TURBO VISION

основы практического использования

© ДИАЛОГ-МИФИ

А. М. Епанешников,
В. А. Епанешников

TURBO VISION.

Основы практического использования

МОСКВА ■ "ДИАЛОГ-МИФИ" ■ 1995

ББК 22.1
Е651
УДК 681.325.5

А. Епанешников, В. Епанешников
Е651 Turbo Vision 2.0. Основы практического использования. — М.: "ДИАЛОГ-МИФИ", 1995. — 240 с.

ISBN 5-86404-053-3

Данное пособие посвящено основам практического использования пакета Turbo Vision 2.0 (фирма Borland International), предназначенного для создания пользовательского интерфейса прикладных программ, работающих в текстовом режиме. Пособие предназначено для первоначального знакомства с пакетом Turbo Vision 2.0, простейшими способами его использования при создании программ. Предполагается знание читателем языка Turbo Pascal 7.0 (Borland Pascal 7.0) и основ объектно-ориентированного программирования. Пособие может быть также полезно для программистов, начинающих знакомство с этим пакетом.

2404000000-001
Е651 ———————— Без объявл. ББК 32.973
Г70(03)-95

Учебно-справочное издание

Алексей Михайлович Епанешников,
Василий Алексеевич Епанешников

Turbo Vision 2.0
Основы практического использования.

Редактор О. А. Кузьминова
Макет Е. К. Виноградовой
Обложка Н. В. Дмитриевой
Корректор В. С. Кустов

Лицензия ЛР N 070109 от 29.08.91. Подписано в печать 15.01.95.
Формат 60x84/16. Бум. офс. Печать офс. Гарнитура Таймс.
Усл. печ. л. 13.95. Уч.- изд. л. 11.3. Тираж 5 000 экз. Заказ 1034

Акционерное общество "ДИАЛОГ-МИФИ"
115409, Москва, ул. Москворечье, 31, корп. 2

Подольская типография
142100, г. Подольск, Московская обл., ул. Кирова, 25

ISBN 5-86404-053-3

© А. Епанешников, В. Епанешников, 1995
© Оригинал-макет, оформление обложки.
АО "ДИАЛОГ-МИФИ", 1995

ВВЕДЕНИЕ

При создании прикладной программы программист большие усилия затрачивает на организацию ввода/вывода информации, т. е. на формирование интерфейса. Они соизмеримы с усилиями, затрачиваемыми на программирование основного алгоритма программы, по которому решается конкретная задача. Работа, связанная с организацией ввода/вывода, повторяется от программы к программе, требует выполнения однотипных операций, отвлекает внимание программиста от решения основной задачи.

Хорошим средством упрощения работы по организации ввода/вывода, его унификации с учетом современных требований к интерфейсу программ является разработанный фирмой Borland International пакет Turbo Vision 2.0, представляющий объектно-ориентированную библиотеку средств формирования пользовательского интерфейса. В данном пособии будет рассмотрена версия пакета Turbo Vision 2.0 применительно к языку Turbo Pascal 7.0 [1].

Структуру передачи информации в достаточно общем случае можно представить в виде, изображенном на рис. 1.

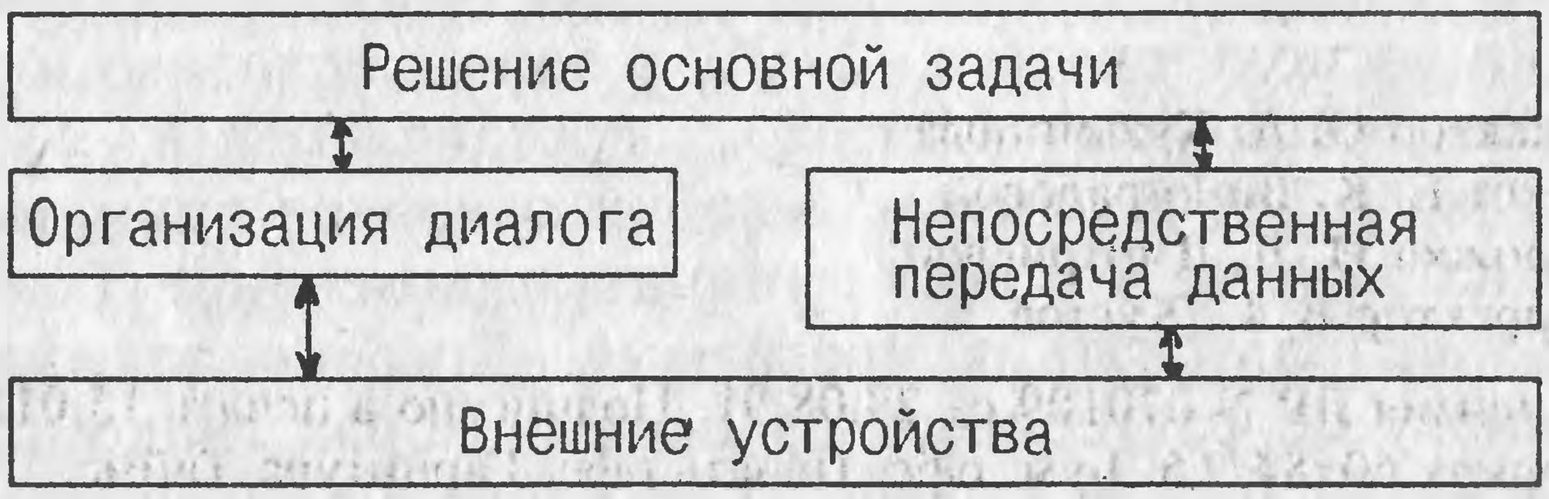


Рис. 1. Структура программы с учетом организации ввода/вывода

Согласно этому рисунку программу можно представить как совокупность двух частей: часть программы, обеспечивающая решение основной задачи, и часть, обеспечивающая организацию ввода/вывода информации с внешних устройств (на внешние устройства), - так называемый интерфейс программы. В основную часть (из основной части) программы информация может передаваться двумя способами - непосредственная передача информации (например, программа формирует какие-то данные и размещает их в конкретном файле на магнитном диске) и передача информации с помощью организации диалога (например,

после формирования тех же данных происходит уточнение, в какой конкретно файл следует поместить сформированную информацию). Способ, основанный на организации диалога, является более универсальным и именно для организации диалога предназначен в первую очередь пакет Turbo Vision.

В современных программах, работающих в текстовом режиме, диалог обычно организуется с помощью трех средств: меню, диалоговых окон и строк состояния. Turbo Vision позволяет реализовать все эти средства диалога. Они и будут рассмотрены в данном пособии.

Зачем написано данное пособие? Ведь существуют отличные материалы фирмы Borland по Turbo Vision 2.0 [2]. Однако объем этой книги - 618 страниц - несколько велик для первого знакомства. Кроме этого, далеко не всем под силу прочитать такую книгу на английском языке, да и мало у кого она есть. Существует перевод Turbo Vision 1.0, выпущенный анонимно фирмой "Диалектика", г. Киев, в 1992 г., но он выполнен настолько неудачно, что скорее затуманивает представления о пакете, чем проясняет их. Книга [3] уже предполагает знание Turbo Vision и описывает в основном новые возможности версии 2.0 по сравнению с версией 1.0. Другие материалы авторам неизвестны. Цель данного пособия - дать читателю начальные представления по пакету Turbo Vision 2.0.

Авторы не ставили перед собой задачу полностью описать все возможности пакета Turbo Vision - для более детального изучения можно воспользоваться [2, 3], а также исходными текстами библиотеки (Run-time Library), поставляемыми с пакетом Borland Pascal 7.0. В данном пособии основное внимание сосредоточено на вопросе - с чего начинать применение Turbo Vision и как его использовать в программе. Пособие должно помочь программисту не растеряться на первых порах знакомства с пакетом, насчитывающем свыше 70 стандартных объектов, каждый из которых обладает большим количеством параметров и методов, и тем не менее научиться создавать свои простейшие интерфейсные элементы. Все стандартные объекты расположены в модулях Turbo Vision (см. приложение А). Все их подробно рассмотреть в кратком пособии невозможно, поэтому здесь рассмотрены наиболее часто используемые объекты.

Для большей наглядности применения Turbo Vision на протяжении большей части пособия будет рассматриваться создание конкретной программы - непрограммируемого калькулятора, типа "Электроника Б3-19М" [4]. В этой программе организация ввода/вывода информации является

одной из важных функций. При этом в ряде случаев для реализации тех или иных возможностей Turbo Vision калькулятору будут придаваться нехарактерные для него свойства (например, возможность использования различных систем счисления), а некоторые возможности реального калькулятора будут либо опущены, либо использованы в минимальных объемах (например, не включено вычисление стандартных функций). Тем не менее окончательный вариант калькулятора является пригодным для выполнения простейших операций.

Основными особенностями данного калькулятора, которые будут учтены при создании программы, следующие. Исходные числа вводятся в форме с фиксированной десятичной точкой. Ввод информации осуществляется на основе бескобочной записи вычислений, когда вначале поочередно вводятся два числа, а затем уже знак операции над ними [4]. Калькулятор выполняет все основные арифметические операции (умножение, деление, сложение и вычитание). Вводимые данные размещаются в стеке. Результаты вычислений также размещаются в стеке, замещая используемые операнды, и выводятся на экран в форме с фиксированной точкой. В отличие от своего прототипа программа позволяет производить ввод информации и получать результаты не только в десятичной системе счисления. Размер стека для размещения данных может изменяться. Может изменяться также и название калькулятора. Пользователь имеет возможность в любое время посмотреть содержимое стека. По желанию он может также работать одновременно с несколькими калькуляторами. О других особенностях калькулятора будет сказано в процессе обсуждения программы.

Окончательный вариант программы приведен в приложении Г. Так как программа является учебным примером, при ее создании не ставилась задача оптимизации - авторы старались написать более понятную, легко читаемую программу. Так, например, здесь не использовался оператор with, который делает программу более эффективной, но более трудной для чтения. При написании этой программы были позаимствованы некоторые идеи создания калькулятора из демонстрационной программы, поставляемой фирмой Borland вместе с пакетом Turbo Vision. Все непустые строки программы, не являющиеся к тому же строками комментариев, пронумерованы для более простой ссылки на них. Такие строки будем называть линиями (л., лл.).

При рассмотрении этой программы основное внимание будет уделено организации интерфейса, вопросы же непосредственных вычислений нас интересовать будут в меньшей степени.

В п. 9 пособия будут рассмотрены еще две программы - текстовый редактор и программа изменения цветов основной палитры Turbo Vision (см. приложения Ж и З). Эти программы с некоторой модификацией позаимствованы из демонстрационных файлов пакета Turbo Vision. Приведены они здесь для иллюстрации соответствующих возможностей пакета.

В первой главе пособия будет кратко дано описание основных понятий Turbo Vision и принципов его работы - далеко не полное и охватывающее лишь те понятия, которые необходимы на первых этапах знакомства с пакетом.

Во второй главе описан порядок работы с отображаемыми объектами.

Третья и четвертая главы посвящены созданию меню и окон (диалоговых окон).

В пятой главе будут рассмотрены вопросы создания справочной системы.

Шестая глава посвящена формированию строк состояния.

В седьмой главе описываются некоторые средства задания цветовой палитры объектов.

Восьмая глава посвящена использованию ресурсов при программировании задач и краткому описанию разработанного авторами генератора ресурсов, позволяющего создавать меню, строки состояния и диалоговые окна.

В девятой главе приведены дополнительные возможности пакета Turbo Vision: создание текстового редактора, отображение древовидных структур, задание цветовой палитры с помощью специального диалогового окна tColorDialog. При первом чтении этот материал может быть безболезненно опущен.

К сожалению, стандартный пакет Turbo Vision предназначен для работы только с латинскими буквами. Это создает определенные трудности, т. к. стандартные сообщения идут на английском языке, а клавиши быстрого управления в системе меню могут быть только латинскими. Для приложений на русском языке это не очень удобно. В связи с этим при рассмотрении конкретных примеров некоторые возможности (такие, как использование клавиш быстрого управления, соответствующих буквам кириллицы) показаны не будут, а некоторые стандартные средства будут русифицированы. На практике целесообразно, имея исходные тексты Turbo Vision, адаптировать их к буквам кириллицы, что сделать не так уж и трудно.

Пособие предназначено для программистов, желающих познакомиться с основными возможностями пакета Turbo Vision с целью использования его в своей практике. Предполагается, что читатель знаком с программированием на языке Turbo Pascal и объектно-ориентированным программированием, на основе которого построен пакет Turbo Vision.

Тексты программ, рассмотренных в пособии, а также демонстрационная версия редактора ресурсов (см. п. 8.5) находятся также на дискете, которую можно приобрести вместе с пособием.

1. ОСНОВНЫЕ ПОНЯТИЯ TURBO VISION

Turbo Vision представляет собой библиотеку объектов, позволяющих создать пользовательский интерфейс и широко использующих свойства инкапсуляции, наследования и полиморфизма [1]. Почти все объекты библиотеки являются потомками общего абстрактного объекта tObject. Мы рассмотрим их классификацию, исходя из задач, возникающих при создании интерфейса.

Объект tObject достаточно прост. Он не имеет параметров и обладает тремя методами, два из которых - конструктор Init:

```
constructor tObject.Init;
```

заполняющий выделенную под объект область нулями, и деструктор Done:

```
destructor tObject.Done;
```

не выполняющий никаких действий, - должны быть переопределены в его потомках, а третий метод Free:

```
procedure tObject.Free;
```

удаляющий объект из динамической области памяти, никогда не переопределяется, т. к. может удалить любой потомок объекта tObject. Фактически последний метод эквивалентен оператору

```
Dispose(pObject(@Self), Done);
```

Параметр Self определяет текущий объект, с которым осуществляется работа в программе. В данном случае Self - удаляемый объект, который вызвал метод Free, а @Self - адрес удаляемого объекта. Done - деструктор удаляемого объекта (а не объекта tObject), т. к. деструктор является виртуальным методом.

В программе-калькуляторе метод Free используется трижды - в лл. 533, 692 и 700.

При организации интерфейса приходится решать две задачи: создание соответствующего образа на экране и возможность посредством этого образа передачи информации как от пользователя к вычислительной машине, так и от машины к пользователю. Помимо этих двух задач при организации интерфейса приходится выполнять различные вспомогательные операции (вычисления, контроль и преобразование данных, хранение информации и т. д.). В связи с этим все объекты Turbo Vision можно разбить на две группы: отображаемые и неотображаемые объекты:

При создании программы-калькулятора также придется решать эти задачи - создание на экране дисплея образа калькулятора с его кнопками управления, индикатором результатов и т. д., обеспечение реакции программы на действия пользователя программы и вывод на экран результатов работы калькулятора, а также контроль и хранение данных.

1.1. Отображаемые и неотображаемые объекты

Отображаемыми объектами являются такие объекты, которые могут быть изображены на экране. Эти объекты позволяют наглядно представить пользователю информацию, дать ему средства ввода информации в программу наиболее естественным образом. К таким объектам относятся, в частности, меню (см. п. 3), строки состояния (см. п. 6), диалоговые окна (см. п. 4) с их компонентами.

В программе-калькуляторе отображаемым объектом является, в частности, образ панели калькулятора, являющийся примером диалогового окна (рис. 1.1).

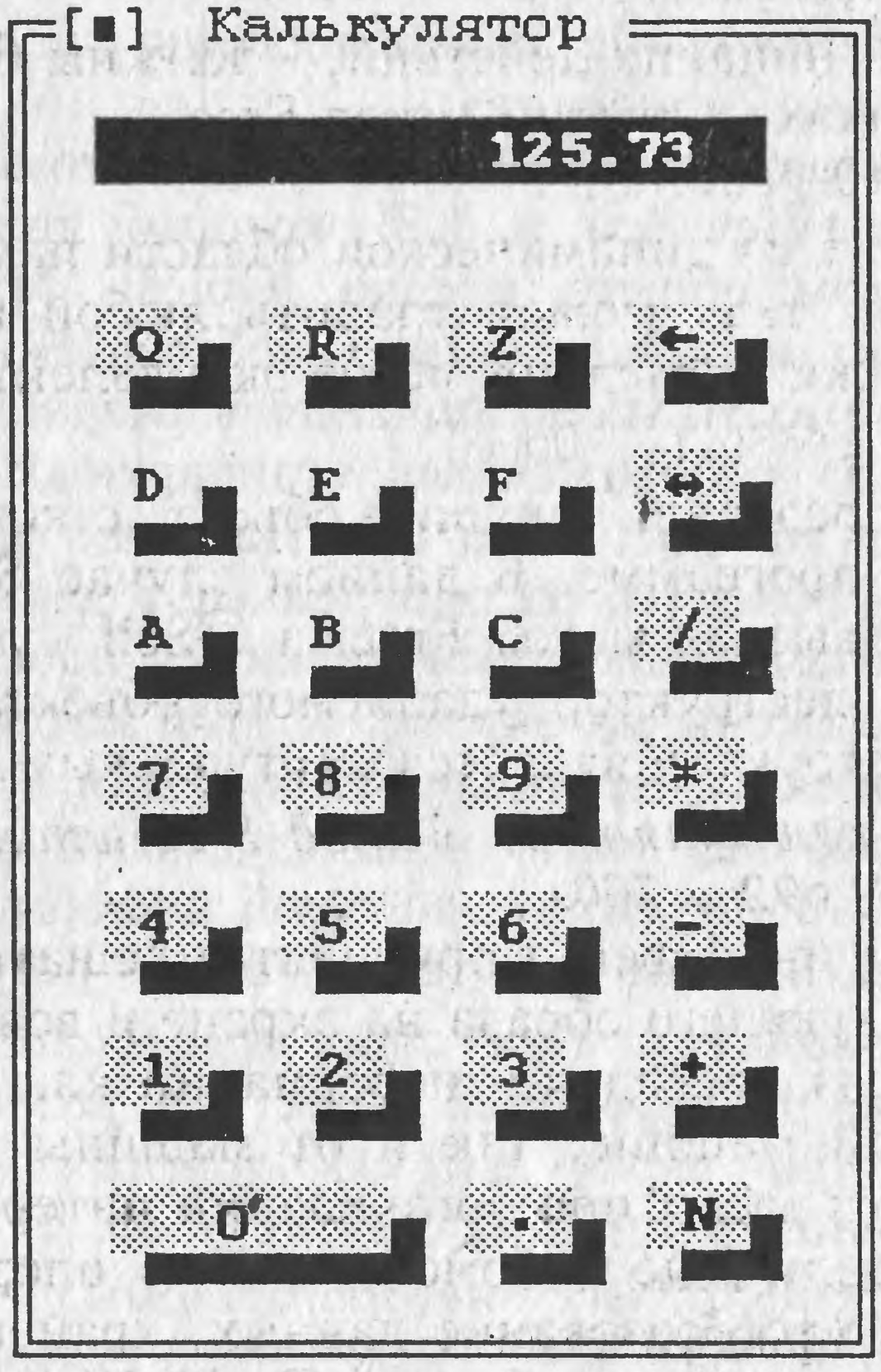


Рис. 1.1. Образ панели калькулятора

У этого калькулятора вверху расположен индикатор, в котором помещаются вводимые числа и результаты вычислений. Далее идут кнопки, назначение которых следующее:
"0"..."9" и "A"..."F" - кнопки, задающие цифры;
"." - задание десятичной точки;
"Q" (Quite) - полная очистка стека и размещение в нем числа 0;
"R" (Remove) - удаление из стека последнего числа; если это число единственное, в стек помещается число 0;
"Z" (Zero) - помещение в стек числа 0 вместо последнего введенного или вычисленного;
"Стрелка влево" - удаление из числа последнего введенного символа (цифры или десятичной точки);
"Двойная стрелка" - изменение знака у числа;
"/", "", "-", "+" - знаки арифметических операций;*
"N" (New) - переход к вводу нового числа.
На рисунке кнопки "A"..."F" имеют цвет, отличный от цвета других кнопок. Это означает, что в данный момент времени эти кнопки не задействованы (используется десятичная система счисления).

Любой отображаемый объект является потомком общего абстрактного объекта tView (см. п. 1.3), который является непосредственным потомком общего предка tObject. Мы познакомимся с наиболее часто используемыми отображаемыми элементами по мере необходимости их включения в разрабатываемую программу. Структура отображаемых стандартных объектов Turbo Vision приведена на рис. 1.2 и 1.3. На этих рисунках двойной рамкой обведены объекты, рассмотренные в данном пособии.

Неотображаемыми объектами являются объекты, которые не могут быть изображены на экране. К этой категории, в частности, относятся объекты, предназначенные для контроля и хранения данных. Наиболее часто используемыми объектами такого класса являются анализаторы вводимой информации (см. п. 8.1), коллекции (см. п. 8.2), потоки (см. п. 8.3) и ресурсы (см. п. 8.4).

В программе-калькуляторе объектом такого класса может быть объект, реализующий стек для размещения данных, получаемых при вводе или вычислениях (тип tNumberCollection, потомок типа tCollection - лл. 17-19).

Структура стандартных неотображаемых объектов Turbo Vision приведена на рис. 1.4. Здесь также двойной рамкой обведены объекты, рассмотренные в пособии.

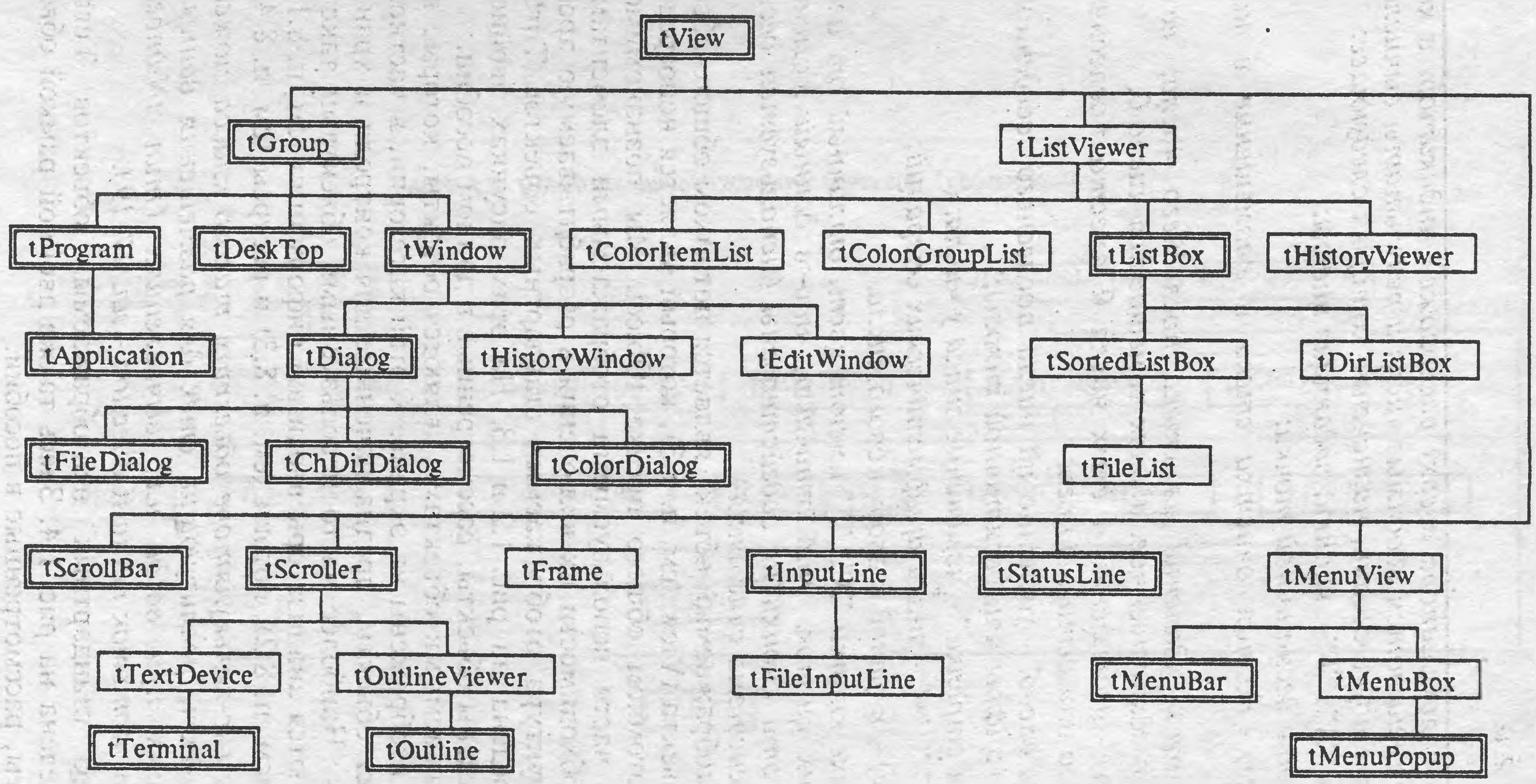


Рис. 1.2. Иерархия отображаемых объектов

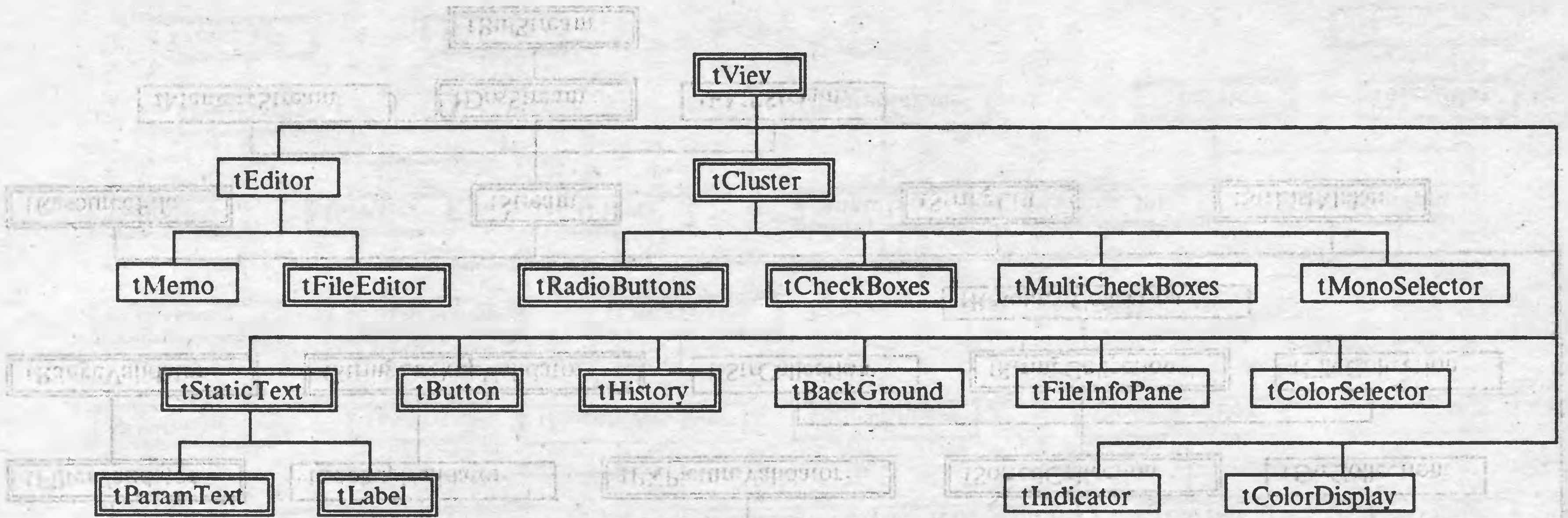


Рис. 1.3. Иерархия отображаемых обьектов (продолжение)

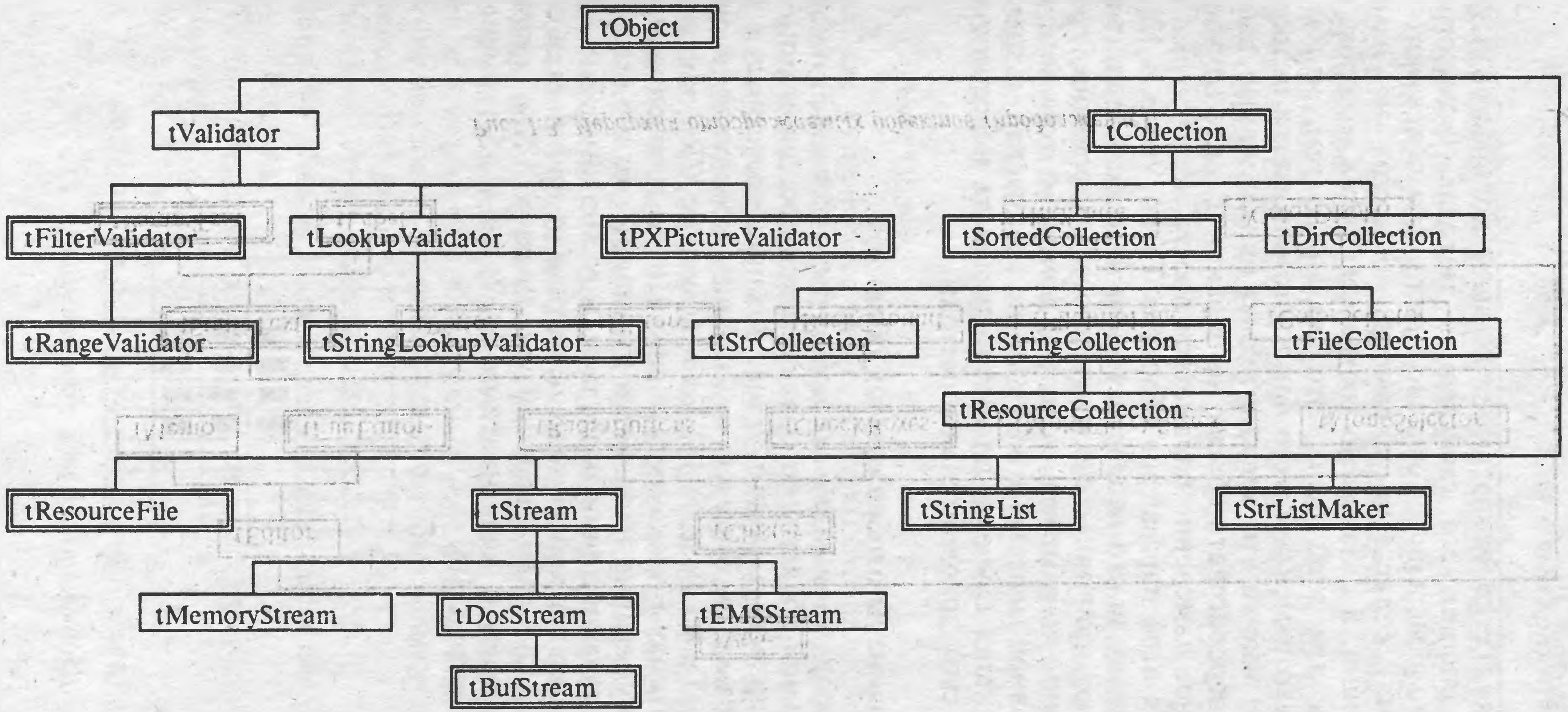


Рис 1.4. Иерархия неотображаемых объектов

Любой объект Turbo Vision имеет большое количество параметров, определяющих его свойства, и методов (подпрограмм), задающих его поведение. Согласно принципам объектного программирования желательно параметры объектов не изменять непосредственно, а с помошью соответствующих методов, предназначенных для этой цели. Это связано с тем, что в ряде случаев только изменение того или иного параметра недостаточно - следует выполнить дополнительно определенные действия (например, перерисовать объект на экране). В ряде случаев непосредственное изменение параметров просто недопустимо. Практически для всех параметров объектов Turbo Vision в [2] указано, допустимо ли его непосредственно изменять (записывать). К сожалению, принцип объектного программирования, требующий изменения параметров только с помощью того или иного метода, в полной мере в Turbo Vision не реализован, и приходится в простейших случаях параметры изменять непосредственно либо писать свои дополнительные методы.

1.2. Координаты отображаемых объектов

Любой отображаемый объект может быть размещен на экране в прямоугольной области, которая задается координатами левого верхнего и правого нижнего углов. Задание координат в Turbo Vision несколько отличается от задания координат в модуле CRT пакета Turbo Pascal [1]. Если в модуле CRT координаты задают координаты знакоместа, причем и по горизонтали, и по вертикали, нумерация начинается с единицы, то в Turbo Vision координаты задают линии раздела между знакоместами, и при этом нумерация по горизонтали и вертикали начинается с нуля (кординаты 0, 0 - левый верхний угол экрана). Так, например, область, задаваемая координатами (3, 2) и (7, 5), имеет вид, показанный на рис. 1.5.

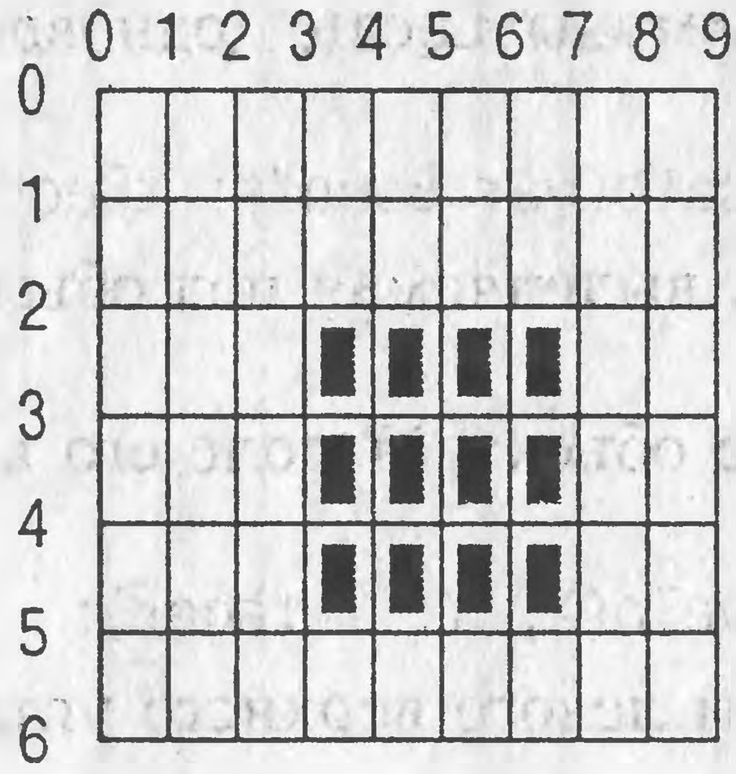


Рис. 1.5. Область экрана с координатами (3, 2) и (7, 5)

Так же как и в модуле CRT, координаты могут быть абсолютными, отсчитываемыми относительно левого верхнего угла экрана, и относительными, отсчитываемыми от левого верхнего угла соответствующего окна (так называемого владельца данного отображаемого объекта).

Для задания координат точки на экране и координат прямоугольной области используются два объекта - tPoint, содержащий две координаты точки - X и Y:

```
type tPoint = object
       X, Y: Integer;
     end;
```

и tRect, определяющий левый верхний и правый нижний углы прямоугольной области. Объект tRect не только содержит соответствующие координаты, но также обладает методами, позволяющими задавать, изменять, получать эти координаты и выполнять над ними ряд других операций.

1.3. Объект tView

Как уже указывалось выше, предком отображаемых объектов является объект tView. Несмотря на то что этот объект абстрактный, т. е. нельзя создать конкретный образец этого типа, он имеет важное значение, т. к. задает общие свойства всех отображаемых элементов и определяет виртуальные методы, которые затем переопределяются для конкретных отображаемых объектов. В связи с этим остановимся на нем поподробнее.

У объекта tView задаются следующие параметры:

Расположение и размеры объекта. Расположение объекта tView задается координатами его левого верхнего угла в поле его владельца (см. п. 1.4) - параметр Origin, а размеры - параметром Size, задающим его ширину и высоту. Непосредственно изменять эти параметры в программе не следует. Для их задания нужно использовать методы Locate (одновременное задание всех координат объекта):

```
procedure tView.Locate(var Bounds: tRect);
```

где Bounds - область, выделяемая под объект в координатах его владельца;

MoveTo (перемещение объекта в поле его владельца без изменения размеров):

```
procedure tView.MoveTo(X, Y: Integer);
```

где X, Y - координаты левого верхнего угла объекта в поле его владельца,

и GrowTo (изменение размеров объекта):

```
procedure tView.GrowTo(X, Y: Integer);
```

где X, Y - горизонтальный и вертикальный размеры объекта.

При изменении размеров объекта с помощью метода GrowTo остается на месте левый верхний угол объекта.

Координаты курсора. Ряд отображаемых объектов, таких, как строка ввода текстовой информации, редакторы и т. д., могут иметь видимый курсор. Координаты этого курсора задаются параметром Cursor. Для объектов, не имеющих видимого курсора, значение этого параметра безразлично. Задать координаты курсора можно с помощью метода SetCursor:

```
procedure tView.SetCursor(X, Y: Integer);
```

где X, Y - координаты курсора в поле объекта.

Курсор отображаемого элемента можно сделать видимым и невидимым (см. ниже параметр sfCursorVis флагов состояния), а также изменять его размеры (см. параметр sfCursorIns).

Флаги свойств. Параметр Options размером в одно 16-разрядное слово задает ряд свойств отображаемого объекта. Назначение битов этого слова следующее (в круглых скобках приведены константы, соответствующие единице в указанном бите):

0-й бит (ofSelectable) - возможность активизации (выделения) данного объекта в его владельце либо с помощью мыши, либо с помощью клавиши табуляции. Это свойство используется у любого объекта, и обычно флаг задается равным единице.

1-й бит (ofTopSelect) - возможность размещения объекта при его активизации (выделении) сверху всех остальных объектов конкретной группы (см. п. 1.4). Это свойство используется у окон (тип tWindow) и их потомков - диалоговых окон, стандартных диалоговых окон и т. д.

2-й бит (ofFirstClick) - возможность использования первого нажатия клавиши не только для выделения объекта (см. 0 бит), но и для выполнения действий, которые активизируются данным объектом. Свойство используется практически у всех отображаемых объектов.

3-й бит (ofFramed) - наличие рамки вокруг отображаемого объекта. Это свойство используется для подэлементов окон. Не следует этот флаг путать с наличием рамки окна (tWindow) и его потомков (см. п. 4.1).

4-й бит (ofPreProcess) - опережающая реакция на событие (см. п. 1.7). Задание флага приводит к изменению стандарт-

ной последовательности реакции отображаемых элементов группы на событие. Свойство может использоваться у любого элемента по мере необходимости.

5-й бит (ofPostProcess) - запаздывающая реакция на событие (см. п. 1.7). Аналогично предыдущему флагу, но с другими последствиями его использования.

6-й бит (ofBuffered) - наличие КЭШ-буфера у отображаемого объекта, когда образ отображаемого объекта не создается каждый раз заново при его прорисовке на экране, а хранится в специальном КЭШ-буфере [5]. Флаг важен для сложных составных объектов типа окна (tWindow) и его потомков.

7-й бит (ofTileable) - расположение отображаемых объектов встык или каскадно (с наложением). Этот флаг используется только у окон (tWindow) и их потомков.

8-й бит (ofCenterX) - центрирование объекта по горизонтали. При задании этого флага объект при его первоначальной прорисовке центрируется относительно вертикальной оси его владельца. Флаг может использоваться у элементов, входящих в группу элементов (см. п. 1.4).

9-й бит (ofCenterY) - центрирование объекта по вертикали. Аналогично предыдущему флагу, но только объект центрируется относительно горизонтальной оси его владельца.

10-й бит (ofValidate) - задается проверка правильности введенной в объект информации при выходе из этого объекта (активизации другого объекта). Флаг используется только со строками ввода.

12-й бит (ofVersion20) - используется Turbo Vision 2.0. Ряд объектов (как, например, строка ввода - см. п. 4.4.1) имеют принципиальные различия в зависимости от версии Turbo Vision. Версия 2.0 имеет дополнительные возможности по сравнению с версией 1.0.

13-й бит (ofVersion10) - используется Turbo Vision 1.0. (см. 12-й бит).

Остальные биты слова флагов свойств не используются. Для задания флагов свойств можно использовать константу ofCentered = $0300 (одновременное центрирование по горизонтали и вертикали), с помощью которой можно одновременно установить 8-й и 9-й биты параметра Options. Имеется константа ofVersion = $3000, позволяющая выделить биты, связанные с версией Turbo Vision.

Флаги перемещения. Параметр DragMode размером в один байт задает характеристики отображаемого объекта при его перемещении:

0-й бит (dmDragMove) - разрешение перемещения объекта либо с помощью мыши, либо с помощью клавиш направления. Используется у тех объектов, которые по тем или иным причинам могут перемещаться в поле своего владельца (см. п. 1.4), например у окон (tWindow) и их потомков.

1-й бит (dmDragGrow) - разрешение изменения размеров объекта либо с помощью мыши, либо с помощью клавиш направления. Используется у окон (tWindow) и их потомков.

4-й бит (dmLimitLoX) - запрещение левой стороне отображаемого объекта выходить за заданные границы. В качестве таких границ часто выступают размеры владельца данного объекта (например, размеры рабочего поля - см. п. 1.5, в котором размещены окна).

5-й бит (dmLimitLoY) - аналогично 4-му биту, но запрещение верхней стороне отображаемого объекта выходить за заданные границы.

6-й бит (dmLimitHiX) - аналогично 4-му биту, но запрещение правой стороне отображаемого объекта выходить за заданные границы.

7-й бит (dmLimitHiY) - аналогично 4-му биту, но запрещение нижней стороне отображаемого объекта выходить за заданные границы.

Остальные биты параметра DragMode не используются. Для одновременного задания ограничений на перемещение во всех направлениях можно использовать константу dmLimitAll = $F0, устанавливающую в единицу биты с 4-го по 7-й.

Флаги изменения размеров. Параметр GrowMode размером в один байт задает свойства отображаемого объекта при попытке изменения размеров его владельца с помощью мыши или клавиш направления.

0-й бит (gfGrowLoX) - левая сторона отображаемого объекта остается на одном и том же расстоянии от правой стороны его владельца (см. п.1.4) при изменении размеров владельца.

1-й бит (gfGrowLoY) - верхняя сторона отображаемого объекта остается на одном и том же расстоянии от нижней сто-

роны его владельца (см. п. 1.4) при изменении размеров владельца.

2-й бит (gfGrowHiX) - правая сторона отображаемого объекта остается на одном и том же расстоянии от правой стороны его владельца (см. п. 1.4) при изменении размеров владельца.

3-й бит (gfGrowHiY) - нижняя сторона отображаемого объекта остается на одном и том же расстоянии от нижней стороны его владельца (см. п. 1.4) при изменении размеров владельца.

4-й бит (gfGrowRel) - отображаемый объект изменяет свои размеры пропорционально размерам его владельца. Такой режим может потребоваться при отображении окна в случае изменения режима адаптера с 25 строк на 43/50 строк и наоборот.

Остальные биты параметра GrowMode не используются. Для одновременной фиксации сторон отображаемого объекта относительно размеров его владельца можно использовать константу gfGrowAll = $0F, устанавливающую в единицу биты с 0-го по 3-й.

Флаги обработки событий. Параметр EventMask размером в одно слово определяет, какие события может обрабатывать данный отображаемый объект. Этот параметр будет рассмотрен подробно при описании возможных видов событий в программе (см. п. 1.7).

Флаги состояния. Параметр State размером в одно слово определяет текущее состояние данного отображаемого объекта. Как правило, изменение этого параметра происходит автоматически в процессе выполнения программы (например, выделенное состояние объекта), и не следует без особых причин менять его в программе. Если же все-таки потребуется произвести те или иные изменения, их необходимо производить с помощью соответствующих методов, специально для этого предназначенных.

Отдельные биты параметра имеют следующее назначение:

0-й бит (sfVisible) - объект может быть видим в поле своего владельца (см. п. 1.4). Если этот флаг установлен (такое состояние задается автоматически при создании объекта), объект будет виден полностью или частично, если только он не закрывается другими отображаемыми объектами, а его владелец сам виден на экране. Для задания этого флага используется метод Show:

```
procedure tView.Show;
```

для сброса - метод Hide:

```
procedure tView.Hide;
```

Чтобы определить, виден ли на самом деле объект, можно воспользоваться методом Exposed:

```
function tView.Exposed: Boolean;
```

где результат - True, если какая-либо часть объекта видна на экране.

1-й бит (sfCursorVis) - курсор объекта видим в поле объекта. У некоторых отображаемых объектов курсор задается видимым (например, у строки ввода - см. п. 4.4.1), у других - невидимым (например, у статического текста - см. п. 4.4.6). Чтобы изменить заданное состояние и сделать курсор видимым, следует воспользоваться методом ShowCursor:

```
procedure tView.ShowCursor;
```

чтобы сделать его невидимым, используется метод HideCursor:

```
procedure tView.HideCursor.
```

2-й бит (sfCursorIns) - увеличенный размер курсора. Для задания курсора увеличенного размера следует воспользоваться методом BlockCursor:

```
procedure tView.BlockCursor;
```

для задания нормального размера курсора (такой размер задается автоматически при создании объекта) используется метод NormalCursor:

```
procedure tView.NormalCursor;
```

3-й бит (sfShadow) - объект имеет тень. При создании одних объектов (например, окон - см. п. 4.1) задается тень, у других объектов (например, списков - см. п. 4.4.4) тень не задается. Если требуется изменить заданное состояние этого флага, следует воспользоваться универсальным методом SetState:

```
procedure tView.SetState(AState: Word; Enable: Boolean);
                                                  virtual;
```

где AState - флаг, который следует изменить;

Enable = True, если флаг следует установить, и False, если сбросить.

С помощью этого метода можно изменить любые флаги состояния, но при каждом обращении к нему можно изменить только один флаг.

4-й бит (sfActive) - объект является либо активным окном (окном, с которым происходит в настоящее время работа в программе), или подэлементом активного окна. Как правило, этот флаг автоматически меняет свое значение в процессе работы программы и изменять его в программе практически не требуется Если же все-таки потребуется изменить этот флаг, следует воспользоваться универсальным методом SetState.

5-й бит (sfSelected) - объект является выделенным в группе (см. п. 1.6). В каждой группе может быть только один выделенный объект. Как и предыдущий флаг, состояние этого флага изменяется автоматически в процессе работы программы, и обычно не возникает необходимости принудительного изменения его состояния (изменить же его можно с помощью метода SetState).

6-й бит (sfFocused) - объект является сфокусированным, т. е. является выделенным в группе, и все его владельцы (см. п. 1.6) также являются выделенными. Сфокусированный объект - это тот отображаемый объект, с которым в настоящее время происходит работа (например, конкретный редактор текста, если на экране открыто несколько редакторов). Поведение этого флага таково же, как и предыдущего флага.

7-й бит (sfDragging) - объект находится в режиме перемещения или изменения своих размеров либо с помощью клавиш направления, либо с помощью мыши. Обычно этот флаг используется у окон (см. п. 4.1) и устанавливается у окна при переходе его в такой режим, а сбрасывается при выходе из этого режима. Принудительное изменение флага возможно с помощью метода SetState.

8-й бит (sfDisabled) - объект является "запрещенным", т. е. не реагирующим ни на какие воздействия. Задать этот флаг можно с помощью метода SetState.

9-й бит (sfModal) - объект является модальным (см. п. 1.6). Этот флаг характерен для окон и устанавливается, если окно находится в модальном режиме (использование метода ExecView или ExecuteDialog - см. п. 2.8). Принудительно изменять этот флаг не следует (в случае необходимости можно воспользоваться методом SetState).

11-й бит (sfExposed) - объект имеет в качестве окончательного владельца объект tApplication, т. е. имеет возможность отображаться на экране (не путать с флагом ofVisible). Принудительно изменять флаг не следует (его можно изменить с помощью SetState).

Остальные биты параметра State не используются.

Следует отметить, что при непосредственном задании флагов следует использовать логические операции, а не арифметические. Последние в ряде случаев дадут неправильный результат. Так, например, если в параметре Options следует установить флаги ofPreProcess и ofPostProcess, то выполнить это следует с помощью операции

```
Options:= Options or ofPreProcess or ofPostProcess;
```

но не с помощью операции

```
Options:= Options + ofPreProcess + ofPostProcess;
```

Контекстная помощь. Этот параметр (HelpCtx) представляет собой число типа Word, который задает значение параметра контекстной помощи в случае, если отображаемый объект является сфокусированным, т. е. объектом, с которым в данный момент осуществляется работа, - см. выше. Этот параметр используется в справочной системе (см. п. 5) и в системе строк состояния (см. п. 6). Если контекстная помощь не нужна, этот параметр следует задать равным нулю (константа hcNoContext). Это же значение принимает параметр и при создании объекта.

Следующий отображаемый элемент. Этот параметр (Next) содержит указатель на следующий объект в группе, в которую входит данный объект (см. Z-порядок группы - п. 1.4). Параметр устанавливается автоматически при включении объекта в группу. Принудительно менять значение этого параметра не следует.

Владелец объекта. Параметр (Owner) представляет собой указатель на владельца объекта (см. п. 1.4) - группу, в которой расположен данный объект. Параметр устанавливается автоматически при размещении объекта в группе - обычно с помощью метода Insert объекта tGroup (см. п. 1.4) или другого метода, включающего в себя метод Insert (см. пп. 2.3, 2.8), и изменять его принудительно не следует.

Объект tView обладает 67 методами плюс 1 метод, унаследованный от объекта tObject. Многие из этих методов являются виртуальными и переопределяются для потомков. Все методы можно условно разбить на несколько групп.

К первой группе можно отнести методы, с помощью которых создается или уничтожается объект. Здесь следует отметить два

конструктора - Init, с помощью которого инициализируется новый объект, и Load, с помощью которого объект загружается из потока (см. п. 8.3), а также деструктор Done, выполняющий действия, обратные действиям конструктора Init.

Ко второй, самой многочисленной группе, можно отнести методы, с помощью которых либо задаются, либо определяются текущие параметры объекта (задание и получение указанных выше параметров).

К третьей группе можно отнести методы, позволяющие поместить объект в группу (см. п. 1.4), отобразить его на экране, удалить из группы, удалить с экрана.

К четвертой группе можно отнести методы обработки событий (см. п. 1.7), наиболее часто используемым из которых является метод HandleEvent, обрабатывающий текущее событие.

Наконец, к пятой группе отнесем все остальные методы, выполняющие разнообразные дополнительные действия (например, получение палитры цветов для объекта, разрешение и запрещение выполнения команд и т. д.).

Наиболее часто используемые методы объекта tView будут рассмотрены ниже в различных примерах, здесь мы рассмотрим конструктор Init. Хотя реальные образцы объекта tView не создаются, т. к. он является абстрактным объектом, у конструктора любого реального отображаемого объекта практически всегда используется обращение к конструктору Init объекта tView:

```
constructor tView.Init(var Bounds: tRect);
```

где Bounds - размеры объекта.

Конструктор создает объект с заданными размерами и устанавливает параметры, приведенные в табл. 1.1.

Таблица. 1.1. Параметры объекта tView при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	0
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+evCommand
State	sfVisible
HelpCtx	hcNoContext
Next	nil
Owner	nil

1.4. Группа (tGroup). Z-порядок

В общем случае отображаемый объект может иметь достаточно сложную структуру и включать в себя другие видимые элементы в качестве своих подэлементов. Эти подэлементы, в свою очередь, могут включать в себя другие подэлементы и т. д.

Совокупность объединенных отображаемых объектов представляет собой группу (тип tGroup) и может рассматриваться как нечто единое, предназначенное для использования в программе. Так же как и объект tView, объект tGroup является абстрактным объектом и не может иметь конкретные образцы в программе, однако является родоначальником целого ряда реальных объектов (в первую очередь различных окон) и определяет основные свойства таких объектов как некоего целого. Сам объект tGroup является потомком объекта tView и наследует или переопределяет его параметры и методы.

Любой объект типа tGroup представляет собой основной элемент группы (например, диалоговое окно), называемый владельцем элементов группы, в который включено то или иное количество подэлементов (например, различные элементы управления диалогового окна). Параметры, наследуемые группой от объекта tView, относятся к основному объекту группы - владельцу группы; отдельные подэлементы, также являющиеся потомками объекта tView, обладают своими характеристиками, часто не совпадающими с характеристиками их владельца.

В дополнение к параметрам объекта tView объект tGroup имеет ряд собственных параметров, наиболее важными и часто используемыми из которых являются два параметра. Первый параметр - Last, указывающий на последний подэлемент группы (фактически он является первым включенным в группу подэлементом). Вторым является параметр Current, указывающий на текущий выделенный подэлемент (см. п. 1.6). Если группа не имеет ни одного подэлемента, параметры Last и Current равны nil.

Основной элемент группы инициализируется конструктором Init:

```
constructor tGroup.Init(var Bounds: tRect);
```

где Bounds - размеры объекта.

После инициализации, но до включения в группу подэлементов, устанавливаются параметры, приведенные в табл. 1.2.

Таблица 1.2. Параметры объекта tGroup при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable+ofBuffered
DragMode	dmLimitLoY
GrowMode	0
EventMask	$FFFF
State	sfVisible
HelpCtx	hcNoContext
Next	nil
Owner	nil
Last	nil
Current	nil

Если после инициализации группа помещается в другую группу, устанавливаются соответствующие значения параметров Next и Owner.

Любой подэлемент группы должен быть включен в эту группу с помощью метода Insert:

```
procedure tGroup.Insert(P: pView);
```

где P - размещаемый объект,

или другого метода, включающего в себя метод Insert, и может быть из нее исключен с помощью метода Delete:

```
procedure tGroup.Delete(P: pView);
```

где P - исключаемый объект,

или другого метода, включающего в себя метод Delete (см. пп. 2.3, 2.8). При включении и исключении подэлементов в группу соответствующим образом изменяются параметры Last и Current. Параметр Last принимает значение указателя на первый размещенный в группу подэлемент или nil, если группа не имеет подэлементов. Параметр Current принимает значение указателя на последний размещенный в группу подэлемент, который может быть выделен, или nil, если группа не имеет таких подэлементов. Изменить значение параметра Current можно затем с помощью метода SelectNext (см. п. 1.6).

Последовательность включения подэлементов в группу очень важна, т. к. от нее зависит порядок работы с подэлементами. Следует начинать расположение подэлементов с левой верхней части основного элемента и далее идти вправо и вниз.

Включенные в группу подэлементы образуют так называемый Z-порядок, и среди них можно выделить первый, последний и промежуточные элементы (рис. 1.6).

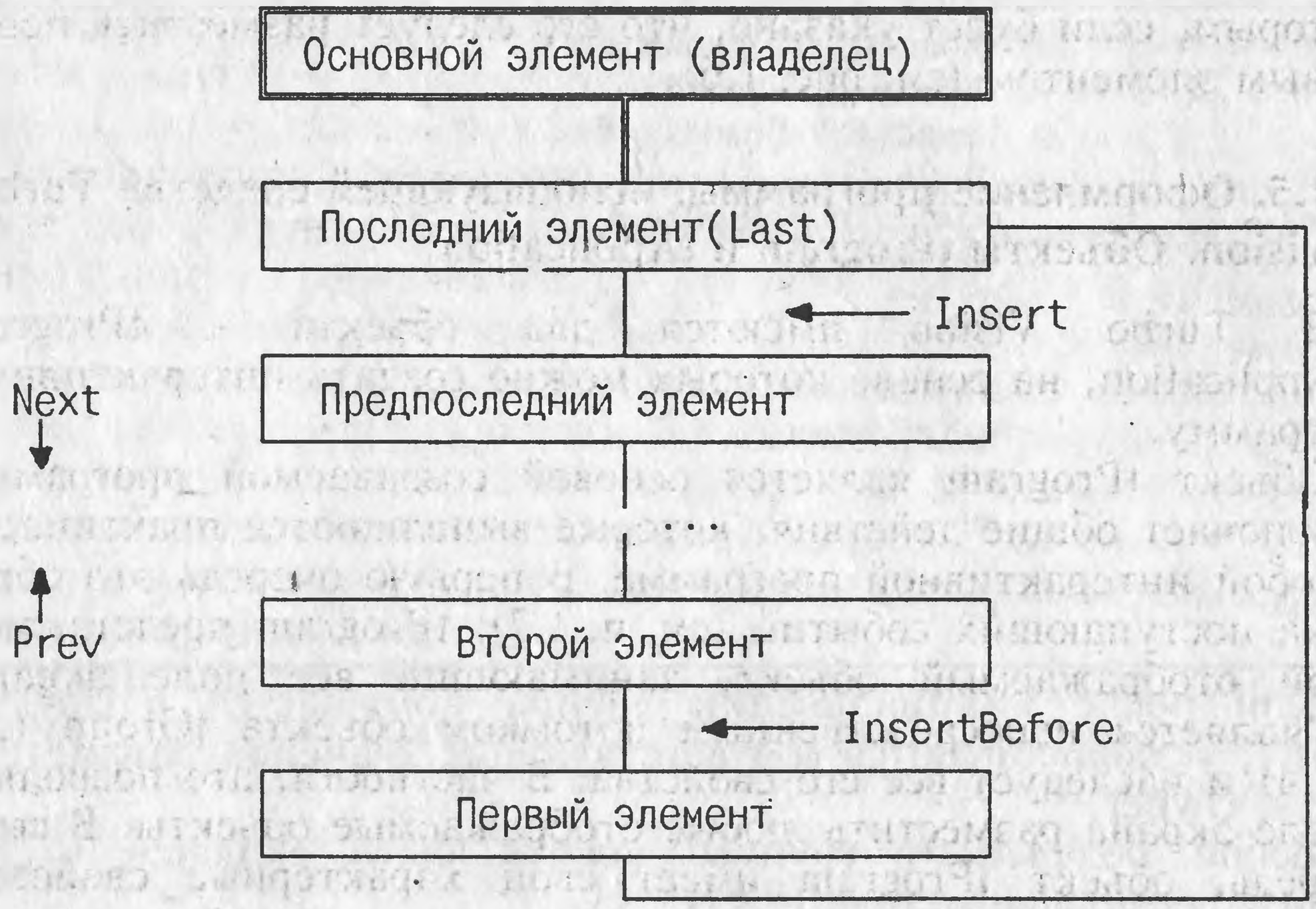


Рис. 1.6. Структура группы

Как указывалось в п. 1.3, у каждого отображаемого объекта есть поле Next, указывающее на следующий подэлемент группы. Поле Next последнего подэлемента указывает на предыдущий подэлемент, поле Next предыдущего подэлемента указывает на его предыдущий подэлемент и т. д. У первого подэлемента это поле указывает на последний подэлемент группы. Таким образом, все подэлементы группы образуют циклическую структуру. У каждого подэлемента группы поле Owner (владелец) указывает на основной элемент группы.

Поля, указывающего на предыдущий подэлемент в группе, у отображаемого объекта нет, однако у него есть метод Prev, который формирует указатель на такой предыдущий подэлемент. Этот метод даст указатель на первый подэлемент, если определяется предыдущий подэлемент последнего (Last) подэлемента.

Для помещения подэлемента в группу можно воспользоваться либо методом Insert, либо методом InsertBefore. Метод Insert (см. выше) размещает подэлемент между последним и предпоследним элементами группы. Метод InsertBefore размещает подэлемент перед указанным в методе подэлементом:

```
procedure tGroup.InsertBefore(P, Target: pView);
```

где P - размещаемый объект;
Target - объект, перед которым размещается новый объект.

Так, этот метод разместит новый подэлемент между первым и вторым, если будет указано, что его следует разместить перед первым элементом (см. рис. 1.6).

1.5. Оформление программы, использующей средства Turbo Vision. Объекты tProgram и tApplication

В Turbo Vision имеются два объекта - tProgram и tApplication, на основе которых можно создать интерактивную программу.

