
Ал. Стивенс

" ТЕХНИКА ПРОГРАММИРОВАНИЯ НА ТУРБО СИ "

Предисловие..... 5
Обзор разделов..... 7
ГЛАВА 1..... 9
Интерактивное программное обеспечение, управляющее изображением. 9
ГЛАВА 2..... 14
Язык Си..... 14
Краткая история языка Си..... 14
Особенности языка Си..... 15
Достоинства языка Си..... 17
Одобрение языка Си..... 18
Рекомендуемая литература по Си 18
ГЛАВА 320 .....
Компилятор Турбо Си 20
Два Турбо Си 21
Настройка интегрированной среды 21
Редактор Турбо Си 22
Компоновщик Турбо Си 23
Утилита построителя задач (Make) в Турбо Си 23
Обнаружение ошибок при компиляции и компоновке24 .....
Программные средства низкого уровня 24
Начальная установка 25
Модели памяти 25
Библиотека исходных модулей 25
Заключение 25
ГЛАВА 4 26
Функции общего назначения 26
Исходные модули функций общего назначения 28
Заключение..... 32
ГЛАВА 5 33
Экранные окна 33
Экранное окно 33
Архитектура видеопамяти 36
" Снег" и обратный ход луча развертки 39
Заключение41 .....
ГЛАВА 6 42
Библиотека оконных функций 42
Стековые окна 42
Слоеные окна 43
Оконные функции 46
Листинги оконных функций 51
Описание программы: twindow.h54 .....
Описание программы: twindow.c 66
Примеры окон 69
Перемещение окна 69
Подъем и опускание окон 72
Назначение заголовков и изменение цветов окна 74
Сравнение стековых и слоеных окон 76
Перемещение, подъем, скрытие окон, меню, изменение интенсив77 .. .
Резюме 82
ГЛАВА 7 83
Контекстно-управляемые окна подсказки 83
Программирование окон подсказки 84

Текстовый файл окна подсказки 86
Функции подсказки 88
Изменение функциональной клавиши подсказки 88
Изменение функции подсказки 88
Выключение подсказки 89
Исходный листинг: thelp.c 89
Описание программы: thelp.c 91
Пример контекстно-управляемой подсказки 92
Резюме 94

ГЛАВА 8 95
Использование данных в окнах95 .....
Шаблон ввода данных 95
Поле ввода данных 96
Позиция 96
Атрибуты 96
Буфер 96
Проверка допустимости значений 96
Help-информация97 .....
Маска вводимых данных 97
Приглашения к вводу в поле (Prompts) 97
Ввод данных 98
Функции сбора данных 98
Исходный текст: entry.c 103
Описание программы: entry.c 109
Пример: Ввод данных в определенном порядке 112
Резюме117 .....
ГЛАВА 9 118
Оконный текстовый редактор 118
Команды тестового редактора 119
Управление курсором 119
Постраничная работа 120
Команды работы с блоками текста 120
Команды редактирования121 .....
Функция, реализующая текстовый редактор 121
Исходный листинг: editor.c 122
Описание программы: editor.c 134
Пример: Использование редактора 138
Резюме 140
ГЛАВА 10 141
Оконные меню141 .....
Меню 141
Процесс, образующий оконное меню 142
Функции поддержки меню 143
Исходный листинг: tmenu.c 144
Описание программы: tmenu.c 147
Пример оконного меню 148
Резюме 151
ГЛАВА 11 152
Резидентные программы 154
Прерывания 154
Векторы прерывания 154
Аппаратные прерывания 155
Программные прерывания 155
ДОС - однозадачная операционная система 155
TSR-программы157 .....
Программы обработки прерываний 158
Резидентные утилиты 158
Что может быть резидентным 159

Построение TSR-программ 160
Превращение программы в резидентную 161
Резидентна ли уже программа? 161
Захват прерывания162 .....
Величина TSR-программы 163
Переключение контекстов 165
Стек 165
Program Segment Prefix (PSP) 166
Дисковый буфер 172
Прерывание от клавиатуры (9) 173
Прерывание от таймера 173
Проблема реентерабельности ДОС 174
Два стека ДОС 174

Системный флажок занятости (0x34) 174
Прерывание DOSOK 175
Дисковое прерывание ROM-BIOS.(0x13) 176
Прерывание Ctrl-Break в ДОС.(0x23) 177
Выполнение TSR-программы 177
Завершение TSR-программы 177
Приостановка и возобновление выполнения TSR-программы 179
Выводы 179
ГЛАВА 12 180
Построение резидентных программ 180
Пример TSR-программы - "часы" 180
Превращение программы в резидентную 180
Прерывание по делению на ноль181 .....
Выполнение обработчика прерываний от таймера 182
Связывание старого вектора прерывания по таймеру 182
Сохранение и переключение контекста стека 182
Вычисление времени 182
Программы TSR-драйвера 185
Действия трех программных модулей 186
Размер TSR-программы186 .....
Присвоение "горячего ключа" 186
Сигнатура TSR-программы 188
Коммуникационные прерывания 188
Подготовка к резидентности 189
Обработчик обращения к диску 190
Обработчик критических ситуаций 191
Обработчик клавиатуры 191
Обработчик таймера 191
Обработчик DOSOK 191
Выполнение TSR-программы 192
Удаление TSR-программы 192
Блоки памяти и управляющие блоки памяти 193
Исходные тексты: popup.c, resident.c 193
TSR-программа - приложение 201
Проверка TSR-программ202 .....
Выводы 203
ЭПИЛОГ..... 203

Предисловие

Поскольку вы читаете данную книгу, то, вероятно, вы программируете на языке Си и уже приобрели или собираетесь приобрести компилятор Турбо Си для своей IBM PC. При чтении от вас потребуется довольно хорошее знание языка Си, а также DOS- операционной системы персональных ЭВМ (ПЭВМ) линии IBM PC - и ее функций. Знание языка ассемблера процессора 8086 и архитектуры IBM PC желательно, но не обязательно. В книге содержится множество исходных модулей функций на языке Си, которые помогут писать программы, работающие с окнами, а также делать ваши программы резидентными в памяти.

Программы, работающие с окнами, и резидентные в памяти программные утилиты составляют в настоящее время основное направление в программировании для IBM PC. По своей природе персональная ЭВМ является настольной интерактивной (диалоговой(системой, которая предоставляет пользователю доступ к набору интерактивных программ. Аппаратура и операционная система обеспечивают возможность разработки программ, работающих с окнами и меню, которые появляются на экране по нажатию клавиши. Большинство пакетов программ, пользующихся в настоящее время наибольшим спросом у пользователей, применяют хотя бы одно из этих средств. В данной книге разбираются основы работы с ними и содержатся исходные тексты функций на языке Си, позволяющие использовать эти средства в ваших программах. Прочитав эту книгу и разобрав содержащиеся в ней программы, а также освоив компилятор Турбо Си и основы программирования на языке Си, вы будете готовы создавать резидентные программные утилиты, использующие окна для организации пользовательского интерфейса.

Эта книга содержит сведения о языках программирования, о развитии программного обеспечения, а также примеры использования языков программирования для написания интерактивных, экранно-ориентированных программ для ЭВМ. Не следует думать, что перед вами очередная книга об IBM PC, но образ этой персональной ЭВМ постоянно присутствует здесь. Если прежде акроним PC обозначал определенную ЭВМ, то теперь он обозначает архитектуру ЭВМ, которая была создана промышленным гигантом и стала общепризнанной. PC в данной книге не является объектом изучения, а обозначает некоторый абстрактный объект, который располагается на вашем столе, работает под управлением MS-DOS и называется PC, XT, AT или чем-либо совместимым с ними.

В данной книге вы столкнетесь с программами, написанными на языке Си. Это замечательный язык, и хотя некоторым он не нравится, но все же большинство программистов его любят. На Си вы можете создавать программы, которые делают все, что вы пожелаете. Нет другого такого языка, который бы так же стимулировал к программированию. Создается впечатление, что остальные языки программирования воздвигают искусственные препятствия для творчества, а Си - нет. Использование этого языка позволяет сократить затраты времени на создание работающих программ. Си позволяет программировать быстро, эффективно и предсказуемо. Еще одно преимущество Си заключается в том, что он позволяет использовать все возможности вашей ЭВМ. Этот язык создан программистом для использования другими программистами, чего о других языках программирования сказать нельзя. Кобол был создан таким, чтобы менеджеры могли разбираться в написанных на этом языке программах; Бэйсик был создан для непрограммистов; Фортран - для научных работников; Ада вообще был создан

прямо-таки правительственным комитетом; Пайлот создан для учителей; Паскаль - для студентов ;Лого - для детей; АПЛ - для марсиан; Форт, Лисп и Пролог - специализированные языки. Один Си -это язык для программистов.

Турбо Си, о котором идет речь в этой книге, - это пакет, который создает программную среду для программирования на языке Си и является первым из компиляторов Си нового поколения. Турбо Си содержит редактор с возможностью установки его параметров пользователем, построитель задач, ориентированный на реализацию программного проекта, "быстрый" компоновщик, а также самый "быстрый" компилятор Си для PC, которые "погружены" в интегрированную, оконно-ориентированную программную среду. Турбо Си также предоставляет возможность работы с библиотекой функций и расширениями языка Си, что обеспечивается использованием вспомогательных программ обработки прерываний и других резидентных в памяти программ. Такое использование законно, поскольку Borland International - создатель Турбо Си- является также основным производителем резидентных программных утилит.

В этой книге содержатся исходные тексты функций, которые вы можете использовать в своих программах, работающих в режиме интерактивного взаимодействия с пользователем. Использование этих функций улучшит пользовательский интерфейс ваших программ. Они обеспечивают возможности работы с окнами, меню, ввода данных по установленному шаблону, оконного редактирования текста, а также создания резидентных программ, которые вызываются нажатием определенных клавиш.

Кроме описания этих функций в книге излагаются также аппаратные и программные принципы, которые лежат в основе создания программ, управляющих выводом изображений и резидентных программ. Подробно рассматриваются система прерываний, видеопамять, а также внутренняя организация DOS, включая множество функций DOS, использование которых необходимо при создании резидентных программ, но по которым нет документации или, наоборот, которые распространяются разработчиками и поставщиками DOS.

Обзор разделов

Глава 1 знакомит с концепцией интерактивных, экранно-ориентированных программных систем, в которых организация обмена с пользователем так же важна, как и прикладное назначение программы.

Глава 2 содержит основные сведения о языке Си.

Глава 3 описывает компилятор Турбо Си и его интегрированную среду.

Глава 4 знакомит с первой группой функций, использующих особенности аппаратной архитектуры PC.

Глава 5 объясняет основные принципы работы с окнами, содержит общие сведения об архитектуре видеосистемы и знакомит с проблемами, возникающими при создании окон в видеопамяти PC.

Глава 6 представляет читателю библиотеку функций для работы с окнами. Эти функции могут применяться в пользовательских программах для отображения различного рода информации, а также быть основой для создания меню, редакторов и функций ввода

данных по формату, которые разбираются в последующих разделах.

Глава 6 содержит также несколько примеров программ, иллюстрирующих использование библиотеки функций для работы с окнами.

Глава 7 описывает контекстно-зависимые информационные окна)Help) и содержит исходные тексты функций, которые позволят реализовать эту возможность.

Глава 8 знакомит с использованием окон для ввода данных по формату; управление вводом при этом осуществляется путем определения набора полей для ввода данных внутри определенного окна. Существуют функции, которые позволяют реализовать эту возможность в ваших программах. В качестве примера приводится программа диалогового ввода данных.

Глава 9 содержит функцию редактирования текстовой информации, использующую окна. Описываемая здесь программа представляет собой текстовый редактор общего назначения для ввода и редактирования текстов свободного формата. Он имеет множество команд, присущих большим системам текстовой обработки и обеспечивающих автоматическое форматирование текста, автоматический перенос слов, выделение и перемещение фрагментов и т.д. Приводится также текст программы интерактивной записной книжки, в которой используется функция редактирования текста.

Глава 10 знакомит с системами меню и содержит ряд функций, позволяющих создавать один из типов меню, который можно встретить в серьезных программах: строковое меню в заголовке окна, выбор каждого из элементов которого вызывает возникновение на экране нового меню. Для иллюстрации использования такого типа меню программные модули объединены в единую программу, которая позволяет с помощью меню выбрать нужный модуль.

Глава 11 знакомит с основами реализации резидентных программ. По этой проблеме дается исчерпывающая информация. Приводятся также разъяснения по тем функциям DOS, по которым не поставляется документация: какие из них можно использовать, а каких следует избегать и почему. Освещается проблема реентерабельности DOS и способы ее решения. Упоминается также проблема параллельно выполняющихся резидентных утилит. В заключение обсуждаются свойства "однозадачности" DOS и объясняется, почему не может быть обеспечена надежная защита резидентных в памяти программ.

Глава 12 на примерах демонстрирует, как можно использовать Турбо Си для создания резидентных программных утилит. Первый пример представляет резидентную в памяти утилиту обработки прерываний по таймеру, которая отображает текущее время в правом верхнем углу экрана. Также приведена управляющая программа общего назначения, которая позволит Вам разрабатывать утилиты, тестировать их в качестве нерезидентных программ в среде Турбо Си, а затем компоновать их в рабочие резидентные модули .Для иллюстрации этого процесса программа управления окнами и меню из главы 10 преобразуется в резидентную программу, которая выполняется при нажатии "горячей клавиши."

Подводя итог, можно сказать, что данная книга содержит разъяснения и исходные тексты программ, касающиеся двух наиболее популярных свойств программного обеспечения для PC- использования окон и резидентности программ. Пользуясь этими инструментами и полными возможностями пакета Турбо Си, вы сможете повысить свою производительность в программировании, а также сделать свои программы более полезными и "дружественными" для пользователя.

Сентябрь, 1987

Меррит Айленд, Флорида

ГЛАВА 1

Интерактивное программное обеспечение, управляющее изображением на экране

Большинство программ для PC, пользующихся в настоящее время наибольшим спросом, рассчитаны на интерактивный режим работы, при котором пользователь обменивается с ЭВМ сообщениями в виде последовательностей нажатий клавиш на клавиатуре и символов на экране дисплея. При этом программы вывода сообщений пользователю широко используют возможности видеотерминала PC. Пользователь реагирует на это набором на клавиатуре соответствующих слов и чисел. Такой способ общения с ЭВМ стал естественным для нового поколения пользователей. Он подразумевает развитие стиля отображения и ввода информации в ЭВМ, который называется "смотреть и чувствовать" и постоянно используется и развивается программистами. Французский язык, Кобол, код Морзе являются средствами общения, для общения с ЭВМ также нужен язык. Можно считать, что каждая новая программа является новым языком или диалектом уже существующего. И поскольку в данном случае язык служит для взаимодействия с ЭВМ, то повышение его эффективности способствует более полному соответствию намерений пользователя и действий ЭВМ.

Эти языки, как правило, не разрабатываются специально, а возникают сами по себе в процессе создания программы. Программист озабочен обычно другими проблемами: средой программирования, структурами данных, функциональными алгоритмами, интерфейсом пользователя. После завершения разработки программы программист вводит в нее запросы к пользователю и сообщения об ошибках. Качество пользовательского интерфейса программы зависит от воли программиста и наличия программных средств, которые помогают разрабатывать пользовательский интерфейс.

Программисты могут использовать множество различных технологий для организации обмена между ПЭВМ и пользователем. Каждая из этих технологий имеет свои области применения, в которых она более предпочтительна, чем другие. Но все эти технологии имеют одну общую цель: они обеспечивают средства для передачи пользователю предназначенной для него информации и для ввода информации им.

Одна из наиболее простых проблем, с которой вы могли сталкиваться в своей практике, заключается в организации ввода и вывода алфавитно-цифровой информации. Вы можете использовать для этой цели функции printf и scanf и этим ограничиться. Если ЭВМ включает в свой состав консольный терминал с клавиатурой типа пишущей машинки, то при этих условиях почти любое программное обеспечение может считаться обеспеченным пользовательским интерфейсом. Эта технология была применима по той причине, что ЭВМ работали под управлением операторов, а пользователями считались те, кто записывал данные на программных бланках, а затем читал распечатки. Теперь же ПЭВМ стоит на столе пользователя, поэтому для их эффективного взаимодействия пользовательский интерфейс должен быть более сложным, чем ранее.

Поскольку сегодняшний пользователь ЭВМ обычно имеет профессиональные интересы, выходящие за рамки программирования, то пользовательский интерфейс должен стимулировать эти интересы, а не сдерживать их.

В интерактивной системе в определенные моменты времени программа выводит пользователю необходимую ему информацию и запрашивает информацию у пользователя путем выдачи соответствующей подсказки. Затем программа должна находиться в состоянии ожидания, пока пользователь не введет всю необходимую информацию. Если программа способна проверять на достоверность введенную информацию, то после ввода данных пользователем она может либо продолжить свое выполнение, либо выдать сообщение об ошибке и ожидать ввода новых значений данных. Если пользователь не понимает, что от него требуется, то он может запросить у программы справочную информацию. Умная программа располагает множеством полезных сообщений, которые выдаются пользователю, если он запрашивает справочную информацию, и разъясняют ему, что от него требуется.

Форматы ввода данных могут быть различными, поскольку существует много различных классов данных. Эти классы данных могут быть в общем случае разбиты на две категории: команды и значения данных.

Команда может быть простой, как, например, нажатие одной из функциональных клавиш, которая в системах текстовой обработки обозначает переход к новому параграфу. Команда может быть сложной ,как, например, загадочный набор букв, цифр и символов, что характерно для многих команд MS-DOS. Некоторые команды запрашиваются программой, как, например, ввод ответов "Y" и "N", когда ПЭВМ переспрашивает, действительно ли вы хотите того, что вы от нее требуете. Другие команды вводятся по инициативе пользователя, когда, например, вы просите систему текстовой обработки сохранить документ в дисковом файле. Программа на ПЭВМ не ожидает ввода именно этой команды, но, тем не менее, принимает ее и выполняет то, что от нее требуется. Иногда программа может выводить перечень допустимых команд, из которого вы можете выбрать любую. Этот перечень называется меню.

Значения данных в свою очередь можно разделить на элементы данных и текстовую информацию. Элементами данных являются данные определенного формата и назначения. Это, например, значения даты, имена, адреса, числовые значения, размер одежды, оттенки цвета. Проверка достоверности вводимых данных для определенного элемента данных может быть произведена по соответствию их формата, длины и значения этому элементу данных. В противоположность этому текстовая информация не имеет определенного формата, длины и значения. Системы баз данных обычно манипулируют элементами данных, а системы обработки текстовой информации- соответственно текстовой информацией.

Интерактивная система должна обладать пользовательским интерфейсом, который облегчает использование клавиатуры и экрана для ввода данных различного типа, которые затем обрабатываются программой. IBM PC имеет такую архитектуру видеосистемы и клавиатуры, которая обеспечивает возможности для создания пользовательских интерфейсов различного типа, что уже реализовано во многих пакетах программ для IBM PC.

Есть различные способы отображения меню, выдачи информационных сообщений, сообщений об ошибках и запросов на ввод данных. Также существуют различные способы ввода текстовой информации и значений данных, поступающих от пользователя. При разработке языка взаимодействия с пользователем программист может выбирать из существующего разнообразия способов. Выбор инструментальных программных средств будет оказывать влияние на конечный программный продукт. Использование инструментальных программных средств, обеспечивающих эффективный ввод и отображение данных, позволяет создавать эффективные программные системы.

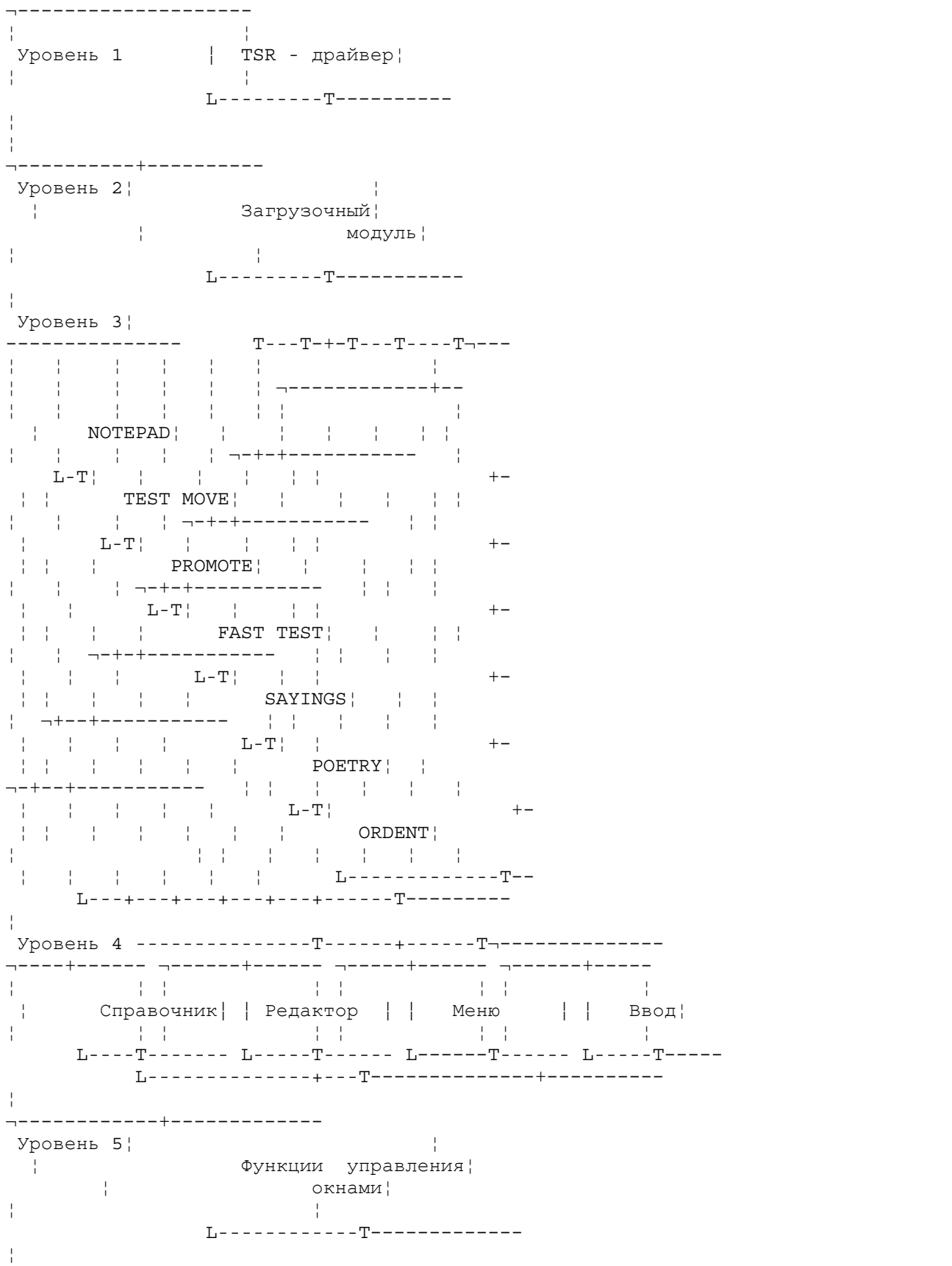
Одним из наиболее популярных в настоящее время способов организации взаимодействия с пользователем является работа с окнами. Окном называется, как правило, прямоугольная область на экране дисплея с видимой границей, изображение в которой формируется независимо от остальной части экрана. Окна используются для всех типов взаимодействия с пользователем: для отображения меню, в качестве областей для ввода значений данных или текстовой информации, для вывода сообщений и справочной информации по требованию пользователя.

Другим важным свойством интерактивных систем, существенно определяющим их качество, является обеспечение для пользователя возможности быстро переходить от одной задачи к другой без утомительных выходов в операционную систему. В интегрированной программной системе такого рода переходы зачастую обеспечиваются операционной средой, в которой исполняется программа. Однако, при увеличении числа независимых задач, для которых необходимо обеспечить в асинхронном режиме быстрый переход от одной задачи к другой, возможности однозадачной операционной системы DOS для IBM PC могут быть превышены. Для обеспечения возможности переключениязадач в этом случае должны использоваться резидентные в памятипрограммы. Эти программы не обеспечивают настоящего мультизадачного режима, но позволяют установить удобный для пользователя режим использования некоторых утилит в командной среде DOS.

Программы из этой книги образуют библиотеку программных инструментальных средств, использующих окна для ввода текстовых и числовых данных, выдачи справочной информации пользователю и организации меню. Эти программы могут быть сделаны резидентными. Программные модули библиотеки написаны для компилятора Турбо Си и предназначены для использования программами, также написанными для Турбо Си. Для того чтобы использовать программы из этой книги, вы должны иметь общее представление о DOS и о тех средствах, которые она предоставляет программисту. В книге разбираются вопросы внутренней организации DOS, а также некоторые из ее функций, обеспечивающие резидентность программ и по которым нет документации. Отличным справочным руководством по программным средствам DOS и ROM-BIOS является книга "Advanced MS-DOS" Рэя Дункана (Microsoft Press, 1986). Данная же книга содержит только сведения, необходимые для изложения вопросов, касающихся предлагаемой программистам библиотеки. Дункан с иронией представляет свою книгу, как нечто облегченное, но вы должны отнестись к ней со всей серьезностью.

Программное обеспечение из этой книги может быть представлено в виде шести уровней, как на рисунке 1.1. Уровни на диаграмме располагаются сверху вниз, но изложение материала будет

соответствовать движению от нижних уровней к верхним.



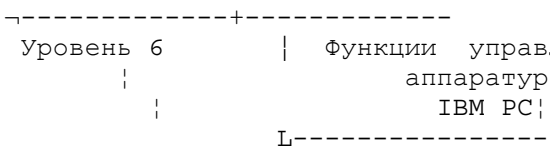


Рисунок 1.1. Уровни программного обеспечения.

Уровень 6 представляет библиотеку функций нижнего уровня, которые управляют определенными действиями IBM PC.

Уровень 5 представляет библиотеку функций для работы с окнами, которые управляют размещением, отображением на экране и сохранением в памяти экранных окон.

Уровень 4 объединяет функции, которые используют окна для конкретных применений. Эти функции управляют контекстно-чувствительными окнами со справочной информацией, окнами редактирования текста, системами меню и окнами для ввода данных по формату.

Уровень 3 соответствует прикладным программам и представлен здесь примерами программ, иллюстрирующих применение библиотек нижнего уровня.

Уровень 2 представляет собой управляющую программу, которая связывает прикладные программы предыдущего уровня в единую программу, управляемую с помощью меню. Эта программа может работать как автономно, так и в качестве резидентной утилиты на последующем уровне программного обеспечения.

Уровень 1 представляет драйвер TSR, который строит из транзитных модулей программы, остающиеся резидентными после завершения выполнения. В данном примере он используется для объединения программ предыдущих уровней в единую резидентную утилиту.

Перед тем как погрузиться в глубины программирования модулей, резидентных в памяти и работающих с окнами, возможно, вам захочется почитать что-нибудь по языку Си. Вы можете найти интересующие вас сведения в данной книге. Раздел 2 содержит краткое изложение истории языка Си и попытку объяснения, почему же программисты так любят этот язык. Раздел 3 продолжает рассмотрение компилятора Турбо Си, выбранного для изучения в данной книге.

ГЛАВА 2

Язык Си

Любая книга по Си превозносит этот язык и рассказывает историю его создания. Настоящий раздел следует этой традиции: в нем приводится краткая хронология создания Си, описываются фундаментальные особенности языка, рассказывается о достоинствах Си как инструмента программирования, а также предпринимается попытка объяснить, почему же он завоевал такое широкое признание среди программистов. В конце раздела содержится список справочных материалов для изучения языка Си.

Краткая история языка Си

Язык Си был создан в начале 70-х годов Дэннисом Ритчи, который работал в компании Bell Telephone Laboratories. Родословная языка Си берет свое начало от языка Алгол и включает в себя Паскаль и ПЛ/I.

Си был разработан как язык для программирования в новой по тем временам операционной системе Unix. ОС Unix была написана на языке ассемблера для ЭВМ PDP-7 и перенесена затем на PDP-11. На язык Си оказал значительное влияние его предшественник, язык Би, созданный Кэном Томпсоном, который в свою очередь является последователем языка BCPL. Язык BCPL был создан в 1969 г. Мартином Ричардсом в рамках проекта "Комбинированный язык программирования" в Кэмбриджском университете в Лондоне. Вскоре Unix была переписана на языке Си, и в 1974 - 75 годах ОС Unix фирмы Bell Laboratories стала первым коммерческим продуктом, реализующим идею о том, что операционная система может быть успешно написана на языке высокого уровня, если этот язык является достаточно мощным и гибким.

В 1978 г. Брайан Керниган и Дэннис Ритчи написали книгу "Язык программирования Си" (издательство Prentice-Hall). Эта работа, которая в своем кругу называлась "белой книгой" и "K & R" в остальном мире, стала стандартом описания языка Си. На момент создания "K & R" существовали компиляторы языка Си для ЭВМ PDP-11, Interdata 8/32, Honeywell 6000 и IBM 370. В дальнейшем этот список был продолжен.

В конце 70-х начали появляться трансляторы Си для микроЭВМ на процессорах 8080 и Z80 с операционной системой CP/M. Скотт Газери и Джим Гибсон разработали и пустили в продажу Tiny-C ")Крошечный Си") - интерпретатор, основанный на подмножестве языка Си. Его интерактивная среда программирования очень похожа на ту, что имеет чрезвычайно популярный транслятор Basic фирмы Microsoft. В 1980 г. Рон Кэйн создал свой компилятор Small-C ")Малый Си") для ОС CP/M и микропроцессора 8080. Компилятор Small-C, основанный на подмножестве языка Си, был написан на самом Small-C. Проблема курицы и яйца была решена, когда Кэйн создал первую версию компилятора на основе интерпретатора Tiny-C. Затем Small-C методом раскрутки создал самого себя, когда Кэйн вместе с другими, используя ранние версии компилятора, сделал более совершенный компилятор. Small-C компилирует исходный модуль на языке Си в модуль на языке ассемблера процессора 8080. Следует отметить, что Кэйн предоставил свой компилятор и его исходный текст в общественную собственность.

Примерно в это же время Лео Золман представил свой компилятор BDS-C для CP/M, также основанный на подмножестве языка Си. Достоинствами этого компилятора были высокая скорость и возможность совместной компоновки перемещаемых объектных модулей в загрузочном модуле.

Вскоре после BDS-C были созданы компиляторы, предназначенные для CP/M и основанные на полном множестве языка Си. Это дало импульс развитию программирования на Си для микроЭВМ. В 1981 г., в связи с созданием IBM PC, в мире микроЭВМ был сделан значительный скачок вперед.

После появления IBM PC стали появляться и компиляторы Си для этой ПЭВМ. Некоторые компиляторы были получены путем преобразования соответствующих компиляторов для процессора 8080, другие были разработаны специально для IBM PC. В настоящее время на рынке представлены по меньшей мере семнадцать компиляторов языка Си для IBM PC.

В 1983 г. Американский Институт Стандартов (ANSI(сформировал Технический Комитет X3J11, устав которого предусматривает создание стандарта языка Си. Стандартизация будет распространяться не только на язык, но и на программную среду компилятора, а также на библиотеку стандартных функций. В работе комитета участвуют представители основных фирм - поставщиков компиляторов Си, в том числе и для IBM PC, а также многие другие светила из мира программирования на языке Си. Усилия комитета X3J11 привлекли внимание средств массовой информации. Предлагаемый стандарт был опубликован, для того чтобы все заинтересованные стороны могли ознакомиться с ним и внести свои предложения. (Сомнительно, что выдающийся программист заинтересуется языком, который создан комитетом, но, видимо, комитет делает хорошее дело, совершенствуя язык, созданный выдающимся программистом.(

Поскольку большинство поставщиков компиляторов для IBM PC участвуют в работе комитета X3J11, то разрабатываемые ими новые версии компиляторов будут в рамках этого стандарта. (Турбо Си, один из последних компиляторов для IBM PC, подчиняется большинству требований стандарта на язык и библиотеку.(

Особенности языка Си

В данной книге не ставится цель научить вас программировать на языке Си, но она может быть полезной для понимания тех особенностей Си, которые заставляют столь многих программистов остановить свой выбор именно на этом языке.

Си является языком функций, типов данных, операторов присваивания и управления последовательностью вычислений. Программируя на Си, вы осуществляете обращение к функциям, и большинство функций возвращают некоторые значения. Значение, возвращаемое функцией, будь то значение переменной или константа, может использоваться в операторе присваивания, который изменяет значение другой переменной. Дополненный операторами управления последовательностью вычислений (while, for, do, switch), Си превращается в язык высокого уровня, способствующий хорошему стилю программирования.

Си имеет небольшой набор типов данных: целые числа, числа с плавающей запятой, битовые поля и перечислимый тип. В языке Си вы можете описать переменную типа указатель, который связывается с объектом, принадлежащим к любому типу данных. Адресная арифметика языка Си является чувствительной к типу данных того объекта, с которым связан используемый указатель. Разрешены также указатели к функциям. Вы можете расширить список типов данных путем создания структур с иерархической зависимостью входящих в него типов данных. Каждый тип данных может принадлежать либо к основному типу, либо к ранее описанному структурному типу. Объединения напоминают структуры, но определяют различные виды

иерархических зависимостей, в которых данные разных типов располагаются в памяти.

Допустимо описание массивов данных различных типов, включая структуры и объединения. Массивы могут быть многомерными.

Функции Си являются рекурсивными по умолчанию. Вы можете, правда, создать функцию, которая не будет рекурсивной, но сам язык по своей природе стремится поддерживать рекурсивность и требует минимальных усилий при программировании рекурсий.

Программа функции на языке Си разбивается на блоки, в каждом из которых могут быть определены свои собственные локальные переменные. Блоки могут выбираться для исполнения по результату выполнения оператора управления последовательностью вычислений. Блоки могут быть вложенными друг в друга.

Переменные и функции могут быть глобальными для программы, глобальными для исходного модуля или локальными для блока, в котором они описаны. Локальные переменные могут быть описаны таким образом, что они будут сохранять свои значения при всех обращениях внутри данного блока (статические переменные) или же будут восприниматься как новые объекты при каждом обращении)автоматические переменные.(

Си позволяет создавать программу в виде нескольких исходных модулей, которые будут транслироваться независимо. Перемещаемые объектные модули, соответствующие исходным модулям, компонуются в единый загрузочный модуль. Эта особенность позволяет компилятору поддерживать объектные библиотеки многократно используемых функций и создавать большие программы из множества небольших исходных модулей.

В языке Си нет операторов ввода/вывода, весь ввод/вывод выполняется с помощью функций. Вследствие этой особенности языка Си разработана стандартная библиотека функций. Существование этого стандарта и составляет главную привлекательность языка Си, ибо делает программы на Си переносимыми.

Достоинства языка Си

Переносимость программ, написанных на Си, является наиболее разрекламированным преимуществом этого языка. Если вы пишете программу на Си и избегаете при этом использовать расширения библиотеки, зависящие от конкретного компилятора, или машинно-зависимые операции, то вы получаете неплохие шансы)значительно большие, чем при любом другом языке) на успешный перенос вашей программы в другую программно-аппаратную среду, включая смену компилятора, операционной системы и ЭВМ.

Приведенные в данной книге программы не претендуют на переносимость. Библиотека функций для работы с окнами привязана к архитектуре видеопамяти IBM PC, а функции резидентных в памяти утилит используют систему прерываний IBM PC, специфические особенности DOS и библиотеку расширений компилятора Турбо Си, которая облегчает разработку программ обработки прерываний. Некоторые функции включают небольшие фрагменты на языке ассемблера.

Расширяемость языков программирования означает существование

потенциальной возможности внести добавления в язык. Си рассчитан на расширение по своему замыслу, поскольку содержит очень небольшое число операторов. Следует помнить, что сам язык позволяет немногим более, чем изменять значения переменных и управлять последовательностью выполнения программы. Самое важное в программах на Си заключено в функциях, а язык сам по себе не имеет другие внутренние функции, кроме основной функции (функции main). Первая группа расширений языка Си размещается в стандартной библиотеке, другие нестандартные расширения поддерживает конкретный компилятор и, наконец, третья группа расширений содержится в дополнительных библиотеках функций Си)как это описано в данной книге). Последняя группа расширений разрабатывается самим программистом, который создает программы для многократного использования.

Кроме своей функциональной расширяемости, Си позволяет расширять стандартный набор типов данных путем определения структур, объединений и использования операторов typedef.

Программистам особенно нравится краткость выражений, которыми в Си кодируются алгоритмы. Большинство операторов, будь то операторы присваивания, условные операторы, обращения к функциям или выражения, кроме операторов управления последовательностью выполнения программы, возвращают некоторые значения. Использование этой особенности языка позволяет представлять выражения в краткой форме.

Си обеспечивает формирование эффективного машинного кода программы, что достигается привязкой языков программирования к структуре памяти и регистровой архитектуре ЭВМ, для которых они создаются. Си часто характеризуется как переносимый язык ассемблера высокого уровня. Сама природа языка позволяет компилятору генерировать эффективный оптимизированный машинный код.

Одобрение языка Си

Не должно оставаться сомнений по поводу преимуществ языка Си по сравнению с другими языками программирования. Программисты любят Си по причинам, указанным выше. Компании по разработке программного обеспечения любят его за то, что он позволяет писать программы, не зависящие от конкретной аппаратуры и операционной системы. Современные менеджеры, многие из которых в прошлом программисты, учитывают оба этих преимущества. Си является языком, на котором написано большинство наиболее популярных в мире пакетов программ.

Рекомендуемая литература по Си

Ниже приводится список литературы по Си, которая может помочь больше узнать о языке и способах его использования:

Брайан В., Керниган, Ритчи Деннис М.. Язык программирования Си. - Prentice-Hall, 1978.

Плам Томас. Стандарты и руководящие принципы

программирования на Си. - Plum-Hall, 1982.

Плам Томас. Изучение программирования на Си. - Plum Hall, .1983

Кочан Стефен Г. Программирование на Си. - Hayden Book Company, 1983.

Пардам Джек. Руководство по программированию на Си. - Que Corporation, 1983.

Харбисон Самюэл, Стил Гай Л. Си: справочное пособие- . Jr.Tartan Laboratories, 1984.

Хоган Том. Руководство для программистов по Си- . Brady,1984.

Хант Вильям Джеймс. Набор инструментальных средств на Си- . Addison-Wesly, 1985.

Плам Томас. Структуры данных в Си. - Plum Hall, 1985.

Компилятор MIX C. - Mix Software, Inc, 1985.
) Этот материал продается вместе с компилятором и отдельной книгой .Независимо от того, используете вы компилятор или нет, руководство и справочное пособие по Си будут для вас полезны.(

Газери Скотт Б. Изучение Си и Tiny-C. - Tab Books, 1985.

Рэдклифф Роберт А., Рааб Томас Д. Утилиты обработки данных на Си. - Sybex, 1986.

Стивенс Ал. Разработка инструментальных средств на Си для IBM PC. - Brady, 1986.

Пособие по инструментальным средствам Си доктора Добса- . Brady, 1986.

Стивенс Ал. Разработка баз данных на Си. - MIS:Press,1987.

Джонсон Нельсон. Усовершенствованная графика на Си: программирование и методы. - Osborne / McGraw-Hill, 1987.

" Наставник по Си" дискеты #577 и #578 библиотеки PC-SIG)Этот пакет программ представляет собой обучающую интерактивную систему, содержащую текстовую информацию и примеры программ, которые вы можете транслировать с помощью своего транслятора.(

ГЛАВА 3

Компилятор Турбо Си

В декабре 1986 г. небольшая компания под названием Wigard Software Systems, Inc. объявила о своем переезде из города Армингтон, штат Массачусетс, в город Монте Серено, штат Калифорния. Эта компания создала и начала продажу компилятора Си стоимостью 450 долларов под названием Wizard C.

Wizard C был компилятором, заслуживающим уважения, всегда получающим хорошие отзывы в обзорах и даже названный, по крайней мере одним из обозревателей, "лучшим" компилятором Си. Его достоинства заключались в высокой скорости компиляции, эффективной оптимизации получаемого кода, соответствии предложениям стандарта ANSI и большом числе расширений языка, позволяющих разрабатывать программы обработки прерываний. Эти расширения заключали в себе функцию прерывания специального типа, возможность встраивания в тело программы фрагментов на языке ассемблера, псевдопеременные, с помощью которых из языка Си выполняется доступ к регистрам микропроцессора.

В феврале 1987 г. фирма Borland International из Скоттс-Веллей, штат Калифорния, объявила о создании компилятора Турбо Си, который ожидался с нетерпением после появления его предшественника, очень удачного компилятора ТурбоПаскаль. Объявление содержало оценку эффективности, заверенную несколькими экспертами. Они утверждали, что скорость компиляции Турбо Си будет достигать 7000 строк в минуту, что превышало скорость самого быстрого на то время компилятора Си. По мнению экспертов было достигнуто предельное значение производительности для компиляторов Си и не ожидалось появление компилятора, который превзошел бы это значение.

В том же месяце Wizard сделала свое последнее объявление. В мае 1987 г. появилась версия 1.0 Турбо Си (совместно с T-shirts.(Перчатка была брошена и соревнование объявлено. Турбо Си подавил всех своими характеристиками.

На самом деле фирма Borland International приобрела фирму Wigard Systems для того, чтобы создать Турбо Си. Промышленность делала предположения о том, когда Borland выйдет на рынок компиляторов Си, после того, как она уже представила свои изделия: Турбо-Паскаль, а вслед за ним очень популярный Турбо-Бейсик. Вместо того, чтобы предпринимать большие усилия, начиная с нуля, фирма Borland приняла мудрое решение: она купила лучший компилятор Си и сконцентрировала усилия на том, чтобы сделать его еще лучше.

На момент анонсирования фирмой Borland своего компилятора на рынке было представлено 17 компиляторов Си для IBM PC. Несколько человек решили, что миру нужен еще один. Таковы краткие сведения о фирме Borland International Филиппа Кана, которые тем не менее приковывают внимание и захватывают воображение. После опубликования информации о Турбо Си многие сомневались, немногие имели представление о нем ,но все жаждали увидеть его собственными глазами. Мир Си готов был принять еще один компилятор, при условии, что он поступит от Borland. Само это состояние ожидания имело очень большое значение. Фирма Borland начала дело не для того, чтобы создать очередной компилятор Си; Borland поставила целью изменить представления о том, как должна выглядеть программная среда для разработки программ на языке Си.

Два Турбо Си

Турбо Си обозначает два программных изделия: пакет программ, обеспечивающий выполнение последовательности команд в стиле Unix: make/compiler/linker, и интегрированную программную среду для разработки программ.

Пакет программ содержит утилиту make, компилятор tcc и настраиваемый компоновщик. Последующие версии несомненно будут включать объектную библиотеку. Компилятор, входящий в пакет, похож на большинство других компиляторов Си для IBM PC, но

является более быстрым. Программисты, которые предпочтут этот пакет Турбо Си, найдут все, что им нужно, включая удобный редактор. Поскольку вы приобрели эту книгу, то, вероятно, вы уже имеете или собираетесь приобрести Турбо Си. Все, что Вам нужно знать по этому пакету, содержится в руководстве пользователя и справочном руководстве.

Интегрированная программная среда представляет собой программу под названием tc, которая объединяет в себе текстовый редактор, ориентированный на создание текстов программ на языке Си, построитель задач, ориентированный на реализацию программного проекта, и утилиты исполнения программ. В будущем планируется включение символьного отладчика. Наличие интегрированной среды выделяет Турбо Си среди конкурентов (также, как и ее безусловно блестящая реализация.(

Интегрированную среду можно считать витриной Турбо Си. Ее большим достоинством является достигнутый уровень интеграции между редактором, компилятором и компоновщиком. Находясь в интегрированной среде, программист может редактировать программу, транслировать ее, компоновать ее с другими исходными модулями и библиотеками и запускать на выполнение. Данное качество является основным для нового поколения компиляторов Си. Это похоже на то, чего фирма Borland достигла тремя годами раньше на компиляторе Турбо-Паскаль, но чего не было достигнуто до этого времени на компиляторе Си. Ожидается, что основные конкуренты в ближайшем будущем достигнут подобного уровня.

Настройка интегрированной среды

Вы имеете возможность настроить интегрированную среду, в которой все, от цвета изображения на экране до уровня контроля за ошибками, может быть установлено по вашему требованию. Некоторые установки производятся при выполнении программы TCINST, другие- путем использования меню, создаваемых интерактивной системой ввода интегрированной среды и возникающих в верхней части экрана. Ниже приводится список параметров, значения которых могут устанавливаться по требованию заказчика:

- модель памяти: крошечная, малая, средняя, компактная, большая, огромная;
- соглашение о вызываемых функциях: Си или Паскаль;
- микропроцессор: 8088/8086 или 80186/80286;
- плавающая арифметика: отсутствует, сопроцессор или эмуляция;
- уровень оптимизации;
- уровень контроля за ошибками.

Вы можете выбирать и большее число параметров Турбо Си. Турбо Си способен осуществлять строгий контроль за ошибками и подозрительными местами в программе и выдавать предупреждающие сообщения. Вы можете использовать Устанавочное Меню для

подавления предупреждающих сообщений. Вы можете установить необходимость соответствия жестким требованиям ANSI или менее жестким требованиям стандарта, изложенного в книге " K & R". Вы можете потребовать выдачи предупреждающего сообщения при любом несоответствии описания функции и прототипа или можете разрешить неявное описание функции и определение случайных параметров, как это делается в так называемых K & R-компиляторах.

Возможно, вам захочется изменить цвета изображений на экране. Вы можете выбрать один из трех цветовых наборов, включая цветовой набор по умолчанию (слишком ярок), бирюзовый набор)неприятен) или малиновый набор (просто ужасен). Не отчаивайтесь, программа TCINST позволит выбрать цвет и яркость для каждого отдельно определяемого компонента интегрированной среды. Следует помнить, что в данном обзоре имеются в виду цветовые наборы, формируемые системой CGA. Вполне возможно, что они вам и нравятся.

Редактор Турбо Си

В первом приближении редактор Турбо Си похож на редактор системы WordStar, работающий в режиме, альтернативном к режиму "документ". Архитектура этого редактора характерна для многих других программных изделий Borland, включая программу Sidekick Notepad и редактор Турбо-Паскаля. Если вы умеете работать с редактором Турбо-Паскаля, то вы умеете работать и с редактором Турбо Си. Однако в случае приобретения именно этого редактора пользователи получат некоторые преимущества. Путем использования программы TCINST вы можете изменить размер окна по умолчанию и назначение клавиш команд редактирования. Программисты, которые раньше работали с другим редактором, оценят предоставляемую им редактором Турбо Си возможность работы с двумя наборами команд редактирования, что позволяет избежать многих затруднений. Диапазон изменения параметров редактора ограничен: определенные функциональные клавиши и комбинации различных клавиш с клавишей ALT зарезервированы под "горячие клавиши" и не могут быть задействованы под команды редактирования.

Редактор Турбо Си не такой мощный, как некоторые специальные программы редактирования, но вполне отвечает требованиям не слишком больших задач по редактированию. Редактор имеет неизменяемое значение интервала для клавиши табуляции, соответствующее восьми символьным промежуткам. Это неудобно при работе с исходными текстами программ, представленных в данной книге, так как их интервалы табуляции соответствуют четырем символьным промежуткам, что обусловлено ограничениями при печати. Borland поставляет программу PATCH.COM и несколько примеров "заплат" на программы Compuserve и BIX. Один из этих примеров позволяет вам поставить "заплату", которая устанавливает интервал табуляции на четыре символьных промежутка, в результате чего редактор становится очень удобным для программ из этой книги. Возможность изменения интервала табуляции, вероятно, будет предусмотрена в следующей реализации Турбо Си.

Компоновщик Турбо Си

Турбо Си имеет свой собственный компоновщик, который называется TLINK. Компоновщик используется для связывания различных объектных модулей, каждый из которых может быть получен путем трансляции с языков Си, ассемблера и других в единый загрузочный модуль. Объектные файлы, формируемые Турбо Си, соответствуют стандарту программы LINK DOS, поэтому они могут быть скомпонованы с объектными библиотеками для других языков, включая ассемблер. Основной причиной использования компоновщика TLINK является его скорость, поскольку TLINK работает значительно быстрее, чем компоновщик LINK DOS.

