качестве заставки она сейчас, конечно, не смотрится (то ли дело времена MS-DOS, когда я ее первый раз писал!), но для демонстрации возможностей и особенностей языка подходит прекрасно.

Помимо графики тут еще используется работа с временем, ведь скорость вращения меняется для создания нужных эффектов. Код программы выглядит длинным и сложным, хотя на самом деле программа очень простая. Если разберетесь – многому научитесь.

7. «palit48»

Это программа подбора цветов и их сочетаний. Она примитивнее своих аналогов из раздела «Исходники», потому что SB не позволяет работать с движками, а имитировать их – себе дороже. В итоге программа почти аналогична варианту, который я когда-то написал под MS-DOS. Но и в таком виде она полезна!

Здесь очень богато представлена работа с мышью, а также работа с цветами – больше и добавить нечего, наверно! Используется перехват событий и присутствует очень большой подготовительный блок, создающий весь интерфейс окна программы. В итоге код программы довольно сложен и велик, но если вы его разберете и освоите, то серьезно продвинетесь в изучении данного языка и программирования в целом.

8. «quine»

Такие программы называются «статусными». Я бы даже советовал не смотреть ее код, а сначала научиться программировать и создать такую программу самостоятельно. Это очень хорошее упражнение!

Назначение программы – «самопечать», то есть программа выводит в консольное окно свой собственный исходный код. Разумеется, код должен выводиться в точности и, конечно, нельзя использовать работу с файлами.

Если вы сами сумеете написать такую программу (кстати, вполне вероятно, что вы решите эту задачу совсем не так, как решил ее я), то вы станете настоящим программистом!

9. «razmno»

Довольно простая (даже примитивная) программа, использующая консольное окно, массивы и математические операции. Ее назначение – разложение заданного числа на простые множители, а также вывод всех пар сомножителей данного числа.

10. «testpc48»

Эта программа не раз упоминалась в данной книге, потому что, несмотря на своей небольшой размер, она содержит немало важных элементов языка SB: это и перехват событий, и ввод данных в графическом окне, и работа с таймером. А также вложенные циклы, математические операции и многое другое.

Назначение данной программы – оценка быстродействия вашего компьютера, причем не только «на холостом ходу», но и при различных сторонних нагрузках на процессор. Как и другие языковые и платформенные ее версии, программу рекомендуется использовать в различных сочетаниях. Впрочем, подробное описание лучше смотреть на странице самой программы.

11. «urav234»

Данная программа (решение уравнений 2-4 степени) не содержит каких-либо особо интересных элементов, но является неплохой иллюстрацией решения математических задач на SB.

Заключение

Вот и закончилась книга. Надеюсь, она помогла вам в освоении поистине волшебной профессии – программирования! Если есть вопросы или предложения – обращайтесь, актуальные контакты есть на всех моих сайтах.

Напомиаю, что книга эта на моем сайте распространяется бесплатно. Однако, если она вам помогла и показалась полезной, вы можете поддержать ее автора, купив этот же файл вместе со всеми исходными кодами **на этой странице**. Цена символическая, это просто знак поддержки.

Там же можно купить и художественные произведения автора.

Напоминаю адрес моего главного сайта – вершины моей пирамиды:

hauserich.vgd.name

Мой компьютерный сайт, где находится данная книга, архивы с примерами и вообще все по теме программирования, расположен на адресе

erichware.com

Желаю успехов!