Объект tProgram является основой создаваемой программы и включает общие действия, которые выполняются практически в любой интерактивной программе. В первую очередь это обработка поступающих событий (см. п. 1.7). tProgram представляет собой отображаемый объект, занимающий все поле экрана. Он является непосредственным потомком объекта tGroup (см. п. 1.4) и наследует все его свойства. В частности, это позволяет в поле экрана разместить любые отображаемые объекты. В свою очередь, объект tProgram имеет свои характерные свойства, которые позволяют создать программу как нечто целое.

Инициализируется объект tProgram конструктором Init:

```
constructor tProgram.Init;
```

После инициализации устанавливаются значения параметров, приведенные в табл. 1.3.

Таблица 1.3. Параметры объекта tProgram при инициализации

Параметр	*Значение*
Origin	(0, 0)
Size	(ScreenWidth, ScreenHeight)
Cursor	(0, 0)
Options	0
DragMode	dmLimitLoY
GrowMode	0
EventMask	$FFFF
State	sfVisible+sfSelected+sfFocused+ sfModal+sfExposed
HelpCtx	hcNoContext
Next	nil
Owner	nil
Last	DeskTop
Current	DeskTop

В этой таблице ScreenWidth и ScreenHeight - параметры модуля Drivers, содержащие размеры экрана в текстовом режиме. Параметр DeskTop (модуль App) - указатель на рабочую область программы. Указатель на созданный объект tProgram помещается в переменную Application (модуль App).

Важным методом объекта tProgram является метод Run:

```
procedure tProgram.Run; virtual;
```

представляющий собой цикл обработки событий (см. п. 1.7) и выполняющий, таким образом, основные действия, связанные с работой программы.

Как уже сказано, объект tProgram позволяет разместить в поле экрана любые отображаемые объекты. Их можно помещать в поле экрана произвольным образом, однако принята следующая система их расположения: в верхней строке экрана размещается строка меню (тип tMenuBar или его потомок), в нижней строке экрана размещается строка состояния (тип tStatusLine или его потомок), остальная часть экрана отдается под рабочую область (тип tDeskTop или его потомок). Эти три объекта (tMenuBar, tStatusLine и tDeskTop) также являются отображаемыми объектами, причем объект tDeskTop является к тому же потомком объекта tGroup (см. п. 1.4).

Объекты типа tMenuBar, tStatusLine и tDeskTop размещаются на экране автоматически при инициализации программы методом Init. Для размещения меню, строки состояния и рабочей области вызываются методы InitMenuBar, InitDeskTop и InitStatusLine. Методы InitMenuBar и InitStatusLine переопределяются (см. пп. 3.1 и 6.1), метод же InitDeskTop обычно не меняется (его следует переопределить, если у программы нет меню или строки состояния, с тем, чтобы соответствующим образом изменить его размеры). При создании стандартным образом меню, строки состояния и рабочей области указатели на эти объекты размещаются в переменных MenuBar, StatusLine и DeskTop соответственно. Именно в объекте типа tDeskTop и размещаются обычно все отображаемые объекты пользователя (окна, диалоговые окна и т. д.).

Меню и строка состояния будут рассмотрены в дальнейшем (в пп. 3 и 6 соответственно). Здесь же рассмотрим рабочую область. Инициализируется рабочая область конструктором Init:

```
consrtuctor tDeskTop.Init((var Bounds: tRect);
```

где Bounds - размеры рабочей области.

При инициализации помимо задания начальных значений параметров создается также объект-фон, заполняющий всю

рабочую область, указатель на который помещается в переменной рабочей области Background. При этом задаются значения параметров рабочей области, приведенные в табл. 1.4.

Таблица 1.4. Параметры объекта tDeskTop при инициализации

Параметр	*Значение*
Origin	(0, 1)
Size	(Application^.Size.X, Application^.Size.Y-2)
Cursor	(0, 0)
Options	ofBuffered+ofSelectable
DragMode	dmLimitLoY
GrowMode	gfGrowHiX+gfGrowHiY
EventMask	$FFFF
State	sfVisible+sfSelected+sfFocused+ sfExposed
HelpCtx	hcNoContext
Next	Зависит от наличия меню и строки состояния
Owner	Application
Last	Backdround
Current	nil

Объект tApplication является непосредственным потомком объекта tProgram и наследует все его свойства. Кроме этого, он предназначен для создания оболочки программы, включающей систему управления памятью, видеосистему, систему обработки событий и т. д., которые могут быть инициализированы с помощью конструктора Init этого объекта:

```
constructor tApplication.Init;
```

а ликвидированы с помощью деструктора Done:

```
destructor tApplication.Done; virtual;
```

При инициализации этого объекта его параметры принимают те же значения, что и у его предка tProgram (фактически это есть те же самые параметры).

Программист обычно формирует свою программу именно на основе объекта tApplication, в который уже включены все общие для интерактивных программ свойства и действия. Лишь в случае использования собственной оболочки можно создавать программу на основе объекта tProgram. Дополнив этот объект соответствующими методами, можно задать все необходимые в программе специфические действия.

Сформированная на основе объекта tApplication программа обычно выполняет последовательно три операции: инициализа-

цию программы, обработку событий (см. п. 1.7) и завершение работы программы.

Инициализация программы, если не требуется каких-либо специальных дополнительных действий, представляет собой простое обращение к конструктору Init объекта tApplication (или его потомка), который инициализирует оболочку программы, после чего обращается к конструктору объекта tProgram, который, в свою очередь, выполняет действия, связанные с инициализацией программы как таковой, устанавливая исходные значения ее параметров и размещая на экране меню, строку состояния и рабочую область.

Обработка событий осуществляется методом Run объекта tProgram (этот метод, естественно, доступен и объекту tApplication или его потомку).

Завершение работы программы выполняет деструктор Done.

Обычно тело любой программы и состоит из обращения к этим трем методам.

При создании интерактивной программы целесообразно начать ее написание с объекта-приложения - потомка объекта tApplication, в котором следует первоначально предусмотреть лишь конструктор Init, деструктор Done, методы обработки событий HandleEvent (см. п. 1.7), формирования меню InitMenuBar (см. п. 3.1) и формирования строки состояния InitStatusLine (см. п. 6.1). Иногда некоторые из этих методов не создаются, если можно воспользоваться стандартным методом предка tApplication (например, методом Done) или если в нем нет необходимости (например, метод InitStatusLine, если у программы нет строки состояния).

Исходя из вышесказанного, создаваемая программа-калькулятор будет упрощенно выглядеть следующим образом.

```
program Calculator;                                                    { 1}
type                                                                   { 9}
   { Объект - приложение }
   pMyApp = ^tMyApp;                                                   { 70}
   tMyApp = object(tApplication)                                       { 71}
     constructor Init;                                                 { 72}
     procedure HandleEvent(var Event: tEvent); virtual;                { 75}
     procedure InitMenuBar; virtual;                                   { 76}
     procedure InitStatusLine; virtual;                                { 77}
   end;                                                                { 78}

   var MyApp: tMyApp;                                                  {759}
```

```
{ Основная часть программы }
begin                                                    {760}
   MyApp.Init;                                           {761}
   MyApp.Run;                                            {762}
   MyApp.Done;                                           {763}
end.                                                     {764}
```

Конкретное содержание методов объекта tMyApp будет рассмотрено ниже. В дальнейшем он будет также дополнен рядом других методов.

При определении новых объектов целесообразно одновременно определять указатель на этот объект (в данном примере pMyApp - указатель на tMyApp). Это совершенно необходимо, если соответствующие объекты будут располагаться в динамической области памяти. Использование указателей позволяет проще работать с такими объектами.

1.6. Структура отображаемых объектов программы

Исходя из вышесказанного, можно сделать вывод, что все отображаемые объекты интерактивной программы представляют древовидную структуру, корнем которой является сама программа - потомок объекта tApplication (см., например, рис. 1.7).

Такая структура может иметь на экране вид, изображенный, например, на рис. 1.8.

На этих рисунках основным отображаемым объектом является программа - потомок tApplication. Он является группой, в которую входят меню (потомок tMenuBar), рабочая область (тип tDeskTop) и строка состояния (потомок tStatusLine). Рабочая область, в свою очередь, является группой, в которую входят окно (см. п. 4.1) и диалоговое окно (см. п. 4.3). Окно и диалоговое окно также являются группами, включающими различные подэлементы (рамки, строки ввода, кнопки).

Каждый отображаемый объект может быть в одном из двух состояний - пассивном и выделенном. На рис. 1.7 выделенные объекты обрамлемы двойной рамкой. В каждой группе может быть только один выделенный объект (или ни одного, если сама группа не является выделенным объектом). Указатель на выделенный элемент находится в переменной Current объекта-группы. Если какой-то объект группы становится выделенным, предыдущий выделенный объект этой группы становится пассивным.

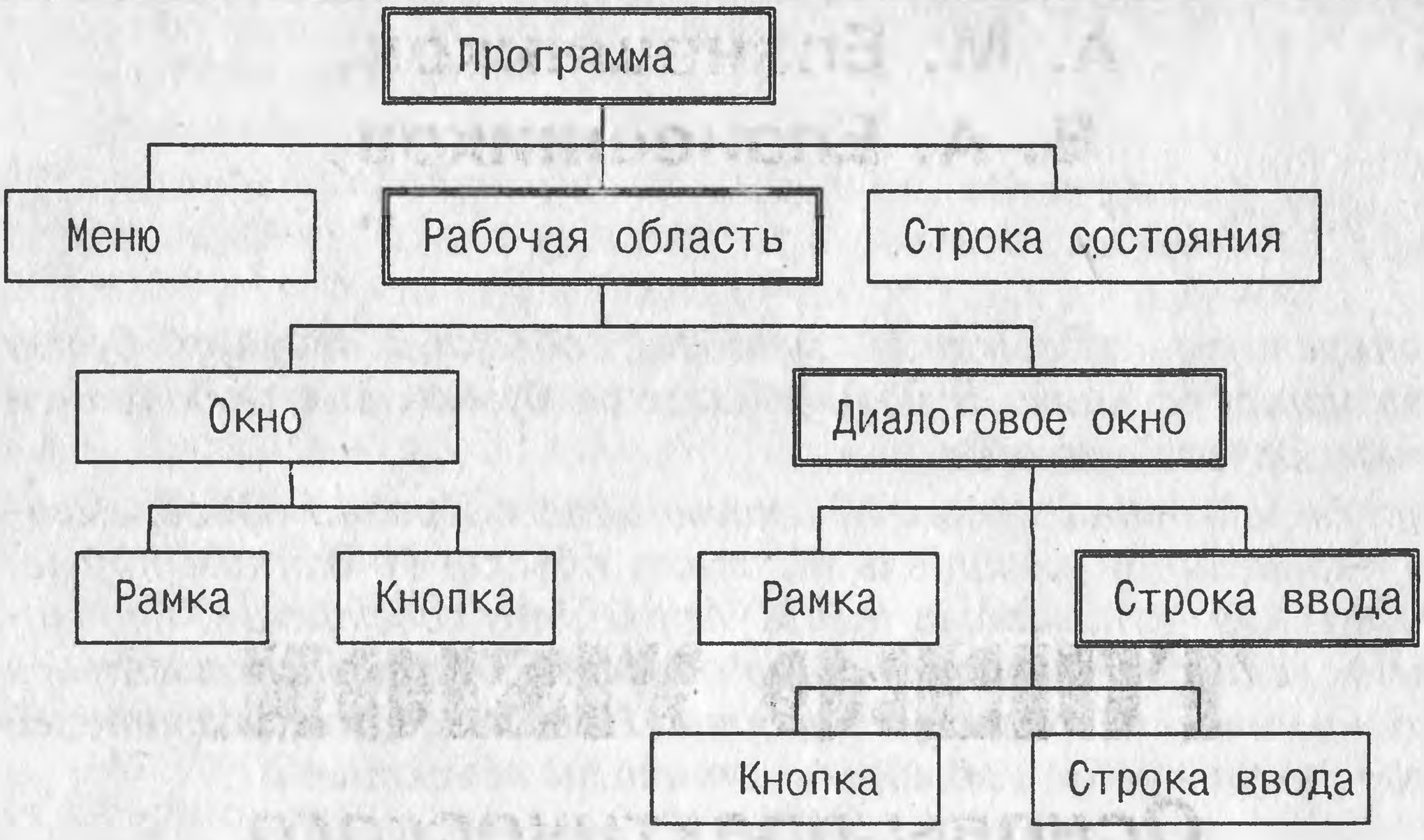


Рис. 1.7. Пример структуры отображаемых объектов программы

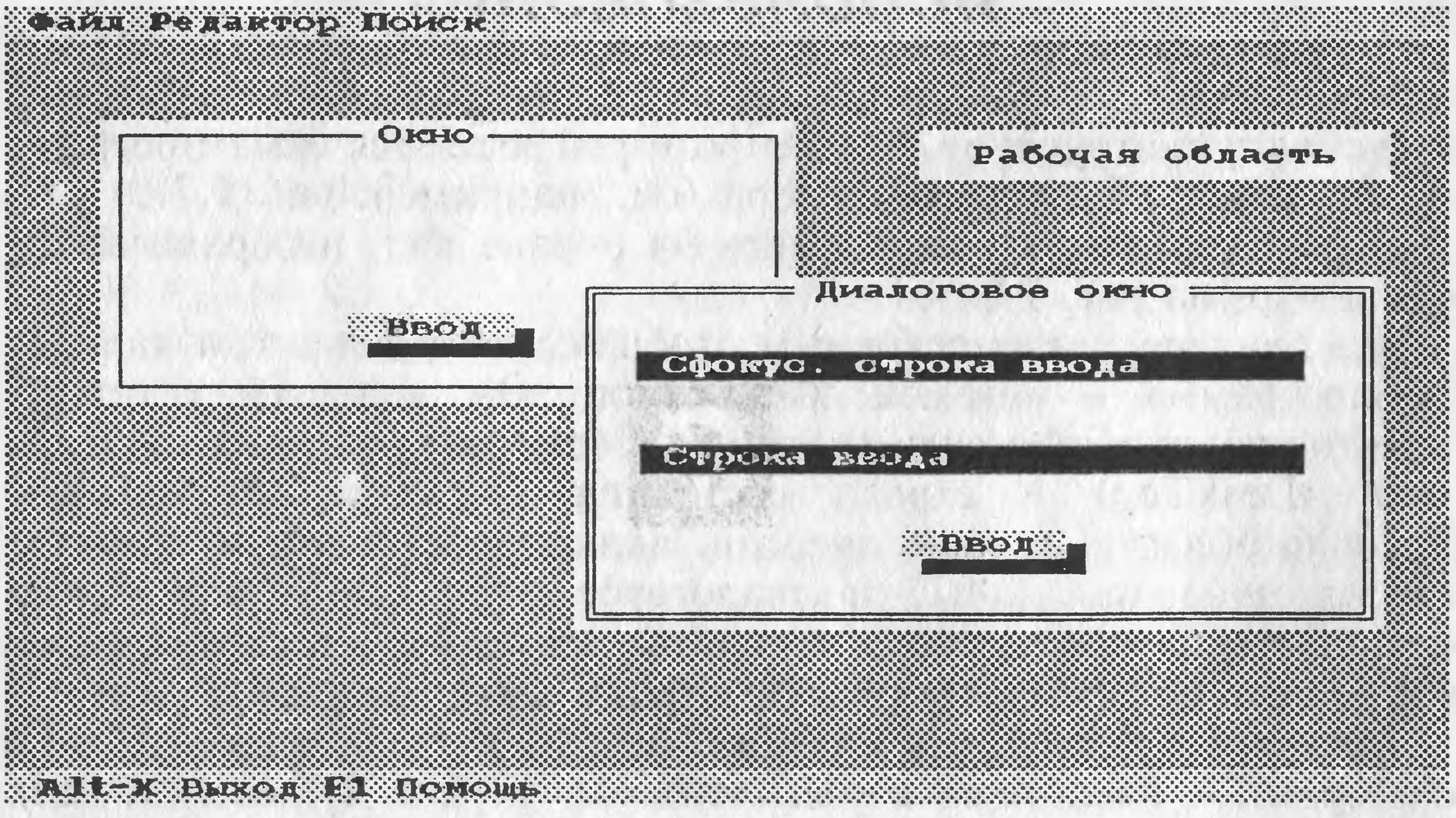


Рис. 1.8. Изображение на экране структуры объектов рис. 1.7.

Сделать выделенным предыдущий или последующий объект группы можно с помощью метода SelectNext:

```
procedure tGroup.SelectNext(Forwards: Boolean);
```

где Forwards - параметр, определяющий, какой объект следует выделить; если Forwards = True, выделяется следующий в Z-порядке объект группы, в противном случае выделяется предыдущий объект.

Этот метод не проверяет правильность введенных данных в текущем выделенном объекте (см. п. 8.1) и всегда выделяет другой элемент. Если необходимо проверять правильность введенных данных, следует вместо этого метода использовать метод FocusNext:

```
function tGroup.FocusNext(Forwards: Boolean): Boolean;
```

где Forwards - тот же параметр, что и у метода SelectNext;
результат - True, если произошло переключение на другой объект, и False - в противном случае.

Этот метод не всегда осуществляет переключение на другой элемент, а только в том случае, если в текущем элементе данные введены правильно. Результат функции говорит о результате переключения.

Выделенный объект - это тот объект, с которым в настоящее время происходит работа. Так, в рассматриваемом примере выделенной является рабочая область - с окнами, расположенными в ней, пользователь может взаимодействовать. Меню и строка состояния в этом случае являются пассивными. Если выделить меню, то рабочая область становится пассивной и уже не будет возможности работать с окнами рабочей области до тех пор, пока рабочая область опять не станет выделенной.

Выделенные объекты образуют цепочку выделенных объектов (в рассматриваемом примере программа - рабочая область - диалоговое окно - строка ввода). Последний объект такой цепочки называется сфокусированным объектом. В рассматриваемом примере сфокусированным объектом является выделенная строка ввода (см. п. 4.4.1) - именно в эту строку ввода будут помещаться набираемые на клавиатуре символы. На экране сфокусированный элемент обычно выделяется особым цветом (или он весь изменяет свой цвет, или его отдельные детали). В некоторых случаях у выделенного элемента меняются отдельные детали, так у окна (см. п. 4.1) изменяется вид рамки вокруг этого окна.

Если выделенным объектом является окно или его потомок, то такой объект и его подэлементы называются активными. В данном примере диалоговое окно и его подэлементы являются активными объектами. Если же выделенным станет, например, меню, то активных объектов в программе просто не будет.

Отображаемые объекты могут находиться в одном из двух режимов: обычном и модальном. Тип режима задается способом размещения объекта в группе. Если объект является модальным, то все события, возникающие в программе (см. п. 1.7), обраба-

тываются только этим объектом и его подэлементами. В рассматриваемом примере если диалоговое окно находится в модальном режиме, то все события будут поступать ему и его подэлементам - рамке, строке ввода и кнопке. Кроме этой особенности наличие модального объекта приводит к тому, что нельзя выйти за пределы этого объекта и его подэлементов без снятия режима модальности (как правило, для этого требуется удалить модальный объект из группы, в которой он находится). Если в примере на рис. 1.7. диалоговое окно является модальным, то нельзя активизировать ни обычное окно, ни меню. Исключение составляет строка состояния, которая позволяет в ряде случаев работу с ней и при наличии модального объекта.

Если объект находится в обычном режиме, то можно выйти за пределы этого объекта и сделать выделенным какой-то другой объект.

Следует заметить, что сама программа как отображаемый объект является модальным объектом.

1.7. Обработка событий

При использовании объектно-ориентированного программирования все объекты являются в некотором смысле обособленными друг от друга и возникают определенные трудности в передаче информации от объекта к объекту. Тем не менее невозможно представить себе программу, где не осуществлялся бы обмен информацией между ее отдельными частями. Для того чтобы осуществить такую передачу информации между объектами, должен использоваться тот или иной механизм. В Turbo Vision таким механизмом является механизм обработки событий. При таком способе любая передаваемая информация оформляется в виде так называемого события, представляющего в точки зрения языка Паскаль запись, отдельные поля которой характеризуют те или иные свойства передаваемой информации. Тип этой записи - tEvent:

```
type
   tEvent = record
      What: Word;
      case Word of
         evNothing: ();
         EvMouse: (
            Buttons: Byte;
            Double: Boolean;
            Where: tPoint);
```

```
    evKeyDown: (
      case Integer of
        0: (KeyCode: Word);
        1: (CharCode: Char;
            ScanCode: Byte));
      end;
    evMessage: (
      Command: Word;
      case Word of
        0: (InfoPtr: Pointer);
        1: (InfoLong: Longint);
        2: (InfoWord: Word);
        3: (InfoInt: Integer);
        4: (InfoByte: Byte);
        5: (InfoChar: Char));
      end;
  end;
end;
```

Запись tEvent состоит из двух частей. Первая часть задает тип события (What), определяющий в первую очередь источник данного события. Под этот параметр выделяется 16-разрядное слово. Отдельные биты этого слова имеют следующие значения (в скобках указано имя константы, имеющей единицу в соответствующем разряде слова):

0-й бит (evMouseDown) - нажата клавиша мыши.
1-й бит (evMouseUp) - отпущена клавиша мыши.
2-й бит (evMouseMove) - мышь перемещена по поверхности.
3-й бит (evMouseAuto) - автоимпульсы от мыши при длительном нажатии клавиши.
4-й бит (evKeyDown) - нажата клавиша клавиатуры.
8-й бит (evCommand) - событие-команда.
9-й бит (evBroadcast) - событие-сообщение.

События-команды и события-сообщения формируются теми или иными объектами программы для передачи информации другим объектам. Основное отличие события-команды от события-сообщения заключается в механизме их обработки.

9-й бит соответствует так называемому стандартному сообщению, определенному в Turbo Vision. Помимо таких стандартных сообщений пользователь может ввести свои сообщения, использовав для них биты с 10-го по 15-й.

Помимо указанных констант в Turbo Vision используются несколько констант, соответствующих группам бит и определяющих источник создания события. Это следующие константы:

evNothing = $0000 - обработанное событие;
evMouse = $000F - событие от мыши;
evKeyboard = $0010 - событие от клавиатуры;
evMessage = $FF00 - событие от объекта (команда или сообщение).

Вторая часть записи tEvent задает информацию, передаваемую с событием. Для разных типов событий содержание информации и ее размер различны.

Для события от мыши задаются три параметра.

Параметр Buttons размером в 1 байт, определяющий, от какой клавиши - левой или правой - поступило событие. Если единица записана в нулевом бите (константа mbLeftButton = 1), то была нажата левая клавиша. Если единица записана в первом бите (константа mbRightButton = 2), то была нажата правая клавиша.

Параметр Double типа Boolean (размером в 1 байт), принимающий значение True, если от мыши поступила информация о двойном нажатии, и False - в противном случае.

Параметр Where типа tPoint (см. п. 1.2) - размером в два 16-разрядных слова, дающий текущие глобальные (относительно левого верхнего угла экрана) координаты курсора мыши.

Для событий от клавиш клавиатуры задается информация о нажатой клавише. Информацию можно рассматривать двояким образом: либо как 16-разрядный код клавиши (KeyCode), либо как совокупность 8-разрядного ASCII-кода символа (CharCode) и 8-разрядного скэн-кода (ScanCode) - фактически порядкового номера клавиши. В разных приложениях может потребоваться как первое, так и второе представление информации.

Константы, соответствующие кодам служебных клавиш и их комбинаций, как F1, Enter, Left, Alt+J, Ctrl+Del, Shift+Tab и т. д., имеют стандартные имена, начинающиеся буквами kb и включающие названия клавиш (kbF1, kbEnter, kbLeft, kbAltJ и т. д. - см. [2]).

Для событий от объектов задаются два параметра.

Параметр Command типа Word, задающий код команды (см. п. 1.8), которую необходимо выполнить при появлении данного события.

Второй параметр определяет передаваемую с событием информацию. Так как передаваемая информация может быть разнообразной по своему типу, предусмотрено несколько вариантов

типов передаваемой информации. Наиболее распространенный вариант - передача указателя (InfoPtr типа Pointer) на область, где находится информация. Кроме этого, для небольших объемов информации: - длинное целое (InfoLong типа Longint), беззнаковое целое (InfoWord типа Word), знаковое целое (InfoInt типа Integer), байт (InfoByte типа Byte), символ (InfoChar типа Char) - предусмотрена их непосредственная передача. Естественно, что для правильной работы программы потребитель информации должен знать, какого типа информация передается с событием.

Вопрос формирования событий (осуществляется формирование событий с помощью метода GetEvent) мы подробно рассматривать не будем - для стандартных случаев они формируются автоматически с помощью имеющихся средств Turbo Vision. Нестандартное формирование событий может потребоваться в специальных случаях, особенно при создании собственных событий-сообщений или при нестандартном способе создания события. В этом случае придется сформировать соответствующий метод GetEvent (как правило, для объекта tApplication). Мы кратко рассмотрим случай создания собственного метода GetEvent позже, когда будем знакомиться с организацией справочной системы в Turbo Vision (см. п. 5).

В рассматриваемом примере программы-калькулятора приведен собственный метод формирования событий GetEvent объекта tMyApp (лл. 73, 667 - 707), который позволяет нестандартным образом обрабатывать события, предназначенные для справочной системы (дает возможность обращаться к справочной системе даже из модальных объектов, что в обычном случае затруднительно).

Обработку же событий следует рассмотреть поподробнее, т. к. обычно приходится создавать свою собственную обработку. Ее последовательность зависит от типа события и режимов отображаемых объектов.

Обработка события всегда начинается с текущего модального объекта (см. п. 1.6), после чего, если необходимо, оно передается тем или иным подэлементам этого модального объекта. Если единственным модальным объектом является сама программа (напомним, что указатель на этот объект находится в переменной Application), обработка начинается с него.

В зависимости от последовательности обработки событий объектами все события можно разбить на три группы.

I группа включает все события от мыши. Эти события воздействуют в зависимости от координат курсора мыши. Модальный объект, воспринявший такое событие, передает его своим

подэлементам в Z-порядке (см. п. 1.4), отыскивая подэлемент, в который попадает курсор мыши. Этот подэлемент, если он существует и является группой, в свою очередь, также передает это событие своим подэлементам в Z-порядке и т. д. пока этот поиск не остановится из-за отсутствия соответствующего подходящего подэлемента. Если соответствующий объект найден, то он будет обрабатывать событие, в противном случае событие обрабатываться не будет.

Для примера, изображенного на рис. 1.7 и 1.8, если курсор мыши находится в видимом поле окна (не диалогового) и нажата клавиша мыши, обработка события возможна двояким образом в зависимости от того, является ли диалоговое окно модальным или нет. Если оно является модальным, то обработка события, начавшись с этого модального окна, не найдет своего потребителя, т. к. ни само диалоговое окно, ни его подэлементы не занимают место экрана, где находится курсор мыши. В этом случае событие обработано не будет. Если же диалоговое окно не является модальным, то обработка начинается с объекта, на который указывает переменная Application, являющегося в этом случае единственным модальным объектом. Этот объект выясняет, что его подэлементом, содержащим курсор мыши, является рабочая область, и передает событие этому объекту. Рабочая область просматривает свои подэлементы и определяет, что курсор находится в поле окна (не диалогового), и передает событие ему. Оно, в свою очередь, просматривает свои подэлементы, и, если курсор находится в поле кнопки, событие передается ей, и она обрабатывает это событие. В противном случае событие обрабатывается самим окном.

II группа предназначена для сфокусированного объекта (см. п. 1.6) и включает события от клавиш клавиатуры (evKeyDown) и команды (evCommand). Эти события передаются по цепочке выделенных объектов непосредственно сфокусированному объекту. Если сфокусированный объект "не знает", как обработать соответствующее событие, то оно передается назад по цепочке выделенных объектов к модальному объекту.

В примере, изображенном на рис. 1.7 и 1.8, в случае появления события от клавиатуры оно будет обработано сфокусированной строкой ввода диалогового окна.

В ряде случаев требуется, чтобы несфокусированный объект обработал событие данной группы. Такая возможность имеется, если использовать флаги опережающей и запаздывающей реакции на событие - флаги ofPreProcess и ofPostProcess (см. п. 1.3). Дело заключается в том, что при последовательной передаче

события по цепочке выделенных объектов каждый из выделенных объектов сначала просматривает в Z-порядке все свои подэлементы, и, если у кого-либо из них будет установлен флаг ofPreProcess, событие передается на обработку этому объекту. Таким образом, сфокусированный элемент будет обрабатывать событие только в том случае, если оно не будет обработано объектом с установленным флагом ofPreProcess. Если в этом случае и сфокусированный объект не обработает событие, то при обратной передаче события по цепочке выделенных объектов к модальному объекту опять-таки просматриваются подэлементы всех выделенных элементов цепочки и теперь уже проверяется установка флага ofPostProcess. При наличии такого объекта событие передается на обработку ему.

В методе Init объекта - индикатора калькулятора (лл. 133-141) установлен флаг ofPreProcess (л. 136) для того, чтобы он реагировал на нажатие клавиш клавиатуры, хотя сам индикатор никогда выделенным (см. п. 1.6) не бывает, т. к. у него сброшен флаг ofSelectable (л. 136).

III группа включает события-сообщения (стандартные и создаваемые пользователем). Эти события передаются всем подэлементам модального объекта в Z-порядке. Те, в свою очередь, передают их своим подэлементам в Z-порядке и т. д. до тех пор, пока это событие не будет обработано каким-либо объектом или же не будут просмотрены все объекты.

При обработке события сначала выполняются соответствующие действия у объекта, которому предназначено событие, после чего событие "очищается" стандартным методом ClearEvent:

```
procedure tView.ClearEvent(var Event: tEvent);
```

где Event - текущее "очищаемое" событие.

При этом поле What (см. выше) принимает значение evNothing, а параметр InfoPtr - значение @Self - указатель на объект, который обработал данное событие, что часто бывает важно для того, чтобы определить, какой объект в конце концов обработал событие. Если событие не "очистить", то оно будет передаваться дальше в соответствии с механизмом его обработки и может быть обработано еще каким-либо объектом.

Если не нашлось подходящего объекта для обработки события, вызывается стандартный метод EventError объекта tGroup, который не выполняет никаких действий, но который можно переопределить, если необходимо выполнить какие-либо конкретные действия, если событие не было обработано.

В ряде случаев необходимо, чтобы тот или иной отображаемый объект реагировал только на определенные виды событий, а другие события игнорировал. Для этой цели используются флаги обработки событий, размещенные в переменной EventMask типа Word, имеющейся у каждого отображаемого объекта - потомка tView. Назначение битов у этой переменной то же, что и у поля What события tEvent. Для того, чтобы объект реагировал на то или иное событие, следует в соответствующем бите переменной EventMask установить единицу. Для стандартных отображаемых объектов, исходя из их назначения и функционирования, уже при их создании в программе задаются те или иные биты этой переменной.

У индикатора калькулятора, в котором отображаются вводимые числа и результаты вычислений, в л. 137 установлены флаги обработки событий от клавиатуры (для того, чтобы калькулятор мог реагировать на соответствующую клавишу клавиатуры) и сообщений (чтобы калькулятор мог реагировать на "нажатие" кнопок калькулятора с помощью мыши - см. рис. 1.1; при своем нажатии кнопки вырабатывают соответствующие сообщения - см. п. 4.4.5).

Изменить стандартную последовательность обработки событий можно и с помощью функции Message модуля Views:

```
function Message(Receiver: pView; What, Command:
                                  Word;InfoPtr: Pointer): Pointer;
```

где Receiver - получатель информации (отображаемый объект, который будет обрабатывать событие);
What - тип события, как правило, из категории evMessage;
Command - код команды;
InfoPtr - указатель на передаваемую информацию.

Функция сначала формирует событие с параметрами What, Command и InfoPtr, а затем передает его на обработку получателю информации (Receiver). Если событие обработано успешно, функция возвращает значение поля InfoPtr события, в противном случае она принимает значение nil.

Обрабатываются события методом HandleEvent:

```
procedure <отображаемый объект>.HandleEvent(
                                  var Event: tEvent); virtual;
```

где Event - обрабатываемое событие,
который имеется у каждог отображаемого объекта. Для стандартных отображаемых объектов (окон, диалоговых окон, строк ввода, кнопок и т. д., включая абстрактные отображаемые объекты, например типа tView, tGroup и т. д.) этот метод обрабаты-

вает вполне определенный набор событий. Естественно, эти же события будут обрабатываться и их потомками. Метод является виртуальным и может быть, в случае необходимости, переопределен для любого потомка отображаемого объекта, что обычно и делается при создании своих конкретных отображаемых объектов. При этом следует в новом методе сначала вызвать наследуемый метод предка, а затем ввести необходимые дополнительные действия (эта последовательность в принципе может быть и другой), например:

```
tMyVisualObject.HandleEvent(var Event:tEvent); virtual;
begin
   inherited HandleEvent(Event);      {Обращение к методу предка}
   if Event.What = evCommand and            {Собственная обработка}
      Event.Command = cmMyCommand then
      . . .
end;
```

Как уже указывалось выше, обработка события методом HandleEvent должна завершаться "очисткой" события стандартным методом ClearEvent.

У индикатора калькулятора метод HandleEvent (лл. 313-330) сначала вызывает наследуемый от tView метод HandleEvent, обрабатывающий события от мыши, а затем обрабатывает события от клавиатуры и сообщения; любая такая обработка заканчивается "очисткой" события с помощью обращения к стандартному методу ClearEvent.

1.8. Команды

Как указывалось выше, события от объектов (команды или сообщения) имеют в качестве одного из параметров параметр Command типа Word, задающий код команды, которую необходимо выполнить при обработке этого события.

Все используемые в Turbo Vision команды делятся на четыре группы:

- команды, зарезервированные за системой и которые можно маскировать методом DisableCommands объекта tView и демаскировать методом EnableCommands объекта tView, - коды с 0 до 99;
- команды, которые может вводить программист и которые можно маскировать и демаскировать аналогично первой группе, - коды со 100 до 225;

- команды, зарезервированные за системой, но которые нельзя маскировать и демаскировать, как команды первой группы, - коды с 256 до 999;
- команды, которые может вводить программист, но которые нельзя маскировать и демаскировать, - коды с 1000 до 65535.

Программист может использовать зарезервированные за Turbo Vision команды, не переопределяя их. Более того, команды этой категории имеют стандартные названия, и средства пакета определенным образом реагируют на них. Так, например, команда cmClose вызывает закрытие активного окна, а cmMenu - активизирует меню и т. д. Все стандартные команды (их перечень и назначение можно посмотреть в [2]) начинаются с префикса cm. По-видимому, так же целесообразно оформлять и свои собственные команды (см., например, лл. 5-7).

Команды с кодами от 0 до 255 могут маскироваться и демаскироваться (т. е. может запрещаться или разрешаться выполнение этих команд). Все эти команды составляют множество типа tCommandSet:

```
type tCommandSet = set of Byte;
```

Для запрещения команд используется метод DisableCommands объекта tView:

```
procedure tView.DisableCommands(Commands: tCommandSet);
```

где Commands - множество запрещаемых команд.

Для разрешения команд используется метод EnableCommands объекта tView:

```
procedure tView.EnableCommands(Commands: tCommandSet);
```

где Commands - множество разрешаемых команд.

В методе EnableOrDisableCommands объекта tCalculator (лл. 466-475) используются методы EnableCommands (л. 472) и DisableCommands (лл. 474), которые в зависимости от системы счисления разрешают или запрещают команды, связанные с кнопками (см. п. 4.4.5), которые соответствуют цифрам.

2. ПОРЯДОК РАБОТЫ С ОТОБРАЖАЕМЫМИ ОБЪЕКТАМИ

Работа с отображаемыми объектами осуществляется в несколько этапов, выполняемых в определенной последовательности. На этих этапах отображаемый объект описывается, создается, помещается в соответствующую группу, задаются его параметры, производится работа с объектом, получаются введенные параметры, наконец, объект удаляется.

Хотя при работе с любыми отображаемыми объектами последовательность работы практически одна и та же, имеются различия в использовании методов для модальных и немодальных объектов. Дело в том, что, как указывалось выше (см. п. 1.6), при работе с модальным объектом нельзя выйти за его пределы не сняв его модальность, которая, в свою очередь, обычно снимается с удалением объекта. В связи с этим при работе с такими объектами используются более сложные, комплексные методы, включающие в себя практически всю перечисленную последовательность действий.

2.1. Описание отображаемого объекта

Если отображаемый объект не является стандартным, его следует описать - ввести необходимые параметры и методы на основе правил объектного программирования. Методы могут быть четырех видов: конструкторы, деструкторы, переопределенные методы предка и методы, специфические для данного объекта.

Конструктор позволяет инициализировать отображаемый объект: задать исходные параметры, создать его подэлементы и т. д. Кроме этого, он создает таблицу виртуальных методов, без которой нельзя будет обратиться к этим методам. Таким образом, любой объект, имеющий виртуальные методы, должен иметь конструктор.

Деструктор выполняет действия, обратные действиям конструктора, и используется при удалении отображаемого объекта, имеющего виртуальные методы. Сам деструктор также является виртуальным.

Переопределенный метод предка позволяет, используя возможности исходного метода, модифицировать их, добавить в них некоторые дополнительные свойства, специфические для данного объекта. Переопределенный метод должен быть виртуальным.

Специфический для данного объекта метод - это метод, который не имеет аналогов у предка данного объекта.

В качестве примера рассмотрим отображаемый объект - калькулятор (лл. 44-50).

В этом объекте, являющемся потомком стандартного диалогового окна (см. п. 4.3), введен дополнительный параметр CalcDisplay - указатель на индикатор калькулятора, который отображает вводимые числа и результаты вычислений, и методы: Init - конструктор, инициализирующий калькулятор, EnableOrDisableCommands - специфический для данного объекта метод, разрешающий или запрещающий обработку кнопок калькулятора с цифрами в зависимости от выбранной системы счисления, HandleEvent - переопределенный метод предка, обрабатывающий события, который в дополнение к методу предка (см. л. 537) обрабатывает команды задания параметров калькулятора (cmParameters - л. 542) и просмотра стека калькулятора (cmStack - л. 543).

Собственный деструктор в данном случае не создается, т. к. достаточно для удаления этого объекта использовать деструктор предка.

2.2. Создание отображаемого объекта

Если отображаемый объект не размещен постоянно в сегменте данных (не задана соответствующая переменная конкретного типа-объекта), необходимо создать соответствующий объект. Модальные объекты, как правило, не размещаются постоянно в сегменте данных (за исключением, быть может, самого объекта-программы) и создаются только на время работы с ними; немодальные объекты могут быть и одного, и другого вида, но чаще все-таки также не размещаются постоянно в сегменте данных. Создается объект с помощью стандартной подпрограммы New, использующей конструктор Init.

Содержание конструктора Init существенно зависит от типа объекта, его предназначения, однако перед заданием конкретных параметров объекту, как правило, следует вызвать наследуемый от предка конструктор Init.

У рассмотренного выше калькулятора конструктор Init (лл. 435-465) сначала вызывает наследуемый от диалогового окна конструктор (л. 440), затем задает флаги свойств (л. 441 - см. п. 1.3), формирует кнопки управления (лл. 442-460), индикатор калькулятора (лл. 461-463) и разрешает использование кнопок с цифрами от 0 до 9 - десятичная система счисления (л. 464).

Сам калькулятор создается с помощью процедуры CalcNew (лл. 716-726) при обработке команды создания нового калькулятора (cmCalcNew - л. 733).

2.3. Помещение отображаемого объекта в группу

После создания отображаемого объекта его следует поместить в соответствующую группу: если он является элементом окна, то в соответствующее окно, если же он сам является окном, то в рабочую область DeskTop (см. п. 1.5).

У каждого объекта-группы имеются методы Insert и InsertBefore, с помощью которых можно поместить немодальный отображаемый объект в группу (см. п. 1.4).

Для размещения немодального окна в рабочей области вместо метода Insert лучше воспользоваться методом InsertWindow:

```
function tProgram.InsertWindow(P: pWindow): pWindow;
```

где P - указатель на размещаемое окно;
результат - указатель на размещенное окно, если окно размещено, и nil - если не размещено.

Метод является функцией, которая перед размещением окна проверяет допустимость такой операции (имеется ли необходимая область памяти, и правильно ли сконструированы окно и его компоненты), и, если операция допустима, окно размещается в динамической области памяти, а в качестве результата возвращается указатель на него, в противном случае результат функции равен nil. При неправильной работе с окном оно удаляется из динамической области памяти.

В примере с калькулятором конструктор Init (лл. 435-465), создавая кнопки управления и индикатор калькулятора, завершает свои действия операцией размещения соответствующего объекта в поле калькулятора с помощью метода Insert (лл. 459, 463).

При создании нового калькулятора с помощью процедуры CalcNew (лл. 716-726) для его размещения в рабочей области используется метод InsertWindow (л. 725). В рассмот-

ренных случаях используются методы, применяемые для немодальных объектов; подэлементы окна обычно являются немодальными, т. к. только тогда с ними можно работать одновременно, а сам калькулятор также является немодальным объектом, т. к. предполагается возможность одновременного открытия нескольких калькуляторов с переходом от одного к другому без их закрытия.

2.4. Задание параметров отображаемому объекту

В Turbo Vision принят следующий стандартный способ передачи параметров. Для объекта-группы или его потомка формируется переменная типа-записи, отдельные поля которой представляют собой параметры для подэлементов группы. Поля должны идти в той же последовательности, в какой подэлементы включались в группу. Для каждого типа подэлемента параметры задаются по-своему. Для стандартных элементов эти параметры должны быть следующими:

- строка ввода (см. п. 4.4.1) - строка того же размера;
- триггерный список (см. п. 4.4.4) - число типа Word, в котором каждый бит определяет состояние соответствующего элемента списка;
- селективный список (см. п. 4.4.4) - число типа Word, определяющее порядковый номер выделенного элемента списка;
- триггерный список с несколькими состояниями - (см. п. 4.4.4) информация размером в 4 байта, определяющая состояние каждого элемента списка; в отличие от обычного триггерного списка под каждый элемент может отводиться 1, 2, 4 и 8 бит информации;
- список коллекции строк (см. п. 4.4.7) - запись размером в 6 байт, первое поле которой является указателем на коллекцию просматриваемых строк (см. п. 8.2), а второе - число типа Word, определяющее порядковый номер выделенного элемента списка;
- параметрический текст (см. п. 4.4.6) - по 4 байта на каждый параметр.

Для объекта, не являющегося группой, передается параметр, соответствующий типу объекта, так же как для того или иного подэлемента группы.

Для остальных стандартных элементов параметры не задаются. Для нестандартных элементов пользователь сам определяет вид и размер параметра.

Задать параметры немодальному отображаемому объекту лучше всего с помощью метода SetData, который имеется у любого отображаемого объекта:

```
procedure <отображаемый объект>.SetData (var Rec); virtual;
```

где Rec - передаваемая запись с данными.

Использовать этот метод можно и непосредственно в конструкторе Init, и после того, как объект будет создан и инициализирован. Для стандартных объектов этот метод работает с параметрами, указанными выше; если же необходимо передавать параметры другой структуры (например, передавать в строку ввода число не в символьной форме, а в двоичной) или другого размера, данный метод следует переопределить.

Пример использования метода SetData см в п. 4.4.7. (л. 656 видоизмененной программы).

2.5. Работа с отображаемым объектом

Работа с отображаемым объектом осуществляется на основе метода обработки событий HandleEvent (см. п. 1.7). В методе HandleEvent должны быть обработаны все события, которые предназначаются данному объекту.

Метод HandleEvent индикатора калькулятора (лл. 313-330) сначала вызывает наследуемый от объекта tView метод HandleEvent, который обрабатывает события от мыши, затем обрабатывает события от клавиш клавиатуры (т. к. установлен флаг ofPreProcess - см. п. 1.7), за исключением событий от клавиши табуляции Tab или комбинации клавиш Shift+Tab, и, наконец, обрабатывает события-команды с кодами 100, 148...159, 165...170, которые формируют кнопки калькулятора с цифрами.

2.6. Получение введенных параметров

Введенную пользователем или каким-либо другим образом информацию в отображаемый объект можно в любой момент получить с помощью метода GetData:

```
procedure <отображаемый объект>.GetData (var Rec); virtual;
```

где Rec - передаваемая запись с данными.

Этот метод имеется у любого отображаемого объекта. Для стандартных объектов он работает с той же информацией, что и метод SetData (см. п. 2.4).

2.7. Удаление отображаемого объекта

Из группы немодальный отображаемый объект удаляется с помощью метода Delete объекта tGroup, который после этого перерисовывает оставшиеся элементы группы. Следует иметь в виду, что при использовании этого метода отображаемый объект из динамической области памяти не удаляется и его можно повторно поместить в ту или иную группу.

Для удаления отображаемого объекта из динамической памяти следует воспользоваться стандартной процедурой Dispose совместно с деструктором Done. Если удаляемый таким образом отображаемый элемент является группой, то из динамической области также автоматически удаляются и все его подэлементы; поэтому заботиться об удалении подэлементов объекта-группы не нужно. Если при создании отображаемого объекта в динамической памяти выделялись какие-то области (за исключением областей, выделяемых для подэлементов группы), все они должны быть освобождены деструктором Done для последующего использования.

2.8. Работа с модальными объектами

Для работы с модальными отображаемыми объектами используются методы, выполняющие комплексные действия.

Универсальным является метод ExecView объекта tGroup:

```
function tGroup.ExecView(P: pView): Word;
```

где P - объект, с которым происходит работа;
результат - код команды, вызвавшей прекращение работы с объектом P.

Этот метод помещает отображаемый объект в группу, делает его модальным, обеспечивает обработку поступающих событий, и, если очередное событие является командой или сообщением (см. п. 1.7), означающими конец работы с отображаемым модальным объектом (например, команда cmCancel у диалогового окна, завершающая работу с окном без фиксации сделанных в нем изменений), удаляет отображаемый объект из группы. Предварительно перед удалением объекта следует снять с него модальность, что осуществляется с помощью процедуры EndModal объекта tView:

```
procedure tGroup.EndModal(Command: Word); virtual;
```

где Command - код команды, при которой снимается модальность.

В качестве результата функция ExecView возвращает код команды (см. п. 1.8), вызвавшей прекращение работы с модальным отображаемым объектом. После удаления из группы сам отображаемый объект из динамической области памяти не удаляется.

В методе GetEvent программы (лл. 667-707), если пришла команда обращения к справочной системе (стандартная команда cmHelp), формируется и помещается в рабочую область окно справочной системы типа tHelpWindow. Размещается окно с помощью метода ExecView (л. 699).

Аналогично содержимое стека калькулятора отображается на экране в соответствующем окне типа tStackWindow (лл. 65-69 и 632-660). При обработке команды cmClose - закрыть окно - в методе обработки событий HandleEvent снимается модальность окна методом EndModal (cmClose) - л. 658.

Если модальный отображаемый объект является диалоговым окном и его следует поместить в рабочую область, вместо метода ExecView целесообразно использовать метод ExecuteDialog объекта tProgram:

```
function tProgram.ExecuteDialog(P: pDialog;
                                Data: Pointer): Word;
```

где P - диалоговое окно, с которым происходит работа;
Data - указатель на запись с передаваемыми данными;
результат - код команды, вызвавшей прекращение работы с диалоговым окном P.

Метод выполняет следующие действия. Сначала он размещает диалоговое окно в рабочей области и делает его модальным. Перед этим проверяется, является ли диалоговое окно допустимым (указатель на окно не равен nil) и есть ли достаточно памяти для размещения окна. Если указатель равен nil, дальнейшие действия не выполняются, а если нет необходимой памяти, выдается сообщение об этом, а само диалоговое окно из динамической памяти удаляется. Далее метод ExecuteDialog передает диалоговому окну исходные параметры. Адрес записи, содержащей исходные значения (в эту же запись после завершения работы с диалоговым окном в случае необходимости будут помещены введенные пользователем значения), передается в качестве параметра при обращении к методу ExecuteDialog. Затем обрабатываются поступающие события. Если очередное событие является командой или сообщением (см. п. 1.7), означающими конец работы с диалоговым окном, и команда не является командой прекращения работы без фиксации введенных в окно данных, в запись, содержащую данные, адрес которой был

передан при обращении к методу ExecuteDialog, заносятся новые значения. Предварительно, если используется контроль введенных данных (см. п. 8.1), проверяется допустимость введенных данных, и в случае неверно введенных данных выдается сообщение, а работа с диалоговым окном продолжается. При прекращении работы с диалоговым окном по команде без фиксации введенных данных содержимое записи не изменяется. Наконец, диалоговое окно удаляется из рабочей области, предварительно с него снимается модальность. После завершения работы с диалоговым окном оно также удаляется и из динамической области памяти.

Объект-калькулятор типа tCalculator (лл. 44-50) имеет метод обработки событий HandleEvent (лл. 476-550), который в случае появления команды задания параметров калькулятора cmParameters (л. 542) вызывает подпрограмму Parameters (лл. 477-525), которая создает диалоговое окно (см. рис. 2.1) и обеспечивает работу с ним.

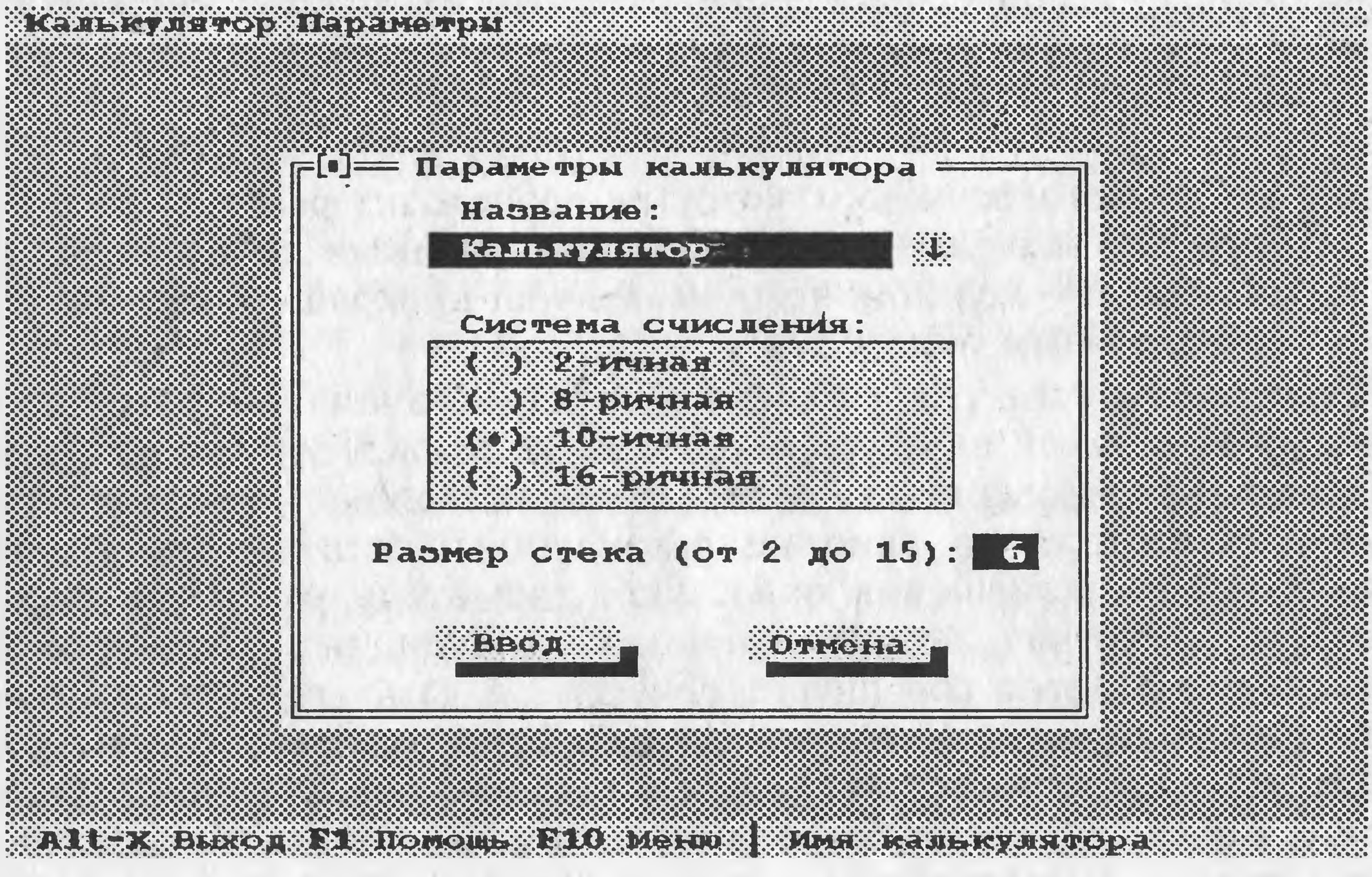


Рис. 2.1. Диалоговое окно задания параметров калькулятора

Так как диалоговое окно содержит строку ввода имени калькулятора (см. п. 4.4.1), селективный список выбора системы счисления (см. п. 4.4.4), строку ввода числа (см. п. 4.4.1), определяющего размер стека, и двух кнопок (см. п. 4.4.5): ввода данных и отмены работы с окном (именно эта кнопка и формирует команду завершения рабо-

ты без фиксации введенных данных), включенных в диалоговое окно именно в этом порядке, при обращении к методу ExecuteDialog (л. 501) указывается адрес записи Par (лл. 479-483), содержащей передаваемые данные - имя калькулятора, номер выбранной системы счисления и символьное представление числа элементов стека. В программе размер стека ограничен величинами 2 и 15, в строку ввода размера стека (лл. 577-583) введен контроль правильности введенного числа (л. 580), который будет перед завершением работы с диалоговым окном проверять допустимость введенного числа.

3. СОЗДАНИЕ МЕНЮ

Как уже указывалось, Turbo Vision позволяет создать меню программы, предназначенное для выбора дальнейшего пути решения задачи.

После написания объекта-приложения - основного объекта программы (см., например, п. 1.5) - целесообразно продолжить создание программы именно с описания меню, т. к. выбор той или иной структуры меню во многом будет определять и структуру программы. В дальнейшем при необходимости его можно будет скорректировать. В Turbo Vision можно использовать либо обычное разворачивающееся меню, либо так называемое локальное меню. Их создание имеет некоторые особенности, о которых будет сказано ниже. Основным меню, как правило, является разворачивающееся меню, располагаемое обычно в верхней строке экрана. Работа с меню осуществляется так же, как и с меню в пакете Turbo Pascal (Borland Pascal) [1].

3.1. Создание разворачивающегося меню

Разворачивающееся меню, которое в дальнейшем для краткости будем называть просто меню, может иметь довольно сложную структуру и любое число уровней вложенности, но составляется из однотипных элементов, правда, имеющих три варианта: собственно элемент меню (фактически определяет команду, которую следует выполнить при выборе этого элемента), подменю, при выборе которого на экране раскрывается соответствующий пункт исходного меню, и разделительная линия, которая имеет чисто декоративное назначение и позволяет отделять в подменю те или иные группы элементов друг от друга.

Элемент меню имеет структуру типа-запись (стандартный тип tMenuItem):

```
tMenuItem = record
    Next: pMenuItem;
    Name: pString;
    Command: Word;
    Disabled: Boolean;
    KeyCode: Word;
    HelpCtx: Word;
```

```
    case Integer of
      0: (Param: pString);
      1: (SubMemu: pMenu);
    end;
  end;
```

где Next - указатель на следующий элемент этого же уровня вложенности; если следующего элемента нет, указатель равен nil;

Name - указатель на строку, содержащую выводимый на экран текст; этот параметр равен nil, если элемент представляет собой разделительную линию;

Command - код команды, которую следует выполнить при выборе данного элемента меню; эта величина должна равняться нулю, если данный элемент представляет собой подменю;

Disabled - признак разрешения или запрета выбора данного элемента меню;

KeyCode - код клавиши или совокупности клавиш быстрого управления (см. п. 1.7), с помощью которых можно выполнить команду данного элемента меню, не вызывая этот элемент, или 0 (константа kbNoKey), если для данного элемента такой клавиши не предусмотрено;

HelpCtx - параметр контекстной помощи, аналогичный параметру объекта tView (см. п. 1.3), но справочная система выдает в этом случае информацию об элементе меню;

Param - указатель на строку с дополнительной информацией, которая выводится в правой части строки подменю, соответствующей данному элементу; обычно это название клавиши (клавиш) быстрого управления;

SubMemu - указатель на структуру меню следующего уровня, если данный элемент представляет собой подменю.

Горизонтальные размеры выводимой на экран строки для любого элемента меню ограничены 31 символом. В связи с этим в Turbo Vision имеется стандартный тип строк tMenuStr:

```
tMenuStr = string[31];
```

который и используется для задания символьной информации элементам меню.

Для каждого элемента меню можно в его названии выделить одну латинскую букву или цифру для быстрого выбора элемента путем нажатия этой клавиши на клавиатуре (для основного меню - в сочетании с клавишей Alt). Такая буква выделяется в тексте символами ~~, например,

```
~N~ew.
```

В данном примере соответствующий элемент можно активизировать нажатием клавиши N (или Alt + N, если это элемент основного меню). На экране такая буква выделяется особым цветом по отношению к другим буквам текста.

Так как каждый элемент меню имеет довольно сложную структуру, имеется три (по числу вариантов) стандартные подпрограммы, позволяющие единообразно создавать эти элементы.

Для создания элемента меню, определяющего выполняемую команду, используется подпрограмма NewItem:

```
function NewItem(Name, Param: tMenuStr; KeyCode: Word;
  Command: Word; AHelpCtx: Word; Next: pMenuItem): pMenuItem;
```

где Name - текстовая информация для данного элемента, она не должна быть пустой строкой, чтобы не спутать элемент с разделительной линией;
Param - дополнительная текстовая информация для элемента;
KeyCode - код клавиши (клавиш) быстрого управления;
Command - код выполняемой команды, он не должен равняться нулю, чтобы не спутать элемент с подменю;
AHelpCtx - параметр контекстной помощи;
Next - указатель на следующий элемент меню данного уровня;
результат - указатель на созданный элемент меню.

При создании такого элемента в случае, когда по указанной команде вызывается то или иное окно (диалоговое окно) для задания конкретных параметров, желательно, как это сделано в пакете Turbo Pascal (Borland Pascal), текстовую информацию завершить тремя точками (см., например, л. 750 программы-калькулятора).

Для создания элемента меню, представляющего собой подменю, используется подпрограмма NewSubMenu:

```
function NewSubMenu(Name: tMenuStr; AHelpCtx:
    Word; SubMenu: pMenu; Next: pMenuItem): pMenuItem;
```

где Name - текстовая информация для данного элемента, она не должна быть пустой строкой, чтобы не спутать элемент с разделительной строкой;
AHelpCtx - параметр контекстной помощи;
SubMenu - указатель на структуру подменю;
Next - указатель на следующий элемент меню данного уровня;
результат - указатель на созданный элемент меню.

При отображении на экране такого элемента меню (за исключением случая главного меню) в правой части строки информации, соответствующей этому элементу, автоматически будет добавлен знак "стрелка вправо".

Если элемент меню является разделительной строкой, то используется подпрограмма NewLine:

```
function NewLine(Next: pMenuItem): pMenuItem;
```

где Next - указатель на следующий элемент меню данного уровня;
результат - указатель на созданный элемент меню.

Связанные друг с другом элементы одного уровня (представляющие собой одно подменю) объединяются вместе и также образуют структуру типа-запись (стандартный тип tMenu):

```
tMenu = record
   Items: pMenuItem;
   Default: pMenuItem;
end;
```

где Items - указатель на первый элемент списка подменю;
Default - указатель на активизируемый элемент списка при обращении к данному подменю.