Утилита построителя задач (Make) в Турбо Си

Компилятор Турбо Си имеет утилиту Make, характерную для ОС Unix и других компиляторов Си для IBM PC. Интегрированная Среда дает уникальную возможность связывать при разработке программ исходные и объектные модули с соответствующими им загрузочными модулями. В этом отношении утилита Make Турбо Си является традиционной. Однако утилита Make Турбо Си является частью интегрированной среды и использует файл сопровождения, называемый "файлом проекта", который является более легким для чтения и понимания, чем у командной утилиты MAKE. В файле проекта перечисляются исходные модули, составляющие программу, по одному в каждой строке. Справа от имени каждого модуля можно указать другие файлы (например, заголовки), с которыми связаны исходные модули. Эти файлы заключаются в скобки и отделяются друг от друга запятыми. Ниже приводится пример записи файла проекта:

```
myprogram (keys.h, twindow.h(
```

Если версия модуля myprogram.c старше версии модуля myprogram.obj либо версии модулей key.h или twindow.h старше версии модуля myprogram.c, то модуль myprogram.c транслируется в модуль myprogram.obj. Если версия модуля myprogram.obj старше версии модуля myprogram.exe, то модуль myprogram.obj компонуется с соответствующими (зависящими от модели памяти) начальным объектным файлом и исполняющей библиотекой. Использование проектной утилиты Make становится насущно необходимым, когда при формировании загрузочного модуля используются многочисленные исходные модули на Си, зависящие от различных файлов заголовков.

Вы можете указывать объектные файлы и объектные библиотеки в проектном файле MAKE. Интегрированная среда будет включать объектные файлы без попытки их компиляции и будет отыскивать соответствующие библиотечные модули для разрешения вызовов внешних функций.

Обнаружение ошибок при компиляции и компоновке

При построении задачи в интегрированной среде Турбо Си производится запись всех сообщений об ошибках и предупреждениях. После завершения построения задачи сообщения об ошибках и предупреждениях выводятся в одно окно, в то время, как исходный текст программы отображается в другом окне. Вы имеете возможность перемещаться по файлу с исходным текстом программы вперед и назад, от одной ошибки к другой. Интегрированная среда отображает

каждое сообщение об ошибке и устанавливает курсор в строке, в которой ошибка была обнаружена. Вы можете внести исправления, какие считаете нужными, и снова запустить процесс построения задачи. Программа обнаружения ошибок следит за тем, удаляете или добавляете вы строки в исходном модуле, и соответствующим образом корректирует положение курсора.

Программные средства низкого уровня

Турбо Си включает несколько расширений языка Си, не обладающих свойством мобильности, доставшиеся в наследство от Wizard, но которые являются весьма существенными для программного обеспечения, представленного в данной книге. Эти расширения содержат программы обработки прерываний и других операций низкого уровня.

Расширения языка включают функцию типа прерывание, при вызове которой производится сохранение регистров процессора 8086 и установка регистра сегмента данных на значение сегмента данных для функции прерывания. Перед тем, как функция возвращает управление вызывающей программе, содержимое регистров восстанавливается. Возврат осуществляется с помощью команды IRET процессора 8086, которая используется для возврата из прерываний.

При включении в программу на Си фрагментов на ассемблере используется ключевое слово asm. Все, что следует после этого ключевого слова, поступает непосредственно на обработку транслятором с ассемблера фирмы Microsoft, который вы должны иметь, чтобы использовать данную возможность. Включаемые ассемблерные фрагменты могут использовать имена переменных из программы на Си. Эта возможность позволит писать функции на ассемблере, которые не будут зависеть от используемой модели памяти. Программы без ассемблерных фрагментов не подвергаются обработке транслятором с ассемблера, а только компилятором с языка Си. Программа с ассемблерными фрагментами должна компилироваться обязательно командным компилятором tcc, поскольку компилятор tc интегрированной среды не допускает использования ассемблерных фрагментов. Фирма Borland планирует убрать это ограничение в последующих реализациях.

Несколько ключевых слов используется в качестве псевдопеременных для осуществления непосредственного доступа к регистрам ЭВМ. Если вам известно, что содержат регистры и как это можно использовать, то вы можете оптимизировать выполнение некоторых операций. Будьте внимательны при использовании этого средства. Наилучшим подходом является трансляция исходной программы на Си в модуль на ассемблере (что достигается использованием ключа- S в командной строке компилятора tcc), а затем внесение изменений в полученный модуль. При переходе на следующие версии Турбо Си вы должны проверить возможность использования каждого из описанных средств. Нет уверенности в том, что Borland не изменит способ доступа к регистрам, и это изменение может сделать ваши программы неработоспособными.

Начальная установка

Самым слабым местом документации по Турбо Си является раздел по начальной установке. Есть ряд фактов, которые обязательно необходимо знать, а структура руководства такова, что не

позволяет легко найти нужную информацию.

И командный компилятор, и интегрированная среда используют специальные файлы, в которых пользователь описывает требуемую ему конфигурацию. Каждому из компиляторов соответствует свой файл. Руководство дает подробные инструкции по подготовке файла конфигурации TURBOC.CFG для командного компилятора и явно недостаточную информацию по файлу конфигурации TCCONFIG.TC для интегрированной среды. После того, как вы произвели начальные установки в интегрированной среде, вы должны запустить процесс инсталляции еще раз и установить значения параметров по умолчанию, включая путь доступа, по которому компилятор tc будет искать библиотеки, стартовую программу, включаемые файлы и себя самого.

Эти параметры могут быть установлены также из интегрированной среды путем выбора меню Options (Параметров.(После установки всех параметров такими, как вы хотели, выберите в меню строку "Запомнить", завершая тем самым создание файла TCCONFIG.TC.

Модели памяти

Турбо Си поддерживает шесть моделей памяти: крошечную, малую, среднюю, компактную, большую и огромную. Руководство пользователя содержит раздел, посвященный моделям памяти и разъясняющий сегментную организацию памяти для процессора 8086 и ее проявления в различных моделях памяти. Советуем вам прочитать и осмыслить этот раздел, поскольку понимание архитектуры процессора 8086 позволит использовать и модифицировать резидентные в памяти программные утилиты, представленные в данной книге.

Библиотека исходных модулей

Турбо Си поставляется без исходных текстов программ библиотеки функций исполняющей системы. Но каждый пользователь Турбо Си может купить лицензию на использование исходных текстов, которые в этом случае поставляются ему фирмой Borland.

Заключение

Раздел 4 начинает описание библиотеки функций Турбо Си. После прочтения этого и последующих разделов вы получите в свое распоряжение инструментальные программные средства, необходимые для создания вашими программами всплывающих окон и последующего преобразования этих программ в резидентные утилиты. На настоящий момент нет другого компилятора, который поддерживал бы эти средства на таком же уровне, что и Турбо Си.

ГЛАВА 4

Функции общего назначения

Эта книга посвящена программному обеспечению, и следующие разделы содержат набор инструментальных программных средств, которые могут быть использованы при создании прикладных систем. Эти инструментальные средства написаны на языке Си, транслируются с помощью компилятора Турбо Си и готовы к тому, чтобы быть включенными в ваши программы. Эти функции могут рассматриваться как расширения языка Си, если будут присоединены к и без того достаточно обширной библиотеке стандартных расширений Турбо Си. Настоящий раздел описывает функции первого уровня, которые необходимо включать в программы, использующие эти библиотеки. Представленные в данном разделе функции являются функциями общего назначения, выполняющими операции низкого уровня (специфическими для аппаратуры IBM PC) по управлению дисплеем и клавиатурой.

Вы можете посчитать недостаточно обоснованным применение некоторых из этих функций в своих программах. Назначением этих функций является поддержка функций библиотеки высокого уровня, также рассматриваемых в этой книге. В этом смысле они полезны, и вы можете найти для них применение. Кроме того, глубина вашего понимания функций, представленных в книге, зависит от осмысления вами всех функций, в том числе и тех, которые вы не будете использовать в своих программах.

При чтении описаний этих функций обращайтесь к листингу 4.1 программы ibmpc.c, который приводится после описаний.

void clear_screen()

Эта функция очищает экран и устанавливает курсор в левый верхний угол. Экран заполняется символами пробела, и атрибуты символов извлекаются из символьной переменной attrib, описанной в программе ibmpc.c. Значение этого атрибута соответствует байту атрибута в видеопамяти, сопутствующему каждому байту ASCII-кода при записи символа в видеопамять. Более подробная информация по этому вопросу содержится в разделе 5. Переменная attrib принимает значение, которое соответствует установке черного цвета для фона символа и белого цвета для самого символа. Если вы желаете другое значение атрибута, то должны изменить значение этой переменной перед вызовом функции clear_screen.

int vmode()

Эта функция возвращает код текущего режима системы формирования изображения. Она прежде всего предназначена для определения того, как программе интерпретировать содержимое видеопамяти: как содержимое видеопамяти в монохромном режиме или в алфавитно-цифровом режиме для Цветного Графического Адаптера)CGA) и для Усовершенствованного Графического Адаптера (EGA). Эти устройства более подробно рассматриваются в разделе 5. Функция vmode возвращает код 7, если IBM PC работает в монохромном режиме. Любое другое значение обозначает алфавитно-цифровой режим для контроллеров CGA и EGA.

void cursor(int x,int y(

Эта функция устанавливает курсор в позицию на экране, определяемую координатами X и Y .Координаты (0,0) соответствуют левому верхнему углу экрана. Значение координаты X изменяется в диапазоне от 0 до 79, а координаты Y - в диапазоне от 0 до 24.

void curr_cursor(int *x, int *y(

Эта функция считывает текущее положение курсора и записывает значения координат X и Y в адреса памяти, определяемые с помощью указателей перед обращением к функции.

int set_cursor_type(int t(

Эта функция устанавливает текущий размер курсора, интерпретируемый программами из этой книги как тип курсора. Размер обозначается целочисленной переменной, которая содержит в старшем байте номер начальной растровой линии курсора и в младшем байте - номер конечной растровой линии курсора.

Редактор текстов и программа ввода данных, представленные в следующих разделах, используют возможность изменения размера курсора для обозначения того, какой из режимов установлен: Вставки или Замены. Курсор прямоугольной формы обозначает, что установлен режим Вставки, он определяется значением переменной, равным 0x0106. При этом курсор занимает растровые линии с 1 по 6 того знакоместа, в котором он находится. Курсор в виде знака подчеркивания обозначает режим Замены и определяется значением переменной, равным 0x0607. При этом курсор занимает растровые линии 6 и 7 знакоместа.

int get_char()

Эта функция является очень важной, так как выполняет несколько крайне необходимых действий в вызывающих ее программах. get_char принимает поступающий от клавиатуры символ путем использования программ ROM-BIOS IBM PC. Ее главное назначение состоит в приеме одиночного символа от клавиатуры без эха, без преобразования и без обращения к функциям DOS. Кроме того, она выполняет следующие дополнительные функции.

В то время, как система ожидает нажатия клавиши ,функция get _char вызывает программные прерывания по вектору 0x28, так называемые прерывания DOSOK. Это прерывание и его значение для создания резидентных утилит более подробно рассматриваются в разделе 11.

Когда вы нажимаете функциональную клавишу, ROM BIOS возвращает двубайтный код. Первый байт имеет нулевое значение и обозначает, что следующий за ним код символа соответствует функциональной клавише. Этот второй байт содержит 7-битный ASCII-код, который является уникальным для каждой функциональной клавиши. Если не учитывать первый нулевой байт, то реакция на нажатие функциональных клавиш сходна с реакцией на нажатие

клавиш, соответствующих ASCII-символам, в частности, буквам.

Функция get_char преобразует двубайтную последовательность, возвращаемую ROM-BIOS в ответ на нажатие функциональной клавиши, в 8-битный код, позволяющий отличать функциональные клавиши от нефункциональных. Функция осуществляет это преобразование путем установки старшего разряда в байте, содержащем ASCII-код символа и следующем за нулевым байтом. Формируемые коды описыватся в исходном файле keys.h как глобальные символы (см. листинг 4.2.(

Функция get_char ожидает нажатия функциональной клавиши, обозначенной Help. Код, соответствующий функциональной клавише Help, присвоен целочисленной глобальной переменной, названной helpkey. Первоначально этой переменной присваивается нулевое значение, но программные средства, которые работают с окном Help, предназначенным для отображения справочной информациии, будут присваивать этой переменной значение, соответствующее функциональной клавише. При нажатии функциональной клавиши Help функция get_char проверяет значение глобального указателя функций, названного helpfunc. Если указатель имеет ненулевое значение, то функция get_char вызывает адресуемую с помощью указателя helpfunc функцию выдачи справочной информации.

void vpoke(unsigned vseg,unsigned adr,unsigned chr) int vpeek (unsigned vseg,unsigned adr(

Эти две функции считывают из видеопамяти коды символов и атрибуты символов и записывают их в видеопамять. Для того, чтобы использовать эти функции, вы должны разобраться в организации видеопамяти IBM PC, а также принципах формирования изображения. В тело функций vpoke и vpeek включены фрагменты на ассемблере. Для того, чтобы оттранслировать эти функции, вы должны иметь программу Macro Assembler (MASM) фирмы Microsoft, поскольку именно она используется в Турбо Си для трансляции ассемблерных фрагментов. Включение ассемблерных фрагментов необходимо только в том случае, если ваши программы работают в системах, использущих Цветной Графический Адаптер (CGA) или совместимый с ним адаптер. Смысл этого требования разъясняется в разделе 5. Если у вас нет контроллера CGA или если вы хотите работать с функциями, не используя ассемблер, то удалите эти функции из исходного модуля ibmpc.c и вставьте в файл twindow.h из раздела 6 следующие операторы:

```
#       define vpoke(vseg,adr,chr) poke(vseg,adr,chr(
#       define vpeek(vseg,adr) peek(vseg,adr(
```

Эти макроопределения заменят функции vpoke и vpeek и избавят от необходимости использования макроассемблера для функций работы с окнами из этой книги.

Исходные модули функций общего назначения

Листинг 4.1 представляет собой исходный текст программы ibmpc.c, которая содержит функции, описанные в данном разделе. Вследствие того, что функции включают фрагменты на ассемблере, программу лучше транслировать с помощью командного компилятора

tcc, а не компилятора tc Интегрированной Среды.

Чтобы оттранслировать файл ibmpc.c, введите следующую команду (не набирая промптер C:(<

```
C>tcc -c ibmpc
```

Листинг 4 :1.ibmpc.c

```
 */ibmpc.c/*
  */Функции нижнего уровня,  обращающиеся к BIOS и аппаратным
средствам PC/*
#pragma  inline #include <dos.h> static union REGS rg;

 */позиция курсора/*
void cursor(int x,int y} (
   rg.x.ax = 0x0200;
   rg.x.bx = 0;
   rg.x.dx = ((y << 8) & 0xff00) + x;
   int86( 16, &rg, &rg;(
{

 */возвратить позицию курсора/*
void curr_cursor( int *x, int *y(
}
   rg.x.ax = 0x0300;
   rg.x.bx = 0;
   int86( 16, &rg, &rg;(
*   x = rg.h.dl;
*   y = rg.h.dh;
{


 */установить тип курсора/*
void set_cursor_type( int t(
}
   rg.x.ax = 0x0100;
   rg.x.bx = 0;
   rg.x.cx = t;
   int86( 16, &rg, &rg;(
{
char attrib = 7;

 */очистить экран/*
void clear_screen()
}
   cursor(0, 0;(
   rg.h.al;' ' =
   rg.h.ah = 9;
   rg.x.bx = attrib;
   rg.x.cx = 2000;
   int86( 16, &rg, &rg;(
{

 */возвратить режим работы видеоконтроллера/*
int vmode()
}
   rg.h.ah = 15;
   int86( 16, &rg, &rg;(
   return rg.h.al;
{
```

```
 */проверить клавишу Scroll Lock/*
int scroll_lock()
}
   rg.x.ax = 0x0200;
   int86( 0x16, &rg, &rg;(
   return rg.h.al & 0x10;
{
void (* helpfunc;()(
int helpkey = 0;
int helping = 0;

 */принять символ от клавиатуры/*
int get_char()
}
   int c;
   while (1}   (
       rg.h.ah = 1;
       int86(0x16, &rg, &rg;(
       if (rg.x.flags & 0x40}  (
   int86(0x28, &rg, &rg;(
   continue;
{
       rg.h.ah = 0;
       int86(0x16, &rg, &rg;(
       if (rg.h.al == 0(
   c = rg.h.ah | 128;
       else
   c = rg.h.al;
       if (c == helpkey && helpfunc}   (
   if (!helping}   (
       helping = 1;
*)       helpfunc;()(
       helping = 0;
       continue;
{
{
       break;
{
    return c;
{

 */занести код символа и его атрибуты в видеопамять/*
void vroke(unsigned vseg, unsigned adr, unsigned chr(
}
    if (vseg == 45056)     /* монохромный режим/*
poke(vseg, adr, chr;(
    else}
_DI = adr; /* смещение до адреса символа в видеопамяти/*
_ES = vseg; /* адрес сегмента видеопамяти/*
asm cld;
_BX = chr; /* атрибуты и код символа/*
_DX = 986; /* состояние видеопорта/*

 */ждать начала обратного хода луча/*
do
    asm in al,dx;
while (_AL & 1;(

 */ждать завершения обратного хода луча/*
do
```

```
    asm in al,dx;
while (!(_AL & 1;((
_AL = _BL;
asm stosb;    /* запомнить символ/*

 */ждать начала обратного хода луча/*
do
    asm in al,dx;
while (_AL & 1;(

 */ждать завершения обратнога хода луча/*
do
    asm in al,dx;
while (!(_AL & 1;((
_AL = _BL;
asm stosb */     ;запомнить атрибуты/*
{
{

 */считать код символа и его атрибуты из видеопамяти/*
int vpeek(unsigned vseg, unsigned adr(
}
    int ch, at;
    if (vseg == 45056)   /* монохромный режим/*
return peek(vseg, adr;(
    asm push ds;
_    DX = 986;    /* состояние видеопорта/*
_    DS = vseg;   /* адрес сегмента видеопамяти/*
_    SI = adr;/* смещение до адреса символа в видеопамяти/*
    asm cld;

 */    ждать начала обратного хода луча/*
    do
asm in al,dx;
    while (_AL;(1 &

 */    ждать завершения обратного хода луча/*
    do
       asm in al,dx;
    while (!(_AL & 1;((
    asm lodsb;     /* считать символ/*
_    BL = _AL;

 */    ждать начала обратного хода луча/*
    do
asm in al,dx;
    while (_AL & 1;(

 */    ждать завершения обратного хода луча/*
    do
asm in al,dx;
    while (!(_AL & 1;((
    asm lodsb;     /* считать атрибут/*
_    BH = _AL;
_    AX = _BX;
    asm pop ds;
    return _AX;
{

 */keys.h/*
#define HT          9
```

```
#define RUBOUT      8
#define BELL        7
#define ESC        27
#define SHIFT_HT  143
#define CTRL_T     20
#define CTRL_B      2
#define CTRL_D      4
#define ALT_D     160

#define F1   187
#define F2   188
#define F3   189
#define F4   190
#define F5   191
#define F6   192
#define F7   193
#define F8   194
#define F9   195
#define F10  196

#define HOME   199
#define UP     200
#define PGUP   201
#define BS     203
#define FWD    205
#define END    207
#define DN     208
#define PGDN   209
#define INS    210
#define DEL    211

#define CTRL_HOME   247
#define CTRL_BS     243
#define CTRL_FWD    244
#define CTRL_END    245
```

Заключение

На основе представленных выше функций нижнего уровня в разделе 5 будет развиваться и объясняться концепция экранных окон, которая составляет следующий, более высокий, уровень в многоуровневом наборе функций, описываемых в данной книге.

ГЛАВА 5

Экранные окна

Этот раздел посвящен вопросу о том, что собой представляют экранные окна и как с ними работать. Раздел 6 разъясняет, как можно использовать окна в ваших программах, создаваемых в среде Турбо Си, а также содержит полную библиотеку функций управления окнами. Следующие разделы содержат расширенную библиотеку функций, поддерживающих работу с окнами для специфических целей, как, например, для контекстно-чувствительного вывода справочной

информации, редактирования текста, ввода данных и создания меню.

После того, как вы прочитали об экранных окнах, способах их создания и применения, постарайтесь вспомнить программные системы, которые используют подобные средства. Подумайте также и о том, какую пользу могли бы принести эти средства для тех программных проектов, в которых вы принимали участие. Затем попытайтесь найти в этих функциях недостатки, устранение которых позволит сделать функции более подходящими для вашей работы. В любой программе вы почти всегда можете отыскать эти недостатки. В данном случае вы имеете большое преимущество: исходные тексты функций предоставлены, вы можете модифицировать их по своему усмотрению.

Экранное окно

Окном называется область экрана дисплея, которая используется для определенных целей. Окно обычно имеет форму прямоугольника или квадрата, а его границей служат символы из набора графических символов. Использование окон становится наиболее популярным способом представления информации, предназначенной для восприятия пользователем ПЭВМ. Этот способ позволяет на ограниченном пространстве экрана отображать информацию, передаваемую множеством задач, выполняющихся асинхронно.

Поскольку отображаемые на экране изображения формируются в видеопамяти с прямым доступом процессора (так называемый способ формирования изображения путем управления содержимым памяти,(то окна на экране создаются мгновенно, возникая как бы из ничего, и так же мгновенно исчезают. Использование окон в программах позволяет очень удобно отображать различного рода информацию. Окна появляются и сменяют друг друга: они появляются на экране, когда это необходимо, и исчезают после того, как информация, которую они содержат, становится вам ненужной. Программа может отображать столько окон, сколько нужно программисту. Окна могут иметь различные размеры, цвета и форматы. Вы можете разместить окно в любом месте экрана, где это необходимо. Если окно становится ненужным, то вы можете удалить его, и окно исчезнет, а на его месте будет то, что было до его появления.

Каждый, кто занимался проблемами автоматизации в последние годы, обязательно сталкивался с окнами. ПЭВМ IBM PC сейчас встречаются везде, в любой сфере общественной и деловой жизни. И почти на каждой из них работает вездесущая программа Sidekick, включающая калькулятор, календарь и средства вызова абонента в

сети. Каждый из этих компонентов отображает свою информацию с помощью окон, которые по мере необходимости появляются и исчезают на экране, не разрушая того изображения, которое было до их появления. Волшебники в области разработки программного обеспечения и ее сбыта из фирмы Borland International (это они дали Вам Турбо Си) внедрили окна во всеобщую практику. Если вы никогда не видели окон, то отложите книгу в сторону, найдите IBM PC и нажмите одновременно клавиши <Alt> и <Ctrl>. Если вы не увидите после этого красно-бело-зелено-голубого окна на экране)или зеленого и черного для монохромного экрана), то спросите у

владельца машины, почему у него не работает программа Sidekick. Если же окно небольшого размера все-таки появится, начните с ним игру. Сделав выбор в меню, содержащемся в первом окне, вы можете создать другое окно. Вы можете перемещать окна, нажимая клавишу со стрелкой. Вы можете удалять окна, нажимая клавишу <Esc>. Вы можете вызвать на экран справочную информацию по работе с программой, нажав клавишу <F1.<

Из этого раздела вы узнаете, как работать с окнами, как накладывать их друг на друга, как перемещать их по экрану и как их использовать в своих программах, разрабатываемых с помощью Турбо Си. Окна используются для отображения меню, информационных и предупреждающих сообщений, текстовых файлов, шаблона для ввода данных и контекстно-чувствительных информационных сообщений. Программные системы могут использовать окна для отображения информации независимо от содержания других окон и размера области пространства экрана, занимаемой другими окнами. Благодаря этому можно обозревать на экране несколько окон, даже если каждое из них занимает больше половины экрана, потому что окна могут как бы "всплывать" друг над другом.

Свойство всплывания окон часто объясняют путем сравнения экрана дисплея с поверхностью стола. Допустим, на вашем столе лежит множество документов, но в каждый момент времени вы можете работать только с одним из этих документов, возможно, наиболее важным для вас. То, что можно делать с бумагами, можно выполнять и на экране дисплея. Если первоначально изображение на экране содержит только одно окно, как это показано на рис. 5.1, и программе необходимо создать второе окно, не уничтожая при этом первого, то в результате получится изображение, как на рис. 5.2. Если затем второе окно становится ненужным, то оно уничтожается, и изображение снова будет соответствовать рис. 5.1.

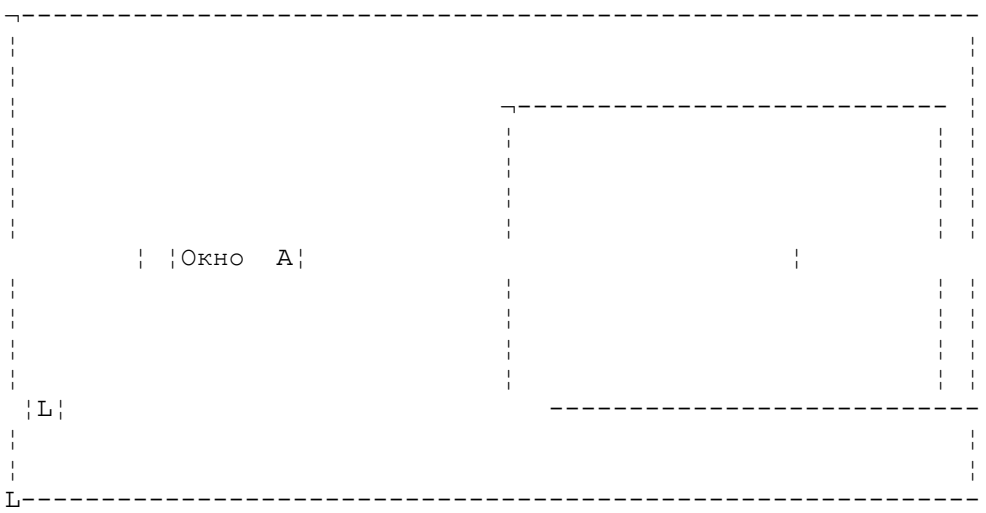
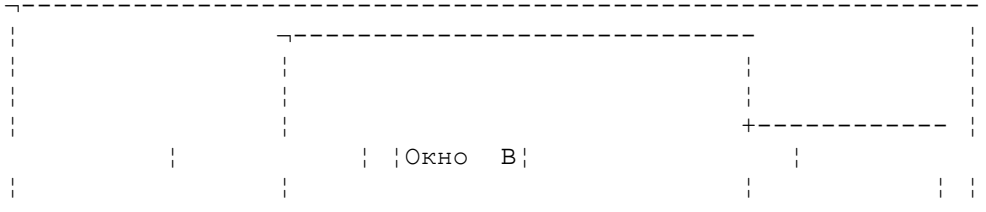


Рис. 5.1 Экранное окно



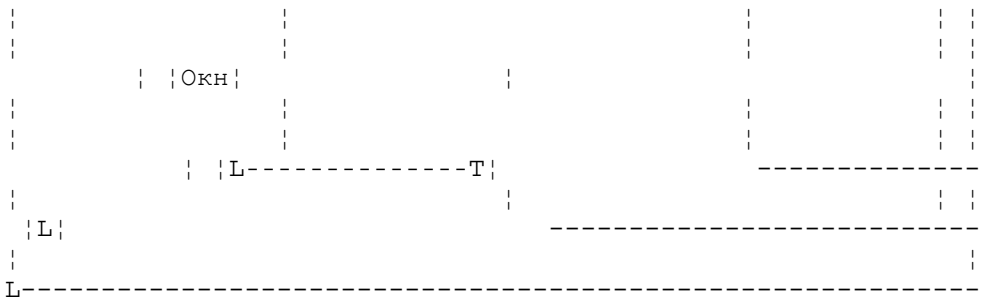


Рис. 5.2 Наложение окон

Для того, чтобы понять, как можно использовать окна, необходимо знать, какие действия с окнами можно выполнять. Помните, что окном является прямоугольная область на экране дисплея. Окна имеют и другие свойства, но главное, что отличает их от других типов экранных изображений, - это их способность как бы всплывать и погружаться относительно остального изображения.

Информация, которая содержалась в соответствующей прямоугольной области экрана до появления нового окна, должна сохраняться. Окно накладывается на предшествующее изображение, как бы всплывает. Когда окно уничтожается (погружается), то информация, которая была до его появления, должна быть восстановлена. Та часть изображения, на которую накладывается новое окно, также может содержать окна. На рис. 5.3 представлено изображение на экране дисплея, которое содержит несколько окон, причем каждое последующее окно накладывается на часть предыдущего.

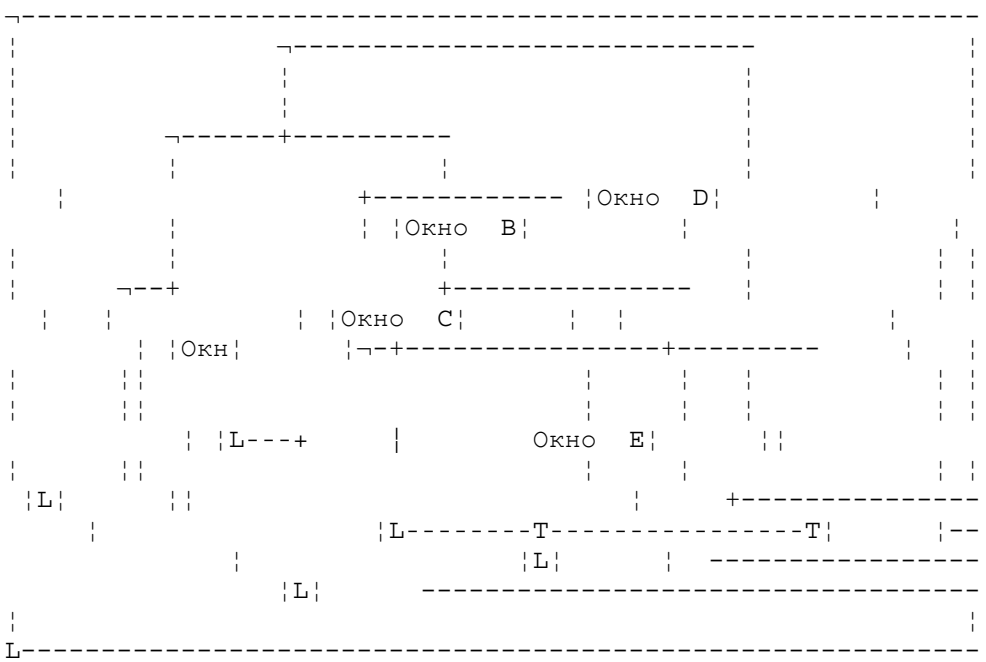


Рис. 5.2 Наложение нескольких окон

Если бы каждая программа, работающая с окнами, должна была управлять размещением окон и восстанавливать содержимое экрана, которое было до их появления, то создание и сопровождение таких программ было бы очень трудным делом. К счастью, в этом нет

необходимости. Поскольку функции и свойства окон являются общими для разных применений, то можно использовать библиотеку функций общего назначения для работы с окнами. Настоящая книга содержит такую библиотеку, ее функции описаны в разделе 6.

Основных операций по работе с окнами, использующихся в большинстве программ, не так уж много, и они не слишком сложны. При работе с окнами вам необходимо установить окно, определив его размеры и местоположение. Вы имеете возможность также устанавливать его цвета, границу и заголовок. Вы можете вписать свой текст внутрь окна, а также при необходимости изменять местоположение окна. Наконец, вы можете удалить окно с экрана, восстановив при этом изображение, которое было на экране до появления окна. Основываясь на этих операциях, вы можете создавать программы ,которые используют чарующие многоцветьем красок окна в качестве пользовательского интерфейса. (Следует помнить, что использование эффективной библиотеки само по себе не гарантирует высокое качество пользовательских характеристик разрабатываемой программы. Программист должен использовать вместе с окнами также звуковые эффекты и другие инструментальные программные средства.(

Чтобы понять, как можно управлять изображением на экране, вы должны разобраться в организации видеопамяти. Приводимые ниже сведения представляют собой введение в архитектуру видеопамяти IBM PC. Для получения исчерпывающей информации по этому вопросу обращайтесь к "Руководству Питера Нортона для программистов по IBM PC" (Питер Нортон, Microsoft Press, 1985.(

Архитектура видеопамяти

Система формирования изображения является неотъемлемой частью ПЭВМ IBM PC. В более ранних моделях персональных ЭВМ видеотерминалы подключались через последовательные порты ввода/вывода, но аппаратная архитектура IBM PC включает в себя и видеосистему.

Предназначенное для вывода на экран изображение создается в видеопамяти. В IBM PC в качестве видеопамяти используется часть оперативной памяти. Видеопамять доступна для чтения и записи процессору и, следовательно, вашим программам. Видеопроцессор, входящий в состав видеоконтроллера, по содержимому видеопамяти постоянно формирует изображение на экране дисплея. Поэтому каждый новый символ, записанный в видеопамять, почти немедленно появляется на экране. Поскольку видеопамять доступна для микропроцессора, то скорость формирования изображения соответствует скорости пересылки содержимого памяти, которая превышает скорость передачи данных при подключении видеотерминала через последовательные порты ввода/вывода.

Адреса видеопамяти и ее характеристики являются стандартными для всех ПЭВМ линии IBM PC, а также совместимых с ними.

ПЭВМ типа IBM PC может иметь одну из трех видеосистем, использующих различные типы видеомониторов и, следовательно, различные типы видеоконтроллеров. Видеоконтроллер первого типа называется Монохромным Адаптером (МА), он обеспечивает работу только монохромного видеомонитора в символьном режиме и не

поддерживает графического режима. Видеоконтроллер второго типа называется Цветным Графическим Адаптером (CGA). С помощью контроллера CGA подключается цветной монитор, который может работать в двух различных режимах. В символьном режиме имеется возможность выбирать цвет фона и цвет символа из восьми возможных цветов, а также один из двух уровней интенсивности цвета для символа. В графическом режиме низкого разрешения (640x200 растровых точек) можно работать только с одним цветом. Третий видеоконтроллер называется Усовершенствованным Графическим Адаптером (EGA), который поддерживает такой же символьный режим, что и CGA, и многоцветный графический режим более высокого разрешения.

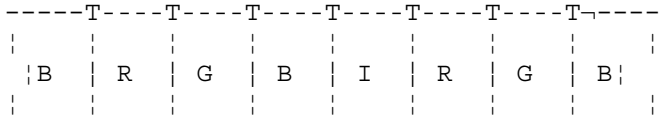
Приведенные в данной книге программы работают с любым из этих контроллеров в символьном режиме. Поскольку в этом режиме контроллеры CGA и EGA функционально эквивалентны, то нижеследующие рассуждения для CGA относятся к обоим этим контроллерам.

Видеопамять организована в виде двумерного массива символов, состоящего из рядов и колонок. Ее можно рассматривать и как набор следующих друг за другом 16-разрядных слов, по одному на каждый символ. Каждый ряд содержит 80 следующих друг за другом символов, все они образуют 25 следующих друг за другом колонок. Слово содержит информацию об одном символе и состоит из двух восьмибитных байт: один - для ASCII-кода символа, второй - для атрибутов символа, определяющих его изображение. Код ASCII записан в младшем (правом) байте слова.

Видеопамять контроллера MA организована в виде одной страницы, а контроллера CGA - в виде четырех страниц. Программы из этой книги используют только первую страницу видеопамяти контроллера CGA.

Видеопамять расположена в верхних областях доступного процессору адресного пространства. Разработчики IBM PC, столкнувшись с ограничением на максимальный объем адресуемого адресного пространства в 1 мегабайт, решили разместить видеопамять и ПЗУ Базовой Системы Ввода-Вывода (ROM BIOS) в верхних областях этого пространства. Для того, чтобы позволить контроллерам MA и CGA работать совместно на одной ПЭВМ, разработчики назначили различные адреса сегментов видеопамяти для разных контроллеров. Память контроллера MA начинается в сегменте 0xB000, а память контроллера CGA - в сегменте 0xB800. Программа может определить, какой из контроллеров используется, путем вызова соответствующей функции ROM BIOS, и настроиться таким образом на соответствующий адрес сегмента видеопамяти. Поскольку архитектура видеопамяти для того и другого случая в основном одинакова, то необходимые действия по настройке программы будут минимальными.

Байт атрибутов символа содержит 2 трехразрядных поля кодирования цвета (одно для цвета фона символа и одно для цвета самого символа), разряд для задания уровня интенсивности цвета символа и разряд для установки режима мерцания символа при его отображении. На рис. 5.4 представлена конфигурация байта атрибутов.



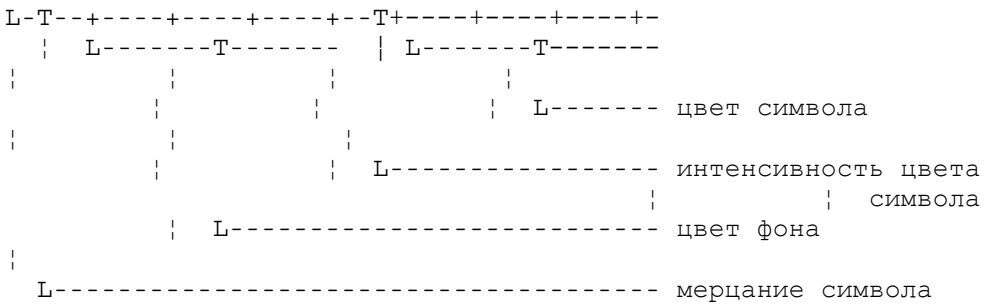


Рис. 5.4 Байт атрибутов символа

Комбинации красной, зеленой и голубой цветовых составляющих позволяют получить набор из восьми отчетливо различаемых цветов: белого, красного, зеленого, голубого, синего, малинового, желтого и черного. Использование разряда интенсивности для цвета символа позволяет получить еще восемь дополнительных оттенков цвета.

В случае использования Монохромного Адаптера значение байта атрибутов немного отличается от описанного выше. Поскольку возможность использования различных цветов не поддерживается контроллером MA, то разрешены только комбинации белого символа на черном фоне или черного символа на белом фоне. Другие коды цвета дают комбинации черного на черном или белого на белом. Хотя следует отметить, что комбинация голубого символа и черного фона формирует изображение подчеркнутого символа, что не поддерживается уже контроллером CGA. Монохромный Адаптер, как и CGA, поддерживает возможности задания интенсивности цвета и режима мерцания символа.

Поскольку вы теперь знаете, где размещена видеопамять и что в нее может быть записано, то можете использовать программу pokes из библиотеки Турбо Си для формирования изображения на экране дисплея. Ниже приводится листинг 5.1 маленькой программы под названием vidpoke.c, которая записывает строку непосредственно в видеопамять. Исходя из предположения, что на вашей машине работает контроллер CGA, программа формирует для каждого символа байт атрибута со значением 7, что соответствут отображению строки в виде белых символов на черном фоне. Если вы используете Монохромный Адаптер, измените в строке # define VSEG значение с 0xB800 на 0xB000.

Листинг 5.1: vidpoke.c

```
 */vidpoke.c/*

#define VSEG 0x6800
char vdata [] =  "Что сделал Кан;"?
main()
}
char *vp;
int v;
for (v=0, vp = vdata* ;vp; v +=2, vp(++
     poke ( VSEG, v, 0x700| *vp;(
{

"        Снег" и обратный ход луча развертки
-----------------------------------------------------------------
```

Если вы запустите программу vidpoke.exe в цикле или значительно увеличите длину выводимой строки, то сможете увидеть на экране так называемый "снег", который появляется при выполнении программы. Вы будете наблюдать "снег" только в случае использования контроллера CGA или его аналога. Контроллеры EGA и MA не дают подобного эффекта. Из нижеследующих объяснений вы узнаете, почему возникает "снег" и как его можно устранить программнми средствами.

" Снег" возникает при использовании контроллера CGA из-за особенностей аппаратной архитектуры видеосистемы. Поскольку и микропроцессор, и видеоконтроллер обращаются к одной видеопамяти, то в случае одновременного обращения возникает необходимость в координировании доступа к памяти. Работа видеоконтроллера синхронизирована по времени с работой схемы развертки. Во время формирования элемента изображения на экране видеоконтроллер должен иметь доступ к ячейке памяти, которая содержит бит, соответствующий этому элементу изображения. Видеопроцессор при этом не может ждать, поскольку он жестко синхронизирован с устройством формирования растра. Следовательно, если видеоконтроллер и микропроцессор пытаются одновременно обратиться к одной и той же ячейке памяти, то видеоконтроллер должен иметь при этом более высокий приоритет. В хорошо разработанной системе микропроцессор будет находиться в состоянии ожидания, при котором запросы микропроцессора к видеопамяти не поступают. В контроллерах MA и EGA использован этот принцип, и вы можете не беспокоиться о конфликтных ситуациях между микропроцессором и видеоконтроллером при доступе к видеопамяти. Но в контроллере CGA в символьном режиме этого не выполняется, поэтому вы должны сами предпринимать некоторые действия.

В случае использования контроллера CGA, если микропроцессору нужно обратиться к видеопамяти для чтения или записи, то это ему разрешается, а видеоконтроллеру доступ на это время запрещается. А поскольку работа видеоконтроллерMZШ□□
+□ □

□□Эit□□□□□-□□$

Глава 6

Библиотека оконных функций

Программы, описанные в этой и нескольких последующих главах, представляют библиотеку оконных функций, которая поддерживает широкий диапазон экранных оконных операций. Функции подразделены на подсистемы, использование которых позволяет организовать меню, контекстно-управляемые подсказки, редактирование текста и форматирование данных в прикладных системах. Эти подсистемы поддерживаются общецелевой оконной библиотекой, которая может использоваться в прикладных программах так же, как и подсистемы. В этой главе описывается общецелевая библиотека оконных функций.

Оконные функции, описанные в данной главе, могут применяться для организации окон в одной из двух конфигураций: стековой и слоеной; одна конфигурация является с точки зрения использующей их программы подмножеством другой. Слоеные окна обладают большими возможностями, чем стековые, однако стековые более эффективны, то

есть выдача на экран и уничтожение их происходит быстрее. Программа может быть связана либо со стековыми ,либо со слоеными оконными функциями, но не с теми и другими одновременно.

При компиляции оконных функций, вы должны принять решение о том, какую именно оконную конфигурацию применить. Для стековых окон определяется переменная времени компиляции FASTWINDOWS, для слоеных окон она удаляется. Прикладная программа может быть связана с любой библиотекой до тех пор, пока она не использует возможности, поддерживаемые только для слоеных окон. Те прикладные программы, которые используют возможности исключительно слоеных окон, должны быть связаны с библиотекой оконных функций, которая была компилирована без определения FASTWINDOWS.

Стековые окна

Конфигурация стековых окон предполагает, что любая выполняемая вами с окном операция (запись в него текста, изменение цвета, уничтожение его и т.д.) производится, когда окно является полностью видимым пользователю. Полная видимость означает, что ни одна часть окна не накрыта другим окном и что окно не скрыто функцией hide_window (о которой будет сказано ниже). Когда установлено стековое окно, оконное программное обеспечение строит буфер для хранения прежнего содержимого видеопамяти, которую будет занимать окно. Видеопамять сохраняется в буфере, а окно записывается в видеопамять. При выполнении любых операций, модифицирующих окно, все изменения выполняются непосредственно в видеопамяти, а программное обеспечение предполагает, что окно является полностью видимым. Когда окно уничтожается, содержимое хранящего его буфера записывается обратно в видеопамять, восстанавливая таким образом видеообраз памяти к состоянию до образования окна.

Вы будете обычно обращаться только к стековому окну, образованному последним. Если вы сначала создали окно A, а затем окно B, которое закрывает часть окна A, то при записи текста в окно A может случиться, что часть текста попадет в часть окна B, закрывающую окно A. Далее, если вы уничтожите окно A до уничтожения окна B, то часть буфера сохранения окна A запишется в начало части окна B.

Большинство коммерческих оконных пакетов поддерживает только стековые окна, поскольку большинство приложений, использующих окна, могут успешно функционировать в среде стековых окон. Обычной практикой в приложениях является использование окна, открытого последним, и уничтожение окон в порядке, обратном их созданию. Такие приложения должны использовать стековые окна из-за преимуществ их функционирования.

Оконные операции, описанные в данной книге, позволяют вам создавать одно или более окон и затем относить различные операции к одному из созданных окон. Вы можете обращаться к конкретному окну или использовать пустую спецификацию для сообщения вызванной функции о своем намерении выполнить операцию в окне, созданном последним .Это соглашение используется как для стековых, так и для слоеных окон, однако пользователи стековых оконных функций должны быть уверены, что любое окно, к которому они обращаются, либо создано последним, либо полностью видимо на экране.

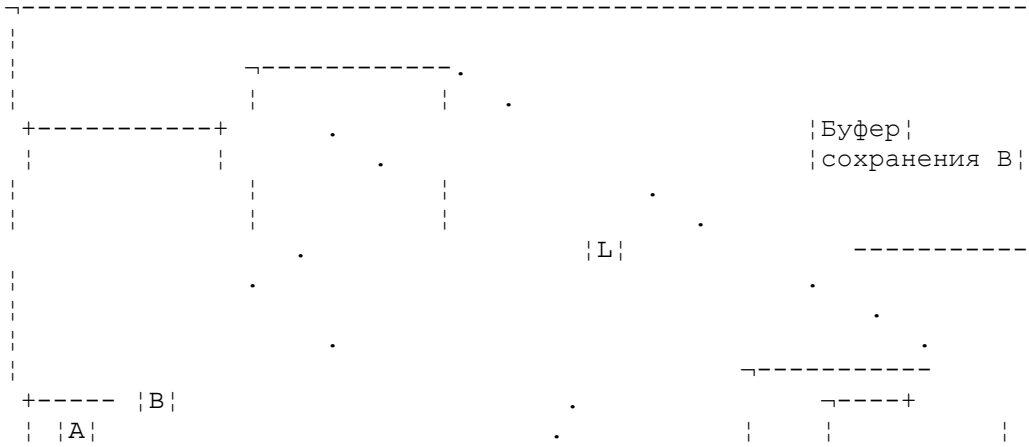
Слоеные окна

Слоеные окна обладают гораздо большей гибкостью, чем стековые, к тому же они предоставляют пользователю гораздо больше возможностей по созданию различных оконных интерфейсов. Когда слоеное окно создано, любая оконная операция может быть адресована ему, независимо от его видимости или близости к другим окнам. В дополнение к обычному набору оконных операций слоеные окна могут перемещаться в двумерной плоскости экрана и могут выдвигаться на передний или убираться на задний планы в слоях созданных окон.

Когда создано слоеное окно, распределяется буфер сохранения видеосодержимого, но окно не отображается. Буфер сохранения инициализируется видеозначениями, которые окно могло бы содержать, если бы оно было видимым. Любые последующие операции, производимые в этом окне, пока оно невидимо, производятся в буфере сохранения.

Когда слоеное окно отображается, видеопамять и буфер сохранения обмениваются содержимым. Затем, до тех пор пока окно является полностью видимым (не закрытым полностью или частично другим окном), любые операции производятся с видеопамятью, а не с буфером сохранения. Когда одно и более других окон покрывают адресуемое окно полностью или частично, определение области изменения является более сложным. Для тех частей окна, которые являются видимыми, изменение производится в видеопамяти. Однако для областей, закрытых другими окнами, изменение производится в буфере сохранения покрывающего окна. Поскольку окно может иметь различные части, закрытые несколькими другими окнами, то алгоритм определения места, где должно быть сделано изменение, обязан сначала просмотреть все окна, созданные позже данного, с целью установления факта выполнения изменения в области, покрытой следующим окном последовательности. Если такое окно найдено, то изменение записывается в его буфер сохранения. Если ни одно окно не закрывает модифицируемый участок, то изменение производится в видеопамяти.

Рассмотрим рисунок 6.1. Три окна расположены так, что часть окна A видима, часть закрыта окном B и часть - окном C. В буферах сохранения каждого окна вы можете видеть границы частей других окон, которые закрыты.



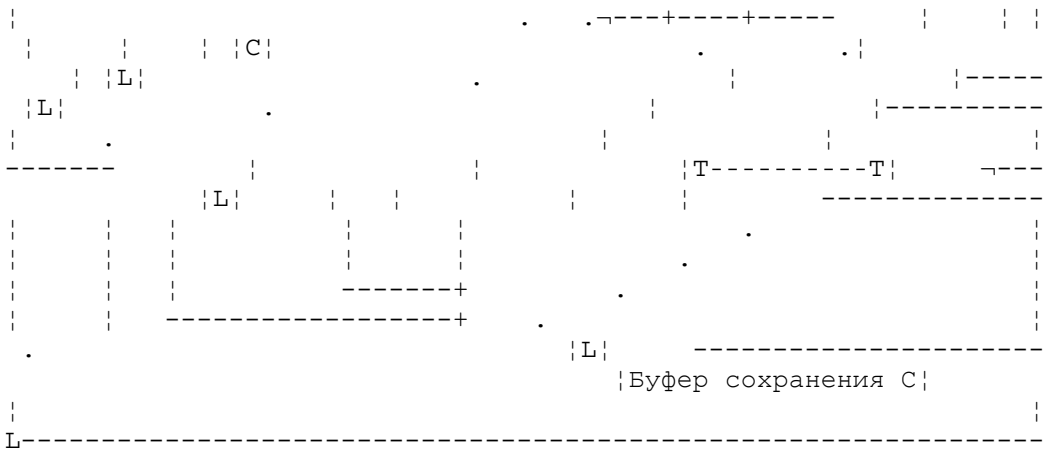


Рисунок 6.1. Три слоеных перекрывающихся окна.

Если вы запишите текстовую строку "now is the time" в окно A, текст будет направлен в три различных места. Результат показан на рисунке 6.2. Так как часть окна A, где записано "now," является видимой, то слово записывается непосредственно в видеопамять и может быть прочитано пользователем. Слова "is the" являются частью окна A, которая закрыта окном B, поэтому эти слова записываются в буфер сохранения окна B. Слово "time" оказывается в той части окна A, которая покрыта окном C, следовательно "time" записывается в буфер сохранения окна C.

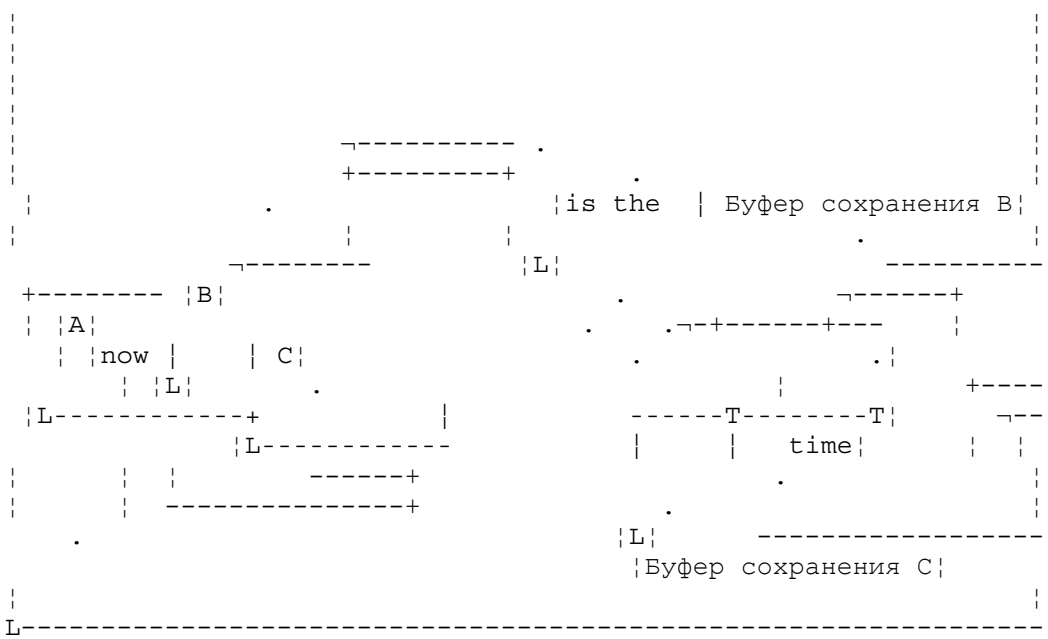


Рисунок 6.2. Слоеные окна с текстом.

Рисунок 6.3. показывает, что происходит при уничтожении окна B. Часть окна A из его буфера сохранения поступает в видеопамять, и слово "is" может теперь быть прочитано пользователем. Однако, поскольку часть окна B была покрыта окном C, то часть буфера сохранения окна B копируется в буфер сохранения окна C, следовательно, буфер сохранения окна C теперь

содержит слова "the time."

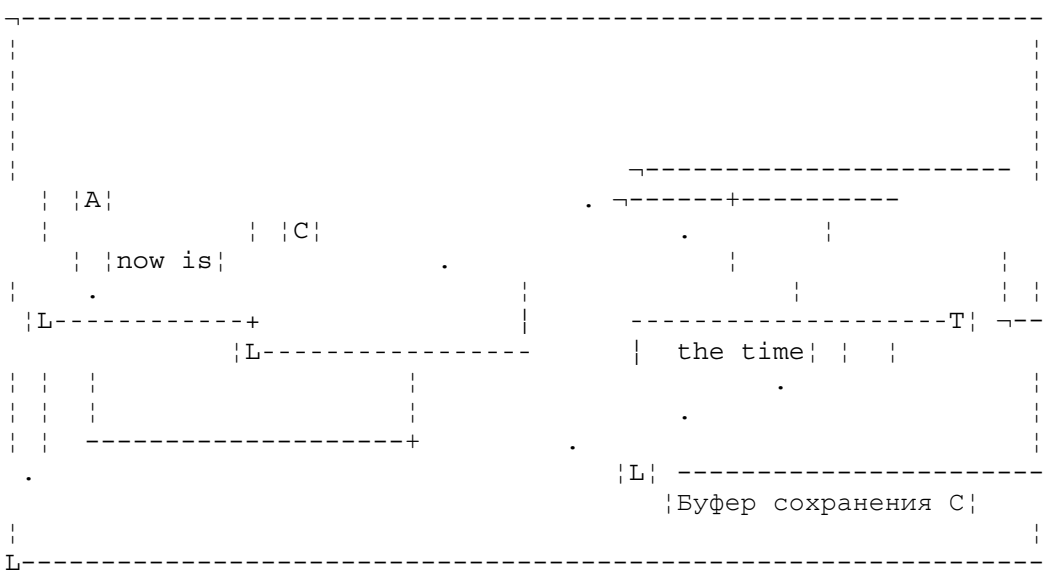


Рисунок 6.3. Уничтожение слоеного окна.

Независимо от использования стековых или слоеных окон, работа их будет зависеть от требований вашей системы. Каждый подход имеет свои преимущества и недостатки. Если вы начали со стековых окон и обнаружили позже, что вам нужны дополнительные свойства слоеной оконной архитектуры, вы можете выполнить изменение путем перекомпиляции оконных функций без определения FASTWINDOWS и перередактировать связи ваших программ.

Оконные функции

Эти функции включены в оконную библиотеку. Для каждой функции описаны ее назначение и способ применения. Далее приводятся примеры использования этих функций.

WINDOW *establish_window(x,y,h,w(

Эта функция создает окно, но не отображает его. (Для показа окна воспользуйтесь функцией display_window). Параметры x и y являются координатами верхнего левого угла окна. Эти параметры выражаются в символьных позициях экрана, где координаты левого верхнего угла самого экрана равны (0,0). Параметры h и w являются высотой и шириной окна в символьных позициях. Эта функция не вызовет никакого изменения экрана. Если вы создадите окно, позиция и размеры которого не позволяют ему быть полностью видимым, программное обеспечение установит позицию, при которой оно полностью поместится на экране. Если ширина больше 80 или высота больше 25, функция преобразует соответствующий размер до максимально допустимого значения.

Окно создается с умалчиваемыми атрибутами. Его рамка образуется одинарными линиями, цвет содержимого - ярко-белый на черном фоне, а заголовок отсутствует. Эти атрибуты могут быть изменены соответствующими вызовами других функций, описанных

ниже.

Окна располагаются в обратном порядке по отношению к их созданию. Окно, созданное самым последним, является верхним окном экрана и будет (при выдаче) закрывать окна, созданные ранее. Эта иерархия зависит от порядка, в котором окна образуются, а не от порядка их показа.

Функция establish_window возвращает указатель на структуру WINDOW, которая определена во включаемом файле twindow.h. Этот указатель используется при последующих вызовах оконных функций для идентификации окна, к которому относится вызов. Передавайте указатель NULL другим оконным функциям, если хотите работать с окном, образованным последним.

void set_border(WINDOW *wnd, int btype(

Эта функция устанавливает тип оконной рамки. Целочисленный параметр btype должен принимать одно из следующих значений:

- 0 .одинарные линии (по умолчанию;(
- 1 .двойные линии;
- 2 .одинарные верх и низ, двойные боковые;
- 3 .двойные верх и низ, одинарные боковые;
- 4 .специальное проталкиваемое вниз окно меню с одинарными линиями и t-блоком от верхнего левого до верхнего правого угла.

void set_colors(WINDOW *wnd, int area, int bg, int fg, int inten(

Эта функция устанавливает цвета для окна. Параметр area может принимать следующие значения:

.ALL
.BORDER
.TITLE
.ACCENT
.NORMAL

Этот параметр определяет части окна, на которые распространяется действие. ALL соответствует всем частям. BORDER устанавливает цвета рамки окна, которая занимает односимвольный ряд позиций вокруг него. TITLE устанавливает цвета заголовка, размещаемого на верхней границе окна. ACCENT - это область, используемая для блоков меню и другого выделенного текста, который появляется в качестве временно подсвечиваемых частей области NORMAL, где отображается весь текст. Целые числа bg и fg задают цвета для фона и переднего плана различных частей окна. Допускаются следующие цвета:

.RED - алый;
.GREEN - зеленый;
.BLUE - голубой;
.WHITE - белый;
.YELLOW - желтый;
.AQUA - аквамариновый;
.MAGENTA - красный;

.BLACK - черный.

Целое число inten определяет интенсивность символов переднего плана и может принимать два значения:

.BRIGHT - яркая
.DIM - обычная

Все значения определены в исходном файле twindow.h, который рассматривается позже в этой главе.

void set_title(WINDOW *wnd, char *title(

Эта функция устанавливает значение заголовка окна. Передаваемая вами строка должна сохраняться в течение всего существования окна, поэтому используйте литеральную константу или внешний массив символов.

void set_intensity(WINDOW *wnd, int inten(

Эта функция устанавливает интенсивность фона для всех частей окна. Значениями inten могут быть BRIGHT или DIM.

void display_window(WINDOW *wnd(

Эта функция отображает окно, которое было ранее создано. Чтобы избежать неудачного отображения, вызывайте эту функцию после установки всех атрибутов и, если возможно, после того, как окно заполнено текстом, который оно должно выдавать.

void delete_window(WINDOW *wnd(

Эта функция уничтожает созданное ранее окно. Если окно является видимым (выдано с помощью display_window), то оно очищается, и экран восстанавливается к его предыдущему образу.

void clear_window(WINDOW *wnd(

Эта функция заполняет текстовую область окна пробелами.

void hide_window(WINDOW *wnd(

Эта функция очищает выданное окно, восстанавливая на экране его предыдущее содержимое. Окно остается существующим и может модифицироваться любым способом. Последующий вызов display_window

восстановит его на экране в соответствующем расположении относительно других окон.

void wcursor(WINDOW *wnd, int x, int y(

Каждое окно обладает логической позицией курсора, которая изменяется от 0,0 (верхний левый угол текстовой области окна) до граничных размеров окна. Эта функция переставляет курсор в окне.

Функции wputehar и wprintf выдают текст относительно данной позиции курсора. Они изменяют позицию курсора точно также, как это происходило бы, если бы управление курсором осуществлялось с помощью клавиатуры дисплея. Символ новой строки (\n(устанавливает курсор в нулевой столбец следующей строки, перемещая текст в окне вверх на одну строку, если курсор уже находится в нижней строке окна. Символ табуляции (\t) перемещает курсор к следующей позиции табуляции в окне. Табуляции располагаются с интервалом в четыре символа.

void wprintf(WINDOW *wnd, char *fmt(... ,

Эта функция является оконной версией стандартной функции printf языка Си. Она использует стандартную функцию sprintf для построения строки, выдаваемой в окно. Убедитесь, что результирующая строка не длиннее 100 символов, или измените длину в массиве dlin функции wprintf в исходном файле twindow.c.

void wputchar(WINDOW *wnd, int ch(

Эта функция является оконной версией putchar. Она записывает символ из ch в окно в текущую позицию курсора. Курсор при этом продвигается дальше. Если символ является символом новой строки \)n), курсор переставляется в нулевую позицию следующей строки. Если символ является табуляцией (\t), курсор продвигается к следующей позиции табуляции окна. Оконные позиции табуляции располагаются в каждой четвертой позиции окна.

void reverse_video(WINDOW *wnd(

После обращения к этой функции все вызовы wprintf и wputchar будут при отображении использовать цвета ACCENT вместо NORMAL.

void normal_video(WINDOW *wnd(

После обращения к этой функции все вызовы wprintf и wputchar будут при отображении использовать цвета NORMAL. Эта функция используется для возврата к нормальному отображению после вызова reverse_video.

void close_all()

Эта функция уничтожает все образованные окна.

void move_windom(WINDOW *wnd, int x, int y(

Эта функция перемещает окно таким образом, что его верхний левый угол устанавливается в символьных координатах, заданных x и y. Эта функция может быть использована только для слоеных окон.

void rmove_window(WINDOW *wnd, int x, int y(

Эта функция перемещает окно путем добавления к значениям x и y текущих координат левого верхнего угла окна. Используйте эту функцию только для слоеных окон.

void forefront(WINDOW *wnd(

Эта функция перемещает окно в самое переднее положение относительно других окон. Окно, если оно видимо, отображается поверх остальных. Используйте эту функцию только для слоеных окон.

void rear_window(WINDOW *wnd(

Эта функция перемещает окно в самое заднее положение относительно других окон. Окно, если оно видимо, отображается под всеми остальными. Используйте эту функцию только для слоеных окон.

int get_selection(WINDOW *wnd, int sel, char *keys(

Эта функция позволяет использовать окно в качестве меню. Вы должны создать окно и записать в него несколько строк текста, скажем, функцией wprintf. Вы можете использовать set_colors для установки значений цветов ACCENT в окне (по умолчанию - черные буквы на белом фоне). Затем вы вызываете get_selection. Функция использует окно в качестве меню, каждая строка которого представляет альтернативу выбора. Альтернативы подсвечиваются цветом ACCENT. Целочисленное значение sel используется для первоначального размещения яркого блока курсора меню на одной из альтернатив. Значение 1 соответствует первой альтернативе, 2- второй и т.д.

Пользователь может перемещать блок меню вверх и вниз клавишами управления курсором и производить выбор в меню клавишей >Ввод>. Клавиша <Ключ> применяется для выхода из этого процесса.

Ссылка keys указывает на строку значений клавиш, которые могут использоваться для выбора из меню. Некоторые системы меню разрешают пользователю применять нажатия клавиш наряду с перемещением блока курсора. Для выключения этой возможности передайте параметру keys ссылку NULL.

Эта функция возвращает целочисленное значение, равное номеру выбранной в меню альтернативы. Значение обычно равняется единице и более, однако, если пользователь нажимает клавишу <Ключ,< функция возвращает нуль. Функция будет также возвращать значения FWD или BS, если пользователь нажимает клавиши управления курсором вправо или влево. Эти значения определены в keys.h.

void error_message(char *s(

Эта функция выдает сообщение об ошибке, указываемое ссылкой s, и включает звуковой сигнал. Сообщение отображается в окне в нижнем правом квадранте экрана. Сообщение остается на экране после завершения функции.

void clear_message()

Эта функция удаляет сообщение об ошибке, если оно было выдано функцией error_message.

Листинги оконных функций

Данные листинги содержат исходные файлы функций поддержки окна. Приводятся два листинга: twindow.h и twindow.c. Каждый листинг сопровождается описанием его содержания.

Исходный листинг: twindow.h

Листинг 6.1, twindow.h, определяет оконные структуры и содержит прототипы для функций. Вы должны включать их в любую исходную программу, которая использует оконные функции.

Листинг 6.1: twindow.h

```
 */twindow.h/*
  */Выделите это определение из комментария для стековых окон,
   *но не для слоеных окон.
*
# *define FASTWINDOWS
*
/*
 */window colors/*
```

```
#define RED   4
#define GREEN 2
#define BLUE  1
#define WHITE (RED+GREEN+BLUE(
#define YELLOW (RED+GREEN(
#define AQUA (GREEN+BLUE(
#define MAGENTA (RED+BLUE(
#define BLACK 0
#define BRIGHT 8
#define DIM 0

#define BORDER 0
#define TITLE 1
#define ACCENT 2
#define NORMAL 3
#define ALL 4

#define TRUE 1
#define FALSE 0
#define ERROR -1
#define OK 0

 */оконные управляющие структуры/*
typedef struct field {     /* описание поля ввода данных/*
    char *fmask;           /* маска поля ввода данных/*
    int fprot;             /* защита поля/*
    char *fbuff;           /* буфер поля/*
    int ftype;             /* тип поля/*
    int from;              /* строка поля/*
    int fcol;              /* столбец поля/*
    void (*fhelp)();       /* функция подсказки поля/*
    char *fhwin;           /* функция подсказки окна/*
    int flx, fly;          /* расположение подсказки окна/*
    int (*fvalid)();       /* функция заполнения поля/*
    struct field *fnxt;    /* следующее поле выдачи/*
    struct field *fprv;    /* предыдущее поле выдачи/*
 {FIELD;
typedef struct _wnd}
    int _wv;             /* истина, если окно видимо/*
    int _hd;             /* истина, если окно скрыто/*
    char *_ws;          /* указывает на блок сохранения окна/*
    char *_tl */          ;указывает на заголовок окна/*
    int _wx;            /* nv x координата/*
    int _wy;            /* nv y координата/*
    int _ww;            /* ширина окна/*
    int _wh;            /* высота окна/*
    int _wsp;           /* указатель прокрутки/*
    int _sp;            /* указатель выбора/*
    int _cr;            /* позиция  x  курсора/*
    int btype;          /* тип рамки/*
    int wcolor[4];      /* цвета окна/*
    int _pn;            /* предыдущий нормальный цвет/*
    struct _wnd *_nx;   /* указывает на следующее окно/*
    struct _wnd *_pv;   /* указывает на предыдущее окно/*
    FIELD *_fh;         /* указывает на 1-е поле ввода данных/*
    FIELD *_ft;    /* указывает на последнее поле ввода данных/*
 {WINDOW;
typedef struct w_menu}
    char *mname;
    char **mselcs;
    void (**func;()(
```

```
 {MENU;
#define SAV      (wnd->_ws(
#define WTITLE   (wnd->_tl(
#define COL      (wnd->_wx(
#define ROW      (wnd->_wy(
#define WIDTH    (wnd->_ww(
#define HEIGHT   (wnd_<-wh(
#define SCROLL   (wnd->_wsp(
#define SELECT   (wnd->_sp(
#define WCURS    (wnd->_cr(
#define WBORDER  (wnd->wcolor[BORDER([
#define WTITLEC  (wnd->wcolor[TITLE([
#define WACCENT  (wnd->wcolor[ACCENT([
#define WNORMAL  (wnd->wcolor[NORMAL([
#define PNORMAL  (wnd->_pn(
#define BTYPE    (wnd->btype(
#define NEXT     (wnd->_nx(
#define PREV     (wnd->_pv(
#define WCOLOR   (wnd->wcolor(
#define VISIBLE  (wnd->_wv(
#define HIDDEN   (wnd->_hd(
#define FHEAD    (wnd->_fh(
#define FTAIL    (wnd->_ft(
#define NW       (wcs[wnd->btype].nw(
#define NE       (wcs[wnd->btype].ne(
#define SE       (wcs[wnd->btype].se(
#define SW       (wcs[wnd->btype].sw(
#define SIDE     (wcs[wnd->btype].side(
#define LINE     (wcs[wnd->btype].line(

 */ПРОТОТИПЫ ФУНКЦИЙ И МАКРОСЫ/*

 */общецелевые функции и макросы/*

void clear_screen(void;(
int vmode(void;(
void cursor(int, int;(
void curr_cursor(int *, int;(*
int cursor_type(void;(
void set_cursor_type(int;(
int get_char(void;(
int scroll_lock(void;(
void vpoke(unsigned, unsigned, unsigned;(
int vpeek(unsigned, unsigned;(

 */оконные функции и макросы/*
WINDOW *establish_window(int, int, int, int;(
void set_border(WINDOW *, int;(
void set_colors(WINDOW *, int, int, int, int;(
void set_intensity(WINDOW *, int;(
void set_title(WINDOW *, char;(*
void display_window(WINDOW;(*
void delete_window(WINDOW;(*
void clear_window(WINDOW;(*
void hide_window(WINDOW;(*
void wprintf(WINDOW *, char;(... ,*
void wputchar(WINDOW *, int;(
void close_all(void;(
void wcursor(WINDOW *, int x, int y;(
void error_message(char;(*
void clear_message(void;(
```

```
int get_selection(WINDOW *, int, char;(*

#define reverse_video(wnd) wnd->wcolor[3]=wnd->wcolor[2[
#define normal_video(wnd) wnd->wcolor[3]=wnd->_pn
#define rmove_window(wnd,x,y) repos_wnd(wnd, x, y, 0(
#define move_window(wnd,x,y) repos_wnd(wnd, COL-x, ROW-y, 0(
#define forefront(wnd) repos_wnd(wnd, 0, 0, 1(
#define rear_window(wnd) repos_wnd(wnd, 0, 0, -1(

 */внутренние для оконных процессов/*
void accent(WINDOW;(*
void deaccent(WINDOW;(*
void scroll(WINDOW *, int;(
void repos_wnd(WINDOW *, int, int, int;(
void acline(WINDOW *, int;(
#define accent(wnd) acline(wnd, WACCENT(
#define deaccent(wnd) acline(wnd, WNORMAL(
#define clr(bg,fg,in) ((fg)|(bg<<4)|(in((
#define vad(x,y) ((y)*160+(x)*2(
#ifdef FASTWINDOWS
#define cht(ch,at) (((ch)&255)|((at)<<8((
#define displ(w,x,y,c,a) vpoke(VSG,vad(x+COL,y+ROW),cht(c,a((
#define dget(w,x,y) vpeek(VSG,vad(x+COL,y+ROW((
#define verify_wnd(w) (*(w)=listtail)!=0
#else
void displ(WINDOW *wnd, int x, int y, int ch, int at;(
#endif
 */функция редактора/*
void text_editor(WINDOW *, char *, unsigned;(

 */функция меню/*
void menu_select(char *name, MENU *mn;(

 */функция подсказки/*
void load_help(char;(*
void set_help(char *, int, int;(

 */функция ввода данных/*
void init_template(WINDOW;(*
FIELD *establish_field(WINDOW *, int, int, char *, char *, int;(
void clear_template(WINDOW;(*
void field_tally(WINDOW;(*
int data_entry(WINDOW;(*
void wprompt(WINDOW *, int, int, char;(*
void error_message(char;(*
void clear_notice(void;(
void field_window(FIELD *, char *, int, int;(
#define field_protect(f,s)   f->fprot=s
#define field_help(f,h)      f->fhelp=h
#define field_validate(f,v)  f->fvalid=v
```

Описание программы: twindow.h

Глобальная переменная FASTWINDOWS определена внутри комментария в представленной программе. Включение переменной рассчитано на применение стековой оконной конфигурации. Без изменений при компиляции будет принята слоеная оконная конфигурация. Для компиляции стековой оконной системы необходимо выделить оператор #define из комментария.

Структура FIELD используется для определения полей ввода данных внутри области данных в окнах. Этот процесс описан в Главе 8.

Структура WINDOW описывает окно для системы . Каждому окну назначается одна структура этого типа.

Структура MENU используется программным обеспечением оконных меню в Главе 10. Должен быть массив структур MENU с одним элементом для каждого проталкиваемого вниз меню.

Список операторов #define используется для придания операторам в twindow.c лучшей читаемости. Мнемонические имена соответствуют элементам структуры WINDOW, указанной ссылкой wnd. Все функции в twindow.c используют имя этой ссылки по соглашению.

twindow.h содержит прототипы для всех оконных функций, которые будут вызываться прикладными программами.

Исходный листинг: twindow.c.

Листинг 6.2 - это twindow.c. Он содержит все описанные ранее в этой главе функции. Вы должны откомпилировать его и связывать его объектный модуль с любой программой, которая использует окна. Поскольку он вызывает функции из ibmpc.c, его объектный модуль должен быть также включен в редактирование связей.

Листинг 6-2: twindow.c

```
 */twindow.c/*
#include <stdio.h<
#include <ctype.h<
#include <stdarg.h<
#include <dos.h<
#include <alloc.h<
#include <stdlib.h<
#include <string.h<
#include "twindow.h"
#include "keys.h"

#define TABS 4
#define SCRNHT 25
#define SCRNWIDTH 80
#define ON  1
#define OFF 0
#define ERROR -1

 */локальные прототипы/*
redraw(WINDOW *wnd;(
wframe(WINDOW *wnd;(
dtitle(WINDOW *wnd;(
int *waddr(WINDOW *wnd, int x, int y;(
vswap(WINDOW* wnd;(
vsave(WINDOW *wnd;(
vrstr(WINDOW *wnd;(
add_list(WINDOW *wnd;(
beg_list(WINDOW *wnd;(
```

```
remove_list(WINDOW *wnd;(
insert_list(WINDOW *wl, WINDOW *w2;(
#ifndef FASTWINDOWS
int dget(WINDOW *wnd, int x, int y;(
verify_wnd(WINDOW **wl;(
#endif

 */массив наборов символов рамки/*
struct}
    int nw, ne, se, sw, side, line;
 {wcs} = []
 */  ,{218,191,217,192,179,196}    одинарная линия/*
 */  ,{201,187,188,200,186,205}    двойная линия/*
 */ ,{214,183,189,211,186,196}    одинарный верх ,двойные бока/*
 */ ,{213,184,190,212,179,205}    двойной верх, одинарные бока/*
 */  ,{194,194,217,192,179,196}    выталкиваемое вниз меню/*
;{

 */голова и хвост связанного списка оконных структур/*
WINDOW *listhead = NULL;
WINDOW *listtail = NULL;
int VSG;    /*  адрес видеосегмента/*

 */создание нового окна/*
WINDOW *establish_window(x, y, h, w(
}
    WINDOW *wnd;
    VSG = (vmode() == 7 ? 0xb000 : 0xb800;(
    if ((wnd = (WINDOW *) malloc(sizeof (WINDOW))) == NULL(
        return NULL;
 */параметры ограничений/*
WTITLE;"" =
HEIGHT = min(h, SCRNHT;(
WIDTH = min(w, SCRNWIDTH;(
COL = max(0, min(x, SCRNWIDTH-WIDTH;((
ROW = max(0, min(y, SCRNHT-HEIGHT;((
WCURS = 0;
SCROLL = 0;
SELECT = 1;
BTYPE = 0;
VISIBLE = HIDDEN = 0;
PREV = NEXT = NULL;
FHEAD = FTAIL = NULL;
WBORDER=WNORMAL=PNORMAL=WTITLEC=
            clr(BLACK, WHITE, BRIGHT;(
WACCENT = clr(WHITE, BLACK, DIM;(
    if ((SAV = malloc(WIDTH * HEIGHT * 2)) == (char *) 0(
        return NULL;
    add_list(wnd;(
#ifndef FASTWINDOWS
    clear_window(wnd;(
    wframe(wnd;(
#endif
    return wnd;
{

 */установить рамку окна/*
void set_border(WINDOW *wnd, int btype(
}
    if (verify_wnd(&wnd}   ((
        BTYPE = btype;
```

```
        redraw(wnd;(
{
{

 */установить цвета/*
void set_colors(WINDOW *wnd,int area, int bg, int fg, int inten(
}
    if (vmode() == 7}   (
        if (bg != WHITE && bg != BLACK(
            return;
        if (fg != WHITE && fg != BLACK(
            return;
{
    if) verify_wnd(&wnd}   ((
        if (area == ALL(
            while (area(
                WCOLOR [--area] = clr(bg, fg, inten;(
        else
            WCOLOR [area] = clr(bg, fg, inten;(
        redraw(wnd;(

{
{

 */установить яркость окна/*
void set_intensity(WINDOW *wnd, int inten(
}
    int area = ALL;

    if (verify_wnd(&wnd}    ((
        while (area}    (
            WCOLOR [--area] &= ~BRIGHT;
            WCOLOR [area] |= inten;
{
       redraw(wnd;(
{
{

 */установить заголовок/*
void set_title(WINDOW *wnd, char *title(
}
    if (verify_wnd(&wnd}   ((
        WTITLE = title;
        redraw(wnd;(
{
{

 */перевыдать окно при изменении атрибута/*
static redraw(WINDOW *wnd(
}
#ifndef FASTWINDOWS
    int x, y, chat, atr;

     for (y = 1; y < HEIGHT-1; y(++
        for (x = 1; x < WIDTH-1; x}       (++
            chat = dget(wnd, x, y;(
            atr = (((chat>>8)&255== (
                PNORMAL ? WNORMAL : WACCENT;(
            displ)wnd, x, y, chat&255, atr;(
{
    wframe(wnd;(
```

```
#endif
    PNORMAL = WNORMAL;
{

 */выдать созданное окно/*
void display_window(WINDOW *wnd(
}
    if (verify_wnd(&wnd) && !VISIBLE}    (
        VISIBLE = 1;
#ifdef FASTWINDOWS
        if (HIDDEN}  (
            HIDDEN = 0;
            vrstr(wnd;(
{
       else}
            vsave(wnd;(
            clear_window(wnd;(
            wframe(wnd;(
{
#else
       vswap(wnd;(
#endif
{
{

 */закрыть все окна/*
void close_all()
}
    WINDOW *sav, *wnd = listtail;

    while (wnd} (
        sav = PREV;
        delete_window(wnd;(
        wnd = sav;
{
{

 */удалить окно/*
void delete_window(WINDOW *wnd(
}
    if (verify_wnd(&wnd}       ((
        hide_window(wnd;(
        free(SAV;(
        remove_list(wnd);   /* удалить окно из списка/*
        free(wnd;(
{
{

 */скрыть окно/*
void hide_window(WINDOW *wnd(
}
    if (verify_wnd(&wnd) && VISIBLE}      (
#ifndef FASTWINDOWS
        vswap(wnd;(
#else
        vrstr(wnd;(
#endif
        HIDDEN = 1;
        VISIBLE = 0;
{
{
```

```
#ifndef FASTWINDOWS
 */перемещение окна в его  3-х мерном плане/*
void repos_wnd(WINDOW *wnd, int x, int y, int z(
}
    WINDOW *twnd;
    int x1, y1, chat;
    if (!verify_wnd(&wnd((
        return;
    twnd = establish_window(x+COL, y+ROW, HEIGHT, WIDTH;(
    twnd -> _tl = WTITLE;
    twnd -> btype = BTYPE;
    twnd -> wcolor[BORDER] = WBORDER;
    twnd -> wcolor[TITLE] = WTITLEC;
    twnd -> wcolor[ACCENT] = WACCENT;
    twnd -> wcolor[NORMAL] = WNORMAL;
    twnd -> _wsp = SCROLL;
    twnd -> _cr = WCURS;

    if (z != 1} (
        remove_list(twnd;(
        if (z == 0(
            insert_list(twnd, wnd;(
        else
            beg_list(twnd;(
{
     for (y1 = 0; y1 < twnd->_wh; y1(++
         for (x1 = 0; x1 < twnd->_ww; x1}   (++
             chat = dget(wnd, x1, y1;(
             displ(twnd, x1, y1, chat&255, (chat>>8)&255;(
{
     twnd->_wv = 1;
     vswap(twnd;(
     hide_window(wnd;(
     free(SAV;(
     remove_list(wnd;(
*     wnd = *twnd;
     insert_list(wnd, twnd;(
     remove_list(twnd;(
     free(twnd;(
{
#endif

 */очистить область окна/*
void clear_window(WINDOW *wnd(
}
   register int x1, y1;

   if (verify_wnd(&wnd((
       for (y1 = 1; y1 < HEIGHT-1; y1(++
           for (x1 = 1; x1 < WIDTH-1; x1(++
               displ(wnd,x1, y1, ' ', WNORMAL;(
{

 */изобразить окно/*
static wframe(WINDOW *wnd(
}
    register int x1, y1;

    if (!verify_wnd(&wnd((
        return;
```

```
 */заголовок окна/*
displ(wnd,0, 0, NW, WBORDER;(
dtitle(wnd;(
displ(wnd,WIDTH-1, 0, NE, WBORDER;(
 */боковые стороны окна/*
for (y1 = 1; y1 < HEIGHT-1; y1}        (++
    displ(wnd,0, y1, SIDE, WBORDER;(
    displ(wnd,WIDTH-1, y1, SIDE, WBORDER;(
{
 */низ окна/*
displ(wnd,0, y1, SW, WBORDER;(
for (x1 = 1; x1 < WIDTH-1; x1(++
    displ(wnd,x1, y1, LINE, WBORDER;(
displ(wnd,x1, y1, SE, WBORDER;(
{

 */выдать заголовок окна/*
static dtitle(WINDOW *wnd(
}
    int x1 = 1, i, ln;
    char *s = WTITLE;

    if (!verify_wnd(&wnd((
        return;
    if (s}  (
        ln = strlen(s;(
        if (ln > WIDTH-2(
            i = 0;
        else
            i = ((WIDTH-2-ln) / 2;(
        if (i > 0(
            while (i(--
                displ(wnd, x1++, 0, LINE, WBORDER;(
        while (*s && x1 < WIDTH-1(
            displ(wnd, x1++, 0, *s++, WTITLEC;(
{
    while (x1 < WIDTH-1(
        displ(wnd, x1++, 0, LINE, WBORDER;(
{

 */оконно-ориентированная printf/*
void wprintf(WINDOW *wnd, char *ln(... ,
}
    char dlin [100], *dl = dlin;

    if (verify_wnd(&wnd}        ((
        va_list ap;
        va_start(ap, ln;(
        vsprintf(dlin, ln, ap;(
        va_end(ap;(
        while (*dl(
            wputchar(wnd, *dl;(++
{
{

 */записать символ в окно/*
void wputchar(WINDOW *wnd, int c(
}
   if (!verify_wnd(&wnd((
       return;
   switch (c}  (
```

```
       case '\n:'
           if (SCROLL == HEIGHT-3(
               scroll(wnd, UP;(
           else
               SCROLL;++
           WCURS = 0;
           break;
       case '\t:'
           do displ(wnd,(WCURS++)+3,SCROLL+1,'',WNORMAL;(
               while ((WCURS%TABS) && (WCURS+1) < WIDTH-1;(
           break;
       default; :
           if ((WCURS+1) < WIDTH-1}    (
               displ(wnd, WCURS+1, SCROLL+1, c, WNORMAL;(
               WCURS;++
{
           break;
{
{

 */установить курсор окна/*
void wcursor(WINDOW *wnd, int x, int y(
}
    if (verify_wnd(&wnd) && x < WIDTH-1 && y < HEIGHT-1}   (
        WCURS = x;
        SCROLL = y;
        cursor(COL+x+1, ROW+y+1;(
{
{

 */позволяет пользователю произвести оконный выбор/*
int get_selections(WINDOW *wnd, int s, char *keys(
}
    int c = 0, ky;
    if (!verify_wnd(&wnd((
        return 0;
    SELECT = s;
    while (c != ESC && c != '\r' && c != BS && c != FWD}    (
        accent(wnd;(
        c = get_char;()
        deaccent(wnd;(
        switch (c}      (
            case UP:    if (SELECT > 1(
                            SELECT;--
                        else
                            SELECT = SCROLL+1;
                                break;
            case DN:    if (SELECT < SCROLL+1(
                            SELECT;++
                        else
                            SELECT = 1;
                        break;
            case '\r:'
            case ESC:
            case FWD:
            case BS    :break;
            default:    if (keys}    (
                            ky = 0;
                            while (*(keys + ky}     ((
                                if (*(keys+ky)==toupper(c|| (
)*                                    keys+ky)==tolower(c((
```

```
                                    return ky + 1;
                                ky;++
{
{
          break;
{
{
    return  c == '\r' ?  SELECT : c == ESC ? 0 : c;
{

union REGS rg;
 */прокручивает содержимое окна вверх или вниз/*
void scroll(WINDOW *wnd, int dir(
}
   int row = HEIGHT-1, col, chat;

   if (!verify_wnd(&wnd((
       return;
   if (NEXT == NULL && HEIGHT > 3 && VISIBLE}   (
       rg.h.ah = dir == UP ? 6 : 7;
       rg.h.al = 1;
       rg.h.bh = WNORMAL;
       rg.h.cl = COL + 1;
       rg.h.ch = ROW + 1;
       rg.h.dl = COL + WIDTH - 2;
       rg.h.dh = ROW + HEIGHT - 2;
       int86(16, &rg, &rg;(
       return;
{
   if (dir == UP}  (
       for (row = 2; row < HEIGHT-1; row(++
           for (col = 1; col < WIDTH-1; col}  (++
               chat = dget(wnd, col, row;(
               displ(wnd,col,row-1,chat&255,(chat>>8)&255;(
{
       for (col = 1; col < WIDTH-1; col(++
           displ(wnd, col, row-1, ' ', WNORMAL;(
{
   else}
       for (row = HEIGHT-2; row > 1; --row(
           for (col = 1; col < WIDTH-1; col}  (++
               chat = dget(wnd, col, row-1;(
               displ(wnd,col,row,chat&255,(chat>>8)&255;(
{
       for (col = 1; col < WIDTH-1; col(++
           displ(wnd, col, row, '', WNORMAL;(
{
{

#ifndef FASTWINDOWS
 */вычисляет абрис отображаемого символа окна/*

static int *waddr(WINDOW *wnd, int x, int y(
}
   WINDOW *nxt = NEXT;
   int *vp;

   if (!VISIBLE(
       return (int *) (SAV+y*(WIDTH*2)+x*2;(
   x += COL;
   y += ROW;
```

```
   while (nxt} (
       if (nxt->_wv(
           if (x >= nxt->_wx && x <= nxt->_wx + nxt->_ww-1(
               if (y >= nxt->_wy&&
                       y <= nxt->_wy + nxt->_wh-1} (
                   x -= nxt->_wx;
                   y -= nxt->_wy;
                   vp = (int(*
))                       nxt->_ws) +y*(nxt->_ww*2)+x*2;(
                   return vp;
{
       nxt = nxt->_nx;
{
    return NULL;
{

 */выдать символ в окно/*
void displ(WINDOW *wnd, int x, int y, int ch, int at(
}

    int *vp;
    int vch = (ch&255)|(at<<8;(

    if ((vp = waddr(wnd, x, y)) != NULL(
*        vp = vch;
    else
        vpoke(VSG,vad(x+COL,y+ROW),vch;(
{

 */получить отображенный символ из окна/*
static int dget(WINDOW *wnd, int x, int y(
}
    int *vp;

    if ((vp = waddr(wnd, x, y)) != NULL(
        return *vp;
    return vpeek(VSG,vad(x+COL,y+ROW;((
{

 */видеофункции низкого уровня/*

 */обменивает содержимое видеообраза и буфера сохранения/*
static vswap(WINDOW *wnd(
}
    int x, y, chat;
    int *bf = (int *) SAV;

    for (y = 0; y < HEIGHT; y(++
        for (x = 0; x < WIDTH; x}  (++
            chat = *bf;
*            bf++ = dget(wnd, x, y;(
            displ(wnd, x, y, chat&255, (chat>>8)&255;(
{
{

#else

 */сохраняет видеопамять в буфере сохранения/*
static vsave(WINDOW *wnd(
}
    int x, y;
```

```
    int *bf = (int *) SAV;

    for (y = 0; y < HEIGHT; y(++
        for (x = 0; x < WIDTH; x(++
*            bf++ = vpeek(VSG, vad(x+COL, y+ROW;((
{

 */восстанавливает видеопамять из буфера сохранения/*
static vrstr(WINDOW* wnd(
}
    int x, y;
    int *bf = (int *) SAV;

    for (y = 0; y < HEIGHT; y(++
        for (x = 0; x < WIDTH; x(++
            vpoke(VSG,vad(x+COL, y+ROW), *bf;(++
{
#endif

 */заменяет яркость строки, указываемой SELECT/*
void acline(WINDOW *wnd, int set(
}
    int x, ch;

    if (!verify_wnd(&wnd((
        return;
    for (x = 1; x < WIDTH - 1; x} (++
        ch = dget(wnd, x, SELECT) & 255;
        displ(wnd, x, SELECT, ch, set;(
{
{

 */ФУНКЦИИ ОБРАБОТКИ СПИСКА/*

 */добавляет окно в конец списка/*
static add_list(WINDOW *wnd(
}
    if (listtail}    (
        PREV = listtail;
        listtail->_nx = wnd;
{
    listtail = wnd;
    if (!listhead(
        listhead = wnd;
{

 */добавляет окно в начало списка/*
static beg_list(WINDOW *wnd(
}
   if (listhead}    (
       NEXT = listhead;
       listhead->_pv = wnd;
{
   listhead = wnd;
   if (!listtail(
       listtail = wnd;
{

 */удаляет окно из списка/*
static remove_list(WINDOW *wnd(
```

```
}

   if (NEXT(
       NEXT->_pv = PREV;
   if (PREV(
       PREV->_nx = NEXT;
   if (listhead == wnd(
       listhead = NEXT;
   if (listtail == wnd(
       listtail = PREV;
   NEXT = PREV = NULL;
{

 */вставляет w 1 после  w 2/*
static insert_list(WINDOW *w1, WINDOW *w2(
}
   w1->_pv = w2;
   w1->_nx = w2->_nx;
   w2->_nx = w1;
   if (w1->_nx == NULL(
       listtail = w1;
   else
       w1->_nx->_pv = w1;
{
#ifndef FASTWINDOWS
 */проверяет наличие окна в списке/*
static verify_wnd)WINDOW **w1(
}
   WINDOW *wnd;

   wnd = listhead;
   if (*w1 == NULL(
*       w1 = listtail;
   else}
       while (wnd != NULL} (
           if (*w1 == wnd(
               break;
           wnd = NEXT;
{
{
   return wnd != NULL;
{
#endif

WINDOW *ewnd = NULL;

 */сообщение об ошибках/*
void error_message(char *s(
}
    ewnd = establish_window(50, 22, 3, max(10, strlen(s)+2;((
    set_colors(ewnd, ALL, RED, YELLOW, BRIGHT;(
    set_title(ewnd, " ERROR;(" !
    display_window(ewnd;(
    wprintf(ewnd, s;(
    putchar(BELL;(
{

void clear_message()
}
   if (ewnd(
       delete_window(ewnd;(
```

```
   ewnd = NULL;
{
```

Описание программы: twindow.c

Далее описывается исходная программа twindow.c. Для каждой функции описывается, что она делает и как работает. Программист может использовать эти описания для понимания текста программы.

Объявления внешних данных в twindow.c включают прототипы для каждой функции, локальные в исходном файле, массив структур для определения пяти типов рамки окна, головной и хвостовой указатели для списка структур WINDOW.

Рамка окна управляется элементом структуры WINDOW, которая устанавливает окно. Этот элемент является целочисленным смещением в таблице типов рамки. Вход, на который указывает смещение, содержит шесть значений, каждое из которых представляет одну из сторон или узлов окна. Первое значение определяет верхний левый или северо-западный угол. Имя переменной (nw, ne, se, sw(сообщает вам, какой угол определяется. Целое число side относится к вертикальным сторонам рамки; целое число line соответствует верхней и нижней горизонтальным линиям рамки. Значения относятся к символам из набора графических символов ПЭВМ.

Две WINDOW-ссылки listhead и listtail являются головным и хвостовым указателями для списка окон. Когда создаются окна, они добавляются к этому списку. Первоначально эти два указателя равны NULL. Когда создается первое окно, выделяется память для структуры WINDOW, и ее адрес копируется в оба указателя. У списка имеется голова, указывающая на первое окно списка, и хвост, указывающий на последнее. Когда создается второе окно, его адрес копируется в хвостовой указатель. Кроме того, адрес второго окна записывается в указатель _nx в первой структуре WINDOW, а адрес первой записывается в указатель _pv второй. Голова списка указывает на первое окно, которое указывает на второе и т.д. Хвост списка указывает на последнее окно. Каждое окно также указывает на своего предшественника в цепи, следовательно, список является двунаправленной структурой данных, называемой двусвязным списком. (Для знакомства со списковыми структурами данных см:. Brady. C Development Tools for the IBM PC. - 1986.(.

Функция establish_window инициализирует переменную VSG адресом сегмента видеопамяти. Функция распределяет память для структуры WINDOW и инициализирует эту структуру оконными характеристиками, принимаемыми по умолчанию, а также размером и координатами, заданными при вызове функции. Она выделяет память для буфера сохранения видеопамяти и записывает адрес буфера в структуру WINDOW. После инициализации структуры функция вызывает add_list для добавления структуры к списку окон. Текстовая область окна очищается, и образ окна выделяется, если обрабатываются слоеные окна. Эти функции оперируют в буфере сохранения, поэтому окно пока не изображается. Функция establish_ window возвращает адрес структуры WINDOW в точку вызова.

Функции set_border, set_colors, set_intensity и set_title модифицируют характеристики созданного окна. Сначала они вызывают verify_wnd для проверки того, что при вызове передан адрес созданного окна. Затем они модифицируют заданный атрибут. В конце

они вызывают функцию redraw для записи изменений на экран.

Функция redraw перевыдает окно ,если обрабатываются слоеные окна.

Функция display_window оперирует по-разному для стековых и слоеных окон. В любом случае она ничего не делает, если окно является видимым для пользователя. Если окно невидимо, то путем вызова функции vswap display_window замещает видеопамять буфером сохранения, если действуют слоеные окна. Для стековых окон делается проверка, не скрыто ли окно. Если окно скрыто, то оконный буфер сохранения записывается в видеопамять вызовом vrstr .Если окно не скрыто, то оно никогда не выдается, поэтому вызывается vsave для сохранения текущего содержимого видеопамяти, a clear_window и wframe вызываются для выдачи пустого окна.

Функция close_all уничтожает все окна путем прохода по списку структур WINDOW и вызова delete_window.

Функция delete_window удаляет окно из системы путем его скрытия, освобождения памяти, занятой буфером сохранения, удаления структуры WINDOW из списка и освобождения памяти, содержащей структуру WINDOW.

Функция hide_window вызывает vswap для замены буфера сохранения видеопамятью для слоеного окна и вызова vrstr для восстановления видеопамяти для стекового окна.

Функция repos_window имеется только для слоеных окон. Она вызывается одним из макросов move_window, rmove_window, rear_window и forefront. Она изменяет положение окна путем создания временного окна, помещая временное окно в список в соответствии с информацией, полученной из макроса, записывая оригинальное содержимое окна в буфер сохранения временного окна, выдавая временное окно и скрывая оригинал.

Crear_window записывает пробелы в область данных окна, а wframe и dtitle изображают окно с заголовком наверху. Эти функции используют функцию displ для записи значений в окно.

Функция wprintf является примером нового предполагаемого стандарта ANSI для функций с переменным числом параметров. В прошлом большинство компиляторов обрабатывали printf на ассемблере для просмотра переменного числа параметров из стека. Предполагаемый стандарт использует многоточие (...) в списке параметров функции для указания присутствия переменного числа параметров с различными типами данных. Специальный тип массива va_list используется для объявления списка, а va_start устанавливает начало и конец списка. Функция vsprintf является версией sprintf, которая допускает параметр va_list. В данном случае параметры, передаваемые в wprintf, перерабатываются vsprintf в строку с именем dlin. Затем строка выдается в окно по одному символу за раз путем вызова wputchar. Если у вас получится вызов wprintf, который образует строку более 100 символов, придется увеличить длину массива dlin.

Функция wputchar выдает символ в окно в текущей позиции курсора. Расположение оконного курсора является функцией двух элементов структуры WINDOW, которые указываются макросами WCURS)столбец) и SCROLL (строка). Функция реагирует на символы новой строки (\n) и табуляции (\t) следующим образом. Для новой

строки, если переменная SCROLL соответствует низу окна, содержимое окна проворачивается вверх на одну строку; иначе значение переменной SCROLL увеличивается. В любом случае переменная WCURS устанавливается на столбец 0. Если в wputchar послан символ табуляции, переменная WCURS продвигается к следующей позиции табуляции в окне. Остальные символы отображаются в окне, а переменная WCURS возрастает. Строки, длина которых превышает ширину окна, не переносятся;

они обрезаются.

Функция wcursor устанавливает переменные WCURS и SCROLL на координаты, заданные при вызове. Она также устанавливает видеокурсор на экранную позицию, соответствующую оконному курсору.