Чтобы создать структуру такого типа, следует воспользоваться стандартной подпрограммой NewMenu:

```
function NewMenu(Items: pMenuItem): pMenu;
```

где Items - указатель на первый элемент подменю;
результат - указатель на созданную структуру.

При выполнении этой подпрограммы поля Items и Default структуры принимают значение указателя на первый элемент подменю.

Функционально разворачивающееся меню состоит из нескольких частей - главного меню, постоянно отображаемого на экране, и ряда подменю, появляющихся на экране по мере необходимости. Главное меню представляет собой объект типа tMenuBar. Создается оно обычно с помощью стандартной подпрограммы New, а инициализируется конструктором Init:

```
constructor tMenuBar.Init(var Bounds: tRect; AMenu: pMenu);
```

где Bounds - область на экране для главного меню;
AMenu - указатель на главное меню.

Подменю представляют собой выпадающие меню, появляющиеся на экране по мере необходимости.

Так как при создании того или иного элемента следует задать указатели на последующие элементы структуры (данного уровня и следующего), а эти указатели получаются при создании соответствующих элементов, можно создание всего меню оформить в виде одного оператора, что обычно и делается.

Создавая меню программы следует позаботиться лишь о размерах основного меню, размеры же выпадающих подменю определяются автоматически при их отображении на экране. Стандартный объект tProgram (а следовательно, и объект tApplication) имеет метод InitMenuBar:

```
procedure tProgram.InitMenuBar; virtual;
```

с помощью которого можно создать меню и который следует переопределить для конкретной программы. Этот метод помещает указатель на созданное меню в стандартную типизированную константу MenuBar пакета Turbo Vision.

Главное меню программы-калькулятора состоит из двух элементов: подменю "Калькулятор", позволяющее создавать в рабочей области новые калькуляторы и удалять их, а также завершать работу с программой, и подменю "Параметры", с помощью которого можно посмотреть и изменить некоторые параметры калькулятора. Подменю "Калькулятор", в свою очередь, состоит из четырех элементов: элемента, открывающего новый калькулятор, элемента, закрывающего текущий калькулятор, элемента, завершающего работу с калькулятором, и, наконец, разделительной линии, отделяющей первые два элемента от последнего. Подменю "Параметры" состоит из двух элементов: элемента, позволяющего посмотреть и изменить некоторые параметры калькулятора, и элемента, позволяющего посмотреть содержимое стека.

Структура этого меню представлена на рис. 3.1.

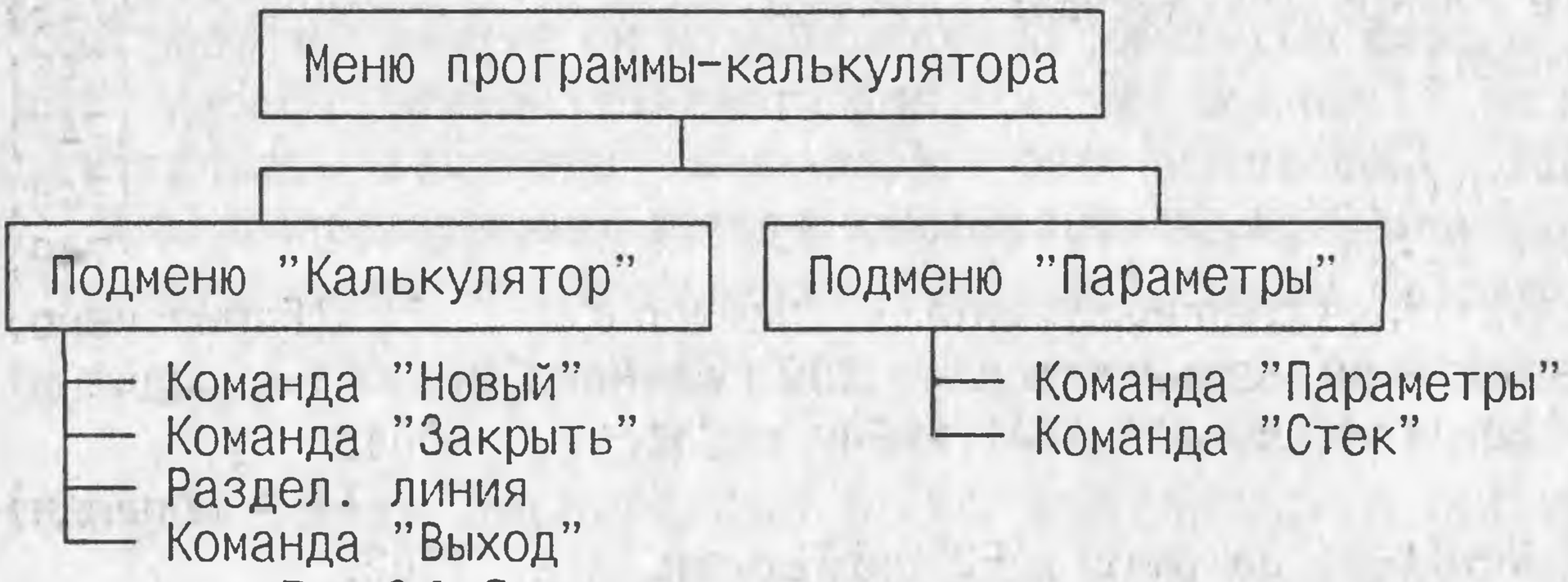


Рис. 3.1. Структура меню программы-калькулятора

Конкретные типы объектов и элементов этого меню, а также подпрограммы и методы, используемые для создания такого меню, представлены на рис. 3.2.

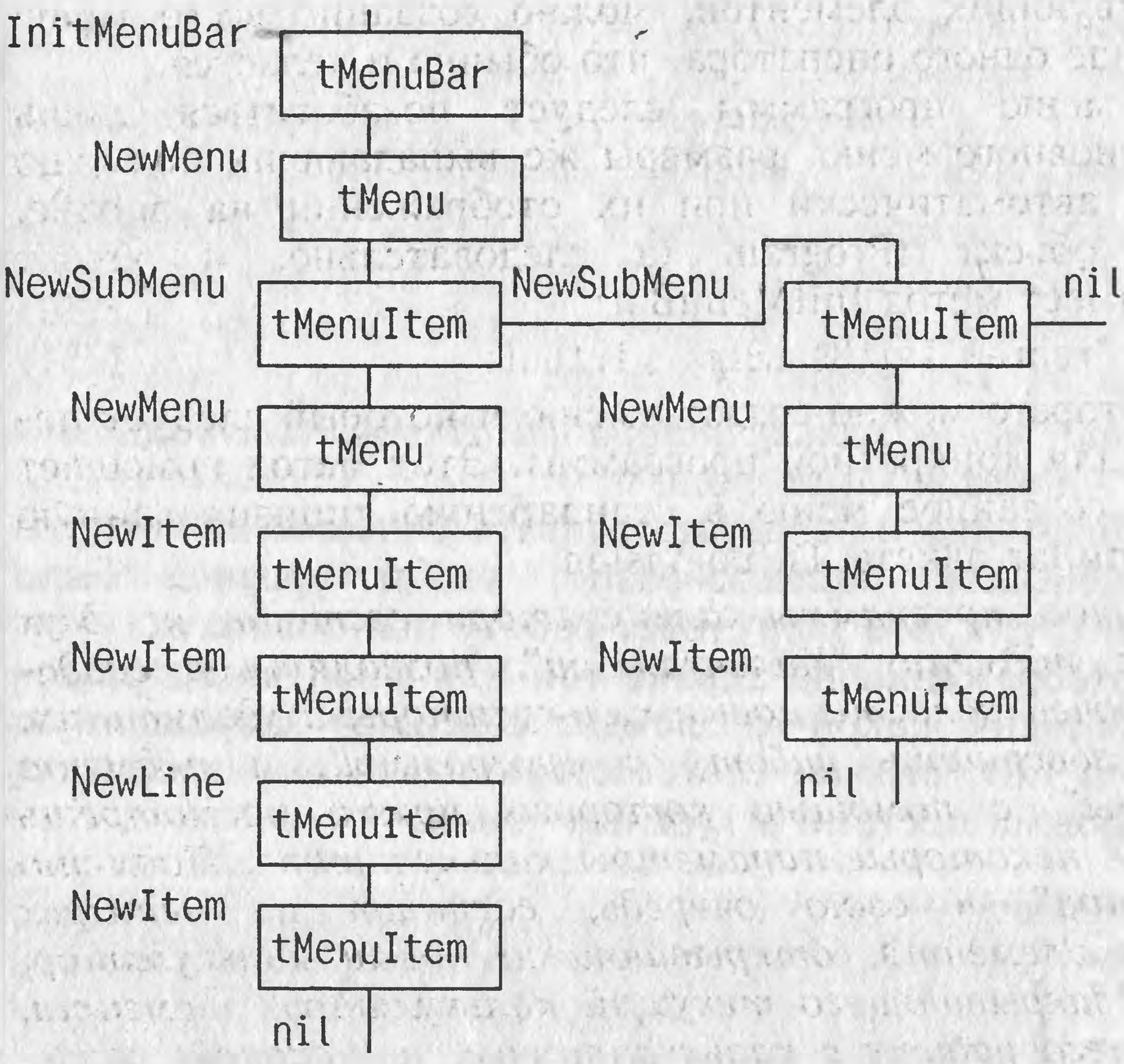


Рис. 3.2. Структура типов элементов меню программы-калькулятора и подпрограмм, формирующих эти элементы

Метод InitMenuBar в данном случае выглядит следующим образом:

```
{ Создание меню }
procedure tMyApp.InitMenuBar;                                    {745}
var R: tRect;                                                    {746}
begin                                                            {747}
   GetExtent(R);                                                 {748}
   R.B.Y:= R.A.Y + 1;                                            {749}
   MenuBar:= New(pMenuBar, Init(R, NewMenu(                      {Гориз.меню}
      NewSubMenu('Калькулятор', 1000, NewMenu(                   {1-е подменю}
         NewItem('Новый','F4',kbF4, cmCalcNew, 1001,
                                                                 {1-я команда}
         NewItem('Закрыть','F2',kbF2, cmClose, 1002,
                                                                 {2-я команда}
         NewLine(                                                {Разд.линия}
```

```
        NewItem('Выход','Alt-X',kbAltX,cmQuit,1003,
                                                    {3-я команда}
        nil))))),
    NewSubMenu('Параметры', 1004, NewMenu(          {2-е подменю}
        NewItem('Параметры...', '', kbNoKey,
                    cmParamrters, 1005,             {1-я команда}
        NewItem('Стек', '', kbNoKey, cmStack, 1006,
                                                    {2-я команда}
        nil))),
    nil)))));                                       {750}
end;                                                {751}
```

Внешний вид меню калькулятора (и строки состояния - см. п. 4.4.1) показан на рис. 3.3.

При создании меню не следует задавать большое количество уровней вложенности (целесообразно, чтобы уровней было не больше двух), т. к. это затрудняет работу пользователя.

Если при отображении на экране той или иной части разворачивающегося меню какой-то элемент содержит запрещенную команду (см. п. 1.8), строка, соответствующая этому элементу, выделяется особыми цветами в тусклых тонах.

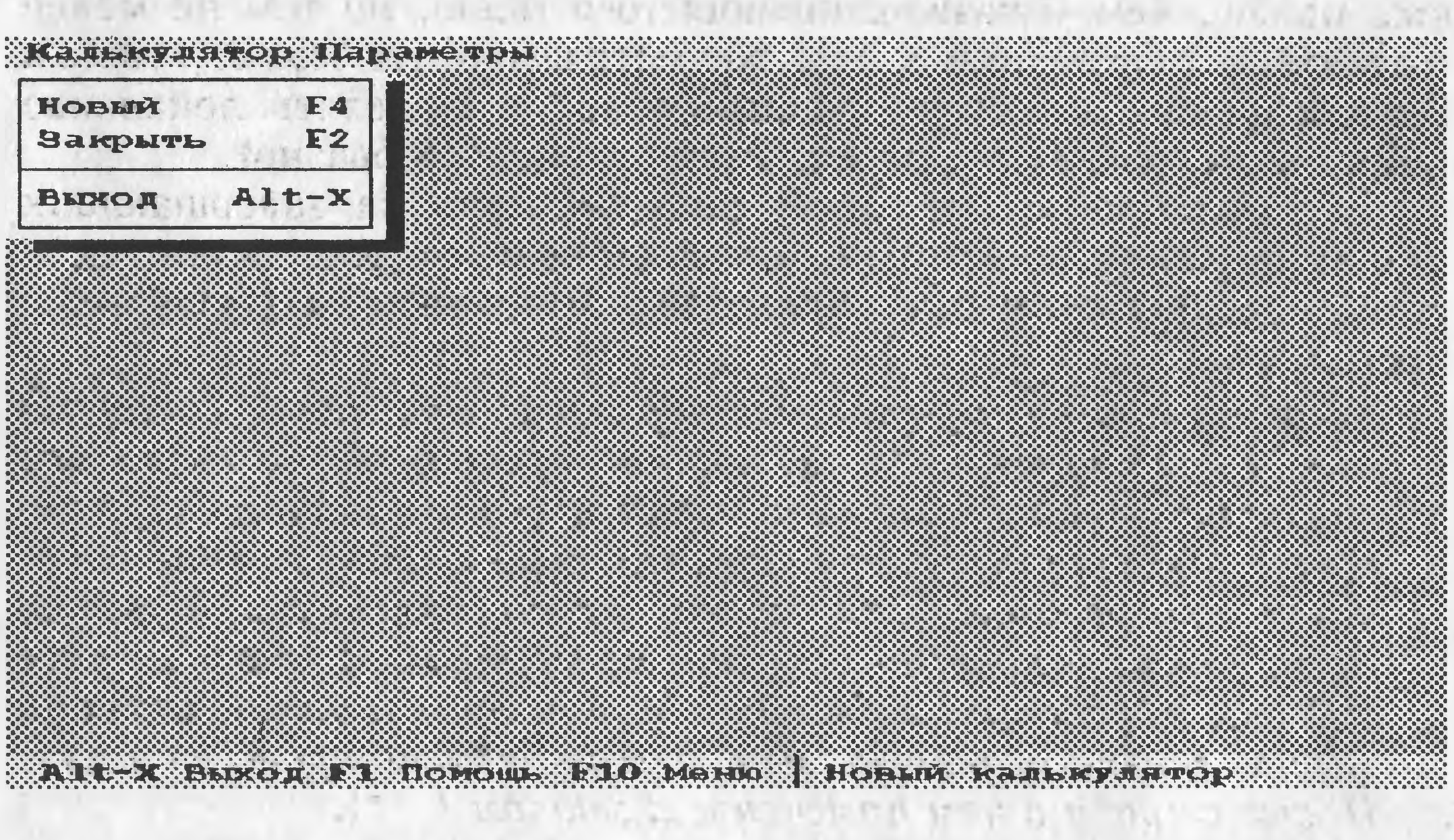


Рис. 3.3. Меню и строка состояния программы-калькулятора

3.2. Создание локального меню

При работе с программой часто приходится многократно повторять одни и те же действия, которые разнесены по различным частям меню (а некоторых в меню вообще может и не быть). Количество их обычно небольшое, а поиск по всему меню неудобен. Целесообразно такие действия собрать в одно дополнительное меню, доступ к которому осуществляется по тому или иному событию, например команде, нажатию клавиши клавиатуры или мыши. В пакете Borland Pascal (Turbo Pascal) такое меню вызывается нажатием правой клавиши мыши. Это довольно удобно, если только правая клавиша уже не предназначена для каких-либо других целей.

Для создания такого меню можно использовать стандартный объект пакета Turbo Vision - tMenuPopup. Для этого в метод HandleEvent основного объекта программы следует включить обработку выбранного для локального меню события и написать подпрограмму вызова локального меню, в которой предусмотреть создание локального меню, очень напоминающее конструирование выпадающего меню. Так как локальное меню должно быть довольно простым, оператор его конструирования будет также проще, чем у разворачивающегося меню, но тем не менее состоять из обращений к тем же методам (быть может, только без обращения к методу NewSubMenu, т. к. делать локальное меню многоуровневым, по-видимому, нецелесообразно).

Создавать локальное меню целесообразно на завершающих этапах создания программы, когда становится ясно, какие действия наиболее часто придется выполнять пользователю программы.

*Введем в программу-калькулятор локальное меню, которого нет в окончательном варианте программы, вызываемое нажатием правой клавиши мыши и выполняющее два действия: удаление текущего калькулятора и прекращение работы с программой. Для этого внесем изменения в метод HandleEvent основного объекта программы. В этом методе добавим обработку события от нажатия правой клавиши мыши. Видоизмененный метод HandleEvent приведен ниже. Новые строки в нем помечены символом { *}.*

```
{ Обработка событий }
procedure tMyApp.HandleEvent(var Event: tEvent);                {715}
   { Создание нового калькулятора }
   procedure CalcNew;                                           {716}
```

```
var P: pCalculator;                                          {717}
      R: tRect;                                              {718}

begin                                                        {719}
   if Desktop^.Current = nil then                            {720}
      R.Assign(5, 1, 29, 20)                                 {721}
   else                                                      {722}
      R.Assign(Desktop^.Current^.Origin.X + 1,
                  Desktop^.Current^.Origin.Y + 1,
                  Desktop^.Current^.Origin.X + 25,
                  Desktop^.Current^.Origin.Y + 20);          {723}
   P:= New(pCalculator, Init(R));                            {724}
   InsertWindow(P);                                          {725}
end;                                                         {726}

{ Работа с локальным меню }
procedure LocalMenu;                                         {  *}
var R: tRect;                                                {  *}
      RMenu: tRect;                                          {  *}
      MenuPopup: pMenuPopup;                                 {  *}
      Result: Word;                                          {  *}

begin                                                        {  *}
   { Определение расположения меню }
   GetExtent(R);                                             {  *}
   RMenu.A:= MouseWhere;                                     {  *}
   if RMenu.A.X > R.B.X - 19 then                            {  *}
      RMenu.A.X:= RMenu.A.X - 18;                            {  *}
   if RMenu.A.Y > R.B.Y - 7 then                             {  *}
      RMenu.A.Y:= RMenu.A.Y - 5;                             {  *}
   RMenu.B.X:= RMenu.A.X + 31;                               {  *}
   RMenu.B.Y:= RMenu.A.Y + 5;                                {  *}

   { Создание меню }
   MenuPopup:= New(pMenuPopup, Init(RMenu,
      NewMenu(
      NewItem('Закрыть','F2',kbF2, cmClose, 1002,
      NewLine(
      NewItem('Выход','Alt-X',kbAltX,cmQuit,1003,
      nil))))));                                             {  *}
   ClearEvent(Event);                                        {  *}

   { Работа с меню }
   Result:= DeskTop^.ExecView(MenuPopup);                    {  *}
```

```
      { Обработка результатов работы с меню }
      if Result <> cmCancel then                                    {  *}
         Message(Application,evCommand,Result, nil);
                                                                    {  *}
   end;

begin                                                               {727}
   inherited HandleEvent(Event);                                    {728}

   case Event.What of                                               {729}
      evCommand:                                                    {730}
         begin                                                      {731}
            case Event.Command of                                   {732}
               cmCalcNew: CalcNew;                                  {733}
            else                                                    {734}
               Exit;                                                {735}
            end;                                                    {736}
            ClearEvent(Event);                                      {737}
         end;                                                       {738}
      evMouseDown:                                                  {  *}
         if Event.Buttons = mbRightButton then                      {  *}
            LocalMenu;                                              {  *}
   end;                                                             {739}
   { Разрешение и запрет команд }
   if DeskTop^.Current = nil then                                   {740}
      DisableCommands([cmParamrters, cmStack])                      {741}
   else                                                             {742}
      EnableCommands([cmParamrters, cmStack]);                      {743}
end;                                                                {744}
```

В этом примере дополнительно анализируется нажатие правой клавиши мыши и в случае возникновения такого события вызывается подпрограмма работы с локальным меню LocalMenu. В этой подпрограмме сначала определяется место расположения локального меню на экране таким образом, чтобы оно было рядом с курсором мыши и полностью размещалось в рабочей области. Затем создается локальное меню, состоящее из двух строк-команд, с разделительной линией между ними. После этого осуществляется работа с меню, используя метод ExecView, позволяющий сделать меню модальным объектом (см. п. 1.6). После завершения работы с меню результат работы (команда, выбранная с помощью меню) передается с помощью стан-

дартной подпрограммы Message (см. п. 1.7) основному объекту программы для дальнейшей обработки.

Следует сделать небольшое замечание: несмотря на свою сложность, меню в силу своей специфики группой не является, и у него имеется свой своеобразный механизм обработки событий. Так, например, если меню получит событие от мыши, когда курсор не находится в поле отображения меню, оно закрывается, как будто поступила команда завершения работы с объектом без фиксации результата работы. В случае же модального объекта-группы (например, диалогового окна) такое событие просто игнорируется (см. п. 1.7).

В Turbo Vision имеется ряд стандартных меню (правда, написанных на английском языке), позволяющих работать с файлами, редактором и окнами. Об этих стандартных меню сказано в п. 9.1.

4. СОЗДАНИЕ ОКОН И ДИАЛОГОВЫХ ОКОН

После того как будет выбрано меню программы, а в случае, если в программе используется строка состояния (см. п. 6), после создания меню и строки состояния, целесообразно приступить к формированию отдельных частей программы, обрабатывающих соответствующие команды меню и строки состояния. Как указывалось в п. 1.7, эта обработка осуществляется методом HandleEvent. В первую очередь следует написать метод HandleEvent основного объекта программы (потомок tApplication), т. к. обычно именно здесь и осуществляется обработка команд меню и строки состояния. Как отмечалось выше, часто эти команды требуют использования различных окон для передачи информации. Окна могут потребоваться и при создании подпрограмм, обрабатывающих те или иные команды. Вопросам создания таких окон и будет посвящена данная глава.

Основным средством взаимодействия пользователя с программой в Turbo Vision являются окна (тип tWindow) и диалоговые окна (тип tDialog). Именно через них в основном пользователь вводит информацию в программу, через них полученная программой информация отображается на экране для дальнейшей обработки пользователем. У окон и диалоговых окон много общего, т. к. диалоговое окно является непосредственным потомком окна, однако есть и принципиальные отличия, главным образом в их предназначении, наборе стандартных элементов, которые можно разместить в них, и в цветовой палитре. Можно сказать, что основное назначение окон - просмотр той или иной (обычно текстовой) информации, а диалоговых окон - ввод в программу данных. Так как диалоговые окна обладают рядом дополнительных возможностей и более логически завершены, они чаще используются в программах. Тем не менее целесообразно рассмотрение начать с общих свойств окон, т. к. после этого возможности диалоговых окон будут восприниматься, по-видимому, проще.

4.1. Окна (tWindow) в Turbo Vision

Окном в Turbo Vision (тип tWindow) называется отображаемый объект-группа, имеющий специальную рамку и который может иметь средства прокрутки (скроллинга) не умещающейся в окне информации. В него можно поместить какую-то (обычно текстовую) информацию. Как правило, в программе используется не окно типа tWindow, а его потомок, т. к. для конкретных окон приходится переопределять некоторые методы окна tWindow, в первую очередь конструктор Init для размещения в окне его компонент и метод HandleEvent для соответствующей обработки событий, предназначенных окну.

В программе-калькуляторе используется окно для отображения содержимого стека калькулятора (рис. 4.1) - тип tStackWindow.

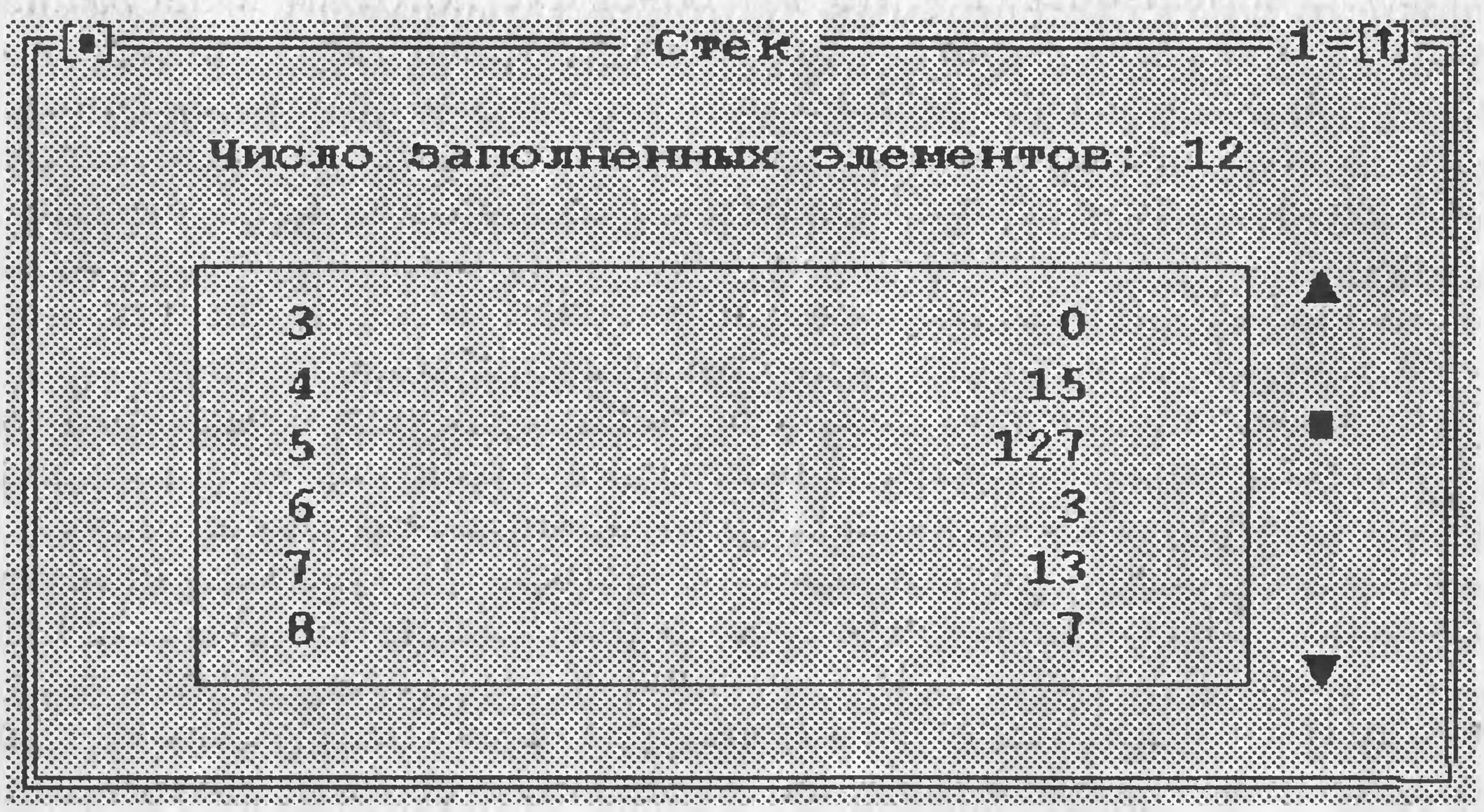


Рис. 4.1. Окно для просмотра содержимого стека

Окно имеет рамку, поле скроллинга, в которое помещается содержимое стека, и линейку вертикального скроллинга. Для иллюстрации возможностей окна здесь приведены некоторые необязательные для данного примера элементы: номер окна, пиктограмма распахивания окна и угол изменения размеров окна.

Это окно в программе-калькуляторе описано в лл. 65-69 и 632-660, а его компонента - поле скроллинга - в лл. 59-64 и 595-631.

В дополнение к параметрам объекта tGroup окно обладает рядом дополнительных параметров, отметим из них два: флаги окна Flags и используемая палитра Palette. Назначение флагов параметра Flags, и константы, соответствующие единице в соответствующем разряде параметра Flags следующие:

0-й бит (wfMove) - возможность перемещения окна по полю его владельца.
1-й бит (wfGrow) - возможность изменения размеров окна.
2-й бит (wfClose) - наличие пиктограммы закрытия окна.
3-й бит (wfZoom) - наличие пиктограммы распахивания окна.

Параметр Palette определяет тип используемой палитры окна и может принимать одно из трех следующих значений:

wpBlueWindow = 0 - синяя палитра окна;
wpCyanWindow = 1 - голубая палитра окна;
wpGrayWindow = 2 - серая палитра окна.

Как правило, синяя палитра используется для окон, серая - для диалоговых окон, голубая - для вспомогательных окон (например, для окон справочной системы).

Отметим также параметр Number, куда помещается закрепленный за окном номер - целое число типа Integer.

Инициализируется окно конструктором Init:

```
constructor tWindow.Init(var Bounds: tRect;
                         ATitle: tTitleStr; ANumber: Integer);
```

где Bounds - размеры окна;
ATitle - имя окна размером до 80 символов;
ANumber - номер окна.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.1.

Таблица 4.1. Параметры объекта tWindow при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable+ofTopSelect+ofBuffered
DragMode	dmLimitLoY
GrowMode	$1F
EventMask	$FFFF
State	sfVisible+sfShadow+sfExposed
HelpCtx	hcNoContext
Flags	wfMove+wfGrow+wfClose+wfZoom
Palette	wpBlueWindow

3-1034

Number	ANumber
Next	nil
Owner	nil
Last	указатель на рамку окна
Current	nil

После того как окно будет помещено в группу (как правило, в рабочую область - Desktop), соответствующим образом заполнятся параметры Next и Owner. После размещения в окне компонент будут заполнены параметры Last и Current.

Характерной особенностью окна и его потомков является наличие рамки - специального отображаемого объекта, помещаемого в окно. Таким образом, окно всегда имеет хотя бы один подэлемент. Эту рамку не следует путать с рамкой вокруг отображаемого элемента, изображаемой в случае задания флага ofFramed параметра Options (см. п. 1.3), - такая рамка также изображена на рис. 4.1 вокруг поля скроллинга. Рамка окна помимо декоративного назначения несет на себе еще дополнительную информацию об этом окне.

Вид рамки окна меняется в зависимости от его состояния. Активное окно (см. п. 1.6) обрамлено двойной рамкой, как показано на рис. 4.1, все неактивные окна - одинарной рамкой. Наличие двойной рамки у активного окна выделяет его среди всех окон и наглядно показывает, с каким окном в настоящий момент происходит работа. К тому же активное окно обычно находится сверху всех остальных отображаемых объектов, т. к. у окна, как правило, установлен флаг ofTopSelect параметра Options (см. п. 1.3).

В левой части верхней линии рамки располагается пиктограмма закрытия окна [■]. При подведении курсора мыши к этой пиктограмме и нажатии клавиши (левой) окно закрывается. Если в окне были произведены какие-то изменения данных (что бывает довольно редко, т. к. для изменения данных более приспособлено диалоговое окно), эти изменения не сохраняются.

В средней части верхней линии рамки располагается имя окна, задаваемое при его инициализации. Если имя было задано в виде пустой строки - '', окно не будет иметь имени.

В правой части верхней линии рамки находится пиктограмма распахивания окна "стрелка вверх" до полного размера группы, в которую включено окно. Распахивание также осуществляется с помощью мыши. После того как окно будет распахнуто, пиктограмма изменит свой вид - "двусторонняя стрелка". Чтобы восстановить исходные размеры окна, следует повторно подвести

окно, а затем снова его сжать можно и другим способом. Для этого следует поместить курсор мыши на верхнюю линию рамки окна (в любое место, кроме пиктограмм закрытия и распахивания) и дважды нажать клавишу мыши.

Правый нижний угол рамки может быть таким же, как и все другие углы, а может, как на рис. 1.4, быть изображен одинарной линией даже в случае активного окна. В последнем случае с помощью мыши можно изменять размеры окна. Для этого курсор мыши следует поместить на этот угол, нажать на клавишу мыши и, не отпуская ее, переместить мышь в нужном направлении, при достижении окном требуемых размеров клавишу мыши следует отпустить. Размеры окна можно изменять до определенных пределов. Так, минимальные размеры окна - 16 позиций по горизонтали и 6 строк по вертикали. Правда, эти ограничения можно изменить с помощью метода SizeLimits:

```
procedure tView.SizeLimits(var Min, Max: tPoint); virtual;
```

где Min, Max - минимальный и максимальный размеры окна.

Верхняя линия позволяет в ряде случаев с помощью мыши перемещать окно по полю его владельца. Для этого курсор мыши следует подвести к любому участку линии за исключением пиктограмм закрытия и распахивания окна, нажать клавишу мыши и, не отпуская ее, переместить мышь в нужном направлении, после чего клавишу отпустить.

Как отмечалось выше, наличие или отсутствие у окна этих возможностей зависит от установленных флагов параметра Flags. При инициализации окна устанавливаются все флаги. Если требуется использование некоторых свойств окна, следует установить лишь те флаги, которые задают эти свойства.

Если необходимо, чтобы окно для отображения содержимого стека калькулятора имело пиктограмму закрытия и могло перемещаться по экрану, в методе Init объекта tStackWindow лл. 632-654 следует поместить оператор

```
Flags:= wfMove + wfClose;
```

Примечание. Те же действия с окном можно осуществить и с помощью клавиш быстрого управления, например так, как это сделано в пакете Turbo Pascal (Borland Pascal): Ctrl+F5 - для перехода в режим перемещения окна и изменения его размеров; F5 - для распахивания окна, Alt+F3 - для закрытия окна. Проще всего внести соответствующее изменение в меню, запрограммировав использование тех или иных клавиш быстрого управления. Так, можно, например, следующим образом представить метод инициализации меню программы-калькулятора:

```
{ Создание меню }
procedure tMyApp.InitMenuBar;                                    {745}
var R: tRect;                                                    {746}
begin                                                            {747}
  GetExtent(R);                                                  {748}
  R.B.Y:= R.A.Y + 1;                                             {749}
    MenuBar:= New(pMenuBar, Init(R, NewMenu(
       NewSubMenu('Калькулятор', 1000, NewMenu(
            NewItem('Новый','F4',kbF4, cmCalcNew, 1001,
            NewItem('Закрыть','F2',kbF2, cmClose, 1002,
            NewLine(
            NewItem('Выход','Alt-X',kbAltX,cmQuit,1003,
            nil))))),
       NewSubMenu('Параметры', 1004, NewMenu(
            NewItem('Параметры', '', kbNoKey, cmParamrters, 1005,
            NewItem('Стек', '', kbNoKey, cmStack, 1006,
            nil))),
       NewSubMenu('Окно', 1007, NewMenu(
            NewItem('Разм./Полож.', 'Ctrl-F5',
                          kbCtrlF5, cmResize, 1008,
            NewItem('Распахнуть', 'F5', kbF5, cmZoom, 1009,
            NewItem('Закрыть', 'Alt-F3', kbAltF3, cmClose, 1010,
            nil)))),
       nil))));                                                  {750}
end;                                                             {751}
```

В этом методе cmResize, cmZoom, cmClose коды стандартных команд изменения размеров, распахивания и закрытия окна. Окно "знает", как реагировать на эти команды. Следует только сделать два замечания. Та или иная команда будет воздействовать на окно, если только у него установлен соответствующий флаг параметра Flags. Кроме этого, команды меню доступны только немодальным окнам (и их потомкам). Поэтому введенные изменения не будут воздействовать на окно для просмотра содержимого стека, т. к. оно является модальным, а будет воздействовать только на диалоговые окна - образы калькулятора (см. рис. 1.1), являющиеся немодальными.

Правее имени на верхней линии рамки находится номер окна. Если этот номер в пределах от 1 до 9, окно можно активизировать нажатием комбинации клавиш Alt и цифры, соответствующей номеру окна. Номер окна задается при инициализации окна (см. выше). Если необходимо задать окно без номера, следует при инициализации в качестве третьего параметра конструктора Init задать число 0 или константу wnNoNumber = 0.

Как уже отмечалось, при инициализации окна задается синяя палитра. Если необходимо установить другую палитру, это нужно сделать явным образом (как, например, в л. 641).

При работе с окном его следует сначала создать и инициализировать с помощью конструктора Init (см. выше). При этом следует создать необходимые компоненты (кроме рамки, которая создается и размещается автоматически) и поместить их в окно.

В окне для отображения содержимого стека калькулятора размещаются поле скроллинга и линейка вертикального скроллинга - лл. 646-653 (см. ниже), а также статический текст - лл. 643-645 (см. п. 4.4.6).

Далее окно следует поместить в рабочую область. Как правило, окно используется в немодальном режиме, но может использоваться и в модальном, как это сделано в следующем примере. Если окно немодальное, его помещают в рабочую область с помощью метода InsertWindow (см. п. 2.3), если окно модальное - с помощью метода ExecView (см. п. 2.8).

В подпрограмме StackContents (лл. 526-535) осуществляется работа с окном для отображения содержимого стека, где сначала создается окно (л. 529), затем проверяется, создано ли окно и помещается ли оно в памяти (л. 530), и, если все благополучно, окно размещается в рабочей области как модальное (л. 532).

Как правило, для окна следует переопределить метод обработки событий HandleEvent, где необходимо предусмотреть обработку предназначенных для окна событий.

Так как окно для отображения содержимого стека работает только со стандартными командами, такими, как cmResize, cmZoom, cmClose (см. выше), формируемыми при нажатии соответствующих пиктограмм или при перемещении окна либо изменении его размеров с помощью мыши и обрабатываемыми автоматически уже имеющимся стандартным методом окна HandleEvent, нет необходимости в методе HandleEvent данного конкретного окна предусматривать специальную обработку событий (за одним-единственным исключением, о котором будет сказано ниже).

Перед завершением работы с модальным окном следует снять с него модальность с помощью метода EndModal (см. п. 2.8). Проще это сделать в том же методе HandleEvent, когда обрабатывается та или иная команда завершения работы с окном.

Окно для отображения содержимого стека находится в модальном режиме, и при завершении работы с ним необходи-

мо снять с него модальность. Собственный метод HandleEvent (лл. 655-660) и предназначен для этих целей, когда при появлении команды ctClose следует снять модальность (лл. 657-658). Конечно же, в этом методе необходимо предусмотреть обращение к наследуемому методу HandleEvent (л. 659) для обработки стандартных событий.

Если окно содержит элементы, которые могут находиться в выделенном состоянии (см. п. 1.6), то только один из этих элементов может в любой момент времени находиться в этом состоянии. Если нужно сделать выделенным другой элемент можно воспользоваться либо мышью, подведя курсор мыши к соответствующему элементу и нажав клавишу мыши, либо с помощью клавиши клавиатуры Tab (или Shift+Tab); при этом нажатие клавиши Tab выделит следующий в Z-порядке (см. п. 1.4) элемент, а Shift+Tab - предыдущий.

В диалоговом окне типа tCalculator (которое, естественно, является потомком объекта tWindow) все кнопки (лл. 442-460) могут находиться в выделенном состоянии. Выделить любую кнопку можно указанными выше способами.

4.2. Элементы окон в Turbo Vision

В качестве стандартных элементов окон Turbo Vision можно использовать поля скроллинга (тип tScroller), предназначенные для отображения не умещающейся на экране информации и ее просмотра, текстовые элементы буферированной информации (тип tTextDevice), текстовые элементы для пассивных терминалов (тип tTerminal) наподобие текстовых файлов и линейки скроллинга, предназначенные для скроллинга (прокрутки) информации, расположенной в поле скроллинга, текстовом элементе, списке строк (см. п. 4.4.7) или списке коллекции строк (см. п. 4.4.7). В окне можно помещать также потомки перечисленных выше стандартных элементов.

Примечание. К сожалению, непосредственно использовать стандартные элементы диалоговых окон, такие, как кнопки, строки ввода, триггерные и селективные списки и т. д. (см. п. 4.4), нельзя, т. к. цветовая палитра окна не предусматривает эти элементы. Исключение, пожалуй, составляет статический текст (см. п. 4.4.6), который можно поместить в окно, что и сделано в окне для отображения содержимого стека калькулятора (лл. 643-645). Если все-таки следует использовать эти элементы, то можно для обычного окна воспользоваться палитрой диалогового окна (см. п. 7).

4.2.1. Поле скроллинга (tScroller)

Поле скроллинга (тип tScroller) предназначено для размещения в нем текстовой информации, которую необходимо вывести в окно и которая, может быть, не умещается целиком в поле окна.

При инициализации поля скроллинга следует задать его размеры, линейки горизонтального и вертикального скроллинга:

```
constructor tScroller.Init(var Bounds: tRect;
                           AHScrollBar, AVScrollBar: pScrollBar);
```

где Bounds - размеры поля;
AHScrollBar, AVScrollBar - указатели на вертикальную и горизонтальную линейку скроллинга; если соответствующая линейка не нужна, задается значение nil.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.2.

Таблица 4.2. Параметры объекта tScroller при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+evCommand+ evBroadcast
State	sfVisible
HelpCtx	hcNoContext
Next	nil
Owner	nil

Кроме этого, при инициализации потомка tScroller могут быть заданы параметры поля скроллинга, в частности максимальный горизонтальный и вертикальный размер текстовой информации (размер строк и число строк), размещаемой в поле скроллинга.

При инициализации поля скроллинга окна для отображения содержимого стека калькулятора (тип tMyScroller - потомок tScroller) задаются размеры поля, линейка вертикального скроллинга (линейка горизонтального скроллинга не предусмотрена), а также указатель на индикатор, у которого используется содержимое стека (лл. 595-604).

Кроме этого, здесь задаются флаги gfGrowHiX и gfGrowHiY параметра GrowMode (см. п. 1.3), а также размер строк, равный горизонтальному размеру поля скроллинга, и число строк, равное числу заполненных элементов стека.

Следует также переопределить метод Draw объекта tScroller, для того чтобы наполнить поле скроллинга соответствующей информацией. Для этого необходимо вначале определить, какая часть информации должна быть отображена (исходя из текущего состояния линеек скроллинга), затем сформировать строки, которые необходимо вывести на экран и, наконец, поместить сформированные строки в поле скроллинга, с атрибутами цвета. Чтобы правильно заполнить поле скроллинга следует иметь в виду, что необходимо обязательно заполнить все поле, иначе незаполненные части будут отображать "мусор". С этой целью можно предложить следующую последовательность действий. Нужно взять строку достаточного размера (не меньше ширины поля скроллинга), заполнить ее пробелами, а затем добавить в нее информацию, которую следует вывести в конкретную строку. После создания строки ее следует поместить в "цветную" строку, содержащую не только символы, но и атрибуты цветов, используя процедуру MoveStr:

```
procedure MoveStr(var Dest; const Str: String; Attrs: Word);
```

где Dest - "цветная" строка типа tDrawBuffer (или эквивалентного типа), куда записывается информация;
Str - "нецветная" строка, которая переписывается в Dest;
Attrs - атрибуты цвета строки.

Тип tDrawBuffer представляет собой массив

```
type
  tDrawBuffer = array[0..MaxViewWidth -1] of Word;
```

где MaxViewWidth = 132 - максимальный размер отображаемой "цветной" строки, а каждый элемент массива отводится под символ, причем в старшем байте помещается сам символ (его код) а в младшем - атрибут цвета.

Наконец, полученную "цветную" строку следует поместить в соответствующее место поля скроллинга, используя, например, метод WriteLine объекта tView:

```
procedure tView.WriteLine(X, Y, W, H: Integer; var Buf);
```

где X, Y - координаты поля окна, откуда начинается размещение строки;
W - ширина поля для записи;

H - высота поля для записи (если эта высота больше 1, строка повторяется H раз);
Buf - буфер с переписываемой "цветной" строкой.

Такую последовательность необходимо выполнить для каждой строки поля скроллинга и даже для тех, которым не хватило информации из исходного источника информации.

Указанное выше поле скроллинга имеет метод Draw (лл. 605-631), в котором сначала определяются атрибуты цвета с помощью метода GetColor - л. 614 (см. п. 7.1), затем организуется цикл заполнения всего поля "цветными" строками.

4.2.2. Текстовый элемент буферированной информации (tTextDevice)

Текстовый элемент буферированной информации (тип tTextDevice) является потомком поля скроллинга и отличается от своего предка наличием двух абстрактных методов, которые читают из некоторого текстового буфера и записывают в текстовый буфер отображаемую информацию. Эти два метода для конкретного объекта следует переопределить с учетом свойств используемого текстового буфера и самого объекта.

4.2.3. Текстовый элемент для пассивного терминала (tTerminal)

Текстовый элемент для пассивного терминала (тип tTerminal) является наследником объекта tTextDevice и предназначен для отображения в поле скроллинга (а в случае необходимости и передачи в обратном направлении) информации от пассивного терминала, чаще всего из текстового файла этого терминала. Под пассивным терминалом здесь понимается терминал, хранящий информацию, чаще всего в виде текстовых файлов, но не имеющий собственных средств редактирования информации и управления экраном (например, таким терминалом может быть дисковод). Объект обладает рядом методов, позволяющих помещать в буфер необходимую часть информации, настраиваться на ту или иную строку текста, помещать информацию в поле скроллинга и выводить ее на экран и т. д.

Для передачи информации от терминала создается буфер, размер которого находится в параметре BufSize, а указатель на начало буфера - в параметре Buffer.

Инициализируется текстовый элемент для пассивного терминала конструктором Init:

```
constructor tTerminal.Init(var Bounds: tRect;
    AHScrollBar, AVScrollBar: pScrollBar; ABufSize: Word);
```

где Bounds - размеры поля;
AHScrollBar, AVScrollBar - указатели на вертикальную и горизонтальную линейку скроллинга; если соответствующая линейка не нужна, задается значение nil;
ABufSize - размер буфера терминала в байтах.

Во время инициализации создается элемент с указанными параметрами, а также буфер для передачи информации размером ABufSize. После инициализации устанавливаются значения параметров, приведенные в табл. 4.3.

Таблица 4.3. Параметры объекта tTerminal при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable
DragMode	dmLimitLoY
GrowMode	gfGrowHiX+gfGrowHiY
EventMask	evMouseDown+evKeyDown+evCommand+evBroadcast
State	sfVisible+sfCursorVis
HelpCtx	hcNoContext
Next	nil
Owner	nil
BufSize	ABufSize
Buffer	Указатель на начало буфера

После помещения текстового элемента для пассивного терминала в группу (как правило, в окно tWindow), будут установлены соответствующие значения параметров Next и Owner.

Перед тем как работать с текстовым элементом, его следует связать с текстовым файлом с помощью стандартной процедуры AssignDevice модуля TextView:

```
procedure AssignDevice(var T: Text; Screen: pTextDevice);
```

где T - текстовый файл;
Screen - текстовый элемент.

Работает процедура наподобие процедуры Assign пакета Turbo Pascal (Borland Pascal), только вместо конкретного физического файла используется текстовый элемент. Текстовый файл

Т следует открыть процедурой Reset или Rewrite, а потом с его помощью записывать информацию в текстовый элемент.

4.2.4. Линейка скроллинга (tScrollBar)

Линейка скроллинга (тип tScrollBar) предназначена для фиксации места нахождения отображаемой на экране в данный момент порции информации во всем объеме отображаемой информации (см. рис. 1.4). Линейка представляет собой либо горизонтальную полосу (линейка горизонтального скроллинга), либо вертикальную полосу (линейка вертикального скроллинга). Полоса оканчивается двумя стрелками, а в средней части расположен ползунок, указывающий относительное расположение отображаемой порции информации во всем объеме информации.

Используя этот ползунок, можно быстро переместиться в любое место отображаемой информации. Для этого необходмо подвести к нему курсор мыши, нажать клавишу мыши и, не отпуская ее, отбуксировать ползунок в нужное место линейки скроллинга. Если курсор мыши поместить не на ползунок, а в другое место линейки скроллинга, но не на завершающую стрелку, и нажать клавишу мыши, будет отображена следующая страница информации (размер страницы определяется параметром PgStep - см. ниже), находящаяся в соответствующем направлении по отношении к первоначально отображаемой порции. Наконец, если курсор мыши поместить на ту или иную завершающую стрелку и нажать клавишу мыши, будет осуществлено перемещение на один шаг в соответствующем направлении (размер шага определяется параметром ArStep - см. ниже).

Перемещать ползунок можно и с помощью клавиш клавиатуры (при некоторых условиях, о которых будет сказано ниже). Для горизонтальной линейки скроллинга можно использовать клавиши Left, Right, Ctrl+Left, Ctrl+Right, Home, End, для вертикальной линейки скроллинга - клавиши Up, Down, PgUp, PgDn, Ctrl+PgUp, Ctrl+PgDn. При этом клавиши Left, Right, Up, Down осуществляют перемещение на один шаг в соответствующем направлении, клавиши Ctrl+Left, Ctrl+Right, PgUp, PgDn - на одну страницу в соответствующем направлении, остальные клавиши осуществляют перемещение в начало или конец информации.

Линейка скроллинга не только качественно фиксирует место отображаемой порции информации, она также определяет точное смещение начала отображаемой порции по отношению к началу всего объема. При этом нумерация элементов информации идет с нуля.

Следует отметить 5 параметров объекта tScrollBar.

Min и Max - определяют минимальное и максимальное значение номера текущего (первого отображаемого) элемента информации.

ArStep - величина шага перемещения по информации в случае нажатия клавиш Left, Right, Up, Down или при нажатии клавиши мыши, если ее курсор находится на завершающей стрелке линейки скроллинга.

PgStep - размер страницы, на которую происходит перемещение в случае нажатия клавиш Ctrl+Left, Ctrl+Right, PgUp, PgDn или при нажатии клавиши мыши, если курсор мыши находится на линейке скроллинга, но не на завершающей стрелке или ползунке.

Value - текущий (первый отображаемый) элемент информации.

Инициализация линейки скроллинга осуществляется конструктором Init:

```
constructor tScrollBar.Init(var Bounds: tRect);
```

где Bounds - область, выделенная под линейку скроллинга в координатах ее владельца.

Если задан горизонтальный размер линейки скроллинга, равный единице, формируется вертикальная линейка скроллинга, в противном случае - горизонтальная линейка.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.4.

Таблица 4.4. Параметры объекта tScrollBar при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	0
DragMode	dmLimitLoY
GrowMode	gfGrowLoX+gfGrowHiX+gfGrowHiY - для вертикальной линейки, gfGrowLoY+gfGrowHiX+gfGrowHiY - для горизонтальной линейки
EventMask	evMouseDown+evKeyDown+evCommand
State	sfVisible
HelpCtx	hcNoContext
Next	nil
Owner	nil
Min	0
Max	0

ArStep	1
PgStep	1
Value	0

После помещения линейки скроллинга в группу (как правило, в окно или диалоговое окно) параметры Next и Owner принимают соответствующие значения.

Если задаваемые при инициализации параметры Min, Max, ArStep, PgStep или Value следует изменить, можно воспользоваться соответствующими методами объекта tScrollBar: методом SetParams:

```
procedure tScrollBar.SetParams(AValue, AMin,
                               AMax, APgStep, AArStep: Integer);
```

где AValue - номер текущего элемента информации;
AMin - минимальный номер элемента информации;
AMax - максимальный номер элемента информации;
APgStep - размер страницы;
AArStep - шаг перемещения,

позволяющим задать сразу все параметры;
методом SetRange:

```
procedure tScrollBar.SetRange(AMin, AMax: Integer);
```

где AMin - минимальный номер элемента информации;
AMax - максимальный номер элемента информации,

задающим границы номеров элементов информации;
методом SetStep:

```
procedure tScrollBar.SetStep(APgStep, AArStep: Integer);
```

где APgStep - размер страницы;
AArStep - шаг перемещения,

задающим размер страницы и шаг перемещения,
либо, наконец, методом SetValue:

```
procedure tScrollBar.SetValue(AValue: Integer);
```

где AValue - номер текущего элемента информации,
который задает номер текущего элемента.

Следует также отметить еще одну особенность линейки скроллинга. Линейка скроллинга не может активизироваться (находиться в выделенном состоянии - см. п. 1.6), т. к. у нее не установлен флаг ofSelectable параметра Options (см. п. 1.3). Поэтому она не может обычным образом реагировать на нажатие клавиш клавиатуры (см. п. 1.7). Для того чтобы линейка скроллинга все-таки реагировала на нажатие клавиш клавиатуры,

следует нарушить стандартный порядок обработки событий, установив флаг ofPostProcess параметра Options.

При создании линейки скроллинга для поля скроллинга окна отображения содержимого стека (лл. 646-649) сначала задаются ее размеры. Следует отметить, что ширина задаваемого поля равна единице, следовательно, создается вертикальная линейка скроллинга. Затем линейка скроллинга создается и инициализируется. После этого задается флаг ofPostProcess для того, чтобы она реагировала на клавиши клавиатуры, и, наконец, она помещается в окно. Линейка скроллинга здесь размещается в поле окна, в принципе ее можно наложить и на рамку окна.

Для упрощения создания линейки (линеек скроллинга) можно воспользоваться методом StandardScrollBar объекта tWindow:

```
function tWindow.StandardScrollBar(AOptions:
                                   Word): pScrollBar;
```

где AOptions - параметр, определяющий, какую линейку - горизонтальную или вертикальную - следует создавать; для горизонтальной линейки следует задать параметр sbHorizontal=0, а для вертикальной - sbVertical=1; если линейка скроллинга должна также реагировать на нажатие клавиш клавиатуры, к указанному параметру следует добавить параметр sbHandleKeyboard = 2;
результат - указатель на созданную линейку скроллинга.

Эта функция создает соответствующую линейку скроллинга, помещает ее в окно (размещает на рамке окна) и возвращает указатель на созданную линейку. Ей удобно пользоваться, когда в окне никаких элементов кроме поля скроллинга (или его потомка), нет. Пример использования метода StandardScrollBar приведен в п. 9.3.

4.3. Диалоговое окно (tDialog)

Диалоговое окно (тип tDialog) представляет специальный тип окна (является потомком tWindow) и предназначено в первую очередь для ввода значений параметров в программу.

Так же как и обычное окно, диалоговое окно может использовать один из трех типов палитры, только константы, определяющие палитру диалогового окна, другие:

```
dpBlueDialog    = 0 - синяя палитра диалогового окна;
dpCyanDialog    = 1 - голубая палитра диалогового окна;
dpGrayDialod    = 2 - серая палитра диалогового окна.
```

В отличие от окна диалоговое окно обычно используется в модальном режиме, хотя его можно использовать и в немодальном режиме, как это сделано у образа калькулятора программы-калькулятора (см. рис. 1.1). В связи с этим есть некоторые отличия в установлении параметров диалогового окна при его инициализации.

Инициализируется диалоговое окно конструктором Init:

```
constructor tDialog.Init(var Bounds: tRect;
                         ATitle: tTitleStr);
```

где Bounds - размеры диалогового окна;
ATitle - имя диалогового окна размером до 80 символов.

В отличие от конструктора обычного окна здесь не задается номер окна, что излишне, если оно находится в модальном режиме. После инициализации устанавливаются значения параметров, приведенные в табл. 4.5.

Таблица 4.5. Параметры объекта tDialog при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable+ofTopSelect+ofBuffered+ofVersion20 (только для версии 2.0)
DragMode	dmLimitLoY
GrowMode	0
EventMask	$FFFF
State	sfVisible+sfShadow
HelpCtx	hcNoContext
Flags	wfMove+wfClose
Palette	dpGrayDialog
Number	wnNoNumber
Next	nil
Owner	nil
Last	nil
Current	nil

Таким образом, в отличие от обычного окна у диалогового окна стандартно нет возможности изменять размеры, оно использует серую палитру, и диалоговое окно не имеет номера.

После того как диалоговое окно будет помещено в группу (в рабочую область - Desktop), соответствующим образом заполнятся параметры Next и Owner. После размещения в диалоговом окне компонент будут заполнены параметры Last и Current.

К другим особенностям диалогового окна можно отнести обработку им клавиш Esc и Enter. Если диалоговое окно находится в модальном режиме, нажатие клавиши Esc закрывает окно без сохранения введенной в окно информации (аналогично пиктограмме закрытия окна - см. п. 4.1). Нажатие клавиши Enter эквивалентно нажатию кнопки, выбираемой по умолчанию (см. п. 4.4.5). Как правило, этой кнопкой является кнопка "Ввод", "OK" или другая кнопка, формирующая команду cmOK, которая закрывает окно с сохранением внесенных в него изменений. Наряду с командой cmOK снимают модальность диалогового окна, если оно используется в модальном режиме, также команды cmCancel - прекращение работы окна без сохранения изменений, cmYes - утвердительный ответ на вопрос - и cmNo - отрицательный ответ на вопрос.

Для удобной работы с диалоговым окном в Turbo Vision имеется большое количество стандартных объектов, которые можно использовать в качестве элементов этого окна. Стандартные элементы позволяют вводить текстовую информацию, выбирать элементы различных списков, выбирать команды и т. д. Можно использовать как сами стандартные элементы, так и формировать их потомки, отличающиеся от предков какими-то свойствами. Все элементы должны быть помещены в диалоговое окно.

Так же как и для обычного окна, на практике используются потомки диалогового окна tDialog, у которых в первую очередь переопределяются конструктор Init для размещения в окне его компонент и метод HandleEvent для обработки соответствующих событий.

Работу с диалоговым окном в модальном режиме можно организовать следующим образом. Сначала следует создать и инициализировать диалоговое окно, при этом в него следует поместить все его элементы.

В подпрограмме Parameters (лл. 477-535) метода HandleEvent объекта tCalculator вначале создается диалоговое окно (л. 492) типа tParamDialog. Конструктор этого окна (лл. 555-594) сначала вызывает наследуемый конструктор для создания диалогового окна, устанавливает соответствующие флаги, затем формирует элементы окна: строку ввода названия калькулятора (см. п. 4.4.1), селективный список выбора системы счисления (см. п. 4.4.4), строку ввода размера стека (см. п. 4.4.1) и кнопки управления (см. п. 4.4.5); затем с помощью метода SelectNext (см. п. 1.6) после размещения последнего элемента активи-

зирует первый размещенный элемент - строку ввода названия калькулятора.

Так как диалоговое окно типа tParamDialog обрабатывает события стандартным образом, метод HandleEvent для него не переопределяется.

Внешний вид диалогового окна этого типа изображен на рис. 2.1.

После создания диалогового окна следует подготовить исходные данные, которые необходимо передать диалоговому окну. Данные готовятся в общем случае в виде параметра типа-записи (см. п. 2.4).

В подпрограмме Parameters (лл. 477-525) после создания диалогового окна формируются начальные значения параметров (лл. 493-500): название калькулятора, текущая система счисления и размер стека (в символьной форме) - с размещением их в переменной Par (лл. 479-483)

Далее диалоговое окно размещается в рабочей области с помощью метода ExecuteDialog (см. п. 2.8) и осуществляется работа с ним. При завершении работы с диалоговым окном анализируется, какое происходит завершение - с сохранением введенных изменений или без сохранения. Если введенные изменения следует сохранить, они возвращаются в том же параметре, в котором передавались исходные данные. Перед выходом с сохранением введенных изменений при наличии контроля вводимых данных (см. п. 8.1) осуществляется этот контроль, и, если данные недопустимы, работа с окном не прекращается.

В подпрограмме Parameters (лл. 477-525) диалоговое окно включается в работу методом ExecuteDialog (см. п. 501) и, если работа с ним завершается не по команде cmCancel, которая соответствует завершению работы без сохранения введенных изменений, формируется новая система счисления, новый размер стека и новый заголовок калькулятора (лл. 501-522); одновременно с помощью метода EnableOrDisableCommands (лл. 466-475, 521) в зависимости от выбранной системы счисления разрешаются или запрещаются команды, связанные с кнопками калькулятора, которые соответствуют цифрам. В конце работы подпрограммы перерисовывается образ калькулятора (л. 523) для отображения внесенных изменений.