Функция get_selection создает блок курсора в окне и позволяет пользователю перемещать блок вверх и вниз, а также производить выбор нажатием клавиши <Ввод>. Макроопределение SELECT ссылается на переменную в структуре WINDOW и используется для перемещения блока курсора в окне. Функции accent и deaccent используются для включения и выключения блока курсора путем изменения видеоатрибута строки на ACCENT и NORMAL. При нажатии верхней и нижней клавиш со стрелками функция изменяет значение переменной SELECT. При вызове можно также передать адрес массива символов, содержащего список клавиш, используемых для выбора из окна. Если пользователь нажимает одну из этих клавиш, производится соответствующий выбор так же, как если бы блок курсора находился в соответствующей строке и была нажата клавиша >Ввод.<

Функция scroll проворачивает порцию текста в окне вверх или вниз на одну строку. Если окно является последним и видимым, функция прокрутки ROM-BIOS применяется для ускорения по сравнению с программной прокруткой. Функция ROM-BIOS не применяется, если окно имеет только одну строку текста из-за ошибки в IBM PC и некоторых моделях AT. Эта ошибка вызывает появление неверных видеорезультатов при попытке провернуть единственную строку. Ошибка была устранена IBM в AT BIOS, но в некоторых моделях она осталась. Если окно не является последним или если оно имеет одну строку текста, текстовая область проворачивается программно с помощью функций dget и displ для чтения и записи символов текста из окна и в окно.

Функция waddr оперирует только со слоеными окнами. Она возвращает целочисленный адрес позиции в окне, где расположены символ и атрибут. Если окно не видимо, функция возвращает адрес в буфере сохранения, вычисленный по координатам x и y. Если окно видимо, сканируется список для поиска окон, более поздних, чем адресуемое окно. Если более позднее окно закрывает позицию адресуемого символа, возвращается адрес, соответствующий сохраненному адресу этого окна .Если ни одно более позднее окно не закрывает позицию символа, возвращается указатель NULL для сообщения в точку вызова об использовании видеопамяти.

Функция displ и функция dget вызываются для выдачи и приема видеосимвола и атрибута в и из слоеного окна. Эти функции вызывают waddr для проверки необходимости чтения или записи в область сохранения. Если нет, адресуется видеопамять.

Функция wsvap меняет местами содержимое буфера сохранения

слоеного окна и видеопамяти или, возможно, буферов сохранения более поздних окон, которые закрывают адресуемое окно. Эта функция использует displ и dget для выполнения изменения.

Функции vsave и vrstr работают со стековыми окнами. Vsave копирует содержимое видеопамяти в буфер сохранения, а vrstr копирует буфер сохранения в видеопамять.

Функция acline вызывается макросами accent и deaccent для изменения выбранной строки окна на цветовую конфигурацию ACCENT или NORMAL.

Функция add_list добавляет структуру WINDOW в конец списка.

Функция beg_list добавляет структуру WINDOW в начало списка.

Функция remove_list удаляет структуру WINDOW из списка.

Функция iusert_list вставляет структуру WINDOW в список после другой заданной структуры WINDOW.

Функция verify_wnd ищет в списке заданный адрес структуры WINDOW. Она возвращает истину или ложь в зависимости от наличия или отсутствия структуры WINDOW в списке. Если заданный указатель WINDOW равен NULL, функция возвращает адрес последней структуры WINDOW в списке.

Функция error_message создает окно для выдачи специального сообщения об ошибке. Сообщение записывается в окно вызовом wprintf, и включается звуковой сигнал.

Функция clear_message очищает последнее сообщение об ошибке.

Примеры окон

Далее рассматриваются возможности оконной библиотеки. Приводятся примеры программ, каждая из которых иллюстрирует рассматриваемые возможности. Примеры состоят из небольшой управляющей программы с главной функцией, которая вызывает функцию примера для демонстрируемой возможности. Функция примера содержит вызовы ранее рассмотренных библиотечных функций и служит иллюстрацией их использования. Каждый пример программы включает проектный (.prj) файл, используемый Турбо Си для построения выполняемой программы.

Затем эти же самые примеры функций будут объединены в один выполняемый модуль, который демонстрирует оконные меню. Поэтому они написаны без собственных main-функций.

Перемещение окна

При использовании слоеных окон вам доступны функции, позволяющие перемещать окно в абсолютную или относительную позицию на экране. Заметим, что эти функции - move_window и rmove _window - недоступны для стековых окон.

Программа, иллюстрирующая перемещение окна, приведена на листингах 6.3, 6.4 и 6.5. Листинг 6.3 является маленькой управляющей программой, а листинг 6.5 представляет проектный make -файл. Обращайтесь к листингу 6.4, testmove.c, при чтении данного описания.

Для запуска примера введите следующую команду:

c>move

) В этом и последующих примерах предполагается, что C является вашим системным дисководом. Не вводите подсказку c.(.<

Помимо иллюстрации движения окна, testmove.c показывает также процесс создания окон, установку их цветов, выдачу на экран и запись текста в них. Программа создает три окна, присваивает каждому из них собственные цвета, выдает их и записывает цитату во второе из трех окон. Этот пример иллюстрирует возможность записи текста в окно, которое частично закрыто другим окном. После запуска программы вы увидите на дисплее то, что показано на рисунке 6.4.

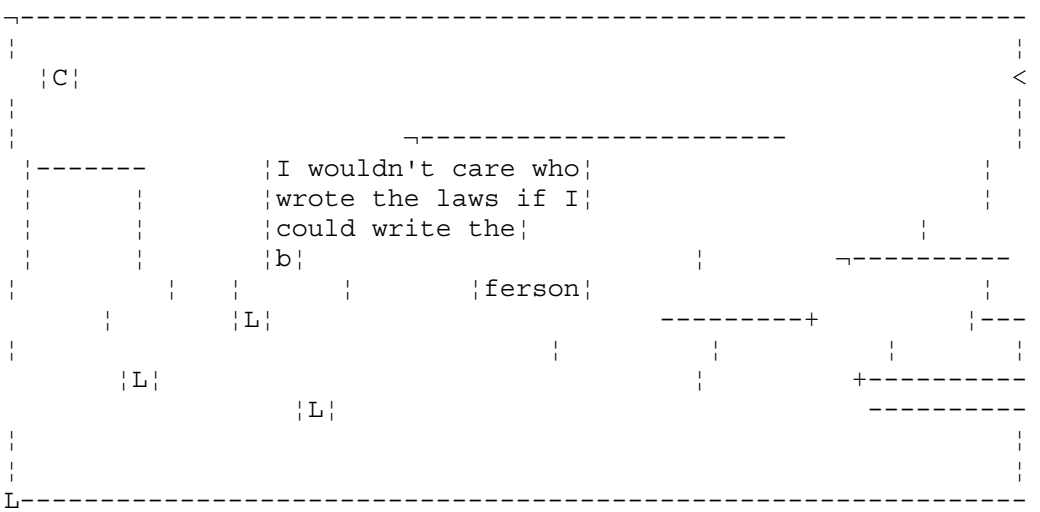


Рисунок 6.4. Перемещение слоеных окон.

Теперь программа ожидает нажатия клавиши. Программа специально ждет нажатия одной из клавиш управления курсором или клавиши <Ключ>. Каждое нажатие клавиши со стрелкой вызывает перемещение окна на одну символьную позицию в направлении стрелки. Функция rmove_window используется для перемещения окна. Обратите внимание на то, как центральное окно перемещается между двумя остальными.

Когда вы нажимаете клавишу <Ключ ,<открытое первым окно уничтожается. Еще одно нажатие вызывает уничтожение верхнего окна. Заключительное нажатие уничтожает среднее окно с цитатой и завершает программу.

Перемещение окна иллюстрирует использование оконных буферов сохранения, запрограммированное в библиотеке. Поскольку требуется определенная обработка для анализа каждого буфера при записи символа в окно, работа функций усложняется по мере увеличения размера окон и их количества. При перемещении большого слоеного

окна на старых медленных моделях ПЭВМ это можно наблюдать визуально.

Листинг 6.3: move.c

```
 */move.c/*
void testmove(void;(

main()
}
   testmove;()
{
```

Листинг 6.4: testmove.c

```
 */testmove.c/*
#include "twindow.h"
#include "keys.h"

void testmove()
}
   WINDOW *wndA, *wndB, *wndC;
   int c;

   wndA = establish_window(5, 5, 9, 19;(
   wndB = establish_window(10, 3, 9, 23;(
   wndC = establish_window(13, 8, 9, 12;(
   set_colors(wndA, ALL, RED,  YELLOW, BRIGHT;(
   set_colors(wndB, ALL, AQUA, YELLOW, BRIGHT;(
   set_colors(wndC, ALL, WHITE, YELLOW, BRIGHT;(
   display_window(wndA;(
   displey_window(wndB;(
   display_window(wndC;(
   wprintf(wndB, "\n I wouldn't care who;("
   wprintf(wndB, "\n wrote the laws if I;("
   wprintf(wndB, "\n could write the;("
   wprintf(wndB, "\n ballads;(".
   wprintf(wndB, "\n\n    Thomas Jefferson;("
   do}
       int x = 0, y = 0;
       c = get_char;()
       switch (c}  (
           case FWD:   x;++
                       break;
           case BS:    --x;
                       break;
           case UP:    --y;
                       break;
           case DN:    y;++
           default:    break;
{
       if (x || y(
           rmove_window(wndB, x, y;(
 {   while (c != ESC;(
     delete_window(wndA;(
     get_char;()
     delete_window(wndC;(
     get_char;()
     delete_window(wndB;(
{
```

Листинг 6.5: move.prj

```
move
testmove (twindow.h, keys.h(
twindow (twindow.h, keys.h(
ibmpc.obj
```

Подъем и опускание окон

С помощью функций **forefront** и **rear_window** вы можете поднять окно в последнюю позицию, помещая его поверх всех остальных, а также опустить окно в первую позицию, помещая его под всеми остальными. Эта возможность применима в программах с несколькими окнами, где пользователь выбирает одно из них для некоторой цели путем временного перевода остальных на фон. Эта возможность полезна в любых приложениях, в которых пользователь часто переходит от окна к окну.

Программа, иллюстрирующая подъемы и опускания окон, приведена в листингах 6.6, 6.7 и 6.8. Листинг 6.6 является маленькой управляющей программой, а листинг 6.8 - проектным make-файлом. Обращайтесь к листингу 6.7, **promote.c**, при чтении данных разъяснений.

Для запуска примера введите следующую команду:

```
c>prom
```

Программа **promote.c** использует те же образцы трех окон, что и программа **testmove.c**. Теперь все три окна включают записанный в них текст, каждый из которых содержит имя окна: **window A**, **window B** и **window C**. На рисунке 6.5 показан первоначально выданный экран.

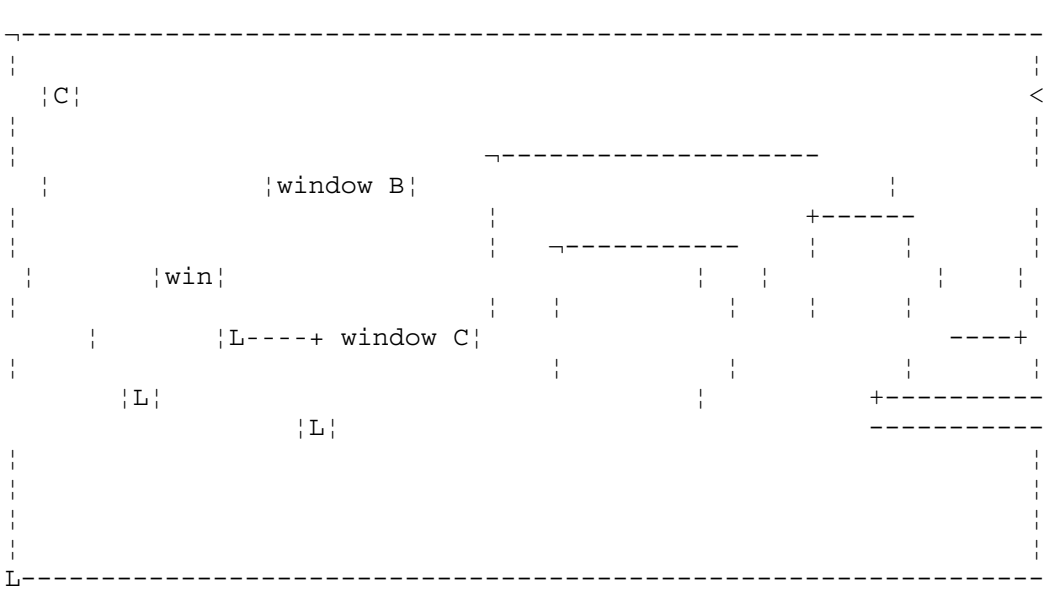


Рисунок 6.5. Подъем слоеных окон.

Для подъема и опускания окон используется клавиатура. Используйте нажатия клавиш с маленькими буквами a, **b** и c для подъема окон, названных этими буквами. Используйте клавиши с большими буквами для их опускания. Этот процесс продолжается до

тех пор, пока вы не нажмете клавишу <Ключ> для уничтожения одного из окон. Еще два нажатия вызовут уничтожение остальных двух окон, и программа завершится.

Листинг 6.6: prom.c

```
 */prom.c/*
void promote(void;(

main()
}
   promote;()
{
```

Листинг 6.7 :promote.c

```
 */promote.c/*
#include "twindow.h"
#include "keys.h"

void promote()
}
    WINDOW *wndA, *wndB, *wndC;
    int c;
    wndA = establish_window(5, 5, 9, 19;(
    wndB = establish_window(10, 3, 9, 20;(
    wndC = establish_window(13, 8, 9, 12;(
    set_colors(wndA, ALL, RED, YELLOW, BRIGHT;(
    set_colors(wndB, ALL, AQUA, YELLOW, BRIGHT;(
    set_colors(wndC, ALL, WHITE, YELLOW, BRIGHT;(
    display_window(wndA;(
    display_window(wndB;(
    display_window(wndC;(
    wprintf)wndA, "\n\n Window A;("
    wprintf(wndB, "\n\n Window B;("
    wprintf(wndC, "\n\n Window C;("
    do}
        c = get_char;()
        switch (c}  (
            case 'a':   forefront(wndA;(
                        break;
            case 'b':   forefront(wndB;(
                        break;
            case 'c':   forefront(wndC;(
                        break;
            case 'A':   rear_window(wndA;(
                        break;
            case 'B':   rear_window(wndB;(
                        break;
            case 'C':   rear_window(wndC;(
                        break;
            default:    break;
{
 {while (c != ESC;(
     delete_window(wndA;(
     get_char;()
     delete_window(wndC;(
     get_char;()
     delete_window(wndB;(
{
```

Листинг 6.8: prom.prj

```
prom
promote (twindow.h, keys.h(
twindow (twindow.h, keys.h(
ibmpc.obj
```

Назначение заголовков и изменение цветов окна

Когда окно создано, вы можете назначить ему заголовок, а также установить цвета переднего плана и фона.

Программа, иллюстрирующая заголовки и цвета окон, приведена в листингах 6.9, 6.10 и 6.11. Листинг 6.9 является маленькой управляющей программой, а листинг 6.11 - проектным make-файлом. Обращайтесь к листингу 6.10, ccolor.c, при чтении данных разъяснений.

Для запуска примера вводите следующую команду

```
c>color
```

Снова выдается три окна. Каждое из них имеет свой цвет (если у вас цветной монитор), но ни у одного из них нет заголовка. Окна расположены в тех же местах и имеют те же размеры, что и в предыдущих примерах. Программа ожидает нажатия клавиши с одной из букв: r, g или b. Она будет использовать эти буквы для изменения заголовка среднего окна на "RED", "GREEN" или "BLUE", а также изменит цвет среднего окна на соответствующий новому заголовку. Рисунок 6.6 показывает экран после выбора алого (red) окна.

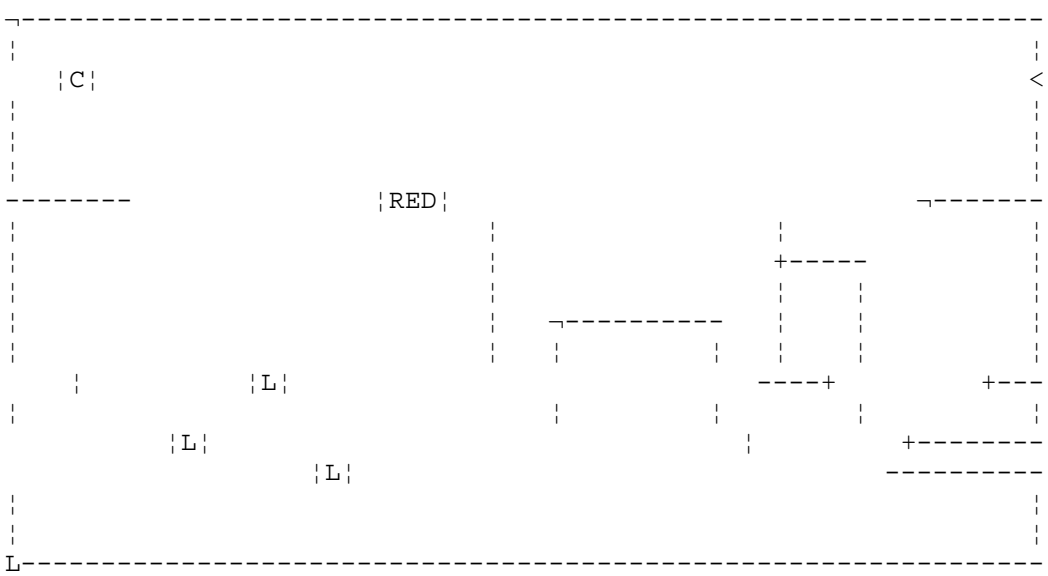


Рисунок 6.6. Изменение цветов и заголовков.

Нажмите клавишу <Ключ> для выхода и уничтожения окна, а также любые две другие клавиши для уничтожения двух остальных окон и завершения программы.

Листинг 6.9: color.c

```
 */color.c/*

void ccolor(void;(

main()
}
   ccolor;()
{
```

Листинг 6.10: ccolor.c

```
 */ccolor.c/*

#include "twindow.h"
#include "keys.h"

void ccolor()
}
   WINDOW *wndA, *wndB, *wndC;
   int c;

   wndA = establish_window(8, 8, 9, 19;(
   wndB = establish_window(13, 6, 9, 20;(
   wndC = establish_window(16, 11, 9, 12;(
   set_colors(wndA, ALL, RED, YELLOW, BRIGHT;(
   set_colors(wndB, ALL, AQUA, YELLOW, BRIGHT;(
   set_colors(wndC, ALL, WHITE, YELLOW, BRIGHT;(
   display_window(wndA;(
   display_window(wndB;(
   display_window(wndC;(

   do}
       c = get_char;()
       switch (c}  (
           case 'r:'
               set_title(wndB, " RED;("
               set_colors(wndB, ALL, RED, WHITE, BRIGHT;(
               break;
           case 'b:'
               set_title(wndB, " BLUE;("
               set_colors(wndB, ALL, BLUE, WHITE, BRIGHT;(
               break;
           case 'g:'
               set_title(wndB, " GREEN;("
               set_colors(wndB, ALL, GREEN, WHITE, BRIGHT;(
               break;
           default:
               break;
{
 {     while (c != ESC;(
     delete_window(wndA;(
     get_char;()
     delete_window(wndC;(
     get_char;()
     delete_window(wndB;(
{
```

Листинг 6.11: color.prj

```
color
```

```
ccolor (twindow.h, keys.h(
twindow (twindow.h, keys.h(
ibmpc.obj
```

Сравнение стековых и слоеных окон

Рассматриваемая здесь программа дает возможность сравнить представления стековых и слоеных окон. Вы можете откомпилировать ее с оконной библиотекой, которая была компилирована для обоих типов окон.

Программа, иллюстрирующая различия между стековыми и слоеными окнами, показана на листингах 6.12, 6.13 и 6.14. Листинг 6.12является маленькой управляющей программой, а листинг 6.14- проектным make-файлом. Обращайтесь к листингу 6.13, fast.c, при чтении данных разъяснений.

Для запуска примера введите следующую команду:

```
c>fast
```

Помните, что программа сама по себе не показывает сравнение. Вам необходимо построить две версии ее, одну - со стековой оконной библиотекой, а вторую - со слоеной оконной библиотекой, и сравнить представления обеих программ.

После запуска программы она создаст и выдаст пятнадцать окон в последовательности, показанной на рисунке 6.7. Когда вы нажимаете любую клавишу, программа уничтожает каждое окно в обратном порядке. Вы можете сравнить соответствующие скорости работы программ, компилированных в двух средах.

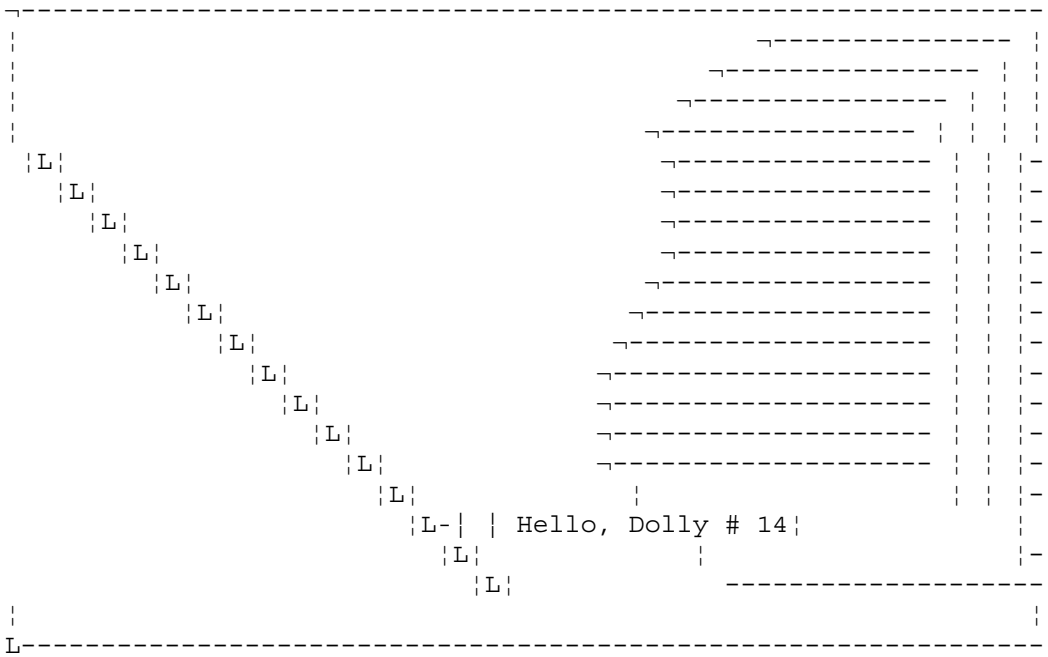


Рисунок 6.7. Сравнение стековых и слоеных окон.

Листинг 6.12: fast.c

```
 */fast.c/*
void fasttest(void;(

main()
}
    fasttest;()
{
```

Листинг 6.13: fasttest.c

```
 */fasttest.c/*
#include <stdio.h<
#include "twindow.h"

void fasttest()
}
   int row, col;

   for (row = 0, col = 0; col < 15; row += 3, col}   (++
       establish_window(row, col, 10, 30;(
       set_colors(NULL, ALL, RED, YELLOW, BRIGHT;(
       display_window(NULL;(
       wprintf(NULL, "\n\n\n  Hello, Dolly # %d", col;(
{
   get_char;()
   while (col(--
       delete_window(NULL;(
{
```

Листинг 6.14: fast.prj

```
fast
fasttest (twindow.h(
twindow (twindow.h, keys.h(
ibm.obj
```

Перемещение, подъем, скрытие окон, меню, изменение интенсивности

В следующем примере комбинируется несколько уже показанных возможностей и даются примеры еще двух возможностей: использование get_selection для обработки простого меню и использование set_intensity для изменения яркости переднего плана окон.

Программа, иллюстрирующая эти возможности, представлена на листингах 6.15, 6.16 и 6.17. Листинг 6.15 является маленькой управляющей программой, а листинг 6.17 - проектным make-файлом. Обращайтесь к листингу 6.16, poems.c, при чтении данных разъяснений.

Для запуска примера введите следующую команду:

```
c>poetry
```

Эта программа выдает пять различных стихотворений на экран и позволяет вам выбрать одно из них, перемещать его, поднять его на передний план, опустить его назад и уничтожить. Программа начинает работу с выдачи оконного меню, которое перечисляет все стихотворения. Вы можете переместить курсор вверх или вниз и выбрать одно стихотворение нажатием клавиши <Ввод>. Вы также

можете нажать одну из цифр от 1 до 5, выбирая номер стихотворения. В результате будет показано выбранное стихотворение. На рисунке 6.8 показано меню стихотворений.

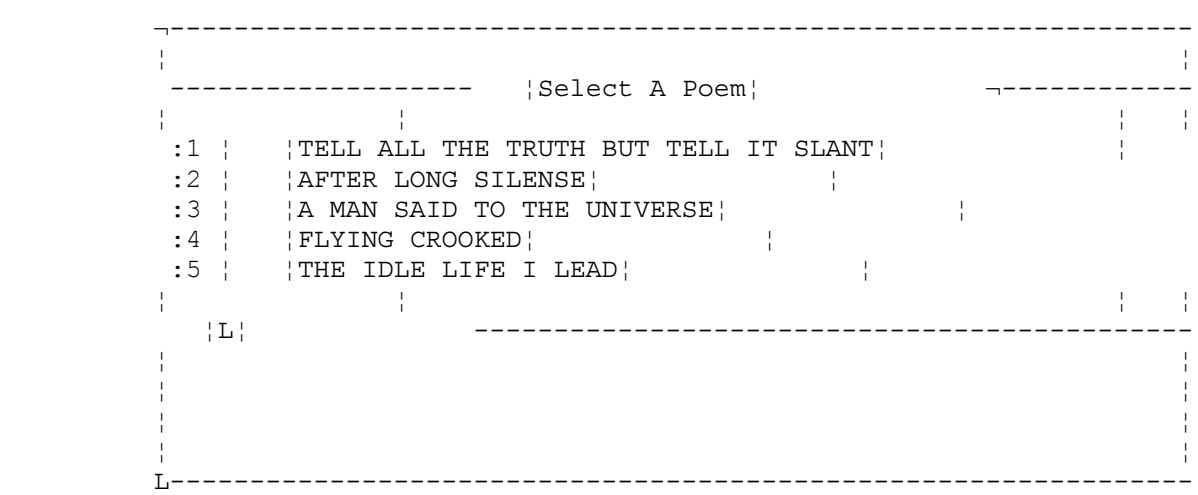
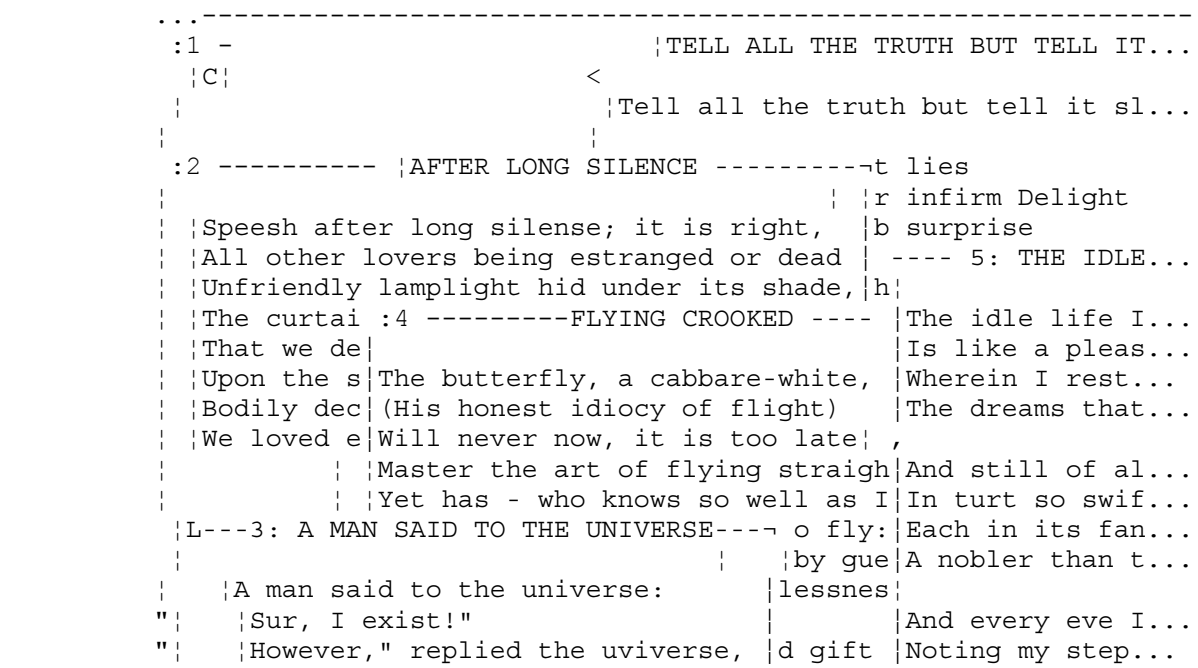


Рисунок 6.8. Меню стихотворений.

После выбора стихотворения вы можете передвинуть его клавишами со стрелками, возвращаясь к меню нажатием клавиши >Ключ> или выбирая другое стихотворение нажатием соответствующей цифровой клавиши. Выбранное текущее стихотворение выдается с повышенной яркостью, а все остальные - с обычной. Если вы выбираете стихотворение, выданное с обычной яркостью, то оно становится ярким, а остальные - обычными. Для перевода

стихотворения на передний план нажмите клавишу <Плюс> (+); для посылки его на фон нажмите клавишу <Минус> (-). Для уничтожения текущего стихотворения нажмите клавишу <Удл>. Вы можете восстановить его нажатием клавиши с номером. После этого нажмите клавишу <Ключ> для возврата к меню, а затем снова клавишу <Ключ< для выхода из программы.

На рисунке 6.9 показаны стихотворения, разбросанные по экрану в различных местах.



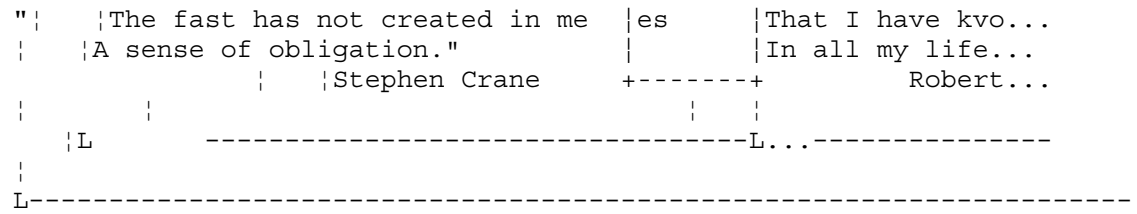


Рисунок 6.9. Стихотворения.

Листинг 6.15: poetry.c

```
 */poetry.c/*
#include "twindow.h"
void poems(void;(

main()
}
    load_help("tcprogs.hlp;("
    poems;()
{
```

Листинг 6.16: poems.c

```
 */poems.c/*
#include <stdio.h<
#include <string.h<
#include <stdlib.h<
#include "twindow.h"
#include "keys.h"

 */локальные прототипы/*
void get_poem(int s;(
int ht (char **tb;(
int wd (char **tb;(
char *titles} = []
 :1 "    TELL ALL THE TRUTH BUT TELL IT SLANT,"
 :2 "    AFTER LONG SILENSE,"
 :3 "    A MAN SAID TO THE UNIVERSE,"
 :4 "    FLYING CROOKED,"
 :5 "    THE IDLE LIFE I LEAD ",0;{
WINDOW *pno [] = {0, 0, 0, 0, 0;{
static int x [] = {20, 15, 29, 10, 17;{
static int y [] = {5,10, 13, 18, 6;{
static int wcl [] [2] = {  {BLUE, WHITE,{
}                           MAGENTA, WHITE,{
}                           RED, WHITE,{
}                           GREEN, WHITE,{
}                           AQUA, WHITE;{   {
char *poem1} = []
"    Tell all the truth but tell it slant,"-
"    Success in Circuit lies,"
"    Too bright for our infirm Delight,"
"    The Truth's superb surprise,"
,""
"    As Lightning to the Children eased,"
"    With explanation kind,"
"    The Truth must dazzle gradually,"
"    Or every man be blind,"-
                "    Emily Dickenson",0;{
char *poem2} = []
"    Speech after long silence; it is right,",
```

```
"    All other lovers being estranged or dead,",
"    Unfriendly lamplight hid under its shade,",
"    The curtains drawn upon unfriendly night,",
"    That we descant and yet again descant,"
"    Upon the supreme theme of Art and Song,":
"    Bodily decrepitude is wisdom; young,"
"    We loved each other and were ignorant,".
             "    William Butler Yeats",0;{


char *poem3} = []
"    A man said to the universe,":
"\"    Sir, I exist,""\!
"\"    However,\" replied the universe,",
"\"    The fast has not created in me,"
"    A sense of obligation,""\.
              "    Stephen Crane",0;{

char *poem4} = []
"    The butterfly, a cabbage-white,",
)"    His honest idiocy of flight,"(
"    Will never now, it is too late,",
"    Master the art of flying straight,",
"    Yet has - who knows so well as I,"- ?
"    A just sense of how not to fly,":
"    He lurches here and there by guess,"
"    And God and hope and hopelessness,".
"    Even the aerobatic swift,"
"    Has not his flyihg-crooked gift,".
              "    Robert Graves",0;{

char *poem5} = []
"    The idle life I lead,"
"    Is like a pleasant sleep,",
"    Wherein I rest and heed,"
"    The dreams that by me sweep,".
,""
"    And still of all my dreams,"
"    In turn so swiftly past,",
"    Each in its fancy seems,",
"    A nobler than the last,".
,""
"    And every eve I say,",
"    Noting my step in bliss,",
"    That I have known no  day,"
"    In all my life like this,".
         "    Robert Bridges",0;{

char **poem [] = { poem1, poem2, poem3, poem4, poem5, 0;{

void poems()
}
    int s = 0, i, c;
    WINDOW *mn;
    char **cp;

    cursor(0, 25;(
    mn = establish_window(0, 0, 7, 45;(
    set_title(mn, " Select A Poem;("
    set_colors(mn, ALL, BLUE, GREEN, BRIGHT;(
    set_colors(mn, ACCENT, GREEN, WHITE, BRIGHT;(
```

```
    display_window(mn;(
    cp = titles;
    while (*cp(
        wprintf(mn, "\n%s", *cp;(++

    while (1}   (
        set_help("poemmenu;(10 ,40 ,"
        s = get_selection(mn, s+1, "12345;("
        if (s == 0(
            break;
        if (s == FWD || s == BS}       (
            s = 0;
            continue;
{
        hide_window(mn;(
        get_poem(--s;(
        c = 0;
        set_help("poems   ", 5, 15;(
        while (c != ESC}    (
            c = get_char;()
            switch (c}  (
                case FWD:   rmove_window(pno[s], 1, 0;(
                            break;
                case BS:    rmove_window(pno[s], -1,0;(
                            break;
                case UP:    rmove_window(pno[s], 0, -1;(
                            break;
                case DN:    rmove_window(pno[s], 0, 1;(
                            break;
                case DEL:   delete_window(pno[s;([
                            pno[s] = NULL;
                            break;
                case '+':   forefront(pno[s;([
                            break;
                case '-':   rear_window(pno[s;([

                default:    break;
{
            if (c > '0' && c < '6('
                get_poem(s = c - '1;('
{
         forefront(mn;(
         display_window(mn;(
{
     close_all;()
     for (i = 0; i < 5; i(++
         pno[i] = NULL;
{
 */активизирует стихотворение по номеру/*
static void get_poem(int s(
}
    char **cp;
    static int lastp = -1;
    if (lastp != -1(
        set_intensity(pno[lastp], DIM;(
    lastp = s;
    if (pno [s([
        set_intensity(pno[s], BRIGHT;(
    else}
        pno [s] = establish_window
)            x[s], y[s], ht(poem[s]), wd(poem[s;(([
```

```
        set_title(pno[s], titles[s;([
        set_colors(pno[s],ALL,wcl[s][0],wcl[s][1], BRIGHT;(
        set_border(pno[s], 1;(
        display_window(pno[s;([
        cp = poem[s;[
        while (*cp(
            wprintf(pno[s], "\n %s", *cp;(++
{
{

 */вычисляет высоту показываемой таблицы окна/*
static int ht(char **tb(
}
    int h = 0;
    while (*(tb + h; ((++
    return h + 3;
{
 */вычисляет ширину показываемой таблицы окна/*
static int wd(char **tb(
}
    int w = 0;
    while (*tb}     (
        w = max(w, strlen(*tb;((
        tb;++
{
    return w + 4;
{
```

Листинг 6.17: poetry.prj

```
poetry
poems (twindow.h, keys.h(
thelp (twindow.h, keys.h(
twindow (twindow.h, keys.h(
ibmpc.obj
```

В программе poetry клавиша <F1> используется в качестве функциональеной клавиши контекстно-управляемой подсказки. Когда вы нажимаете <F1 ,<появляется окно с подсказывающим сообщением, относящимся к тому, что вы сейчас делаете. В Главе 7 объясняется, как включается эта возможность.

Резюме

Теперь у вас имеется основа для создания оконного программного инструментария. С помощью этих функций вы можете добавить окна к своему программному обеспечению, а также отображать в них текст. Однако приложение окон может быть в дальнейшем развито до возможностей более высокого уровня, которые являются общими во многих прикладных системах. Несколько последующих глав добавляют эти возможности к вашей оконной библиотеке. Глава 7 вводит использование окон для добавления контекстно-управляемой пользовательской подсказки в ваши программы.

ГЛАВА 7

Контекстно-управляемые окна подсказки

Первой проблемой, обычно возникающей при запуске новой программы, является незнакомство программы с языком своего пользователя. Какая клавиша должна быть нажата? Какое действие будет следующим? Что выполняет данный элемент меню? Независимо от того, сколько усилий вложено в разработку самообъясняемого пользовательского языка, у пользователей всегда возникают вопросы, поскольку язык новой системы для него всегда иностранный. Дело не только в знакомстве пользователей с языком, часто они даже не знают, что система может, а чего не может делать. По традиции эта проблема решается обращением к печатным руководствам пользователя и, возможно, автоматизированным справочникам. Недостатком этих решений является необходимость переключения внимания пользователя от работы с системой к чтению руководства или запуску справочника.

Если бы экраны были достаточно большими, система могла бы поддерживать постоянный показ руководства пользователя. Как только пользователю потребовалась бы информация, руководство было бы доступно. Это решение не продержалось бы долго; когда пользователь уже знает систему, информация подсказки не нужна и нежелательна. Значимая для новичка информация - это мусор для ветерана.

Большинство интерактивных систем имеют общее свойство: когда пользователю необходима информация, он смотрит на экран и гадает, какую клавишу нажать. Представляется вполне естественным, что в число нажатий клавиш, которые система будет распознавать в любое время, нужно включить функциональную клавишу <Help> (Подсказка.(Нажмите требуемую прикладную клавишу, и программа выполнит ваш запрос; нажмите функциональную клавишу подсказки, и программа сообщит вам кое-что о том, какие клавиши она ожидает и что случится при их нажатии.

Поскольку интерактивные системы используют экран для общения с пользователем и показа данных, сообщение подсказки - это материал для выталкиваемого вверх окна. Такое построение окна позволяет получить подсказку без нарушения прикладного использования экрана. Такое окно называется контекстно-управляемым окном подсказки, окном, которое выскакивает для подсказки пользователю при нажатии назначенной функциональной клавиши подсказки. (Программная промышленность признала <F1< стандартом для функциональной клавиши подсказки, однако многие программные пакеты используют другие клавиши). Окно подсказки содержит текст, который объясняет некоторую часть программы. Когда пользователи работают с программой и переходят от возможности к возможности, содержимое и расположение окон подсказки изменяются для отражения текущего контекста. Эти изменения не видны, поскольку они происходят внутри программ. Когда пользователь нажимает функциональную клавишу подсказки, соответствующее сообщение-подсказка выдается в выскакивающем окне.

Поскольку сообщение-подсказка зависит от текущего положения в программе, окно подсказки называется контекстно-управляемым. Опытные пользователи могут игнорировать возможности подсказки программы; новички могут нажимать функциональную клавишу подсказки при каждом изменении состояния программы и получать

советы, напоминания или детальные инструкции.

Разработчик системы решает, как много и какого вида подсказки система будет предусматривать для пользователя. Некоторые системы обладают несколькими уровнями подсказки в зависимости от опыта пользователя. Программа обработки слов Word Star применяла эту технологию на протяжении многих лет. Уровни подсказки изменяются от простых сообщений типа "нажмите <Ключ< для возврата к..." до полных пользовательских руководств. Многие разработчики программ предпочитают передавать руководство пользователя таким способом, а не в виде больших громоздких документов. Эта процедура имеет два последствия: пользователи связывают экстравагантные руководства с качеством программ, а большие книги способствуют борьбе с "программными пиратами."

Как бы там ни было, пользователи также пришли к ожиданию систем, требующих минимального использования пользовательских руководств. Пользователи хотят интерактивную подсказку.

Программирование окон подсказки

Функции подсказки в этой книге поддерживают концепции контекстно-управляемой подсказки с помощью использования оконных функций, текстового файла и функциональных вызовов из прикладной программы. Каждая прикладная программа сообщает функциям подсказки, какой файл подсказки использовать и какое окно подсказки является текущим. Файл подсказки содержит текст для каждого окна подсказки. Функции подсказки отслеживают сигналы от клавиатуры и выдают текущее окно подсказки при нажатии функциональной клавиши <Help.<

Программа, использующая программное обеспечение подсказки, должна предусматривать следующие интерфейсы с функциями подсказки:

- функциональный вызов для задания имени текстового файла подсказки;
- значение функциональной клавиши подсказки;
- функциональные вызовы для идентификации текущего окна подсказки;
- использование функции клавиатурного ввода get_char для любого ввода с клавиатуры (get_char описана в Главе 4.(

Последнее из этих четырех требований может потребовать некоторых разъяснений. Во время работы программы программное обеспечение подсказки перехватывает каждое прерывание от клавиатуры для того, чтобы убедиться, не нажата ли функциональная клавиша подсказки. Если это так, то управление передается оконному процессу подсказки. Если нет, значение клавиши пересылается в ожидающую его программную функцию. Программное обеспечение подсказки может управлять этим перехватом только в том случае, если вы используете функцию get_char для ввода с клавиатуры. Как разъяснено выше, get-char следит за нажатием на клавиатуре функциональной клавиши подсказки. Если вы используете другие способы чтения символов с клавиатуры, то переключение функциональной клавишей подсказки не будет сделано.

Имеются и другие методы наблюдения за клавиатурой с ожиданием нажатия функциональной клавиши подсказки. Некоторые из

этих методов привлекают присоединяющее перехватывающее программное обеспечение к вектору клавиатурного прерывания или вектору клавиатурной BIOS. По различным причинам было решено, что эти методы не будут использоваться. Во-первых, программное обеспечение, описываемое в данной книге, предполагает пользовательскую среду, включающую окна для меню, ввода данных и текста. Весь клавиатурный ввод, необходимый программе, может управляться одной из этих возможностей, причем все они используют функцию get_char. Вам никогда не понадобится снова программировать клавиатурный ввод. Во-вторых, если программа присоединяет себя к вектору прерывания, то при ненормальном завершении программы происходят странные вещи; обычно ПЭВМ приходится перезапускать. В-третьих, разрабатываемые вами программы могут быть резидентными в памяти. Резидентные в памяти программы часто присоединяются к клавиатуре для других целей.

Вы узнаете больше о резидентных в памяти программах, векторах прерываний и о том, как присоединять одно окно к другому, в Главе 11. Допустим, что ваше программное обеспечение читает клавиатуру с помощью функции get_char. Тогда вектора клавиатуры остаются одни, и система хорошо себя ведет независимо от резидентных программ, аварийных завершений программ при тестировании или неожиданных завершений программы в производственной системе.

Требуемое использование get_char лишает вас трех стандартных возможностей Си:

- вы не сможете использовать стандартные библиотечные функции scanf или getchar, поскольку ни одна из этих функций не использует функцию get_char;

- вы не сможете воспользоваться стандартной функцией ввода с консоли (getch), предусмотренной в Турбо Си;

- вы не сможете использовать логическое устройство stdin для клавиатурного ввода, что означает для вашей программы невозможность назначать файлы или программные каналы вместо клавиатуры.

Потеря функций scanf и getchar - это небольшая потеря для программирования диалога. Эти функции бесполезны в интерактивной среде. Они являются функциями буферизованного ввода, требующими нажатия клавиши <Ввод> для завершения ввода символа или строки; кроме того, они нечувствительны к расположению курсора и длине поля. Эти функции дублируют свой ввод в стандартный вывод, соблюдая соглашения командной строки ДОС, и они будут дублировать изображения двойных символов при вводе управляющего символа. Если введена клавишная комбинация Упр/С, функции могут аварийно завершить программу. Функции плохо работают с функциональными или курсорными клавишами. Турбо Си включает их для поддержки совместимости с UNIX. Эти функции происходят из компьютерных систем с телетайпными терминалами.

Может показаться, что вы теряете ряд возможностей, когда лишаетесь устройства stdin, однако это устройство предназначено для программ, которые могут принимать входные данные из файлов и выходов других программ так же, как и с клавиатуры. Эти программы)или, по крайней мере, stdin используется для ввода данных в них(обычно не предназначены для использования в интерактивной среде.

Текстовый файл окна подсказки

Окна подсказки появляются, когда пользователь нажимает функциональную клавишу подсказки, описанную в текстовом файле. Текстовый файл подсказки - это ASCII-файл, который вы строите текстовым редактором Турбо Си или любым другим редактором, который может создавать ASCII-файлы.

Окна подсказки описываются мнемоническим идентификатором, который используется программным обеспечением подсказки для идентификации и расположения текста. Строки, следующие за идентификатором, содержат текст, количество строк и длина строки которого определяют высоту и ширину окна подсказки. На рисунке 7.1показан небольшой файл подсказки, описывающий два окна подсказки.

```
¬------------------------------------------------------------
¦                                                            ¦
>                    ¦name¦                                  <
                     ¦Enter the name of the¦
                     ¦employee as last name¦                  ,
                     ¦comma, first name, viddle¦
                     ¦initial. Example¦                       :
                     ¦Hart, William S¦
>                    ¦emplnumb¦                              <
                     ¦Enter the employee number¦
                     ¦with from I to life digits¦            .
                     ¦Example: 12345¦
>                    ¦end¦                                   <
¦                                                            ¦
L------------------------------------------------------------
```

Рис. 7.1. Пример файла подсказки

Идентификатор окна подсказки состоит ровно из восьми символов и окружается угловыми скобками, как показано на рисунке .7.1Каждый идентификатор появляется в своей собственной строке, последней строкой файла должен быть лексический вход. На рисунке 7.2приведены два окна подсказки, которые будут отображены в результате этих описаний.

```
¬------------------------------------------------------------
¦                                                            ¦
¦                          ¬-----------------------------    ¦
 ¦      ¦Enter the name of the¦                          ¦
 ¦      ¦employee as last name¦                          ¦     ,
 ¦      ¦comma, first name, middle¦                          ¦ ,
 ¦      ¦initial. Example¦                          ¦          :
 ¦      ¦Hart, William S¦                          ¦
]  ¦      ¦Help] to return¦                          ¦
     ¦L¦                          ------------------------------
¦                                                            ¦
¦          ¬------------------------------                   ¦
 ¦                     ¦Enter the employee number¦          ¦
 ¦                     ¦with from 1 to five digits¦          ¦ .
 ¦                     ¦Example: 12345¦          ¦
]  ¦                     ¦Help] to return¦          ¦
                    ¦L¦          ------------------------------
L------------------------------------------------------------
```

Рис. 7.2. Примеры окон подсказки

Все примеры программ, которые приводятся в этой и последующих главах, используют программное обеспечение подсказки. Далее рассматривается простой пример для иллюстрации самой подсказки, а затем - использование подсказки по прямому назначению - для предоставления пользователю интерактивной контекстно-управляемой подсказки. С этой целью один и тот же файл подсказки предусматривается для всех программ. Он называется tcprogs.hlp и показан на листинге 7.1.

Листинг 7.1: tcprogs.hlp

```
>maxims<
 Press 1, 2, or 3
 for a pithy maxim.
>menu<
 Use arrow keys to move the cursor bars.
 Use Enter to make the selection.
>poemmenu<
 Arrows move the cursor bar.
 Enter will select the poem
 under the cursor bar.
 Press a digit (1-5) to
 select any other poem.
>poems<
 Move poem with arrow keys
 Select poem with 1 - 5
 Use + to bring poem forward
 Use - to push poem back
>name<
 Enter the name of the person
 who is placing the order.
>address<
 Enter the address of the
 person who is placing the
 order.
>state<
 The State may be one of these:
 VA, NC, SC, GA, FL
>phome<
 Enter the phone number
 of the person who is
 placing the order
>amount<
 Enter the amount
 of the order
>notepad<
--------Cursor Movement----- --------Page Movement--------
arrows  = move text cursor   Ctrl-Home = Beginning of File
Ctrl-T  = Top of Window      Ctrl-End  = End of File
Ctrl-B  = Bottom of Window   PgUp      = Previous Page
Ctrl -> = Next Word          PgDn      = Next Page
Ctrl <- = Previous Word
Home    = Beginning of Line --------Editor Control-------
End     = End of Line       Scroll Lock = No Auto Reform

--------Block Controls----- ---------Edit Commands--------
F4  = Form Paragraph        F2 or Esc = Done
F5  = Mark Block Beginning  F3        = Erase File
F6  = Mark Block End        Ins       = Toggle Insert Mode
```

```
F7  = Move Block            Del       = Delete Char
F8  = Copy Block            <--       = Rubout
F9  = Delete Block          Ctrl-D    = Delete Word
F10 = Unmark Block          Alt-D     = Delete Line
>end<
```

Функции подсказки

Для введения контекстно-управляемых окон подсказки в ваши программы необходимо к ним добавить два вызова функций, которые находятся в исходном файле thelp.c, представленном ниже в листинге 7.2.

```
void load_help(char *filename(
```

Вызовите эту функцию для загрузки файла подсказки или перехода к другому файлу подсказки. Функция открывает файл подсказки и анализирует в нем сообщения подсказки, строя таблицу идентификаторов окон подсказки, их размеров и расположений в файле подсказки.

```
void set_help(char *helpname, int x, int y(
```

Эта функция задает текущее окно подсказки с восьмисимвольным массивом, соответствующим наименованию окна в файле подсказки. Наименование окружается в файле подсказки угловыми скобками, причем не включает в себя эти скобки. Целые числа x и y задают координаты верхнего левого угла (в символьных позициях) окна подсказки, что позволяет пользователям использовать одно и то же окно в различных контекстах, но в различных листах экрана.

Изменение функциональной клавиши подсказки

Если вам нужно задать значение функциональной клавиши, отличной от <F1> (по умолчанию), вы должны изменить значение глобальной целочисленной переменной helpkey. Эта переменная объявлена в ibmpc.c в Главе 4. Вы можете включить исходный файл keys.h (см. Главу 4) в вашу программу и использовать одно из определенных в этом файле значений клавиш. Следующий фрагмент изменяет функциональную клавишу подсказки на <F2.<

```
#include "keys.h"
extern int helpkey;
helpkey = F2;
```

Изменение функции подсказки

Файл с именем ibmpc.c в Главе 4 включает указатель на функцию с именем helpfunc. Обычно этот указатель содержит значение NULL. Когда используются функции подсказки, указатель инициализируется адресом функции с именем help. Если вам нужно использовать другую функцию, скажем, вместо стандартной функции help, вы можете изменить значение указателя helpfunc. Для применения функциональной клавиши подсказки для вызова вашей

функции используйте следующие операторы:

```
extern void (*helpfunc;()(
void yourfunc;()
helpfunc = yourfunc;
```

Вы можете вернуться к указанию helpfunc на стандартную функцию подсказки с помощью следующих операторов:

```
extern void (*helpfunc;()(
extern void help;()
helpfunc = help;
```

Выключение подсказки

Имеются три пути для выключения подсказки: вы можете установить значение функциональной клавиши подсказки равным 0; можно установить указатель функции подсказки равным NULL; можно вызвать set_help, передавая указатель на строку нулевой длины. Для включения подсказки необходимо отменить выбранное действие.

Исходный листинг: thelp.c

Теперь перейдем к листингу 7.2 thelp.c. Этот файл является исходным текстом функций ,поддерживающих контекстно-управляемые окна подсказки.

Листинг 7.2: thelp.c

```
----------------------*/thelp.c/*------------------

#include <stdio.h<
#include <string.h<
#include <stdlib.h<
#include "twindow.h"
#include "keys.h"

#define MAXHELPS 25
#define HBG WHITE
#define HFG BLACK
#define HINT DIM

#define TRUE 1
#define FALSE 0

static struct helps}
    char hname [9;[
    int h, w;
    long hptr;
 {hps [MAXHELPS+1;[

static int hp = 0;
static int ch = 0;
static int hx ,hy;
FILE *helpfp = NULL;
```

```
long ftell;()
char *fgets;()
void help;()
char helpname[64;[
void getline(char *lineh;(

--------------*/загружает файл определения HELP/*-------------!
void load_help(char *hn(
}
    extern void (*helpfunc;()(
    extern int helpkey;
    char lineh [80;[

    if (strcmp(helpname, hn) == 0(
        return;
    helpfunc = help;
    helpkey = F1;
    hp = 0;
    strcpy(helpname, hn;(
    if ((helpfp = fopen(helpname, "r")) == NULL(
        return;
    getline)lineh;(
    while (1}   (
        if (hp == MAXHELPS(
            break;
        if (strncmp(lineh, "<end>", 5) == 0(
            break;
        if (*lineh('>' =!
            continue;
        hps[hp].h = 3;
        hps[hp].w =18;
        strncpy(hps[hp].hname, lineh+1, 8;(
        hps[hp].hptr = ftell(helpfp;(
        getline(lineh;(
        while (*lineh}   ('>' =!
            hps[hp].h;++
            hps[hp].w = max(hps[hp].w, strlen(lineh)+2;(
            getline(lineh;(
{
        hp;++
{
{

--------*/получает строку текста из файла подсказки/*---------
static void getline(char *lineh(
}
    if (fgets(lineh, 80, helpfp) == NULL(
        strcpy(lineh, "<end;("<
{
-----*/устанавливает текущий активный экран подсказки/*--------
void set_help(char *s, int x, int y(
}
    for (ch = 0; ch < hp; ch(++
        if (strncmp(s, hps[ch].hname, 8) == 0(
            break;
    hx = x;
    hy = y;
{

-------------*/выдает текущее окно подсказки/*-----------------
void help()
```

```
}
    char ln [80;[
    int i, xx, yy;
    WINDOW *wnd;
    extern int helpkey;
    if (hp && ch != hp} (
        curr_cursor(&xx, &yy;(
        cursor(0, 25;(
        wnd = establish_window(hx, hy, hps[ch].h, hps[ch].w;(
        set_colors(wnd, ALL, HBG, HFG, HINT;(
        display_window(wnd;(
        fseek(helpfp, hps[ch].hptr, 0;(
        for (i = 0; i < hps[ch].h-3; i}   (++
            getline(ln;(
            wprintf(wnd, ln;(
{
        wprintf(wnd, "[Help] to return;("
        while (get_char() != helpkey(
            putchar(BELL;(
        delete_window(wnd;(
        cursor(xx, yy;(
{
{
```

Описание программы: thelp.c

Программа thelp.c содержит четыре имени #define, устанавливающих глобальные параметры системы подсказки. MAXHELPS получает значение максимального количества окон подсказки, которое программа может поддерживать одновременно. HBG, HFG и HINT являются цветами фона, переднего плана и яркостью окон подсказки.

Структура helps описывает окно подсказки. Она содержит мнемоническое имя окна, его высоту и ширину, а также символьное смещение описания окна в текстовом файле, включающем окна подсказки. Массив hps структур helps содержит один элемент для каждого окна в текстовом файле и строится функцией load_help. Функция load_help читает текстовый файл подсказки, заданный при вызове, и строит массив hps. Она распознает угловую скобку, идентифицирующую каждое имя окна, копируя имя и символьное смещение в файле в структуру. Затем она читает текст окна для определения длины и ширины окна. Длина яляется функцией количества строк текста, а ширина является функцией размера самой длинной строки текста окна.

Функция set_help просматривает массив для поиска окна с именем, заданным при вызове. Целочисленная переменная ch, которая используется для индексации в массиве при поиске, будет содержать индекс текущего окна при нажатии функциональной клавиши подсказки. Переменные hx и hy получают значения при вызове. Эта процедура устанавливает позицию окна.

Функция help вызывается функцией get_char из ibmpc.c при нажатии заданной функциональной клавиши подсказки. Функция help сохраняет текущую позицию курсора и убирает курсор с экрана. Создается окно подсказки с координатами и размерами, записанными в элементе массива с индексом ch. Текущая символьная позиция в текстовом файле перемещается к месту, указанному в элементе

массива. Каждая строка текста читается из файла и записывается в окно. Дописывается последняя строка, сообщающая пользователю о необходимости повторного нажатия клавиши подсказки для очистки окна подсказки и возврата. Программа ожидает нажатия клавиши подсказки, уничтожает окно и восстанавливает курсор на экране в прежнем положении.

Пример контекстно-управляемой подсказки

Программы на листингах 7.3, 7.4 и 7.5 представляют пример использования программного обеспечения подсказки. Листинг 7.3, sayings.c, содержит главную функцию, вызывающую функцию примера, maxims.c, показанную в листинге 7.4. Листинг 7.5 является проектным make-файлом, используемым Турбо Си для построения этого примера.

Для запуска программы вводите следующую команду:

```
c>sayings
```

Программа sayings.c загружает файл teprogs.hlp с помощью вызова load_help и вызывает функцию с именем maxims. Такая последовательность была выбрана для того, чтобы maxims.c могла быть встроена в последующие программы, предусматривающие примеры обработки меню и резидентных в памяти утилит. Maxims.c показывает, как использовать set_help и get_char в ваших программах. Используется только одно окно подсказки. Maxims.c открывает окно и ожидает нажатия клавиши. Если вы нажимаете 1, 2 или 3, в окне показывается одна из трех старых пословиц. На рисунке 7.3 изображено окно с одной из таких выданных пословиц. Если вы нажмете <Ключ>, программа завершится. Если будет нажата >F1>, выдается окно подсказки, как показано на рисунке 7.4.

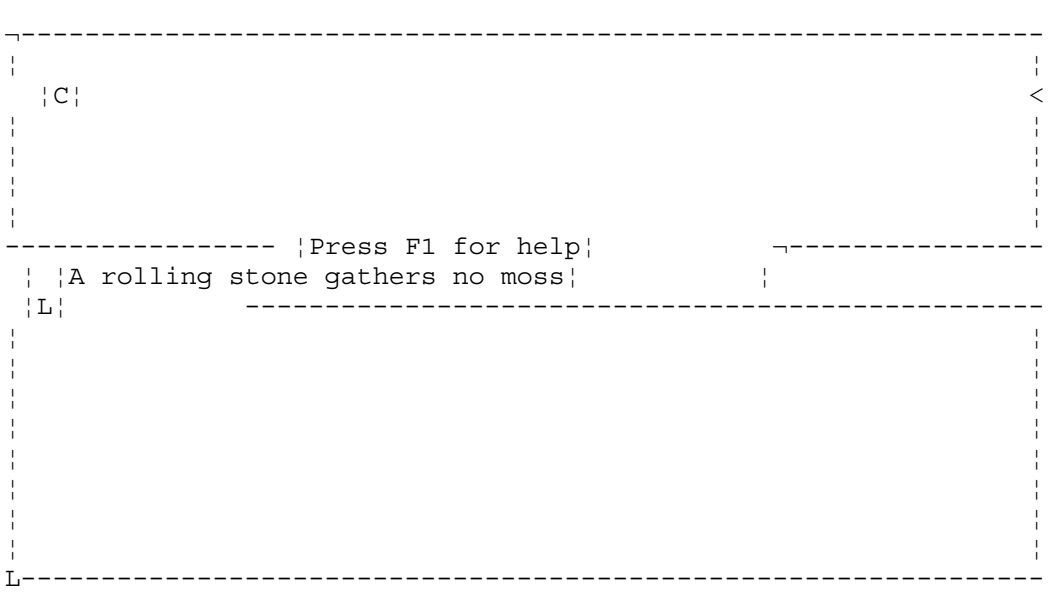
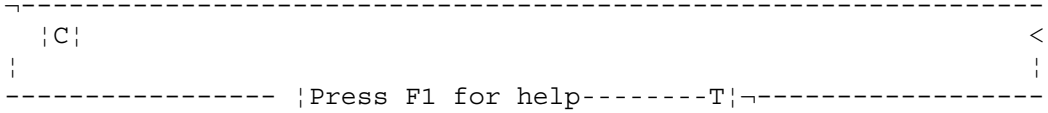


Рис. 7.3. Выданная пословица



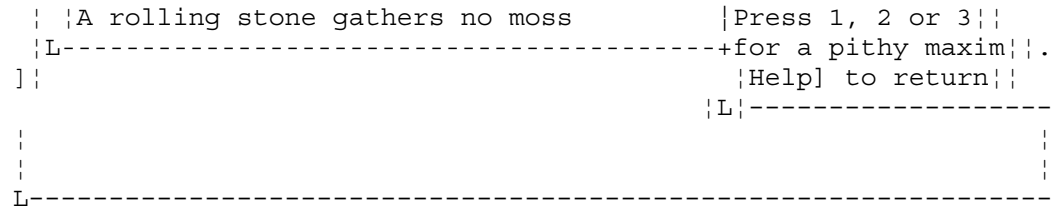


Рис. 7.4. Подсказка для пословиц

Листинг 7.3: sayings.c

```
--------------*/sayings.c/*--------------
#include "twindow.h"
void maxims(void;(

main()
}
   load_help("tcprogs.hlp;("
   maxims;()
{
```

Листинг 7.4: maxims.c

```
--------------------*/maxims.c/*-----------------
#include "twindow.h"
#include "keys.h"

void maxims()
}
   int c;
   WINDOW *wnd;

   set_help("maxims ", 50, 10;(
   wnd = establish_window(5, 10, 3, 50;(
   set_title(wnd, "Press F1 for help;("
   set_colors(wnd, ALL, RED, WHITE, DIM;(
   display_window(wnd;(
   while ((c = get_char()) != ESC} (
       switch (c}      (
           case '1:'
               wprintf(wnd, "\nA stitch in time\
saves nine;("
               break;
           case '2:'
               wprintf(wnd, "\nA rolling stone\
gathers no moss;("
               break;
           case '3:'
               wprintf(wnd, "\nA penny saved\
is a penny earned;("
               break;
           default:
               break;
{
{
   delete_window(wnd;(
{
```

Листинг 7.5: sayings.prj

```
saying
```

```
maxims (twindow.h, keys.h(
thelp (twindow.h, keys.h(
twindow (twindow.h, keys.h(
ibmpc.obj
```

Резюме

Ваши оконные средства теперь включают возможность контекстно-управляемой подсказки. Следующие главы содержат дополнительные возможности обработки окон и примеры их использования. Каждый пример использует окна контекстно-управляемой подсказки, поэтому ваше понимание средств подсказки возрастает с каждой добавленной возможностью. Это обучение имеет существенное значение, поскольку, как правило, диалоговые программы должны предусматривать оперативную подсказку пользователю. Следующее дополнение к использованию окон в диалоговой программе представлено в Главе 8, которая описывает использование окон в качестве форм для ввода данных. Глава 8 включает функции, позволяющие вам описывать содержимое и формат шаблона ввода данных, и описывает программное обеспечение для сбора элементов данных, вводимых пользователем в форму.

ГЛАВА 8

Использование данных в окнах

Интерактивные программы интерпретируют значения данных внутри компьютера в соответствии с типами данных, определенными пользователем. Эти значения данных могут интерпретироваться по-разному, поэтому программы могут использовать разнообразные технические приемы для интерпретации значений вводимых данных. Например, текстовый процессор интерпретирует данные, поступающие на его вход, как текст произвольной формы. Электронный бланк интерпретирует данные в соответствии с совокупностью форматов строки и столбца, в которой в данный момент находится курсор. Однако одни и те же данные каждая программа "видит и чувствует" по-разному. Поэтому даже правильное копирование входного потока одной программы на вход другой может привести к массе спорных результатов, которые в этом случае выдает вторая программа.

Многие интерактивные программы воспринимают значения данных непосредственно в той форме, в которой пользователь осуществляет их физический ввод (например, данные в печатной форме (бланк.((Каждая, сходная с печатной, форма представления данных определялась Государственной налоговой службой. Формо-ориентированные интерактивные программы интерпретируют программы как лист бумаги, содержащий поля со значениями данных и текст, описывающий значение каждого поля. После того, как пользователь введет значение данного в соответствующие позиции поля, программное обеспечение проверяет допустимость введенного значения и дальнейшее управление курсором. В этой главе формо-ориентированный ввод данных, указание доступа к ним и контроль операций над ними реализован путем использования окна как шаблона для ввода данных. Шаблон ввода данных включает в себя окно вместе с описанием вводимых в каждое поле данных и именами

полей ввода данных.

Шаблон ввода данных

Перед вводом данных в окно вы должны прежде всего открыть окно, задать атрибуты шаблона ввода данных, соответствующие описанию данных в прикладной программе, обрабатывающей их, а также описать расположение полей ввода данных и присвоить им имена. После того, как шаблон готов, вы можете вызвать функцию ввода данных, которая осуществит управление чтением всех элементов данных с шаблона. По завершению работы функции, когда все значения элементов данных размещаются в указанном вами буфере, производится проверка на их допустимость, преобразование по указанным форматам и считывание в некоторой последовательности для обработки вашей прикладной программой.

Поле ввода данных

Поле ввода данных, по сути дела, представляет собой элемент данных. Оно может быть датой, результатом вычисления суммы чего-либо, именем или вычисленной средней заработной платой. Поле ввода данных имеет следующие характеристики: оно имеет определенную длину и формат и должно занимать одну строку окна. Поле ввода данных - традиционный элемент систем управления базами данных (СУБД.(

После того, как вы описали поля шаблона, вы должны специфицировать позицию размещения поля в шаблоне, атрибуты, выходной буфер, функцию проверки допустимости данных, help-информацию и маску вводимых данных для каждого поля. Эти компоненты описания полей рассматриваются ниже.

Позиция

Позиция поля задается выражением, состоящим из номера строки и столбца символьных координат, соответствующих местоположению текста в шаблоне окна. Предпочтительнее задавать координаты позиции поля относительно координат окна, а не экрана; если вы вдруг решите изменить координаты расположения самого окна, то в случае описания позиции поля в координатах окна описание позиции поля вам изменять не придется.

Атрибуты

Атрибут поля позволяет описать вводимые в него данные как имеющие один из существующих типов данных. Тип данных задается отдельной буквой (которые рассмотрены в данной главе). Однако уже сейчас вам необходимо знать, что вы можете специфицировать поле как поле даты, спецификации, денежной единицы или как числовое. Задание атрибута поля позволяет контролировать, каким способом будут интерпретированы значения данных в поле.

Буфер

Для каждого поля назначается выходной буфер данных, который является адресом символьного массива, в котором будет происходить накопление данных. Ваша программа резервирует память под этот буфер и передает его адрес данным внутри прикладной программы.

Проверка допустимости значений

Ваша программа генерирует адрес функции проверки допустимости вводимых значений так, как будто сама эта функция используется. Программное обеспечение ввода данных будет вызывать эту функцию всегда после того, как вы ввели данное в поле. Следует заметить, что при вводе данных программное обеспечение само осуществляет автоматическую проверку некоторых основных ограничений в соответствии с атрибутами поля, однако, используя дополнительные подпрограммы проверки значений, вы можете существенно расширить и усложнить проверку данных по различным критериям.

Help-информация

Вы можете специфицировать вспомогательное help-окно (которое описано в Главе 7) для каждого поля, а также help-функцию для каждого поля, которая будет вам выдавать справочную информацию о предназначении каждого поля в случае, если постоянный текст, описывающий поле и отображаемый в окне, не удовлетворяет вашему любопытству. Help-окно получает информацию из специального help-файла, в связи с чем включение help-функций в программу обязательно. При желании вы можете не специфицировать свои help-функции, а использовать стандартные help-функции пакета, которые могут быть вызваны пользователем путем нажатия на соответствующую help-клавишу. Следует помнить, что использование соответствующей help-спецификации эффективно лишь в процессе ввода пользователем данных в поле, для которого эта help-информация предназначена.

Маска вводимых данных

Когда вы определяете поле для ввода данных, вы можете задать маску для всех вводимых в это поле значений. Эта маска специфицируется в массиве символов, включающем в себя символы нижнего подчеркивания и пунктуации. Символ нижнего подчеркивания соответствует позиции маски, в которой возможен ввод данных, а пунктуация служит для обозначения других (различных) отображаемых символов кода ASCII. Длина элементов данных описывается количеством символов нижнего подчеркивания, а длина буфера, выделяемого под вводимые данные, описываемые маской, должна быть не меньше, чем длина элемента данных плюс 1. Символы пунктуации в буфер не перекачиваются. Например, маска вводимых данных, касающихся номера телефона (включая код местности и расширение(может иметь следующий вид:

```
char phone_mask [] = "(____)____-____ ext;"_____:
```

Приглашения к вводу в поле (Prompts(

Каждое поле может иметь свое специфичное приглашение к вводу в него данных, которое обычно содержит информацию, поясняющую пользователю, для чего предназначено данное поле (семантика поля(и какая именно информация должна быть в него введена. Такое приглашение представляет собой символьную строку, которую пользователь изменить не может. Вы должны сами специфицировать содержание этой строки и позицию ее размещения в окне. Приглашение к вводу данных должно иметь длину, не превышающую длину строки окна.

Ввод данных

В процессе своей работы функция ввода данных обрабатывает поля шаблона в той последовательности, в которой вы осуществляли их описание (не принимая во внимание позиции их размещения в шаблоне), осуществляя при этом простейшую проверку допустимости находящихся в полях шаблона данных (правильность даты, поле должно содержать только цифры и т.д.). Вы можете также определить обычную (в смысле Си) функцию, которая будет осуществлять дальнейшую, возможно, сложную и разнообразную, проверку вводимых данных или определить функцию, которая будет выдавать в случае ввода недопустимых данных специальную help-информацию пользователю. В процессе ввода данных в шаблон используется выделение в соответствии с указанными атрибутами (ACCENT) цвета окна более ярким тоном полей, в которые предполагается ввод данных. При этом курсор на экране компьютера переходит в соответствующую позицию поля. В случае, если пользователь инициирует другой порядок ввода данных, переходя от поля к полю, то выделение следующего, выбранного пользователем поля ввода, и перемещение курсора также осуществляется программно.

Функции сбора данных

Эти функции, являющиеся библиотечными функциями, ваша программа так или иначе вызывает при использовании окна с шаблоном ввода данных. Запомните, что вы должны вначале установить окно, а лишь затем использовать эти функции.

```
void init_template(WINDOW *wnd(
```

Эта функция инициализирует окно для использования в нем шаблона ввода данных. Она устанавливает связной список структур FIELD и производит очистку всех существующих дескрипторов. Структура FIELD, описывающая характеристики полей ввода данных, определена в twindow.h (см. Главу 6). Функция inint_template предназначена для выполнения двух задач. Во-первых, она инициализирует окно; во-вторых, если окно инициализировано, она осуществляет поиск и уничтожение всех установленных на данный момент полей окна; следовательно, использование этой функции возможно в ситуации, когда шаблон окна не является достаточно большим, так как использование функции приводит к освобождению структур FIELD.

FIELD *establish_field
)WINDOW *wnd,int x,int y,char *msk,char *bf,int t(

Эта функция устанавливает поля ввода данных в окне, которое было инициализировано функцией init_template как окно с шаблоном ввода данных. Символы x и y суть координаты, специфицирующие размещение маски поля ввода, причем предпочтительнее указывать координаты относительно окна, а не экрана. Указатель bf суть указатель на определяемый при обращении буфер сбора данных. Параметр t (значение типа integer) указывает тип поля ввода данных. Различают следующие типы полей:

A = алфавитно-цифровое
N = цифровое, незначащие разряды заполнены пробелами
Z = цифровое, незначащие разряды заполнены нулями
C = поле денежной единицы
D = поле даты

Указатель msk является указателем на символ маски, который определяет длину поля и управляет отображением символа пунктуации на экране (причем последние не контролируются в буфер сбора данных поля). Маска включает в себя обычно несколько символов. Символ подчеркивания указывает позицию символа, который должен быть введен в данном месте. Массив bf должен содержать столько символов, сколько символов подчеркивания содержит маска, плюс один символ (для символа конца строки \0.(

Поле денежной единицы может иметь произвольное число цифр слева от десятичной точки и две цифры справа от нее.

Дата должна соответствовать формату "ММДДГГ". Если пользователь неправильно ввел дату, программа выдает сообщение об ошибке и требует от пользователя повторного ввода даты.

Функция establish_field возвращает указатель на структуру FIELD, определенную в twindow.h. Вы можете использовать этот указатель при обращении к field_window, field_help, field_protect и field_validate, которые описаны ниже.

void wprompt(WINDOW *wnd, int x, int y, char *s(

Эта функция осуществляет выдачу приглашения на ввод данных в окно. Приглашение на ввод данных является строкой s, которая выдается, начиная с координат окна, задаваемых значениями x и y.

void field_tally(WINDOW *wnd(

При обращении к этой функции на экран дисплея выводятся значения всех данных, хранящихся в буфере, для всех полей шаблона. Вы можете использовать эту функцию, когда, например, вы заполнили буфер значениями из записи базы данных.

void field_window(FIELD *fld, char *helpname, int x, int y(

Эта функция обеспечивает выдачу контекстно-зависимого help-окна для каждого поля шаблона. Параметр helpname является строкой, содержащей мнемонику help-окна в текущем help-файле, и специфицируется предварительно путем обращения к функции load_help (см. Главу 7). Точно так же, как функция data_entry

осуществляет переход от поля к полю, обращение к функции set_help для каждого из полей позволяет привязать конкретное help-окно для каждого из них. Параметры x и y позволяют специфицировать позицию экрана, с которой будет осуществляться выдача help-окна.

Заметьте, что эта функция, а также еще три, рассмотренные ниже, не требуют наличия в списке параметров указателя WINDOW. Для этих функций достаточно указателей FIELD, так как эти указатели представляют собой указатели на цепочку связного списка, который и осуществляет привязку функции к конкретному окну. Эти функции модифицируют определенные ранее поля, не обращая внимания на то, что они принадлежат конкретному шаблону окна.

void clear_template(WINDOW *wnd(

Эта функция освобождает буферы сбора данных всех полей шаблона, обращая их в пустую строку, заканчивающуюся нулевым символом конца строки, а также отображает на экране все поля шаблона.

void field_validate(FIELD *fld, int (*validfn(()(

Эта функция относится к макросам. Она использует адрес функции проверки допустимости вводимых значений (если таковая будет написана пользователем). Стандартные функции, осуществляющие проверку допустимости вводимых в поля данных, обычно не осуществляют требуемой проверки на ошибочную ситуацию. По этой причине пользователь обычно сам пишет функцию, производящую контроль данных. Функция data_entry будет осуществлять обращение к определенным вами функциям контроля данных после того, как ею самой будет осуществлен их первичный контроль. Если это так, то она будет передавать в эти ваши функции адрес буфера сбора данных, и разработанные вами функции контроля данных смогут осуществить проверку введенных пользователем значений, находящихся уже в буфере. Ваша функция контроля допустимости данных может содержать вызов функции error_message (см. Главу 6) для обработки ситуации, когда будет обнаружено недопустимое значение. При этом определенная вами функция контроля данных должна возвращать лишь два значения OK или ERROR, которые, в свою очередь, определены в twindow.h. Если функция возвращает значение OK, то функция data_entry будет обрабатывать следующее поле шаблона. Если же функция контроля возвращает значение ERROR, то функция data_entry "остановится" на поле, в данных которого обнаружена ошибка.

void field_protect(FIELD *fld, int prot(

Эта функция также относится к макросам. Она устанавливает или отменяет (в зависимости от значения параметра prot) защиту характеристик поля. Защищенное поле не обрабатывается)игнорируется) функцией data_entry. Используя функцию field_protect, вы можете управлять доступом пользователей к полям шаблона, разрешая или запрещая запись в них данных.

Эта функция может использоваться совместно с функциями field _validate, clear_template и field_fally для контроля над изменениями, производимыми в записях базы данных, в случае, если элементы записи базы данных отображаются в шаблоне.

Для понимания того, как значительно расширяет возможности

программы использование функции field_protect, рассмотрим шаблон, изображенный на рисунке 8.1. Примите во внимание, что этот шаблон содержит поля, образующие запись по одному служащему в файле базы данных. Шаблон ввода данных используется для ввода, отображения, поиска и изменения записей в файле .Перед началом ввода данных ваша программа может обратиться к функции field_protect для защиты всех полей, кроме поля "номер служащего". Затем вы можете обратиться к функции field_validate, передав ей указатель на обычные, разработанные вами функции контроля данных на допустимость. После того, как пользователь введет данные в поле "номер служащего", функция data_entry (будет описана позже(осуществит обращение к разработанной вами функции контроля данных, которая осуществит поиск и сравнение по ключу (номер служащего) нужной записи в базе данных. Функция контроля данных произведет загрузку соответствующих буферов накопления данных элементами данных из найденной записи базы данных, а затем обратится к функции field_fally для вывода значений элементов данных на экран. После этого будет вызвана функция field_protect для установки защиты для поля "номер служащего" и снятия защиты с остальных полей. В итоге функция контроля данных передаст управление функции data_entry, и пользователь получит возможность произвести коррекцию значений элементов данных. По завершении пользователем процесса обработки записи система передаст управление прикладным функциям, которые непосредственно будут обращаться в data_entry. Эти функции могут перезаписать откорректированную запись в файл, очистить буфера данных и шаблоны путем обращения к функции clear_template, переопределить защиту полей шаблона (а значит ,и записи), сняв защиту с поля "номер служащего" и установив защиту для других полей, и повторить весь процесс обработки записи заново.

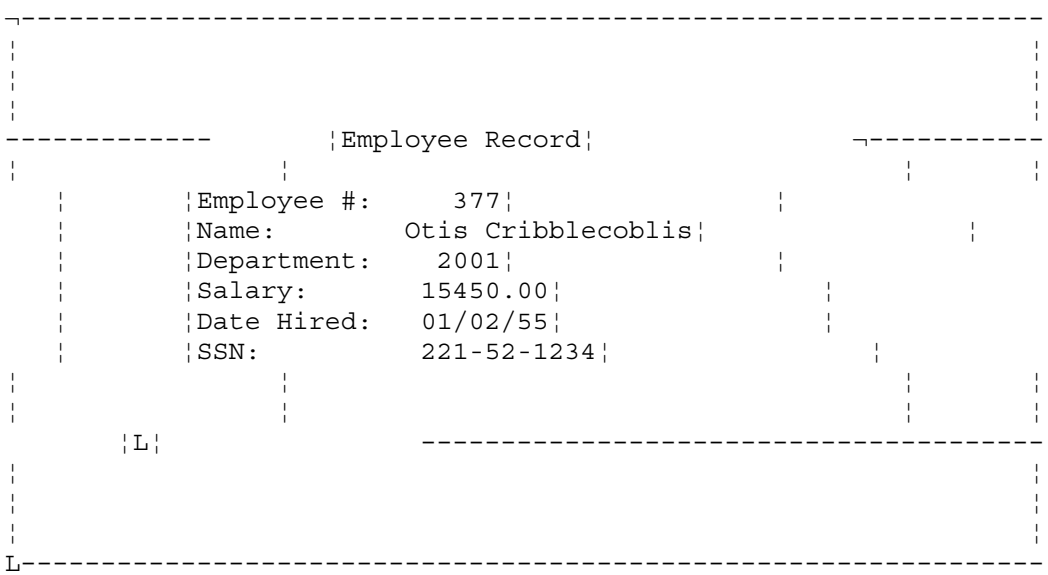


Рис. 8.1. Пример шаблона ввода данных

void field_help(FIELD *fld, int (*helpfn(()(

Эта функция относится к макросам. Она позволяет вам установить специальную help-функцию, которая будет вызываться вместо стандартной help-функции. Эта функция используется для отдельных полей шаблона, когда пользователю требуется получить по ним более полную help-информацию, чем ему предоставлено текстом,

отображаемым в окне.

В процессе ввода данных в окно у пользователя могут возникнуть различные вопросы относительно как предназначения окна, так и вида входных данных. Для получения ответов на свои вопросы пользователю достаточно нажать на специально выделенную help-клавишу, которая инициирует обращение к специальной help-функции. Функция data_entry передаст адрес буфера накопления данных поля, что впоследствии даст возможность специальной help-функции вернуть текущее значение в поле.

Эта отличительная черта функции field_help используется, когда имеется список возможных значений, из которых пользователь должен осуществлять выбор, но отображение всего списка в окне постоянно нецелесообразно. Вы можете открыть еще одно окно для выдачи help-информации с помощью вашей специальной help-функции, отобразить в нем весь список возможных значений и затем использовать функцию get_selection (см. Главу 6) для обеспечения выбора пользователем нужного ему значения. Вы также можете, как это делается в примере, который представлен в этой главе ниже, записывать текущую дату в буферы накопления данных полей. Вам предоставлена возможность осуществлять запросы к базе данных. Очевидно, что при работе с записями файла "Служащий" базы данных ваша help-функция может выдавать список номеров служащих и их фамилии, выбирая эту информацию непосредственно из самого файла "Служащий."

int data_entry(WINDOW *wnd(

Эта функция обрабатывает вводимые в шаблон данные. Пользователь может просмотреть весь шаблон со значениями всех полей, выдавая на дисплей текущие значения данных из соответствующих буферов накопления данных. Пользователь осуществляет ввод данных в поля в той последовательности, в которой поля были установлены в шаблоне с помощью функции establish_field, при этом позиция размещения поля в шаблоне во внимание не принимается.

Поле, которое предназначено в данный момент для ввода в него данных, отображается пользователю с учетом суммарного пространства под вводимые данные, включая пунктуацию, и выделяется повышенной яркостью в соответствии со значением параметра ACCENT, определяющим цвет окна. Курсор устанавливается в первой позиции поля, и функция переходит в режим ожидания ввода данных. Пользователь может начать ввод данных или в текущее поле, или установить текущим другое поле, используя клавишу <ВВОД,< клавишу табуляции <Таб> или клавиши управления курсором. Клавиши >КЛЮЧ>, <Страница вверх> (<PgUp>), <Страница вниз> (<PgDn,(< >КОН> (<End>), перемещения курсора в первую позицию первой строки >)Home>) и функциональные клавиши (с <ПФ1> до <ПФ10>, исключая клавишу, за которой закреплен вызов help-функции) предназначены для завершения процесса ввода данных и возврата из функции data_entry, к которой перед этим было осуществлено обращение, в точку вызова.

После ввода пользователем последнего символа поля происходит выделение повышенной яркостью следующего поля и перемещение к нему курсора. Перед этим функция data_entry обращается к своим собственным функциям контроля данных и (если вы установили еще и свои специальные функции контроля данных) к специальным функциям контроля. Все эти функции должны в результате работы выдать

значение ОК, в противном случае функция data_entry не переместит курсор к следующему полю.

Если пользователь нажал клавишу, за которой закреплен вызов help-функции, то будет осуществлен вызов специальной help-функции)если она вами определена), предназначенной для текущего поля, путем передачи ей адреса буфера накопления данных поля. Если help-функция вами не создана и help-окно "привязано" к полю, то help-функции, описанные в Главе 7, обеспечат выдачу на экран дисплея help-окна. Если ни help-окно, ни help-функция не определены, то нажатие пользователем help-клавиши не приведет ни к какому результату, если, конечно, программа использует help-функции независимо от контекста вводимых в окно данных. После того, как происходит возврат из функции data_entry в точку ее вызова, все буферы полей содержат значения данных, которые ввел пользователь. Эта функция возвращает управление в точку вызова при нажатии клавиш, означающих конец ввода данных в шаблон. Таковыми являются клавиша <КЛЮЧ>, клавиши листания страниц (<PgUp>, <PgDn>) или функциональные клавиши, приводящие к завершению процесса ввода данных. Прикладное программное обеспечение анализирует значение последней нажатой пользователем клавиши для определения его намерений. Нажатие клавиши <КЛЮЧ,< например, может означать "Игнорируй все внесенные мной изменения и верни меня к предыдущему содержанию шаблона". Клавиши <Страница вверх> (<PgUp>) и <Страница вниз> (<PgDn>) означают (как определено в keys.h), что пользователь хочет перейти к следующей или предыдущей записи базы данных. Когда пользователь завершает ввод данных, то код завершения процесса ввода передается в программу, вызвавшую функцию data_entry, для дальнейшего использования.

Исходный текст: entry.c

Листинг 8.1 представляет собой исходный текст файла entry.c, который содержит библиотечные функции поддержки оконного шаблона ввода данных.

Листинг 8.1: entry.c

```
----------------*/entry.c/*----------------

#include <stdio.h<
#include <ctype.h<
#include <stdlib.h<
#include <alloc.h<
#include <mem.h<
#include <string.h<
#include "twindow.h"
#include "keys.h"

#define FIELDCHAR'_'
int insert_mode = FALSE;      /* режим вставки, ВКЛ/ВЫКЛ/*
extern int helpkey;


----------------*/локальные прототипы/*----------------
void addfield(WINDOW *wnd, FIELD *fld;(
void disp_field(WINDOW *wnd, char *bf, char *msk;(
```

```
void data_value(WINDOW *wnd, FIELD *fld;(
void insert_status(void;(
int read_field(WINDOW *wnd, FIELD *fld;(
void right_justify(char *s;(
void right_justify_zero_fill(char *s;(
int validate_date(char *s;(
int endstroke(int c;(
int spaces(char *c;(

----------------*/инициализация шаблона/*---------------
void init_template(WINDOW *wnd(
}
    FIELD *fld, *fl;

    fld = FHEAD;
    while (fld} (
        fl = fld->fnxt;
        free(fld;(
        fld = fl;
{
    FHEAD = NULL;
{
---------------*/установка полей шаблона/*--------------
FIELD *establish_field(wnd, cl, rw, msk, bf, ty(
WINDOW *wnd;
int rw;
int cl;
char *msk;
char *bf;
int ty;
}
    FIELD *fld;

    if ( (fld = malloc(sizeof(FIELD))) == NULL(
        return NULL;
    fld->fmask = msk;
    fld->frow = rw;
    fld->fcol = cl;
    fld->fbuff = bf;
    fld->ftype = ty;
    fld->fprot = 0;
    fld->fnxt = fld->fprv = NULL;
    fld->fvalid  =NULL;
    fld->fhelp = NULL;
    fld->fhwin = NULL;
    fld->flx = fld->fly = 0;
    addfield(wnd, fld;(
    return fld;
{

--------*/добавление поля в конец списка/*-----------
static void addfield(WINDOW *wnd, FIELD *fld(
}
    if (FTAIL}  (
        fld->fprv = FTAIL;
        FTAIL->fnxt = fld;
{
    FTAIL = fld;
    if (!FHEAD(
        FHEAD = fld;
{
```

```
---------*/отображение данных в полях/*----------
static void disp_field(WINDOW *wnd, char *bf, char *msk(
}
    while (*msk}    (
        wputchar(wnd, *msk != FIELDCHAR ? *msk : *bf;(++
        msk;++
{
{

--------*/отображение значений данных в полях/*-----------
static void data_value(WINDOW *wnd, FIELD *fld(
}
    wcursor(wnd, fld->fcol, fld->frow;(
    disp_field(wnd, fld->fbuff, fld->fmask;(
{

----------*/отображение всех полей в окне/*---------------
void field_tally(WINDOW *wnd(
}
    FIELD *fld;

    fld = FHEAD;
    while (fld != NULL} (
        data_value(wnd, fld;(
        fld = fld->fnxt;
{
{

-----------*/установка help-окон для полей/*--------------
void field_window(FIELD *fld, char *hwin, int x, int y(
}
    fld->fhwin=hwin;
    fld->flx = x;
    fld->fly = y;
{

-----------*/очистка всех полей шаблона/*--------------
void clear_template(WINDOW *wnd(
}
    FIELD *fld;
    char *bf, *msk;

    fld = FHEAD;
    while (fld != NULL} (
        bf = fld->fbuff;
        msk = fld->fmask;
        while (*msk}    (
            if (*msk == FIELDCHAR(
*               bf;' ' = ++
            msk;++
{
        fld = fld->fnxt;
{
    field_tally(wnd;(
{

---------*/установка режима вставки/замены курсора/*------------
static void insert_status()
}
    set_cursor_type(insert_mode ? 0x0106 : 0x0607;(
```

```
{

------------*/чтение поля с клавиатуры/*--------------
static int read_field(WINDOW *wnd, FIELD *fld(
}
    char *mask = fld->fmask, *buff = fld->fbuff;
    int done = FALSE, c, column;

    column = fld->fcol;
    while (*mask != FIELDCHAR}  (
        column;++
        mask;++
{
    while (TRUE}    (
        wcursor(wnd, column, fld->frow;(
        c = get_char;()
        if (fld->ftype == 'A('
            c = toupper(c;(
        clear_message;()
        switch (c}  (
            case '\b:'
            case BS:
                if (buff == fld->fbuff} (
                     done = c == BS;
                     break;
{
--                buff;
                do}
--                   mask;
--                   column;
 {                while (*mask != FIELDCHAR;(
                if (c == BS(
                    break;
            case DEL:
                movmem(buff+1, buff, strlen(buff;((
)*                buff+strlen(buff;' ' = ((
                wcursor(wnd, column, fld->frow;(
                disp_field(wnd, buff, mask;(
                break;

            case FWD:
                do}
                    column;++
                    mask;++
 {                while (*mask && *mask != FIELDCHAR;(
                buff;++
                break;
            case INS:
                insert_mode ^= TRUE;
                insert_status;()
                break;
            case:'.'
                if (fld->ftype == 'C}   ('
                    if (*mask++ && *buff}     (' ' ==
*                        buff++ = '0;'
                       if (*mask++ && *buff(' ' ==
*                           buff++ = '0;'
{
                      right_justify(fld->fbuff;(
                      wcursor(wnd, fld->fcol, fld->frow;(
                      disp_field(wnd, fld->fbuff, fld->fmask;(
```

```
                        column = fld->fcol+strlen(fld->fmask)-2;
                        mask = fld->fmask+strlen(fld->fmask)-2;
                        buff = fld->fbuff+strlen(fld->fbuff)-2;
                        break;
{
                default:
                    if (endstroke(c}   ((
                        done = TRUE;
                        break;
{
                    if (toupper(fld->ftype)!='A'&&!isdigit(c}   ((
                        error_message("Numbers only;("
                        break;
{
                    if (insert_mode}    (
                        movmem(buff, buff+1, strlen(buff)-1;(
                        disp_field(wnd ,buff, mask;(
                        wcursor(wnd, column, fld->frow;(
{
*                    buff++ = c;
                    wputchar(wnd, c;(
                    do}
                        column;++
                        mask;++
 {                    while (*mask && *mask != FIELDCHAR;(
                    if (! *mask(
                        c = FWD;
                    break;
{
        if (!*mask(
            done = TRUE;
        if (done}   (
             if (fld<-ftype == 'D&& '
                     c != ESC&&
                         validate_date(fld->fbuff) !=OK(
                 return ERROR;
             break;
{
{
    if (c != ESC && toupper(fld->ftype) != 'A}  ('
        if (fld->ftype == 'C}  ('
            if (*mask++ && *buff}    (' ' ==
*                buff++ = '0;'
                if (*mask++ && *buff(' ' ==
*                    buff++ = '0;'
{
{
       if (fld->ftype == 'Z' || fld->ftype == 'D('
           right_justify_zero_fill(fld->fbuff;(
       else
           right_justify(fld->fbuff;(
       wcursor(wnd, fld->fcol, fld->frow;(
       disp_field(wnd, fld->fbuff, fld->fmask;(
{
    return c;
{

-------*/проверка значения с на код клавиши завершения/*------
static int endstroke(int c(
}
    switch (c}  (
```

```
        case '\r:'
        case '\n:'
        case '\t:'
        case ESC:
        case F1:
        case F2:
        case F3:
        case F4:
        case F5:
        case F6:
        case F7:
        case F8:
        case F9:
        case F10:
        case PGUP:
        case PGDN:
        case HOME:
        case END:
        case UP:
        case DN:
            return TRUE;
        default:
            return FALSE;
{
{

------------*/выравнивание вправо, заполнение пробелами/*------
static void right_justify(char *s(
}
   int len;

   len = strlen(s;(
   while (*s == ' ' || *s == '0' && len}  (
       len;--
*       s;' ' = ++
{
   if (len(
       while (*(s+(len-1} (' ' == ((
           movmem (s, s+1, len-1;(
*           s;' ' =
{
{

-----------*/выравнивание вправо, заполнение нулями/*----------
static void right_justify_zero_fill(char *s(
}
    int len;

    if (spaces(s((
        return;
    len = strlen(s;(
    while (*(s + len - 1}   (' ' == (
        movmem(s, s + 1, len-1;(
*        s = '0;'
{
{

---------------*/контроль пробелов/*-------------
int spaces(char *c(
}
   while (*c(' ' ==
```

```
       c;++

   return !*c;
{

----------------*/проверка даты/*---------------------
static int validate_date(char *s(
}
    static int days= []
;{ 31,28,31,30,31,30,31,31,30,31,30,31 }
    char date [7;[
    int mo;

    strcpy(date, s;(
    if (spaces(date((
        return OK;
    days[1] = (atoi(date+4)%4) ? 28 : 29;
)*    date + 4) = '\0;'
    mo = atoi(date+2;(
)*    date+2) = '\0;'
    if (mo && mo<13 && atoi(date) && atoi(date)<=days[mo-1([
        return OK;
    error_message("Invalid date;("
    return ERROR;
{

---------*/Процесс ввода данных в шаблон экрана/*-----------
int data_entry(WINDOW *wnd(
}
    FIELD *fld;
    int exitcode, isvalid, done=FALSE, oldhelpkey=helpkey;
    field_tally(wnd;(
    fld = FHEAD;
--*/    накопление данных, поступающих с клавиатуры на экране/*--
    while (fld != NULL && done == FALSE}    (
        set_help(fld->fhwin, fld->flx, fld->fly;(
        helpkey = (fld->fhelp) ? 0 : oldhelpkey;
        wcursor(wnd, fld->fcol, fld->frow;(
        if (fld->fprot ==FALSE}     (
            reverse_video(wnd;(
            data_value(wnd, fld;(
            wcursor(wnd, fld->fcol, fld->frow;(
            exitcode = read_field(wnd, fld;(
            isvalid = (exitcode != ESC && fld->fvalid? (
)*)                        fld->fvalid))(fld->fbuff) : OK;
{

          else}
              exitcode = FWD;
              isvalid = OK;
{
          if (isvalid == OK}   (
              normal_vileo(wnd;(
              data_value(wnd, fld;(
              switch (exitcode)   {     /* передано редактору/*
                   case F1:    if (fld->fhelp} (
)*)                                  fld->fhelp))(fld->fbuff;(
                                  data_value(wnd, fld;(
{
                              break;
                  case DN:
```

```
                    case '\r:'
                    case '\t:'
                    case FWD:   fld = fld->fnxt;
                                if (fld == NULL(
                                    fld = FHEAD;
                                break;
                    case UP:
                    case BS:    fld = fld->fprv;
                                if (fld == NULL(
                                    fld = FTAIL;
                                break;
                    default:    done = endstroke(exitcode;(
                                break;
{
{
{
     helpkey = oldhelpkey;
     return (exitcode;(
{
 --------*/отображение приглашения к вводу/*---------
void wprompt(WINDOW *wnd, int x, int y, char *s(
}
   wcursor(wnd, x, y;(
   wprintf(wnd, s;(
{
```

Описание программы: entry.c

Макроопределение FIELDCHAR программы entry.c идентифицирует специальные символы, используемые в символьной маске поля при определении байтов данных. По сути дела, идентифицированные с помощью FIELDCHAR символы не являются символами в смысле данных, так как используются в маске. В качестве значения FIELDCHAR выступает символ нижнего подчеркивания. Вы можете изменить определение FIELDCHAR, используя другие символы.

Привязка полей к окнам осуществляется с помощью двойного связного списка. Список начинается и заканчивается в структуре WINDOW для поля. Каждое поле представлено посредством структуры FIELD, которая размещается в памяти при установке поля.

Функция init_template используется для инициализации окна с шаблоном ввода данных. Она выполняет трассировку связного списка FIELD и освобождает некоторые размещенные ранее в памяти структуры FIELD.

Функция establish_field размещает в памяти и инициализирует буфер FIELD, используя передаваемые ей при обращении значения или принимая значения передаваемых переменных по умолчанию. Маска, позиция размещения поля, адрес буфера и тип данных поля передаются функции при обращении к ней. Очередная структура FIELD добавляется в конец соответствующего связного списка FIELD, специфицированного WINDOW. Указатель на очередную структуру FIELD возвращается в точку вызова функции.