Если окно используется в немодальном режиме, последовательность работы с ним аналогична последовательности работы с обычным окном (см. п. 4.1).

Объект типа tCalculator (лл. 44-50, 435-550), являющийся потомком tDialog, используется в немодальном режиме, т. к. размещается в рабочем поле методом InsertWindow (л. 725). Метод Init этого объекта после обращения к наследуемому методу Init для создания диалогового окна формирует кнопки, являющиеся клавишами калькулятора, и индикатор калькулятора, в котором отображаются вводимые числа и результаты вычислений. Здесь же устанавливаются исходные значения параметров: название калькулятора - 'Калькулятор' (л. 440), десят-ичная система счисления (лл. 464 и 140 в конструкторе индикатора калькулятора), размер стека, равный 6 (л. 139 в конструкторе индикатора калькулятора).

Метод HandleEvent дополнительно к наследуемому методу обрабатывает команды cmParamrters (параметры калькулятора) и cmStack (содержимое стека калькулятора) - лл. 542, 543, - которые пользователь выбирает с помощью меню (см. п. 3.1).

4.4. Элементы диалоговых окон

Как уже говорилось, в диалоговых окнах можно использовать большой набор стандартных элементов (или их потомков), позволяющих вводить в программу разнообразную информацию. К ним относятся:

- строки ввода;
- метки;
- протоколы;
- селективные списки;
- триггерные списки;
- кнопки;
- статические и параметрические тексты;
- списки строк;
- линейки скроллинга.

Все они либо непосредственно, либо опосредованно являются потомками объекта tView и, следовательно, обладают всеми свойствами этого объекта (см. п. 1.3). Линейки скроллинга рассмотрены в п.4.2.4, остальные элементы будут кратко рассмотрены ниже. Как правило, методы стандартных элементов не переопределяются, и мы эту возможность рассматривать не будем.

4.4.1. Строка ввода (tInputLine)

Строка ввода (тип tInputLine) предназначена для ввода в программу символьной информации. Фактически строка ввода представляет собой простейший редактор строки символов и обладает большим набором возможностей, имеющихся у редакторов (добавление и удаление символов в любом месте строки, режимы вставки и замены символов, скроллинг информации, если она не умещается в поле строки ввода, выделение символов и т. д.). Основным элементом строки ввода является буфер, представляющий собой строку, в который помещаются вводимые символы (указатель на этот буфер находится в переменной Data).

Инициализируется строка ввода методом Init:

```
constructor tInputLine.Init(var Bounds: tRect;
                            AMaxLen: Integer);
```

где Bounds - размер поля строки ввода;
AMaxLen - размер буфера.

Высота поля строки ввода должна равняться единице (хотя можно задать и другую высоту, делать это не следует, т. к. это приведет к созданию нескольких одинаковых строк одна под другой). Ширина поля может быть любой, но не менее 3, причем ширина поля может быть меньше размера строки, отводимой под буфер: в этом случае будет обеспечен при необходимости скроллинг информации в строке ввода. Размер буфера может иметь любое значение до 255 (для буфера формируется строка типа string[AMaxLen]. После создания строки ввода непосредственно изменять размеры буфера нельзя.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.6.

Таблица 4.6. Параметры объекта tInputLine при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable+ofFirstClick+ ofVersion20 (только для версии 2.0)
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+evCommand
State	sfVisible+sfCursorVis
HelpCtx	hcNoContext
Next	nil
Owner	nil

После размещения строки ввода в группе (как правило, в диалоговом окне) устанавливаются соответствующие значения параметров Next и Owner.

В диалоговом окне ввода параметров калькулятора (тип tParamDialog) используются две строки ввода: строка ввода названия калькулятора (лл. 563-566) и размера стека (лл. 577-581).

Так как строка ввода работает только с символьной информацией, то перед вводом в нее какого-либо числа оно должно быть предварительно преобразовано в символьную форму, а при получении информации из строки ввода - обратно преобразовано в двоичную форму.

При подготовке исходных данных для диалогового окна ввода параметров калькулятора число, определяющее размер стека, преобразуется в символьную форму (л. 500), а после работы с диалоговым окном полученная информация преобразуется обратно в двоичную форму (л. 509).

Примечание. Можно и не преобразовывать числа в символьную форму, но тогда следует переопределить методы SetData и GetData, передающие в строку ввода информацию и получающие из нее информацию. В этих новых методах все равно придется преобразовывать информацию в символьную форму и наоборот. Какой выбрать вариант - дело вкуса программиста.

К строке ввода можно подключить метку (см. п. 4.4.2), протокол (см. п. 4.4.3) и анализатор введенной информации (см. п. 8.1).

4.4.2. Метка (tLabel)

Метка (тип tLabel) всегда связана с каким-либо другим элементом окна и представляет собой текст, поясняющий смысл соответствующего объекта. Кроме этого, с помощью метки можно довольно просто активизировать связанный с ней элемент. Для этого следует поместить курсор мыши на поле метки и нажать клавишу мыши. При этом будет активизирована не метка, а связанный с ней элемент. Так как сама метка нас, как правило, после ее создания уже не интересует, обычно не вводится переменная, являющаяся указателем на введенную метку.

Отметим один параметр метки - Link. Он содержит указатель на объект, с которым связана метка.

Конструктор метки Init имеет следующий вид:

```
constructor tLabel.Init(var Bounds: tRect;
                        const AText: string; ALink: pView);
```

где Bounds - размер поля метки;
AText - текст метки;
ALink - указатель на объект, с которым связана метка.

Метка располагается либо над объектом, с которым она связана, либо слева от него. Высота поля метки должна быть равна единице (хотя можно задать и другую высоту, делать это не следует, т. к. Turbo Vision не поддерживает изображение такой метки). Если текст метки состоит из цифр и латинских букв, то в нем можно выделить какую-либо цифру или букву, поместив ее между символами ~ ~. Такая буква на экране будет выделена особым цветом, а нажав клавишу с этим символом, можно активизировать элемент, с которым связана метка.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.7.

Таблица 4.7. Параметры объекта tLabel при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofPreProcess+ofPostProcess
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+evCommand+evBroadcast
State	sfVisible
HelpCtx	hcNoContext
Link	ALink
Next	nil
Owner	nil

После размещения метки в группе (как правило, в диалоговом окне) устанавливаются соответствующие значения параметров Next и Owner.

Строка ввода названия калькулятора диалогового окна параметров калькулятора имеет метку 'Название:' (лл. 567-568). Можно заметить, что указатель на создаваемую метку в программе не запоминается.

Метка не требует передачи ей параметра, она также не формирует никакого параметра после работы с ней.

4.4.3. Протокол (tHistory)

Протокол (тип tHistory) всегда связан со строкой ввода и предназначен для сохранения в своем буфере вводимой в строку

ввода информации. Если эту информацию потребуется повторно использовать, можно вызвать необходимую информацию из буфера, а не набирать на клавиатуре. Для вызова ранее набираемой информации следует, находясь в строке ввода, либо подвести курсор к полю пиктограммы протокола и нажать клавишу мыши, либо нажать клавишу Down клавиатуры. В результате таких операций откроется окно протокола, содержащее вводимые ранее строки в данную строку ввода. Следует выбрать интересующую строку либо с помощью мыши, либо с помощью клавиш Up и Down. Далее необходимо либо дважды нажать клавишу мыши, либо нажать клавишу Enter клавиатуры. Выбранная информация появится в окне ввода.

В дополнение к параметрам объекта tView протокол имеет еще два параметра: HistoryID - номер протокола, определяющий буфер с сохраняемой информацией, и Link - указатель на строку ввода, с которой связан протокол.

Протокол инициализируется методом Init:

```
constructor tHistory.Init(var Bounds: tRect;
                          ALink: pInputLine; AHistoryID: Word);
```

где Bounds - поле для размещения пиктограммы протокола;
ALink - указатель на строку ввода, с которым связан протокол;
AHistoryID - номер протокола - число, определяющее данный тип протокола.

Обычно протокол размещается за строкой ввода, к которой он относится. Ширина поля должна равняться двум, высота - единице. Как правило, для протокола, так же как и для метки, необязательно вводить переменную, содержащую указатель на него. Несколько протоколов, относящихся к разным строкам ввода, могут иметь один и тот же номер AHistoryID. В этом случае у них будет один общий список вводимых строк, независимо от того, через какую строку ввода они вводились.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.8.

Таблица 4.8. Параметры объекта tHistory при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X,Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofPostProcess
DragMode	dmLimitLoY
GrowMode	0

EventMask	evMouseDown+evKeyDown+evCommand+evBroadcast
State	sfVisible
HelpCtx	hcNoContext
HistoryID	AHistoryID
Link	ALink
Next	nil
Owner	nil

После размещения протокола в группе (как правило, в диалоговом окне) устанавливаются соответствующие значения параметров Next и Owner.

Строка ввода названия калькулятора диалогового окна параметров калькулятора имеет протокол (лл. 569-570). Протоколу присвоен номер 100. Можно заметить, что указатель на протокол в программе не запоминается.

Протокол не требует передачи ему какого-либо параметра, помещаемые в строку ввода данные автоматически сохраняются в его буфере. Нельзя от протокола и получить какие-либо данные после окончания работы с ним.

4.4.4. Селективный (tRadioButtons) и триггерный (tCheckBoxes) списки

Селективный (тип tRadioButtons) и триггерный (тип tCheckBoxes) списки очень похожи друг на друга, поэтому рассмотрим их одновременно. В связи с их большим сходством у них есть общий предок (tCluster), заключающий в себе общие свойства.

Селективный список позволяет выбрать один-единственный элемент из этого списка (см., например, рис. 2.1). Количество элементов в списке практически не ограничено (на самом деле ограничено числом 65536).

Триггерный список позволяет выбрать из списка любое число элементов (или ни одного), однако при стандартном использовании число элементов этого списка ограничено шестнадцатью. Это количество можно увеличить до 32, но в этом случае следует модифицировать сам объект.

У предка этих двух списков - tCluster - следует отметить параметр Value типа Longint, где фиксируется номер выделенного элемента селективного списка (нумерация начинается с нуля) или совокупность выделенных элементов триггерного списка (под каждый элемент выделяется один бит информации), и параметр EnableMask типа Longint, который позволяет маскировать и демаскировать первые 32 элемента списка. Если флаг, соответствующий элементу списка (а нумерация элементов на-

чинается с нуля) установлен, данный элемент списка может быть выбран, в противном случае - нет. Автоматически при инициализации списка в параметр записывается число $FFFFFFFF, т. е. все элементы списка доступны для выбора.

Оба списка инициализируются одним и тем же конструктором Init объекта tCluster:

```
constructor tCluster.Init(var Bounds: tRect;
                          AStrings: pSItem);
```

где Bounds - поле, выделяемое под образ списка;

AStrings - указатель на список информационных строк.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.9.

Таблица 4.9. Параметры объекта tCluster при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable+ofFirstClick+ ofPreProcess+ofPostProcess+ ofVersion20 (только для версии 2.0)
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+evCommand
State	sfVisible
HelpCtx	hcNoContext
Value	$FFFFFFFF
Next	nil
Owner	nil

После размещения селективного или триггерного списка в группе (как правило, в диалоговом окне) устанавливаются соответствующие значения параметров Next и Owner.

Информационные строки, являющиеся названиями выбираемых пунктов списка, представляют линейный список строк с указанием следующего элемента в каждом элементе списка:

```
type tSItem = record
  Value: pString;                {Указатель на строку символов}
  Next: pSItem;                  {Указатель на следующий элемент}
end;
```

Чтобы создать элемент этого типа, можно воспользоваться стандартной функцией NewSItem:

```
function NewSItem(const Str: String;
                  ANext: pSItem): pSItem;
```

где Str - строка символов элемента;
ANext - указатель на следующий элемент списка информационных строк;
результат - указатель на созданный элемент.

Для задания нескольких информационных строк, как это и бывает у селективных или триггерных списков, следует многократно воспользоваться этой функцией.

У диалогового окна параметров калькулятора имеется селективный список выбора системы счисления (лл. 571-574), содержащий четыре информационные строки.

При создании списка следует выделять поле Bounds достаточного размера для размещения строк информации. Информационные строки автоматически размещаются в выделенном поле. Сначала делается попытка разместить их в один столбец, если это не удается, то в два столбца и т. д.

Чтобы выбрать элемент списка, можно воспользоваться мышью, подведя курсор к соответствующей строке и нажав клавишу мыши, либо клавиатурой, используя клавиши направления (Left, Right, Up, Down). В триггерном списке при использовании клавиш направления следует завершить выбор элемента списка нажатием клавиши пробела.

В строке символов каждого элемента списка можно выделить какую-либо латинскую букву или цифру, поместив ее между символами ~ ~. Эта буква или цифра будет выделена на экране особым цветом, и можно выбрать соответствующий элемент списка нажатием клавиши клавиатуры с этим символом.

У селективного списка выбора системы счисления калькулятора (см. выше) в информационных строках выделены цифры 2, 8 1 и 6, что позволяет, нажав клавишу с соответствующей цифрой, выбрать требуемую систему счисления.

Следует иметь в виду, что параметры контекстной помощи для селективных и триггерных списков задаются для каждого элемента списка, хотя при инициализации указывается только номер первого элемента (л. 573). При создании справочной системы (см. п. 5.2) все фрагменты, относящиеся к одному списку, должны идти непосредственно один за другим либо в случае одного фрагмента для всего списка этот фрагмент должен быть помечен числом идентификаторов, равным числу элементов списка (см., например, приложение Д, фрагменты CalcSystem2, CalcSystem8, CalcSystem10 и CalcSystem16).

Как и для строки ввода, для триггерного или селективного списка можно задать метку (см. п. 4.4.2).

У селективного списка выбора системы счисления калькулятора (см. выше) имеется метка 'Система счисления:' (лл. 575-576).

Для передачи селективному или триггерному списку значения его начального состояния используется параметр типа Word, в котором передается номер исходного выделенного элемента для селективного списка (нумерация начинается с нуля) или множество выделенных элементов для триггерного списка (под каждый элемент отводится один бит информации). Аналогичным же образом возвращается информация при завершении работы со списком.

Существует разновидность триггерного списка - триггерный список с многими состояниями (тип tMultiCheckBoxes), отличающийся от обычного триггерного списка тем, что каждый его элемент может находиться не в двух, а в большем числе состояний (четырех, восьми, шестнадцати, тридцати двух), правда, за счет соответствующего уменьшения максимального числа элементов списка.

4.4.5. Кнопка (tButton)

Кнопка (тип tButton) позволяет выбрать команду, которая с ней связана. При выборе команды работа с диалоговым окном часто прекращается.

Кнопка может иметь флаги, находящиеся в ее параметре Flags и задаваемые при инициализации. Эти флаги (и константы, соответствующие единице в конкретном разряде параметра Flags) следующие.

0-й бит (bfDefault) - кнопка, выбираемая по умолчанию. Одна из кнопок диалогового окна может иметь этот флаг; такая кнопка может быть выбрана наряду с обычными вариантами (с помощью мыши или клавиши табуляции) также и нажатием клавиши Enter.

1-й бит (bfLeftJust) - текст кнопки выравнивается по левому краю кнопки (при отсутствии флага текст центрируется).

2-й бит (bfBroadcast) - кнопка формирует сообщение вместо команды. Если установлен флаг, кнопка формирует сообщение (см. п. 1.7) для своего владельца:
Message(Owner, evBroadcast, Command, @Self);

3-й бит (bfGrabFocus) - фокусирование кнопки при ее активизации мышью. Обычно этот флаг сброшен, т. к. для кнопок, при активизации которых прекращается работа с окном, значение его безразлично.

Если ни один из флагов не установлен (что соответствует константе bfNormal=0), кнопка называется "нормальной".

Инициализируется кнопка конструктором Init:

```
constructor tButton.Init(var Bounds: tRect;
                         ATitle: tTitleStr; ACommand:Word;AFlags:Byte);
```

где Bounds - поле, выделяемое под кнопку;
ATitle - текст, размещаемый на кнопке;
ACommand - код команды, связанной с кнопкой;
AFlags - задаваемые флаги кнопки.

Выделяемое под кнопку поле должно быть достаточных размеров, чтобы в нем отобразилась вся кнопка, включая размещенный на ней текст, и ее тень. В связи с этим высоту кнопки не следует брать меньше двух, а ширина в основном определяется размещаемым на кнопке текстом. Следует иметь в виду, что текст кнопки всегда размещается в одной строке.

Строка символов, помещаемых на кнопке, может быть практически любого размера в пределах ширины кнопки.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.10.

Таблица 4.10. Параметры объекта tButton при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofSelectable+ofFirstClick+ ofPreProcess+ofPostProcess
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+ evCommand+evBroadcast
State	sfVisible+sfDisable (если запрещена команда кнопки)
HelpCtx	hcNoContext
Flags	AFlags
Next	nil
Owner	nil

После размещения кнопки в группе (как правило, в диалоговом окне) устанавливаются соответствующие значения параметров Next и Owner. Так как кнопка в зависимости от того, разрешена или запрещена связанная с ней команда, отображается разными цветами (см. ниже), флаг sfShadow у метки не устанавливается, а тень прорисовывается у кнопки особым образом.

У диалогового окна параметров калькулятора имеются две кнопки (лл. 584-592) - ввода параметров и отмены работы с диалоговым окном. Первая кнопка связана со стандартной командой cmOK, которая вызывает завершение работы с окном с фиксацией введенных изменений, а вторая - со стандартной командой cmCancel - завершение работы с окном без сохранения внесенных изменений.

Выбрать кнопку можно с помощью мыши, подведя курсор мыши к кнопке и нажимая клавишу мыши, либо с помощью клавиши Tab (или Shift+Tab), а затем клавиши Enter. При выборе той или иной кнопки она формирует событие-команду с кодом команды, который был задан кнопке при инициализации. В качестве параметра кнопка передает указатель на себя. Таким образом, формируется событие E (см. п. 1.7):

```
E.What:= evCommand;
E.Command:= Command;                {Command-параметр с кодом команды}
E.InfoPtr:= @Self;                  { @Self - указатель на себя }
```

У кнопок калькулятора, соответствующих его клавишам (лл. 442-460), установлены флаги bfBroadcast + bfGrabFocus. Первый флаг позволяет передать диалоговому окну код команды нажатой клавиши для его дальнейшей обработки, а второй фокусирует активизируемую мышью кнопку, чтобы зафиксировать последнюю нажатую кнопку.

Если команда, связанная с кнопкой, является запрещенной, например с помощью метода DisableCommands (см. п. 1.8), то такая кнопка выделяется на экране особым цветом и ее нельзя активизировать никакими средствами.

Образ калькулятора (см. рис. 1.1) имеет кнопки (клавиши) - "A", "B", "C", "D", "E" и "F". При десятичной системе счисления такие цифры недопустимы, поэтому команды, соответствующие этим кнопкам, запрещены (лл. 521, 466-475). Если будет использоваться шестнадцатеричная система счисления, то соответствующие команды будут разрешены, а клавиши - доступны для работы с ними.

Кнопка не требует передачи ей какого-либо параметра, в свою очередь, она также не формирует никаких параметров, кроме команды (или сообщения).

4.4.6. Статический (tStaticText) и параметрический (tParamText) тексты

Статический текст (тип tStaticText) предназначен для размещения в диалоговом окне различной поясняющей информации (заголовков, пояснений, подсказок и т. д.).

Инициализация статического текста осуществляется методом Init:

```
constructor tStaticText.Init(var Bounds: tRect;
                             const AText: String);
```

где Bounds - поле, выделенное под статический текст;

AText - размещаемый текст.

Поле, выделяемое под статический текст, должно быть достаточных размеров, чтобы можно было расположить все слова текста, в противном случае часть его будет скрыта. Текст автоматически делится на слова и размещается в выделенном поле без переносов внутри слов. В тексте можно использовать два служебных символа: ^C (эквивалент Ctrl+C) для центрирования по строкам последующего текста в выделенном поле и ^M (эквивалент Ctrl+M) для безусловного переноса последующего текста на следующую строку.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.11.

Таблица 4.11. Параметры объекта tStaticText при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X, Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	0
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+evCommand
State	sfVisible
HelpCtx	hcNoContext
Next	nil
Owner	nil

После размещения статического текста в группе (как правило, в диалоговом окне) устанавливаются соответствующие значения параметров Next и Owner.

В окне просмотра содержимого стека калькулятора используется статический текст (лл. 643-645) "Число заполненных элементов...". Здесь используется служебный символ ^C для того, чтобы текст центрировался в выделенном поле.

Статический текст никаким образом активизировать нельзя, не нужно также и передавать ему какие-либо параметры.

Разновидностью статического текста является параметрический текст (тип tParamText), позволяющий модифицировать выводимый текст в зависимости от задаваемых параметров (например, "Имя файла ...", где вместо точек может быть помещено любое имя). Для того чтобы указать, в каком месте текста следует разместить параметры и какого типа, в исходном тексте используются спецификации там, где должны быть размещены параметры. Спецификация имеет формат:

```
%[-][nnn]X,
```

где % - признак начала спецификации;
- - означает, что информация выравнивается
по левому краю выделенного под параметр поля;
без этого символа информация выравнивается
по правому краю поля;
nnn - размер выделенного под параметр поля; если размер
не указан, выделяется необходимое
для размещения параметра поле;
X - может принимать следующие значения:
s - параметр - указатель на строку;
d - параметр - число типа Longint выводится
в десятичной системе;
c - младший байт параметра представляет собой символ;
x - параметр - число типа Longint выводится
в шестнадцатеричной системе;
- индекс параметра равен nnn.

Параметры, размещаемые в строке, компонуются в запись, поля которой представляют собой либо величины типа Longint, либо указатели. Из этой записи параметры в строку выбираются поочередно, если только не используется спецификация #.

Инициализируется параметрический текст конструктором Init:

```
constructor tParamText.Init(var Bounds: tRect;
                            const AText: string; AParamCount: Integer);
```

где Bounds - поле, выделенное под параметрический текст;
AText - исходный текст со спецификациями параметров;
AParamCount - число параметров текста.

При инициализации объекта tParamText задаются те же параметры, что и при инициализации статического текста.

Параметры для модификации параметрического текста передаются с помощью метода SetData (см. п. 2.4):

```
procedure tParamText.SetData(var Rec); virtual;
```

где Rec - передаваемая запись с параметрами.

Получив запись с параметрами, объект tParamText вызывает стандартную процедуру FormatStr:

```
procedure FormatStr(var Result: string;
                    const: Format: string; var Params);
```

где Result - сформированная строка;
Format - исходная строка;
Params - запись с параметрами.

Эта процедура формирует из исходной строки Format, содержащей спецификации форматов параметров, новую строку Result, в которой все спецификации заменены на соответствующие параметры в виде, определяемом спецификацией.

Если в диалоговое окно следует поместить параметрический текст с именем файла, то это можно оформить следующим образом.

Пусть имя файла находится в переменной FileName, а запись передаваемых параметрическому тексту параметров - в переменной Rec:

```
var FileName: string[16];
var Rec record
        Par: Pointer
    end;
```

Для создания и инициализации параметрического текста с указателем ParamText и последующего размещения в диалоговом окне типа tMyDialog можно представить фрагмент конструктора Init диалогового окна следующим образом:

```
constructor tMyDialog.Init(var Bounds: tRect;
                           ATitle: tTitleStr);
begin
    . . .
  Bounds.Assign(5,2,35,4);     {Выделение области под объект}
  ParamText:=New(pParamText,Init(Bounds,'Имя файла:%s', 1);
```

```
                                {Инициализация параметрического текста}
    Insert(ParamText);              {Размещение в диалоговом окне}
    Rec.Par:= @FileName;        {Формирование записи параметров}
    ParamText^.SetData(Rec);                    {Задание параметров}
        . . .
end;
```

4.4.7. Списки строк

Списки строк в диалоговом окне напоминают поле скроллинга и его потомков для обычных окон и предназначены для просмотра тех или иных списков.

Список строк (тип tListViewer) является абстрактным объектом, из него можно получить реальный объект, который будет работать с конкретным списком информации. Он может иметь как вертикальную, так и горизонтальную линейку скроллинга (см. п. 4.2.4), механизм перемещения по рассматриваемой информации как по вертикали, так и по горизонтали, возможность выделения элементов просматриваемой информации.

Непосредственным потомком списка строк является список коллекции строк (тип tListBox), предназначенный в первую очередь для просмотра коллекции строк (см. п. 8.2), хотя может при некоторой модификации использоваться и с другими коллекциями.

В отличие от списка строк список коллекции строк обеспечивает скроллинг только в вертикальном направлении (предполагается, что просматриваемые строки полностью умещаются по горизонтали в выделенном поле).

Инициализируется список коллекции строк с помощью конструктора Init:

```
constructor tListBox.Init(var Bounds: tRect;
                          ANumCols: Word; AScrollBar: pScrollBar);
```

где Bounds - поле, выделенное под список;
ANumCols - число колонок, в которые выводятся строки;
AScrollBar - указатель на линейку вертикального скроллинга.

После инициализации устанавливаются значения параметров, приведенные в табл. 4.12.

Таблица 4.12. Параметры объекта tListBox при инициализации

Параметр	*Значение*
Origin	(Bounds.A.X, Bounds.A.Y)
Size	(Bounds.B.X-Bounds.A.X,Bounds.B.Y-Bounds.A.Y)
Cursor	(0, 0)
Options	ofFirstClick+ofSelectable
DragMode	dmLimitLoY
GrowMode	0
EventMask	evMouseDown+evKeyDown+ evCommand+evBroadcast
State	sfVisible
HelpCtx	hcNoContext
Next	nil
Owner	nil

После размещения списка коллекции строк в группе (как правило, в диалоговом окне) устанавливаются соответствующие значения параметров Next и Owner.

Если список предназначен для работы с коллекцией строк (см. п. 8.2), то никакие методы объекта переопределять не следует, если же используется коллекция какого-либо другого типа, необходимо переопределить метод GetText:

```
function tListBox.GetText(Item: Integer;
                          MaxLen: Integer): string; virtual;
```

где Item - номер элемента коллекции;
MaxLen - максимальное число символов в формируемой строке;
результат - сформированная строка информации, которая будет выведена на экран для элемента коллекции.

Так как при инициализации списка коллекции строк не указывается конкретная коллекция, с которой будет происходить работа, следует при начале работы со списком передать ему в качестве параметра этот список. Передаваемый параметр следует оформить в виде записи, первое поле которого представляет собой указатель на используемую коллекцию, а второе поле - номер выделенного в коллекции элемента (нумерация начинается с нуля), например:

```
type
   tRec = record
      List: pCollection;
      Selection: Word;
   end;
```

Вместо окна для просмотра содержимого стека типа tStackWindow (лл. 65-69) и используемого в нем поля скроллинга типа tMyScroller (лл. 59-64) можно использовать диалоговое окно просмотра содержимого стека со списком коллекции строк, приспособленным для просмотра коллекции чисел повышенной точности типа tNumberCollection (лл. 16-19):

```
{ Список коллекции чисел }
pMyListBox = ^tMyListBox;                                          { 59}
tMyListBox = object(tListBox)                                      { 60}
   Calculator: pCalculator;                                        { 61}
   constructor Init(var Bounds: tRect; ANumCols:
      Word; AScrollBar: pScrollBar; Calc: pCalculator);            { 62}
   function GetText(Item: Integer;
            MaxLen: Integer): string; virtual;                     { 63}
end;                                                               { 64}

{ Диалоговое окно стека }
pStackDialog = ^tStackDialog;                                      { 65}
tStackDialog = object(tDialog)                                     { 66}
   constructor Init(Calc: pCalculator);                            { 67}
end;                                                               { 68}
```

Содержание методов списка коллекций чисел выглядит следующим образом.

```
{*****************************************}
{*            Объект tMyListBox          *}
{*****************************************}

constructor tMyListBox.Init(var Bounds: tRect;
            ANumCols: Word; AScrollBar: pScrollBar;
                        Calc: pCalculator);                        {595}
begin                                                              {596}
   inherited Init(Bounds, ANumCols, AScrollBar);                   {597}
   Calculator:= Calc;                                              {598}
end;                                                               {599}

function tMyListBox.GetText(Item: Integer;
                        MaxLen: Integer): string;                  {600}
var S: string[3];                                                  {601}
      Number: Double;                                              {602}
      Sign: Char;                                                  {603}
      NumberStr: string[15];                                       {604}
begin                                                              {605}
```

```
if List <> nil then                                              {606}
  begin                                                          {607}
    { Номер элемента }
    Str(Item, S)                                                 {608}
    { Число }
    Number:= pDouble(List^.At(Item))^;                           {609}
    { Получение компонент числа }
    Calculator^.CalcDisplay^.SetDisplay(
                    Number, Sign, NumberStr);                    {610}
    { Формирование строки }
    GetText:= S + '          ' + Sign +
                                  NumberStr);                    {611}
  end                                                            {612}
  else                                                           {613}
    GetText:= '';                                                {614}
end;                                                             {615}
```

Метод Init объекта tStackDialog выглядит следующим образом:

```
{*****************************************}
{*          Объект tStackDialog          *}
{*****************************************}
constructor tStackDialog.Init(Calc: pCalculator);                {632}
var R: tRect;                                                    {633}
      S: string[3];                                              {634}
      ScrollBar: pScrollBar;                                     {635}
      ListBox: pMyListBox;                                       {636}
      Data: record                                               {637}
        List: pNumberCollection;                                 {638}
        Selection: Word;                                         {639}
      end;                                                       {640}
begin                                                            {641}
  { Формирование окна }
  R.Assign(0, 0, 50, 14);                                        {642}
  inherited Init(R, 'Стек');                                     {643}
  Options:= Options or
       (ofCentered + ofSelectable + ofFirstClick);               {644}
  Flags:= wfMove + wfClose;                                      {645}

  { Формирование статического текста }
  R.Assign(9, 2, 41, 3);                                         {646}
  Str(pCalculator(DeskTop^.Current)^.
                  CalcDisplay^.Stack^.Count, S);                 {647}
```

```
  Insert(New(pStaticText,Init(
     R, ^C'Число заполненных элементов: ' + S)));                    {648}

  { Формирование линейки скроллинга }
  R.Assign(45, 4, 46, 10);                                           {649}
  ScrollBar:= New(pScrollBar, Init(R));                              {650}
  Insert(ScrollBar);                                                 {651}

  { Формирование списка коллекции чисел }
  R.Assign(10, 4, 40, 10);                                           {652}
  ListBox:= New(pMyListBox,
                Init(R, 1, ScrollBar, Calc));                        {653}

  { Задание начальных условий списка }
  Data.List:= pCalculator(DeskTop^.Current)^.
                        CalcDisplay^.Stack;                          {654}
  Data.Selection:= pCalculator(
       DeskTop^.Current)^.CalcDisplay^.Stack^.Count - 1;             {655}
  ListBox^.SetData(Data);                                            {656}
  ListBox^.Options:= ListBox^.Options or ofFramed;                   {657}
  ListBox^.HelpCtx:= 1200;                                           {658}
  Insert(ListBox);                                                   {659}

  { Формирование кнопки управления }
    R.Assign(20, 11, 30, 13);                                        {660}
    Insert(New(pButton, Init(R, 'Ввод', cmOk, bfDefault)));          {661}

    SelectNext(False);                                               {662}
end;                                                                 {663}
```

Наконец, подпрограмма вывода содержимого стека (см. пп. 527-536) будет такой:

```
  { Вывод содержимого стека }
  procedure StackContents;                                           {526}
  var StackDialog: pStackDialog;                                     {527}
  begin                                                              {528}
     StackDialog:= New(pStackDialog, Init(@Self));                   {529}
     Application^.ExecuteDialog(StackDialog, nil);                   {530}
  end;                                                               {531}
```

Какой из двух вариантов (обычное окно или диалоговое окно) выбрать - во многом дело вкуса разработчика.

4.5. Стандартные окна

В Turbo Vision имеется большое количество стандартных окон,. имеющих специальное назначение: окно выбора файла из каталога, задания цветовой палитры и т. д. Мы рассмотрим здесь только стандартные информационные окна, которые создаются с помощью функций MessageBox и MessageBoxRect. Заголовок функции MessageBox имеет вид:

```
function MessageBox(const Msg: string; Params:
                                    Pointer; AOptions: Word): Word;
```

где Msg - размещаемое в окне сообщение;
Params - указатель на параметры сообщения;
AOptions - флаги (см. ниже);
результат - код команды, связанной с нажатой кнопкой диалогового окна.

Функция формирует окно размером 40 x 60, в котором помещается сообщение Msg, скорректированное параметрами, находящимися по адресу, определяемому значением Params (аналогично тому, как это происходит в параметрическом тексте, - см. п. 4.4.6), и набор кнопок, определяемый параметром AOptions. Параметр AOptions определяет также тип окна (его название).

Для функции MessageBox можно задать следующие флаги (или их сочетание):

mfWarning = 0 - в окне выводится предупреждение (окно имеет имя "Warning");

mfError = 1 - в окне выводится сообщение об ошибке (окно имеет имя "Error");

mfInformation = 2 - окно является информационным (окно имеет имя "Information");

mfConfirmation = 3 - выводимая информация требует подтверждения (окно имеет имя "Confirmation");

mfYesButton = $0100 - окно имеет кнопку "Yes", формирующую команду cmYes;

mfNoButton = $0200 - окно имеет кнопку "No", формирующую команду cmNo;

mfOKButton = $0400 - окно имеет кнопку "OK", формирующую команду cmOK;

mfCancelButton = $0800 - окно имеет кнопку "Cancel", формирующую команду cmCancel;

mfYesNoCancel = $0B00 - окно имеет кнопки "Yes", "No" и "Cancel";

mfOKCancel = $0C00 - окно имеет кнопки "OK" и "Cancel".

В методе GetEvent объекта типа tMyApp (лл. 667-707) при работе со справочной системой (лл. 684-705) в случае неудачной попытки открыть поток справочной системы (л. 688) с помощью функции MessageBox (л. 691) формируется диалоговое окно с сообщением об ошибке. В выводимый текст в качестве параметра передается имя файла справочной системы через параметр Par (лл. 676-678), содержащий адрес имени файла (л. 674 и 690). У функции MessageBox используются флаги mfError (окно с сообщением об ошибке) и mfOkButton (используется кнопка "OK").

Если требуется создать диалоговое окно с такими же свойствами, но другого размера, можно воспользоваться функцией MessageBoxRect, которая отличается от MessageBox тем, что дополнительно указываются размеры создаваемого окна:

```
function MessageBoxRect(var R: tRect; const Msg:
                        string; Params: Pointer; AOptions:Word): Word;
```

где R - размер создаваемого окна;
Msg - размещаемое в окне сообщение;
Params - указатель на параметры сообщения;
AOptions - флаги (см. ниже);
результат - код команды, связанной с нажатой кнопкой диалогового окна.

Примечание. У окна, формируемого функцией MessageBox, имеется та же неприятность, что и у клавиш быстрого управления. Название типа окна и названия кнопок отображаются на английском языке. Можно рекомендовать в исходном тексте модуля MsgBox внести соответствующие изменения на русском языке либо написать собственную функцию MessageBox. Так, например, в текстовом редакторе (см. приложение Ж) приведен пример русифицированной функции MessageBox.

5. СПРАВОЧНАЯ СИСТЕМА TURBO VISION

Turbo Vision имеет средства создания справочной системы программы. С этой целью каждый отображаемый объект, а также каждый элемент меню имеет параметр HelpCtx типа Word, куда для объектов или элементов, по которым имеется справка, записывается число, соответствующее этой справке. Если для объекта или элемента нет справки, этот параметр равен нулю. При обращении к справочной системе она выдает справку по сфокусированному (см. п. 1.6) в данный момент объекту или элементу меню, реализуя так называемую контекстнозависимую справку.

Справочная система позволяет реализовать возможности гипертекста посредством организации перекрестных ссылок, просмотра ранее вызванной справки и т. д.

Второе назначение параметра HelpCtx - вызов на экран соответствующего варианта строки состояния (см. п. 6.1) и сообщения о текущем процессе (см. п. 6.2).

В соответствии с общепринятым стандартом рекомендуется вызов справки осуществлять с помощью клавиши F1, хотя Turbo Vision позволяет использовать и другие клавиши для этой цели.

Если решено создавать справочную систему разрабатываемой программы, начинать ее создание следует с самого начала создания программы и вести параллельно разработке других частей программы (в частности, постоянно накапливать фрагменты справочной системы и константы, определяющие эти фрагменты), т. к. в противном случае многие характерные особенности различных объектов программы, которые неплохо отразить в справочной системе, к концу создания программы могут быть забыты.

5.1. Включение в программу справочной системы

Основные элементы справочной системы (оформление фрагментов справочной системы, окно для вывода информации, цветовая палитра и т. д.) можно взять из демонстрационного файла HelpFile пакета Turbo Vision (можно использовать и обычное окно или диалоговое окно для вывода информации справочной системы, но тогда придется вносить больше изменений в программу - фактически заново писать все те объекты, которые

имеются в файле HelpFile). Для того чтобы использовать их в программе, следует выполнить следующие операции (не считая подключения модуля HelpFile к программе).

Во-первых, следует скорректировать палитру программы, добавив в нее палитру для окна справочной системы. С этой целью следует переопределить метод GetPalette объекта tApplication (лл. 708-714). О методе GetPalette подробнее см. п. 7.1).

Далее необходимо обеспечить обработку запросов к справочной системе. Эту обработку целесообразно осуществить в основном объекте программы (потомок объекта типа tApplication) с тем, чтобы все запросы к справочной системе обрабатывались в одном месте и выполнялись единообразно. Проще всего организовать эту обработку в методе GetEvent, формирующем события (см. п. 1.7). Это связано с особенностями работы модальных объектов программы. Если в программе в какой-либо момент имеется модальный объект (кроме объекта типа tApplication), то вся обработка событий с помощью метода HandleEvent сосредоточена внутри этого модального объекта и не выходит за его границы. Таким образом, если обработка запросов к справочной системе будет осуществлена в методе HandleEvent объекта tApplication, то при наличии модального элемента справочная система будет отключена (вряд ли разумно в каждом модальном объекте формировать обработку запросов к справочной системе). Метод же GetEvent в отличие от метода HandleEvent работает иначе. Метод GetEvent всегда вызывает аналогичный метод своего владельца (см. п. 1.4), независимо от того, модальный ли объект или нет; тот, в свою очередь, вызывает метод GetEvent своего владельца, и т. д. вплоть до объекта tApplication. Таким образом, если в этом методе поместить обработку запросов справочной системы, любой такой запрос будет обработан.

Эта обработка включает в себя открытие потока (см. п. 8.3), связанного с файлом справочной системы, создание окна справочной системы, размещение его в рабочей области, работу с ним и, наконец, завершение работы с ним и удаление окна из рабочей области и динамической памяти.

Инициализация диалогового окна осуществляется конструктором Init:

```
constructor tHelpWindow.Init(HFile: pHelpFile;
                             Context: Word);
```

где HFile - объект, позволяющий работать с файлом справочной системы через поток справочной системы;
Context - параметр контекстной помощи.

Пример окна справочной системы приведен на рис. 5.1.

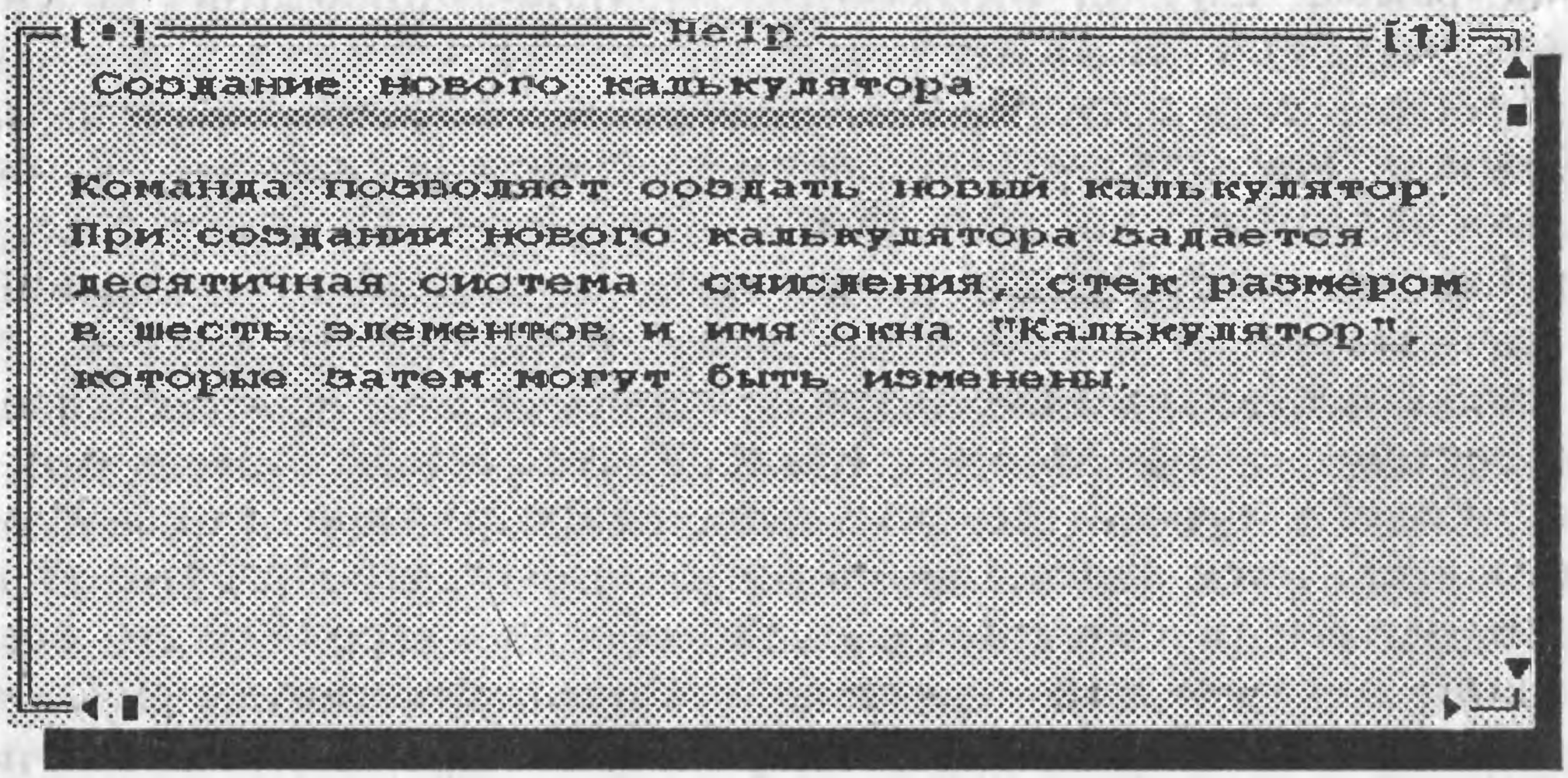


Рис. 5.1. Окно справочной системы

Для инициализации параметра типа tHelpFile следует использовать конструктор Init:

```
constructor tHelpFile.Init(S: pStream);
```

где S - поток (см. п. 8.3), связанный с файлом справочной системы.

Параметр контекстной помощи - уникальное число типа Word. Для каждого такого числа в файле справочной системы (см. п. 5.2) должен быть фрагмент, который выводится в окне справочной системы. Если какому-либо числу в файле нет соответствующего фрагмента, никакая справочная информация по этому числу не выводится. Два значения этого параметра имеют специальное назначение:

hcNoContext = 0 - отсутствие информации,
hcDragging = 1 - объект находится в режиме перемещения и изменения размеров,

и их следует использовать только в этих целях. Не следует также использовать значения параметра больше $FF00, предназначенные для стандартных меню.

Для получения параметра контекстной помощи следует, находясь в основном объекте программы (тип tApplication или его потомок), воспользоваться методом GetHelpCtx объекта tGroup, потомком которого является объект tApplication:

```
function tGroup.GetHelpCtx: Word; virtual;
```

Метод действует следующим образом. Если у основного объекта есть подэлемент, находящийся в выделенном состоянии

(см. п. 1.6), то вызывается метод GetHelpCtx этого подэлемента, и если этот подэлемент является группой, то операция повторяется уже для него. Такая последовательность действий повторяется до тех пор, пока будет существовать очередной выделенный элемент. Это позволяет пройти по цепочке выделенных элементов (см. п. 1.6) до последнего элемента. На каждом этапе результат либо равен значению параметра контекстной помощи, полученному на следующем этапе, если оно не равно hcNoContext, либо значению параметра контекстной помощи текущего элемента в противном случае. Таким образом будет найден последний элемент в цепочке выделенных элементов, у которого параметр контекстной помощи не равен hcNoContext. Чаще всего это будет сфокусированный элемент (см. п. 1.6), т. е. последний элемент этой цепочки. У найденного объекта и берется в качестве возвращаемого параметра параметр контекстной помощи HelpCtx.

Если найденный таким образом объект находится в режиме перемещения или изменения своих размеров (установлен флаг sfDragging параметра State - см. п. 1.3), то вместо значения параметра HelpCtx этого объекта возвращается значение hcDragging (см. выше).

При работе со справочной системой (лл. 684-705) сначала проверяется, не осуществляется ли уже работа со справочной системой, и если нет, то открывается поток (см. п. 8.3), связанный с файлом справочной системы HELPCALC.HLP (л. 686), затем создается объект типа tHelpFile (л. 687) и, если поток открыт нормально, формируется окно справочной системы (л. 696), и при удачном создании окна производится работа с ним (л. 699), после чего окно справочной системы уничтожается.

У большинства отображаемых объектов (меню, кнопки калькулятора, элементы диалогового окна задания параметров калькулятора, окно просмотра стека калькулятора и т. д.) имеются отличные от hcNoContext значения параметра HelpCtx.

5.2. Создание файла справочной системы

В составе Turbo Vision имеется утилита TVCH.EXE, позволяющая создать файл справочной системы, с которым могут работать средства, имеющиеся в модуле HelpFile (см. п. 5.1).

Перед тем как использовать эту утилиту, следует с помощью какого-либо редактора текста (например, используя редактор

пакета Turbo Pascal (Borland Pascal)) создать исходный файл фрагментов справочной системы, являющихся отдельными темами справки (лучше всего, если этот файл будет иметь расширение .TXT, но ни в коем случае не .PAS и не .HLP, т. к. утилита при своей работе формирует именно такие файлы). При его создании следует учитывать следующее. Каждый фрагмент должен начинаться со строки

```
.topic <идентификатор фрагмента>[=<число>]
                    [,<идентификатор фрагмента>[=<число>][...]]
```

В этой строке .topic - признак начала нового фрагмента; идентификатор фрагмента любой идентификатор, присвоенный данному фрагменту, причем первые 20 символов идентификатора должны быть уникальными (лучше не использовать идентификаторы размером больше 20 символов); число (типа Word) - значение параметра контекстной помощи, по которому вызывается этот фрагмент. Если число, соответствующее текущему фрагменту, не задано, оно определяется как число, на единицу большее числа, соответствующего предыдущему идентификатору (как правило, определяющему предыдущий фрагмент. Фрагмент может иметь и несколько идентификаторов, перечисленных через запятую. Это может понадобиться для организации перекрестных ссылок между различными фрагментами.

Если первая строка фрагмента имеет вид

```
.topic File=100, Files, TextFile
```

фрагменту, следующему за этой строкой, будут соответствовать три идентификатора: File, Files и TextFile. При этом первому идентификатору будет соответствовать число 100, второму - число 101, третьему - число 102, а текст этого фрагмента будет вызываться при использовании любого из трех идентификаторов.

Примечание. В идентификаторах можно использовать любые отображаемые символы, кроме знака равенства и запятой, в том числе и пробелы; более того, если после идентификатора перед следующим за ним знаком равенства или запятой стоит пробел, то этот пробел будет входить в идентификатор.

Сам фрагмент справочной системы располагается в одну или несколько строк, причем не следует заботиться о формировании соответствующих строк - справочная система сама распределит информацию по строкам, осуществляя соответствующее разбиение текста по словам. Это связано с тем, что окно справочной системы может менять свои размеры и неизвестно заранее, каким образом следует разбивать текст на строки. При этом, если

в тексте фрагмента строка начинается с пробела, она не будет ни разбиваться на отдельные строки, ни объединяться с другими строками, в противном случае любое разбиение и объединение строк допустимо.

В тексте фрагмента можно выделить отдельные его части для организации перекрестных ссылок. Выделенные участки следует поместить в фигурные скобки { }. Такие ссылки выделяются на экране особым цветом. Ссылку можно выделить либо с помощью мыши, либо с помощью клавиши табуляции Tab (Shift+Tab), после чего следует нажать клавишу Enter. Эти манипуляции приведут к тому, что на экран будет выведен фрагмент, идентификатором которого является выделенный текст. В некоторых случаях неудобно в тексте фрагмента использовать идентификатор другого фрагмента (например, в случае, когда текст написан на русском языке, а идентификаторы - на английском). В этом случае также можно выделить нужную ссылку, но после нее через двоеточие следует указать идентификатор требуемого фрагмента:

```
{<текст>:<идентификатор>}
```

Сам идентификатор в этом случае отображаться на экране не будет, но ссылка будет осуществляться именно через него, а не через выделенный текст. Фрагмент с идентификатором, на который имеется ссылка, может находиться в файле справочной системы не только перед соответствующей ссылкой, но и после нее.

Примечание. При указании идентификатора не следует после двоеточия и после идентификатора ставить дополнительные пробелы, т. к. они будут отнесены к самому идентификатору.

В двух последующих фрагментах идет ссылка друг на друга.

```
.topic TextFile:1000
```

Текстовый файл

Текстовый файл представляет собой файл символов, разделенных на строки (см. {Strings}).

```
.topic Strings
```

Строки

Строкой называется последовательность символов, завершающаяся признаком конца строки (символами "возврат каретки" и "перевод строки"). Файл, содержащий строки символов, называется {текстовым файлом:TextFile}.

В этом примере первая ссылка {Strings} осуществляется непосредственно через идентификатор, во втором случае - {текстовым файлом:TextFile} - вводится дополнительно необходимый идентификатор.

Если очередная строка начинается в первой позиции символом "точка с запятой" (;), то соответствующая строка является строкой комментария и не входит ни в какой фрагмент.

Для программы-калькулятора написан исходный файл фрагментов справочной системы (HELPCALC.TXT), содержимое которого находится в приложении Д.

Созданный исходный файл следует скомпилировать утилитой TVCH.EXE, которая формирует два новых файла, имеющих то же имя, что и исходный файл, но другие расширения. Первый файл имеет расширение .HLP, он и является окончательным файлом фрагментов справочной системы, который будет использовать программа. Именно с этим файлом и следует связать справочную систему программы.

В программе-калькуляторе справочная система связывается с файлом HELPCALC.HLP (л. 686).

Вторым файлом является файл с расширением .PAS. В этом файле, созданном в соответствии с правилами создания файлов модулей Turbo Pascal, размещаются все константы контекстной помощи справочной системы. Эти константы имеют идентификаторы, полученные из идентификаторов фрагментов справочной системы добавлением в начале префикса hc. Этот файл можно использовать в основной программе для того, чтобы записывать идентификаторы соответствующих констант вместо их значений.

В программе-калькуляторе файл HELPCALC.PAS не используется, но формируется утилитой TVCH.EXE (см. приложение Е).

6. СОЗДАНИЕ СТРОК СОСТОЯНИЯ

Строка состояния не является необходимым атрибутом любой программы, многие программы ее не имеют. Однако использование строки состояния помогает пользователю проще работать с программой, т. к. он может здесь получить информацию о текущем состоянии программы и клавишах быстрого управления, которые действуют в данный момент и которые можно выбрать также с помощью мыши непосредственно из строки состояния. Если программа должна иметь строку состояния, то ее создание следует начинать непосредственно после создания меню программы с тем, чтобы впоследствии одновременно рассматривать команды, формируемые меню и строкой состояния (кстати, многие из них могут совпадать). Метод же Hint строки состояния (см. ниже), который обеспечивает краткую информацию о состоянии программы, следует писать параллельно с написанием справочной системы.

Как правило, строка состояния располагается в нижней строке экрана (см. рис. 3.3).

6.1. Строка состояния, отображающая клавиши быстрого управления

Строка состояния может иметь один или несколько вариантов, каждый из которых соответствует определенному диапазону значений параметра контекстной помощи сфокусированного элемента. При создании обычной строки состояния в ней отображаются только клавиши быстрого управления, которые можно использовать в настоящее время.

Каждый элемент строки состояния, содержащий информацию о конкретной клавише (совокупности клавиш) быстрого управления, имеет стандартный тип-запись tStatusItem:

```
tStatusItem = record
   Next: pStatusItem;
   Text: pString;
   KeyCode: Word;
   Command: Word
end;
```

где Next - указатель на следующий элемент строки состояния; если следующего элемента нет, указатель равен nil;
Text - указатель на строку, содержащую выводимый на экран текст для данной клавиши (совокупности клавиш) быстрого управления;
KeyCode - код клавиши (совокупности клавиш) быстрого управления - см. п. 1.7, с помощью которой можно выбрать данный элемент строки состояния;
Command - код команды, которую следует выполнить при выборе данного элемента строки состояния.

Текст для той или иной клавиши быстрого управления может содержать выделенные символы (помещенные между символами ~ ~), которые будут отображаться в строке состояния особым цветом. Обычно это название клавиши (совокупности клавиш) быстрого управления.

Создать элемент строки состояния можно с помощью стандартной функции NewStatusKey:

```
function NewStatusKey(const AText: string;
                      AKeyCode: Word; ACommand: Word; ANext:
                      pStatusItem): pStatusItem;
```

где AText - выводимый на экран текст;
AKeyCode - код клавиши (совокупности клавиш) быстрого управления;
ACommand - код команды, которую следует выполнить при выборе данного элемента строки состояния;
ANext - указатель на следующий элемент строки состояния;
результат - указатель на созданный элемент строки состояния.

Все такие элементы строки состояния представляют собой линейный список, по которому последовательно можно выбрать любой из этих элементов.

Сама строка состояния имеет стандартный тип-запись tStatusDef:

```
tStatusDef = record
   Next: pStatusDef;
   Min, Max: Word;
   Items: pStatusItem;
end;
```

где Next - указатель на следующую строку состояния (следующий вариант строки состояния); если следующей строки нет, указатель равен nil;

Min, Max - минимальная и максимальная границы диапазона параметра контекстной помощи, для которого отображается именно эта строка состояния;
Items - указатель на первый элемент списка элементов строки состояния (см. выше).

Таким образом, и строки состояния в общем случае представляют собой линейный список, по которому можно выбрать любую из них.

Для создания строки состояния можно воспользоваться стандартной функцией NewStatusDef:

```
function NewStatusDef(AMin, AMax: Word;
            AItems: pStatusItem; ANext: pStatusDef): pStatusDef;
```

где AMin, AMax - минимальная и максимальная границы диапазона параметра контекстной помощи, для которого отображается именно эта строка состояния;
AItems - указатель на первый элемент списка элементов строки состояния;
ANext - указатель на следующую строку состояния.

Совокупность всех строк состояния представляет собой стандартный объект типа tStatusLine, который инициализируется методом Init:

```
constructor tStatusLine.Init(var Bounds: tRect;
                             ADefs: pStatusDef);
```

где Bounds - область экрана, выделенная под строку состояния (как правило, нижняя строка);
ADefs - указатель на первый элемент списка строк состояния.

Объект tProgram имеет метод InitStatusLine:

```
procedure tProgram.InitStatusLine; virtual;
```

с помощью которого создается строка состояния и который следует переопределить для программы с тем, чтобы он создавал требуемую строку состояния (совокупность строк состояния). Этот метод размещает указатель на созданную строку состояния (совокупность строк состояния) в стандартной переменной StatusLine пакета Turbo Vision.

Пример структуры строки состояния с указанием методов, с помощью которых создаются отдельные ее части, приведен на рис. 6.1.

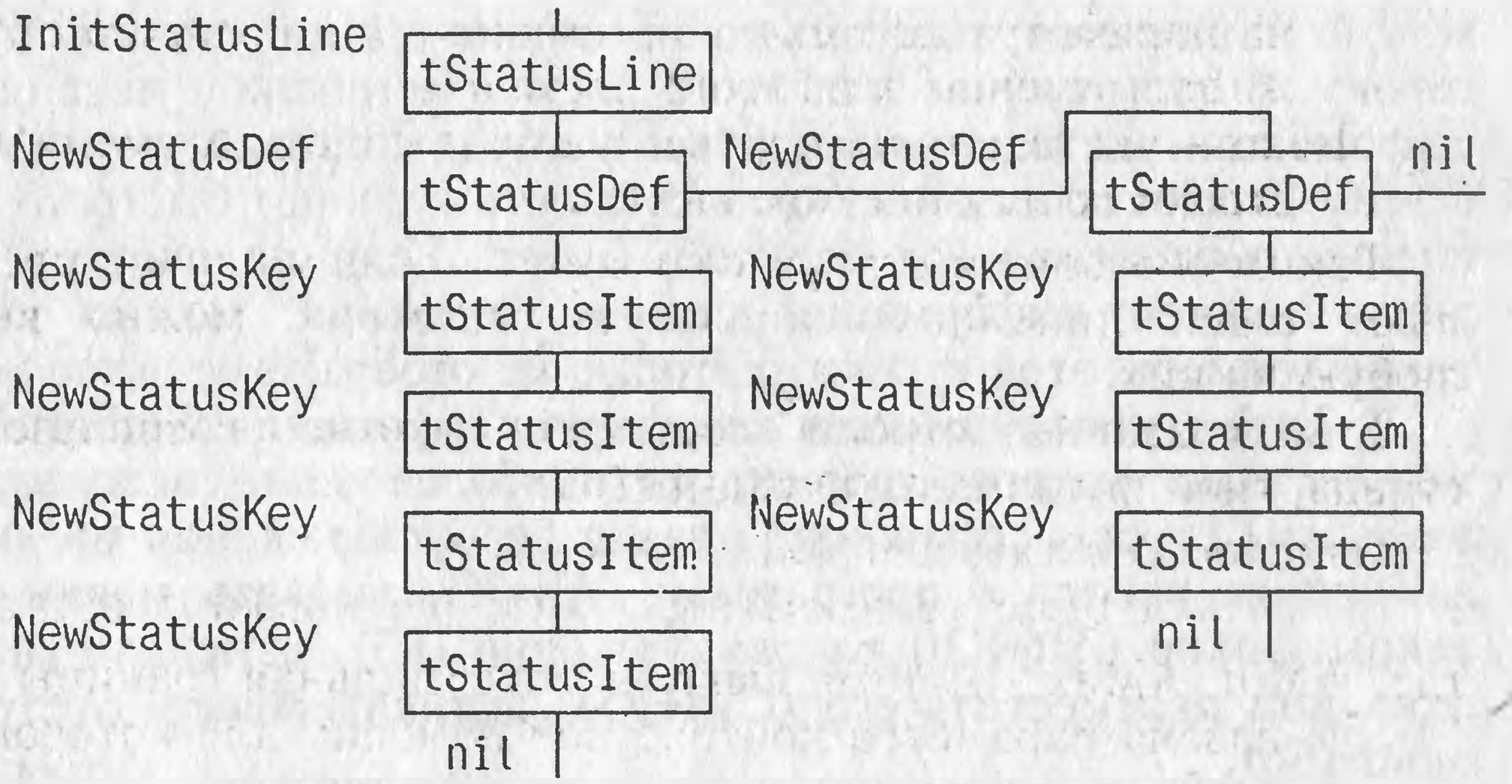


Рис. 6.1. Пример структуры типов элементов строки состояния и подпрограмм, формирующих эти элементы

Не обращая пока внимания на то, что в программе-калькуляторе создается строка состояния типа tHintStatusLine, а не tStatusLine, можно метод InitStatusLine представить так, как это изображено в лл. 752-758. Непосредственно строка состояния создается в л. 757:

```
StatusLine:= New(pHintStatusLine, Init(R,
   NewStatusDef(0, $FFFF,     { Диапазон контекстной помощи }
      NewStatusKey('~Alt-X~ Выход', kbAltX, cmQuit,
                                  { Первый элемент строки }
      NewStatusKey('~F1~ Помощь', kbF1, cmHelp,
                                  { Второй элемент строки }
      NewStatusKey('~F10~ Меню', kbF10, cmMenu,
                                  { Третий элемент строки }
      nil))),                      { Нет больше элементов }
   nil)));                      { Нет больше строк состояния }
```

Так как создается строка состояния, состоящая из одного варианта, диапазон параметра контекстной помощи включает все допустимые беззнаковые целые числа от 0 до $FFFF. Строка состояния состоит из трех элементов обработки клавиш быстрого управления: Alt+X - выход из программы, F1 - обращение к справочной системе, F10 - обращение к меню. Предпоследний элемент nil указывает на завершение элементов строки состояния, последний элемент nil указывает, что других вариантов строки состояния нет.