Функция addfield вызывается для добавления структуры FIELD к связному списку, специфицированному WINDOW.

Функция disp_field используется для вывода значений данных, введенных в поля, из буфера на экран дисплея. Маска данных,

передаваемая при обращении к функции, используется при отображении содержимого поля. Обращение к функции содержит WINDOW, указатель буфера поля и указатель маски. Подразумевается, что курсор в окне всегда устанавливается на первую позицию первого поля окна. Символы, хранящиеся в буфере, отображаются на экране вместе с символами пунктуации, составляющими маску поля. Функция data_value предназначается для отображения текущего значения поля в окне. Функция устанавливает курсор в окне в позицию первого символа поля и вызывает функцию disp_field, передавая ей адрес WINDOW и адреса буфера поля и маски.

Функция field_fally используется для выдачи на экран значений всех полей шаблона. Она осуществляет просмотр "сверху-вниз" связного списка FIELD и осуществляет обращение к функции data_value для каждого поля шаблона.

Функция field_window предназначена для установки оконной help-информации для поля. Указанные при обращении к функции имя help-информации и координаты экрана для ее выдачи копируются в специфицированную при вызове функции структуру FIELD.

Функция clear_template обрабатывает "сверху-вниз" связной список FIELD для специфицированного окна. Функция преобразует буфер каждого поля шаблона в пустую строку, заканчивающуюся нулевым символом, используя маску поля для определения установленной при описании поля длины. Когда все поля обработаны таким образом (почищены), функция обращается к field_fally.

Программа ввода данных работает в режимах вставки и замены символов. Клавиша <ВСТ> (<Ins>) используется для переключения режимов работы программы, а переменная insert_mode служит для индикации текущего режима работы. Функция insert_status изменяет тип курсора в соответствии со значением переменной insert_mode. Режим вставки обуславливает наличие квадратного курсора, а режим замены символов - наличие курсора в виде нижнего подчеркивания. Для изменения формы курсора используется функция set_cursor_type из библиотеки ibmpc.c (см. Главу 4.(

Функция read_field используется для чтения введенных пользователем в поля данных. Она устанавливает курсор в начало поля и считывает последовательность символов в нем. Два локальных указателя используются для трассировки вводимых данных. Указатель mask позволяет получить текущую позицию в маске, а указатель buff -текущую позицию в буфере. При перемещении пользователем курсора, уничтожении или вводе символов значение этих указателей корректируется.

При выполнении операции по уничтожению символов используется функция Турбо Си movmem, которая используется для сдвига символов в буфере, а также функция disp_field для отображения результатов операции на экране.

Клавиша <ВСТ> (<Ins>) переключает режимы Вставки/Замены.

Если текущее поле включает в себя точку (.), то оно содержит нули в двух крайних правых позициях (если другие значения в них не указаны), выравнено вправо, а соответствующие указатели и курсор устанавливаются на значащие позиции, предназначенные для указания пенни (денежной единицы.(

Если пользователь нажал клавишу, символизирующую завершение

ввода данных, то данная ситуация обрабатывается функцией endstroke, и ввод данных завершается. В противном случае символ будет записан в буфер. Если в данный момент обрабатывалось не алфавитно-цифровое поле, то анализируется ситуация на предмет ввода пользователем числа. Если было введено не число, то выдается сообщение об ошибке, и программа не воспринимает введенные данные, требуя их повторного ввода. Если включен режим вставки, то данные в буфере смещаются на одну позицию вправо, а значение поля после вставки символа отображается с помощью функции disp_field. При этом символ записывается в буфер и отображается на экране. Совместное использование указателей маски и буфера позволяет "перескакивать" через символы пунктуации. После ввода последнего символа в поле дальнейший ввод данных в него прекращается.

После завершения ввода данные проверяются на допустимость, даты и числа выравниваются, курсор устанавливается в соответствующую позицию поля и содержимое полей отображается на экране.

Функция endstroke анализирует значение клавиш и устанавливает клавиши, которые могут привести к завершению процесса ввода данных.

Есть две функции, занимающиеся выравниванием данных. Первая -right_justify, выравнивающая справа и заполняющая поле пробелами. Вторая - right_justify_zero_fill, выравнивающая справа и заполняющая поле нулями.

Функция spaces анализирует поле на наличие всех пробелов.

Функция validate_date осуществляет проверку даты.

Функция data_entry вызывается для обработки всех полей шаблона. Функция вызывает field_fally для выдачи на экран всех значений данных из соответствующих буферов полей. Затем функция обрабатывает "сверху-вниз" связный список структур FIELD и управляет вводом данных в каждом поле.

Функция set_help вызывается для осуществления привязки оконной help-информации к конкретному полю. Поскольку значение глобальной переменной helpkey определено заранее, то попытка получить для этого поля специальную help-информацию закончится безуспешно, поскольку программа data_entry будет перехватывать прерывание от клавиши help-функции.

Если поле не является защищенным, то функция reverse_video вызывается каждый раз, когда необходимо получить выделенное отображение поля в соответствии с конфигурацией цветов, определенной параметром ACCENT. Для выдачи на экран значения текущего поля в акцентированном режиме, определяемом параметром ACCENT, используется функция data_value, а для ввода в программу введенных в поле пользователем данных применяется функция read_field. Значением, возвращаемым функцией read_field, является итоговая последовательность символов, суть которой - введенные в поле данные. Если поле имеет свою специальную функцию контроля данных, то вызывается эта функция.

При условии, что контроль данных по всем параметрам прошел нормально, функция normal_video переводит окно в так называемый контрольный режим отображения данных, цвет которого определяется

значением NORMAL, и функция data_value повторно выводит на экран значение поля, но уже в NORMAL-цвете. Затем анализируется завершающая последовательность нажатия клавиш. Если ввод данных прерван клавишей, закрепленной за вызовом специальной help-функции, то осуществляется обращение к этой функции. В связи с тем, что в процессе обработки полей реализована возможность заменять буферизованные данные полей полностью, функция data_value снова вызывается для повторного отображения поля после его обработки.

Завершающей клавишей в процессе ввода данных могут быть либо клавиши движения вперед (стрелка вниз, ВВОД, Таб), либо клавиши движения назад (стрелка вверх, стрелка влево). Следующее поле для обработки выбирается из связного списка FIELD, специфицированного параметром WINDOW, в зависимости от значений этих клавиш (вперед или назад). При этом выбирается следующее или предыдущее поле, и ввод данных продолжается.

Если последней была нажата одна из тех клавиш, которые анализируются функцией endstroke и не являются аналогом ранее встречающихся и анализируемых функцией последовательностей, то ввод данных в оконный шаблон завершается, и функция data_entry возвращает управление в точку вызова.

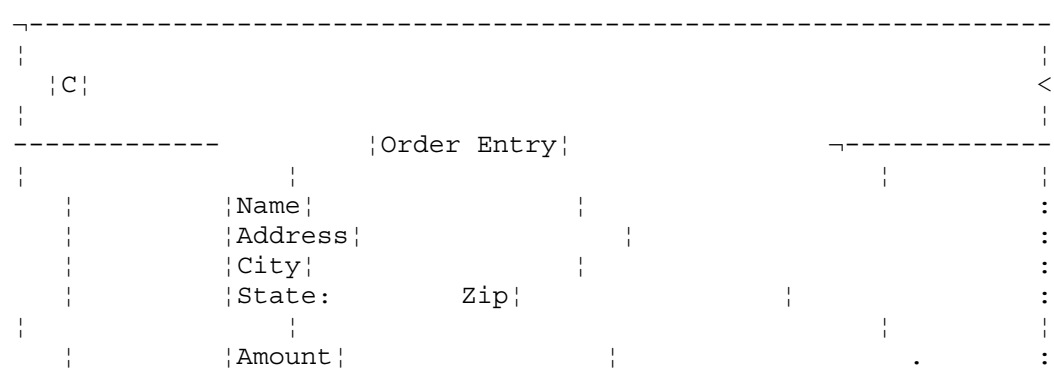
Пример: Ввод данных в определенном порядке

Исходные файлы, представленные на листингах 8.2, 8.3 и 8.4, суть примеры использования шаблона ввода данных в окне. Листинг ,8.2order.c, представляет собой текст главной функции (main,(которая осуществляет обращение к функции ordent.c, непосредственно реализующей пример. Функция ordent.c будет в дальнейшем присутствовать при рассмотрении примера меню в Главе 10и резидентной утилиты в Главе 12. Листинг 8.4 представляет собой make-файл order.prj, используемый утилитой make Турбо Си для построения программы примера.

Для выполнения программы примера введите команду

```
C>order
```

Окно, представленное на рисунке 8.2, является по сути дела "всплывающим", то есть появляющимся в позиции курсора. Однако обратитесь к листингу 8.3, ordent.c, предварительно ознакомившись с описанием программы, приведенным ниже. После того, как вы ознакомились с листингом, выполните программу.



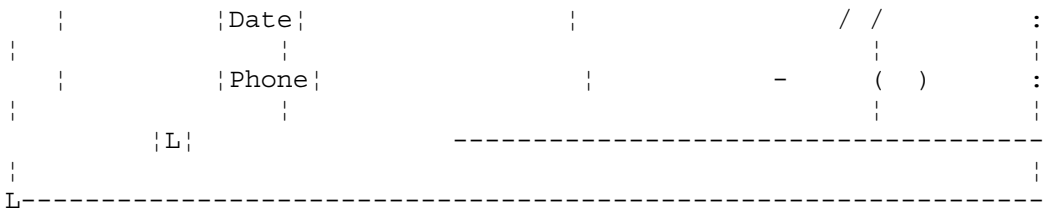


Рис.8.2. Образец ввода шаблона ввода данных

Пример программы начинается с установки окна и передачи ему значения заголовка и цвета. Окно отображается на экране дисплея, и в шаблоне появляется приглашение к вводу данных. После обращения к функции init_template осуществляется идентификация help-информации для каждого поля (help-информация размещается в пределах окна), которая поясняет назначение вводимых в каждое поле данных. Установка оконной help-информации осуществляется путем обращения к функции field_window. Каждое поле непосредственно устанавливается в шаблоне путем обращения к функции establish_field. Последнее поле устанавливается в этом примере как не использующее оконную help-информацию. Поэтому для данного поля с помощью функции field_help идентифицируется специальная help-функция. Обратите внимание, что одно поле шаблона - "state" - описано как имеющее специальную функцию контроля данных, вводимых в него, на достоверность. Эта специальная функция контроля данных специфицирована в обращении к функции field_validate.

Программа обращается к функции clear_template для получения пустого шаблона ввода данных, а затем - к функции data_entry, позволяющей обрабатывать данные, вводимые в шаблон. По завершению работы функции data_entry программа уничтожает окно. В реальной задаче в этой точке программы вы должны предусмотреть обработку введенных данных. В нашем примере после уничтожения окна программа завершает свою работу.

Во время выполнения программы обратите внимание, каким образом вводимые символы помещаются в позиции определенных полей, а также на управление курсором. Режим Вставки/Замены символов управляется с помощью клавиши <Ins> и индуцируется соответствующей формой курсора. Если курсор отображается как мерцающий символ подчеркивания в обычном своем виде, то это означает, что данные вводятся в режиме замены символов. После того, как вы нажмете клавишу <Ins>, курсор примет форму мерцающего прямоугольника, и вы будете продолжать работу в режиме

вставки символов.

Нажмите клавишу <F1> для получения справочной информации по каждому из полей шаблона. На рисунке 8.3 изображен шаблон с некоторыми значениями полей в момент выдачи по запросу пользователя оконной help-информации. Нажмите клавишу <F1> для поля "State" и получите оконную help-информацию для него. Обратите внимание, что в качестве help-информации выступает перечень допустимых значений поля "State". Выберите одно из них и введите в качестве значения поля. На примере этого поля внимательно изучите, как работает специальная функция контроля данных, как она использует функцию error_message для выдачи сообщений о вводе недопустимых данных. Рисунок 8.4 наглядно демонстрирует именно случай неправильного ввода данных в поле

"State". Специальная help-функция для поля даты ("Date(" демонстрирует, как можно получить значение даты извне функции data_entry. В данном примере для чтения текущей даты используется функция Турбо Си getdate, после чего текущая дата записывается в соответствующий буфер поля "Date."

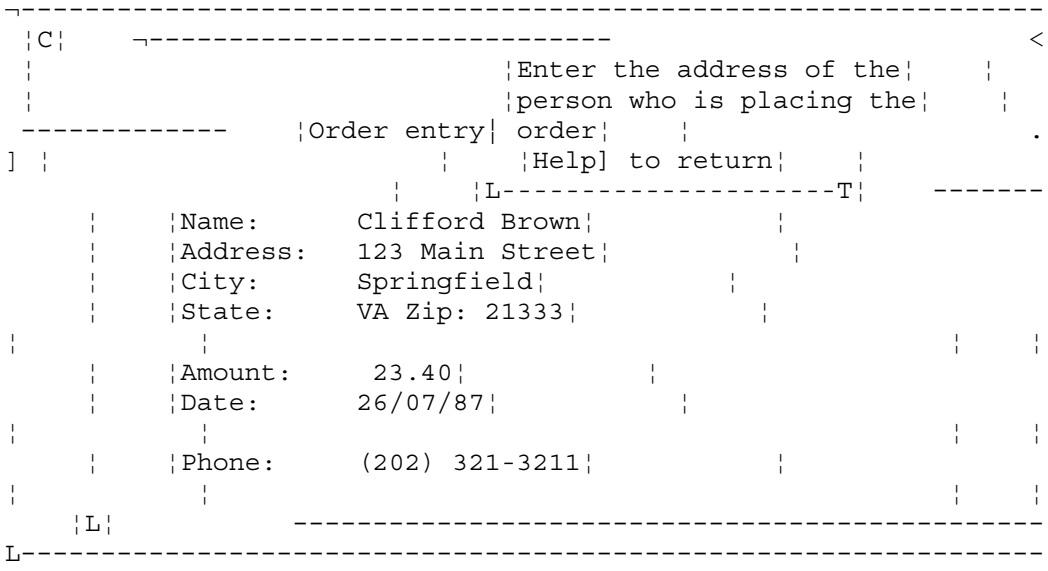


Рис. 8.3. Образец ввода шаблона ввода данных с данными и help-информацией

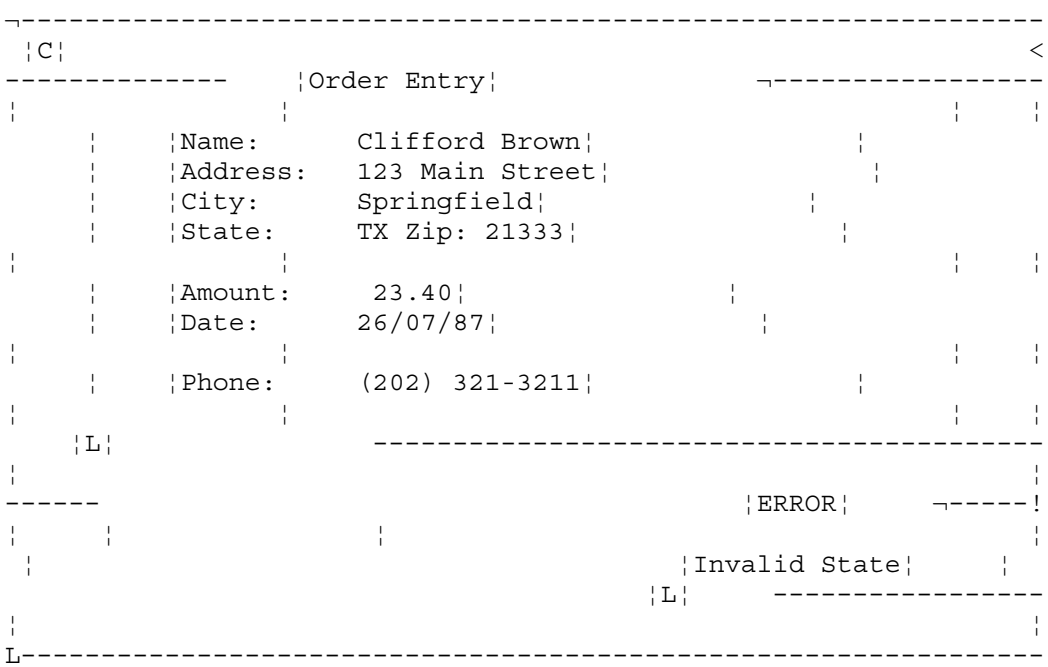


Рис. 8.4. Проверка корректности ввода данных

Листинг 8.2: order.c

```
----------*/order.c/*---------

#include "twindow.h"
void ordent(void;(

main()
```

```
}
   load_help("tcprogs.hlp;("
   ordent;()
{
```

Листинг 8.3: ordent.c

```
-----------------*/ordent.c/*----------------

#include <dos.h<
#include <stdio.h<
#include <string.h<
#include "twindow.h"

struct}
    char name [26;[
    char addr [26;[
    char city [26;[
    char state [3;[
    char zip [6;[
    char amt [6;[
    char dt [7;[
    char phone [11;[
 {rcd;

char msk25;"_____" =      []
char mskamt;"____.____" =   []
char mskdate;"/___/___/___" =  []
char mskphone;"____-____ (___)" = []
#define mskst msk25+23
#define mskzip msk25+20

int validate_state(char *, int;(
void help_date(char;(*

void ordent()
}
   WINDOW *wnd;
   FIELD *fld;

   wnd = establish_window(10, 5, 15, 50;(
   set_title(wnd, " Order Entry;("
   set_colors(wnd, ALL, BLUE, AQUA, BRIGHT;(
   set_colors(wnd, ACCENT, WHITE, BLACK, DIM;(
   display_window(wnd;(
   wprompt(wnd, 5, 2, "Name;(":
   wprompt(wnd, 5, 3, "Address;(":
   wprompt(wnd, 5, 4, "City;(":
   wprompt(wnd, 5, 5, "State;(":
   wprompt(wnd, 18, 5, "Zip;(":
   wprompt(wnd, 5, 10, "Phone;(":
   wprompt(wnd, 5, 7, "Amount;(":
   wprompt(wnd, 5, 8, "Date;(":

   init_template(wnd;(
   fld = establish_field(wnd, 15, 2, msk25, rcd.name, 'a;('
   field_window(fld,"name    ", 40, 1;(
   fld = establish_field(wnd, 15, 3, msk25, rcd.addr, 'a;('
   field_window(fld,"address ", 40, 2;(
   fld = establish_field(wnd, 15, 4, msk25, rcd.city, 'a;('
```

```
   field_window)fld, "address ", 40, 3;(
   fld = establish_field(wnd, 15, 5, mskst, rcd.state, 'A;('
   field_validate(fld, validate_state;(
   field_window(fld,"state   ", 40, 4;(
   fld = establish_field(wnd, 23, 5, mskzip, rcd.zip, 'Z;('
   field_window(fld,"address ", 40, 4;(
   fld = establish_field(wnd,15,10,mskphone,rcd.phone, 'N;('
   field_window(fld,"phone   ", 40, 9;(
   fld = establish_field(wnd, 15, 7, mskamt, rcd.amt, 'C;('
   field_window(fld,"amount  ", 40, 8;(
   fld = establish_field(wnd, 15, 8 ,mskdate, rcd.dt, 'D;('
   field_help(fld, help_date;(
   clear_template(wnd;(
   data_entry(wnd;(
   delete_window(wnd;(
{
--------*/проверка состояния нажатой клавиши/*----------
int validate_state(bf, key(
char *bf;
}
   static char *states= []
","  "}       VA","NC","SC","GA","FL",0;{
   char **st = states;

   while (*st(
       if (strcmp(*st++, bf) == 0(
           return OK;
   error_message("Недопустимое состояние;("
   return ERROR;
{


-------*/выдает сегодняшнюю дату/*------
void help_date(bf(
char *bf;
}
   struct date dat;

   getdate(&dat;(
   sprintf(bf, "%02d%02d%02d,"
           dat.da_day, dat.da_mon, dat.da_year % 100;(
{
```

Листинг 8.4: order.prj

```
order
ordent (twindow.h(
entry (twindow.h, keys.h(
thelp (twindow.h, keys.h(
twindow (twindow.h, keys.h(
ibmpc.obj
```

Резюме

Оконная библиотека содержит сейчас средства создания контекстно-зависимой help-информации и форматного ввода данных.

Эти средства вы можете с успехом использовать при создании систем интерактивного ввода данных. Шаблон ввода данных поддерживает фиксированные форматы вводимых данных. В Главе 9 в функции текстового редактора добавлены новые, одна из них - возможность использования окна для ввода и модификации произвольного текста.

ГЛАВА 9

Оконный текстовый редактор

В Главе 8 обсуждалось применение видеоокон для использования ввода данных в поля формо-ориентированного шаблона ввода данных. Естественно, что не все поля данных имеют заранее определенный, фиксированный формат. Многие поля и файлы состоят из текстов свободной, неопределенной формы и последовательности, например, слова в книге или комментарии к программе. Текстовые процессоры и редакторы текстов позволяют вам строить текстовые файлы, игнорируя формат исходных данных (текста), однако, если вы создаете действительно интегрированную систему , то доступ к функциям ввода текста в вашу программу и его редактирование может вам понадобиться непосредственно из вашей прикладной программы.

Эта глава является предварительным описанием оконного текстового редактора, который использует видеоокна для ввода и модификации текста точно так же, как большинство программ редактирования текста и текстовых процессоров. Эта отличительная черта оконного редактора используется при создании программных систем, в которых требуется ввод текста произвольной формы. Пакет Sidekick является примером системы, в которой имеется сходный с рассматриваемым в этой главе редактор текстов. Пример программы, представленный в этой главе, вполне может использоваться в качестве редактора текстов в системах типа Sidekick. Оконный текстовый редактор может также применяться в системах ввода данных, когда требуется ввести в качестве данных текст произвольной формы, занимающий более одной строки. Прикладная программа типа программы ввода данных в заданном порядке с помощью шаблона ввода данных из Главы 8 может потребовать ввода поля, содержащего описательный текст. Это требование можно легко реализовать, применяя оконный текстовый редактор. Многие базы данных оперируют с данными, характеризующимися наличием поясняющего их текста, различных описаний или определенной терминологии предметной области. При использовании шаблонов ввода данных поля данных такого типа могут с успехом обрабатываться оконным редактором. вы также можете использовать текстовый редактор для подготовки текстов help-окон в вашей программе путем обработки файла описаний help-окон, описанного в Главе 7.

Для использования текстового редактора программа вначале устанавливает окно, а затем резервирует буфер, в котором будут храниться и редактироваться данные, вводимые пользователем. Текстовый буфер редактора представляет собой массив, резервирующий память, достаточную для хранения фиксированного числа строк, каждая из которых равна ширине области окна, отводимой для ввода и редактирования текста. Например, если вы установили ширину окна в 42 символа, то область текста будет иметь ширину 40 символов (рамка окна занимает 2 позиции). Если вы в этом случае отвели под буфер 4000 символов (длина массива), то

редактор сможет обрабатывать одновременно только 100 строк текста, так как буфер может содержать не более 100 строк текста по 40 символов в строке. При вводе пользователем текста он накапливается в буфере без специальных символов новой строки (\n(и табуляции (\t.(

Если текстовый буфер уже содержит текст, который был занесен в него при первом обращении к редактору, то этот старый текст отображается на экране дисплея в виде строк фиксированной длины в соответствии с описанием окна.

Команды тестового редактора

Текстовый редактор содержит полный набор команд редактирования. Команды сведены в один перечень и могут быть отображены на экране дисплея в виде, представленном на рисунке .9.1В прошлой главе была рассмотрена программа, которая использовала help-окно для выдачи различной информации. Help-окно отображается при нажатии клавиши <F1>. Точно такой же прием используется для отображения набора команд редактора. Большинство из команд редактирования закреплено за функциональными клавишами, остальные используют комбинации клавиш <Ctrl> или <ALT> с другими клавишами. Ниже будут рассмотрены все команды оконного текстового редактора.

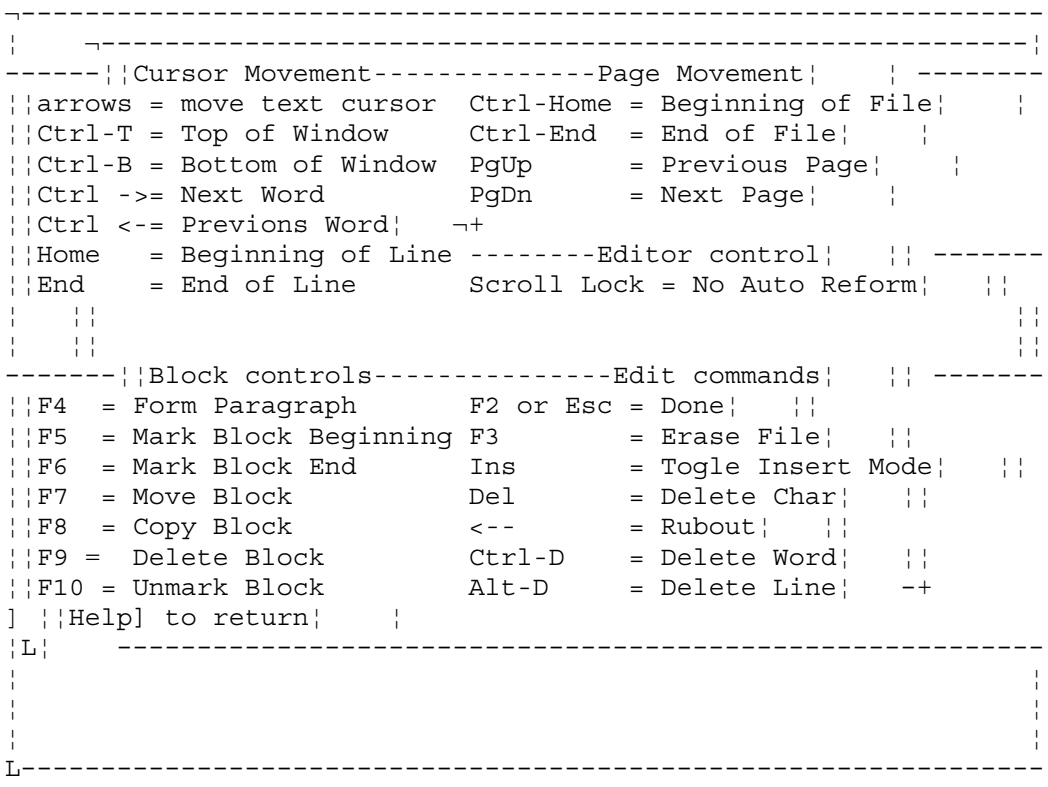


Рис. 9.1. Команды текстового редактора

Управление курсором

Используя клавиши управления курсором, его можно перемещать

по всему экрану. Если вы переместили курсор в начало или в конец экрана и продолжаете нажимать клавишу перемещения курсора вверх)или вниз), то тем самым вы инициируете скроллинг текста, который будет продолжаться до достижения начала (конца) буфера данных.

> - Ctrl/T> и <Ctrl/B> перемещают буфер в начало и конец экрана;

> - Ctrl/(курсор вправо)> перемещает курсор в начало следующего слова в буфере хранения текста;

> - Ctrl/(курсор влево)> перемещает курсор в начало предыдущего слова в буфере хранения текста;

> - курсор в начало экрана> позиционирует курсор в начало текущей строки;

> - КОН> позиционирует курсор в конец текущей строки;

> - ТАБ> перемещает курсор к следующей позиции табуляции;

> - РЕГИСТР/ТАБ> перемещает курсор к предыдущей позиции табуляции.

Постраничная работа

> - Страница вверх> и <Страница вниз> перемещает текст на страницу вперед или назад;

> - Ctrl/(курсор в начало экрана)> устанавливает текст на первую страницу, позиционируя курсор в первой позиции первой строки текста;

> - Ctrl/КОН> устанавливает текст на последнюю страницу в буфере и позиционирует курсор в первой позиции строки, которая следует за последней строкой текста в буфере.

Команды работы с блоками текста

Текстовый редактор имеет в своем составе команды работы с блоками текста. Эти команды позволяют оперировать с блоками текстов, которые определены ограничительными линиями.

> - F4> формирует параграф из текста, помеченного как блок;

> - F5> помечает первую строку блока. Строка текста, в которой находится курсор, становится первой строкой блока. Текст, образующий блок, отображается в цвете, определяемом значением параметра ACCENT для данного окна;

> - F6> помечает последнюю строку блока текста;

> - F7> перемещает весь блок текста к строке текста, в которой расположен в данный момент курсор. Перемещение блока происходит без его разрушения;

пространство для размещения блока выделяется автоматически;

> - F8> копирует блок к текущей строке текста. Отличие этой команды от предыдущей в том, что настоящая команда не приводит к разрушению блока-оригинала на его прежнем месторасположении в тексте.

> - F9> уничтожает блок. Строки текста, следующие за удаленным блоком, перемещаются вверх, заполняя освободившееся пространство в тексте;

> - F10> отменяет маркировку фрагмента текста как блока.

Команды редактирования

После того, как вы ввели текст и осуществили перемещение курсора, редактор автоматически переформатирует текущий параграф. Редактор определяет параграф как группу строк текста, начинающуюся с первой строки, состоящей из пробелов. Переформатирование параграфов может подавляться путем нажатия перечисленных ниже клавиш в режиме (Scroll Lock:(

> - F3> уничтожает весь текст, хранящийся в буфере "Команда." Команда требует подтверждения;

> - INS> переключает режимы работы редактора Вставка/Замена, а также изменяет форму курсора, индицирующего режим работы редактор;

> - DEL> уничтожает символ, находящийся после курсора в тексте, перемещая весь текст в строке на одну позицию влево;

\- Забой <Backspace> (длинная стрелка в правом верхнем углу клавиатуры) уничтожает символ, находящийся левее курсора, перемещая сам курсор и весь следующий за ним текст строки на одну позицию влево;

> - Ctrl/D> (<CTRL/D>) уничтожает слово, перед которым позицирован курсор;

> - ALT/D> уничтожает строку текста, в которой был предварительно позицирован курсор;

> - F2 <или <ESC> приводит к выходу из текстового редактора в точку его вызова из главной программы.

Функция, реализующая текстовый редактор

Текстовый редактор реализован в виде отдельной функции, которую может вызвать ваша программа. Для использования этой функции вы должны вначале организовать окно, в которое будут вводиться и в котором будут обрабатываться текстовые данные.

```
void text_editor (WINDOW *wnd, char *bf, unsigned bsize(
```

Эта функция обрабатывает текстовые данные, вводимые в окно с помощью текстового редактора. Указатель wnd специфицирует предварительно установленное вами окно. Указатель bf определяет текстовый буфер, а целое число bsize специфицирует размер буфера. Количество строк в буфере является функцией, зависящей от размера буфера и размеров (ширины) окна, которые устанавливаются, когда вы организуете окно.

При вызове функции она отображает текст, начиная с первой страницы из текстового буфера, и позволяет как ввод текстовых данных, так и ввод команд редактирования и обработки текста с клавиатуры. При выходе из текстового редактора буфер содержит введенный или модифицированный пользователем текст.

Исходный листинг: editor.c

Листинг 9.1 содержит исходный текст оконного текстового редактора.

Листинг 9.1: editor.c

```
 ------------------------ */editor.c/* ------------------------

#include <stdio.h<
#include <ctype.h<
#include <mem.h<
#include <conio.h<
#include <alloc.h<
#include "twindow.h"
#include "keys.h"

#define TRUE 1
#define FALSE 0
#define TAB 4
#define NEXTTAB (TAB-(x%TAB((
#define LASTTAB (((wwd-1)/TAB)*TAB(
#define PREVTAB (((x-1)%TAB)+1(
#define curr(x,y) (bfptr+(y)*wwd+(x((
#define lineno(y) ((int)(bfptr-topptr)/wwd+(y((

extern int VSG;
int last_x, last_y;
static int wht;
static int wwd;
static int wsz;
static char *topptr;
static char *bfptr;
static char *lstptr;
static int lines;
static char *endptr;
static int blkbeg;
static int blkend;
static int inserting;
static WINDOW *wnd;
static int do_display_text =1;

 ---------- */local function prototypes/* -----------
void erase_buffer(int *x, int*y;(
int lastword(int x, int y;(
void last_char(int *x, int *y;(
```

```
void test_para(int x, int y;(
int traling_spaces(int;(
int first_wordlen(int y;(
void paraform(int x, int y;(
int blankline(int line;(
void delete_word(int x, int y;(
void delete_line(int y;(
void delete_block(void;(
void copy_block(int y;(
void move_block(int y;(
void mvblock(int y, int moving;(
void findlast(void;(
void find_end(int *x, int *y;(
void carrtn(int *x, int *y, int insert;(
void backspace(int *x, int *y;(
void fore_word(int *x, int *y, char *bf;(
int spaceup(int *x, int *y, char **bf;(
void back_word(int *x, int *y, char *bf;(
int spacedn(int *x, int *y, char **bf;(
void forward(int *x, int* y;(
int downward(int *y;(
void upward(int *y;(
void display_text(void;(
void disp_line(int y;(
void insert_line(void;(

 ----- */Process text entry for a window/* ----- .
void text_editor(WINDOW *wnd1, char *bf, unsigned bsize(
}
    char *b, *buff;
    int depart = FALSE, i, c;
    int x, y, svx, svlw, tx, tabctr = 0;

    wnd = wnd1;
    wht = HEIGHT-2;
    wwd = WIDTH-2;
    topptr = bfptr = bf;
    lines = bsize / wwd;
    endptr = bf + wwd * lines;
    blkbeg = 0;
    blkend = 0;
    inserting = FALSE;
    x = 0;
    y = 0;
    display_text;()
 --------- */read in text from the keyboard/* ----------
findlast;()
while (TRUE}    (
    last_x = COL + 1 + x;
    last_y = ROW + 1 + y;
    cursor(last_x, last_y;(
    buff = curr(x, y;(
    if (tabctr} (
--        tabctr;
        c;' ' =
{
    else}
        c = get_char;()
        clear_message;()
{
    switch (c}  (
```

```
        case '\r': carrtn(&x, &y, inserting;(
                   break;
        case DN:  downward(&y;(
                   break;
        case PGUP: y = 0;
                   for (i = 0; i < wht; i(++
                       upward(&y;(
                   break;
        case PGDN: y = HEIGHT - 2;
                   for (i = 0; i < wht; i(++
                       downward(&y;(
                   y = 0;
                   break;
        case '\t': if (x + NEXTTAB < wwd}  (
                       if (inserting(
                           tabctr = NEXTTAB;
                       else
                           x += NEXTTAB;
{
                    else
                        carrtn(&x, &y, inserting;(
                    break;
        case SHIFT_HT:
                    if (x < TAB}    (
                        upward(&y;(
                        x = LASTTAB;
{
                       else
                           x -= PREVTAB;
                       break;

           case CTRL_FWD:
                       fore_word(&x, &y ,buff;(
                       break;
           case CTRL_BS:
                       back_word(&x, &y, buff;(
                       break;
           case CTRL_B:
                       y = wht - 1;
                       break;
           case CTRL_T:
                       y = 0;
                       break;
           case CTRL_HOME:
                       x = y = 0;
                       bfptr = topptr;
                       display_text;()
                       break;
           case HOME:  x = 0;
                       break;
           case CTRL_END:
                       find_end(&x, &y;(
                       display_text;()
                       break;
           case END:   last_char(&x, &y;(
                       break;

           case UP:    upward(&y;(
                       break;
```

```
            case F2:
            case ESC:   depart = TRUE;
                        break;
            case '\b:'
            case BS:    if (curr(x, y) == topptr(
                        break;
                        backspace(&x, &y;(
                        if (x == wwd - 1(
                            last_char(&x, &y;(
                        if (c == BS(
                            break;
                        buff = curr(x, y;(
            case DEL:   movmem(buff+1, buff, wwd-1-x;(
)*                        buff+wwd-1-x;' ' = (
                        disp_line(y;(
                        test_para(x+1, y;(
                        break;
            case ALT_D: delete_line(y;(
                        break;
            case INS:   inserting ^= TRUE;
                        insert_line;()
                        break;

            case F3:    erase_buffer(&x, &y;(
                        break;
            case F4:    paraform(0, y;(
                        break;
            case F5:    blkbeg = lineno(y) + 1;
                        if (blkbeg > blkend(
                            blkend = lines;
                        display_text;()
                        break;
            case F6:    blkbeg = lineno(y) + 1;
                        if (blkbeg < blkend(
                            blkend = 1;
                        display_text;()
                        break;
            case F7:    move_block(y;(
                        break;
            case F8:    copy_block(y;(
                        break;
            case F9:    delete_block;()
                        break;
            case F10:   blkbeg = blkend = 0;
                        display_text;()
                        break;
            case FWD:   forward(&x, &y;(
                        break;
            default:    if (!isprint(c((
                            break;
                        if (curr(x, y) == endptr-1||
)                           lineno(y)+1 >= lines && inserting
* &&                                    curr(wwd-2, y} ((' ' =! (
                            error_message(" End of Buffer;("
                            break;
{
                        if (inserting}  (
                             buff = curr(x, y;(
                             movmem(buff, buff + 1, wwd-1-x;(
{
```

```
                        buff = curr(x, y;(
                        if (buff < endptr}  (
                            if (buff >= lstptr(
                                lstptr = buff + 1;
*                            buff = c;
                            disp_line(y;(
{
                        buff = curr(wwd-1, y;(
                        if (endptr* && buff} (' ' =!
                            for (b = buff+1; b < endptr; b(++
                        if (*b==' ' && *(b + 1(' '==(
                                    break;
                            movmem(buff+1, buff+2, b-buff-1;(
)*                        buff+1;' ' = (
                        svx = x;
                        svlw = lastword(x, y;(
                        x = wwd-1;
                        if (*(buff-1(' ' =! (
                            back_word(&x, &y, buff;(
                            tx = x;
                            carrtn(&x, &y, TRUE;(
                        if (svlw(
                            x = svx-tx;

                        else}
                            x = svx;
--                            y;
{
{
                    forward(&x, &y;(
                    break;
{
              if (depart(
              break;
{
         inserting = FALSE;
         insert_line;()
{

 -------- */erase the buffer/* ----------------
static void erase_buffer(int *x, int *y(
}
    int c = 0;
    WINDOW *sur;

    sur = establish_window(28, 11, 4, 24;(
    set_colors(sur, ALL, RED, YELLOW, BRIGHT;(
    display_window(sur;(
    wprintf(sur, " Erase text window\n Are you sure? (y/n;("(
    while (c != 'y' && c != 'n}    ('
        c = get_char;()
        c = tolower(c;(
        if (c == 'y}   ('
            lstptr = bfptr =topptr;
*            x = *y = 0;
            setmem(bfptr, lines * wwd;(' ' ,
            blkbeg = blkend = 0;
            display_text;()
{
{
    delete_window(sur;(
```

```
{
 ----- */see if a word is the last word on the line/* ------
static int lastword(int x, int y(
}
    char *bf = curr(x, y;(

    while (x++ < wwd-1(
        if (*bf(' ' == ++
            return 0;
    return 1;

{
 */str 188/*
 --- */go to last displayable character on the line/* ---
static void last_char(int *x, int *y(
}
    char *bf;

*    x = wwd-1;
    bf = curr(0, *y;(
    while(*x)* && bf + *x(' ' == (
*)--         x;(
    if (*x && *x < wwd-1(
*)         x;++(
{

 ----- */test to see if paragraph should be reformed/* -----
static void test_para(int x, int y(
}
    int ts, fw;

    if(!scroll_lock() && y < lines}    (
         ts= trailing_spaces(y;(
         fw = fierst_wordlen(y+1;(
         if (fw && ts > fw(
              paraform(x, y;(
{
{

 ---- */count the trailing spaces on a line/* -----
static int trailing_spaces(int y(
}
    int x = wwd-1 ,ct = 0;

    char *bf=curr(0, y;(

    while (x >= 0}  (
        if (*(bf + x(' ' =! (
            break;
--        x;
        ct;++
{
    return ct;
{

 ----- */count the length of the first word on a line/* ---
static int fiest_wordlen(int y(
}
    int ct = 0, x = 0;
    char *bf = curr(0, y;(
```

```
    while (x < wwd-1 && *(bf+x(' ' == (
         x;++
    while (x+ct < wwd-1 && *(bf+x+ct(' ' == (
         ct;++
    return ct;
{

 -----------------*/form a paragraph/* --------------------
static void paraform(int x, int y(
}
    char *cp1, *cp2, *cpend, *svcp;
    int x1;

    if (blankline(lineno(y)+1((
         return;
    if (!blkbeg} (
         blkbeg = blkend = lineno(y)+1;
         blkend;++
         while(blkend < lines} (
              if (blankline(blkend((
                   break;
              blkend;++
{
--         blkend;
{

    if(lineno(y) != blkend-1(
         x=0;
    x1=x;
    cp1=cp2=topptr + (blkend-1) * wwd + x;
    cpend = topptr + blkend * wwd;
    while(cp2 < cpend}(
         while(*cp2 == ' ' && cp2 < cpend(
                   cp2;++

         if(cp2 == cpend(
              break;
 */          at a word/*
         while(*cp2 != ' ' && cp2 < cpend} (
              if(x1 >= wwd-1}  (
 */                    wrap the word/*
                   svcp = cp1 + (wwd - 1;(
                   while(*--cp1} (' ' =!
*                        cp1;' ' =
--                        cp2;
{
                   x1 = 0;
                   blkbeg;++
                   cp1 = svcp;
{
*                cp1++ = *cp2;++
                 x1;++
{
              if(cp2 < cpend} (
*                      cp1;' ' = ++
                     x1;++
{
{
         while(cp1 < cpend(
*              cp1;' ' = ++
                blkbeg;++
```

```
         if(blkbeg <= blkend(
              delete_block;()
         blkbeg =blkend =0;
         display_text;()
         findlast;()
{

 ------------ */test for a blank line/* ----------------
static int blankline(int line(
}
    char *cp;
    int x;

        cp = topptr + (line-1) * wwd;
    for(x=0; x < wwd; x(++
         if(*(cp+x(' ' =! (
              break;
    return(x == wwd;(
{

 ------------*/delete a word/* ------------------------
static void delete_word(int x, int y(
}
    int wct = 0;
    char *cp1, *cp2;
    cp1= cp2= curr(x, y;(
    if(*cp2(' ' ==
         while(*cp2 == ' ' && x+wct < wwd} (
              wct;++
              cp2;++
{
    else}
         while(*cp2 != ' ' && x+wct < wwd} (
              wct;++
              cp2;++
{
         while(*cp2 == ' ' && x+wct < wwd} (
              wct;++
              cp2;++
{
{
    movmem(cp2, cp1, wwd-x-wct;(
    setmem(cp1+wwd-x-wct, wct;(' ' ,
        dispay_text;()
    findlast;()
{

 ------------------*/delete a line/*-----------------
static void delete_line(int y(
}
    char *cp1, *cp2;
    int len;

        cp1=bfptr + y + wwd;
    cp2=cp1 + wwd;
    if(cp1<lstptr} (
         len=endptr - cp2;
         movmem(cp2, cp1, len;(
         lstptr -= wwd;
         setmem(endptr-wwd, wwd;(' ',
         display_text;()
```

```
{
{

 ----------*/delete a block/*--------------------
static void delete_block()
}
    char *cp1, *cp2;
    int len;

        if(!blkbeg  || !blkend} (
         putchar(BELL;(
         return;
{
    cp1=topptr + blkend * wwd;
    cp2=topptr + (blkend-1 * (wwd;
    len=endptr - cp1;
    movmem(cp2, cp1, len;(
    setmem(cp2+len, endptr-(cp2+len;(' ' ,(
    blkbeg = blkend = 0;
    lstptr-=(cp1 - cp2;(
    display_text;()
{

 -----------*/move and copy text blocks/*--------------
static void mvblock(int y , int moving(
}
    char *cp1, *cp2, *hd;
    int len;
    if(!blkbeg || ! blkend} (
         putchar(BELL;(
         return;
{
    if(lineno(y) > blkbeg-1 && lineno(y) <= blkend-1} (
         error_message("Can't move/copy a blok into itself;("
         return;
{
    len=(blkend - blkbeg + 1) * wwd;
    if((hd=malloc(len)) == 0(
         return;
    cp1 = topptr + (blkbeg + 1) * wwd;
    movmem(cp1, hd, len;(
    cp2 = topptr + lineno(y) * wwd;
    if (moving} (
         if(lineno(y) > blkbeg-1(
              cp2 -= len;
         do_display_text =0;
         delete_block;()
         do_display_text =1;
{
    if(cp2+len <= endptr} (
         movemem(cp2, cp2 + len, endptr - cp2 - len;(
                movemem)hd, cp2, len;(
{
    free(hd;(
    blkbeg = blkend = 0;
    display_text;()
{


 ------------*/copy a block/*-----------------------
static void copy_block(int y(
```

```
}
    mvblock(y, FALSE;(
    findlast;()
{

 -------------------*/move a block/*-----------------
static void move_block(int y(
}
    mvblock(y, TRUE;(
{

 --------*/find the last  caracter in the buffer/*-------
static void findlast()
}
    register char *lp = endptr - 1;
    register char *tp = topptr;

        while(lp > tp && (*lp == ' ' || *lp == '\0}  (('
         if (*lp == '\0('
*              lp;' ' =
--         lp;
{
        if(*lp(' ' =!
           lp;++
        lstptr = lp;
{

 -------*/go to end of the data in the buffer/*---------
static void find_end(int *x, int *y(
}
    int ct;

        bfptr = lstptr;
    ct = (lstptr - topptr) % wsz;
    bfptr -= ct;
    if (bfptr + wsz > endptr(
         bfptr = endptr - wsz;
*    x = 0;
*    y = (ct / wwd;(
    downward(y;(
{

----------------------*/carriage return/*----------------
static void carrtn(int *x, int *y, int insert(
}
    int insct;
    char *cp, *nl;
    int ctl = 2;

        cp = curr(*x, *y;(
    nl = cp + ((cp - topptr) % wwd;(
    if (lineno(*y)+2 < lines(
         if (insert && nl < endptr} (
              insct = wwd - *x;
              while (ctl}   (--
                  if (endptr > cp + insct} (
                     movemem(cp, cp+insct, endptr-insct-cp;(
                     setmem(cp, insct;(' ' ,
{
                  else if(ctl == 1(
                      setmem(cp, endptr - cp;(' ' ,
```

```
                  cp += insct * 2;
                   insct= *x;
{
{
*    x = 0;
    downward(y;(
    if (insert} (
         testpara(*x, *y;(
         display_text;()
{
    if (lineno(*y)+2 < lines(
         if (insert(
              if ((lstptr + wwd) <= endptr(
                   if(lstptr > curr(*x, *y((
                        lstptr += wwd;
{
 ------- */move the buffer offset back one position/* ------
static void backspase(int *x, int *y(
}
    if (*x == 0}    (
*        x = wwd - 1;
        upward(y;(
{
    else
*)--        x;(
{
 ------- */move the buffer offset forward one word/* ------
static void fore_word(int *x, int *y, char *bf(
}
    while (*bf}  (' ' =!
        if (spaceup(x, y, &bf) == 0(
            return;
        if (*x == 0(
            break;
{
    while (*bf(' ' ==
        if (spaceup(x, y& ,bf) == 0(
            return;
{
static int spaceup(int *x, int *y, char **bf(
}
    if (*bf == lstptr(
        return 0;

*)    bf;++(
    forward(x, y;(
    return 1;
{
 ------- */move the buffer offset backrward one word/* ------
static void back_word(int *x, int *y, char *bf(
}
    spacedn(x, y, &bf;(
    while (*bf(' ' ==
        if (spacedn(x, y, &bf) == 0(
            return;
    while (*bf}  (' ' =!
        if (*x == 0(
            return;
        if (spacedn(x, y, &bf(0 == (
            return;
{
```

```
    spaceup(x, y, &bf;(
            return;
{
static int spacedn(int *x, int *y, char **bf(
}
    if (*bf == topptr(
        return 0;
*)--    bf;(
    backspace(x, y;(
    return 1;
{
 ------- */move the buffer offset forward one position/* ------
static void backspace(int *x, int *y(
}
    if (*x == 0}    (
*       x = wwd - 1;
       upward(y;(
{
    else
*)--       x;(
{

/* ----------------------------------------------------------------
static void forward(int *x, int *y(
}
    int ww = wwd;

*)    x;++(
    if (*x == ww}    (
        downward(y;(
*        x = 0;
{
{
 ------- */move the buffer offset down one position/* ------
static int downward(int *y(
}
    if (*y < wht - 1}    (
*)    y;++(
    return 1;
{
    else if ((bfptr + wsz) < endptr}    (
        bfptr += wwd;
        scroll(wnd, UP;(
        disp_line(wht-1;(
        return 1;
{
    return 0;
{
 --------- */move the buffer offset up one position/* ------
static void upward(int *y(
}
    if (*y/*    !!!!!!!!!!    */        (
*)--       y;(
    else if ((topptr + wwd) <= bfptr}    (
        bfptr -= wwd;
        scroll(wnd, DN;(
        disp_line(0;(
{
{
 ---- */display all the lines in a window/* ------
static void display_text()
```

```
}
    int y = 0;

    if (do_display_text(
        while (y < wht(
        disp_line(y;(++
{
 ---------- */Display a line/* --------
static void disp_line(int y(
}
    int x = 0, atr = WNORMAL;

    if (blkbeg || blkend(
        if (lineno(y) >= blkbeg-1(
            if (lineno(y) <= blkend-1(
                atr = WACCENT;

    while (x < wwd} (
        displine(wnd, x+1, y+1, *(bfptr+y * wwd+x), atr;(
        x;++
{
{
 ---------set insert /exchange cursor shape/*-----------
static void insert_line()
}
    set_cursor_type(inserting ? 0x0106 : 0x0607;(
{
```

Описание программы: editor.c

Программа editor.c содержит ряд операторов #define, которые управляют установкой позиции табуляции в окне редактора. Значение глобальной переменной TAB определено равным 4, что устанавливает позицию табуляции в четыре любых символа. Остальные глобальные переменные - NEXTAB, LASTTAB и PREVTAB - являются макровыражениями, позволяющими улучшить удобочитаемость текста программы в целом. Макрос curr возвращает указатель на символ в буфере редактирования в зависимости от значения его координат в окне, задаваемых аргументами X и Y. Макрос lineno возвращает номер строки в буфере, который соответствует относительному номеру строки в окне редактора, задаваемого значением аргумента y.