Для отдельных элементов строки состояния при их создании можно в качестве выводимого на экран текста указать пустую строку. В этом случае для этого элемента никакой поясняющей информации на экран выводиться не будет, однако упоминаемая в этом элементе клавиша (совокупность клавиш) быстрого управления обрабатываться все-таки будет. Если не следует выводить никакой информации строки состояния, можно указать такие размеры этой строки, которые не отображаются на экране.

В Turbo Vision имеется стандартная функция StdStatusKeys модуля App, которая формирует наиболее часто используемые элементы строки состояния, правда без отображения на экране: завершить работу с программой (Alt+X), вызвать меню (F10), закрыть окно (Alt+F3), распахнуть окно (F5), изменить размеры окна или переместить его (Ctrl+F5), активизировать следующее окно (F6).

Если в строке состояния не указывать, что клавиша F1 является клавишей вызова справочной системы (во многих случаях так обычно и происходит), но работу справочной системы все-таки следует обеспечить, то метод, создающий строку состояния (лл. 752-758), следует оформить следующим образом.

```
{ Создание строки состояния }
procedure tMyApp.InitStatusLine;                                    {752}
var R: tRect;                                                       {753}
begin                                                               {754}
   GetExtent(R);                                                    {755}
   R.A.Y:= R.B.Y - 1;                                               {756}
   StatusLine:= New(pHintStatusLine, Init(R,
      NewStatusDef(0, $FFFF,
      NewStatusKey('~Alt-X~ Выход', kbAltX, cmQuit,
         NewStatusKey('', kbF1, cmHelp,
         NewStatusKey('~F10~ Меню', kbF10, cmMenu,
         nil))),
      nil)));                                                       {757}
end;                                                                {758}
```

6.2. Добавление в строку состояния краткой информации о текущих процессах, протекающих в программе

В строке состояния помимо информации о действующих в настоящее время клавишах быстрого управления можно отразить также информацию о состоянии программы в конкретный

момент времени (действие, связанное с активизированным в данный момент элементом меню, назначение сфокусированного элемента, процесс, протекающий в программе, и т. д.). Информация такого вида зависит от значения параметра контекстной помощи активизированного в настоящий момент элемента программы. Вывести такую информацию можно с помощью метода Hint объекта tStatusLine:

```
function tStatusLine.Hint(AHelpCtx: Word): string; virtual;
```

где AHelpCtx - значение параметра контекстной помощи;
результат - строка символов, выводимая на экран.

У стандартного объекта tStatusLine метод Hint не выводит в строку состояния никаких символов, поэтому, если требуется выводить в строку состояния какую-либо информацию, этот метод следует переопределить в объекте - потомке объекта tStatusLine.

Именно по этой причине в программе-калькуляторе используется строка состояния типа tHintStatusLine, а не tStatusLine.

В переопределенном методе следует просто перечислить значения параметра контекстной помощи и указать выводимую для них информацию. Эта информация выводится за символами строки состояния, определяющими клавиши быстрого управления, и отделяется вертикальной линией ( ). Естественно, она должна быть небольшого объема, чтобы уместиться в строке после всех перечисленных клавиш быстрого управления.

В программе-калькуляторе используется метод Hint (лл. 79-127), выводящий на экран соответствующую информацию. Пример работы этого метода изображен на рис. 3.3.

7. ЦВЕТОВАЯ ПАЛИТРА TURBO VISION

Turbo Vision позволяет манипулировать цветом изображения объектов на экране. При этом можно использовать стандартный набор цветов, заданный в пакете и охватывающий расцвечивание всех стандартных объектов, или создавать свою палитру. На первых порах знакомства с Turbo Vision можно порекомендовать использование стандартных цветовых средств, т. к. предлагаемые палитры объектов выбраны удачно и создают благоприятные цветовые ощущения у пользователя программы. Нестандартные палитры используются при создании нестандартных объектов или когда разработчик хочет создать интерфейс программы в стиле, отличном от того, который принят у фирмы Borland.

7.1. Стандартная палитра Turbo Vision

Все стандартные отображаемые объекты Turbo Vision имеют свою цветовую палитру - набор цветов для отдельных элементов этого объекта, а окна и диалоговые окна - по три палитры. Конкретные атрибуты цветов [1] для стандартных объектов Turbo Vision размещены в массивах символов (строках) модуля App для цветного, черно-белого и монохромного адаптеров. Каждый элемент строки (символ) дает атрибуты цветов какого-либо элемента стандартного объекта. Стандартная цветовая палитра Turbo Vision приведена в приложении Б и является палитрой объекта tProgram (а следовательно, и tApplication).

Учитывая структуру отображаемых объектов программы (см. п. 1.6) и то, что все отображаемые объекты программы, за исключением основного объекта типа tApplication, являются подэлементами какой-либо группы, для каждого такого объекта палитра представляется строкой, компонентами которой являются не атрибуты цветов, а номерами элементов палитры группы, в которую включается данный объект. Окна и диалоговые окна имеют по три палитры. При переопределении цветов важно знать, каким атрибутам цветов соответствует тот или иной элемент палитры. Это можно определить проследив цепочку от исходной палитры элемента до основной палитры Turbo Vision.

В качестве примера рассмотрим серую палитру окна. Эта палитра выглядит так, как указано в табл. 7.1.

Таблица 7.1. Серая палитра окна

Вход	*Элемент*	*Параметр*
1	Рамка пассивного окна	24
2	Рамка активного окна	25
3	Пиктограмма	26
4	Линейка скроллинга	27
5	Резерв линейки скроллинга	28
6	Нормальный текст поля скроллинга	29
7	Выделенный текст поля скроллинга	30
8	Резерв окна	31

В этой палитре значения элементов соответствуют номерам элементов основной палитры (см. приложение Б) Turbo Vision. Несмотря на то что окно может быть помещено не только в основной объект типа tApplication, но чаще даже в рабочее поле (тип tDeskTop), все равно берется основная палитра, т. к. у рабочего поля нет своей палитры и оно использует палитру своего владельца.

Если теперь посмотреть палитру объекта-рамки окна (тип tFrame) (см табл. 7.2), то значениями элементов палитры рамки будут входы палитры окна (или диалогового окна, которое может иметь рамку, а элементы, соответствующие рамке, у окна и диалогового окна совпадают).

Таблица 7.2. Палитра рамки окна

Вход	*Элемент*	*Параметр*
1	Рамка пассивного окна	1
2	Заголовок пассивного окна	1
3	Рамка активного окна	2
4	Заголовок активного окна	2
5	Пиктограмма	3

Если теперь нас интересуют атрибуты пиктограммы рамки окна, то их можно определить следующим образом. Пиктограмма в палитре рамки окна имеет параметр 3. Третий вход в серой палитре окна имеет значение 26. Двадцать шестой вход в основной палитре (см. приложение Б) соответствует светло-зеленым символам на сером фоне.

Если в программе достаточно использовать цвета основной палитры по их прямому назначению, то никаких действий по раскрашиванию стандартных объектов и их потомков предпринимать не надо, т. к. цвета выбираются при отображении на экране соответствующего объекта с помощью метода Draw:

```
procedure <отображаемый объект>.Draw; virtual;
```

который, в свою очередь, использует методы GetPalette:

```
function <отображаемый объект>.GetPalette:
                                        pPalette; virtual;
```

определяющий указатель на конкретную палитру объекта, и GetColor:

```
function tView.GetColor(Color: Word): Word;
```

где Color - номер входа в палитру;
результат - атрибуты соответствующего цвета,
определяющий конкретные атрибуты цвета
по его номеру в палитре.

7.2. Изменение стандартной палитры

Если по каким-либо причинам необходимо изменить палитру отображаемых элементов, то здесь имеются по крайней мере три возможности в зависимости от того, какие изменения следует произвести.

Во-первых, если требуется изменить уже существующий цвет у всех используемых его объектов, то следует просто внести соответствующие изменения в основную палитру. Для этого нужно либо внести изменения в исходный текст модуля App (константа CAppColor) и перекомпилировать его, либо внести изменения в метод GetPalette объекта tProgram.

Если в диалоговом окне с серой палитрой следует для пассивного окна сделать рамку темно-серой на светло-сером фоне (вместо черного на светло-сером фоне), то можно написать следующий виртуальный метод объекта tMyApp:

```
function tMyApp.GetPalette: pPalette;
const
   P: array[apColor..apMonochrome] of
                                string[Length(CAppColor)] =
              (CAppColor, CAppBlackWhite, CAppMonochrome);
begin
   P[apColor, 24]:= $87;                { Задание нового цвета }
   GetPalette:= @P[AppPalette];
end;
```

Естественно, следует поместить заголовок метода с директивой virtual в объявлении объекта tMyApp (лл. 70- 78):

```
function GetPalette: pPalette; virtual;
```

Второй вариант - когда нужно изменить цвета у какого-то объекта группы, а в этой группе есть другой объект, обладающий необходимыми цветами. В этом случае можно изменить соответствующим образом константу, определяющую палитру у метода GetPalette.

Предположим, что у селективного списка выбора системы счисления диалогового окна задания параметров калькулятора (лл. 55-58) поле следует сделать таким же, как и у строки ввода, т. е. белые символы на синем фоне. Так как у селективного списка следует произвести изменения цветов, стандартный объект уже использовать нельзя - необходимо создать потомок этого объекта, у которого будет собственный метод GetPalette:

```
{ Нестандартно раскрашенный селективный список }
  pMyRadioButtons = ^tMyRadioButtons;
  tMyRadioButtons = object(tRadioButtons)
    function GetPalette: pPalette; virtual;
  end;

{*****************************************}
{*          Объект tMyRadioButtons       *}
{*****************************************}
function tMyRadioButtons.GetPalette: pPalette;
const CMyCluster = #19#19#19#19;
      P: string[4] = CMyCluster;
begin
  GetPalette:= @P
end;
```

В этом объекте палитра вместо стандартных входов 16, 17 и 18 в палитру диалогового окна использует вход 19, соответствующий нормальной раскраске строки ввода (белые символы на синем фоне).

По сути дела, тот же прием был использован, когда в окне, отображающем содержимое стека (тип tStackWindow), был размещен статический текст. Статический текст не может быть элементом обычного окна, т. к. для него не предусмотрен вход в палитру окна. Однако его вход в палитру диалогового окна (6) совпадает с входом в палитру обычно-

го окна для нормального текста поля скроллинга (черный на светло-сером для серой палитры), и его можно использовать для отображения статического текста в обычном окне, если такое сочетание цветов нас устраивает.

Наконец, третий вариант - когда необходимо изменить цвета конкретного объекта, а требуемого сочетания цветов у его владельца нет. В этом случае следует дополнить основную палитру программы новыми элементами, соответствующими требуемым цветам. Так как основная палитра исходно имеет 127 элементов, то примерно столько же элементов можно к ней добавить в случае необходимости. Естественно, что после этого следует переопределить метод GetPalette основного объекта программы.

Окно справочной системы является разновидностью "голубого" окна. Его особенность в том, что текст, размещенный в поле скроллинга, может иметь три цвета: нормальный текст, выделенный и выбранный (текст, для которого предполагается получить дополнительную справку, - см. п. 5.2). В связи с этим используется специальная цветовая палитра CHelpColor модуля HelpFile, а в методе GetPalette объекта tMyApp (лл. 708-714) эта палитра присоединяется к основной палитре CAppColor.

В тех случаях, когда следует предусмотреть частое изменение цветов отдельных объектов в процессе выполнения программы, можно воспользоваться средствами модуля ColorSel, в частности диалоговым окном tColorDialog, позволяющим задавать атрибуты цветов (цвет символа и фона) всех компонент объектов программы. Этот объект подробно описан в п. 9.3.

8. НЕОТОБРАЖАЕМЫЕ ОБЪЕКТЫ TURBO VISION

Помимо отображаемых объектов в Turbo Vision имеется ряд неотображаемых объектов, предназначенных для выполнения действий, не связанных непосредственно с представлением информации на экране, в частности проверки вводимой информации или для размещения в них той или иной информации. Основными неотображаемыми объектами Turbo Vision являются анализаторы введенной информации, коллекции, потоки и ресурсы. Эти объекты, являющиеся потомками объекта tObject, и будут рассмотрены в данной главе.

8.1. Анализаторы вводимой информации

В Turbo Vision 2.0 имеется возможность контроля размещаемой в строке ввода информации. Для этого служит серия объектов, называемых анализаторами вводимой информации, исходным из которых является абстрактый объект tValidator, а его потомками являются:

- анализатор множества допустимых символов - тип tFilterValidator;
- анализатор границ вводимого числа - тип tRangeValidator;
- анализатор по списку строк - тип tStringLookupValidator;
- анализатор по шаблону - тип tPXPictureValidator.

Кроме этих анализаторов можно создать их потомков (а также потомков абстрактного типа tLookupValidator - анализатора по какому-либо списку, являющегося потомком объекта tValidator и предком объекта tStringLookupValidator).

Связь между анализатором и строкой ввода осуществляется при инициализации строки ввода.

Рассмотрим коротко стандарные анализаторы Turbo Vision.

Анализатор множества допустимых символов (тип tFilterValidator) проверяет, входят ли вводимые символы в множество допустимых символов, и если входят, то они помещаются в буфер строки ввода, в противном случае символы игнорируются. Инициализация такого анализатора осуществляется конструктором Init:

constructor tFilterValidator.Init(AValidChars: tCharSet);

где AValidChars - множество допустимых символов типа tCharSet (tCharSet=set of Char - стандартный тип Turbo Vision).

Если необходимо помещать в строку ввода только цифры, можно оформить создание такого анализатора следующим образом:

```
var MyFilterValidator: pFilterValidator;
                    . . .
    MyFilterValidator:= New(pFilterValidator, Init(['1'..'0']));
```

Если в строку ввода, с которой связан такой анализатор, попытаться ввести последовательность символов

27 марта,

в действительности будет помещена в буфер и отображена на экране лишь информация

27

Анализатор границ вводимого числа (тип tRangeValidator) предназначен для контроля вводимых в символьной форме целых чисел (в общем случае типа Longint). Этот анализатор, являясь потомком объекта tFilterValidator, помимо проверки допустимых символов (допускается вводить только цифры и знаки '+' или '-', причем знаки можно вводить только в первой позиции вводимой информации) проверяет также значение введенного числа и его нахождение в заданном диапазоне.

Анализатор инициализируется конструктором Init:

```
constructor tRangeValidator.Init(AMin, AMax: Longint);
```

где AMin, AMax - минимальная и максимальная допустимые границы вводимого числа.

Значение введенного числа проверяется при попытке перейти от строки ввода числа к любому другому элементу диалогового окна или при попытке закрыть окно с сохранением введенных данных. Если введенное число не попадает в заданный диапазон, выдается сообщение об ошибке в виде информационного окна, при выходе из которого происходит возврат к строке ввода для внесения соответствующих изменений. Окно, естественно, не закрывается. Пример анализатора будет приведен ниже.

Анализатор по списку строк (тип tStringLookupValidator) сравнивает строку введенных символов со списком допустимых строк, находящимся в коллекции строк (см. п. 8.2). Проверка осуществляется в тех же случаях, что и при проверке значения числа анализатором границ введенного числа. Реакция при неп-

равильно введенной строке символов тоже аналогичная. Инициализируется такой анализатор конструктором Init:

```
constructor tStringLookupValidator.Init(
                                AStrings: pStringCollection);
```

где AStrings - коллекция строк, с которыми сравнивается вводимая информация.

Анализатор по шаблону (тип tPXPictureValidator) позволяет создать шаблон, по которому проверяется вводимая информация. Здесь можно проверять количество вводимых символов, вид каждого символа и т. д. Инициализируется такой анализатор конструктором Init:

```
constructor tPXPictureValidator.Init(
                        const APic: string; AutoFill: Boolean);
```

где APic - шаблон, по которому проверяется вводимая информация;
AutoFill - если равен True, вводимая информация автоматически дополняется символами шаблона, не являющимися служебными символами.

Символы, используемые в шаблоне, имеют следующее назначение, указанное в табл. 8.1 (не указанные в таблице символы используются буквально).

Таблица 8.1. Назначение символов шаблона объекта tPXPictureValidator

Символ	*Назначение*
#	Можно поместить только цифру
?	Можно поместить только букву (прописную или строчную)
&	Можно поместить только букву (буква преобразуется в прописную)
@	Можно поместить любой символ
!	Можно поместить любой символ, буквы преобразуются в прописные
;	Следующий символ используется буквально
*	Число повторений
[]	Необязательные символы
{}	Группа
,	Перечисление альтернативных символов

Анализатор должен быть подключен к соответствующей строке ввода с помощью метода SetValidator строки ввода:

```
procedure tInputLine.SetValidator(AValid: pValidator);
```

где AValid - указатель на подключаемый к строке ввода анализатор.

В программе-калькуляторе используется анализатор границ вводимого числа типа tMyRangeValidator, отличающийся от стандартного анализатора типа tRangeValidator методом Error, который выдает русифицированное сообщение в случае неправильного ввода числа. В л. 580 создается этот анализатор с границами для вводимого числа 2 и 15, здесь же этот анализатор подключается к строке ввода размера стека.

8.2. Объект-коллекция

Объект-коллекция (тип tCollection) напоминает в некотором роде тип-массив языка Паскаль, но с некоторыми дополнительными возможностями и предназначен в первую очередь для размещения в нем объектов (точнее, указателей на объекты). Основным отличием коллекции от массива является то, что размеры ее при необходимости могут изменяться (у массива ни его размер, ни тип элементов изменять нельзя). Кроме этого, коллекция может содержать объекты разных типов и элементы, не являющиеся объектами. В последнем случае, правда, возникают некоторые трудности с удалением таких элементов из динамической области памяти, но эта задача в принципе решается.

Создается коллекция с помощью конструктора Init:

```
constructor tCollection.Init(ALimit, ADelta: Integer);
```

где ALimit - первоначальный размер коллекции;
ADelta - величина, на которую изменяется размер коллекции в случае необходимости.

Работа с коллекцией осуществляется следующим образом. Вначале создается коллекция, и она инициализируется конструктором Init. После этого в нее можно помещать компоненты. При этом в коллекции помещается указатель на размещаемый объект. Если при размещении очередного объекта в коллекции нет достаточного места, она автоматически увеличивается на величину ADelta.

Текущий размер коллекции находится в параметре Limit типа Integer. Фактическое количество размещенных в коллекции компонент находится в параметре Count (также типа Integer), который увеличивается на единицу при добавлении новой компоненты. Естественно, что всегда выполняется условие

```
Count <= Limit.
```

Каждая компонента коллекции имеет индекс, причем нумерация начинается с нуля. Пример структуры коллекции для случая Limit=5, Delta=2, Count=4 приведен на рис. 8.1.

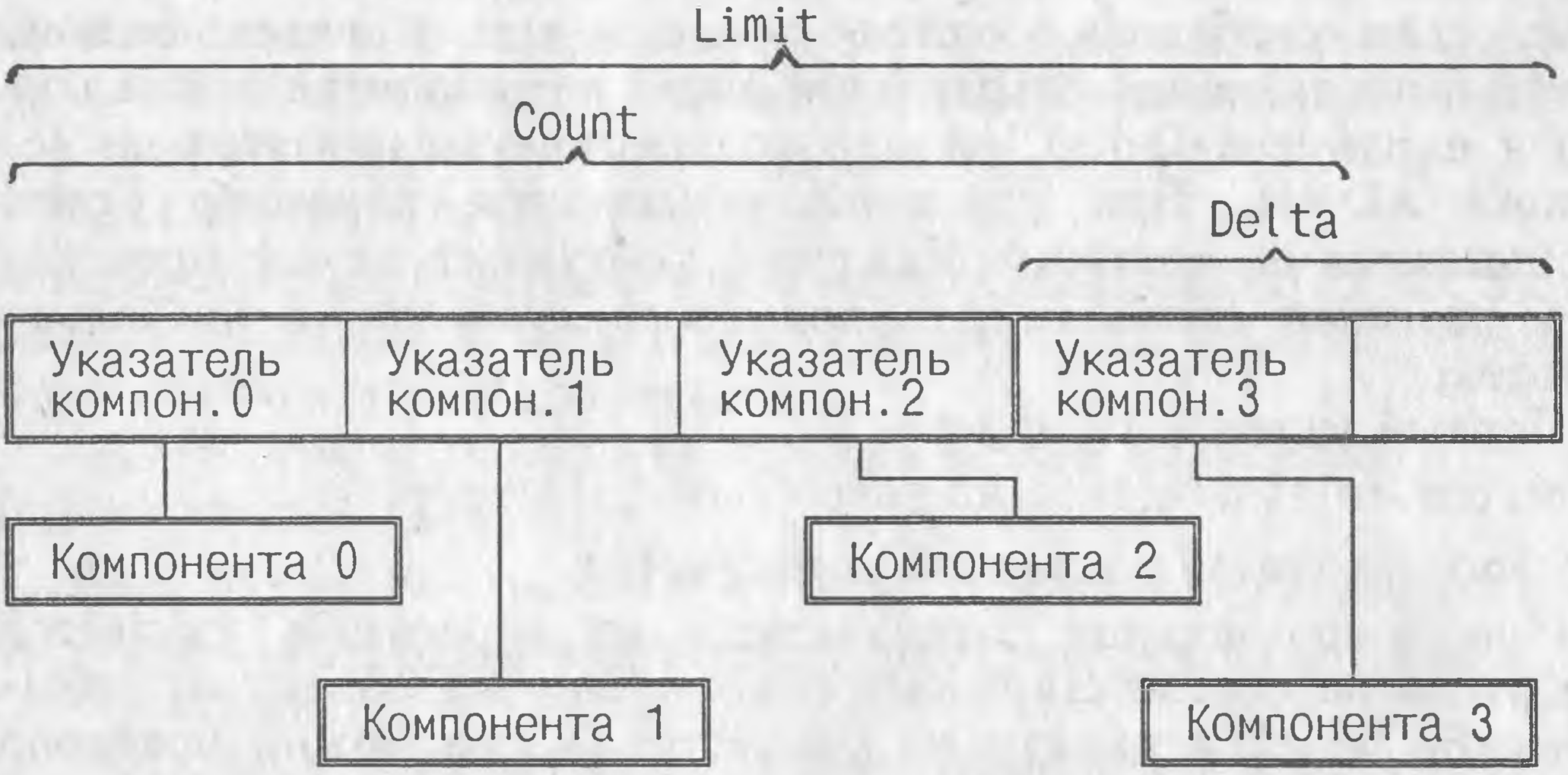


Рис. 8.1. Пример структуры коллекции

Компоненту в коллекцию можно поместить либо в конце списка, либо после компоненты с указанным значением индекса. В первом случае используется метод Insert:

```
procedure tCollection.Insert(Item: Pointer); virtual;
```

где Item - указатель на размещаемый в коллекции объект,

во втором случае используется метод AtInsert:

```
procedure tCollection.AtInsert(Index:Integer;Item: Pointer);
```

где Index - номер компоненты, который получает размещаемый объект;

Item - указатель на размещаемый в коллекции объект.

Следует отметить следующие особенности этих методов.

У потомков объекта tCollection, таких, например, как отсортированные коллекции (см. ниже), метод Insert может размещать новый объект и не в конец списка (например, размещать строки по алфавиту). В этом случае после размещения нового объекта возможно изменение индексов у каких-то ранее расположенных компонент.

Метод AtInsert работает успешно только в том случае, если правильно указан номер компоненты Index: он должен быть в пределах от нуля до Limit (см. выше). В противном случае новая компонента в коллекции не размещается, а вместо этого вызывается метод Error, сигнализирующий о неправильной работе с коллекцией.

Ни метод Insert, ни метод AtInsert копии размещаемого в коллекции объекта не создают.

Из коллекции можно также и удалять компоненты. При этом, если свободная область коллекции становится больше ADelta (это значение сохраняется после инициализации коллекции в параметре Delta), размер коллекции уменьшается на величину ADelta. При удалении компоненты параметр Count уменьшается на единицу. Удаление компонент может происходить разными способами, и для этого существует несколько методов.

Первый метод - AtDelete:

```
procedure tCollection.AtDelete(Index: Integer);
```

где Index - номер удаляемой компоненты.

При использовании этого метода из коллекции удаляется указатель на соответствующий объект, сам же объект из динамической области памяти не удаляется, и его можно повторно разместить в ту же или другую коллекцию или использовать в других целях. При необходимости происходит корректировка индексов оставшихся компонент. Если была попытка удаления несуществующего элемента, изменения в коллекции не происходят и вызывается метод Error.

Второй метод для удаления компоненты - Delete:

```
procedure tCollection.Delete(Item: Pointer);
```

где Item - указатель на удаляемый объект.

Метод работает аналогично методу AtDelete, только здесь вместо номера компоненты задается указатель на удаляемый объект. При отсутствии в коллекции компоненты с таким указателем вызывается метод Error. Как и в случае метода AtDelete, этот метод лишь удаляет указатель на объект из коллекции и не удаляет сам объект из динамической области памяти.

Третий метод удаления компоненты - метод AtFree:

```
procedure tCollection.AtFree(Index: Integer);
```

где Index - номер удаляемой компоненты.

Метод работает аналогично методу AtDelete, но кроме всего прочего он удаляет сам объект из динамической области памяти.

Четвертый метод - Free:

```
procedure tCollection.Free(Item: Pointer);
```

где Item - указатель на удаляемый объект.

Этот метод работает аналогично методу AtFree, только здесь вместо номера компоненты задается указатель на удаляемый объект.

Следующий метод - DeleteAll:

```
procedure tCollection.DeleteAll;
```

Этот метод удаляет все указатели на объекты, являющиеся компонентами коллекции, но не удаляет сами объекты из динамической области памяти. Параметр Count принимает значение 0.

Наконец, метод FreeAll:

```
procedure tCollection.FreeAll;
```

выполняет те же действия, что и метод DeleteAll, но кроме этого удаляет из динамической области памяти все объекты, являющиеся компонентами коллекции.

Следует отметить еще три важных метода коллекции.

Первый метод - FirstThat:

```
function tCollection.FirstThat(Test: Pointer): Pointer;
```

где Test - указатель на функцию, проверяющую некоторое условие;

результат - указатель на первую компоненту коллекции, для которой верно условие, проверяемое функцией, указатель на которую равен Test либо nil, если ни для какой компоненты коллекции условие не выполняется,

определяет указатель на первую компоненту коллекции, для которой выполняется условие, проверяемое функцией, указатель на которую равен Test.

Функция с указателем Test не должна быть методом и должна компилироваться с директивой far [1]. Ее заголовок выглядит следующим образом:

```
function <имя функции>(Item: Pointer): Boolean; far;
```

В этом заголовке Item - указатель на компоненту коллекции; результат функции равен True, если проверяемое функцией условие выполняется, и False - в противном случае.

Если в коллекции вещественных чисел повышенной точности (тип Double) следует определить первый отрицательный элемент, то эту операцию можно выполнить следующим образом. Функцию проверки условия запишем так:

```
function Negative(Item: Pointer): Boolean; far;
begin
   if pDouble(Item)^ < 0.0 then                    {см. л. 10 программы}
      Negative:= True
   else
      Negative:= False;
end;
```

Использование метода FirstThat для получения указателя на первую отрицательную компоненту может выглядеть так:

```
P:= FirstThat(@Negative);
```

В этом случае переменная P будет содержать указатель на первую отрицательную компоненту коллекции или примет значение nil, если отрицательных чисел в коллекции нет.

Второй метод - LastThat:

```
function tCollection.LastThat(Test: Pointer): Pointer;
```

где Test - указатель на функцию, проверяющую условие;
результат - указатель на последнюю компоненту коллекции, для которой верно условие, проверяемое функцией, указатель на которую равен Test, либо nil, если ни для какой компоненты коллекции условие не выполняется,

определяет указатель на последнюю компоненту коллекции, для которой выполняется условие, проверяемое функцией, указатель на которую равен Test. Функция с указателем Test аналогична такой же функции метода FirstThat.

Если в коллекции вещественных чисел повышенной точности (тип Double) следует определить последний отрицательный элемент, функция, проверяющая знак компоненты коллекции, такая же, что и в предыдущем примере, а определение указателя на последнюю искомую компоненту можно записать так:

```
P:= LastThat(@Negative);
```

Третий метод - ForEach:

```
procedure tCollection.ForEach(Action: Pointer);
```

где Action - указатель на процедуру, которая выполняет требуемые действия с каждой компонентой,

выполняет действия, определяемые процедурой, указатель на которую равен Action, для каждой компоненты коллекции.

Процедура с указателем Action не должна быть методом, т. е. не должна принадлежать ни одному из объектов, и должна компилироваться с директивой far [1]. Ее заголовок выглядит следующим образом:

```
procedure <имя процедуры>(Item: Pointer); far;
```

В этом заголовке Item - указатель на компоненту коллекции.

Если для коллекции вещественных чисел повышенной точности (тип Double) следует у всех компонент поменять знак на противоположный, это можно выполнить следующим образом. Процедура, меняющая знак у компоненты коллекции, может быть записана так:

```
procedure SignChange(Item: Pointer); far;
begin
   pDouble(Item)^:= -pDouble(Item)^;
end;
```

Выполнить изменение знака у всех компонент коллекции можно с помощью оператора

```
ForEach(@SignChange);
```

Разновидностью коллекции является отсортированная коллекция (tSortedCollection). В этой коллекции порядок следования компонент определяется некоторым условием, задаваемым методом Compare:

```
function tSortedCollection.Compare(
                        Key1, Key2: Pointer): Integer; virtual;
```

где Key1, Key2 - указатели на ключи двух сравниваемых компонент, по которым происходит сравнение;
результат - равен -1, если ключ первой компоненты меньше ключа второй компоненты; равен 0, если ключи двух компонент равны; равен 1, если ключ первой компоненты больше ключа второй компоненты.

В качестве ключа компоненты может выступать любой параметр компоненты (или совокупность параметров, и даже сам параметр). Указатель на этот ключ определяется методом KeyOf:

```
function tSortedCollection.KeyOf(
                              Item: Pointer): Pointer; virtual;
```

где Item - указатель на компоненту, для которой определяется ключ;
результат - указатель на найденный ключ компоненты.

Объект tSortedCollection является абстрактным объектом, у которого метод KeyOf просто возвращает указатель на компоненту коллекции Item, а метод Compare не выполняет никаких действий (точнее, выдает сообщение об ошибке и прекращает выполнение программы). Метод Compare, а в ряде случаев и KeyOf, должен быть переопределен для конкретных отсортированных коллекций пользователя.

Примером реальной отсортированной коллекции является коллекция строк (tStringCollection), компонентами которой являются строки символов. Для них в качестве ключей выступают сами строки (метод KeyOf объекта tSortedCollection не переопределяется), а сравнение строк осуществляется на основе правил Turbo Pascal (Borland Pascal).

С-1034

Коллекция может содержать объекты не только одного типа. Основные проблемы здесь - в распознавании типа компонент и правильном освобождении динамической памяти при удалении компонент коллекции. Первая проблема решается самим программистом, когда он знает, в какой последовательности и какого типа объекты размещаются в коллекции. Если компонентами коллекции являются объекты, имеющие таблицы виртуальных методов, можно также воспользоваться стандартной функцией TypeOf пакета Turbo Pascal (Borland Pascal):

```
function TypeOf(Param: ObjType): Pointer;
```

где Param - либо конкретный объект, либо тип объекта;
результат - указатель на таблицу виртуальных методов объекта данного типа.

Сравнив тип конкретного объекта со всеми возможными типами объектов данной коллекции, можно определить тип этого объекта.

Вторая проблема решается автоматически, т. к. при удалении объекта вызывается именно его виртуальный метод Done, который "знает", какую память занимает удаляемый объект.

Если в коллекции MyCollection имеются компоненты трех типов: Type1, Type2 и Type3, то можно вывести сообщения о типах компонент коллекции:

```
for i:= 0 to MyCollection.Count - 1 do
   if TypeOf((pObject(MyCollection.At(i)))^) =
                                          TypeOf(Type1) then
      Writeln(Lst, 'Компонента N ', i, ' типа Type1')
   else if TypeOf((pObject(MyCollection.At(i)))^) =
                                          TypeOf(Type2) then
      Writeln(Lst, 'Компонента N ', i, ' типа Type2')
   else if TypeOf((pObject(MyCollection.At(i)))^) =
                                          TypeOf(Type3) then
      Writeln(Lst, 'Компонента N ', i, ' типа Type3')
   else
      Writeln(Lst,'Компонента N',i,'- тип неизвестен');
```

Коллекция может содержать не только объекты, но и элементы других типов. В частности, имеется стандартная коллекция строк (тип tStringCollection). Как уже говорилось, у таких коллекций следует предусмотреть меры по правильному освобождению динамической памяти при удалении компонент. Так, в коллекции строк переопределен метод FreeItem, который удаляет компоненту из коллекции и из динамической памяти.

В программе-калькуляторе в качестве стека используется коллекция вещественных чисел повышенной точности (тип Double) - лл. 16-19. У этой коллекции переопределен метод FreeItem (лл. 128-132), позволяющий освобождать область памяти, занимаемую вещественным числом повышенной точности.

8.3. Потоки Turbo Vision

Поток (тип tStream) и его потомки: поток ДОС (тип tDosStream), буферированный поток (тип tBufStream) и т. д. - предназначены для хранения в них объектов (а также и данных других типов, однако в дальнейшем мы рассмотрим только потоки объектов). В этом смысле потоки напоминают обычные файлы, но в отличие от них они могут существовать не только на внешних устройствах, но и в оперативной памяти. Другим важным отличием потока от файла является то, что он может хранить данные разных типов. В поток можно записывать те или иные данные, а также читать из него эти данные.

Работа с потоком напоминает работу с файлом последовательного доступа. При создании потока в нем не находится никакой информации. Записываемая в поток информация последовательно приписывается к его концу, так что условный указатель на текущий элемент файла в этом случае указывает на его конец. Этот указатель можно методом Seek потока установить в любую позицию потока, как это делается у файла прямого доступа, однако у потока затруднительно определить начало очередной записанной компоненты, и поэтому целесообразно использовать этот метод при настройке на начало потока для последовательного чтения из него компонент. После чтения очередной компоненты из потока указатель автоматически настраивается на начало следующей компоненты (если это чтение выполнено правильно). Когда требуется все-таки использовать поток как поток прямого доступа с произвольной настройкой на любую его компоненту, лучше вместо обычного потока использовать объект-ресурс (см. п. 8.4), где настройка на компоненту осуществляется по ее имени.

Поток типа tStream является абстрактным потоком, потому что он не привязан к конкретному носителю информации и реальные потоки такого типа не создаются. На практике чаще всего используются потоки ДОС (тип tDosStream) и буферированные поток (тип tBufStream). Оба эти типа позволяют хранить информацию во внешних файлах, причем у буферированного

потока информация передается через промежуточный буфер, выделенный для этого потока, что в ряде случаев повышает быстродействие работы с потоком.

Поток ДОС инициализируется конструктором Init:

```
constructor tDosStream.Init(FileName: FNameStr; Mode: Word);
```

где FileName - имя файла, с которым связан поток и написанное в соответствии с правилами MS-DOS (размер имени - до 79 символов);
Mode - категория файла (см. ниже).

Буферированный поток инициализируется конструктором Init:

```
constructor tBufStream.Init(FileName: FNameStr;
                            Mode, Size: Word);
```

где FileName - имя файла, с которым связан поток и написанное в соответствии с правилами MS-DOS (размер имени - до 79 символов);
Mode - категория файла (см. ниже);
Size - размер буфера (обычно в пределах от 512 до 2048 байт).

Файл, с которым связан поток, может быть разных категорий, определяемых параметром Mode. Для задания той или иной категории используются специальные константы:

stCreate	= $3C00	-	вновь создаваемый файл;
stOpenRead	= $3D00	-	имеющийся файл, открываемый только для чтения;
stOpenWrite	= $3D01	-	имеющийся файл, открываемый только для записи;
stOpen	= $3D02	-	имеющийся файл, открываемый и для чтения, и для записи.

У любого потока имеется два параметра - Status и ErrorInfo, определяющих текущее состояние потока. Параметр Status может принимать следующие значения:

stOk	= 0	-	нормальная работа с потоком;
stError	= -1	-	ошибка при обращении к потоку;
stInitError	= -2	-	нельзя инициализировать поток;
stReadError	= -3	-	попытка чтения за пределами потока;
stWriteError	= -4	-	нельзя добавить элемент в поток;
stGetError	= -5	-	чтение объекта незарегистрированного типа (см. ниже);
stPutError	= -6	-	размещение в потоке объекта незарегистрированного типа (см. ниже).

Параметр ErrorInfo несет дополнительную информацию об ошибке, если работа с потоком неправильна. Так, при значении параметра Status равном stError, stInitError, stReadError или stWriteError параметр ErrorInfo содержит код ошибки MS-DOS. Если была попытка чтения объекта незарегистрированного типа (значение stGetError), параметр ErrorInfo содержит индивидуальный номер незарегистрированного объекта (см. ниже). Наконец, если была попытка записи объекта незарегистрированного типа (значение stPutError), параметр ErrorInfo содержит смещение адреса таблицы виртуальных методов незарегистрированного объекта.

Если параметр Status принимает значение отличное от stOk, т. е. произошла ошибка при работе с потоком, дальнейшие операции с этим потоком прекращаются до тех пор, пока не будет выполнен метод потока Reset:

```
procedure tStrem.Reset;
```

Перед тем как поместить объект в поток или взять из потока, его следует соответствующим образом подготовить.

Во-первых, следует создать методы, с помощью которых будет записываться информация в поток и читаться из него. У стандартных отображаемых объектов Turbo Vision такие методы уже имеются. Первый метод - Store:

```
procedure <отображаемый объект>.Store (var S: tStream);
```

где S - поток, в который помещается объект.

Метод Store размещает объект в потоке. Вторым является конструктор Load:

```
constructor <отображаемый объект>.Load (var S: tStream);
```

где S - поток, из которого берется объект.

Этот конструктор загружает объект из потока в память и используется вместо конструктора Init при создании нового объекта, если его параметры находятся в потоке.

Если стандартный объект является группой, то его методы Store и Load не только загружают и выгружают сам объект, но и его подэлементы. При этом автоматически устанавливаются все необходимые связи.

Для нестандартных объектов в большинстве случаев можно воспользоваться методами Store и Load их предков. Однако, если у такого объекта имеется нестандартный параметр, который необходимо сохранить, следует переопределить эти методы. При этом в новом методе следует сначала вызвать наследуемый метод, а затем записать или прочитать дополнительную информацию. Для записи и чтения этой дополнительной информации

можно использовать различные стандартные методы Turbo Vision, наиболее часто используемые из них следующие.

Для того чтобы записать в поток тот или иной параметр объекта (группу параметров, следующих друг за другом), можно воспользоваться методом Write соответствующего потока (этот метод не следует путать с процедурой Write языка Паскаль):

```
procedure <тип потока>.Write(var Buf; Count: Word); virtual;
```

где <тип потока> - идентификатор типа потока (например, tDosStream, tBufStream и т. д.);
Buf - буфер с переписываемой информацией;
в качестве этого параметра используется либо сам переписывамый параметр, если он один, либо первый параметр, если переписывается группа параметров;
Count - размер переписываемой информации
в байтах.

Метод Write является основным методом, с помощью которого записывается информация в поток. На нем основаны все другие методы, размещающие информацию в потоке, в том числе и методы Store стандартных объектов.

Если имеется диалоговое окно следующего типа:

```
type
   pMyDialog ^tMyDialog;
   tMyDialog = object(tDialog)
      Param1: array[1..5] of Integer;
      Param2: Boolean;
   end;
```

и следует наряду со стандартными параметрами диалогового окна сохранить в потоке параметры Param1 и Param2, метод Store можно написать следующим образом:

```
procedure tMyDialog.Store(var S: tStream);
begin
   inherited Store(S);                    { Вызов наследуемого метода }
   S.Write(Param1, SizeOf(Param1) + SizeOf(Param2));
                                          { Запись дополнительных параметров }
end;
```

Если записываемым параметром является указатель на строку, то вместо метода Write для сохранения строки в потоке следует использовать метод WriteStr объекта tStream:

```
procedure tStream.WriteStr(P: pString);
```

где P - указатель на переписываемую строку.

Если в диалоговое окно из предыдущего примера добавить третий параметр, являющийся указателем на строку:

```
type
   pMyDialog ^tMyDialog;
   tMyDialog = object(tDialog)
      Param1: array[1..5] of Integer;
      Param2: Boolean;
      Param3: pString;
   end;
```

и следует сохранить в потоке строку, на которую указывает параметр Param3, метод Store будет выглядеть так:

```
procedure tMyDialog.Store(var S: tStream);
begin
   inherited Store(S);                    { Вызов наследуемого метода }
   S.Write(Param1, SizeOf(Param1) + SizeOf(Param2);
                                  { Запись дополнительных параметров }
   S.WriteStr(Param3)                                { Запись строки }
end;
```

Отметим еще один метод записи информации в поток. Это метод PutSubViewPtr объекта tGroup:

```
procedure tGroup.PutSubViewPtr(var S: tStream; P: pView);
```

где S - поток, в который помещается объект P;
P - указатель на объект, размещаемый в потоке.

Этот метод размещает в потоке указатель на объект P, являющийся подэлементом группы. Проблема здесь заключается в том, что после чтения группы из потока и размещения ее в оперативной памяти значение этого указателя в общем случае не будет тем, каким оно было при записи группы в поток. Метод PutSubViewPtr подготавливает дальнейшую возможность задания правильного значения указателя на объект после чтения группы из потока.

Если в предыдущем примере в диалоговом окне используется указатель на какой-то элемент окна (например, указатель PMyLabel на метку):

```
type
   pMyDialog ^tMyDialog;
   tMyDialog = object(tDialog)
      Param1: array[1..5] of Integer;
      Param2: Boolean;
      Param3: pString;
      PMyLabel: pLabel;
   end;
```

и следует сохранить в потоке этот указатель с тем, чтобы он в дальнейшем был автоматически восстановлен, метод Store будет выглядеть так:

```
procedure tMyDialog.Store(var S: tStream);
begin
   inherited Store(S);                    { Вызов наследуемого метода }
   S.Write(Param1, SizeOf(Param1) + SizeOf(Param2));
                                  { Запись дополнительных параметров }
   S.WriteStr(Param3)                                  { Запись строки }
   PutSubViewPtr(S, PMyLabel);
                                        { Запись указателя на подэлемент }
end;
```

Для всех перечисленных методов имеются аналоги, осуществляющие чтение соответствующей информации из потока. Основным методом здесь является метод Read соответствующего потока (этот метод не следует путать с процедурой Read языка Паскаль), и позволяющий читать из потока значение параметра объекта (группы параметров):

```
procedure <тип потока>.Read(var Buf; Count: Word); virtual;
```

где <тип потока> - идентификатор типа потока (например, tDosStream, tBufStream и т. д.);
Buf - буфер, в который помещается информация; в качестве этого параметра используется либо сам переписываемый параметр, если он один, либо первый параметр, если переписывается группа параметров;
Count - размер переписываемой информации в байтах.

Для чтения сохраненной в потоке с помощью метода WriteStr строки используется метод ReadStr объекта tStream:

```
function tStream.ReadStr: pString;
```

где результат - указатель на созданную в памяти строку информации.

Этот метод создает в оперативной памяти строку соответствующего размера, заполняет его информацией и возвращает указатель на эту строку.

Наконец, если следует восстановить указатель на подэлемент группы, записанный в поток методом PutSubViewPtr, следует воспользоваться методом GetSubViewPtr:

```
procedure tGroup.GetSubViewPtr(var S: tString; var P);
```

где S - поток, из которого читается информация;
P - переменная типа-указателя на объект соответствующего типа.

Этот метод определяет фактический новый адрес подэлемента и записывает именно его, а не тот адрес, который был у подэлемента в момент его записи в поток.

Следует иметь в виду, что чтение информации из потока должно происходить в той же последовательности, в какой она записывалась в поток.

Для чтения из потока параметров диалогового окна предыдущего примера можно записать следующий конструктор Load:

```
constructor tMyDialog.Load(var S: tStream);
begin
   inherited Load(S);                    { Вызов наследуемого метода }
   S.Read(Param1, SizeOf(Param1) + SizeOf(Param2));
                                 { Чтение дополнительных параметров }
   Param3:= S.ReadStr;                                 { Чтение строки }
   GetSubViewPtr(S, PMyLabel); {Чтение указателя на подэлемент}
end;
```

Вторая операция, связанная с подготовкой объекта к записи в поток, - его регистрация. Так как в потоке могут быть объекты разных типов, возникает проблема в определении типов этих объектов при необходимости работы с ними. Для решения этой проблемы используется система регистрации, заключающаяся в том, что каждому типу объекта, который предполагается записать в поток, присваивается индивидуальный регистрационный номер. По этому номеру в дальнейшем будут определяться размер записанной или записываемой информации, соответствующая таблица виртуальных методов, а также адреса методов, с помощью которых данный объект загружается в поток и читается из него.

Для регистрации объекта следует сначала создать регистрационную запись, представляющую собой запись типа tStreamRec:

```
type
   tStreamRec = record
      ObjType: Word;
      VmtLink: Word;
      Load: Pointer;
      Store: Pointer;
      Next: Word;
   end;
```

где ObjType - индивидуальный регистрационный номер объекта;
VmtLink - смещение адреса таблицы виртуальных методов объекта;
Load - адрес конструктора Load объекта (см. выше);
Store - адрес метода Store объекта (см. выше);
Next - смещение адреса следующей регистрационной записи.

В качестве регистрационной записи, как правило, используется типизированная константа, у которой заполнены первые четыре поля. В Turbo Vision принято следующее правило назначения имен этим константам, которого следует придерживаться. Как уже можно было заметить, имена типов объектов Turbo Vision начинаются с префикса t (имя типа-указателя на объект имеет префикс p). Имя регистрационной записи получается из имени типа объекта заменой префикса t на префикс R. Так, для диалогового окна типа tDialog регистрационная запись имеет имя RDialog и т. д. У всех стандартных объектов Turbo Vision, которые могут быть помещены в поток, уже имеются такие регистрационные записи, имена которых сконструированы именно таким образом, и заботиться в этом плане о стандартных элементах не нужно. За стандартными объектами Turbo Vision зарезервированы индивидуальные регистрационные номера с 1 до 999, так что программист может назначать своим объектам номера с 1000 до 65535. Это число и следует записать в первом поле регистрационной записи ObjType. Во второе поле регистрационной записи следует поместить смещение адреса таблицы виртуальных методов объекта. Для этого можно воспользоваться уже упоминавшейся ранее функцией TypeOf (см. п. 8.1), которая в качестве результата возвращает указатель на таблицу виртуальных методов объекта. В этом случае во второе поле регистрационной записи VmtLink записывается значение Ofs(TypeOf(tMyObject)^), где tMyObject - имя типа объекта, для которого формируется регистрационная запись. В третье поле записывается адрес конструктора Load объекта - @tMyObject.Load, а в четвертое поле - адрес метода Store - @tMyObject.Store. Пятое поле заполняется автоматически при регистрации объекта.

Для объекта типа tMyObject регистрационная запись может иметь, например, следующий вид:

```
type
  RMyObject: tStreamRec = (
    ObjType: 1500;
    VmtLink: Ofs(TypeOf(tMyObject)^);
```

```
    Load: @tMyObject.Load;
    Store: @tMyObject.Store;
  );
```

После создания регистрационной записи объекта его следует зарегистрировать с помощью стандартной процедуры RegisterType пакета Turbo Vision:

```
procedure RegisterType(var S: tStreamRec);
```

где S - регистрационная запись объекта.

Зарегистрировать следует все объекты, которые предполагается размещать в потоке или брать из него, в том числе и стандартные объекты. Более того, если помещаемый в поток объект является группой, следует зарегистрировать и все его подэлементы. Для стандартных объектов имеются специальные процедуры регистрации таких сложных объектов, включающие регистрацию как самого объекта, так и его компонент. Так, например, для диалогового окна имеется процедура RegisterDialogs, которая регистрирует само диалоговое окно (стандартное) и его возможные компоненты: строку ввода, кнопку, метку, протокол и т. д. Перечень таких стандартных процедур с указанием, какие объекты они регистрируют, приведен в приложении В.

На этом подготовка объекта к размещению в потоке или чтению из потока завершается. Далее можно записывать зарегистрированные объекты в поток или читать их из него.

Запись объекта в поток осуществляется с помощью метода Put объекта tStream:

```
procedure tStream.Put(P: pObject);
```

где P - указатель на размещаемый в потоке объект.

Метод Put действует следующим образом. Вначале он пробегает по списку регистрационных записей и определяет по адресу таблицы виртуальных методов размещаемого объекта его индивидуальный регистрационный номер. Этот номер записывается в поток. После этого метод Put вызывает метод Store размещаемого объекта, который и помещает в поток сам объект.

Чтение объекта из потока осуществляется с помощью метода Get объекта tStream:

```
function tStream.Get: pObject;
```

где результат - указатель на созданный объект.

Метод Get читает сначала из потока индивидуальный регистрационный номер объекта и, просматривая список регистрационных записей, находит запись с этим номером. Из этой записи

определяется конструктор Load, с помощью которого объект читается из потока и размещается в оперативной памяти.

В программе-калькуляторе фрагменты справочной системы расположены в потоке ДОС (тип tDosStream), связанном с файлом 'HELPCALC.HLP' (лл. 674 и 686). Для работы с этим потоком зарегистрированы необходимые объекты модуля HelpFile (л. 664), открыт этот поток (л. 686). Все остальные действия с потоком (использование конструктора Load, чтение информации из потока с помощью метода Get) осуществляются методами модуля HelpFile.

После работы с потоком его следует закрыть процедурой Dispose:

```
Dispose(MyStream, Done);
```

где MyStream - поток, с которым завершается работа;
Done - деструктор потока.

Закрыть поток следует с тем, чтобы не потерять внесенные в него в процессе работы изменения.

8.4. Ресурсы

У рассмотренных выше потоков как хранилищ информации об объектах есть один недостаток - непросто осуществить произвольный доступ к отдельным компонентам потока. В связи с этим в Turbo Vision имеется специальный объект - файл ресурсов (tResourceFile), связанный с потоком, в котором обеспечен произвольный доступ к любой компоненте по ее имени (идентификатору), представляющем строку любых символов (в отличие от языка Паскаль прописные и строчные буквы в этих идентификаторах различаются). Этот поток дополнен потомком коллекции строк (см. п. 8.1), содержащим адреса компонент (ресурсов) в потоке, размеры компонент и их имена. Кроме этого, в файле ресурсов имеется механизм добавления, удаления и поиска информации в связанном с ним потоке.

Инициализируется файл ресурсов конструктором Init:

```
constructor tResourceFile.Init(AStream: pStream);
```

где AStream - поток, который связывается с файлом ресурсов.

Практически прежде чем инициализировать файл ресурсов, следует создать соответствующий поток.

Если следует открыть файл ресурсов, связав его с буферированным потоком, открытым на чтение и запись, который, в свою очередь, связан с файлом 'MYRES.RES', эти операции можно записать следующим образом:

```
Stream:= New(tBufStream, Init('MYRES.RES', stOpen, 1024))
        { Открытие буферированного потока для чтения и записи,
    связанного с файлом 'MYRES.RES' и с буфером в 1024 байта }
ResFile:= New(pResourceFile, Init(Stream));
                        { Создание и инициализация файла ресурсов }
```

Размещение объекта в файле ресурсов осуществляется с помощью метода Put:

```
procedure tResourceFile.Put(Item: pObject; Key: String);
```

где Item - указатель на размещаемый в файле ресурсов объект;
Key - имя, присвоенное размещаемому объекту.

Этот метод помещает объект в конец ресурса, проверяет, есть ли уже в ресурсе объект с этим именем, и, если нет, помещает информацию о размещенном объекте в коллекцию, содержащую сведения об адресах, размерах и именах компонент. Если же в потоке уже существует объект с таким же именем, информация о старом объекте из упомянутой выше коллекции удаляется и туда помещается информация о новом объекте. Это приводит к тому, что в потоке остается старый объект, но добраться до него уже становится невозможно. Естественно, что все это произойдет лишь в том случае, если размещаемый объект и его компоненты (если объект является группой) зарегистрированы (см. п. 8.2). Существует способ сжатия информации в файле ресурсов с удалением "мусора", о чем будет сказано ниже.

Если в созданный в предыдущем примере файл ресурсов следует поместить зарегистрированный объект с указателем MyDialog, присвоив ему имя "Окно диалога", выполнить это можно с помощью оператора

```
ResFile^.Put(MyDialog, 'Окно диалога');
```

Извлечение объекта из файла ресурсов осуществляется с помощью метода Get:

```
function tResourceFile.Get(Key: String): pObject;
```

где Key - имя, присвоенное размещенному объекту;
результат - указатель на найденный объект или nil, если в файле ресурсов нет объекта с таким именем.

Метод работает следующим образом. Сначала по имени Key определяется наличие объекта в файле, и, если такого объекта нет, работа метода прекращается с возвращением значения nil. Если же такой объект в файле имеется, в оперативной памяти создается копия объекта и возвращается указатель на этот объект.

Если объект, размещенный в файле ресурсов под именем "Окно диалога", следует взять из файла и разместить в рабочей области, можно воспользоваться оператором

```
DeskTop^.Insert(ResFile^.Get('Окно диалога'));
```

Тот или иной объект можно удалить из файла ресурсов с помощью метода Delete:

```
procedure tResourceFile.Delete(Key: String);
```

где Key - имя, присвоенное удаляемому объекту.

При использовании этого метода из коллекции, содержащей информацию об элементах файла ресурсов (адрес, размер, имя), удаляется информация об удаляемом объекте. Сам же объект из файла ресурсов не удаляется, но становится недоступным (появляется "мусор").

Как уже говорилось выше, в ряде случаев в файле ресурсов появляются недоступные для использования участки - так называемый "мусор". Удалить его из файла ресурсов можно с помощью метода SwitchTo копирования файла в другой поток:

```
function tResourceFile.SwitchTo(AStream:
                                pStream; Pack: Boolean): pStream;
```

где AStream - новый поток, в который переписывается информация;

Pack - признак способа копирования: если Pack = True, копирование происходит с удалением неиспользуемых участков, если Pack = False, копирование происходит без удаления таких участков;

результат - указатель на исходный поток.

Этот метод копирует информацию из потока, связанного с файлом ресурсов, в другой файл. При этом копирование можно осуществить двояким образом - с удалением неиспользуемых участков и без удаления, что зависит от значения параметра Pack. После копирования файл ресурсов будет уже связан с новым потоком.

В файле ресурсов можно хранить библиотеку интерфейсных элементов. Использование таких файлов является хорошим средством повышения эффективности программирования, ускоряя процесс создания программ и уменьшая кодовый сегмент программы примерно на 10-15%.

8.5. Списки строк

В Turbo Vision имеется специальный объект - список строк (тип tStringList), предназначенный для хранения в потоке последовательности строк, которые можно вызывать по номеру,

присвоенному каждой строке. Использование такого объекта позволяет сократить объем кодовой части программы и сравнительно просто адаптировать программу к тому или иному языку, для чего следует лишь заменить содержимое этого объекта на текст соответствующего языка.

Прежде чем рассматривать объект tStringList, познакомимся сначала с объектом, с помощью которого можно сконструировать список строк (тип tStrListMaker). Этот объект позволяет создать список строк и наполнить его соответствующим содержанием. Инициализируется он конструктором Init:

```
constructor tStrListMaker.Init(AStrSize, AIndexSize: Word);
```

где **AStrSize - размер буфера для размещения строк; этот размер должен быть достаточным, чтобы полностью разместить все строки;**

AIndexSize - максимальный индекс размещаемой строки.

Размещаются строки в список строк с помощью метода Put:

```
procedure tStrListMaker.Put(Key: Word; S: string);
```

где **Key - индекс размещаемой строки;**

S - размещаемая строка.

Если в программе-калькуляторе использовать список строк для метода Hint объекта tHintStatusLine, можно написать небольшую программу, создающую этот список и размещающую его в файл ресурсов (см. п. 8.4):

```
program Resource;
uses Objects;
var ResFile: tResourceFile;
      S: tStrListMaker;
begin
   { Регистрация типа tStrListMaker }
   RegisterType(RStrListMaker);
   { Создание файла ресурсов }
   ResFile.Init(New(pBufStream, Init(
                        '_HINT.RES', stCreate, 1024)));
   { Инициализация объекта создания списка строк }
   S.Init(1024, 2500);
   { Заполнение списка строк }
   S.Put(1000, 'Работа с калькулятором');
   S.Put(1001, 'Новый калькулятор');
   S.Put(1002, 'Закрыть калькулятор');
   S.Put(1003, 'Прекращение работы');
   S.Put(1004, 'Параметры калькулятора');
   S.Put(1005, 'Задание параметров');
```

```
    S.Put(1006, 'Параметры стека');
    S.Put(1100, 'Имя калькулятора');
    S.Put(1101, '2-ная система счисления');
    S.Put(1102, '8-ная система счисления');
    S.Put(1103, '10-ная система счисления');
    S.Put(1104, '16-ная система счисления');
    S.Put(1105, 'Стек');
    S.Put(1106, 'Ввод параметров');
    S.Put(1107, 'Отмена');
    S.Put(1200, 'Содержимое стека');
    S.Put(2000, 'Очистка стека');
    S.Put(2001, 'Удаление числа из стека');
    S.Put(2002, 'Очистка последнего числа');
    S.Put(2003, 'Удаление символа из числа');
    S.Put(2004, 'Цифра "D"');
    S.Put(2005, 'Цифра "E"');
    S.Put(2006, 'Цифра "F"');
    S.Put(2007, 'Изменение знака');
    S.Put(2008, 'Цифра "A"');
    S.Put(2009, 'Цифра "B"');
    S.Put(2010, 'Цифра "C"');
    S.Put(2011, 'Знак деления');
    S.Put(2012, 'Цифра "7"');
    S.Put(2013, 'Цифра "8"');
    S.Put(2014, 'Цифра "9"');
    S.Put(2015, 'Знак умножения');
    S.Put(2016, 'Цифра "4"');
    S.Put(2017, 'Цифра "5"');
    S.Put(2018, 'Цифра "6"');
    S.Put(2019, 'Знак вычитания');
    S.Put(2020, 'Цифра "1"');
    S.Put(2021, 'Цифра "2"');
    S.Put(2022, 'Цифра "3"');
    S.Put(2023, 'Знак сложения');
    S.Put(2024, 'Цифра "0"');
    S.Put(2025, 'Десятичная точка');
    S.Put(2026, 'Новый элемент стека');
    { Размещение списка строк в файле ресурсов }
    ResFile.Put(@S, 'Hints');
    { Завершение работы программы }
    S.Done;
    ResFile.Done;
end.
```

Сам список строк tStringList конструктора не имеет, получить из него строку можно с помощью метода Get:

```
function tStringList.Get(Key: Word): string;
```

где Key - индекс строки;
результат - найденная строка; если строки с указанным индексом нет, возвращается пустая строка ''.