Некоторые переменные, объявленные как external, являются вычисляемыми и предназначены прежде всего для сокращения числа вычислительных операций в тексте программы, что делает листинг более удобочитаемым. Переменная wht принимает значение высоты максимально возможной области в окне, отводимой под текст; при вычислении значения wht делается коррекция на наличие символов рамки окна. Переменная wwd принимает значение ширины максимально возможной области, отводимой под текст, с учетом наличия символов рамки окна. Переменная wsz содержит размер области окна, отводимой под текст. Переменная lines содержит количество строк текста, которые могут быть сохранены в текстовом буфере. Параметр endptr является указателем на позицию последнего символа в буфере плюс единица. Значение lstptr - указатель на последний, отличный от пробела, символ, хранящийся в буфере. Значение topptr является указателем на первый символ, хранящийся в буфере. Указатель bfptr используется при листании текста постранично или скроллинга

текста; он всегда указывает на символ в буфере текста, который в данный момент отображается в левом верхнем углу окна.

Функция text_editor вызывается в случае, если пользователю понадобилось вводить или модифицировать текст в буфере. После инициализации функции происходит вычисление соответствующих значений переменных, и курсор устанавливается в начальную позицию с координатами (0,0). При вводе пользователем символов с клавиатуры автоматически осуществляется проверка ввода на наличие прерываний от функциональных клавиш и управляющих сигналов. При нажатии пользователем клавиши <ВВОД> автоматически вызывается функция carrtn. Нажатие клавиши управления курсором <Стрелка вниз> приведет к вызову функции downward. Нажатие клавиш >Страница вверх> и <Страница вниз> приведут к вызову функций upward или downward соответственно для обработки отображенных в окне строк текста. Клавиши <TAB> и <SHIFT/TAB>) вызовут изменение значения текущей координаты X курсора, переместив последний в следующую или предыдущую позицию табуляции. Клавиши <Ctrl/стрелка вправо> приведут к вызову функций fore_word или back_word соответственно. <Ctrl/B> приведет к изменению координаты у курсора, позиционировав его в нижней части экрана. <Ctrl/T< позиционирует курсор в верхней части экрана, изменяя значение его координаты Y. <Ctrl/Home> приведет к перемещению курсора в точку с координатами (0,0) (изменив соответственно значения координатных аргументов курсора X и Y), переопределит значение bfptr на адрес начала буфера и вызовет функцию display_text для выдачи новой информации на экран из буфера. Клавиша <Курсор в начало экрана> (<Home>) обнуляет координату курсора X. >Ctrl/END>) использует функцию find_end для поиска последнего символа в буфере и выдачи текста на экран. <END> использует функцию find_end для позиционирования курсора (и изменения его координаты X) в конце текущей строки текста. <Up arrow> вызовет функцию upward. Нажатие <F2> или <ESC> сигнализирует о завершении ввода текста. <Left arrow> и <Backspace>) приводят к перемещению курсора на одну позицию влево, вызывая для этого функцию backspace. Клавиша <Забой> (<Backspace>) в этом случае не генерирует в программе кода, соответствующего клавише <DEL,< которая уничтожает символ в позиции курсора, обращаясь для этого к функции movmem. <ALT/D> приводит к вызову delete_line. <Ctrl/D< вызывает функцию delete _word. Клавиша <INS> переключает флаг inserting и вызывает функцию insert_line для изменения формы курсора. <F3> вызывает erase_buffer. <F4> вызывает paraform. Клавиши <F5> и <F6> устанавливают значения переменных blkbeg и blkend для текущей строки текста. Эти же клавиши приводят к обращению к функции display_text для отображения блока текста в инверсном режиме. Клавиши <F7>, <F8> и <F9> приводят к вызову функций move_ block, copy_block и delete_block соответственно. Клавиша <F10> обнуляет значения переменных blkbeg и blkend и вызывает функцию display_text. Нажатие клавиши <Стрелка вправо< >)Right arrow>) приводит к вызову функции forward. Если вводимый пользователем символ отличается от рассмотренных выше и является одним из отображаемых символов кода ASCII, то он копируется в буфер. При этом вначале анализируется ситуация на переполнение буфера в результате ввода символа (ввод символа может привести впоследствии к выходу за пределы буфера точно так же, как и вставка символа в режиме ВСТАВКА может привести к потере последнего символа результата операции). Затем, если включен режим ВСТАВКА, текущая строка сдвигается на один символ вправо, после чего символ записывается в буфер. Если в результате добавления символа в буфер был внесен последний символ, то указатель lstptr соответственно корректируется, и текущая строка

отображается на экране дисплея. Далее функция анализирует состояние "конец слова". Если последний символ строки отличен от пробела, то очередное слово считается завершенным. Функция сканирует буфер с начала до конца (от младшего адреса по возрастанию адресов) до тех пор, пока не достигнет конца буфера или не обнаружит двух следующих подряд пробелов. Затем она сдвигает текст на одну позицию вправо, начиная со следующей строки относительно строки текста, в которой обнаружен факт завершения слова. Таким образом, эта процедура создает промежуток между завершенным (законченным) словом и текстом следующей строки. Функция lastword вызывается для анализа нового символа, если он является символом, вводимым в последнее слово, или если факт завершения очередного слова явился результатом вставки в предыдущее слово строки. Эта проверка осуществляется программой в случае изменения местоположения курсора после обнаружения конца слова. Результат такой проверки запоминается в переменной svlw. Координата X принимает значение конца строки текста. Если символ, предшествующий последнему символу, отличен от пробела, то для позиционирования курсора в начале следующего слова (и изменения координаты X) используется функция back_word. Функция carrtn вызывается для логической вставки новой строки в текущей X-позиции. (На самом деле физической вставки строки непосредственно в буфер не происходит в силу того, что буфер представляет собой обычный двумерный массив прямоугольной формы. Поэтому осуществляется лишь сдвиг текста и освобождение внутри него пространства). Курсор позиционируется на символе, следующим за символом, непосредственно добавленным в буфер. Функция forward вызывается, не обращая внимания даже на факт окончания слова, для коррекции координаты X.

Функция erase_buffer открывает окно и запрашивает у пользователя разрешение на выполнение команды очистки (erase). По соглашению пользователя функция очищает буфер и переопределяет значения всех переменных и указателей.

Функция lastword проводит анализ координаты X расположения курсора в последнем слове строки, определяемой координатой Y.

Функция last_char переопределяет координату Y расположения курсора, присваивая ей значение на единицу больше, чем позиция расположения последнего отображаемого символа в строке.

Функция test_para вызывается в случае необходимости переформатирования параграфа для проверки возможности его выполнения. Переключатель клавиатуры <Scroll Lock> должен находиться в выключенном состоянии (off), и число завершающих пробелов текущей строки текста должно быть больше длины первого слова следующей строки текста. Если это условие выполняется, функция test_para вызывает функцию paraform, которая позволяет переформатировать параграф.

Функция trailing_spaces осуществляет подсчет завершающих пробелов строки текста.

Функция first_woldlen подсчитывает длину первого слова следующей строки текста.

Функция paraform осуществляет переформатирование параграфа, если, естественно, предварительно параграф был отделен от остального текста двумя пустыми строками. Функция paraform вызывается в результате нажатия пользователем клавиши <F4> или в

результате работы функции test_para, осуществляющей автоматическое принятие решения о возможности переформатирования параграфа. Во всех этих случаях, если предварительно переменные blkbeg и blkend не получили соответствующих значений для блока текста, включающего текущий параграф, функция устанавливает соответствующие значения этих переменных. Значение переменной blkbeg устанавливается равным текущей строке текста. Функция просматривает текст вперед, начиная с текущей строки, на предмет наличия двух пустых строк (строк, целиком состоящих из пробелов(или достижения конца буфера, устанавливая при этом значение переменной blkend.

Переформатирование параграфа функция начинает со сканирования текста после предшествующих параграфу лидирующих пробелов. Если она обнаружила непустое слово, то сразу же начинается процесс копирования этого слова в начало буфера. Если обнаружен пробел, то функция логически уничтожает этот пробел. В процессе копирования слова функция анализирует ситуацию на предмет завершения строки текста, затем возвращается в начало копируемого слова, вставляет пробелы в буфер и продолжает работу далее. В связи с этим возможно логическое сжатие области, занимаемой группой слов, путем удаления избыточных пробелов. В результате работы функции могут быть образованы одна или более пустых строк, для удаления которых используется функция delete_block.

Функция blank_line используется в том случае, если специфицированная строка целиком состоит из пробелов для анализа строки.

Функция delete_word уничтожает слово в буфере. Если текущее значение координаты месторасположения курсора X соответствует пробелу, то функция уничтожает пробел, предшествующий следующему слову; таким образом, функция уничтожает слово, начиная с позиции, специфицированной кординатой X, но после пробела ,стоящего за предшествующим словом, до начала следующего слова.

Функция delete_line уничтожает строку текста с последующим сдвигом всего текста. Вначале функция вычисляет адрес первого символа текущей и следующей строки. Затем функция определяет объем текста, который должен быть сдвинут, как расстояние от следующей (за уничтоженной) строки до конца буфера текста. После перечисленных действий осуществляется сдвижка текста, и переменная lstptr принимает новое значение. Буфер заполняется до конца избыточными пробелами.

Функция delete_block реботает точно так же, как и функция delete_line, однако оперирует с блоком текста, который содержит более одной строки текста.

Функция mvblock используется для перемещения и копирования блока текста. Вначале происходит размещение в памяти буфера для хранения текста, и блок текста помещается в этот буфер. Если операция перемещения блока текста предпочтительнее, чем его копирование, то вызывается функция уничтожения блока текста delete_block() в буфере редактора. Текст в буфере редактора смещается вправо, начиная с текущей строки, освобождая место для блока текста, который будет перемещен сюда или скопирован на это место. Текст из промежуточного буфера перемещается на освобожденное после сдвижки текста пространство. Промежуточный буфер удаляется из памяти и восстанавливается изображение окна

после обращения к функции display_text.

Функции copy_block и move_block вызываются для копирования и перемещения блока текста соответственно. Эти функции используют mvblock для выполнения возложенных на них операций.

Функция findlast организует поиск последнего значащего символа в буфере и устанавливает указатель lstptr на следующую за ним позицию.

Функция carrtn отслеживает ситуацию возврата каретки при окончании строки текста внутри буфера редактора вслед за движением курсора пользователя. Она вычисляет адрес первого символа следующей строки текста по текущей позиции символа, в которой позиционирован курсор. Если включен режим "Вставка символа", функция должна обеспечить "раздвигание" строки текста, которое достигается смещением текста, находящегося в буфере редактора, вправо, начиная с позиции, в которой расположен курсор, к концу текущей строки и заполнения образовавшихся пустот избыточными пробелами. Новая строка подвергается точно такому же сдвигу и заполнению пробелами до минимальной ее длины после операции вставки.

Функция downward используется для отслеживания изменения координат следующей (возможно, новой) строки текста и, если включен режим "Вставка", для переопределения значения указателя lstptr.

Большинство функций, к которым осуществляется обращение, используют прямо или косвенно координаты позиции расположения курсора. Такими функциями являются find_end, backspace, fore_word, spaceup, back_word, spacedn, forward, downward и upward.

Пример: Использование редактора

Листинги 9.2, 9.3 и 9.4 демонстрируют пример использования оконного редактора текстов в диалоговой программе.

Листинг 9.2 содержит текст программы note.c, которая является главной функцией, обращающейся к функции notepad.c, исходный текст которой приведен в листинге 9.3. Функция notepad.c выделена в отдельный файл в связи с тем, что она используется в примерах меню (Глава 10) и при описании резидентных утилит (Глава .(12

Листинг 9.4 содержит пример make-файла Турбо Си, в соответствии с которым строится пример использования оконного редактора.

Функция notepad.c обрабатывает файл, имя которого определено в массиве, описанном как external. В данном примере этот массив определен в note.c, и имя файла указано как note.pad. Если такой файл уже существует на момент запуска примера, то notepad.c считывает его в буфер редактора.

Notepad.c устанавливает окно, заключает его в рамку, присваивает окну название, назначает цвета для окна и отображает его на экране дисплея. Затем notepad.c вызывает функцию оконного редактора text_editor. По завершении работы функции text_editor notepad.c уничтожает окно и сохраняет строки текста в буфере, в который был записан последний нужный текст. Затем notepad.c

записывает содержимое буфера в файл (note.pad.(

Листинг 9.2. note.c

```
 ------ */note.c/*---------

#include "twindow.h"

void notepad(void;(
char notefile [] = "note.pad;"

main()
}
  load_help("tcprogs.hlp;("
  notepad;()
{
```

Листинг 9.3: notepad.c

```
 ----------------- */notepad.c/*-----------------

#include <stdio.h<
#include <mem.h<
#include "twindow.h"

#define LWID 60
#define WHT 10
#define PADHT 20

char bf [PADHT] [LWID;[
extern char notefile;[]

void notepad()
}
        WINDOW *wnd;
        FILE *fp, *fopen;()
        int i, ctr = 0;

        set_help("notepad ",0,0;(
        setmem(bf, sizeof bf;(' ' ,
        if ((fp = fopen(notefile, "rt")) != NULL}  (
            while (fread(bf [ctr], LWID, 1, fp((
                ctr;++
            fclose(fp;(
{
        wnd = establish_window
)-80))               LWID+2))/2, (25-(WHT+2))/2, WHT+2, LWID+2;(
        set_border(wnd, 3;(
        set_title(wnd, "  Note Pad;("
        set_colors(wnd, ALL, BLUE, AQUA, BRIGHT;(
        set_colors(wnd, ACCENT, WHITE, BLACK, DIM;(
        display_window(wnd;(
        text_editor(wnd, bf[0], (unsigned) LWID * PADHT;(
        delete_window(wnd;(
        ctr = PADHT;
        while (--ctr} (
                for (i = 0; i < LWID; i(++
```

```
                        if( bf [ctr] [i(' ' =! [
                                break;
                if (i < LWID(
                        break;
{
        fp = fopen(notefile, "w;("
        for (i = 0; i < ctr+1; i(++
                fwrite(bf[i], LWID, 1, fp;(
        fclose(fp;(
{
```

Листинг 9.4 note.prj

```
note
notepad(twindow.h(
editor(twindow.h, keys.h(
thelp(twindow.h, keys.h(
twindow(twindow.h, keys.h(
ibmpc.obj
```

Для выполнения примера введите команду

```
C > note
```

После трансляции и редактирования ваших программ вы увидите на экране дисплея копию рисунка 9.2. При первом запуске примера файл note.pad будет пустым. После того, как вы введете с помощью редактора какую-либо информацию, при выходе из редактора по клавише <ESC> или <F2> данные будут запомнены в файле note.pad для дальнейшего использования. Естественно, что если вы запомнили текст, то при следующем выполнении программы он будет вам отображаться в окне редактора.

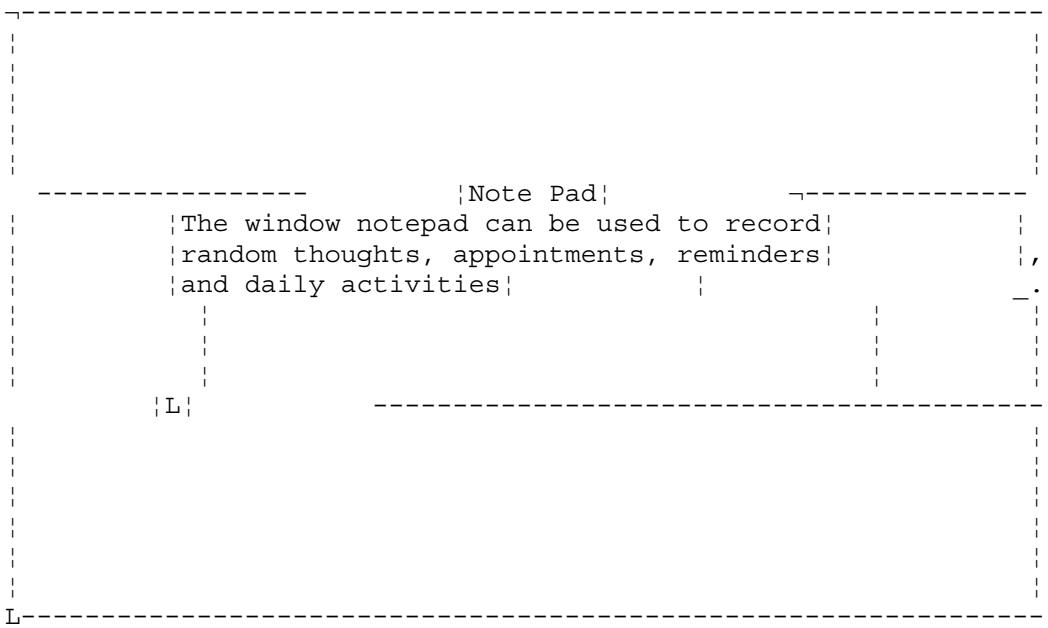


Рис. 9.2 Пример использования редактора.

При вводе текста и его дальнейшей обработке используйте перечень команд, отображенный на рисунке 9.1. Нажав клавишу <F1,< вы можете получить перечень этих команд на экране своего дисплея.

Резюме

Итак, мы рассмотрели еще одну дополнительную возможность, которая может быть добавлена в список библиотеки поддержки оконной технологии. Таким образом, изученное вами на настоящий момент программное обеспечение может использоваться как функциональное средство поддержки специализированных задач, ориентированных на использование оконной технологии. Программное обеспечение позволяет организовать диалоговую help-поддержку, шаблоны ввода данных, а также использовать оконный текстовый редактор. Следующей дополнительной возможностью, которая может вами с успехом применяться, является иерархическая система меню, используемых в окне. Эта система меню включает в себя скользящее меню-строку и систему появляющихся меню типа "pop-down" (этот тип меню еще называют "всплывающим", так как его появление на экране очень напоминает эффект всплытия из "глубины" дисплея) для выбора и выполнения различных функций прикладной программой, в которой система меню используется.

ГЛАВА 10

Оконные меню

Итак, сейчас вы уже можете адресовать область пользовательского интерфейса интерактивной системы, ссответствующую определенным операциям. Для ввода различных полей данных - это средства поддержки формо-ориентированного ввода; для ввода текста - это оконный текстовый редактор; для обеспечения пользователей help-информацией по этим операциям - это контекстно -зависимая help-система; для поддержки каждой из этих операций- библиотека функций окна. Ранее мы с вами рассмотрели примеры адресации области пользовательского интерфейса для случая, когда прикладная программа использует только одну из операций. Однако диалоговые системы обычно обеспечивают поддержку целого ряда операций, которые может инициализировать пользователь. Выбор конкретной операции, которую необходимо выполнить в данный момент, чаще осуществляется пользователем интерактивной системы, чем самой системой. Это связано с тем, что зачастую требуется выбор из группы сложных независимых операций, а пользователь всегда лучше знает, какая именно операция должна быть выполнена в данный момент.

Меню

Часто перечень возможностей системы отображается пользователю в виде списка, из которого он может выбрать нужные ему функции. Наиболее часто этот подход носит название меню. Перечень выполняемых функций является одной из основных частей

пользовательского интерфейса системы и обсуждается далее.

Вы уже встречались с одним из видов меню в Главе 6 при рассмотрении программы poetry.exe. Это меню содержало список опций окна. Пользователь мог осуществлять выбор из этого меню путем нажатия клавиш, соответствующих одной из опций меню, или путем перемещения курсора по элементам меню и нажатия клавиши >ВВОД> в позиции соответствующего элемента. Такой тип меню является наиболее часто используемым в интерактивных системах, но в то же время относится к разряду наиболее эффективных и понимаемых пользователем приемов.

Другим популярным типом меню, который часто используется в компьютерах, снабженных графическим пользовательским интерфейсом и манипулятором типа "мышь", является меню в виде скользящей строки (sliding bar menu). Этот тип меню представляет собой горизонтальное меню, расположенное в верхней части, с "всплывающими" вертикальными меню, раскрывающими и конкретизирующими содержание выбранных из горизонтального меню элементов. Вертикальные меню "всплывают" под соответствующими элементами горизонтального меню, и после этого пользователь получает возможность осуществлять выборку из вертикального меню, перемещаясь по нему вверх и вниз. Примером такого меню может служить система меню, принятая во Framework-II - широко распространенном продукте фирмы Ashton Tate.

К преимуществам такого типа меню относится то, что оно занимает минимальную область экрана (обычно - одна строка) и наиболее полно отражает взгляд пользователя на конкретное приложение программы. Вертикальные меню реализованы в виде "всплывающих" окон. Вследствие того, что они лишь временно перекрывают изображение на экране, не уничтожая его, применение такого меню раскрывает новые перспективы в разработке диалоговых систем. На рисунке 10.1 представлен пример именно такого меню. Пользователи системы Borland's Superkey могут видоизменить это меню.

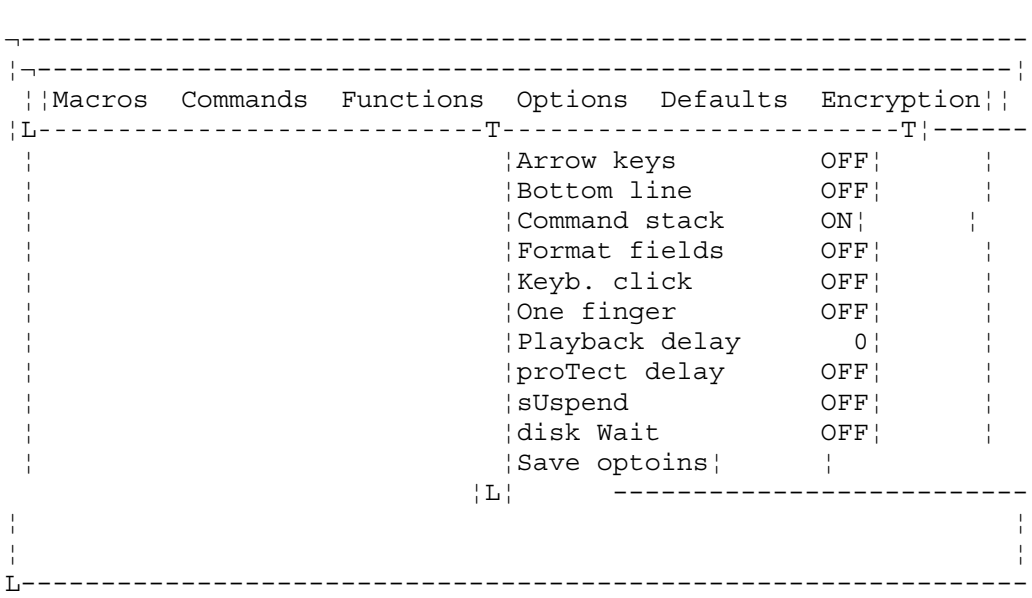


Рис. 10.1 Пример меню

Процесс, образующий оконное меню

В этой главе содержится вводная информация о функциях-драйверах меню, которые используются для создания и поддержки скользящего меню-строки, которое описано выше. Такое меню создается как окно и управляется с помощью последовательности управляющих таблиц, образующихся после обращения к программе. Эти таблицы описывают последовательность выборки в скользящем меню-строке и каждом из "всплывающих" вертикальных меню. Функции-драйверы меню осуществляют обслуживание основного (целевого) процесса выполнения вашей программы. Они управляют пользовательским интерфейсом путем контроля процесса отображения меню и ввода пользователем выбранного элемента меню. Процесс ввода управляется набором таблиц меню, которые кодируются в вашей программе.

Основной таблицей меню является массив MENU-структур. Структура MENU определена в twindow.h (см. Главу .(6 Этот массив является входным для каждого процесса выборки из скользящей меню-строки. Каждый ввод выбранного элемента меню содержит отображаемое пользователю имя элемента меню и ряд указателей, описывающих содержание "всплывающего" вертикального меню, соответствующего указанному пользователю элементу горизонтального меню. Эти указатели включают в себя указатель на массив имен элементов "всплывающего" меню и указатель на массив указателей функций на языке Си. Имена отображаются во "всплывающем" меню, а функции входят в состав системы, использующей это меню, и выполняются, когда пользователь отмечает в качестве выбранных соответствующие имена элементов меню.

Иерархическая система меню состоит из двух уровней. Первый уровень включает в себя выборку из горизонтального меню и ограничен лишь шестью элементами вследствие того, что элементы этого уровня располагаются на одной строке. Любая выборка элемента на этом уровне приводит к появлению соответствующего "всплывающего" меню на втором уровне. Размер "всплывающего" меню ограничен 21 элементами, что соответствует максимальному числу строк, которое может быть включено в "всплывающее" меню. Каждая выборка на этом уровне приводит к вызову определенной Си-функции из закрепленных за элементами меню.

Если вы хотите получить дополнительные уровни в вашем иерархическом меню, то должны описать дополнительные управляющие таблицы меню, присвоить им значения и определить рекурсивные вызовы функций поддержки системы меню. Вследствие того, что функции поддержки меню реентерабельны, выборка на втором уровне "всплывающего" меню может привести к появлению нового горизонтального меню.

Описание иерархии системы меню фактически осуществляет ваша прикладная программа (или программная система), которая непосредственно генерирует массивы меню. Пример, представленный на листинге 10.3 (листинг см. ниже в этой главе), поясняет процесс формирования массивов. Обсуждение этого примера включает обсуждение полученного изображения как результата предварительной генерации массивов меню, а также детальное описание самого процесса их генерации.

Функции поддержки меню

Для использования оконных меню в этой главе вы должны вначале сгенерировать массив MENU, затем массив, на который ссылается массив MENU ,а также написать и оттранслировать прикладные функции, которые будут выполняться, когда пользователь выберет конкретный элемент "всплывающего" меню. Затем вы так или иначе обращаетесь к функции menu_select, которая описана ниже.

```
void menu_select(char *name, Menu *mn(
```

Эта функция активизирует меню-процесс, отображая на экране дисплея горизонтальное скользящее меню-строку и переходя в состояние ожидания использования пользователем клавиатуры для выборки из меню. Указатель name является именем заголовка скользящей меню-строки. Указатель mn содержит адрес массива структур MENU в вызывающей программе. Этот массив и указатель на массив MENU определяют иерархию меню для меню-процесса.

После того, как функция menu_select отобразила горизонтальное меню, пользователь, используя клавиши управления курсором вправо и влево, может перемещаться по меню. Если пользователь нажал клавишу <КЛЮЧ> (<ESC>), то меню-процесс прерывается, и управление передается в точку вызова функции menu_ select. Если же пользователь нажал клавишу <ВВОД> (<ENTER>), то осуществляется привязка конкретного "всплывающего" вертикального меню к текущему элементу горизонтального меню (определяется текущим положением маркера меню или курсора) и отображение нужного вертикального меню.

Во время отображения вертикального "всплывающего" меню пользователь может использовать клавиши передвижения курсора вправо и влево для выбора других элементов горизонтального меню-строки. В этом режиме в соответствии с движением курсора в меню-строке осуществляется последовательное отображение вертикальных меню, причем перед отображением очередного вертикального меню предыдущее вертикальное меню уничтожается. Если пользователь нажмет клавишу <КЛЮЧ> (<ESC>), то текущее "всплывающее" меню уничтожается, и процесс возвращается к обработке скользящего меню-строки аналогично описанному выше. Пользователь может использовать клавиши перемещения курсора вверх и вниз для навигации по вертикальному меню и выбора нужного элемента меню. После того, как нужный элемент выбран, нужно нажать клавишу <ВВОД> (<ENTER>). Нажатие этой клавиши приведет к исчезновению как вертикального, так и горизонтального меню и вызову функции, соответствующей указателю, хранимому в массиве указателей функций "всплывающего"меню и, в свою очередь, соответствующего выбранному элементу меню. Выполнение функций осуществляется путем обычного обращения к ним.

После того как управление от вызванной функции передается в точку ее вызова, вся система меню восстанавливается на экране дисплея в том состоянии, которое предшествовало вызову функции, реализующей идентифицированное пользователем действие, и меню-процесс продолжается в соответствии с его описанием.

Исходный листинг: tmenu.c

Листинг 10.1 содержит текст библиотечной функции tmenu.c, предназначенной для поддержки предварительного описания

меню-процесса.

Листинг 10.1: tmenu.c

```
 ------------ */    tmenu.c/* ------------

#    include <stdio.h<
#    include <conio.h<
#    include <stdlib.h<
#    include "keys.h"
#    include "twindow.h"

    extern int VSG;

    WINDOW *open_menu(char *mnm, MENU *mn, int hsel;(
    int gethmenu(MENU *mn, WINDOW *hmenu, int hsel;(
    int getvmn(MENU *mn ,WINDOW *hmenu, int *hsel, int vsel;(
    int haccent(MENU *mn, WINDOW *hmenu, int hsel, int vsel;(
    void dimension(char *sl[], int *ht, int *wd;(
    void light(MENU *mn, WINDOW *hmenu, int hsel, int d;(

 ------------ */   Отображение и обработка меню/* ---------
   void menu_select(char *name, MENU *mn(
}
WINDOW *open_menu;()
WINDOW *hmenu;
int sx, sy;
int hsel = 1, vsel;
curr_cursor(&sx, &sy;(
cursor(0, 26;(
hmenu = open_menu(name, mn, hsel;(
while (hsel = gethmenu(mn, hmenu, hsel}     ((
    vsel = 1;
    while (vsel = getvmn(mn, hmenu, &hsel, vsel}       ((
 delete_window(hmenu;(
 set_help("", 0, 0;(
)*) mn+hsel-1)->func [vsel-1])(hsel, vsel;(
 hmenu = open_menu(name, mn, hsel;(
{
{
delete_window(hmenu;(
cursor(sx, sy;(
{
---- */    Инициализация горизонтального меню/*-------
   static WINDOW *open_menu(char *mnm, MENU *mn, int hsel(
}
int i = 0;
WINDOW *hmenu;
set_help("menu    ", 30, 10;(
hmenu = establish_window(0, 0, 3, 80;(
set_title(hmenu, mnm;(
set_colors(hmenu, ALL, BLUE, AQUA, BRIGHT;(
set_colors(hmenu, ACCENT, WHITE, BLACK, DIM;(
display_window(hmenu;(
while ((mn+i)->mname(
    wprintf(hmenu, " %-10.10s ", (mn+i++)->mname;(
light(mn, hmenu, hsel, 1;(
cursor(0, 26;(
return hmenu;
```

```
{
---- */    Выборка из горизонтальногоь меню/*------
   static int gethmenu(MENU *mn, WINDOW *hmenu, int hsel(
}
int sel;
 light(mn, hmenu, hsel, 1;(
while (TRUE}      (
    switch (sel = get_char}     (()
 case FWD:
 case BS:    hsel = haccent(mn, hmenu, hsel, sel;(
      break;
 case ESC:   return 0;
 case '\r':  return hsel;
 default:    putchar(BELL;(
      break;
{
{
{
----- */    Всплывающее вертикальное меню/*--------
   static int getvmn(MENU *mn,WINDOW *hmenu,int *hsel,int vsel(
}
WINDOW *vmenu;
int ht = 10, wd = 20;
char **mp;
 while (1}    (
    dimension((mn+*hsel-1)->mselcs, &ht, &wd;(
    vmenu = establish_window(2+(*hsel-1)*12, 2, ht, wd;(
    set_colors(vmenu, ALL, BLUE, AQUA, BRIGHT;(
    set_colors(vmenu, ACCENT, WHITE, BLACK, DIM;(
    set_border(vmenu, 4;(
    display_window(vmenu;(
    mp = (mn+*hsel-1)->mselcs;
    while (*mp(
 wprintf(vmenu, "\n%s", *mp;(++
    vsel = get_selection(vmenu, vsel;("" ,
    delete_window(vmenu;(
    if (vsel == FWD || vsel == BS}  (
* hsel = haccent(mn, hmenu, *hsel, vsel;(
 vsel = 1;
{
    else
 return vsel;
{
{
----- */    Управление отображением выбранных элементов
                        горизонтального меню/*-----
   static int haccent(MENU *mn,WINDOW *hmenu,int hsel,int sel(
}
switch (sel}      (
    case FWD:
 light(mn, hmenu, hsel ,0;(
 if ((mn+hsel)->mname(
     hsel;++
 else
     hsel = 1;
 light(mn, hmenu, hsel ,1;(
 break;
    case BS:
 light(mn, hmenu, hsel ,0;(
 if (hsel == 1(
     while ((mn+hsel)->mname(
```

```
  hsel;++
 else
--    hsel;
 light(mn, hmenu, hsel ,1;(
 break;
    default:
 break;
{
return hsel;
{
----- */    Вычисление высоты и ширины меню/*-------
   static void dimension(char *sl[], int *ht, int *wd(
}
unsigned strlen(char;(*
* ht = *wd = 0;
while (sl [*ht}        ([
*    wd = max(*wd, strlen(sl [*ht;(([
*)    ht;++(
{
*ht += 2;
*wd += 2;
{
-------- */    Отображение в соответствии с параметром
                    accent элемента горизонтального меню/*---
   static void light(MENU *mn, WINDOW *hmenu, int hsel, int d(
}
if (d(
    revers_video(hmenu;(
wcursor(hmenu, (hsel-1)*12+2, 0;(
wprintf(hmenu, (mn+hsel-1)->mname;(
normal_video(hmenu;(
cursor(0, 26;(
{
```

Описание программы: tmenu.c

Функция menu_select вызывается для обработки меню, описанного в массиве структур MENU. Эта функция запоминает текущее положение системы меню и позицию курсора относительно координат экрана. Затем вызывается функция open_menu, которая осуществляет инициализацию и отображение на экране (в его верхней части) горизонтального меню-строки. Цикл while осуществляет обработку горизонтального меню-строки до тех пор, пока пользователь не нажмет клавишу <КЛЮЧ> (<ESC>). На каждой итерации цикла вызывается функция getmenu. Если функция getmenu возвращает значение 0, значит, пользователь нажал клавишу <КЛЮЧ> (<ESC.(< Если функция getmenu возвращает ненулевое значение, значит, пользователь произвел выборку одного из элементов горизонтального меню. Возвращаемое функцией значение специфицирует выбранный пользователем элемент меню. Следующий цикл while обрабатывает "всплывающее" вертикальное меню, соответствующее выбранному ранее элементу горизонтального меню-строки. На каждой операции этого цикла происходит обращение к функции getvmn, которая осуществляет обработку "всплывающего" меню. При нажатии пользователем клавиши >КЛЮЧ> (<ESC>) функция getvmn возвращает значение 0, в противном случае getvmn возвращает значение, специфицирующее выбранный пользователем элемент вертикального меню. После этого окно,

выделенное для горизонтального и вертикальных меню, уничтожается, вызывается функция, реализующая действия, соответствующие выбранному пользователем элементу вертикального меню. По завершении работы этой функции open_menu вызывает восстановление изображения на экране горизонтального меню-строки.

Функция open_menu открывает окно в верхней части экрана на всю его ширину. Это окно предназначено для горизонтального меню-строки. Меню выбора функций вашей программы отображается в этом окне в соответствии с таблицей MENU. Первый из выбранных пользователем элементов этого меню отображается в акцентированном цвете функцией light.

Функция getmenu анализирует прерывания от клавиатуры при выборе элементов меню или при передвижении курсора по меню. Если пользователь нажал клавишу перемещения курсора вправо или влево, то вызывается функция haccent для перемещения курсора по меню-строке. Если пользователь нажал клавишу <КЛЮЧ> (<ESC>), то функция getmenu возвращает 0. При нажатии клавиши <ВВОД> (<ESC(< функция возвращает значение, соответствующее выбранному элементу меню, и передает управление в точку ее вызова.

Функция getvmn предназначена для обработки вертикального меню, "всплывшего" под соответствующим элементом горизонтального меню. Вертикальное меню представляет собой окно, а его состав определяется в соответствии с таблицей MENU. Функция get_selection (см. Главу 6) вызывается для считывания пользовательского выбора. После того, как в вызывающую программу передано значение выбранного пользователем элемента меню, окно, выделяемое для "всплывающего" меню, уничтожается. При нажатии пользователем клавиш перемещения курсора вправо или влево вызывается функция haccent для перемещения вперед или назад курсора по элементам горизонтального меню-строки, а обработка вновь появляющихся (в процессе перемещения курсора по горизонтальному меню) вертикальных меню осуществляется опять функцией getvmn; в противном случае значение, возвращаемое функцией get_selection приводит к возврату управления в точку вызова функции getvmn.

Пример оконного меню

Листинги 10.2, 10.3 и 10.4 содержат пример программы, иллюстрирующей использование и обработку меню. Этот пример строит меню, позволяющее выполнять в рамках одной интегрированной системы все оконные функции, рассмотренные в качестве примеров, иллюстрирующих содержание глав с шестой по девятую.

Листинг 10.2, программа menu.c, является простейшей управляющей программой, которая вызывает программу, реализующую пример (exec.c) и представленную листингом 10.3. Листинг 10.4 программы menu.prj является make-файлом Турбо Си, по которому строится выполняемая программа, реализующая пример.

Листинг 10.2:menu.c

```
------------------ */     menu.c/*-------------------

#     include "twindow.h"
     void exec(void;(

     char notefile [] = "note.pad;"

     main()
}
       load_help("tcprogs.hlp;("
       exec;()
{
```

Листинг 10.3:exec.c

```
---------------- */      exec.c/* -------------------
#      include <stdio.h<
#      include "twindow.h"
---------- */      Локальные прототипы/* -----------
      void testmove(void;(
      void promote(void;(
      void ccolor(void;(
      void fasttest(void;(
      void notepad(void;(
      void ordent)void;(
      void poems(void;(
      void maxims(void;(

----------- */      Таблицы меню/* ------------------
      char *dselcs} = []
 "          move,"
 "          promote,"
 "          colors,"
 "          fast,"
          NULL
;{
      char *pselcs} = []
 "          notepad,"
 "          orders,"
 "          poetry,"
 "          sayings,"
          NULL
;{
      static void (*dfuncs[])()={testmove,promote,ccolor,fasttest;{
      static void (*pfuncs[])()={notepad,ordent,poems,maxims;{
      static MENU tmn} = []
 "}          demos ",      dselcs, dfuncs,{
 "}          programs ",   pselcs, pfuncs,{
}          NULL,NULL,NULL{
;{
             void exec()
}
          menu_select(" TC Executive ", tmn;(
{
```

Листинг 10.4:menu.prj

```
      menu.c
      exec.c (twindow.h, keys.h(
      testmove (twindow.h, keys.h(
      promote (twindow.h, keys.h(
```

```
ccolor (twindow.h, keys.h(
fasttest (twindow.h(
notepad (twindow.h(
ordent (twindow.h(
maxims (twindow.h, keys.h(
poems (twindow.h, keys.h(
editor (twindow.h, keys.h(
entry (twindow.h, keys.h(
thelp (twindow.h, keys.h(
tmenu (twindow.h(
twindow (twindow.h, keys.h(
ibmpc.obj
```

Для запуска программы-примера введите следующую команду:

```
C>MENU
```

Вид экрана дисплея, изображенный на рисунке 10.2, будет соответствовать ситуации, когда на экране отображается скользящее меню-строка. В этом примере доступны для выбора только два элемента меню, однако скользящее меню-строка может включать в себя до шести элементов, доступных для выбора пользователем.

Обратимся теперь к массиву структур MENU, поименованному как tmn (см. листинг 10.3 программы exec.c). Массив содержит лишь два элемента, соответствующих элементам горизонтального меню, отображаемых пользователю. Структура MENUI определена в twindow.h)см. Главу 6) и включает три элемента: указатель на имя выбранного элемента меню, указатель на массив имен элементов "всплывающего" меню, соответствующего элементу горизонтального меню, и указатель на массив указателей функции, реализующей ту или иную операцию.

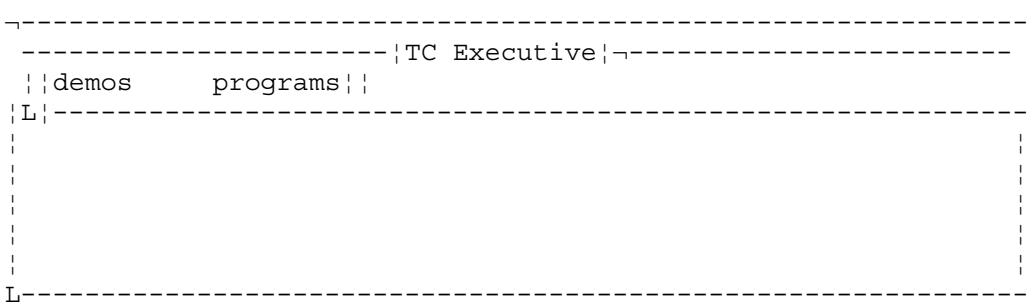


Рис. 10.2 Горизонтальное скользящее меню.

Для перемещения по меню с целью выборки нужного элемента из горизонтального меню используются клавиши перемещения курсора вправо и влево. Если текущим является элемент меню demos, нажмите клавишу <Ввод>. Теперь вы можете увидеть на экране картинку, аналогичную изображенной на рисунке 10.3, на котором также изображено и вертикальное меню, соответствующее элементу горизонтального меню demos. Теперь вернемся к листингу 10.3 и обратим внимание на содержимое первого элемента массива tmn, который соответствует элементу горизонтального меню demos. Здесь dselcs - адрес, указывающий размещение массива указателей типа char на имена элементов demosвертикального "всплывающего" меню, dfuncs - адрес размещения массива указателей функций. Каждый из указателей этого массива адресует одну из функций, которые рассматривались в качестве примеров в предыдущих главах. Для перемещения по вертикальному меню применяются клавиши перемещения

курсора вверх и вниз. Если вы остановились на каком-то выборе, нажмите клавишу <Ввод>. После этого будет выполнена одна из функций в зависимости от того, какой элемент меню вы выбрали. После передачи управления из этой функции в точку ее вызова (как было показано в этой главе, для функций, требующих какого-либо выбора от пользователя, это клавиша <ESC>, на экране восстанавливается изображение горизонтального и вертикального меню в том состоянии, которое предшествовало вызову функции, и разрешена дальнейшая работа с ними. Для уничтожения текущего вертикального меню нажмите клавишу <ESC>. Для выхода из программы и уничтожения горизонтального меню требуется повторное нажатие клавиши <ESC.<

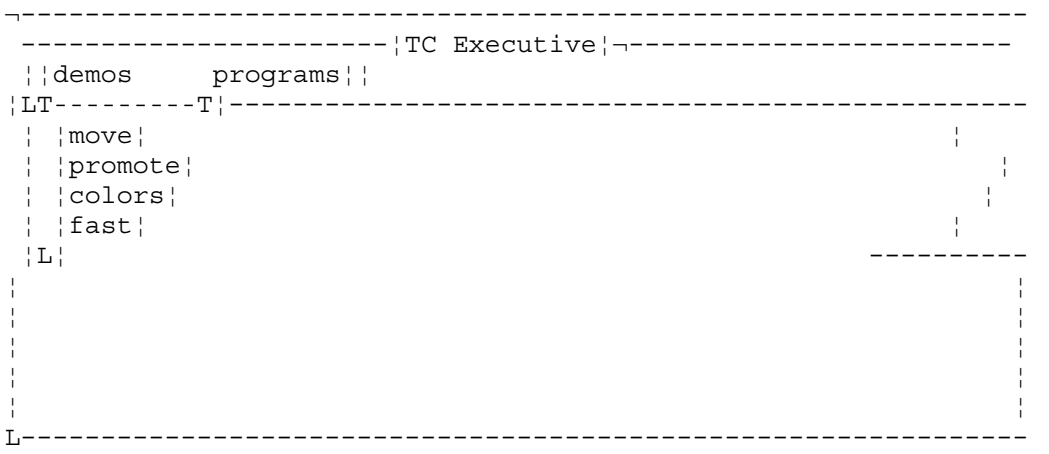


Рис. 10.3 "Всплывающее" вертикальное меню

Резюме

Итак, описание библиотеки функций оконной технологии завершено. Они позволяют поддерживать программное обеспечение на таком профессиональном уровне развития, для которого традиционно использовались возможности, предоставляемые фирмами-поставщиками библиотек оконной поддержки, изучение которых требовало много времени, да и цена их была побольше, чем стоимость этой книги. Версия описанного в этой книге пакета программ может быть использована для дальнейшего комплексного развития программных систем, после чего эти системы не требуют какой-то дополнительной перенастройки в связи с мобильностью программ, составляющих описываемый пакет.

В следующей главе будут рассмотрены резидентные утилиты. Функции поддержки окон, описанные в этой книге, могут быть включены в обычные однозадачные программы, выполняемые в среде DOS, или могут быть преобразованы в программы, которые постоянно находятся в памяти, и инициализация их выполнения закрепляется за определенными клавишами. Технические приемы написания таких программ (они еще носят название резидентных программ или резидентных утилит), а также примеры их использования при разработке систем рассматриваются в следующих главах.

Г Л А В А 11

Резидентные программы

Повсюду в этой книге "PC" (ПК) относится к семейству IBM PC и совместимым с ними компьютерам; однако персональные компьютеры не всегда были такими, как PC. Первые ПК создавались на знаниях и интересе любителей. ПК развивались вместе с развитием микрокомпьютерной технологии от домашних игрушек до серьезных систем. Программное обеспечение и операционные системы развивались вместе с компьютерами. Ранние операционные системы)ОС) представляли собой немногим более, чем командные процессоры Бейсика или Паскаля, которые обеспечивали поддержку языков и простое управление файлами. Эти системы обычно ориентировались на конкретный компьютер и были несовместимы с другими. Когда компьютеры начали ориентироваться на массовое применение, операционные системы начали стабилизироваться. Среди них были Apple-DOS, NorthStar DOS, TRSDOS, и CP/M. Все они имели похожие характеристики:

- однопрограммный однопользовательский режим работы
- поддержка файлового каталога
- поддержка интерпретаторов и компиляторов языков программирования
- несовместимость программ и данных между системами

С великой проницательностью Гарри Килдалл создал CP/M как открытую ОС. Изначально предназначавшаяся для поддержки создания программ для Intel MDS, эта ОС не ориентирована на какую-либо модель или марку компьютеров. CP/M состоит из базовой системы ввода-вывода (BIOS), базовой дисковой операционной системы)BDOS), и командного процессора (CCP). BIOS ориентирована на работу с аппаратурой компьютера и предназначена для управления консолью, принтером и дисковой системой. Путем написания соответствующей BIOS производители компьютеров могли использовать приемлемую и развитую операционную систему с тысячами доступных программ. Благодаря своей адаптируемости CP/M стала промышленным стандартом ОС для ПК на микропроцессорах 8080 или Z80.