Не имеет список строк и средств для размещения в нем новых строк, их размещение можно осуществлять только с помощью объекта tStrListMaker.

*Для того чтобы в программе-калькуляторе воспользоваться списком строк, сформированным программой Resource и размещенным в файле ресурсов, следует сделать следующие изменения (новые строки программы помечены далее символами { *}). Во-первых, следует ввести две глобальные переменные, разместив их в начале программы до описания методов объектов:*

```
var                                                    { *}
   ResFile: tResourceFile;                             { *}
   S: pStringList;                                     { *}
```

Затем следует изменить конструктор объекта tMyApp с тем, чтобы он открывал файл ресурсов и читал из него список строк.

```
constructor tMyApp.Init;                               {661}
begin                                                  {662}
   { Регистрация списка строк }
   RegisterType(RStringList);                          { *}
   { Открытие файла ресурсов }
   ResFile.Init(New(pBufStream, Init(
                '_HINT.RES', stOpenRead, 1024)));      { *}
   { Чтение списка строк }
   S:= pStringList(ResFile.Get('Hints'));              { *}
   inherited Init;                                     {663}
   HelpCtx:= 500;
   RegisterHelpFile;                                   {664}
   DisableCommands([cmParamrters, cmStack]);           {665}
end;                                                   {666}
```

Наконец, следует переписать метод Hint объекта tHintStatusLine:

```
function tHintStatusLine.Hint(AHelpCtx: Word): string;  { 79}
begin                                                   { 80}
   Hint:= S^.Get(AHelpCtx);                             { *}
end;                                                    {127}
```

8.6. Использование редакторов ресурсов

Если программисту приходится часто создавать ресурсы, то при применении описанной выше последовательности действий по созданию и использованию ресурсов приходится сталкиваться с рядом проблем. Во-первых, при любой, даже весьма незначительной, модификации содержимого файла ресурсов, уже созданного с помощью некоторой программы, приходится писать специальную программу для внесения изменения в файл ресурсов. Во-вторых, программирование характеристик интерфейсных элементов, а они-то наиболее часто и используются в ресурсах, требует обычно как при создании, так и при модификации отображения их на экране и последующей корректировки при необходимости, что довольно трудоемко. Поэтому для решения этих проблем желательно иметь средство, позволяющее создавать и редактировать ресурсы, работающее на уровне файла ресурсов, независимо от основной программы. К сожалению, в стандартную поставку Turbo Vision не входит редактор ресурсов.

Можно самому создать такой редактор (кстати, здесь очень полезны будут средства пакета Turbo Vision), но для создания качественного редактора потребуется довольно много времени. В АО "Диалог-МИФИ" можно приобрести уже готовый редактор ресурсов. Демонстрационная версия этого редактора имеется на прилагаемой к пособию дискете. Эта версия выполняет все операции за исключением записи созданного ресурса в файл и удаления ресурса из файла.

Основные возможности, предоставляемые этим редактором, следующие.

В качестве ресурсов создаются меню, сроки состояния и диалоговые окна.

Меню и строки состояния создаются с помощью объекта tOutline (см. п. 9.2), отображающего иерархическую структуру, свойственную этим элементам. Компоненты, входящие в меню и строки состояния, перечислены в пп. 3 и 6.

Диалоговые окна могут содержать строку состояния, селективные и триггерные списки, кнопки, метки, протоколы и статический текст. Строка ввода может содержать анализатор по шаблону, по диапазону и по определенному множеству символов вводимой информации. Диалоговое окно в отличие от двух других видов ресурсов представляется на экране за малым исключением в том виде, в котором оно будет использовано в программе. Компоненты окна непосредственно реализуют при отображении на экране заданные параметры. Как компонентам, так

и диалоговому окну в целом можно задать различные флаги характеристик (см. пп. 1.3, 4.1, 4.3, 4.4).

Параметры всех ресурсов и их компонент определяются через меню, единое для всех этих компонент. Предусмотрено средство копирования, вырезания и вставки компонент из одного ресурса в другой (разумеется, ресурсы должны быть одного типа). При этом работа может производиться одновременно с несколькими ресурсами.

Редактор ресурсов позволяет создавать, открывать, сохранять (в том числе и под другим именем) файлы ресурсов. Ресурсы этих файлов можно редактировать, удалять, дополнять.

Редактор ресурсов является русифицированным программным продуктом и имеет справочную систему, позволяющую на любом этапе работы с пакетом определить дальнейшие действия.

По ряду причин в создаваемых ресурсах отсутствует возможность включения любых стандартных элементов, однако если создаваемый объект должен содержать элементы, не предусмотренные в пакете (например, список коллекции строк с линейкой скроллинга в диалоговом окне), можно создать объект без этих элементов и разместить его в файле ресурсов. Далее после вызова этого объекта в программе, его можно дополнить необходимыми элементами.

Редактор особенно удобен в тех случаях, когда следует внести в ресурс непринципиальные изменения, т. е. связанные лишь с переопределением параметров ресурса и его элементов, флагов, взаимным расположением элементов, их размерами и т. д. В этом случае вносимые изменения в файл ресурсов не потребуют никаких изменений и даже перекомпиляции основной программы, где используется этот файл ресурсов.

9. НЕКОТОРЫЕ ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ TURBO VISION

Пакет Turbo Vision обладает, конечно, и множеством других возможностей, которые можно использовать при создании пользовательских программ (см., например, [2], [3]). В данной главе будут рассмотрены три таких возможности: создание текстовых редакторов, отображение древовидных структур и задание цветовой палитры с помощью специального диалогового окна. Далеко не во всех программах может потребоваться использование этих возможностей, и материалы данной главы можно читать лишь по мере необходимости.

9.1. Текстовый редактор

В настоящее время создано большое количество текстовых редакторов, обладающих различными возможностями, тем не менее в некоторых случаях требуется создать свой собственный текстовый редактор, особенно когда он должен входить составной частью какого-либо пакета, как сделано, например, в пакете Borland Pascal (Turbo Pascal). Пакет Turbo Vision содержит ряд объектов, на основе которых можно создать неплохие текстовые редакторы. Исходным здесь является объект tEditor, выполняющий практически все операции по редактированию текста, однако не связанный с конкретным хранилищем информации. Мы рассмотрим потомок этого объекта - объект tFileEditor, позволяющий создавать редакторы, связанные с текстовыми файлами. Без особых изменений можно создавать редакторы, работающие одновременно с любым количеством файлов, содержащих тексты размером до 64 К, обеспечивающие работу с клавиатурой и мышью, поиск в тексте и замену информации, использование буфера, позволяющие копировать, вырезать и размещать фрагменты информации в редактируемом тексте, отменять сделанные в строке изменения, задавать автоматические отступы в тексте и т. д. Короче говоря, такие редакторы обладают почти всеми возможностями, которые имеются у редакторов пакета Borland Pascal (Turbo Pascal), см., например, [1].

В приложении Ж приведен заимствованный из демонстрационных файлов пакета Turbo Vision, несколько модифицированный и русифицированный текстовый редактор, обладающий всеми перечисленными выше возможностями. На основе этого редактора, внося те или иные изменения, можно построить целый класс редакторов с требуемым набором возможностей. К сожалению, без более существенных изменений исходных текстов не удается использовать в качестве клавиш быстрого управления клавиши, соответствующие буквам кириллицы. Поэтому меню, используемое в этом редакторе, не имеет клавиш быстрого управления, позволяющих вызывать с их помощью те или иные элементы меню, а, следовательно, в текстах меню нет и выделенных особым цветом букв - см. п. 3.1.

При создании редактора следует выполнить ряд этапов работы, связанных с вспомогательными операциями и непосредственным созданием самого редактора. Выполнение этих этапов упрощается наличием в пакете Turbo Vision специализированных стандартных объектов.

По-видимому, целесообразно начать работу с организации определения файла, в котором находится редактируемая информация. С этой целью можно использовать стандартное меню работы с файлами, определяемое функцией StdFileMenuItems и которое позволяет создать новый файл, найти имеющийся, сохранить информацию в файле, сохранить информацию из одного файла в другом, сохранить информацию всех измененных файлов, сменить текущий каталог, временно выйти в ДОС, завершить работу в программы.

В текстовом редакторе приведен русифицированный вариант этой функции MyStdFileMenuItems (лл. 196-199).

Обработка команд, формируемых этим меню, осуществляется либо автоматически (таким образом обрабатываются, например, команды cmSave - сохранить содержимое файла - или cmDosShell - временный выход в ДОС), либо с помощью собственных подпрограмм, но опять-таки использующих стандартные средства Turbo Vision (например, диалоговое окно выбора файла tFileDialog или диалоговое окно смены текущего каталога tChDirDialog).

В текстовом редакторе команды cmSave, cmSaveAs, cmSaveAll, cmDosShell и cmQuit (л. 198) обрабатываются автоматически стандартными средствами пакета. Для обработки команды cmNew (создание нового файла) используется процедура FileNew (лл. 428-431), которая, в свою оче-

*редь вызывает метод OpenEditor (лл. 380-411) с параметрами, позволяющими создать окно редактора (см. ниже), связанное с файлом, не имеющим имени, не являющееся буфером (см. ниже) и отображаемое на экране. Для обработки команды cmOpen (открыть файл) используется метод FileOpen (лл. 412-419), который вызывает диалоговое окно типа tMyFileDialog (лл. 9-11, 217-277) - русифицированный потомок диалогового окна выбора файла tFileDialog. Обращение к нему осуществляется в данном случае с маской *.*, позволяющей просмотреть все файлы текущего каталога. Вид этого окна приведен на рис. 9.1.*

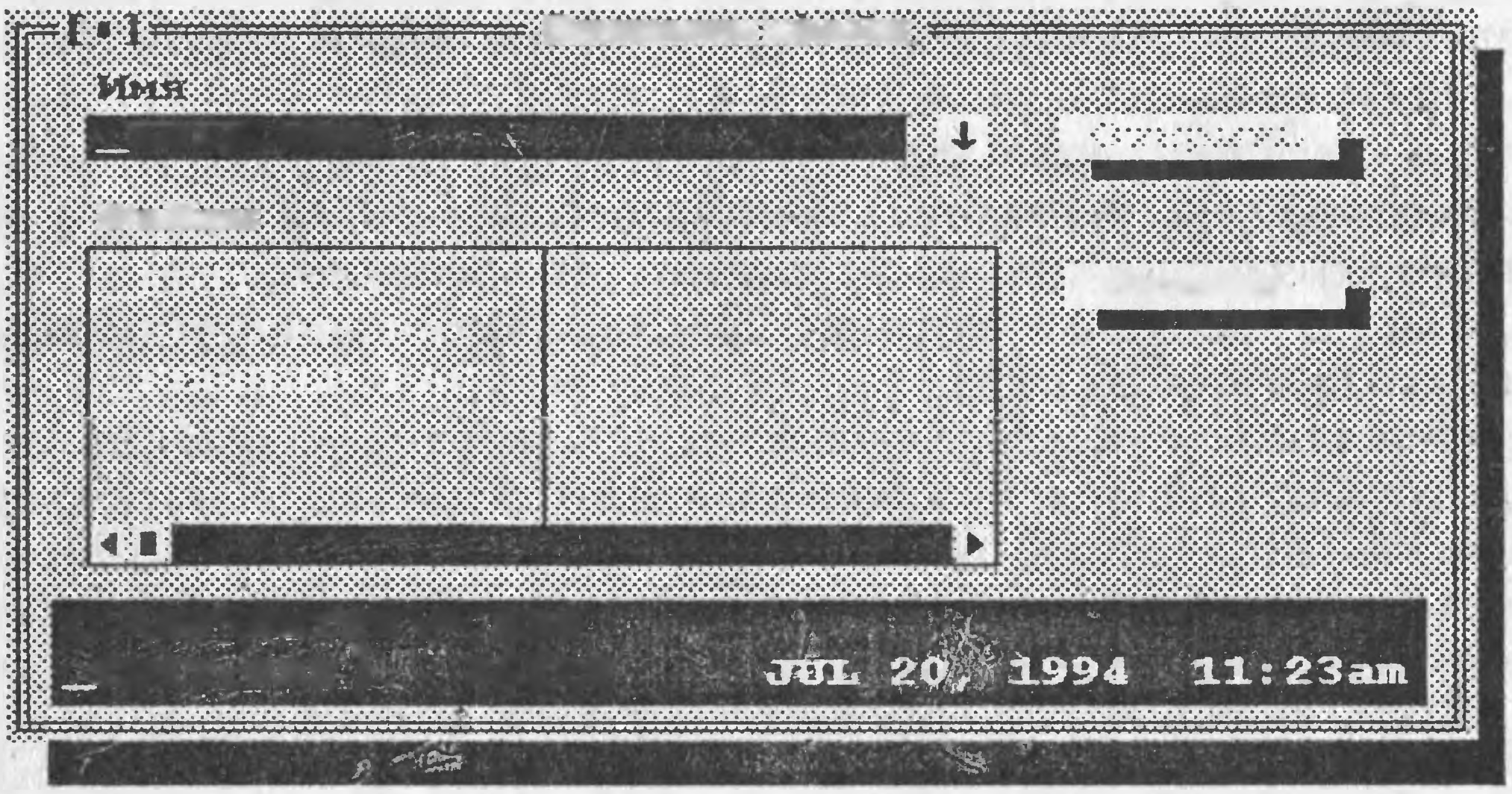


Рис. 9.1. Диалоговое окно выбора файла

После выбора файла опять-таки вызывается метод OpenEditor (лл. 380-411) с параметрами, позволяющими создать окно редактора (см. ниже), связанное с выбранным файлом, не являющееся буфером (см. ниже) и отображаемое на экране. Команда cmChangeDir (сменить текущий каталог) обрабатывается подпрограммой ChangeDir (лл. 421-427), которая вызывает диалоговое окно tMyChDirDialog (лл. 17-21, 278-352) - русифицированный потомок окна смены текущего каталога tChDirDialog, которое позволяет по дереву каталогов выбрать новый текущий каталог, в котором можно просматривать файлы. Внешний вид этого окна изображен на рис. 9.2.

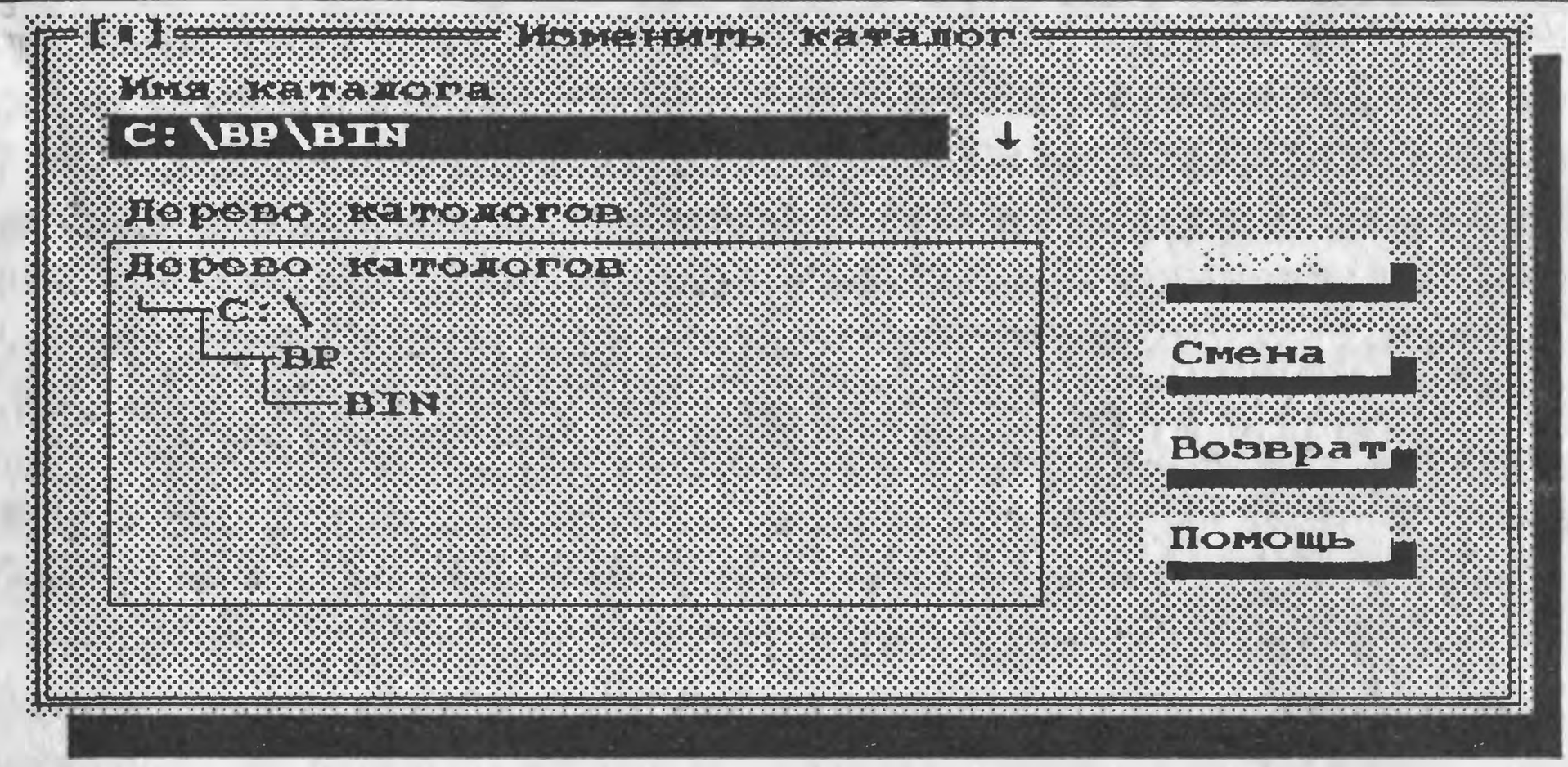


Рис. 9.2. Диалоговое окно смены текущего каталога

После выбора файла можно приступить к созданию самого редактора. Как уже говорилось, создать текстовый редактор можно на основе объекта tFileEditor, который обеспечивает перечисленные выше особенности редакторов. Инициализируется такой редактор конструктором Init:

```
constructor tFileEditor.Init(var Bounds: tRect;
  AHScrollBar, AVScrollBar: pScrollBar;
  AIndicator: pIndicator; AFileName: pNameStr);
```

где Bounds - область, выделенная под редактируемый текст;
AHScrollBar, AVScrollBar - линейки горизонтального и вертикального скроллинга редактируемого текста;
AIndicator - индикатор положения курсора в тексте (номер строки и позиции в строке);
AFileName - имя текстового файла.

Объект tFileEditor, так же как и объект tEditor, обладает большим числом параметров. Отметим только те параметры, которые определяют режим работы редактора. Эти параметры либо должны быть заданы непосредственно в программе, либо в процессе организации соответствующего диалога.

Основным здесь является глобальный параметр EditorFlags (доступный всем открытым редакторам и общий для всех них), являющийся типизированной константой:

```
const
  EditorFlags: Word = efBackupFiles + efPromptOnReplace;
```

Назначение битов слова следующее (в круглых скобках приведены константы, соответствующие единице в указанном бите).

0-й бит (efCaseSensitive) - если данный бит установлен, то при поиске информации различаются строчные и прописные буквы.

1-й бит (efWholeWordsOnly) - если установлен данный бит, осуществляется поиск только целых слов.

2-й бит (efPromptOnReplace) - если установлен этот бит, перед заменой информации требуется подтверждение пользователя.

3-й бит (eReplaceAll) - при установленном данном бите осуществляется поиск и замена найденных в тексте фрагментов.

4-й бит (efDoReplace) - если установлен данный бит, найденный фрагмент заменяется новым (этим флагом различаются режим поиска и режим замены).

5-й бит (efBackupFiles) - если установлен данный бит, при сохранении отредактированной информации в файле старая информация сохраняется с расширением .BAK.

Кроме этого параметра следует отметить три параметра логического типа: CanUndo, позволяющий осуществлять отмену изменений, произведенных в строке (этот флаг должен быть сброшен для буфера), Overwrite, задающий режим замены символов вместо режима вставки, и AutoIndent - автоматическое создание отступов, когда при переходе на новую строку курсор помещается в позицию, соответствующую началу предыдущей строки.

Как правило, непосредственно объект tFileEditor в программе не используется, т. к. он не имеет соответствующего оформления на экране. Желательно поместить редактор в соответствующее окно. В связи с этим в Turbo Vision имеется специализированное окно - окно редактора (тип tEditWindow), которое инициализируется конструктором Init:

```
constructor tEditWindow.Init(var Bounds: tRect;
                             FileName: pNameStr; ANumber: Integer);
```

где Bounds - размеры окна;
FileName - имя текстового файла с редактируемой информацией;
ANumber - номер окна.

Этот конструктор сначала формирует окно с заданными размерами и номером, а затем создает редактор типа tFileEditor. Для редактора создаются линейки скроллинга с помощью стандартной функции StandardScrollBar объекта tWindow (см. п. 4.2.4), которая размещает их на рамке окна редактора, созда-

ется индикатор, позволяющий фиксировать положение курсора в тексте (номер строки и позиции в строке), наконец, редактору передается имя файла с редактируемой информацией. Окно редактора обладает двумя особенностями по сравнению с обычным окном: во-первых, если в окне размещен буфер, то при закрытии окна (например, с помощью пиктограммы закрытия) окно не закрывается, а становится невидимым (см. п. 1.3); и во-вторых, если изменяется файл, связанный с редактором (например, при обработке команды cmSaveAs), изменяется и имя окна.

В текстовом редакторе окно редактора создается методом OpenEditor (лл. 380-411), которому в качестве параметров передается имя файла, с которым будет работать редактор (или пустая строка, если создается новый файл), признак, является ли создаваемый редактор буфером, и признак непосредственного отображения окна редактора на экране. Метод OpenEditor определяет размеры создаваемого окна в зависимости от наличия других окон редактора, выбирает номер окна из множества незанятых номеров, создает окно и размещает его в рабочей области. Размер буфера (см. ниже) всегда выбирается во все рабочее поле экрана. Если окно не должно быть непосредственно отображено на экране, оно временно скрывается (л. 407). Это требуется для буфера, т. к. он отображается на экране только по запросу. Пример окна редактора представлен на рис. 9.3.

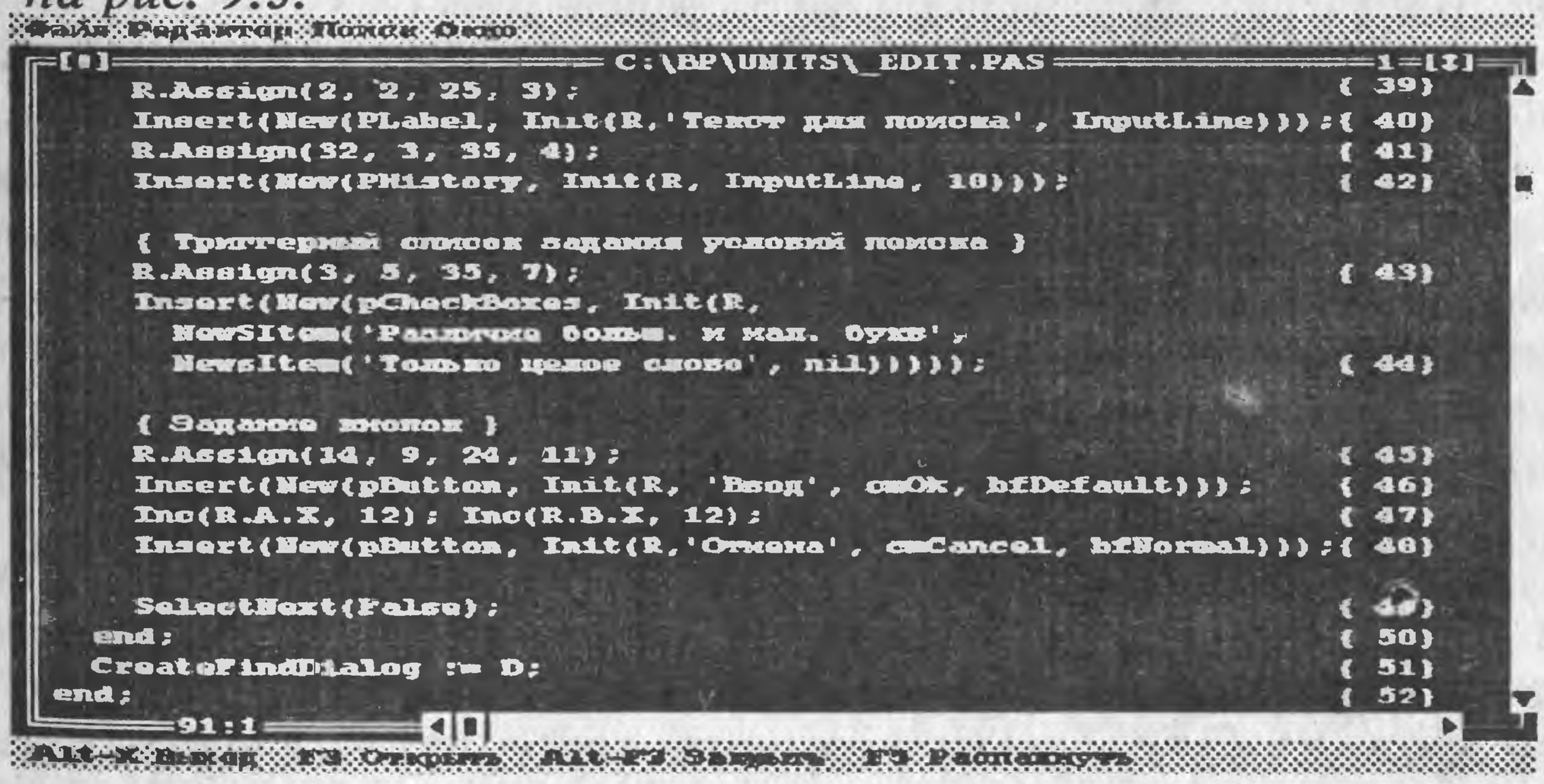


Рис. 9.3. Окно редактора

Далее выполняются работы по совершенствованию редактора. Если предполагается копировать или переносить фрагменты текста из одного места в другое, следует использовать буфер - спе-

циальный вид редактора, в который временно помещается переносимая информация. Особенностями буфера являются отсутствие имени файла, с которым он связан, отображение окна буфера на экране только по мере необходимости; как правило, окно буфера не имеет номера и занимает все рабочее поле. В буфере можно, как и в обычном редакторе, редактировавать помещенную информацию, однако отмена введенных в строку изменений у него отключается (л. 353). Буфер редактора обычно создается конструктором Init основного объекта программы.

В текстовом редакторе буфер создается в конструкторе объекта tMyEditor (лл. 363-368).

После создания буфера можно приступить к оформлению операций редактора, связанных с перемещением информации, - копированием, удалением и т. д. С этой целью можно использовать стандартное меню, формируемое функцией StdEditMenuItems. С помощью этого меню можно отменить изменения, сделанные в текущей строке, переместить фрагмент текста в буфер с удалением из текста, скопировать фрагмент текста в буфер, поместить в текст фрагмент из буфера, удалить фрагмент из текста. Сюда же можно добавить дополнительные элементы меню, позволяющие работать с текстом, например элемент, дающий возможность просмотреть содержимое буфера. Как правило, команды стандартного меню обрабатываются автоматически уже имеющимися стандартными средствами. Выделение удаляемого или перемещаемого фрагмента осуществляется теми же средствами, что и в среде Borland Pascal (Turbo Pascal) - с помощью мыши и клавиатуры [1].

В текстовом редакторе при создании меню (л. 480) используется функция MyStdEditMenuItems (лл. 200-203) - русифицированный вариант функции StdEditMenuItems. Здесь же добавлен элемент, с помощью которого можно выбрать команду cmShowClip, позволяющую посмотреть содержимое буфера. Эта команда обрабатывается процедурой ShowClip (лл. 432-436).

Другая группа операций с редактором относится к поиску и замене фрагментов текста. Стандартного меню, с помощью которого выполняются эти операции, нет, а команды, используемые здесь (cmFind - найти фрагмент, cmReplace - заменить фрагмент, cmSearchAgain - найти следующий фрагмент), обрабатываются автоматически имеющимися стандартными средствами. Все эти команды целесообразно объединить в одно подменю поиска и замены.

Процедура инициализации меню текстового редактора (лл. 474-481) формирует подменю "Поиск", объединяющее указанные выше команды. В редакторе имеются две функции формирования диалоговых окон. Первое диалоговое окно (лл. 36-63) предназначено для ввода фрагмента, который будет разыскиваться в тексте, второе (лл. 64-98) - для ввода разыскиваемого фрагмента и фрагмента, на который после нахождения он будет заменен. Кроме этого, в окнах задаются условия поиска и замены: следует ли различать прописные и строчные буквы, нужно ли подтверждение на замену и т. д. Внешний вид диалогового окна замены одного фрагмента текста на другой представлен на рис. 9.4.

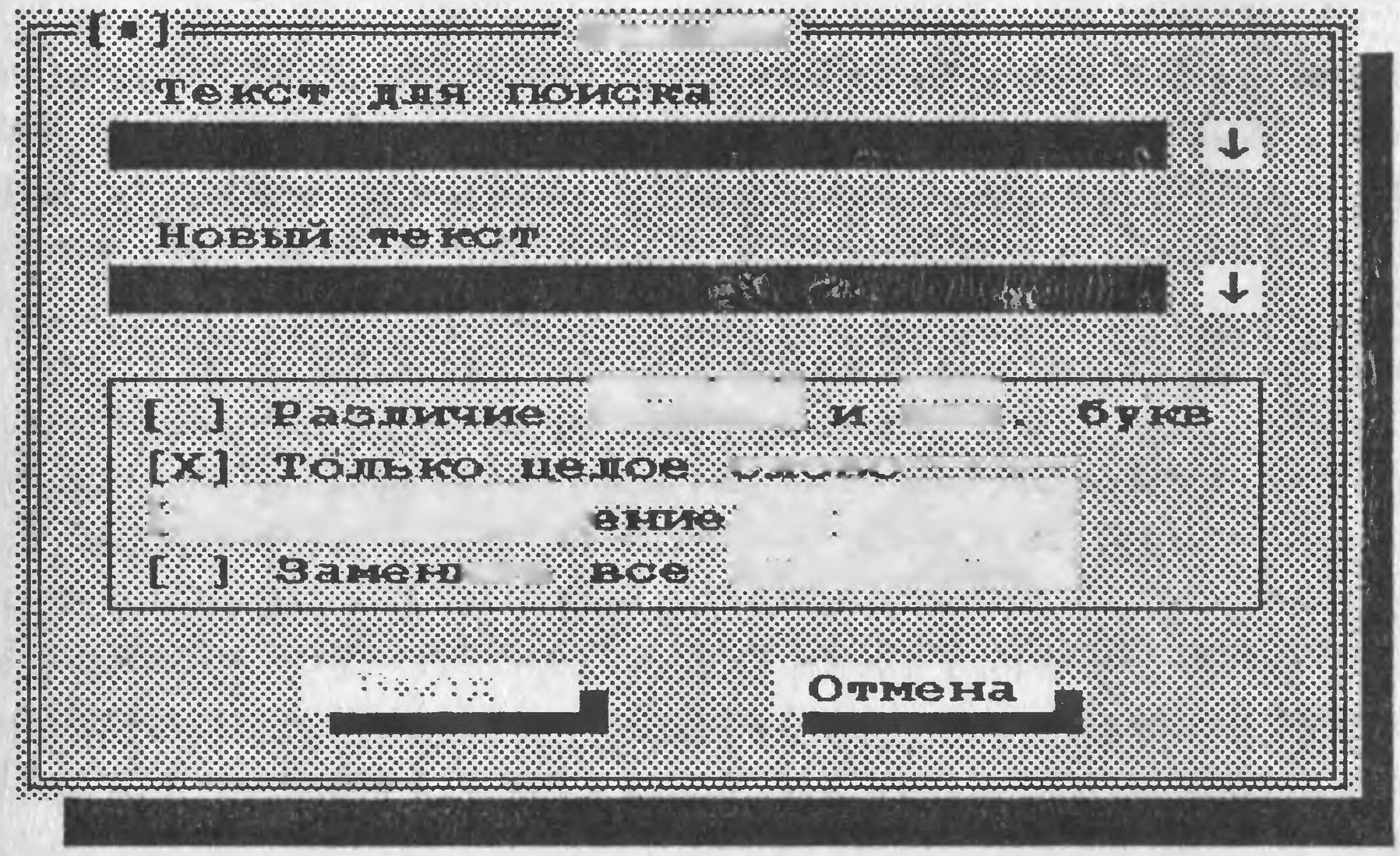


Рис. 9.4. Окно замены фрагментов текста

Наконец, следует предусмотреть операции с окнами. Для этого также можно использовать стандартное меню, формируемое функцией StdWindowMenuItems. Это меню позволяет разместить открытые окна встык или каскадно, закрыть все окна, перейти в режим перемещения или изменения размеров окна, распахнуть окно и вернуть его в исходное состояние, активизировать следующее или предыдущее окно, закрыть окно.

В текстовом редакторе используется функция MyStdWindowMenuItems (лл. 204-207) - русифицированный вариант стандартной функции StdWindowMenuItems, позволяющая создавать подменю работы с окнами.

Еще один важный момент следует отметить. В отличие от других объектов программы редактор размещает свой буфер

не в динамической области памяти, а над ней. Область для размещения буфера редактора следует создать, ограничив динамическую область памяти, отводимую программе и которая стандартно занимает всю свободную область оперативной памяти. Для этого следует в глобальную переменную MaxHeapSize поместить размер динамической области памяти, отводимой программе, в 16-байтовых параграфах. Это необходимо сделать до первого размещения данных в динамической области памяти, лучше первым оператором конструктора Init основного объекта программы.

Конструктор объекта tMyEditor текстового редактора имеет оператор (л. 359), присваивающий переменной MaxHeapSize значение 4096, что соответствует динамической области памяти в 64 К.

Мы не будем рассматривать тонкости работы редактора и то, каким образом формируется и изменяется редактируемая информация, хотя это может вызвать вопросы, связанные с имеющимися в рассматриваемой программе подпрограммами и методами MyStdEditorDialog (лл. 146-183), GetTitle (лл. 208-216), OpenEditor (лл. 380-411), OutOfMemory (лл. 490-493). К этим подпрограммам и методам следует относиться как к русифицированным вариантам стандартных объектов (что и есть на самом деле), которые имеют именно такую структуру и используются в соответствии с алгоритмом работы этих объектов.

Рассмотренный текстовый редактор является отличным примером преимуществ использования средств пакета Turbo Vision. Обладая большим набором возможностей, удобный и быстродействующий, он занимает всего 500 строк, причем свыше 250 из них связаны с русификацией соответствующих стандартных методов и подпрограмм. При этом загрузочный модуль занимает около 95 К памяти. Таким образом, основная часть программы компонуется из стандартных средств пакета Turbo Vision.

Несколько слов относительно используемого в текстовом редакторе методе Idle объекта tProgram

```
procedure tProgram.Idle; virtual;
```

Этот метод выполняется программой в свободное время, когда нет событий, которые программа должна обрабатывать. Стандартно этот метод работает со строкой состояния, выбирая тот или иной его вариант в зависимости от текущего значения параметра контекстной помощи (см. п. 5). Этот метод можно переопределить для выполнения каких-либо других операций. Следует только иметь в виду, что эти операции должны быть быстродей-

ствующими, т. к. в противном случае может задержаться обработка очередного поступившего события.

В текстовом редакторе метод Idle переопределен таким образом, что наряду со своими обычными действиями он разрешает или запрещает выполнение команд размещения открытых окон встык или каскадно в зависимости от того, установлен ли у открытых окон флаг ofTileable (см. п. 1.3). Метод Idle в данной программе можно было бы и не переопределять, т. к. у используемых окон редактора типа tEditWindow устанавливается флаг ofTileable, команды размещения открытых окон встык или каскадно можно разрешать постоянно. Метод Idle приведен здесь лишь как пример его использования в программе.

9.2. Отображение древовидных структур

Часто в программе используются древовидные иерархические структуры, которые желательно отобразить на экране с изображением связей между отдельными элементами. К таким структурам, в частности, относятся меню и строки состояния. Для отображения подобных структур в Turbo Vision имеются два специальных объекта, первый из которых - абстрактый объект tOutlineViewer - осуществляет отображение структуры на экране, а второй - tOutline - осуществляет работу с конкретной древовидной структурой.

Для того чтобы отобразить ту или иную структуру на экране, прежде всего следует создать специальное дерево структуры, каждый из элементов которого содержит информацию о выводимом на экране тексте и связях с соседними элементами. Имеется стандартный тип tNode для описания элементов структуры:

```
type
   pNode = ^tNode;
   tNode = record
      Next: pNode;
      Text: pString;
      ChildList: pString;
      Expanded: Boolean
   end;
```

где Next - указатель на следующий элемент структуры этого же уровня;
Text - указатель на строку, которая выводится на экран для данного элемента;

ChildList - указатель на список элементов структуры следующего уровня, связанных с данным элементом (точнее, указатель на первый элемент списка);
Expanded - параметр, определяющий, будут ли отображены на экране элементы следующего уровня; если он равен True, элементы следующего уровня отображаются, в противном случае - нет.

Для формирования элемента такого типа можно использовать стандартную функцию NewNode:

```
function NewNode(const AText: string;
                                AChildren, ANext: pNode): pNode;
```

где AText - текст, выводимый на экране для данного элемента структуры;
AChildren - указатель на первый элемент структуры следующего уровня (равен nil, если элементов следующего уровня нет);
ANext - указатель на следующий элемент структуры текущего уровня (равен nil, если такого элемента нет);
результат - указатель на созданный элемент.

Многократно применяя эту функцию, можно создать все дерево структуры.

В редакторе ресурсов (см. п. 8.5) имеется рекурсивная функция BuildMenuTree формирования дерева меню:

```
function BuildMenuTree(Items: pMenuItem): pNode;
   {Items - указатель на первый элемент отображаемого меню}
begin
   if Items = nil then,
      BuildMenuTree:= nil
   else if Items^.Name = nil then
      { Разделительная линия }
      BuildMenuTree:= NewNode (' ———————— (разделитель)', nil,
                                        BuildMenuTree(Items^.Next));
   else if Items^.Command = 0 then
      { Подменю }
      BuildMenuTree:= NewNode(Items^.Name^ + ' (меню)',
                        BuildMenuTree(Items^.SubMenu^.Items),
                        BuildMenuTree(Items^.Next))
   else
      { Команда }
      BuildMenuTree:= NewNode(Items^.Name^ + (команда)',
                          nil, BuildMenuTree(Items^.Next));
end;
```

Добавить к сформированной функцией BuildMenuTree структуре основной заголовок структуры (корень дерева) можно, например, с помощью функции NewTree:

```
function NewTree(Menu: tMenuView): pNode;
begin
   NewTree:= NewNode(
   'МЕНЮ', BuildMemuTree(Menu^.Menu^.Items), nil);
end;
```

После формирования дерева структуры можно создать объект tOutline отображения древовидной структуры. Заголовок его конструктора Init имеет вид:

```
constructor tOutline.Init(var Bounds: tRect;
   AHScrollBar, AVScrollBar: pScrollBar;
   ARoot: pNode);
```

где Bounds - размеры формируемого объекта;
AHScrollBar, AVScrollBar - горизонтальная и вертикальная линейки скроллинга;
ARoot - указатель на корневой элемент отображаемой структуры.

В редакторе ресурсов создается диалоговое окно отображения структуры меню:

```
type
   pMenuDialog = ^.tMenuDialog;
   tMenuDialog = object(tDialog)
      Tree: pNode;                          { Дерево структуры }
      OutLin: pOutline;          { Элемент отображения структуры }
      constructor Init(Menu: pMenuView);
   end;

                . . .

constructor tMenuDialog.Init(Menu: pMenuView);
var
   R: tRect;
   HScrollBar: pScrollBar;
   VScrollBar: pScrollBar;
begin
   { Создание окна }
   R.Assign(0, 0, 74, 15);
   inherited Init(R, 'Меню');
   Palette:= wpGrayWindow;
   Options:= Options or ofCentered;
   Flags:= wfMove + wfGrow + wfClose + wfZoom;
```

```
  { Создание линеек скроллинга - см. п. 4.2.4 }
  VScrollBar:= StandardScrollBar(
                              sbVertical or sbHandleKeyboard);
  HScrollBar:= StandardScrollBar(
                             sbHorizontal or sbHandleKeyboard);

  { Создание структуры }
  Tree:= NewTree(Menu);

  { Создание элемента отображения структуры }
    R.Grow(-1, -1);
    OutLin:= New(pOutline,
    Init(R, HScrollBar, VScrollBar, Tree);

    Insert(Outlin);
end;
```

Пример такого окна приведен на рис. 9.5.

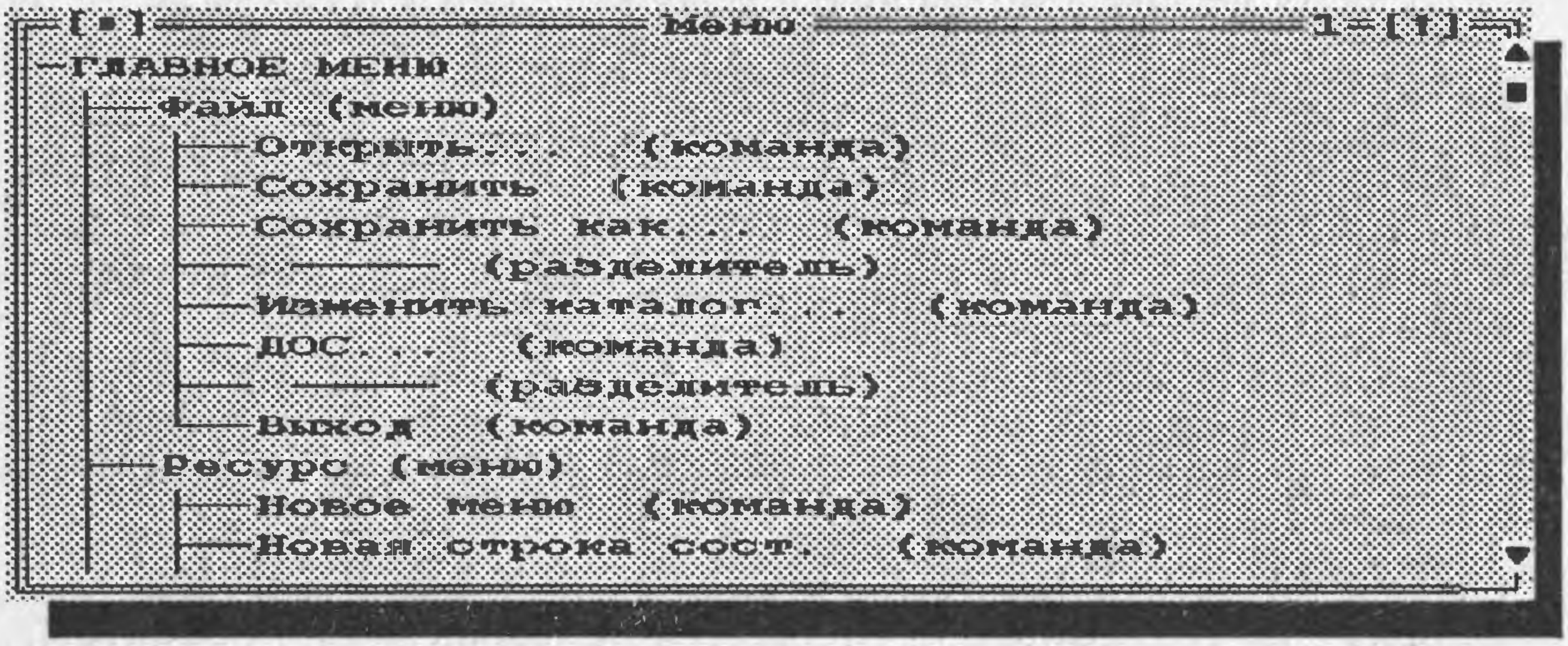


Рис. 9.5. Окно отображения структуры меню

Объект tOutline не только отображает древовидную структуру на экране, но и позволяет скрывать или показавать на экране те или иные ветви структуры с помощью мыши или клавиатуры. Для того чтобы скрыть какую-либо ветвь структуры, следует подвести курсор к вершине скрываемой ветви и либо нажать клавишу мыши, либо нажать клавишу клавиатуры ”-”. Вся ветвь за исключением ее вершины будет скрыта, а вместо нее появится символ ”+”. Для того чтобы вновь отобразить скрытую ветвь, следует подвести курсор к вершине скрытой ветви и либо нажать клавишу мыши, либо клавишу клавиатуры ”+”.

9.3. Диалоговое окно задания цветовой палитры (tColorDialog)

Помимо способов изменения цветовой палитры, отмеченных в п. 7, можно для оперативного изменения цветов непосредственно из программы воспользоваться стандартным диалоговым окном задания цветов tColorDialog. Следует только помнить, что с помощью этого окна можно изменять лишь имеющиеся цвета (но не создавать новые) и, кроме этого, изменение цветов произойдет одновременно у всех отображаемых объектов программы соответствующего типа.

Для того чтобы изменить цвета с помощью этого диалогового окна, ему следует задать исходную палитру с указанием, к какому конкретно элементу и к какому отображаемому объекту относится тот или иной атрибут цвета. Эта операция осуществляется наподобие операций создания меню и строк состояния. Каждому элементу отображаемого объекта в том или ином состоянии (например, рамка пассивного окна, рамка активного окна и т. д.) ставится в соответствие специальный элемент цветовой структуры типа tColorItem:

```
type
   tColorItem = record
      Name: pString;
      Index: Byte;
      Next: pColorItem;
   end;
```

где Name - имя элемента с заданным атрибутом; в качестве имени может выступать имя реального объекта (например, "Frame Active" или "Активная рамка" и т. д.);
Index - номер элемента цветовой палитры, в котором находится атрибут для данного элемента;
Next - указатель на следующий элемент данной группы элементов (о группе элементов см. ниже); этот указатель равен nil, если следующего элемента нет.

Все элементы цветовой структуры объединяются в группы. Эти группы должны соответствовать конкретным отображаемым объектам (диалоговое окно, окно, меню и т. д.). Здесь целесообразно придерживаться структуры, принятой для основной палитры Turbo Vision. Тип такой группы - tColorGroup:

```
type
   tColorGroup = record
      Name: pString;
```

```
    Index: Byte;
    Items: pColorItem;
    Next: pColorGroup;
  end;
```

где Name - имя группы;
Index - номер элемента цветовой палитры, в котором находится атрибут для данной группы;
Items - указатель на первый элемент группы;
Next - указатель на следующую группу; этот указатель равен nil, если следующей группы нет.

Для создания элементов цветовой структуры и групп можно воспользоваться двумя стандартными функциями. Первая из них - ColorItem:

```
function ColorItem(Name: String;
                   Index: Byte; Next: pColorItem): pColorItem;
```

где Name - имя элемента;
Index - номер элемента цветовой палитры, в котором находится атрибут для данного элемента;
Next - указатель на следующий элемент данной группы элементов; этот указатель равен nil, если следующего элемента нет;
результат - указатель на созданный элемент.

Эта функция позволяет создать элемент цветовой структуры. Если с помощью этой функции создается элемент для отображения в диалоговом окне tColorDialog, размер имени Name элемента должен быть не больше 19 символов. Вторая функция - ColorGroup - создает группу элементов цветовой структуры:

```
function ColorGroup(Name: String; Items:
                    pColorItem; Next: pColorGroup): pColorGroup;
```

где Name - название группы;
Items - указатель на первый элемент группы;
Next - указатель на следующую группу элементов (равен nil, если следующей группы нет);
результат - указатель на созданную группу элементов.

Если с помощью этой функции создается группа для отображения в диалоговом окне tColorDialog, размер имени Name группы должен быт не больше 14 символов.

Для отображения в диалоговом окне стандартной палитры Turbo Vision можно использовать стандартные функции формирования групп рабочего поля (DesktopColorItems), меню и стро-

ки состояния (MenuColorItems), окна (WindowColorItems) и диалогового окна (DialogColorItems).

В приложении 3 приведена заимствованная из демонстрационного файла Turbo Vision программа, несколько видоизменненная и русифицированная. Она позволяет задавать изменения цветов в исходной стандартной палитре Turbo Vision. Эта программа позволяет также посмотреть изменения некоторых цветов у диалогового окна выбора файла, взятого из текстового редактора (приложение Ж). В этой программе приведены русифицированные функции MyDesktopColorItems (лл. 17-20), MyMenuColorItems (лл. 21-24), MyWindowColorItems (лл. 25-33) и MyDialogColorItems (лл. 34-42), создающие группы, соответствующие рабочей области, меню (строке состояния), окну и диалоговому окну. На основе этих функций создается структура, соответствующая полной стандартной палитре Turbo Vision, и размещается в диалоговом окне tColorDialog (л. 97).

Заключение

В данном пособии авторы не ставили задачу описать все особенности, возможности и элементы пакета Turbo Vision. Здесь отражены лишь основные моменты, которые, на наш взгляд, необходимо знать для понимания организации Turbo Vision и возможности использования его в программах при создании простейших интерфейсных элементов. Если потребуется создать какие-либо специфичные элементы, следует более подробно рассмотреть объекты-предки создаваемых интерфейсных элементов [2] с тем, чтобы наиболее полно использовать возможности этих объектов и не создавать те параметры и методы, которые уже имеются у стандартных объектов. Неплохие варианты создания нестандартных интерфейсных элементов имеются в [3].

Примеры, рассмотренные в данном пособии, относительно небольшие, могут быть использованы своими частями как шаблоны для создания собственных объектов. Для этого можно взять нужный объект и наполнить его требуемым содержанием.

ПРИЛОЖЕНИЯ

A. Распределение стандартных объектов Turbo Vision по модулям

Объект	*Назначение*	*Модуль*
tApplication	Объект-приложение	App
tBackground	Фон рабочей области	App
tBufStream	Буферированный поток	Objects
tButton	Кнопка	Dialogs
tChDirDialog	Диал. окно изменения каталога	StdDlg
tCheckBoxes	Селективный список	Dialogs
tCluster	Список выбора	Dialogs
tCollection	Коллекция	Objects
tColorDialog	Диал. окно изменения палитры	Colorsel
tColorDisplay	Эл-т цветного отображения	Colorsel
tColorGroupList	Список цветовых групп	Colorsel
tColorItemList	Список строк в цветовой группе	Colorsel
tColorSelector	Селектор цветов фона и символов	Colorsel
tDeskTop	Рабочая область	App
tDialog	Диал. окно	Dialogs
tDirCollection	Коллекция параметров каталогов	StdDlg
tDirListBox	Дерево каталогов	StdDlg
tDosStream	Поток ДОС	Objects
tEditor	Редактор	Editors
tEditWindow	Окно редактора	Editors
tEmsStream	Поток расширенной области	Objects
tFileCollection	Отсорт. коллекция файлов	StdDlg
tFileDialog	Диал. окно имен файлов	StdDlg
tFileEditor	Редактор текстового файла	Editors
tFileInfoPane	Панель параметров файла	StdDlg
tFileInputLine	Строка ввода имени файла	StdDlg
tFileList	Отсорт. список файлов	StdDlg
tFilterValidator	Анализатор множества символов	Validate
tFrame	Рамка окна	Views
tGroup	Объект-группа	Views
tHistory	Протокол	Dialogs
tHistoryViewer	Список строк протокола	Dialogs
tHistoryWindow	Окно протокола	Dialogs
tIndicator	Индикатор	Editors

tInputLine	Строка ввода	Dialogs
tLabel	Метка	Dialogs
tListBox	Список коллекции строк	Dialogs
tListViewer	Список строк	Dialogs
tLookupValidator	Анализатор списка	Validate
tMemo	Объект для просмотра буфера	Editors
tMemoryStream	Поток в памяти	Objects
tMenuBar	Горизонтальное меню	Menus
tMenuBox	Вертикальное меню	Menus
tMenuPopup	Локальное меню	Menus
tMenuView	Меню	Menus
tMonoSelector	Монохромный селектор	ColorSel
tMultiCheckBoxes	Многостабильный селект. список	Dialogs
tObject	Тип-объект	Objects
tOutline	Эл-т отображения списка строк	Outline
tOutlineViewer	Эл-т отображения списка	Outline
tParamText	Параметрический текст	Dialogs
tPoint	Объект-точка	Objects
tProgram	Объект-программа	App
tPXPictureValidator	Анализатор по маске	Validate
tRadioButtons	Селективный список	Dialogs
tRangeValidator	Анализатор границ	Validate
tRect	Тип-прямоугольник	Objects
tResourceCollection	Коллекция ресурса	Objects
tResourceFile	Файл ресурса	Objects
tScrollBar	Линейка скроллинга	Views
tScroller	Поле скроллинга	Views
tSortedCollection	Отсорт. коллекция	Objects
tSortedListBox	Список отсорт. коллекции строк	StdDlg
tStaticText	Статический текст	Dialogs
tStatusLine	Строка состояния	Menus
tStream	Поток	Objects
tStrCollection	Коллекция ASCIIZ-строк	Objects
tStringCollection	Коллекция строк	Objects
tStringList	Список строк в потоке	Objects
tStrListMaker	Эл-т создания списка строк	Objects
tStringLookupValidator	Анализатор списка строк	Validate
tTerminal	Текст. эл-т для терминала	TextView
tTextDevice	Текстовый эл-т	TextView
tValidator	Анализатор	Validate
tView	Отображаемый объект	Views
tWindow	Окно	Views

Б. Стандартная цветовая палитра Turbo Vision

Вход	*Элемент*	*Совокупность цветов*	*Код*
		Рабочее поле	
1	Фон	Синий на св.-сером	$71
		Меню и строка состояния	
2	Норм. текст	Черный на св.-сером	$70
3	Текст запрещенного эл-та	Темно-серый на св.-сером	$78
4	Текст клавиш быстрого упр-я	Красный на св.-сером	$74
5	Выделенный норм. текст	Черный на зеленом	$20
6	Выделенный запрещен. эл-т	Темно-серый на зеленом	$28
7	Выделенный текст клавиш быстрого упр-я	Красный на зеленом	$24
		Окно с синей палитрой	
8	Рамка пассивного окна	Св.-серый на синем	$17
9	Рамка активного окна	Белый на синем	$1F
10	Пиктограмма	Св.-зеленый на синем	$1A
11	Линейка скрол-га	Синий на голубом	$31
12	Резерв линейки скрол-га	Синий на голубом	$31
13	Норм. текст поля скрол-га	Желтый на синем	$1E
14	Выделен. текст поля скрол-га	Синий на св.-сером	$71
15	Резерв окна	Черный	$00
		Окно с голубой палитрой	
16	Рамка пассивного окна	Св.-серый на голубом	$37
17	Рамка активного окна	Белый на голубом	$3F
18	Пиктограмма	Св.-зеленый на голубом	$1A
19	Линейка скрол-га	Голубой на синем	$13
20	Резерв линейки скрол-га	Голубой на синем	$13
21	Норм. текст поля скрол-га	Желтый на голубом	$3E
22	Выделен. текст поля скрол-га	Синий на зеленом	$21
23	Резерв окна	Черный	$00
		Окно с серой палитрой	
24	Рамка пассивного окна	Черный на св.-сером	$70
25	Рамка активного окна	Белый на св.-сером	$7F
26	Пиктограмма	Св.-зеленый на сером	$7A
27	Линейка скрол-га	Голубой на синем	$13
28	Резерв линейки скрол-га	Голубой на синем	$13
29	Норм. текст поля скрол-га	Черный на св.-сером	$70
30	Выделен. текст поля скрол-га	Белый на св.-сером	$7F
31	Резерв окна	Черный	$00

Диалоговое окно с серой палитрой

32	Рамка пассивного окна	Черный на св.-сером	$70
33	Рамка активного окна	Белый на св.-сером	$7F
34	Пиктограмма	Св.-зеленый на сером	$7A
35	Линейка скрол-га	Голубой на синем	$13
36	Эл-ты упр-я линейки скрол-га	Голубой на синем	$13
37	Статический текст	Черный на св.-сером	$70
38	Метка пассивная	Черный на св.-сером	$70
39	Метка выделенная	Белый на св.-сером	$7F
40	Текст клавиш быстрого упр-я метки	Желтый на св.-сером	$7E
41	Кнопка пассивная	Черный на зеленом	$20
42	Кнопка, выбираемая по умолчанию	Св.-голубой на зеленом	$2B
43	Кнопка выделенная	Белый на зеленом	$2F
44	Кнопка запрещенная	Темно-серый на св.-сером	$78
45	Текст клавиш быстрого упр-я кнопки	Желтый на зеленом	$2E
46	Тень кнопки	Черный на св.-сером	$70
47	Список пассивный	Черный на голубом	$30
48	Список выделенный	Белый на голубом	$3F
49	Текст клавиш быстрого упр-я списка	Желтый на голубом	$3E
50	Строка ввода нормальная	Белый на синем	$1F
51	Строка ввода выделенная	Белый на зеленом	$2F
52	Стрелки строки ввода	Св.-зеленый на синем	$1A
53	Стрелка пиктограммы протокола	Черный на зеленом	$20
54	Остальная часть пиктограммы протокола	Зеленый на темно-сером	$72
55	Линейка скрол-га окна протокола	Синий на голубом	$31
56	Эл-ты упр-я линейки скрол-га окна протокола	Синий на голубом	$31
57	Список строк пассивный	Черный на голубом	$30
58	Список строк сфокус-ый	Белый на зеленом	$2F
59	Список строк выделенный	Желтый на голубом	$3E
60	Разделитель списка строк	Синий на голубом	$31
61	Информационное поле	Голубой на синем	$13
62	Список запрещенный	Темно-серый на голубом	$38
63	Резерв диал. окна	Черный	$00

Диалоговое окно с синей палитрой

64	Рамка пассивного окна	Св.-серый на синем	$17
65	Рамка активного окна	Белый на синем	$1F
66	Пиктограмма	Св.-зеленый на синем	$1A
67	Линейка скрол-га	Синий на св.-сером	$71
68	Эл-ты упр-я линейки скрол-га	Синий на св.-сером	$71
69	Статический текст	Желтый на синем	$1E
70	Метка пассивная	Св.-серый на синем	$17
71	Метка выделенная	Белый на синем	$1F
72	Текст клавиш быстрого упр-я метки	Желтый на синем	$1E
73	Кнопка пассивная	Черный на зеленом	$20
74	Кнопка, выбираемая по умолчанию	Св.-голубой на зеленом	$2B
75	Кнопка выделенная	Белый на зеленом	$2F
76	Кнопка запрещенная	Темно-серый на св.сером	$78
77	Текст клавиш быстрого упр-я кнопки	Желтый на зеленом	$2E
78	Тень кнопки	Черный на синем	$10
79	Список пассивный	Черный на голубом	$30
80	Список выделенный	Белый на голубом	$3F
81	Текст клавиш быстрого упр-я списка	Желтый на голубом	$3E
82	Строка ввода нормальная	Черный на св.-сером	$70
83	Строка ввода выделенная	Белый на зеленом	$2F
84	Стрелки строки ввода	Св.-зеленый на св.-сером	$1A
85	Стрелка пиктограммы протокола	Черный на зеленом	$20
86	Остальная часть пиктограммы протокола	Зеленый на синем	$12
87	Линейка скрол-га окна протокола	Синий на голубом	$31
88	Эл-ты упр-я линейки скрол-га окна протокола	Синий на голубом	$31
89	Список строк пассивный	Черный на голубом	$30
90	Список строк сфокус-ый	Белый на зеленом	$2F
91	Список строк выделенный	Желтый на голубом	$3E
92	Разделитель списка строк	Синий на голубом	$31
93	Информационное поле	Голубой на синем	$13
94	Список запрещенный	Темно-серый на голубом	$38
95	Резерв диал. окна	Черный	$00

Диалоговое окно с голубой палитрой

96	Рамка пассивного окна	Св.-серый на голубом	$37
97	Рамка активного окна	Белый на голубом	$3F
98	Пиктограмма	Св.-зеленый на голубом	$3A
99	Линейка скрол-га	Голубой на синем	$13
100	Эл-ты упр-я линейки скрол-га	Голубой на синем	$13
101	Статический текст	Желтый на голубом	$3E
102	Метка пассивная	Черный на голубом	$30
103	Метка выделенная	Белый на голубом	$3F
104	Текст клавиш быстрого упр-я метки	Желтый на голубом	$3E
105	Кнопка пассивная	Черный на зеленом	$20
106	Кнопка, выбираемая по умолчанию	Св.-голубой на зеленом	$2B
107	Кнопка выделенная	Белый на зеленом	$2F
108	Кнопка запрещенная	Темно-серый на св.-сером	$78
109	Текст клавиш быстрого упр-я кнопки	Желтый на зеленом	$2E
110	Тень кнопки	Черный на голубом	$30
111	Список пассивный	Черный на св.-сером	$70
112	Список выделенный	Белый на св.-сером	$7F
113	Текст клавиш быстрого упр-я списка	Желтый на св.-сером	$7E
114	Строка ввода нормальная	Белый на синем	$1F
115	Строка ввода выделенная	Белый на зеленом	$2F
116	Стрелки строки ввода	Св.-зеленый на св.-сером	$1A
117	Стрелка пиктограммы протокола	Черный на зеленом	$20
118	Остальная часть пиктограммы протокола	Зеленый на голубом	$32
119	Линейка скрол-га окна протокола	Синий на голубом	$31
120	Эл-ты упр-я линейки скрол-га окна протокола	Синий на св.-сером	$71
121	Список строк пассивный	Черный на св.-сером	$70
122	Список строк сфокус-ый	Белый на зеленом	$2F
123	Список строк выделенный	Желтый на св.-сером	$7E
124	Разделитель списка строк	Синий на св.-сером	$71
125	Информационное поле	Голубой на синем	$13
126	Список запрещенный	Темно-серый на голубом	$38
127	Резерв диал. окна	Черный	$00

B. Перечень процедур регистрации стандартных объектов Turbo Vision

Процедура	*Модуль*	*Регистрируемые объекты*
RegisterApp	App	tBackGround tDeskTop
RegisterColorSel	ColorSel	tColorSelector tMonoSelector tColorDisplay tColorGroupList tColorItemList tColorDialog
RegisterDialogs	Dialogs	tDialog tInputLine tButton tCluster tRadioButtons tCheckBoxes tMultiCheckBoxes tListBox tStaticText tParamText tLabel tHistory
RegisterEditors	Editors	tEditor tMemo tFileEditor tIndicator tEditWindow
RegisterMenus	Menus	tMenuBar tMenuBox tStatusLine tMenuPopup
RegisterObjects	Objects	tCollection tStringCollection tStrCollection
RegisterOutline	Outline	tOutline
RegisterStdDlg	StdDlg	tFileInputLine tFileCollection tFileList tFileInfoPane tFileDialog

		tDirCollection
		tDirListBox
		tChDirDialog
RegisterValidate	Validate	tPXPictureValidator
		tFilterValidator
		tRangeValidator
		tLookupValidator
		tStringLookupValidator
RegisterViews	Views	tView
		tFrame
		tScrollBar
		tScroller
		tListViewer
		tGroup
		tWindow

Г. Текст программы-калькулятора

```
program Calculator;                                                    {1}
uses App, Dialogs, Drivers, HelpFile, Menus,
                MsgBox, Objects, Validate, Views;                      {2}

const                                                                  {3}

  BegStackDimension=6;          {Начальный размер стека}               {4}

  { Константы команд }
  cmParamrters =100;            {Параметры калькулятора}               {5}
  cmStack =101;                 {Содержимое стека}                     {6}
  cmCalcNew =1001;              {Открыть новый калькулятор}            {7}

  { Константа символов кнопок }
  KeySymb: array[0..26] of Char =
       'QRZ'#27'DEF'#29'ABC/789*456-123+0.N';                           {8}

type                                                                   {9}

  { Указатель на вещественное число типа Double }
  pDouble = ^Double;                                                   {10}

  { Тип состояния калькулятора }

  tCalcState = (csValid, csError);                                     {11}

  { Тип - строка состояния с подсказкой }
  pHintStatusLine = ^tHintStatusLine;                                  {12}
  tHintStatusLine = object(tStatusLine)                                {13}
    function Hint(AHelpCtx: Word): string; virtual;                    {14}
  end;                                                                 {15}

  { Тип - числовая коллекция }
  pNumberCollection = ^tNumberCollection;                              {16}
  tNumberCollection = object(tCollection)                              {17}
    procedure FreeItem(Item: Pointer); virtual;                        {18}
  end;                                                                 {19}
  { Объект - окно калькулятора }
  pCalcDisplay = ^tCalcDisplay;                                        { 20}
  tCalcDisplay = object(TView)                                         { 21}
```

```
    Status: tCalcState;      { Состояние калькулятора }    { 22}
    Number: string[15];      { Число без знака }           { 23}
    Sign: Char;              { Знак числа }                { 24}
    Stack: pNumberCollection; { Указатель на стек }        { 25}
    Sys: Char;               { Система счисления -    }    { 26}
                    { максимальная допустимая цифра }
    constructor Init(R: tRect);                            { 27}
    destructor Done; virtual;                              { 28}
    procedure CalcKey(Key: Char);                          { 29}
    procedure Clear;                                       { 30}
    procedure Draw; virtual;                               { 31}
    procedure Error;                                       { 32}
    procedure FreeStack;                                   { 33}
    function GetDisplay: Double;                           { 34}
    function GetPalette: PPalette; virtual;                { 35}
    function GetStack: Double;                             { 36}
    procedure HandleEvent(var Event: TEvent); virtual;     { 37}
    procedure LoadStack(D: Double);                        { 38}
    procedure NewNumber;                                   { 39}
    procedure NewStack(ALimit: Integer);                   { 40}
    procedure NewSystem(ASys: Char);                       { 41}
    procedure SetDisplay(D: Double;
              var Sig: Char; var Numb: string);            { 42}
  end;                                                     { 43}

  { Объект - калькулятор }
  pCalculator = ^tCalculator;                              { 44}
  tCalculator = object(tDialog)                            { 45}
     CalcDisplay: pCalcDisplay;
                       { Указатель на окно калькулятора } { 46}
     constructor Init(R: tRect);                           { 47}
     procedure EnableOrDisableCommands
        (ASys: Char);                                      { 48}
     procedure HandleEvent(var Event: TEvent); virtual;    { 49}
  end;                                                     { 50}

  { Анализатор границ числа }
  pMyRangeValidator = ^tMyRangeValidator;                  { 51}
  tMyRangeValidator = object(tRangeValidator)              { 52}
     procedure Error; virtual;                             { 53}
  end;                                                     { 54}
  { Диалоговое окно параметров }
```

```
  pParamDialog = ^tParamDialog;                                    { 55}
  tParamDialog = object(tDialog)                                   { 56}
     constructor Init;                                             { 57}
  end;                                                             { 58}

  { Поле скроллинга }
  pMyScroller = ^tMyScroller;                                      { 59}
  tMyScroller = object(tScroller)                                  { 60}
     Calculator: pCalculator;                                      { 61}
     constructor Init(var Bounds: tRect;
        AVScrollBar: pScrollBar; Calc: pCalculator);               { 62}
     procedure Draw; virtual;                                      { 63}
. end;                                                             { 64}

  { Окно стека }
  pStackWindow = ^tStackWindow;                                    { 65}
  tStackWindow = object(tWindow)                                   { 66}
     constructor Init(Calc: pCalculator);                          { 67}
     procedure HandleEvent(var Event: tEvent); virtual;            { 68}
  end;                                                             { 69}

  { Объект - приложение }
  pMyApp = ^tMyApp;                                                { 70}
  tMyApp = object(tApplication)                                    { 71}
     constructor Init;                                             { 72}
     procedure GetEvent(var Event: tEvent); virtual;               { 73}
     function GetPalette: pPalette; virtual;                       { 74}
     procedure HandleEvent(var Event: tEvent); virtual;            { 75}
     procedure InitMenuBar; virtual;                               { 76}
     procedure InitStatusLine; virtual;                            { 77}
  end;                                                             { 78}

  {*****************************************}
  {*          Объект tHintStatusLine       *}
  {*****************************************}

  function tHintStatusLine.Hint(AHelpCtx: Word): string;           { 79}
  begin                                                            { 80}
     case AHelpCtx of                                              { 81}
        1000: Hint:='Работа с калькулятором';                      { 82}
        1001: Hint:='Новый калькулятор';                           { 83}
        1002: Hint:='Закрыть калькулятор';                         { 84}
        1003: Hint:='Прекращение работы';                          { 85}
```

```
      1004: Hint:='Параметры калькулятора';                    { 86}
      1005: Hint:='Задание параметров';                        { 87}
      1006: Hint:='Параметры стека';                           { 88}
      1100: Hint:='Имя калькулятора';                          { 89}
      1101: Hint:='2-ная система счисления';                   { 90}
      1102: Hint:='8-ная система счисления';                   { 91}
      1103: Hint:='10-ная система счисления';                  { 92}
      1104: Hint:='16-ная система счисления';                  { 93}
      1105: Hint:='Стек';                                      { 94}
      1106: Hint:='Ввод параметров';                           { 95}
      1107: Hint:='Отмена';                                    { 96}
      1200: Hint:='Содержимое стека';                          { 97}
      2000: Hint:='Очистка стека';                             { 98}
      2001: Hint:='Удаление числа из стека';                   { 99}
      2002: Hint:='Очистка последнего числа';                  {100}
      2003: Hint:='Удаление символа из числа';                 {101}
      2004: Hint:='Цифра "D"';                                 {102}
      2005: Hint:='Цифра "E"';                                 {103}
      2006: Hint:='Цифра "F"';                                 {104}
      2007: Hint:='Изменение знака';                           {105}
      2008: Hint:='Цифра "A"';                                 {106}
      2009: Hint:='Цифра "B"';                                 {107}
      2010: Hint:='Цифра "C"';                                 {108}
      2011: Hint:='Знак деления';                              {109}
      2012: Hint:='Цифра "7"';                                 {110}
      2013: Hint:='Цифра "8"';                                 {111}
      2014: Hint:='Цифра "9"';                                 {112}
      2015: Hint:='Знак умножения';                            {113}
      2016: Hint:='Цифра "4"';                                 {114}
      2017: Hint:='Цифра "5"';                                 {115}
      2018: Hint:='Цифра "6"';                                 {116}
      2019: Hint:='Знак вычитания';                            {117}
      2020: Hint:='Цифра "1"';                                 {118}
      2021: Hint:='Цифра "2"';                                 {119}
      2022: Hint:='Цифра "3"';                                 {120}
      2023: Hint:='Знак сложения';                             {121}
      2024: Hint:='Цифра "0"';                                 {122}
      2025: Hint:='Десятичная точка';                          {123}
      2026: Hint:='Новый элемент стека';                       {124}
    else Hint:= '';                                            {125}
    end;                                                       {126}
end;                                                           {127}
```

```
{*****************************************}
{*          Объект tNumberCollection     *}
{*****************************************}

procedure tNumberCollection.FreeItem(Item: Pointer);          {128}
begin                                                         {129}
   if Item <> nil then                                        {130}
      Dispose(pDouble(Item));                                 {131}
end;                                                          {132}

{*****************************************}
{*          Объект tCalcDisplay          *}
{*****************************************}

constructor tCalcDisplay.Init(R: tRect);                      {133}
begin                                                         {134}
   inherited Init(R);                                         {135}
   Options:= Options and not ofSelectable or ofPreProcess;    {136}
   EventMask:= evKeyDown + evBroadcast;                       {137}
   Stack:= nil;                                               {138}
   NewStack(BegStackDimension);                               {139}
   NewSystem('9');                                            {140}
end;                                                          {141}

destructor tCalcDisplay.Done;                                 {142}
begin                                                         {143}
   Stack^.Done;                                               {144}
   inherited Done;                                            {145}
end;                                                          {146}
procedure tCalcDisplay.CalcKey(Key: Char);                    {147}
var                                                           {148}
   D, D1: Double;                                             {149}

begin                                                         {150}
   Key := UpCase(Key);                                        {151}
if (Status = csError) and (Key <> 'Q') and
                                  (Key <> 'R') then           {152}
   Key := ' ';                                                {153}

case Key of                                                   {154}
   { Цифры }
   '0'..'9', 'A'..'F':                                        {155}
      if Key in ['0'..Sys] then                               {156}
```

```
        begin                                                    {157}
          if Number = '0' then Number := '';                     {158}
          Number := Number + Key;                                {159}
          LoadStack(GetDisplay);                                 {160}
        end;                                                     {161}

    { Десятичная точка }
    '.':                                                         {162}
      if Pos('.', Number) = 0 then Number :=
                                          Number + '.';          {163}

    { Удаление символа }
    #8, #27:                                                     {164}
      begin                                                      {165}
        if Length(Number) = 1 then                               {166}
          Number := '0'                                          {167}
        else                                                     {168}
          Dec(Number[0]);                                        {169}
        LoadStack(GetDisplay);                                   {170}
      end;                                                       {171}
    { Изменение знака числа }
    '_', #29:                                                    {172}
      begin                                                      {173}
        if Sign = ' ' then                                       {174}
          Sign := '-'                                            {175}
        else                                                     {176}
          Sign := ' ';                                           {177}
        LoadStack(GetDisplay);                                   {178}
      end;                                                       {179}
    { Знаки операций }
    '+', '-', '*', '/':                                          {180}
      if Stack^.Count > 1 then                                   {181}
        begin                                                    {182}
          D := GetStack;                                         {183}
          D1:= pDouble(Stack^.At(Stack^.Count - 2))^;            {184}
          case Key of                                            {185}
            '+': D1:= D1 + D;                                    {186}
            '-': D1:= D1 - D;                                    {187}
            '*': D1:= D1 * D;                                    {188}
            '/': if D = 0 then Error else D1:=
                                              D1 / D;            {189}
        end;                                                     {190}
        if (Key <> '/') or (D <> 0) then                         {191}
```

```
                begin                                                    {192}
                   SetDisplay(D1, Sign, Number);                         {193}
                   FreeStack;                                            {194}
                   LoadStack(D1);                                        {195}
                end;                                                     {196}
          end;                                                           {197}

    { Очистка стека }
    'Q':                                                                 {198}
       begin                                                             {199}
          Stack^.FreeAll;                                                {200}
          NewNumber;                                                     {201}
       end;                                                              {202}
    { Удаление из стека последнего числа }
    'R':                                                                 {203}
       begin                                                             {204}
          FreeStack;                                                     {205}
          if Stack^.Count = 0 then                                       {206}
             NewNumber                                                   {207}
          else                                                           {208}
             SetDisplay(GetStack,Sign, Number);                          {209}
       end;                                                              {210}
    { Преобразование последнего числа в 0 }
    'Z':                                                                 {211}
       begin                                                             {212}
          Clear;                                                         {213}
          LoadStack(0.0);                                                {214}
       end;                                                              {215}
      { Открытие нового элемента стека }
      'N':                                                               {216}
         if Stack^.Count < Stack^.Limit then                             {217}
            NewNumber;                                                   {218}
   end;                                                                  {219}
   DrawView;                                                             {220}
end;                                                                     {221}
{ Задание символьного представления числа 0 }
procedure tCalcDisplay.Clear;                                            {222}
begin                                                                    {223}
   Status := csValid;                                                    {224}
   Number := '0';                                                        {225}
   Sign := ' ';                                                          {226}
end;                                                                     {227}
```

```
{ Отображение окна калькулятора }
procedure tCalcDisplay.Draw;                                    {228}
var                                                             {229}
   Color: Byte;                                                 {230}
   i: Integer;                                                  {231}
   B: TDrawBuffer;                                              {232}
begin                                                           {233}
   Color := GetColor(1);                                        {234}
   i := Size.X - Length(Number) - 2;                            {235}
   MoveChar(B, ' ', Color, Size.X);                             {236}
   MoveChar(B[i], Sign, Color, 1);                              {237}
   MoveStr(B[i + 1], Number, Color);                            {238}
   WriteBuf(0, 0, Size.X, 1, B);                                {239}
end;                                                            {240}

{ Обработка ошибки }
procedure tCalcDisplay.Error;                                   {241}
begin                                                           {242}
   Status := csError;                                           {243}
   Number := 'ERROR';                                           {244}
   Sign := ' ';                                                 {245}
end;                                                            {246}

{ Удаление последнего элемента стека }
procedure tCalcDisplay.FreeStack;                               {247}
begin                                                           {248}
   Stack^.AtFree(Stack^.Count - 1);                             {249}
end;                                                            {250}
{ Получение значения набранного числа }
function tCalcDisplay.GetDisplay: Double;                       {251}
var                                                             {252}
   D: Double;                                                   {253}
   Point: Boolean;                                              {254}
   i: Byte;                                                     {255}
   M: Double;                                                   {256}
   Sy: Double;                                                  {257}

   { Определение числового значения цифры }
   function Digit(N: Byte): Double;                             {258}
   var Num: Byte;                                               {259}
   begin                                                        {260}
      case Number[N] of                                         {261}
         '0'..'9': Num:= Ord(Number[N]) - Ord('0');             {262}
```

```
          'A'..'F': Num:= Ord(Number[N]) - Ord('A') + 10;   {263}
      end;                                                  {264}
      Digit:= Num;                                          {265}
   end;                                                     {266}

begin                                                       {267}
   { Основание системы счисления }
   case Sys of                                              {268}
      '1': Sy:= 2.0;                                        {269}
      '7': Sy:= 8.0;                                        {270}
      '9': Sy:= 10.0;                                       {271}
      'F': Sy:= 16.0;                                       {272}
   end;                                                     {273}

   { Начальное значение числа }
   if Number[1] = '.' then                                  {274}
      begin                                                 {275}
         D:= 0.0;                                           {276}
         Point:= True;                                      {277}
      end                                                   {278}
   else                                                     {279}
      begin                                                 {280}
         D:= Digit(1);                                      {281}
         Point:= False;                                     {282}
      end;                                                  {283}
   {Корректировка числа в соответствии с очередным символом}
   for i:= 2 to Length(Number) do                           {284}
      begin                                                 {285}
         if Number[i] = '.' then                            {286}
            begin                                           {287}
               Point:= True;                                {288}
               M:= 1 / Sy;                                  {289}
            end                                             {290}
         else if not Point then                             {291}
            D:= D * Sy + Digit(i)                           {292}
         else                                               {293}
            begin                                           {294}
               D:= D + Digit(i) * M;                        {295}
               M:= M / Sy;                                  {296}
            end;                                            {297}
      end;                                                  {298}
   { Учет знака }
   if Sign = '-' then                                       {299}
```

```
    D:= -D;                                                    {300}

  GetDisplay:= D;                                              {301}
end;                                                           {302}

{ Задание палитры }
function tCalcDisplay.GetPalette: PPalette;                    {303}
const                                                          {304}
  P: string[1] = #19;                                          {305}
begin                                                          {306}
  GetPalette := @P;                                            {307}
end;                                                           {308}

{ Получение значения последнего числа стека }
function tCalcDisplay.GetStack: Double;                        {309}
begin                                                          {310}
  GetStack:= pDouble(Stack^.At(Stack^.Count - 1))^             {311}
end;                                                           {312}

{ Обработка событий }
procedure tCalcDisplay.HandleEvent(var Event: TEvent);         {313}
begin                                                          {314}
  inherited HandleEvent(Event);                                {315}
  case Event.What of                                           {316}
  { Обработка нажатия клавиши }
  evKeyDown:                                                   {317}
    if (Event.KeyCode <> kbTab) and
        (Event.KeyCode <> kbShiftTab) then                     {318}
      begin                                                    {319}
        CalcKey(Event.CharCode);                               {320}
        ClearEvent(Event);                                     {321}
      end;                                                     {322}
  { Обработка сообщения }
  evBroadcast:                                                 {323}
    if Event.Command in [100,
      100 + Ord('0')..100 + Ord('9'),
      100 + Ord('A')..100 + Ord('F')] then                     {324}
      begin                                                    {325}
        CalcKey(PButton(Event.InfoPtr)^.Title^ [1]);           {326}
        ClearEvent(Event);                                     {327}
      end;                                                     {328}
  end;                                                         {329}
end;                                                           {330}
```

```
{ Задание последнего числа стека }
procedure tCalcDisplay.LoadStack(D: Double);                    {331}
begin                                                           {332}
   pDouble(Stack^.Items^[Stack^.Count - 1])^:= D;               {333}
end;                                                            {334}

{ Задание нового элемента стека }
procedure tCalcDisplay.NewNumber;                               {335}
var                                                             {336}
   PR: pDouble;                                                 {337}
begin                                                           {338}
   New(PR);                                                     {339}
   PR^:= 0.0;                                                   {340}
Stack^.AtInsert(Stack^.Count, PR);                              {341}
   Clear;                                                       {342}
end;                                                            {343}
{ Создание стека }
procedure tCalcDisplay.NewStack(ALimit: Integer);               {344}
begin                                                           {345}
   if Stack <> nil then                                         {346}
      Stack^.Done;                                              {347}
   Stack:= New(pNumberCollection, Init(ALimit, 0));             {348}
   NewNumber;                                                   {349}
end;                                                            {350}

{ Задание системы счисления }
procedure tCalcDisplay.NewSystem(ASys: Char);                   {351}
begin                                                           {352}
   Sys:= ASys;                                                  {353}
   SetDisplay(GetStack, Sign, Number);                          {354}
end;                                                            {355}

{ Задание символьного представления числа }
procedure tCalcDisplay.SetDisplay(
                D: Double; var Sig: Char; var Numb: string);    {356}
var                                                             {357}
   S: string[63];                                               {358}
   N: Byte;                                                     {359}
   Position: Integer;                                           {360}
   i: Integer;                                                  {361}
   Sy: Double;                                                  {362}

begin                                                           {363}
```

```
  { Основание системы счисления }
  case Sys of                                                    {364}
     '1': Sy:= 2.0;                                              {365}
     '7': Sy:= 8.0;                                              {366}
     '9': Sy:= 10.0;                                             {367}
     'F': Sy:= 16.0;                                             {368}
  end;                                                           {369}

  { Знак числа }
  if D < 0 then                                                  {370}
     begin                                                       {371}
        Sig:= '-';                                               {372}
        D:= -D;                                                  {373}
     end                                                         {374}
  else                                                           {375}
     Sig:= ' ';                                                  {376}
  { Нормализация числа }
  Position:= 1;                                                  {377}
  if D <> 0.0 then                                               {378}
     begin                                                       {379}
        while D >= Sy do                                         {380}
           begin                                                 {381}
              D:= D / Sy;                                        {382}
              Inc(Position);                                     {383}
           end;                                                  {384}
        while D < 1.0 do                                         {385}
           begin                                                 {386}
              D:= D * Sy;                                        {387}
              Dec(Position);                                     {388}
           end;                                                  {389}
     end;                                                        {390}

  { Выделение цифр }
  S:= '';                                                        {391}
  for i:= 1 to 16 do                                             {392}
     begin                                                       {393}
        N:= Round(Int(D));                                       {394}
        case N of                                                {395}
           0..9  : S:= S + Chr(Ord('0') + N);                    {396}
           10..15: S:= S + Chr(Ord('A') + N - 10);               {397}
        end;                                                     {398}
        D:= Frac(D) * Sy;                                        {399}
     end;                                                        {400}
```

```
{ Округление числа }
i:= 16;                                                    {401}
if S[i] > '5' then                                         {402}
   repeat                                                  {403}
      Dec(i);                                              {404}
      if S[i] = '9' then                                   {405}
         S[i]:= '0'                                        {406}
      else                                                 {407}
         Inc(S[i]);                                        {408}
      if (i = 1) and (S[i] = '0') then                     {409}
         begin                                             {410}
            Insert('1', S, 1);                             {411}
            Inc(Position);                                 {412}
         end;                                              {413}
   until S[i] <> '0';                                      {414}
{ Добавление десятичной точки }
if (Position > 15) then                                    {415}
   Error                                                   {416}
else if Position < -13 then                                {417}
   S:= '0'                                                 {418}
else if Position > 0 then                                  {419}
   Insert('.', S, Position + 1)                            {420}
else                                                       {421}
   begin                                                   {422}
      for i:= Position to -1 do                            {423}
         Insert('0', S, 1);                                {424}
      Insert('0.', S, 1);                                  {425}
   end;                                                    {426}

{ Удаление завершающих нулей }
for S[0]:= Chr(15) downto Chr(1) do                        {427}
   if (S[Ord(S[0])] <> '0') or (S[0] = Chr(1)) then        {428}
      Break;                                               {429}

{ Удаление десятичной точки }
if S[Length(S)] = '.' then                                 {430}
   Dec(S[0]);                                              {431}
if Status <> csError then                                  {432}
   Numb:= S;                                               {433}

end;                                                       {434}
```

```
{*****************************************}
{*            Объект tCalculator         *}
{*****************************************}

constructor tCalculator.Init(R: tRect);                    {435}
var i: Integer;                                            {436}
    P: pView;                                              {437}
    cmButton: Word;                                        {438}
begin                                                      {439}
  inherited Init(R, 'Калькулятор');                        {440}
  Options:= Options or ofFirstClick;                       {441}
  { Формирование кнопок }
  for i:= 0 to 26 do                                       {442}
    begin                                                  {443}
      R.A.X:= (i mod 4) * 5 + 2;                           {444}
      if i >= 25 then                                      {445}
        R.A.X:= R.A.X + 5;                                 {446}
      R.A.Y:= (i div 4) * 2 + 4;                           {447}
      if i <> 24 then                                      {448}
        R.B.X:= R.A.X + 5                                  {449}
      else                                                 {450}
        R.B.X:= R.A.X + 10;                                {451}
      R.B.Y:= R.A.Y + 2;                                   {452}
      if KeySymb[i] in ['0'..'9', 'A'..'F'] then           {453}
        cmButton:= 100 + Ord(KeySymb[i])                   {454}
      else                                                 {455}
        cmButton:= 100;                                    {456}

      P:= New(pButton, Init(R, KeySymb[i],
      cmButton, bfBroadcast + bfGrabFocus));               {457}

      P^.HelpCtx:= 2000 + i;                               {458}
      Insert(P);                                           {459}
    end;                                                   {460}

  { Формирование индикатора калькулятора }
  R.Assign(3, 2, 21, 3);                                   {461}
  CalcDisplay:= New(pCalcDisplay, Init(R));                {462}
  Insert(CalcDisplay);                                     {463}

  EnableOrDisableCommands('9');                            {464}
end;                                                       {465}
```

```
{ Разрешение или запрещение клавиш цифр }
procedure tCalculator.EnableOrDisableCommands(
                                        ASys: Char);                {466}
var i: Integer;                                                     {467}
begin                                                               {468}
  for i:= 4 to 22 do                                                {469}
    if KeySymb[i] in ['0'..'9', 'A'..'F'] then                      {470}
      if KeySymb[i] in ['0'..ASys] then                             {471}
        EnableCommands([100 + Ord(KeySymb[i])])                     {472}
      else                                                          {473}
        DisableCommands([100 + Ord(KeySymb[i])]);                   {474}
end;                                                                {475}
{ Обработка событий }
procedure tCalculator.HandleEvent(var Event: tEvent);               {476}
  { Задание параметров калькулятора }
  procedure Parameters;                                             {477}
  var ParamDialog: pParamDialog;                                    {478}
      Par: record                                                   {479}
        Name: string[15];                                           {480}
        Sys: Word;                                                  {481}
        Stack: string[3];                                           {482}
      end;                                                          {483}
      S: Char;                                                      {484}
      P: pCalculator;                                               {485}
      R: tRect;                                                     {486}
      St: Integer;                                                  {487}
      Code: Integer;                                                {488}
      D: Double;                                                    {489}
      i: Byte;                                                      {490}
  begin                                                             {491}
    { Создание окна диалога }
    ParamDialog:= New(pParamDialog, Init);                          {492}
    { Задание исходных параметров калькулятора }
    Par.Name:= Title^;                                              {493}
    case CalcDisplay^.Sys of                                        {494}
      '1': Par.Sys:= 0;                                             {495}
      '7': Par.Sys:= 1;                                             {496}
      '9': Par.Sys:= 2;                                             {497}
      'F': Par.Sys:= 3;                                             {498}
    end;                                                            {499}
    Str(CalcDisplay^.Stack^.Limit, Par.Stack);                      {500}
    { Работа с диалоговым окном }
    if Application^.ExecuteDialog(ParamDialog, @Par) <>
```

```
                                    cmCancel then                    {501}
    begin                                                            {502}
      { Новая система счисления }
      case Par.Sys of                                                {503}
        0: S:= '1';                                                  {504}
        1: S:= '7';                                                  {505}
        2: S:= '9';                                                  {506}
        3: S:= 'F';                                                  {507}
      end;                                                           {508}

      { Новый размер стека }
      Val(Par.Stack, St, Code);                                      {509}

      { Задание нового заголовка калькулятора }
      if Par.Name <> Title^ then                                     {510}
        begin                                                        {511}
          DisposeStr(Title);                                         {512}
          Title:= NewStr(Par.Name);                                  {513}
          Redraw;                                                    {514}
        end;                                                         {515}

      { Задание нового размера стека }
      if St<> CalcDisplay^.Stack^.Limit then                         {516}
      CalcDisplay^.NewStack(St);                                     {517}

      { Задание новой системы счисления }
      if S <> CalcDisplay^.Sys then                                  {518}
        begin                                                        {519}
          CalcDisplay^.NewSystem(S);                                 {520}

          EnableOrDisableCommands(S);                                {521}
        end;                                                         {522}

      { Перерисовка калькулятора }
      CalcDisplay^.DrawView;                                         {523}
    end;                                                             {524}
  end;                                                               {525}

  { Вывод содержимого стека }
  procedure StackContents;                                           {526}
  var StackWindow: pStackWindow;                                     {527}
  begin                                                              {528}
    StackWindow:= New(pStackWindow, Init(@Self));                    {529}
```

```
    if Application^.ValidView(StackWindow) <> nil then    {530}
      begin                                               {531}
        DeskTop^.ExecView(StackWindow);                   {532}
        StackWindow^.Free;                                {533}
      end;                                                {534}
  end;                                                    {535}

begin                                                     {536}
  inherited HandleEvent(Event);                           {537}

  case Event.What of                                      {538}
    evCommand:                                            {539}
      begin                                               {540}
        case Event.Command of                             {541}
          cmParamrters: Parameters;                       {542}
          cmStack: StackContents;                         {543}
        else                                              {544}
          Exit;                                           {545}
        end;                                              {546}
        ClearEvent(Event);                                {547}
      end;                                                {548}
  end;                                                    {549}
end;                                                      {550}

  {*************************************************}
  {*          Объект tMyRangeValidator             *}
  {*************************************************}

{ Обработка ошибки }
procedure tMyRangeValidator.Error;                        {551}
begin                                                     {552}
  MessageBox('Величина на в границах от 2 до 15',
                  nil, mfError + mfOKButton);             {553}
end;                                                      {554}

  {*************************************************}
  {*              Объект tParamDialog              *}
  {*************************************************}

constructor tParamDialog.Init;                            {555}
var R: tRect;                                             {556}
    P: pView;                                             {557}
```

```
begin                                                          {558}
  { Создание окна диалога }
  R.Assign(0, 0, 50, 15);                                      {559}
  inherited Init(R,'Параметры калькулятора');                  {560}
  Options:= Options or
      (ofCentered + ofSelectable + ofFirstClick);              {561}
  Flags:= wfMove + wfClose;                                    {562}
{ Формирование строки ввода названия калькулятора }
  R.Assign(15, 2, 33, 3);                                      {563}
  P:= New(pInputLine, Init(R, 15));                            {564}
  P^.HelpCtx:= 1100;                                           {565}
  Insert(P);                                                   {566}

  R.Assign(15, 1, 32, 2);                                      {567}
  Insert(New(pLabel, Init(R,'Название:',P)));                  {568}

  R.Assign(33, 2, 35, 3);                                      {569}
  Insert(New(pHistory, Init(R, pInputLine(P), 100)));          {570}

  { Формирование селективного списка системы счисления }
  R.Assign(15, 5, 35, 9);                                      {571}
  P:= New(pRadioButtons, Init(R,
    NewSItem('~2~-ичная',
    NewSItem('~8~-ричная',
    NewSItem('~1~0-ичная',
    NewSItem('1~6~-ричная',
    nil))))));                                                 {572}
  P^.HelpCtx:= 1101;                                           {573}
  Insert(P);                                                   {574}

  R.Assign(15, 4, 35, 5);                                      {575}
  Insert(New(pLabel, Init(R, 'Система счисления:', P)));       {576}

  { Формирование строки ввода размера стека }
  R.Assign(36, 10, 40, 11);                                    {577}
  P:= New(pInputLine, Init(R, 3));                             {578}
  P^.HelpCtx:= 1105;                                           {579}
  pInputLine(P)^.SetValidator(New(
              pMyRangeValidator, Init(2, 15)));                {580}
  Insert(P);                                                   {581}
  R.Assign(9, 10, 36, 11);                                     {582}
  Insert(New(pLabel, Init(
        R, 'Размер стека (от 2 до 15):', P)));                 {583}
```

```
  { Формирование кнопок управления }
  R.Assign(10, 12, 20, 14);                                    {584}
  P:= New(pButton, Init(R, 'Ввод', cmOK, bfDefault));          {585}
  P^.HelpCtx:= 1106;                                           {586}
  Insert(P);                                                   {587}
  Inc(R.A.X, 20);                                              {588}
  Inc(R.B.X, 20);                                              {589}
  P:= New(pButton, Init(
            R, 'Отмена', cmCancel, bfNormal));                 {590}
  P^.HelpCtx:= 1107;                                           {591}
  Insert(P);                                                   {592}
  SelectNext(False);                                           {593}
end;                                                           {594}

  {*****************************************}
  {*          Объект tMyScroller           *}
  {*****************************************}

constructor tMyScroller.Init(var Bounds: tRect;
     AVScrollBar: pScrollBar; Calc: pCalculator);              {595}
var LimitX, LimitY: Word;                                      {596}
begin                                                          {597}
  inherited Init(Bounds, nil, AVScrollBar);                    {598}
  GrowMode:= gfGrowHiX + gfGrowHiY;                            {599}
  LimitX:= Bounds.B.X - Bounds.A.X;                            {600}
  LimitY:= Calc^.CalcDisplay^.Stack^.Count;                    {601}
  SetLimit(LimitX, LimitY);                                    {602}
  Calculator:= Calc;                                           {603}
end;                                                           {604}

procedure tMyScroller.Draw;                                    {605}
var Color: Byte;                                               {606}
    i, j: Integer;                                             {607}
    Buf: tDrawBuffer;                                          {608}
    S: string[MaxViewWidth-1];                                 {609}
    Number: Double;                                            {610}
    Sign: Char;                                                {611}
    NumberStr: string[15];                                     {612}
begin                                                          {613}
  Color:= GetColor(1);                                         {614}
  for j:= 0 to Size.Y - 1 do                                   {615}
    begin                                                      {616}
      { "Цветная" строка из пробелов }
```

```
      MoveChar(Buf, ' ', Color, Size.X);                          {617}
      i:= Delta.Y + j;                                            {618}
      if i < Limit.Y then                                         {619}
         { Есть очередной элемент стека }
         begin                                                    {620}
            { Формирование "нецветной" строки }
            FillChar(S, Size.X, ' ');                             {621}
            { Порядковый номер }
            Str(i: 3, S);                                         {622}
            { Добавление пробелов }
            S[0]:= Chr(Size.X);                                   {623}
            { Получение символов числа и знака }
            Number:= pDouble(
               Calculator^.CalcDisplay^.Stack^.At(i))^;           {624}
            Calculator^.CalcDisplay^.SetDisplay(
                     Number, Sign, NumberStr);                    {625}
            Insert(Sign + NumberStr,
                  S, 25 - Length(NumberStr));                     {626}
            { Формирование "цветной" строки }
            MoveStr(Buf, S, Color);                               {627}
         end;                                                     {628}
      { Вывод строки в окно }
      WriteLine(0, j, Size.X, 1, Buf);                            {629}
   end;                                                           {630}
end;                                                              {631}

   {*****************************************}
   {*          Объект tStackWindow          *}
   {*****************************************}

constructor tStackWindow.Init(Calc: pCalculator);                 {632}
var R: tRect;                                                     {633}
    S: string[3];                                                 {634}
    ScrollBar: pScrollBar;                                        {635}
    Scroller: pMyScroller;                                        {636}
begin                                                             {637}
   { Формирование окна }
   R.Assign(0, 0, 50, 14);                                        {638}
   inherited Init(R, 'Стек', 1);                                  {639}
   Options:= Options or
      (ofCentered + ofSelectable + ofFirstClick);                 {640}
   Palette:= wpGrayWindow;                                        {641}
   HelpCtx:= 1200;                                                {642}
```

```
    { Формирование статического текста }
  R.Assign(9, 2, 41, 3);                                          {643}
  Str(pCalculator(DeskTop^.Current)^.
                  CalcDisplay^.Stack^.Count, S);                  {644}
  Insert(New(pStaticText,Init(
       R, ^C'Число заполненных элементов: ' + S)));               {645}

    { Формирование линейки скроллинга }
    R.Assign(45, 4, 46, 10);                                      {646}
    ScrollBar:= New(pScrollBar, Init(R));                         {647}
    ScrollBar^.Options:= ScrollBar^.Options or
                                      ofPostProcess;              {648}
    Insert(ScrollBar);                                            {649}

    { Формирование поля скроллинга }
    R.Assign(10, 4, 40, 10);                                      {650}
    Scroller:= New(pMyScroller, Init(R, ScrollBar, Calc));        {651}
    Scroller^.Options:= Scroller^.Options or ofFramed;            {652}
    Insert(Scroller);                                             {653}
end;                                                              {654}

procedure tStackWindow.HandleEvent(var Event: tEvent);           {655}
begin                                                             {656}
    if (Event.What = evCommand) and
                  (Event.Command = cmClose) then                  {657}

       { Снятие модальности окна }
       EndModal(cmClose);                                         {658}

       inherited HandleEvent(Event);                              {659}
end;                                                              {660}

    {*****************************************}
    {*              Объект tMyApp            *}
    {*****************************************}

constructor tMyApp.Init;                                          {661}
begin                                                             {662}
    inherited Init;                                               {663}
    RegisterHelpFile;                                             {664}
    DisableCommands([cmParamrters, cmStack]);                     {665}
end;                                                              {666}
```

```
{ Получение события }
procedure tMyApp.GetEvent(var Event: tEvent);                    {667}
var                                                              {668}
   W: pWindow;                                                   {669}
   HFile: pHelpFile;                                             {670}
   HelpStrm: pDosStream;                                         {671}
const                                                            {672}
   HelpInUse: Boolean = False;                                   {673}
   HelpFile: string[12] = 'HELPCALC.HLP';                        {674}
var                                                              {675}
   Par: record                                                   {676}
      PHelpF: Pointer;                                           {677}
   end;                                                          {678}
begin                                                            {679}
   inherited GetEvent(Event);                                    {680}
   case Event.What of                                            {681}
      evCommand:                                                 {682}
         if (Event.Command = cmHelp) and not HelpInUse
                                                then               {683}

         { Обработка запроса к справочной системе }
            begin                                                {684}
               HelpInUse:= True;                                 {685}
HelpStrm:= New(pDosStream,
   Init(HelpFile, stOpenRead));                                  {686}
HFile:= New(pHelpFile, Init(HelpStrm));                          {687}
if HelpStrm^.Status <> stOk then                                 {688}
   begin                                                         {689}
      Par.PHelpF:= @HelpFile;                                    {690}
      MessageBox(
         'Нельзя открыть файл контекстной помощи
         %s', @Par, mfError + mfOkButton);                       {691}
      HFile^.Free;                                               {692}
   end                                                           {693}
else                                                             {694}
   begin                                                         {695}
      W := New(pHelpWindow,Init(
               HFile, GetHelpCtx));                            {696}
   if ValidView(W) <> nil then                                   {697}
      begin                                                      {698}
         ExecView(W);                                            {699}
         W^.Free;                                                {700}
      end;                                                       {701}
```

```
                        ClearEvent(Event);                              {702}
                    end;                                                {703}
                HelpInUse := False;                                     {704}
            end;                                                        {705}
    end;                                                                {706}
end;                                                                    {707}

{ Получение палитры с учетом справочной системы }
function tMyApp.GetPalette: pPalette;                                   {708}
const                                                                   {709}
    CNewColor = CAppColor + CHelpColor;                                 {710}
    P: string[Length(CNewColor)] = CNewColor;                           {711}
begin                                                                   {712}
    GetPalette := @P;                                                   {713}
end;                                                                    {714}

{ Обработка событий }
procedure tMyApp.HandleEvent(var Event:
    tEvent);                                                            {715}

    { Создание нового калькулятора }
    procedure CalcNew;                                                  {716}
    var P: pCalculator;                                                 {717}
        R: tRect;                                                       {718}

    begin                                                               {719}
        if Desktop^.Current = nil then                                  {720}
            R.Assign(5, 1, 29, 20)                                      {721}
        else                                                            {722}
            R.Assign(Desktop^.Current^.Origin.X + 1,
                     Desktop^.Current^.Origin.Y + 1,
                     Desktop^.Current^.Origin.X + 25,
                     Desktop^.Current^.Origin.Y + 20);                  {723}
            P:= New(pCalculator, Init(R));                              {724}
            InsertWindow(P);                                            {725}
    end;                                                                {726}
begin                                                                   {727}
    inherited HandleEvent(Event);                                       {728}
    case Event.What of                                                  {729}
        evCommand:                                                      {730}
            begin                                                       {731}
                case Event.Command of                                   {732}
                    cmCalcNew: CalcNew;                                 {733}
```

```
            else                                                    {734}
               Exit;                                                {735}
            end;                                                    {736}
            ClearEvent(Event);                                      {737}
         end;                                                       {738}
end;                                                                {739}

   { Разрешение и запрет команд }
   if DeskTop^.Current = nil then                                   {740}
      DisableCommands([cmParamrters, cmStack])                      {741}
   else                                                             {742}
      EnableCommands([cmParamrters, cmStack]);                      {743}
end;                                                                {744}

{ Создание меню }
procedure tMyApp.InitMenuBar;                                       {745}
var R: tRect;                                                       {746}
begin                                                               {747}
   GetExtent(R);                                                    {748}
   R.B.Y:= R.A.Y + 1;                                               {749}
   MenuBar:= New(pMenuBar, Init(R, NewMenu(
      NewSubMenu('Калькулятор', 1000, NewMenu(
      NewItem('Новый', 'F4', kbF4, cmCalcNew, 1001,
      NewItem('Закрыть', 'F2', kbF2, cmClose, 1002,
      NewLine(NewItem('Выход', 'Alt-X', kbAltX,
      cmQuit,1003, nil))))),
      NewSubMenu('Параметры', 1004, NewMenu(
         NewItem('Параметры...', '', kbNoKey,
         cmParamrters, 1005,
         NewItem('Стек', '', kbNoKey, cmStack,
         1006, nil))), nil))))));                                   {750}
end;                                                                {751}

{ Создание строки состояния }
procedure tMyApp.InitStatusLine;                                    {752}
var R: tRect;                                                       {753}
begin                                                               {754}
   GetExtent(R);                                                    {755}
   R.A.Y:= R.B.Y - 1;                                               {756}
   StatusLine:= New(pHintStatusLine, Init(R,
      NewStatusDef(0, $FFFF,
         NewStatusKey('~Alt-X~ Выход', kbAltX, cmQuit,
         NewStatusKey('~F1~ Помощь', kbF1, cmHelp,
```

```
        NewStatusKey('~F10~ Меню', kbF10, cmMenu,
        nil))),
      nil)));                                                    {757}
end;                                                             {758}

var MyApp: tMyApp;                                               {759}

{ Основная часть программы }
begin                                                            {760}
   MyApp.Init;                                                   {761}
   MyApp.Run;                                                    {762}
   MyApp.Done;                                                   {763}
end.                                                             {764}
```

Д. Содержимое исходного файла фрагментов справочной системы

```
;       Исходный файл справочной системы программы
;       "Калькулятор"
;       Замечание: файл следует скомпилировать программой
;       TVHC.EXE.

.topic NoContext=0
```

Программа "Калькулятор"

Программа "Калькулятор" позволяет создать простейший калькулятор для элементарных вычислений (сложение, вычитание, умножение, деление). Она позволяет организовывать вычисления в различных системах счисления (двоичной, восьмеричной, десятичной шестнадцатеричной). Вычисления проводятся на основе бесскобочной записи вычислений. Калькулятор может иметь стек размером от 2 до 16 регистров. Числа представляются только в форме с фиксированной десятичной точкой.

```
.topic Calc=1000
```

Меню работы с калькуляторами

Меню позволяет создавать и удалять новые калькуляторы. При создании нового калькулятора задается десятичная система счисления, стек размером в шесть элементов и имя окна "Калькулятор", которые затем могут быть изменены.

```
.topic CalcNew
```

Создание нового калькулятора

Команда позволяет создать новый калькулятор. При создании нового калькулятора задается десятичная система счисления, стек размером в шесть элементов и имя окна "Калькулятор", которые затем могут быть изменены.

```
.topic CalcClose
```

Удаление калькулятора

Команда позволяет удалить активный калькулятор.

```
.topic CalcQuit
```

Завершение работы

Команда позволяет завершить работу с программой.

```
.topic Parameters
```

Параметры калькулятора

Меню позволяет задать или посмотреть некоторые параметры активного калькулятора (систему счисления, имя калькулятора, размер стека, содержимое стека).

```
.topic NewParam
```

Задание параметров калькулятора

Команда позволяет открыть диалоговое окно, с помощью которого можно задать ряд параметров активного калькулятора: систему счисления, имя калькулятора, размер стека. Первоначально (при создании калькулятора) задается десятичная система счисления, стек размером в 6 элементов и имя окна "Калькулятор". При изменении размера стека его содержимое пропадает, в противном случае - сохраняется.

```
.topic Stack
```

Параметры стека

Команда позволяет открыть окно с параметрами стека: размером и содержимым стека.

```
.topic CalcName=1100
```

Имя калькулятора

Строка ввода позволяет задать новое имя окна калькулятора. Имя может содержать до 16 символов.

```
.topic CalcSystem2,CalcSystem8,CalcSystem10,CalcSystem16
```

Система счисления

Селективный список позволяет задать систему счисления калькулятора (двоичную, восьмеричную, десятичную или шестнадцатеричную). Первоначально (при создании калькулятора)

задается десятичная система счисления. При задании новой системы счисления содержимое стека не изменяется, а текущее число (последнее число, помещенное в стек) отображается на экране в новой системе счисления. В зависимости от системы счисления разрешается или запрещается использование тех или иных кнопок ввода цифр.

```
.topic CalcStack
```

Размер стека

Строка ввода позволяет задать размер стека в пределах от 2 до 16. Первоначально (при создании калькулятора) задается размер стека в 6 элементов.

```
.topic CalcOK
```

Ввод параметров

Кнопка позволяет ввести в программу заданные в диалоговом окне параметры (систему счисления, имя калькулятора, размер стека).

```
.topic CalcCancel
```

Отмена

Кнопка позволяет отменить выделенные в диалоговом окне изменения параметров калькулятора (систему счисления, имя калькулятора, размер стека).

```
.topic StackList=1200
```

Содержимое стека

Команда позволяет открыть окно, в котором отображаются размер стека и его содержимое. Для каждого заполненного элемента стека указывается его порядковый номер (начиная с нуля) и помещенное число в соответствующей системе счисления. Чтобы закрыть окно, следует мышью нажать пиктограмму закрытия.

```
.topic QuiteN=2000
```

Кнопка удаления всех параметров

Кнопка "Q" (Quite) позволяет полностью очистить стек регистров калькулятора и настроиться на первый регистр стека.

```
.topic Remove
```

Кнопка удаления последнего параметра

Кнопка "R" (Remove) позволяет удалить последний параметр из стека и настроиться на предыдущий параметр. Если в стеке всего лишь один параметр, он принимает значение "0".

```
.topic Zero
```

Кнопка очистки последнего параметра

Кнопка "Z" (Zero) позволяет не удаляя последний параметр из стека превратить его в "0".

```
.topic Delete
```

Удаление цифры

Кнопка позволяет удалить в числе последнюю введенную цифру. Если эта цифра единственная, число обращается в "0".

```
.topic D
```

Цифра "D"

Кнопка позволяет ввести в число цифру "D" (13 - в шестнадцатеричной системе).

```
.topic E
```

Цифра "E"

Кнопка позволяет ввести в число цифру "E" (14 - в шестнадцатеричной системе).

```
.topic F
```

Цифра "F"

Кнопка позволяет ввести в число цифру "F" (15 - в шестнадцатеричной системе).

```
.topic Negative
```

Изменение знака

Кнопка позволяет изменить знак у числа на противоположный. Вместо кнопки можно нажать клавишу "_" (подчеркивание).

```
.topic A
```

Цифра "A"

Кнопка позволяет ввести в число цифру "A" (10 - в шестнадцатеричной системе).

```
.topic B
```

Цифра "B"

Кнопка позволяет ввести в число цифру "B" (11 - в шестнадцатеричной системе).

```
.topic C
```

Цифра "C"

Кнопка позволяет ввести в число цифру "C" (12 - в шестнадцатеричной системе).

```
.topic Divide
```

Деление

Кнопка позволяет выполнить операцию деления двух чисел (предпоследнее введенное число делится на последнее).

```
.topic 7
```

Цифра "7"

Кнопка позволяет ввести в число цифру "7".

```
.topic 8
```

Цифра "8"

Кнопка позволяет ввести в число цифру "8".

```
.topic 9
```

Цифра "9"

Кнопка позволяет ввести в число цифру "9".

```
.topic Multiply
```

Умножение

Кнопка позволяет выполнить операции умножения двух последних введенных чисел.

```
.topic 4
```

Цифра "4"

Кнопка позволяет ввести в число цифру "4".

```
.topic 5
```

Цифра "5"

Кнопка позволяет ввести в число цифру "5".

```
.topic 6
```

Цифра "6"

Кнопка позволяет ввести в число цифру "6".

```
.topic Subtract
```

Вычитание

Кнопка позволяет вычесть из предпоследнего введенного числа последнее число.

```
.topic 1
```

Цифра "1"

Кнопка позволяет ввести в число цифру "1".

```
.topic 2
```

Цифра "2"

Кнопка позволяет ввести в число цифру "2".

```
.topic 3
```

Цифра "3"

Кнопка позволяет ввести в число цифру "3".

```
.topic Add
```

Сложение

Кнопка позволяет выполнить операцию сложения двух последних введенных чисел.

```
.topic 0
```

Цифра "0"

Кнопка позволяет ввести в число цифру "0".

```
.topic Point
```

Десятичная точка

Кнопка позволяет ввести в число десятичную точку.

```
.topic New
```

Новое число

Кнопка позволяет перейти к вводу нового (New) числа в стек.

E. Модуль констант справочной системы

```
unit HELPCALC;

interface

const

  hc0                   = 2024;
  hc1                   = 2020;
  hc2                   = 2021;
  hc3                   = 2022;
  hc4                   = 2016;
  hc5                   = 2017;
  hc6                   = 2018;
  hc7                   = 2012;
  hc8                   = 2013;
  hc9                   = 2014;
  hcA                   = 2008;
  hcAdd                 = 2023;
  hcB                   = 2009;
  hcC                   = 2010;
  hcCalc                = 1000;
  hcCalcCancel          = 1107;
  hcCalcClose           = 1002;
  hcCalcName            = 1100;
  hcCalcNew             = 1001;
  hcCalcOK              = 1106;
  hcCalcQuit            = 1003;
  hcCalcStack           = 1105;
  hcCalcSystem10        = 1103;
  hcCalcSystem16        = 1104;
  hcCalcSystem2         = 1101;
  hcCalcSystem8         = 1102;
  hcD                   = 2004;
  hcDelete              = 2003;
  hcDivide              = 2011;
  hcE                   = 2005;
  hcF                   = 2006;
  hcMultiply            = 2015;
  hcNegative            = 2007;
  hcNew                 = 2026;
```

```
    hcNewParam                  = 1005;
    hcParameters                = 1004;
    hcPoint                     = 2025;
    hcQuiteN                    = 2000;
    hcRemove                    = 2001;
    hcStack                     = 1006;
    hcStackList                 = 1200;
    hcSubtract                  = 2019;
    hcZero                      = 2002;

implementation

end.
```

Ж. Текстовый редактор

```
program _Edit;                                                    {  1}

uses                                                              {  2}
   Dos, Objects, Drivers, Memory, Views, Menus,
   Dialogs, StdDlg, MsgBox, App, Editors;                         {  3}

const                                                             {  4}
   { 48 K динамической памяти для самой программы }
   HeapSize = 4096;                                               {  5}

   { Код команды отображения буфера }
   cmShowClip = 1012;                                             {  6}

type                                                              {  7}
   { Русифицированное окно редактора }
   pMyEditWindow = ^tMyEditWindow;                                {  8}
   tMyEditWindow = object(tEditWindow)                            {  9}
      function GetTitle(
         MaxSize: Integer): TTitleStr; virtual;                   { 10}
end;                                                              { 11}

   { Русифицированное диалоговое окно выбора файла }
   pMyFileDialog = ^tMyFileDialog;                                { 12}
   tMyFileDialog = object(tFileDialog)                            { 13}
      constructor Init(AWildCard: tWildStr; const ATitle,
      InputName: string; AOptions: Word; HistoryId: Byte); {  14}
   end;                                                           { 15}

{ Русифицированное диалоговое окно выбора каталога}
   pMyChDirDialog = ^tMyChDirDialog;                              { 16}
   tMyChDirDialog = object(tChDirDialog)                          { 17}
      constructor Init(AOptions: Word; HistoryId: Word);     {  18}
      procedure SetUpDialog;                                      { 19}
      function Valid(Command: Word): Boolean; virtual;       {  20}
   end;                                                           { 21}

   { Основной объект-редактор }
   pMyEditor = ^tMyEditor;                                        { 22}
   tMyEditor = object(tApplication)                               { 23}
      SetOfWindows: set of Byte;                                  { 24}
```

```
    constructor Init;                                              { 25}
    procedure FileOpen(WildCard: PathStr);                         { 26}
    function OpenEditor(FileName: FNameStr;
      Clip: Boolean; Visible: Boolean): pMyEditWindow;             { 27}
    procedure HandleEvent(var Event: tEvent); virtual;             { 28}
    procedure Idle; virtual;                                       { 29}
    procedure InitMenuBar; virtual;                                { 30}
    procedure InitStatusLine; virtual;                             { 31}
    procedure OutOfMemory; virtual;                                { 32}
  end;                                                             { 33}

var                                                                { 34}
  { Указатель на буфер редактора }
  ClipWindow: pMyEditWindow;                                       { 35}

  {*****************************************}
  {*     Подпрограммы общего назначения    *}
  {*****************************************}

{Создание окна диалога определения текста для поиска}
function CreateFindDialog: pDialog;                                { 36}
var                                                                { 37}
  D: pDialog;                                                      { 38}
  InputLine: pInputLine;                                           { 39}
  R: tRect;                                                        { 40}
begin                                                              { 41}
  { Создание диалогового окна }
  R.Assign(0, 0, 40, 12);                                          { 42}
  D:= New(PDialog, Init(R, 'Найти'));                              { 43}
  with D^ do                                                       { 44}
  begin                                                            { 45}
    Options:= Options or ofCentered;                               { 46}

    { Строка ввода текста для поиска }
    R.Assign(3, 3, 34, 4);                                         { 47}
    InputLine:= New(pInputLine, Init(R, 80));                      { 48}
    Insert(InputLine);                                             { 49}
    R.Assign(2, 2, 25, 3);                                         { 50}
    Insert(New(PLabel, Init(R,'Текст для поиска',
                  InputLine)));                                    { 51}
    R.Assign(34, 3, 37, 4);                                        { 52}
    Insert(New(PHistory, Init(R, InputLine, 10)));                 { 53}
```

```
    { Триггерный список задания условий поиска }
    R.Assign(3, 5, 37, 7);                                          { 54}
    Insert(New(pCheckBoxes, Init(R,
       NewSItem('Различие больш. и мал. букв',
       NewSItem('Только целое слово', nil))));                      { 55}

    { Задание кнопок }
    R.Assign(9, 9, 19, 11);                                         { 56}
    Insert(New(pButton, Init(
                R, 'Ввод', cmOk, bfDefault)));                      { 57}
       Inc(R.A.X, 12); Inc(R.B.X, 12);                              { 58}
       Insert(New(pButton, Init(
                R,'Отмена', cmCancel, bfNormal)));                  { 59}

       SelectNext(False);                                           { 60}
    end;                                                            { 61}
    CreateFindDialog:= D;                                           { 62}
end;                                                                { 63}

{ Определение параметров замены текста }
function CreateReplaceDialog: pDialog;                              { 64}
var                                                                 { 65}
    D: pDialog;                                                     { 66}
    InputLine: pInputLine;                                          { 67}
    R: tRect;                                                       { 68}
begin                                                               { 69}
    { Создание диалогового окна }
    R.Assign(0, 0, 40, 16);                                         { 70}
    D:= New(pDialog, Init(R, 'Замена'));                            { 71}
    with D^ do                                                      { 72}
    begin                                                           { 73}
       Options:= Options or ofCentered;                             { 74}

       { Строка ввода текста для поиска }
       R.Assign(3, 3, 34, 4);                                       { 75}
       InputLine:= New(pInputLine, Init(R, 80));                    { 76}
       Insert(InputLine);                                           { 77}
       R.Assign(2, 2, 25, 3);                                       { 78}
       Insert(New(pLabel, Init(
            R,'Текст для поиска', InputLine)));                     { 79}
       R.Assign(34, 3, 37, 4);                                      { 80}
       Insert(New(pHistory, Init(R, InputLine, 10)));               { 81}
```

```
    { Строка ввода текста для замены }
    R.Assign(3, 6, 34, 7);                                    { 82}
    InputLine:= New(pInputLine, Init(R, 80));                 { 83}
    Insert(InputLine);                                        { 84}
    R.Assign(2, 5, 22, 6);                                    { 85}
    Insert(New(pLabel, Init(
              R, 'Новый текст', InputLine)));                 { 86}
    R.Assign(34, 6, 37, 7);                                   { 87}
    Insert(New(pHistory, Init(R, InputLine, 11)));            { 88}

{ Триггерный список задания условий поиска и замены }
    R.Assign(3, 8, 37, 12);                                   { 89}
    Insert(New(pCheckBoxes, Init(R,
      NewSItem('Различие больш. и мал. букв',
      NewSItem('Только целое слово',
      NewSItem('Подтверждение на замену',
      NewSItem('Заменить все вхождения', nil))))));           { 90}

    { Задание кнопок }
    R.Assign(9, 13, 19, 15);                                  { 91}
    Insert(New(pButton, Init(
              R, 'Ввод', cmOk, bfDefault)));                  { 92}
    Inc(R.A.X, 12); Inc(R.B.X, 12);                           { 93}
    Insert(New(pButton, Init(
             R,'Отмена', cmCancel, bfNormal)));               { 94}

    SelectNext(False);                                        { 95}
  end;                                                        { 96}
  CreateReplaceDialog:= D;                                    { 97}
end;                                                          { 98}

{ Русифицированная функция формирования сообщения }
function MyMessageBoxRect(var R: tRect;
        const Msg: string; Params: Pointer;
                           AOptions: Word): Word;             { 99}
const                                                         {100}
  ButtonName: array[0..3] of string[6] =
    ('Да', 'Нет', 'Ввод', 'Отмена');                          {101}
  Commands: array[0..3] of Word =
    (cmYes, cmNo, cmOK, cmCancel);                            {102}
  Titles: array[0..3] of string[11] =
    ('Предупрежд.', 'Ошибка', 'Информация',
                              'Подтвержд.');                  {103}
```

```
var                                                              {104}
   I, X: Integer;                                                {105}
   Dialog: pDialog;                                              {106}
   Control: pView;                                               {107}
   S: string;                                                    {108}
begin                                                            {109}
   Dialog:= New(pDialog, Init(
                     R, Titles[AOptions and $3]));               {110}
with Dialog^ do                                                  {111}
begin                                                            {112}
   Options:= Options or ofCentered;                              {113}

   R.Assign(3, 2, Size.X - 2, Size.Y - 3);                       {114}
   FormatStr(S, Msg, Params^);                                   {115}
   Insert(New(pStaticText, Init(R, S)));                         {116}
   X:= -2;                                                       {117}
   R.Assign(0, 0, 10, 2);                                        {118}
   for I:= 0 to 3 do                                             {119}
      if AOptions and ($0100 shl I) <> 0 then                    {120}
         Inc(X, R.B.X - R.A.X + 2);                              {121}
   X:= (Size.X - X) shr 1;                                       {122}
   for I:= 0 to 3 do                                             {123}
      if AOptions and ($0100 shl I) <> 0 then                    {124}
         begin                                                   {125}
            Control:= New(pButton, Init(
               R, ButtonName[I], Commands[i],
               bfNormal));                                       {126}
            Insert(Control);                                     {127}
            Control^.MoveTo(X, Size.Y - 3);                      {128}
            Inc(X, Control^.Size.X + 2);                         {129}
         end;                                                    {130}
   SelectNext(False);                                            {131}
end;                                                             {132}

   if AOptions and mfInsertInApp = 0 then                        {133}
      MyMessageBoxRect:=
      DeskTop^.ExecView(Dialog)                                  {134}
   else                                                          {135}
      MyMessageBoxRect:=
      Application^.ExecView(Dialog);                             {136}
   Dispose(Dialog, Done);                                        {137}
end;                                                             {138}
```

```
{ Русифицированная функция формирования сообщения
                              стандартного размера }
function MyMessageBox(const Msg: String;
      Params: Pointer; AOptions: Word): Word;                    {139}
var                                                              {140}
   R: TRect;                                                     {141}
begin                                                            {142}
   R.Assign(0, 0, 40, 9);                                        {143}
   MyMessageBox:= MyMessageBoxRect(R,
                     Msg, Params, AOptions);                     {144}
end;                                                             {145}

{ Организация диалога редактора }
function MyStdEditorDialog(Dialog: Integer;
                  Info: Pointer): Word; far;                     {146}
var                                                              {147}
   R: tRect;                                                     {148}
   T: tPoint;                                                    {149}
begin                                                            {150}
   case Dialog of                                                {151}
      edOutOfMemory:                                             {152}
         MyStdEditorDialog:= MyMessageBox(
            'Нет достаточной памяти для этой операции.',
            nil, mfError + mfOkButton);                          {153}
   edReadError:                                                  {154}
      MyStdEditorDialog:= MyMessageBox(
            'Ошибка в чтении файла %s.',
            @Info, mfError + mfOkButton);                        {155}
   edWriteError:                                                 {156}
      MyStdEditorDialog:= MyMessageBox(
            'Ошибка в записи файла %s.',
            @Info, mfError + mfOkButton);                        {157}
   edCreateError:                                                {158}
      MyStdEditorDialog:= MyMessageBox(
            'Ошибка в создании файла %s.',
            @Info, mfError + mfOkButton);                        {159}
   edSaveModify:                                                 {160}
      MyStdEditorDialog:= MyMessageBox(
         '%s был изменен. Сохранить?',
      @Info, mfInformation + mfYesNoCancel);                     {161}
   edSaveUntitled:                                               {162}
      MyStdEditorDialog:= MyMessageBox(
            'Сохранить файл без имени?',
```

```
          nil, mfInformation + mfYesNoCancel);                    {163}
    edSaveAs:                                                     {164}
      MyStdEditorDialog:=
            Application^.ExecuteDialog(
            New(pMyFileDialog, Init('*.*',
            'Сохранить файл с именем', 'Имя',
            fdOkButton, 101)), Info);                             {165}
    edFind:                                                       {166}
MyStdEditorDialog:= Application^.ExecuteDialog(
            CreateFindDialog, Info);                              {167}
    edSearchFailed:                                               {168}
      MyStdEditorDialog:= MyMessageBox(
            'Искомой информации не найдено.',
            nil, mfError + mfOkButton);                           {169}
    edReplace:                                                    {170}
      MyStdEditorDialog:=
            Application^.ExecuteDialog(
            CreateReplaceDialog, Info);                           {171}
    edReplacePrompt:                                              {172}
      begin                                                       {173}
        R.Assign(0, 1, 40, 8);                                    {174}
        R.Move((Desktop^.Size.X - R.B.X) div 2, 0);               {175}
        Desktop^.MakeGlobal(R.B, T);                              {176}
        Inc(T.Y);                                                 {177}
        if TPoint(Info).Y <= T.Y then                             {178}
          R.Move(0, Desktop^.Size.Y - R.B.Y - 2);                 {179}
        MyStdEditorDialog:= MyMessageBoxRect(
        R, 'Заменить это вхождение?',
        nil, mfYesNoCancel + mfInformation);                      {180}
      end;                                                        {181}
  end;                                                            {182}
end;                                                              {183}

{ Получение строки с названием текущего каталога }
function GetCurDir: DirStr;                                       {184}
var                                                               {185}
  CurDir: DirStr;                                                 {186}
begin                                                             {187}
  GetDir(0, CurDir);                                              {188}
  if Length(CurDir) > 3 then                                      {189}
    begin                                                         {190}
      Inc(CurDir[0]);                                             {191}
      CurDir[Length(CurDir)]:= '\';                               {192}
```

```
      end;                                                          {193}
   GetCurDir:= CurDir;                                              {194}
end;                                                                {195}

{ Русифицированное стандартное меню работы с файлами}
function MyStdFileMenuItems(Next: pMenuItem): pMenuItem;            {196}
begin                                                               {197}
   MyStdFileMenuItems:=
      NewItem('Новый', '', kbNoKey, cmNew, hcNew,
      NewItem('Открыть...', 'F3', kbF3, cmOpen, hcOpen,
      NewItem('Сохранить', 'F2', kbF2, cmSave, hcSave,
      NewItem('Сохранить с именем...', '',
                        kbNoKey, cmSaveAs, hcSaveAs,
      NewItem('Сохранить все', '', kbNoKey,
                        cmSaveAll, hcSaveAll,
      NewLine(
      NewItem('Сменить каталог...', '',
                  kbNoKey, cmChangeDir, hcChangeDir,
      NewItem('Выход в ДОС', '', kbNoKey,
                        cmDosShell, hcDosShell,
      NewItem('Завершение', 'Alt+X', kbAltX, cmQuit, hcExit,
      Next)))))))));                                                {198}
end;                                                                {199}

{ Русифицированное стандартное меню работы с редактором }
function MyStdEditMenuItems(Next: pMenuItem): pMenuItem;            {200}
begin                                                               {201}
   MyStdEditMenuItems:=
      NewItem('Отменить', '', kbAltBack, cmUndo, hcUndo,
      NewLine(
      NewItem('Скопировать с удал.', 'Shift+Del',
                        kbShiftDel, cmCut, hcCut,
      NewItem('Скопировать', 'Ctrl+Ins',
                        kbCtrlIns, cmCopy, hcCopy,
      NewItem('Разместить', 'Shift+Ins',
                        kbShiftIns, cmPaste, hcPaste,
      NewItem('Удалить', 'Ctrl+Del',
                        kbCtrlDel, cmClear, hcClear,
      Next))))));                                                   {202}
end;                                                                {203}

{ Русифицированное стандартное меню работы с окнами }
function MyStdWindowMenuItems(Next: pMenuItem): pMenuItem; {204}
```

```
begin                                                          {205}
   MyStdWindowMenuItems:=
      NewItem('Размещение встык', '', kbNoKey,
                        cmTile, hcTile,
      NewItem('Каскадное размещение', '',
                        kbNoKey, cmCascade, hcCascade,
      NewItem('Закрыть все окна', '',
                        kbNoKey, cmCloseAll, hcCloseAll,
      NewLine(
      NewItem('Размеры/Перемещение','Ctrl+F5',
                        kbCtrlF5, cmResize, hcResize,
      NewItem('Распахнуть', 'F5', kbF5, cmZoom, hcZoom,
      NewItem('Следующее', 'F6', kbF6, cmNext, hcNext,
      NewItem('Предыдущее', 'Shift+F6',
                        kbShiftF6, cmPrev, hcPrev,
      NewItem('Закрыть', 'Alt+F3', kbAltF3, cmClose, hcClose,
      Next)))))))));                                           {206}
end;                                                           {207}
   {*****************************************}
   {*          Объект tMyEditWindow         *}
   {*****************************************}

function tMyEditWindow.GetTitle(
            MaxSize: Integer): tTitleStr;                      {208}
begin                                                          {209}
   if Clipboard = pEditor(Editor) then                         {210}
      GetTitle := 'Буфер'                                      {211}
   else if Editor^.FileName = '' then                          {212}
      GetTitle := 'Без имени'                                  {213}
   else                                                        {214}
      GetTitle := Editor^.FileName;                            {215}
end;                                                           {216}

   {*****************************************}
   {*          Объект tMyFileDialog         *}
   {*****************************************}

constructor tMyFileDialog.Init(AWildCard: tWildStr;
               const ATitle, InputName: string;
               AOptions: Word; HistoryId: Byte);               {217}
var                                                            {218}
   ScrollBar: pScrollBar;                                      {219}
   R: tRect;                                                   {220}
```

```
begin                                                          {221}
   { Создание окна диалога }
   R.Assign(15, 1, 64, 20);                                    {222}
   tDialog.Init(R, ATitle);                                    {223}
   Options:= Options or ofCentered;                            {224}
   WildCard:= AWildCard;                                       {225}

   { Строка ввода имени файла }
   R.Assign(3, 3, 31, 4);                                      {226}
   FileName:= New(pFileInputLine, Init(R,79));                 {227}
   FileName^.Data^:= WildCard;                                 {228}
   Insert(FileName);                                           {229}
   R.Assign(2, 2, 6, 3);                                       {230}
   Insert(New(pLabel, Init(R, InputName, FileName)));          {231}
   R.Assign(31, 3, 34, 4);                                     {232}
   Insert(New(pHistory, Init(R, FileName, HistoryId)));        {233}
   { Линейка скроллинга и список файлов }
   R.Assign(3, 14, 34, 15);                                    {234}
   ScrollBar:= New(pScrollBar, Init(R));                       {235}
   Insert(ScrollBar);                                          {236}
   R.Assign(3, 6, 34, 14);                                     {237}
   FileList:= New(pFileList, Init(R, ScrollBar));              {238}
   Insert(FileList);                                           {239}
   R.Assign(2, 5, 8, 6);                                       {240}
   Insert(New(pLabel, Init(R, 'Файлы', FileList)));            {241}

   { Задание кнопок }
   R.Assign(35, 3, 46, 5);                                     {242}
   if AOptions and fdOpenButton <> 0 then                      {243}
      begin                                                    {244}
         Insert(New(pButton, Init(
               R, 'Открыть', cmFileOpen, bfDefault)));         {245}
         Inc(R.A.Y,3); Inc(R.B.Y,3);                           {246}
      end;                                                     {247}
   if AOptions and fdOkButton <> 0 then                        {248}
      begin                                                    {249}
         Insert(New(pButton, Init(
               R, 'Ввод', cmFileOpen,bfNormal)));              {250}
         Inc(R.A.Y,3); Inc(R.B.Y,3);                           {251}
      end;                                                     {252}
   if AOptions and fdReplaceButton <> 0 then                   {253}
      begin                                                    {254}
         Insert(New(pButton, Init(
```

```
            R, 'Замена',cmFileReplace, bfNormal)));      {255}
      Inc(R.A.Y,3); Inc(R.B.Y,3);                        {256}
    end;                                                 {257}
  if AOptions and fdClearButton <> 0 then                {258}
    begin                                                {259}
      Insert(New(pButton, Init(
            R,'Удал.',cmFileClear,bfNormal)));           {260}
      Inc(R.A.Y,3); Inc(R.B.Y,3);                        {261}
    end;                                                 {262}
  Insert(New(pButton, Init(
            R, 'Отмена', cmCancel,bfNormal)));           {263}
  if AOptions and fdHelpButton <> 0 then                 {264}
    begin                                                {265}
      Insert(New(pButton, Init(
            R, 'Помощь', cmHelp, bfNormal)));            {266}
      Inc(R.A.Y,3); Inc(R.B.Y,3);                        {267}
    end;                                                 {268}

  { Информационная панель с параметрами файла }
  R.Assign(1, 16, 48, 18);                               {269}
  Insert(New(pFileInfoPane, Init(R)));                   {270}

  SelectNext(False);                                     {271}

  { Загрузка каталога }
  if AOptions and fdNoLoadDir = 0 then                   {272}
    begin                                                {273}
      FileList^.ReadDirectory(WildCard);                 {274}
      Directory:= NewStr(GetCurDir);                     {275}
    end;                                                 {276}
end;                                                     {277}

  {*****************************************}
  {*          Объект tMyChDirDialog        *}
  {*****************************************}

constructor tMyChDirDialog.Init(AOptions: Word;
                                HistoryId: Word);        {278}
var                                                      {279}
  R: tRect;                                              {280}
  ScrollBar: pScrollBar;                                 {281}
  CurDir: DirStr;                                        {282}
begin                                                    {283}
```

```
{ Создание окна }
R.Assign(16, 2, 64, 20);                                        {284}
tDialog.Init(R, 'Изменить каталог');                            {285}

Options := Options or ofCentered;                               {286}

{ Строка ввода имени каталога }
R.Assign(3, 3, 30, 4);                                          {287}
DirInput := New(pInputLine, Init(R, 68));                       {288}
Insert(DirInput);                                               {289}
R.Assign(2, 2, 17, 3);                                          {290}
Insert(New(pLabel, Init(
          R, '~И~мя каталога', DirInput)));                     {291}
R.Assign(30, 3, 33, 4);                                         {292}
Insert(New(pHistory, Init(R, DirInput, HistoryId)));            {293}

{ Список каталогов }
R.Assign(32, 6, 33, 16);                                        {294}
ScrollBar := New(pScrollBar, Init(R));                          {295}
Insert(ScrollBar);                                              {296}
R.Assign(3, 6, 32, 16);                                         {297}
DirList := New(pDirListBox, Init(R, ScrollBar));                {298}
Insert(DirList);                                                {299}
R.Assign(2, 5, 19, 6);                                          {300}
Insert(New(pLabel, Init(
        R, '~Д~ерево каталогов', DirList)));                    {301}

{ Формирование кнопок }
R.Assign(35, 6, 45, 8);                                         {302}
OkButton := New(pButton, Init(
           R, '~В~вод', cmOK, bfDefault));                      {303}
Insert(OkButton);                                               {304}
Inc(R.A.Y,3); Inc(R.B.Y,3);                                     {305}
ChDirButton := New(pButton, Init(
     R, '~С~мена', cmChangeDir, bfNormal));                     {306}
Insert(ChDirButton);                                            {307}
Inc(R.A.Y,3); Inc(R.B.Y,3);                                     {308}
Insert(New(pButton, Init(
     R, 'В~о~зврат', cmRevert, bfNormal)));                     {309}
if AOptions and cdHelpButton <> 0 then                          {310}
   begin                                                        {311}
      Inc(R.A.Y,3); Inc(R.B.Y,3);                               {312}
      Insert(New(pButton, Init(
```

```
                R, 'Помощь', cmHelp, bfNormal));                    {313}
      end;                                                          {314}

   if AOptions and cdNoLoadDir = 0 then SetUpDialog;                {315}

   SelectNext(False);                                               {316}
end;                                                                {317}

procedure tMyChDirDialog.SetUpDialog;                               {318}
var                                                                 {319}
   CurDir: DirStr;                                                  {320}
begin                                                               {321}
   if DirList <> nil then                                           {322}
      begin                                                         {323}
         CurDir := GetCurDir;                                       {324}
         DirList^.NewDirectory(CurDir);                             {325}
         if (Length(CurDir) > 3) and
            (CurDir[Length(CurDir)] = '\') then                     {326}
            CurDir := Copy(CurDir,1, Length(CurDir)-1);             {327}
         if DirInput <> nil then                                    {328}
            begin                                                   {329}
               DirInput^.Data^ := CurDir;                           {330}
               DirInput^.DrawView;                                  {331}
            end;                                                    {332}
      end;                                                          {333}
end;                                                                {334}

function tMyChDirDialog.Valid(Command: Word): Boolean;              {335}
var                                                                 {336}
   P: PathStr;                                                      {337}
begin                                                               {338}
   Valid := True;                                                   {339}
   if Command = cmOk then                                           {340}
      begin                                                         {341}
         P := FExpand(DirInput^.Data^);                             {342}
         if (Length(P) > 3) and (P[Length(P)] = '\') then           {343}
            Dec(P[0]);                                              {344}
      {$I-}
      ChDir(P);                                                     {345}
      if IOResult <> 0 then                                         {346}
         begin                                                      {347}
            MyMessageBox('Неправильный каталог.',
                  nil, mfError + mfOkButton);                       {348}
```

```
                Valid := False;                                    {349}
              end;                                                 {350}
            {$I+}
        end;                                                       {351}
end;                                                               {352}

{*****************************************}
{*              Объект TMyEditor         *}
{*****************************************}

constructor tMyEditor.Init;                                        {353}
var                                                                {354}
   R: tRect;                                                       {355}
   I: Integer;                                                     {356}
   FileName: PathStr;                                              {357}
begin                                                              {358}
   MaxHeapSize:= HeapSize;                                         {359}
   inherited Init;                                                 {360}

   DisableCommands([cmSave, cmSaveAs, cmCut,
      cmCopy, cmPaste, cmClear, cmUndo, cmFind,
      cmReplace, cmSearchAgain, cmCloseAll]);                      {361}
EditorDialog:= MyStdEditorDialog;                                  {362}
ClipWindow:= OpenEditor('', True, False);                          {363}
if ClipWindow <> nil then                                          {364}
   begin                                                           {365}
      Clipboard:= ClipWindow^.Editor;                              {366}
      Clipboard^.CanUndo:= False;                                  {367}
   end;                                                            {368}

for I:= 1 to ParamCount do                                         {369}
   begin                                                           {370}
      FileName:= ParamStr(I);                                      {371}
      if FileName[Length(FileName)] = '\' then                     {372}
         FileName:= FileName + '*.*';                              {373}
      if (Pos('?', FileName) = 0) and
            (Pos('*', FileName) = 0) then                          {374}
      OpenEditor(FExpand(FileName), True, True)                    {375}
      else FileOpen(FileName);                                     {376}
     end;                                                          {377}

   SetOfWindows:= [];                                              {378}
end;                                                               {379}
```

```
   { Создание редактора }
function tMyEditor.OpenEditor(FileName: FNameStr;
     Clip: Boolean; Visible: Boolean): pMyEditWindow;      {380}
var                                                        {381}
   P: pView;                                               {382}
   R: tRect;                                               {383}
   Number: Byte;                                           {384}
begin                                                      {385}

   { Задание размеров окна редактора }
   if DeskTop^.Current = nil then                          {386}
      DeskTop^.GetExtent(R)                                {387}
   else                                                    {388}
      begin                                                {389}
         R.A.X:= DeskTop^.Current^.Origin.X;               {390}
         R.B.X:=R.A.X+DeskTop^.Current^.Size.X;            {391}
         if DeskTop^.Current^.Size.X > 16 then             {392}
            Inc(R.A.X);                                    {393}
         R.A.Y:= DeskTop^.Current^.Origin.Y;               {394}
         R.B.Y:= R.A.Y + DeskTop^.Current^.Size.Y;         {395}
         if DeskTop^.Current^.Size.Y > 6 then              {396}
            Inc(R.A.Y);                                    {397}
      end;                                                 {398}

   { Задание номера окна }
   if Clip then                                            {399}
      Number:= wnNoNumber                                  {400}
   else                                                    {401}
      for Number:= 1 to 254 do                             {402}
         if not (Number in SetOfWindows) then              {403}
            Break;                                         {404}

   { Создание окна редактора }
   P:= Application^.ValidView(New(pMyEditWindow,
                  Init(R, FileName, Number)));             {405}
      SetOfWindows:= SetOfWindows + [Number];              {406}
      if not Visible then P^.Hide                          {407}
      else EnableCommands([cmCloseAll]);                   {408}
      DeskTop^.Insert(P);                                  {409}
      OpenEditor:= pMyEditWindow(P);                       {410}
end;                                                       {411}
   { Открытие файла }
procedure tMyEditor.FileOpen(WildCard: PathStr);           {412}
```

```
var                                                              {413}
   FileName: FNameStr;                                           {414}
begin                                                            {415}
   FileName:= '*.*';                                             {416}
   if ExecuteDialog(New(pMyFileDialog, Init(
                WildCard, 'Открыть файл', 'Имя',
                fdOpenButton, 100)),
                @FileName) <> cmCancel then                      {417}
      OpenEditor(FileName, False, True);                         {418}
end;                                                             {419}

procedure tMyEditor.HandleEvent(var Event: tEvent);              {420}

   { Смена каталога }
   procedure ChangeDir;                                          {421}
   var                                                           {422}
      D: pMyChDirDialog;                                         {423}
   begin                                                         {424}
      D:= New(pMyChDirDialog, Init(
               cdNormal + cdHelpButton, 101));                   {425}
      ExecuteDialog(D, nil);                                     {426}
   end;                                                          {427}

   { Новый файл }
   procedure FileNew;                                            {428}
   begin                                                         {429}
      OpenEditor('', False, True);                               {430}
   end;                                                          {431}

   { Показать буфер }
   procedure ShowClip;                                           {432}
   begin                                                         {433}
      ClipWindow^.Select;                                        {434}
      ClipWindow^.Show;                                          {435}
   end;                                                          {436}
   { Закрыть все окна }
   procedure CloseAll;                                           {437}
   begin                                                         {438}
      while DeskTop^.Current <> nil do                           {439}
         pEditWindow(DeskTop^.Current)^.Close;                   {440}
         DisableCommands([cmCloseAll]);                          {441}
end;                                                             {442}
begin
```

```
{ Удаление номера закрываемого окна }
if (Event.What = evCommand) and
          (Event.Command = cmClose) then                    {443}
   SetOfWindows:= SetOfWindows -
        [pWindow(DeskTop^.Current)^.Number];                {444}

inherited HandleEvent(Event);                               {445}

case Event.What of                                          {446}
   evCommand:                                               {447}
      begin                                                 {448}
         case Event.Command of                              {449}
            cmOpen: FileOpen('*.*');                        {450}
            cmNew: FileNew;                                 {451}
            cmShowClip: ShowClip;                           {452}
            cmChangeDir: ChangeDir;                         {453}
            cmCloseAll: CloseAll;                           {454}
         else                                               {455}
            Exit;                                           {456}
         end;                                               {457}
         ClearEvent(Event);                                 {458}
      end;                                                  {459}
   end;                                                     {460}
end;                                                        {461}

   { Процедура, выполняемая в свободное время }
procedure tMyEditor.Idle;                                   {462}

   function IsTileable(P: pView): Boolean;far;              {463}
   begin                                                    {464}
      IsTileable:= (P^.Options and ofTileable <> 0)
            and (P^.State and sfVisible <> 0);              {465}
   end;                                                     {466}

begin                                                       {467}
   inherited Idle;                                          {468}
   if Desktop^.FirstThat(@IsTileable) <> nil then           {469}
      EnableCommands([cmTile, cmCascade])                   {470}
   else                                                     {471}
      DisableCommands([cmTile, cmCascade]);                 {472}
end;                                                        {473}
   { Инициализация русифицированного меню }
procedure tMyEditor.InitMenuBar;                            {474}
```

```
var                                                              {475}
   R: tRect;                                                     {476}
begin                                                            {477}
   GetExtent(R);                                                 {478}
   R.B.Y:= R.A.Y+1;                                              {479}
   MenuBar:= New(pMenuBar, Init(R, NewMenu(
      NewSubMenu('Файл', hcNoContext, NewMenu(
         MyStdFileMenuItems(nil)),
      NewSubMenu('Редактор', hcNoContext, NewMenu(
         MyStdEditMenuItems(
         NewLine(
         NewItem('Буфер', '', kbNoKey, cmShowClip,
            hcNoContext, nil)))),
      NewSubMenu('Поиск', hcNoContext, NewMenu(
         NewItem('Найти...', '', kbNoKey, cmFind,
            hcNoContext, NewItem('Заменить...', '',
            kbNoKey, cmReplace, hcNoContext,
         NewItem('Найти следующий', '',
            kbNoKey, cmSearchAgain, hcNoContext,
            nil)))),
      NewSubMenu('Окно', hcNoContext, NewMenu(
         MyStdWindowMenuItems(nil)),
         nil))))));                                              {480}
end;                                                             {481}

   { Инициализация русифицированной строки состояния }
procedure tMyEditor.InitStatusLine;                              {482}
var                                                              {483}
   R: tRect;                                                     {484}
begin                                                            {485}
   GetExtent(R);                                                 {486}
   R.A.Y:= R.B.Y - 1;                                            {487}
   StatusLine:= New(pStatusLine, Init(R,
      NewStatusDef(0, $FFFF,
         NewStatusKey('~Alt-X~ Выход', kbAltX, cmQuit,
         NewStatusKey('~F2~ Сохранить', kbF2, cmSave,
         NewStatusKey('~F3~ Открыть', kbF3, cmOpen,
         NewStatusKey('~Alt-F3~ Закрыть', kbAltF3, cmClose,
         NewStatusKey('', kbF10, cmMenu,
         NewStatusKey('', kbCtrlF5, cmResize,
         nil))))))),
      nil)));                                                    {488}
end;                                                             {489}
```

```
procedure tMyEditor.OutOfMemory;                                    {490}
begin                                                               {491}
   MyMessageBox('Нет достаточной памяти завершить операцию.',
      nil, mfError + mfOkButton);                                   {492}
end;                                                                {493}

var                                                                 {494}
   Editor: TMyEditor;                                               {495}

begin                                                               {496}
   Editor.Init;                                                     {497}
   Editor.Run;                                                      {498}
   Editor.Done;                                                     {499}
end.                                                                {500}
```

3. Программа корректировки цветовой палитры

```
program Colors;                                                    { 1}
uses App, Dos, Objects, Dialogs, Drivers,
     Menus, StdDlg, Views, ColorSel, Memory;                       { 2}
                                                                   { 3}
const
   cmColor = 1000;                                                 { 4}
   cmControlDialog = 1001;                                         { 5}
                                                                   { 6}
type
   pControlDialog = ^tControlDialog;                               { 7}
   tControlDialog = object(tFileDialog)                            { 8}
      constructor Init;                                            { 9}
   end;                                                            { 10}

   tMyApp = object(tApplication)                                   { 11}
      procedure HandleEvent(var Event: tEvent); virtual;           { 12}
      procedure InitStatusLine; virtual;                           { 13}
   end;                                                            { 14}

var                                                                { 15}
   MyApp: tMyApp;                                                  { 16}

   {*****************************************}
   {*      Подпрограммы общего назначения   *}
   {*****************************************}

{ Структура атрибутов рабочего поля }
function MyDesktopColorItems(
            const Next: PColorItem): PColorItem;                   { 17}
begin                                                              { 18}
   MyDesktopColorItems :=
      ColorItem('Фон',                    1,
      Next);                                                       { 19}
end;                                                               { 20}

{ Структура атрибутов меню и строки состояния поля}
function MyMenuColorItems(
            const Next: PColorItem): PColorItem;                   { 21}
begin                                                              { 22}
   MyMenuColorItems :=
      ColorItem('Нормальный текст',    2,
      ColorItem('Запрещенный элемент',3,
      ColorItem('Клав.быстр.управл.', 4,
```

```
      ColorItem('Выдел.норм.текст',    5,
      ColorItem('Выдел.запрещ.эл-т',   6,
      ColorItem('Выдел.кл.быстр.упр.',7,
      Next))))));                                                    { 23}
end;                                                                 { 24}

{ Структура атрибутов окна }
function MyWindowColorItems(Palette: Word;
           const Next: PColorItem): PColorItem;                      { 25}
const                                                                { 26}
   COffset: array[wpBlueWindow..wpGrayWindow] of Byte =
      (8, 16, 24);                                                   { 27}
var                                                                  { 28}
   Offset: Word;                                                     { 29}
begin                                                                { 30}
   Offset := COffset[Palette];                                       { 31}
   MyWindowColorItems :=
      ColorItem('Рамка/фон пасс.окна',Offset + 0,
      ColorItem('Рамка/фон акт. окна',Offset + 1,
      ColorItem('Пиктограмма',         Offset + 2,
      ColorItem('Линейка скроллинга', Offset + 3,
      ColorItem('Пиктогр.лин.скролл.',Offset + 4,
      ColorItem('Норм. текст поля',    Offset + 5,
      ColorItem('Выдел. текст поля',   Offset + 6,
      ColorItem('Резерв окна',         Offset + 7,
      Next)))))))));                                                 { 32}
end;                                                                 { 33}

function MyDialogColorItems(Palette: Word;
           const Next: PColorItem): PColorItem;                      { 34}
const                                                                { 35}
   COffset: array[dpBlueDialog..dpGrayDialog]
      of Byte = (64, 96, 32);                                        { 36}
   var                                                               { 37}
      Offset: Byte;                                                  { 38}
begin                                                                { 39}
   Offset := COffset[Palette];                                       { 40}
   MyDialogColorItems :=
      ColorItem('Рамка/фон пасс.окна',Offset + 0,
      ColorItem('Рамка/фон акт. окна',Offset + 1,
      ColorItem('Пиктограмма',         Offset + 2,
      ColorItem('Линейка скроллинга', Offset + 3,
      ColorItem('Эл-ты линейки скр.', Offset + 4,
```

```
      ColorItem('Статический текст',  Offset + 5,

      ColorItem('Метка пассивная',    Offset + 6,
      ColorItem('Метка выделенная',   Offset + 7,
      ColorItem('Кл.быстр.упр.метки', Offset + 8,

      ColorItem('Кнопка пассивная',   Offset + 9,
      ColorItem('Кнопка по умолчанию',Offset + 10,
      ColorItem('Кнопка выделенная',  Offset + 11,
      ColorItem('Кнопка запрещенная', Offset + 12,
      ColorItem('Кл.быстр.упр.кнопки',Offset + 13,
      ColorItem('Тень кнопки',        Offset + 14,

      ColorItem('Список пассивный',   Offset + 15,
      ColorItem('Список выделенный',  Offset + 16,
      ColorItem('Кл.быстр.упр.списка',Offset + 17,

      ColorItem('Стр.ввода норм.',    Offset + 18,
      ColorItem('Стр.ввода выдел.',   Offset + 19,
      ColorItem('Стрелки стр.ввода',  Offset + 20,

      ColorItem('Стрелка протокола',  Offset + 21,
      ColorItem('Пиктогр. протокола', Offset + 22,
      ColorItem('Окно протокола',     Offset + 23,
      ColorItem('Пиктогр.окна прот.', Offset + 24,

      ColorItem('Список строк пасс.', Offset + 25,
      ColorItem('Список строк сфок.', Offset + 26,
      ColorItem('Список строк выдел.',Offset + 27,
      ColorItem('Раздел.списка строк',Offset + 28,

      ColorItem('Информационное поле',Offset + 29,
      ColorItem('Список запрещенный', Offset + 30,
      ColorItem('Резерв диал.окна',   Offset + 31,
      Next)))))))))))))))))))))))))))))));                    { 41}
end;                                                          { 42}

{ Получение строки с названием текущего каталога }
function GetCurDir: DirStr;                                   { 43}
var                                                           { 44}
   CurDir: DirStr;                                            { 45}
begin                                                         { 46}
   GetDir(0, CurDir);                                         { 47}
```

```
  if Length(CurDir) > 3 then                                    { 48}
    begin                                                       { 49}
      Inc(CurDir[0]);                                           { 50}
      CurDir[Length(CurDir)]:= '\';                             { 51}
    end;                                                        { 52}
  GetCurDir:= CurDir;                                           { 53}
end;                                                            { 54}

  {*****************************************}
  {*          Объект tControlDialog        *}
  {*****************************************}

constructor tControlDialog.Init;                               { 55}
var                                                             { 56}
  ScrollBar: pScrollBar;                                        { 57}
  R: tRect;                                                     { 58}
  InputName: string;                                            { 59}
begin                                                           { 60}
  { Создание окна диалога }
  R.Assign(15, 1, 64, 20);                                      { 61}
  tDialog.Init(R, 'Контр. диалоговое окно');                    { 62}
  Options:= Options or ofCentered;                              { 63}
  WildCard:= '*.*';                                             { 64}
  InputName:= 'Имя';                                            { 65}

  { Строка ввода имени файла }
  R.Assign(3, 3, 31, 4);                                        { 66}
  FileName:= New(pFileInputLine, Init(R,79));                   { 67}
  FileName^.Data^:= WildCard;                                   { 68}
  Insert(FileName);                                             { 69}
  R.Assign(2, 2, 6, 3);                                         { 70}
  Insert(New(pLabel, Init(R, InputName, FileName)));            { 71}
  R.Assign(31, 3, 34, 4);                                       { 72}
  Insert(New(pHistory, Init(R, FileName, 10)));                 { 73}

  { Линейка скроллинга и список файлов }
  R.Assign(3, 14, 34, 15);                                      { 74}
  ScrollBar:= New(pScrollBar, Init(R));                         { 75}
  Insert(ScrollBar);                                            { 76}
  R.Assign(3, 6, 34, 14);                                       { 77}
  FileList:= New(pFileList, Init(R, ScrollBar));                { 78}
  Insert(FileList);                                             { 79}
  R.Assign(2, 5, 8, 6);                                         { 80}
```

```
  Insert(New(pLabel, Init(R, 'Файлы', FileList)));          { 81}

  { Задание кнопок }
  R.Assign(35, 3, 46, 5);                                   { 82}
  Insert(New(pButton, Init(
            R, 'Открыть', cmFileOpen, bfDefault)));         { 83}
  Inc(R.A.Y,3); Inc(R.B.Y,3);                               { 84}
  Insert(New(pButton, Init(
            R, 'Отмена', cmCancel, bfNormal)));             { 85}

  { Информационная панель с параметрами файла }
  R.Assign(1, 16, 48, 18);                                  { 86}
  Insert(New(pFileInfoPane, Init(R)));                      { 87}

  SelectNext(False);                                        { 88}

  { Загрузка каталога }
  FileList^.ReadDirectory(WildCard);                        { 89}
  Directory:= NewStr(GetCurDir);                            { 90}
end;                                                        { 91}

  {*****************************************}
  {*              Объект tMyApp            *}
  {*****************************************}

procedure tMyApp.HandleEvent(var Event: tEvent);            { 92}

  procedure Color;                                          { 93}
  var                                                       { 94}
    ColorDialog: pColorDialog;                              { 95}
  begin                                                     { 96}
    ColorDialog:= New(pColorDialog, Init('',
      ColorGroup('Рабочее поле',
        MyDesktopColorItems(nil),
      ColorGroup('Меню,стр.сост.',
        MyMenuColorItems(nil),
      ColorGroup('Синее окно',
        MyWindowColorItems(wpBlueWindow, nil),
      ColorGroup('Голубое окно',
        MyWindowColorItems(wpCyanWindow, nil),
      ColorGroup('Серое окно',
        MyWindowColorItems(wpGrayWindow, nil),
      ColorGroup('Сер.диал.окно',
```

```
      MyDialogColorItems(dpGrayDialog, nil),
    ColorGroup('Син.диал.окно',
      MyDialogColorItems(dpBlueDialog, nil),
    ColorGroup('Гол.диал.окно',
      MyDialogColorItems(dpCyanDialog, nil),
    nil))))))))));                                          { 97}

  if ExecuteDialog(ColorDialog,
  Application^.GetPalette) <> cmCancel then                 { 98}
    begin                                                   { 99}
      DoneMemory;                                           {100}
      Redraw;                                               {101}
    end;                                                    {102}
  end;                                                      {103}

  procedure ControlDialog;                                  {104}
  var D: pControlDialog;                                    {105}
  begin                                                     {106}
    D:= New(pControlDialog, Init);                          {107}
    ExecuteDialog(D, nil);                                  {108}
  end;                                                      {109}

begin                                                       {110}
  inherited HandleEvent(Event);                             {111}
    case Event.What of                                      {112}
      evCommand:                                            {113}
        begin                                               {114}
          case Event.Command of                             {115}
            cmColor: Color;                                 {116}
            cmControlDialog: ControlDialog;                 {117}
          else                                              {118}
            Exit;                                           {119}
          end;                                              {120}
          ClearEvent(Event);                                {121}
        end;                                                {122}
    end;                                                    {123}
end;                                                        {124}

procedure tMyApp.InitStatusLine;                            {125}
var                                                         {126}
  R: tRect;                                                 {127}
begin                                                       {128}
  GetExtent(R);                                             {129}
  R.A.Y:= R.B.Y - 1;                                        {130}
```

```
    StatusLine:= New(pStatusLine, Init(R,
      NewStatusDef(0, $FFFF,
        NewStatusKey('~Alt-X~ Выход', kbAltX, cmQuit,
        NewStatusKey('~F2~ Палитра', kbF2, cmColor,
        NewStatusKey('~F3~ Пример окна', kbF3,
          cmControlDialog,
        nil))),
      nil)));                                                    {131}
end;                                                           {132}

begin                                                          {133}
    MyApp.Init;                                                {134}.
    MyApp.Run;                                                 {135}
    MyApp.Done;                                                {136}
end.                                                           {137}
```

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3
1. ОСНОВНЫЕ ПОНЯТИЯ TURBO VISION 8
1.1. Отображаемые и неотображаемые объекты 9
1.2. Координаты отображаемых объектов 14
1.3. Объект tView 15
1.4. Группа (tGroup). Z-порядок 24
1.5. Оформление программы, использующей средства Turbo Vision. Объекты tProgram и tApplication 27
1.6. Структура отображаемых объектов программы 31
1.7. Обработка событий 34
1.8. Команды 41
2. ПОРЯДОК РАБОТЫ С ОТОБРАЖАЕМЫМИ ОБЪЕКТАМИ 43
2.1. Описание отображаемого объекта 43
2.2. Создание отображаемого объекта 44
2.3. Помещение отображаемого объекта в группу 45
2.4. Задание параметров отображаемому объекту 46
2.5. Работа с отображаемым объектом 47
2.6. Получение введенных параметров 47
2.7. Удаление отображаемого объекта 48
2.8. Работа с модальными объектами 48
3. СОЗДАНИЕ МЕНЮ 52
3.1. Создание разворачивающегося меню 52
3.2. Создание локального меню 59
4. СОЗДАНИЕ ОКОН И ДИАЛОГОВЫХ ОКОН 63
4.1. Окна (tWindow) в Turbo Vision 64
4.2. Элементы окон в Turbo Vision 70
4.2.1. Поле скроллинга (tScroller) 71
4.2.2. Текстовый элемент буферированной информации (tTextDevice) 73
4.2.3. Текстовый элемент для пассивного терминала (tTerminal) 73

4.2.4. Линейка скроллинга (tScrollBar) 75
4.3. Диалоговое окно (tDialog) 78
4.4. Элементы диалоговых окон.....82
4.4.1. Строка ввода (tInputLine).....83
4.4.2. Метка (tLabel) 84
4.4.3. Протокол (tHistory) 86
4.4.4. Селективный (tRadioButtons) и триггерный (tCheckBoxes) списки 87
4.4.5. Кнопка (tButton).....90
4.4.6. Статический (tStaticText) и параметрический (tParamText) тексты 93
4.4.7. Списки строк.....96
4.5. Стандартные окна 101
5. СПРАВОЧНАЯ СИСТЕМА TURBO VISION.....103
5.1. Включение в программу справочной системы.....103
5.2. Создание файла справочной системы.....106
6. СОЗДАНИЕ СТРОК СОСТОЯНИЯ 110
6.1. Строка состояния, отображающая клавиши быстрого управления 110
6.2. Добавление в строку состояния краткой информации о текущих процессах, протекающих в программе.....114
7. ЦВЕТОВАЯ ПАЛИТРА TURBO VISION 116
7.1. Стандартная палитра Turbo Vision.....116
7.2. Изменение стандартной палитры 118
8. НЕОТОБРАЖАЕМЫЕ ОБЪЕКТЫ TURBO VISION.....121
8.1. Анализаторы вводимой информации 121
8.2. Объект-коллекция 124
8.3. Потоки Turbo Vision 131
8.4. Ресурсы.....140
8.5. Списки строк 142
8.6. Использование редакторов ресурсов.....146
9. НЕКОТОРЫЕ ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ TURBO VISION.....148
9.1. Текстовый редактор 148
9.2. Отображение древовидных структур 157
9.3. Диалоговое окно задания цветовой палитры (tColorDialog).....161

Заключение 163
ПРИЛОЖЕНИЯ 164
А. Распределение стандартных объектов Turbo Vision по модулям 164
Б. Стандартная цветовая палитра Turbo Vision 166
В. Перечень процедур регистрации стандартных объектов Turbo Vision 170
Г. Текст программы-калькулятора 172
Д. Содержимое исходного файла фрагментов справочной системы 197
Е. Модуль констант справочной системы 204
Ж. Текстовый редактор 206
З. Программа корректировки цветовой палитры 225

ДИАЛОГ-МИФИ

320-43-77, 320-43-55
факс: 324-30-55

предлагает литературу по программированию и вычислительной технике,

расчитанную на широкий круг пользователей персональных компьютеров

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АВТОЛИСП В СИСТЕМЕ САПР АВТОКАД

С. А. Гладков

Учебно-справочное пособие по программированию. Описываются структура языка, даются методика написания и примеры программ. Рассмотрены вопросы написания макроопределений, меню Автокада и настройки рабочей среды. Ориентировано на пользователей персональных компьютеров, работающих в области САПР.

КУРС ПРАКТИЧЕСКОЙ РАБОТЫ С СИСТЕМОЙ АВТОКАД 12

Ю. А. Кречко, В. В. Полищук

В новом учебном пособии описаны основные отличия команд версии 12 от предыдущих версий. Дано описание команд рисования, редактирования, настройки вывода твердых копий. Дан список всех команд и системных переменных Автокада 10, 11 и 12 версий. Параллельно рассматривается и оригинальная (англоязычная), и русифицированная версии.

МИКРОПРОЦЕССОР i486. АРХИТЕКТУРА, ПРОГРАММИРОВАНИЕ, ИНТЕРФЕЙС

В. Б. Бродин, И. И. Шагурин

Книга, подготовленная при поддержке фирмы Intel, описывает архитектуру, программирование и механизмы обмена микропроцессора i486. Первая глава на примере i486 знакомит читателя с основными принципами функционирования современных высокопроизводительных 32-разрядных микропроцессоров. Структура и режимы работы i486 описаны в общем виде. В последующих главах подробно рассматривается система команд, работа в реальном и защищенном режимах, механизм прерываний и сообщения об ошибках, магистральные циклы обмена, встроенные средства отладки прогрммного обеспечения. Книга адресована как специалистам в области микропроцессорной техники, так и широкому кругу читателей, самостоятельно осваивающих эту область знаний.

SYMANTEC C++ 6.1 Professional.

Н. И. Березина, В. В. Лопушенко, И. Н. Посохов

Книга представляет собой практическое руководство по разработке приложений в среде SYMANTEC C++ 6.1 Professional. Во введении подчеркнуты наиболее интересные особенности пакета. В первой части описана работа в среде IDDE для начинающих пользователей. Приведенная в ней информация о среде дает возможность создать и отладить простейший проект. Вторая часть посвящена особенностям реализации языка C++. Также приведена информация, необходимая для разработки 32-разрядных приложений, и кратко освещены некоторые аспекты работы с плавающей запятой. Часть третья является основной и содержит подробное описание компонент пакета, связанных с компиляцией и отладкой приложений. При написании книги авторы ориентировались на пользователей, которые умеют работать в среде Windows 3.1 и знакомы с языком C++.

ТЕКСТОВЫЙ ПРОЦЕССОР Microsoft Word for Windows.

Краткий справочник пользователя

О. А. Кузьминова

Книга представляет собой краткий справочник по текстовому процессору Microsoft Word for Windows версии 2.0. В ней описаны все основные приемы редактирования документов, работа с макрокомандами и графикой, процедуры сборки оглавления документа и других таблиц. Для удобства приведены таблицы комбинаций клавиш, используемых для выполнения основных операций редактирования документов, а также другая справочная информация.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА

В. Н. Пильщиков

Книга представляет собой учебное пособие по языку ассемблер для персональных компьютеров типа IBM PC. Подробно рассмотрены команды этих ЭВМ, конструкция языка и методы программирования на нем. Изложение сопровождается многочисленными примерами. Рекомендуется для студентов и преподавателей ВУЗов, для всех желающих детально изучить язык ассемблер и приемы программирования на нем.

ПРОГРАММИРОВАНИЕ В СРЕДЕ TURBO PASCAL 7.0.

А. Епанешников, В. Епанешников

В данном пособии описана версия 7.0 широко распространенного пакета программирования Turbo Pascal, разработанного фирмой Borland International. Приведены основные характеристики языка и среды программирования. Пособие может быть полезно как при изучении языка Turbo Pascal, так и при создании программ на этом языке.

ПЛАНИРОВАНИЕ И УПРАВЛЕНИЕ ПРОЕКТОМ С ИСПОЛЬЗОВАНИЕМ TIME LINE

Е. В. Полковникова, А. В. Полковников

Данная книга посвящена использованию систем планирования и управления проектами Time Line корпорации Symantec.

Наряду с описанием команд меню и функций последних версий системы для DOS и Windows, авторы рассматривают практические вопросы эффективности использования данной технологии управления в организации. Книга содержит последовательное описание применения средств Time Line на разных стадиях планирования и управления проектом. Отдельно рассматриваются вопросы, связанные с внедрением новой технологии управления на основе Time Line.

Книга может быть полезна как специалистам, ответственным за информационное обеспечение управленческой деятельности организации, так и непосредственно руководителям различного уровня, а также студентам соответствующих специальностей.

Из серии

Библиотека системного программиста

Том 10. КОМПЬЮТЕР IBM PC/AT, MS-DOS и Windows. ВОПРОСЫ И ОТВЕТЫ

А. В. Фролов, Г. В. Фролов

Книга охватывает ряд вопросов, наиболее часто возникающих при работе с компьютером. Мы рассмотрели проблемы, постоянно преследующие пользователей компьютерной техники, в том числе установку операционных систем MS-DOS 6.0 и Windows 3.1, русификацию компьютера и программ, восстановление данных. Приведены рекомендации по увеличению объема доступной оперативной и внешней памяти. Описаны средства защиты компьютерных данных от несанкционированного доступа.

Том 11. ОПЕРАЦИОННАЯ СИСТЕМА Microsoft Windows. ВВЕДЕНИЕ ДЛЯ ПРОГРАММИСТА. Часть 1

А. В. Фролов, Г. В. Фролов

Первый из нескольких томов, посвященный программированию для операционной системы Microsoft Windows. В книге рассмотрены основные принципы работы Microsoft Windows версии 3.1. Приведены все сведения, необходимые для того чтобы приступить к программированию в среде этой операционной системы. Книга содержит большое количество примеров программ на языке C++. Все примеры подготовлены для транслятора Borland C++ for Windows версии 3.1.

Том 12. ОПЕРАЦИОННАЯ СИСТЕМА Microsoft Windows. ДЛЯ ПРОГРАММИСТА. Часть 2

А. В. Фролов, Г. В. Фролов

В ней продолжается изучение программного интерфейса Windows. Рассмотрены ресурсы, предопределенные классы окон, а также диалоговые панели. Вы научитесь работать как со стандартными ресурсами, так и с ресурсами, имеющими произвольный формат; создавать в приложениях Windows различные органы управления, такие как кнопки, переключатели, списки, редакторы текста и т. д. Отдельная глава посвящена диалоговым панелям - мощному средству организации пользовательского интерфейса. Приведено большое количество исходных текстов приложений на языке программирования C++.

Том 13. ОПЕРАЦИОННАЯ СИСТЕМА MICROSOFT WINDOWS 3.1 ДЛЯ ПРОГРАММИСТА. Часть 3.

А. В. Фролов, Г. В. Фролов

Третья в серии книг, посвященных программированию для операционной системы Microsoft Windows. В ней мы научим вас создавать меню различных типов (в том числе меню с использованием графических изображений). Вы узнаете о способах создания меню, а также о том, как подключить к меню клавиатурный интерфейс и создать орган управления Toolbar. Подробно описаны средства управления памятью, особенности работы программ в защищенном режиме, использование локальной и глобальной области памяти, а также работа с моделями памяти. Вы научитесь работать с блоками памяти размером больше 64 Кбайт и узнаете о функциях, позволяющих работать с локальной таблицей дескрипторов. Рассмотрены вопросы создания DLL-библиотек, приведены конкретные области их применения. Отдельная глава посвящена работе с файлами и стандартными диалоговыми панелями, предназначенными для открытия файлов.

Том 14. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС GDI В MICROSOFT WINDOWS.

А. В. Фролов, Г. В. Фролов

Книга представляет собой практическое руководство по использованию графического интерфейса GDI операционной системы Microsoft Windows. В ней подробно рассмотрены принципы, положенные в основу GDI, приведены все необходимые сведения об использовании различных систем координат, перьев и кистей, цветов и цветовых палитр. Вы научитесь работать с файлами графических изображений bitmap, использовать различные шрифты для вывода текста, узнаете о метафайлах и о том, как приложения Windows работают с принтером.

Том 15. МУЛЬТИМЕДИА ДЛЯ WINDOWS. РУКОВОДСТВО ПРОГРАММИСТА.

А. В. Фролов, Г. В. Фролов

Книга предназначена для тех программистов, которые стремятся использовать в своих разработках новейшую технологию мультимедиа. Эта технология предполагает естественные для человека способы представления информации - через звук и видео. Мы привели основные сведения, необходимые для создания систем мультимедиа с помощью программного интерфейса MS Windows, а также Microsoft Video for Windows версии 1.1. Пользуясь нашей книгой, вы сможете создавать приложения Windows для проигрывания и записи звука, проигрывания музыки (MIDI-файлов), управления устройством чтения компакт-дисков CD-ROM, проигрывания видеофильмов в формате Microsoft Video for Windows, научитесь использовать другие средства мультимедиа.

ПК шаг за шагом

Серия книг для самостоятельно постигающих тайны компьютерного мира

Том 1. ВВЕДЕНИЕ В MS-DOS, MS Windows, MS Word for Windows.

А. В. Фролов, Г. В. Фролов

Дается краткий обзор аппаратных и программных средств, который поможет вам понять назначение и оценить возможности персонального компьютера. В книге рассказывается о подготовке компьютера к работе и установке операционной системы MS-DOS. Отдельная глава посвящена текстовому процессору MS Word for Windows (версии 2.0 и 6.0). Главное внимание авторы уделяют обучению работе на ПК. Эта книга и вся серия ориентирована на начинающего пользователя персонального компьютера, не имевшего возможности его освоить в школе или институте. Она также может быть полезна для начинающих программистов.

Том 2. ОПЕРАЦИОННАЯ СИСТЕМА Microsoft Windows.

Руководство пользователя

А. В. Фролов, Г. В. Фролов

В книге описываются процедуры установки и настройки операционной системы Microsoft Windows версии 3.1. Рассмотрены характеристики стандартного и расширенного режимов Windows. Описаны настройка Microsoft Windows, использование механизма обмена и разделения информации между приложениями Windows - OLE. Рассмотрены основные приложения Windows, входящие в состав дистрибутива, приведены примеры их использования.

Издательство АО "Диалог-МИФИ" готовит к изданию:

БСП 16 том. МОДЕМЫ И ФАКС-МОДЕМЫ.

Программирование для MS-DOS и Windows

А. В. Фролов, Г. В. Фролов

Книга представляет собой новую редакцию четвертого тома "Библиотеки системного программиста" и посвящена использованию модемов и факс-модемов. В ней рассмотрены различные типы современных модемов, описаны их команды и регистры. Приведены рекомендации по покупке и установке модема. Описана процедура установки и настройки нескольких наиболее распространенных телекоммуникационных программ. Вы научитесь передавать и принимать документы через модемы и факс-модемы.

В книге описаны приемы программирования портов асинхронного последовательного адаптера и модемов в среде операционных систем MS-DOS и Windows, приведены несколько примеров программ.

По вопросам рассылки книг почтой просим Вас обращаться к нашим дилерам:

Издательство **"Титул"**
(только оптовые поставки)
249020, г. Обнинск, а/я 5055.
Телефон (08439) 2-03-52,
факс (08439) 3-31-19

Редакция **"Софт Маркет"**
(оптом и в розницу)
127562, Москва, а/я 120
Телефон (095) 209-18-97,
факс (095) 209-49-04

Книги издательства "Диалог - МИФИ" можно купить по адресам:

Владивосток

Салон деловой, профессиональной и образовательной информации "Оферта-Дело". 690002, Океанский пр-т, 104; тел.: (4232) 25-85-11

Екатеринбург

АОЗТ "ЕПИКО"; тел.: (3432) 34-92-02, 28-75-20.
Почтовый адрес: 620149, а/я 483

Ижевск

АО "DATUM pro" ул. Репина, 22. Тел.-факс (3412) 25-30-35
426000, Ижевск, Главпочтамт, а/я 1371

Кемерово

Магазин "Книжный мир", Советсткий пр-т. 43

Киев

ТОО " В Е К " , тел. (044) 510-93-65, тел./факс 483-29-85.
Киоск "Техническая книга", пр. Победы, 37, КПИ, уч. корп. N 7 (вестибюль)

Минск

МНВЦ "КЕЙ". 220040, ул. М. Богдановича, 153, к.609. Тел./факс: 32-64-16

Москва

- **"Московский дом книги", ул. Новый Арбат, дом 26.**
- **"Дом технической книги", Ленинский пр-т., дом 40.**
- **ТОО ТД "Библио-глобус", 101861, ул. Мясницкая, 6**
- **АО "ДИАЛОГ-МИФИ"**
  115409, Москва, ул. Москворечье, дом 31, корп. 2.
  Телефон: (095) 320-43-77, факс: (095) 3243055
- **"ДИАЛОГ-САЛОН", ВВЦ (бывшая ВДНХ)**
  павильон "Вычислительная техника", тел.: (095) 974-60-11.
- **"СОФТ МАРКЕТ", редакция, (*рассылка по почте*)**
  ул. Медведева, 13, 4 этаж. Тел.: 209-18-97

Новосибирск

Магазин "Н А У К А ", Академгородок, Морской пр-т, 22.

Обнинск

Из-во "Титул", Центр реализации деловых изданий (*оптовые поставки*)
249020, а/я 5055. Тел.: (08439) 2-03-52

Самара

ТОО "Чакона", ул. Урицкого, 1. Тел.: (8462) 36-14-87, факс: 32-02-87

Санкт-Петербург

"Техническая книга", ул. Пушкинская, д.2. Тел. 164-65-65

Челябинск

Челябинский региональный центр высшей школы пр-т Ленина, 87,
тел.: (3512) 65-49-77, 65-59-53. Почтовый адрес: 454080, а/я 9520

Приглашаем распространителей по регионам

Учебный центр ДИАЛОГ-МИФИ

авторизован фирмами
BORLAND, SYMANTEC, AUTODESK

КУРСЫ УСКОРЕННОЙ ПОДГОТОВКИ К ПРАКТИЧЕСКОЙ РАБОТЕ НА ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРАХ IBM PC

УСЛОВИЯ ПРИЕМА И ОБУЧЕНИЯ

- На курсы принимаются граждане нашей страны и иностранцы без каких-либо ограничений.
- Для обучения на курсах предварительной подготовки в области вычислительной техники, как правило, не требуется.
- Каждый день занятий - это трехчасовая лекция и четыре часа индивидуальной практической работы на компьютерах IBM PC.
- Продолжительность курса: одна или две недели с отрывом от производства (5 раз в неделю).
- Занятия проводятся под руководством высококвалифицированных преподавателей Московского инженерно-физического института и Учебного центра .
- Оплата производится по наличному или безналичному расчету.
- Окончившим курсы выдается удостоверение-сертификат международного образца.
- Учебный центр проводит занятия на базе Заказчика.
- В Учебном центре для всех желающих проводятся платные консультации и принимаются платные экзамены по преподаваемым курсам с выдачей удостоверения-сертификата.

ДВУХНЕДЕЛЬНЫЕ КУРСЫ

1. Практическая работа на компьютерах семейства IBM PC в операционной среде MS-DOS (начальный курс).
2. Основы системного программирования на языке Ассемблера.
3. Программирование на Turbo C++ (начальный курс).
4. Программирование на Turbo C++ (для знающих C).
5. Программирование под Windows на Borland C++.
6. Программирование на Turbo Pascal.
7. Основы практического применения Turbo Vision (Turbo Pascal).
8. Практическая работа с БД Paradox.
9. Практическая работа с Paradox for Windows.
10. Использование САПР Автокад.
11. Локальные сети.
12. Практическая работа с Microsoft Works.
13. Техническое обслуживание ПК IBM PC

ОДНОНЕДЕЛЬНЫЕ КУРСЫ

1. Использование электронных таблиц Microsoft Excel.
2. Изучение графической среды Microsoft Windows.
3. Использование Time Line для планирования и управления проектом.
4. Программирование на Автолиспе (Автокад).
5. Программирование на PAL (в среде СУБД Paradox).
6. Практическая работа с текстовым процессором Word for Windows.
7. Работа с графической системой Corel Draw.
8. Настольная издательская система Page Maker.

115409, Москва, ул. Москворечье, 31, корп. 2
Телефон: (095) 320-30-88
Телефакс: (095) 3243055

Учебный центр

ДИАЛОГ-МИФИ

Проводит курс занятий на тему

"TURBO VISION 2.0. Основы практического использования"

BORLAND

TURBO

VISION

Занятия ведут авторы книги

115409, Москва, ул. Москворечье, 31, корп. 2
Телефон: (095) 320-30-88
Телефакс: (095) 324-30-55