Микропроцессоры 8080 и Z80 могли адресовать только 64К памяти, поэтому CP/M строилась как однопользовательская однопрограммная ОС.Была построена и многопользовательская версия, названная MP/M, но она не стала промышленным фаворитом, каким была CP/M, в основном из-за медленнной работы (8080 не очень быстрый процессор), и потому, что ПК не особенно подходят для многопользовательской работы.

ДОС, управляющая работой PC, является адаптацией ОС, созданной для использования на 8086, и заметно похожей на CP/M. Microsoft купила ее у создателя - фирмы Seattle Computer Products, а IBM купила лицензию у Microsoft. ДОС до сих пор во многом похожа на CP/M. ДОС состоит из трех модулей, схожих по функциям с BDOS, CCP и BIOS в CP/M; и пользовательский интерфейс почти идентичен используемому в CP/M. У ДОС есть дополнительные возможности, такие ,как каналы, фильтры, переназначение ввода-вывода, пометка файлов временем и датой, и иерархическая структура файлового каталога. ДОС по-прежнему однопользовательская однозадачная ОС, в чем совпадает с CP/M.

Первые компьютеры IBM PC были очень похожи на своих предшественников с микропроцессорами 8080 и Z80; у них было 64К памяти, флоппи-диски и процессор немного побыстрее, чем Z80. Однозадачная ДОС была подходящей и адекватной этим компьютерам,

но PC имели три архитектурные особенности, предназначенные для расширения. Микропроцессор 8088 адресует до 1M памяти; он имеет векторную систему прерываний; клавиатура и дисплей IBM PC являются составной частью компьютера, в отличие от видеотерминалов, присоединенных через последовательный порт. Эти характеристики легли в основу ограниченного вида мультизадачности, который развивался на PC и сейчас известен как программы, остающиеся в памяти (TSR.(

ДОС включает две функции, позволяющие программе объявить себя резидентной. Эти функции похожи, но имеют небольшие различия. Функция 0x31 прерывания ДОС 0x21 заканчивает выполнение программы, но оставляет ее резидентной. ДОС не будет покушаться на память, принадлежащую программе. Прерывание ДОС 0x27 делает то же самое, но ограничивает размер программы величиной 64K.

Эти две функции предназначались в ДОС не для написания резидентных утилит, а для того, чтобы создатели системы могли написать программы обработки прерываний (ISR), поддерживающие дополнительные устройства ввода-вывода, такие, как мышь, графический планшет, или джойстик. Эти устройства не являются стандартной частью PC и, соответственно, не имеют стандартного программного интерфейса с ДОС. ISR могут поддерживать и другой вид программ, не обязательно связанных с дополнительными устройствами, но расширяющих пользовательский интерфейс с компьютером. Это TSR-программы; два наиболее популярных вида таких программ - это расширители клавиатуры и программы-секретари. Расширители клавиатуры, такие, как Prokey и Superkey, позволяют пользователю присвоить значения символов функциональным клавишам, <Alt>-key комбинациям, или любым другим клавишам. Программы-секретари, такие, как Sidekick и Homebase, предлагают записную книжку, калькулятор, календарь, автовызывное телефонное устройство и другие возможности рабочего стола, вызываемые нажатием клавиши.

Среди других TSR-программ : программы-корректоры, системы обработки структуризованных текстов, печать со спулингом, расширения командного процессора ДОС, отладчики, часы с будильником. Эти программы и многочисленные представители других резидентных программ имеются в продаже, в источниках бесплатного программного обеспечения, или в исходных текстах, публикуемых в журналах.

Эта глава представляет и объясняет класс программ для PC, известных под разными именами, среди которых всплывающие)pop-up), TSR, резидентные утилиты, и программы-секретари. Такие программы уникальны, так как после исполнения они остаются резидентными в памяти и часто ее не покидают до выключения компьютера. Будучи резидентными, они выполняются (или "всплывают") при вызове.

Типичная TSR-программа вызывается внешним событием, обычно называемым "горячим ключом". Горячий ключ - это ключевая строка, формируемая при нажатии пользователем комбинации клавиш, зарезервированной для вызова утилиты. Естественно, эта комбинация не должна обычно использоваться для других целей.

Активизация TSR-программы прерывает выполнение текущей программы на время работы TSR-программы. После окончания TSR-программы прерванная программа продолжает работу. Прерванный процесс может быть нерезидентной программой, другой

TSR-программой, или самой ДОС.

Загрузка нескольких TSR-программ в память превращает ДОС- однозадачную по сути операционную систему - в ограниченную, в чем -то калечную мультизадачную ОС.

Прерывания

Чтобы понять сущность TSR-программ, вы должны понять систему прерываний, потому что эти программы используют структуру прерываний ДОС и PC. Это обсуждение ни в коей мере не является исчерпывающим описанием прерываний, и этим вы поощряетесь к исследованию предмета с использованием материалов, посвященных архитектуре 8086/80286 и PC. Здесь объясняется, что такое прерывания и как они используются, но без детальных спецификаций. Этого вам хватит, чтобы понять TSR-программы.

Прерывание - это кратковременное приостановка текущей прочедуры программы, позволяющая выполнить другую процедуру. После завершения прерывания прерванная программа продолжает выполняться так, как будто бы ничего не происходило. Эти две процедуры могут быть несвязанными - и прерывание не окажет никакого воздействия на прерванную процедуру. Они могут быть взаимозависимы - прерванная программа может быть модифицирована процедурой обработки прерывания. Прерывание может быть вызвано внешним по отношению к выполняемой программе событием или в результате действий самой программы. Прерывание может быть вызвано аппаратно или командой из программы.

Векторы прерывания

В компьютере PC имеется 256 различных прерываний, с номерами от 0 до 0xff. Некоторые из них определены для использования процессором. Например, прерывание 0 возникает при делении на 0. Другие определены для вызова функций BIOS, третьи - для использования ДОС. Напомним, что 8088/8086/80286 - это микропроцессор, PC - компьютер, построенный на его базе, а ДОС- это операционная система. Для каждого из этих трех архитектурных слоев определен свой набор прерываний. Оставшиеся прерывания доступны для использования прикладными программами и программами обслуживания устройств.

Каждое прерывание представлено в памяти четырехбайтным значением адреса. Эти значения располагаются в памяти со смещениями от 0 до 0x3ff. При прерывании содержимое регистра признаков и четырехбайтный адрес выполняемой команды сохраняется в стеке.После этого прерывания запрещаются, и начинает выполняться программа с адреса, соответствующего происшедшему прерыванию. Эта программа должна сохранить используемые ей регистры, выполнить свою задачу, восстановить значения регистров, и выполнить команду возврата из прерывания, которая восстанавливает адрес прерванной программы и регистр признаков, так что прерванная программа продолжит свое выполнение с того места, где была прервана.

Аппаратные прерывания

Аппаратные прерывания вызываются событиями, физически связанными в аппаратуре с соответствующими векторами прерываний. Например, клавиатура в PC связана с прерыванием 9. Нажатие клавиши вызывает прерывание выполняемой программы, как было описано выше, и переход по адресу, находящемуся в векторе прерывания, соответствующему прерыванию 9. В памяти этот вектор находится по адресу 0x24 (9*4 байт.(

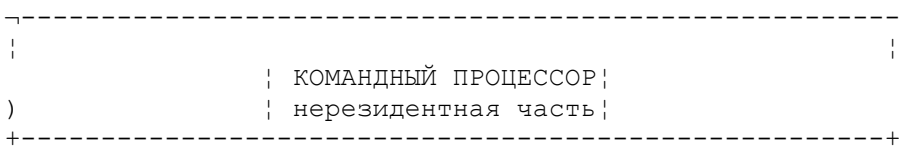
Программные прерывания

Программные прерывания происходят при выполнении в текущей программе команды INT с номером прерывания в качестве операнда. В остальном нет никакой разницы между программным и аппаратным прерыванием.

ДОС - однозадачная операционная система

Работу TSR-программ можно понять, изучая условия, в которых выполняются нерезидентные программы. ДОС была сделана для поддержки работы только одной задачи. ОС организует загрузку и выполнение задач и выполняет запросы на ввод-вывод. Она управляет дисковыми каталогами и файлами, работает с системными часами, выводит данные на печать, консоль, и возвращает программе символы, введенные с клавиатуры. ДОС - это в сущности сервер, обслуживающий иерархическую файловую систему и записе-ориентированные устройства, и обеспечивающий одному пользователю выполнение одной задачи. И с этой службой ДОС справляется.

После первоначальной загрузки ДОС память размером в 640K, имеющаяся в PC (или в размерах памяти вашего компьютера(распределена, как показано на рис.11.1. Память с адресами от 0 до 0x400 зарезервирована для векторов прерываний.За ними следует программа ДОС. Затем идут драйверы устройств,загруженные вместе с ДОС.Например, при использовании виртуального диска или драйвера терминала ANSI, программы этих драйверов располагаются после ДОС. После драйверов идет резидентная часть командного процессора. Эта программа обрабатывает командную строку и выполняет программы, и она разделена на резидентную и нерезидентную части. Нерезидентная область (Transient Program Area - TPA) находится после резидентной части командного процессора. Запущенная пользователем из командной строки программа загружается в TPA. В конце TPA находится область нерезидентной части командного процессора. Пользовательская программа может использовать эту область. В этом случае резидентная часть командного процессора подгружает нерезидентную после окончания программы.



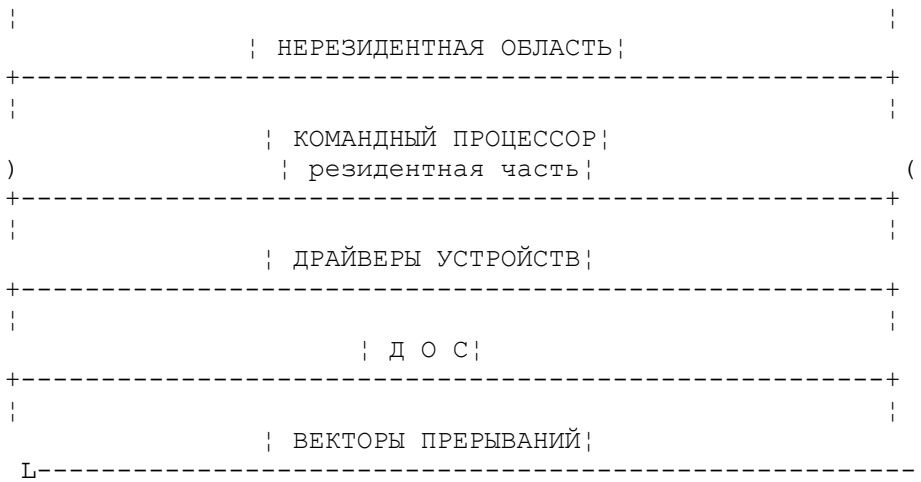


рис.11.1. Карта памяти ДОС.

После загрузки командным процессором программы она начинает выполняться. При необходимости обращения к ДОС, например, для выполнения операции с файлом, вызов ДОС осуществляется программным прерыванием с передачей параметров через регистры. В зависимости от параметров ДОС выполняет одну из своих функций. Вызвавшая ДОС программа находится в состоянии ожидания. Результаты функций ДОС возвращаются в регистрах и с помощью установки флажка переноса.

Такая последовательность событий описывает работу типичной однозадачной ОС. ДОС не обеспечивает одновременное нахождение в памяти и выполнение пользовательских программ, или поддержание информации о более, чем одной задаче в памяти. Единственный метод сделать несколько задач активными в памяти - оформить)несколько) задач как программы обработки прерываний. Для ДОС эти программы будут выглядеть как ISR-программы, поддерживающие выполнение единственной задачи. В ДОС задаче разрешается порождать подчиненную подзадачу, но только одна из них может быть активной в текущее время. Главная задача будет бездействовать, пока не завершится подзадача.

Для правильной работы ISR-программ должно быть соблюдено одно правило: избегать вызова функций ДОС, если ISR-программа может быть вызвана в результате прерывания работы самой ДОС. ISR-программы, вызываемые только из нерезидентных программ с помощью программных прерываний, не имеют таких ограничений, но асинхронные по отношению к текущей задаче ISR-программы)например, обработки прерываний от таймера или клавиатуры) не должны вызывать ДОС. Программисты узнали через некоторое время, что TSR-программы, будучи вызванными, не должны использовать функции ДОС. Если резидентная программа читает с клавиатуры с помощью функций ROM-BIOS (BIOS, "зашитый" в ПЗУ) и пишет на экран теми же функциями или путем прямого доступа к буферу экрана, она выполняется без проблем; но при попытке использования такой программой функции ДОС для чего-либо, система "ломается". Функции ДОС не реентерабельны; когда резидентная программа прерывает выполнение функции ДОС и сама вызывает функцию ДОС, система превращается в мусор. Казалось бы, что нереентерабельность программ ДОС ограничивает TSR-программы функциями, позволяющими

работать с памятью и использовать возможности ROM-BIOS. Это ограничение не приводит к большим жертвам. После того, как вы

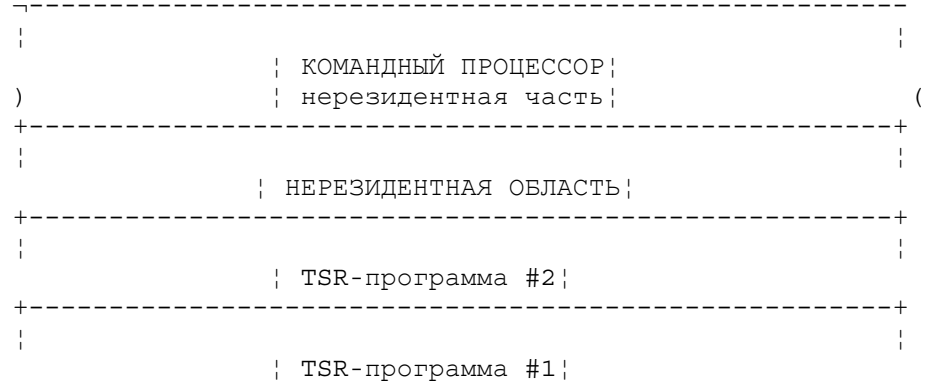
узнаете, как были созданы функции "окон", вы сможете прекрасно обойтись без ДОС.

К этой аномалии создатели ДОС добавили трюк, включив в состав ДОС программу печати со спулингом по имени PRINT.COM. Спулинг - это слово из прошлых времен, означающий "одновременное оперативное выполнение периферийных операций". Спулинг при печати позволяет печатать файл в то время, как пользователь выполняет на компьютере другую задачу, для которой не требуется принтер. PRINT.COM является резидентной программой, остающейся в памяти, поддерживающей очередь запросов на печать, и печатающей заданные файлы; одновременно с этим возможности компьютера и ДОС остаются доступными пользователю. Наличие программы, читающей дисковые файлы, читающей и записывающей в файл очереди на печать, и переводящей страницы на принтере, пока пользователь делает что-нибудь еще, позволяет предположить, что в ДОС реализуется некоторая мультизадачность без объяснения миру, как это делается.

Это предположение и пытливая натура поколения хэккеров, конечно, подталкивали к открытию возможностей ДОС по ограниченной мультизадачности .Некоторое время ключи не удавалось подобрать; затем те, кто понял технику написания функционально полных TSR-программ, хранили секрет, так они продавали эти программы, и, наверно, хотели задушить конкурентов. Но остальные, однако, начали "взламывать" чужие программы и открыли секрет. Сегодня аккуратный автор может скомбинировать шаги, необходимые для написания такой программы, прочитав массу технических журналов и сотни выдержек из электронных бюллетеней. В то время, как вы читаете эту книгу, возможно печатаются еще книги, подобные этой, чтобы помочь объяснить суть дела.

TSR-ПРОГРАММЫ

При работе ДОС и запуске из командной строки TSR-программы выполняются как нормальные нерезидентные программы. По сути, ни у ДОС, ни у командного процессора нет способов узнать, что эти программы станут резидентными, до того, как они завершатся с помощью одной из TSR-функций ДОС. При завершении программы информируют ДОС, сколько памяти надо зарезервировать. Насколько известно, в ДОС это действие приведет лишь к установке нижнего адреса нерезидентной области на значение сразу после конца TSR-программы и уменьшению размера доступной для нерезидентных программ памяти на размер TSR-программы. На рис.11.2 показана карта памяти системы при наличии двух TSR-программ.



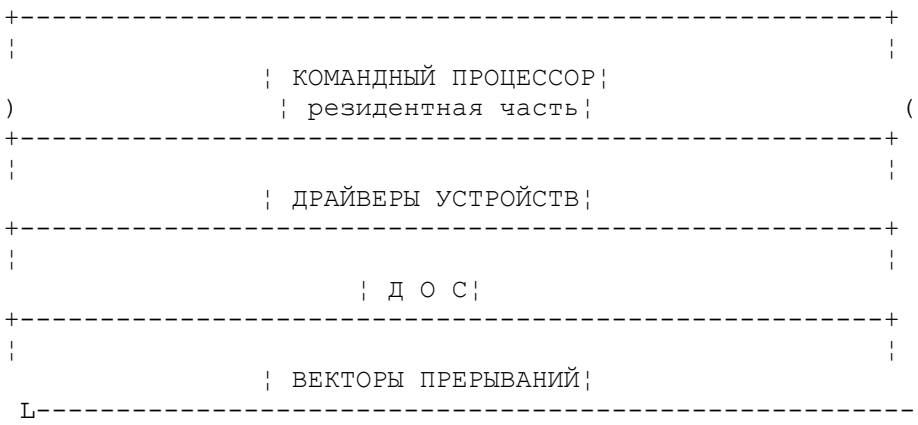


рис.11.2. Карта памяти ДОС с двумя TSR-программами.

TSR-программы существуют в двух вариантах: программы обработки прерываний и резидентные утилиты. Разница между ними незначительна, но она есть.

Программы обработки прерываний.

Программы обработки прерываний реагируют на прерывания от аппаратуры или от программ и обычно предназначены для поддержки различных устройств. Примером такой программы является программа, вызываемая прерыванием от таймера. Пример в главе 12 показывает, как можно использовать прерывание от системного таймера для отображения даты и времени на экран. Программа, поставляемая вместе с "мышью" фирмы Microsoft под названием MOUSE.COM, обрабатывает аппаратные прерывания, возникающие при перемещении "мышки". MOUSE.COM также обрабатывает программные прерывания от программ, которым требуется определить местонахождение "мышки" и состояние ее кнопок. Резидентные утилиты могут включать несколько таких программ.

Резидентные утилиты.

Резидентные утилиты - это программы обработки прерываний, обычно не поддерживающие какое-либо устройство, но реагирующие на нажатие определенной "горячего ключа" и запускающие процессы по запросу пользователя. Эти утилиты сохраняют состояние компьютера на момент своего вызова, и восстанавливают это состояние после окончания своей работы. Типичные резидентные утилиты используют технику "окон" для связи с пользователем.

Что может быть резидентным.

После того, как вы прочтете эту и следующую главу, вы научитесь писать TSR-программы на Турбо Си. У вас может возникнуть тенденция писать все в виде резидентных программ и вызывать все, что возможно, по нажатию клавиши, но себя надо сдерживать. Не все может или должно быть резидентным.

Помните, что после добавления резидентной программы в память

ее величина уменьшается. Вы можете дойти до того, что не останется памяти для выполнения обычных программ. Можно быть богатым на "всплывающие" утилиты, но не иметь памяти, чтобы сделать маленькую табличку или составить записку на своем текстовом редакторе.

Помните также, что вы увеличиваете встроенные ограничения ДОС, вводя первую резидентную программу в память. Не является сюрпризом, что TSR-программы из разных источников не уживаются. Далее, некоторые TSR-программы плохо работают с некоторыми нерезидентными программами.

Некоторые TSR-программы должны быть загруженными после всех других резидентных программ; они должны быть первыми в наборе программ, обрабатывающих прерывания. Несколько таких программ не могут работать вместе, так как только одна из них может быть последней. Программа Sidekick является примером такой программы; она должна быть последней загруженной, так как она перехватывает прерывания таймера из любой программы, загруженной после нее.

Такие несообразности являются результатом попыток вставить мультизадачность в ДОС. К чести создателей TSR-программ они старались прийти к соглашениям по стандартам таких программ. Они делали попытки сотрудничества и определения "хорошо ведущих" себя TSR-программ, но ни один такой стандарт до сих пор не опубликован. Также делались попытки определения такой структуры нерезидентных программ, которая позволяла бы им гармонично сосуществовать с "правильными" TSR-программами. Создатели программ для PC привыкли работать с однозадачной ДОС и ограничениями оборудования PC определенным образом. Чтобы преодолеть ограничения ДОС и получить требуемые характеристики, многие программисты "напрямую" сканировали клавиатуру и выводили данные прямо в видеопамять. Этот метод предполагал наличие только одной программы и отсутствие соперничества из-за этих ресурсов. Такие ограничения встают на дороге TSR-программ, которым нужно быть второй задачей, не мешающей первой.

Фирма Borland поставляет программу Sidekick Plus, управляющую работой TSR-программ. Она обеспечивает запуск и взаимодействие семейства TSR-программ, которые созданы с ее помощью. Преймущество такого решения - в достижении совместимости TSR-программ. Явным недостатком является необходимость иметь Sidekick Plus для запуска таких программ и то, что многие резидентные программы, созданные независимо, не будут работать в этой среде.

Чтобы решить, делать свою программу резидентной или нет, мы предлагаем использовать следующие правила:

- Размер программы. Если программа не может быть создана в крохотной модели памяти (64K на код, данные и стек), то вероятнее всего она не должна быть резидентной. Внушите себе это правило- и это убережет вас от засорения памяти резидентными утилитами за счет нужных вам обычных программ. Отметим, что TSR-драйвер, описанный в главе 12, одинаково хорошо работает с программами в крохотной и малой моделях.

- Частота использования утилиты. Не советуем вам делать резидентной программу расчета доходов. Программы, запускаемые с частотой один раз в день должны быть нерезидентными, а один раз в час - могут стать TSR. Не перегружайте систему редко

используемыми резидентными программами.

- Ресурсы, необходимые утилите. Если программа должна запрашивать дополнительную память или ее характеристики снижаются при ограничениях на память, она не должна быть резидентной; ведь она может быть вызвана во время выполнения любой другой программы, которая может занять всю или большую часть памяти. Если выполняется COM-программа, ДОС считает, что занята вся память.

" - Всплывающая" природа задачи. Если программа используется в дополнение к другим - это хороший кандидат в резидентные утилиты. Например, удобно ,когда можно вызвать систему компоновки текстов при работе с текстовым процессором. То же самое относится к программе-словарю. Калькулятор и записная книжка необходимы почти все время. Синтаксический анализатор программ на Си может стать полезной TSR-программой при использовании текстового редактора для подготовки программ. Но, однако, не нужно делать все программы вызываемыми по нажатию клавиши.

- Время, нужное для выполнения. Главное преймущество "всплывающих" программ - это их немедленная доступность и возможность выполнения без отрыва от основной работы, выполняемой на компьютере. Если же утилита требует для выполнения целый день, то нет смысла делать ее резидентной.

) Люди привыкают к удобствам. Еще несколько лет назад пользователи были довольны, если ответ на запрос к базе данных приходил из центральной машины за ночь. Сейчас же они ворчат, если им надо сохранить данные из электронной таблицы и возвратиться в ДОС ,чтобы использовать программу для работы с модемом(.

ПОСТРОЕНИЕ TSR-ПРОГРАММ

При написании резидентной программы вам придется решить много проблем. Некоторые из них незначительны, некоторые разрешаются при использовании расширений стандартных библиотек Турбо Си, но некоторые являются крепким орешком. Использование ассемблера для некоторых задач является более удобным, но в этой книге везде старались максимально использовать Си, где это только предоставлялось возможным.

Превращение программы в резидентную.

Чтобы стать резидентной, программа должна объявить себя соответствующим образом. Здесь программисту может помочь документация на ДОС. О двух нужных для этого функциях ДОС уже было упомянуто. Чтобы их использовать, надо знать размер программы, а для этого надо знать, как она строится в Турбо Си. Это в дальнейшем будет коротко объяснено. Если программа запущена на выполнение, присоединила себя к нужному вектору прерывания и сделала все, что требуется для превращения в резидентную, то она может вызвать одну из TSR-функций ДОС. Действие этих функций одинаково, и можно использовать функцию 0x31 прерывания 0x21 чтобы объявить программу резидентной. Далее следует фрагмент программы на Турбо Си, демонстрирующий эту возможность:

```
#include <dos.h<
static struct REGS rg;
unsigned int sizeprogram;

rg.x.ax = 0x3100;
rg.x.dx = sizeprogram;
intdos(&rg,&rg;(
```

Переменная **sizeprogram** должна содержать размер программы в шестнадцатибайтных параграфах.

Резидентна ли уже программа?

Помните, что ДОС неизвестно, что ваша программа стала резидентной и каково ее имя (эта информация доступна, но ДОС не работает с ней). После выполнения TSR-программы несколько раз несколько ее копий будет находиться в памяти, поэтому программе надо проверять, не загружена ли уже ее копия. Простейшим способом для организации такой проверки является применение одного из неиспользуемых прерываний. При первом старте TSR-программа выполняет это прерывание, проверяя возвращаемое значение. Если это значение не равно установленному в программе, то можно объявлять себя резидентной, и программа присоединяется к этому прерыванию. После этого вызов этого прерывания будет приводить к возврату определенного значения, и другие копии загруженной программы после проверки не будут объявлять себя резидентными.

Векторы 0x60-0x67 всегда доступны для использования, и можно выбрать один из них. Но нет уверенности, что другая программа, взятая из другого источника, не выберет тот же вектор. Помните, что ДОС ориентирована на одну выполняемую задачу, и этой задаче дозволено использовать любое прерывание в системе.

Более предпочтительным является использование вектора прерывания, указывающего на сигнатуру (уникальную запись) в памяти программы. Вместо запуска прерывания программа проверяет, не указывает ли один из векторов 0x60-0x67 на эту сигнатуру. Если таковой указатель существует, то в памяти уже есть копия программы; если же такого не находится, то программа объявляет себя резидентной и устанавливает адрес одного из свободных прерываний на сигнатуру. К несчастью, нет способов оградить этот вектор от посягательтв другой программы. Тут уж ни в чем нельзя быть уверенным.

Чтобы найти неиспользуемый вектор прерывания, надо проверить их все на нулевое значение. Вектор, в который записан ноль, и есть неиспользуемый. В документации указано, что прерывания 0x60-0x67 доступны для использования программами.

Неиспользуемые векторы прерывания можно применять для других видов связи между программами. Некоторые TSR-программы можно применять для задания параметров для своей же копии в памяти. При выполнении таких программ они передают новые параметры своим копиям через вектор прерывания. Вектор может указывать на подпрограмму обработки прерывания в TSR-программе; а сигнатура находиться с определенным смещением с том же сегменте. Найдя сигнатуру, вторая копия программы может взаимодействовать с первой через этот вектор. TSR-драйвер, описанный в главе 12,

демонстрирует такой способ.

Захват прерывания.

Вы можете использовать прерывания не только для связи между программами. По прерыванию может выполняться ваша программа, если на нее указывает соответствующий вектор. В Турбо Си легко можно выполнить подобный захват прерывания. С помощью функции setvect можно оперативно выполнить эту задачу. Надо объявить функцию типа interrupt, которая будет обрабатывать прерывания, и записать ее адрес в нужный вектор.

Например:

```
setvect(vno,isr;(
```

Параметр vno - это int от 0 до 255, а isr - это адрес подпрограммы обработки прерывания, которая может быть описана так:

```
void interrupt isr;()
```

Совместное использование прерываний.

Если вы используете прерывания, нужные другим программам, то надо делать это так, чтобы другие не замечали ваших действий. Например, если ваша программа присоединится к прерыванию от таймера и не позволит оставшейся части системы использовать это прерывание, все процессы, работающие по таймеру, будут недоступны все то время, пока ваша программа будет находиться в памяти и удерживать прерывание. В этом случае, например, остановятся системные часы.

Вы можете присоединиться также к прерываниям от клавиатуры, прерываниям для вызова функций ДОС, дисковых функций BIOS и другим прерываниям поддержки TSR-программ. В любом из этих случаев надо совместно использовать прерывания с другими процессами, нуждающимися в них.

Возможность работать с прерыванием другим программам выполняется следующим образом. Надо прочитать старый адрес прерывания, по которому передавалось управление до загрузки вашей программы. В библиотеке Турбо Си есть функция getvect для чтения содержимого вектора прерывания. Надо объявить interrupt указатель на функцию, и записать адрес из вектора в этот указатель:

```
void interrupt (*oldisr;()(

oldisr = getvect(vno;(
```

Затем, запишите адрес вашей подпрограммы обработки прерывания в вектор, используя функцию setvect, описанную выше.

В этой подпрограмме вы можете обеспечить выполнение старой ISR, после предварительной обработки или без нее передавая управление по адресу старой ISR. Эти принципы обсуждаются в главе .12

Величина TSR-программы.

При объявлении программы резидентной надо специфицировать ее размер, чтобы ДОС было известно, сколько памяти отвести для нее. Для безопасности можно отвести каждой программе по 64К, но при этом значительная часть занятой памяти может не использоваться. Легко определить размеры программы, написанной на ассемблере, но внутренняя структура программы на Си скрыта от программиста. Чтобы вычислить размер программы, надо знать, как компилятор Турбо Си строит программу.

Рисунок 11.3 демонстрирует структуру типичной программы в крохотной модели памяти, построенную Турбо Си. Program Segment Prefix (PSP) - это конструкция ДОС, помещаемая в начале каждой программы. Она будет описана позже. Машинный код программы следует сразу за PSP. Переменные,объявленные static или external, и инициализированные при объявлении, следуют за кодом, а за ними идут неинициализированные static или external переменные. Следующая часть программы - "куча", область динамически распределяемой памяти. Ее размер зависит от того, сколько программа будет распределять памяти. Область стека располагается

за "кучей", и его размеры увеличиваются в обратном направлении. Первоначально вершина стека указывает на конец области в 64К, занимаемой программой в крохотной модели. При использовании стека его указатель двигается в сторону уменьшения адресов. Величина стека зависит от глубины вложенности функций и количества локальных данных, используемых в этих функциях. При вызове функции в стек помещаются параметры и значения регистров, которые необходимо сохранить. Если функция использует локальные automatic-переменные, они тоже запоминаются в стеке. Функция, вызываемая рекурсивно и использующая большое количество параметров и automatic-переменных, будет использовать большую часть стека. Из-за динамической природы "кучи" и стека вы можете только делать некоторые предположения о размерах программы.

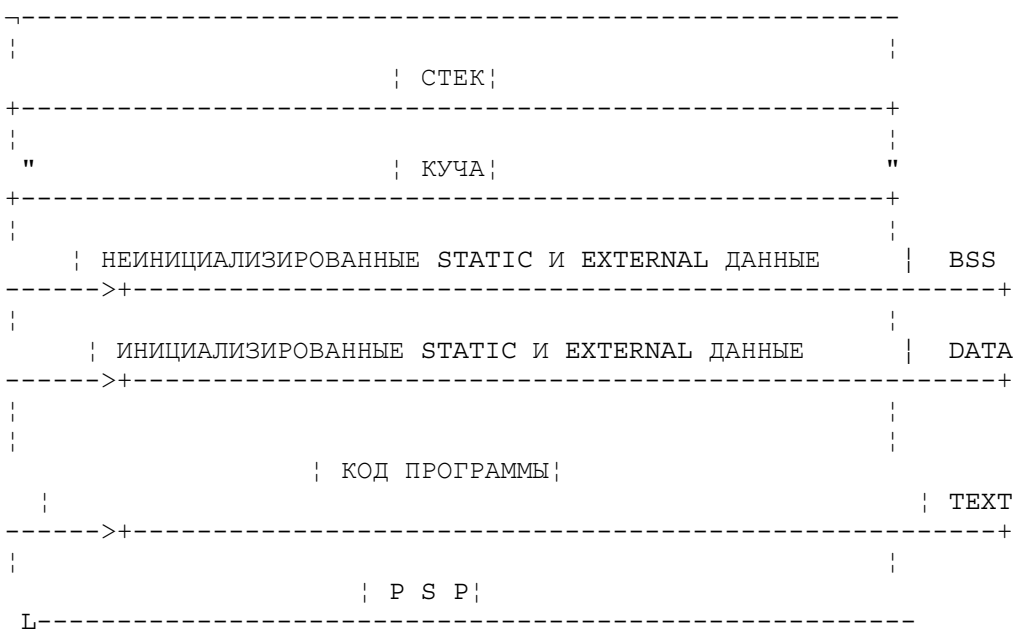


рис.11.3. Структура программ с крохотной моделью памяти

в Турбо Си.

Приблизительно подсчитать размер программы можно, используя MAP-файл, генерируемый программой TLINK. В его начале находится информация, подобная следующей:

```
     Start  Stop   Length Name               Class

00000     H 010BAH 010BBH _TEXT              CODE
010      C0H 013D8H 00319H _DATA             DATA
013      DAH 013DDH 00004H _EMUSEG           DATA
013      DEH 013DFH 00002H _CVTSEG           DATA
013      E0H 013E5H 00006H _SCNSEG           DATA
013      E6H 014EDH 00108H _BSS              BSS
014      EEH 014EEH 00000H _BSSEND           BSSEND
```

Самая правая колонка, с заголовком class, показывает тип сегментов, представленных значениями в левых колонках. CODE содержит код программы, DATA содержит инициализированные переменные, а BSS - неинициализированные. Значение в колонке Stop равно шестнадцатиричному адресу конца области неинициализированных переменных и начала "кучи". Программа, не использующая стека и "кучи", будет иметь размер, равный этому значению + 256 байт на PSP.

Чтобы оценить размер "кучи", посмотрите, сколько и как вы используете функции распределения памяти. При использовании оконных функций из предыдущих глав требования к "куче" можно оценить исходя из количества и размеров одновременно существующих окон. Каждое окно требует буфера размером с удвоенное произведение ширины на высоту окна. Этот буфер удаляется из "кучи" при закрытии окна, поэтому максимальное использование "кучи" будет в момент создания максимального количества окон на экране. Надо учитывать также и использование вами функций распределения памяти. Если ваша программа распределяет память в зависимости от внешних условий, таких, как ввод пользователя или зависимости между данными, необходимы надежные обнаружение и обработка ошибок. Не взирая на ошибку, никогда не вызывайте функцию exit из резидентной программы.

Вместе с определением размеров "кучи" определяется и ее верхняя граница, а, следовательно, и нижняя граница стека. Осталось теперь найти его верхнюю границу. При первом выполнении программы начало стека устанавливается на отметку 64К. При завершения и объявления себя резидентной программа сообщает ДОС свои размеры, и ДОС использует всю остальную память для загрузки других программ. При вызове TSR-программа должна установить указатель стека внутри себя, а не на другую программу. Если размер объявлен меньше 64К, то указатель стека должен быть также установлен ниже 64К.

Лучший метод найти оптимальный размер стека - метод проб и ошибок. Сначала запустите программу на 64К, а затем смещайте вершину стека к меньшим адресам. При каждом таком передвижении испытывайте программу в условиях максимального использования стека и "кучи". Продолжайте эксперименты до тех пор, пока ваша программа не будет "вешать" систему или неправильно выполняться. Затем поставьте вершину стека на безопасное смещение и интенсивно используйте программу, пока не поверите наконец, что стек безопасен для "кучи."

Было бы хорошо, если бы имелся более научный и точный способ

определения размеров программы, но этот подход работает, и кажется, что это единственно реальный метод.

Переключение контекстов.

При первом исполнении TSR-программы она использует все ресурсы, предоставляемые ДОС нормальной задаче. После завершения и объявления себя резидентной эти ресурсы отдаются другим программам или, при отсутствии выполняемых программ, командному процессору ДОС. При выполнении TSR-программы в результате "горячего ключа" она "паразитирует" на прерванной программе. ДОС неизвестно, что начала выполняться другая задача, и все ресурсы по-прежнему принадлежат выполнявшейся раннее задаче. Поэтому системные указатели на эти ресурсы должны быть изменены так, чтобы TSR-программа стала выполняемой задачей, "известной" ДОС. Такая передача ресурсов между задачами называется переключением контекстов, и мультизадачные ДОС делают это автоматически. В однозадачной ДОС PC однако, переключения контекстов не производится, и прерывающая задача должна делать это сама.

Стек.

Для всех программ нужен стек. У резидентной программы есть свой стек, но после прерывания текущей задачи указатель стека и сегмент стека в компьютере указывают на стек прерванной задачи. Может показаться, что лучшее решение - это использовать стек прерванной программы. На деле многие ассемблерные TSR-программы так и делают, но для этого приходится ограничивать использование стека. Но, во-первых, неизвестно, какой размер стека был у прерванной программы. А во-вторых, ДОС гарантирует достаточный размер стека только для сохранения регистров. Си-язык с интенсивным использованием стека, и вам понадобится больший его размер, чем обеспечивает ДОС, и это значит, что надо переключаться на собственный стек.

Переключение на собственный стек означает, что надо запомнить значение регистра сегмента стека до переключения. И это значение должно быть восстановлено до передачи управления в прерванную программу. Регистры сегментов и указателей могут быть прямо адресованы в Турбо Си использованием псевдопеременных _SS и _SP. При объявлении резидентной TSR-программа запоминает собственный сегмент стека. После вызова она запоминает контекст стека прерванной программы и устанавливает свой стек. Это производится с помощью установки регистра сегмента стека на значение, запомненное при первом запуске, и указателя стека на величину, вычисленную из размеров программы.

Если TSR-программа реентерабельна, то есть может прерывать сама себя, переключение стеков может привести к ошибке. При втором переключении стека вы перезапишете область сохранения стековых регистров. Чтобы избежать этого, надо писать нереентерабельные резидентные программы. Это небольшая потеря-резидентные программы не нуждаются в том, чтобы быть реентерабельными (вам не нужно прерывать свою программу-калькулятор, чтобы запустить еще одну такую же.(

Чтобы сделать TSR-программу нереентерабельной, устанавливайте флаг при ее вызове. Он должен оставаться установленным то окончания работы TSR-программы. При повторном вызове (например, из-за случайного нажатия "горячего ключа(" проверяется установка флага и вторичного запуска не производится.

Program Segment Prefix (PSP.(

PSP - это управляющая область в 256 байт, которая строится в памяти в начале каждой программы. Она содержит различные поля, используемые ДОС для управления выполнением программы. На рис. 11.4показана ее структура. Далее будут обсуждаться все поля PSP. Заметим, что многие эти поля не были официально описаны фирмами Microsoft или IBM. Они используются так, как описано ниже, но их использование или модификация в прикладной задаче не санкционировано при продаже. Знание этих полей - подарок от хэккеров, расчленивших ДОС и напечатавших о своих находках. Эти данные верны для популярных версий ДОС - 2.0, 2.1, 3.0, 3.1, 3.2, ,3.3за исключением специально оговоренных случаев. ДОС 4.0 не публиковалась в США, и, как утверждается, будущие версии ДОС будут поддерживать мультизадачность и будут предназначены только для компьютеров с процессорами 80286/80386. Использование полей PSP описанным способом совершенно безопасно. Многие популярные коммерческие программы делают это точно так же.

Вызов прерывания для завершения процесса	0000
Сегментный адрес верхней границы памяти	0002
0	0004
Команда вызова диспетчера функций ДОС	0005
Адрес обработчика завершения	000A
Адрес обработчика Ctrl-Break	000E
Адрес обработчика критических ошибок	0012
Сегментный адрес PSP родителя	0016
Таблица указателей файлов	0018
Сегментный адрес области системных параметров	002C
Адрес стека на время вызова функции ДОС	002E

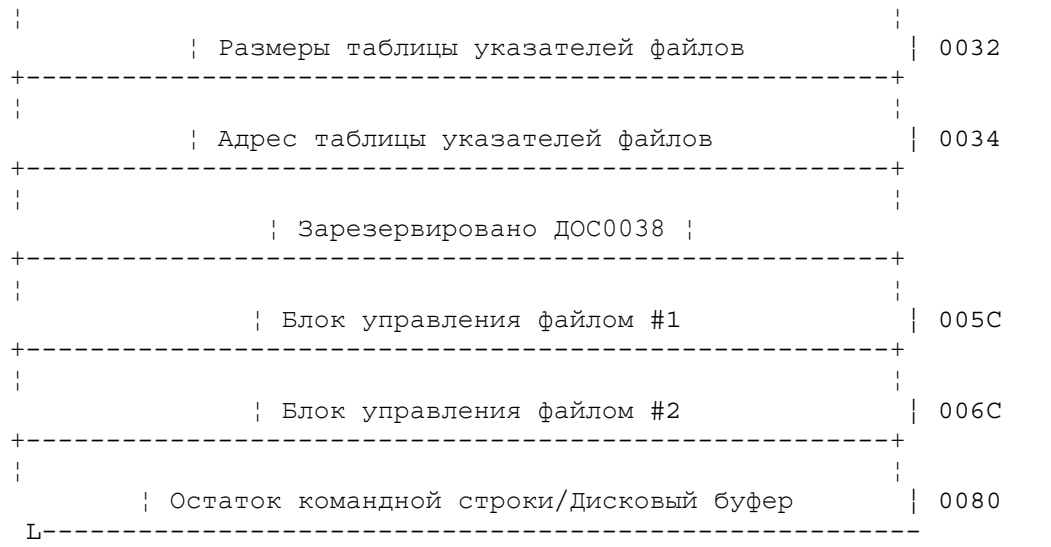


рис.11.3. Структура PSP.

Вызов прерывания для завершения процесса.(PSP:0(

Это поле содержит команду INT 0x20; это сделано для поддержки программ, перенесенных из CP/M в ДОС. В CP/M программа завершает свое выполнение переходом на свой нулевой адрес.

Сегментный адрес верхней границы памяти.(PSP:2(

При выполнении программы ДОС выделяет ей участок памяти, в который программа загружается. Это поле содержит сегментный адрес конца этого участка памяти.

Адрес обработчика завершения.(PSP:0xa(

При выполнении программы ДОС запоминает текущее содержание вектора прерывания 0x22 в этом поле. После завершения программы ДОС восстанавливает значение, используя это поле. Вектор прерывания 0x22 указывает на системный обработчик завершения программ.

Адрес обработчика Ctrl-Break.(PSP:0xe(

При выполнении программы ДОС запоминает текущее содержание вектора прерывания 0x23 в этом поле. После завершения программы ДОС восстанавливает значение, используя это поле. Вектор прерывания 0x23 указывает на системный обработчик Ctrl-Break.

Адрес обработчика критических ошибок.(PSP:0x12(

При выполнении программы ДОС запоминает текущее содержание вектора прерывания 0x24 в этом поле. После завершения программы

ДОС восстанавливает значение, используя это поле. Вектор прерывания 0x24 указывает на системный обработчик критических ошибок.

Отметим, что ДОС восстанавливает значения этих трех векторов при завершении программы и объявлении себя резидентной. Если TSR надо перехватывать эти прерывания, то придется присоединять себя к ним при каждом вызове.

Сегментный адрес PSP родителя.(PSP:0x16(

Любая программа выполняется в результате обращения другой программы к ДОС. Обычно программой-отцом является командный процессор ДОС (COMMAND.COM), хотя любая программа может быть родителем любой другой. Это поле в PSP содержит сегментный адрес PSP программы-отца.

У командного процессора нет "папаши", поэтому это поле в PSP командного процессора содержит сегментный адрес собственного PSP, что является указателем на самого себя.

Таблица указателей файлов.(PSP:0x18(

Это поле представляет собой массив в 20 байт,каждый из которых представляет указатель файла(file handler). При открытии файла в программе ДОС возвращает в программу номер файла для использования при обращении к ДОС для записи в файл и чтения из него. (Программы на Си, использующие потоковый ввод-вывод, прямо не обращаются к этим номерам - они используют указатель FILE, определенный в stdio; однако стандартные библиотечные функции Си, поддерживающие такой ввод-вывод, используют эти номера скрытым от вызывающей программы способом.) Эти номера файлов - номера байт в таблице указателей файлов, а элементы этой таблицы хранят значения, указывающие на соответствующую файлу структуру в системных таблицах управления файлами.

Сегментный адрес области системных параметров.(PSP:0x2c(

Это поле содержит сегментный адрес области системных параметров, создаваемой ДОС для выполняемой программы. Область системных параметров - это выделенный задаче участок памяти, который может быть освобожден, если значения параметров не используются.

Адрес стека на время вызова функции ДОС.(PSP:0x2e(

В это поле ДОС записывает значения регистров сегмента стека и указателя стека текущей программы при вызове из программы функции ДОС. Затем ДОС переключается на свой собственный стек. Перед возвратом в вызвавшую программу ДОС использует эти значения для восстановления значения этих регистров.

Размеры таблицы указателей файлов.(PSP:0x32(

Это поле содержит счетчик количества вхождений в таблицу указателей файлов. Обычно его значение равно 20. Заметим, что это поле не используется в версиях ДОС до 3.0.

Адрес таблицы указателей файлов.(PSP:0x32(

Это поле содержит полный (long) адрес таблицы указателей файлов; сегмент этого адреса равен сегменту PSP; а смещение- 0x18. Заметим, что это поле не используется в версиях ДОС до 3.0.

Вероятно, предыдущие два поля добавлены, чтобы позволить программе использовать более, чем 20 одновременно открытых файлов. После выделения большей таблицы надо поместить ее адрес и количество файлов в эти поля, скопировать в нее 20 значений из PSP, и таким образом программа может увеличить количество одновременно открытых файлов по сравнению со значением из оператора FILES=файла CONFIG.SYS.

Блок управления файлом #1.(PSP:0x5c(

Это поле содержит блок управления файлом, который строится ДОС в случае, если в командной строке в качестве первого параметра было указано имя файла.

Блок управления файлом #2.(PSP:0x6c(

Это поле содержит блок управления файлом, который строится ДОС в случае, если в командной строке в качестве второго параметра было указано имя файла.

Предыдущие два поля созданы для поддержки программ, конвертированных из CP/M.

Остаток командной строки/Дисковый буфер.(PSP:80(

Последнее поле также создано под влиянием CP/M. После запуска программы все, что находилось в командной строке, начиная со второго символа после имени программы, строится в виде строки слов и записывается в это поле. Лишние пробелы удаляются. В первый байт поля записывается количество символов в этой строке.

Во время выполнения программы это поле служит дисковым буфером для работы с файлами при помощи старых функций ДОС, использующих блоки управления файлами. Это поле также используется функциями ДОС, работающими с дисковыми каталогами.

Контекстное переключение PSP.

У каждой программы есть свой PSP. Но ДОС известен только

один PSP - находящийся перед программой, запущенной последней. Программа в ДОС может порождать выполнение другой программы, и программы-дети могут наследовать значения из PSP "папаши", но, так как ДОС знает только об одной активной задаче, то и только об одном PSP.

PSP содержит несколько интересных полей, но самое интересное для дальнейшего обсуждения - это массив из 20 указателей файлов. Программа может открыть до 20 файлов одновременно. С каждым файлом ассоциируется указатель, являющийся элементом массива в PSP. В этом массиве есть место для 20 указателей, и первые пять из них отданы для логических устройств stdin, stdout, stderr, stdaux, stdprn. При открытии файлов новые указатели записываются в массив, неиспользуемые позиции массива имеют значение -1. К файлам программа обращается по номерам указателя в массиве, а в элементе массива с таким номером хранится ссылка на соответствующую структуру в системных таблицах управления файлами.

При открытии файла в TSR-программе его указатель записывается в PSP этой программы. При вызове TSR-программы ДОС считает, что выполняется по-прежнему прерванная программа. При обращении к ДОС для работы с файлом, открытым при первоначальном запуске TSR-программы, ДОС будет брать указатель файла с данным номером из PSP прерванной программы. Поэтому возможно или обращение к чужому файлу, если файл с таким номером открыт и в прерванной программе, или ошибка по обращению к неоткрытому файлу.

Разрешить эту проблему можно следующим образом:

- не открывать файлов во время инициализации TSR-программы; открывать их после вызова и прерывания другой программы. Тогда для файлов TSR-программы будут использоваться указатели в PSP прерванной программы.

- не использовать работу с указателями файлов ДОС 2.0 и выше. Пользоваться FCB(File Control Block) из ДОС 1.1. Эти функции не используют PSP. Таблицы FCB строятся в области данных программы, использующих файлы.

- переключить системный указатель PSP при вызове TSR-программы на ее PSP, и после отработки вернуть все на свои места.

Каждое из этих решений имеет свои недостатки:

- При использовании таблицы указателей файлов из PSP прерванной программы имеются два серьезных недостатка. Во-первых, может оказаться неприемлемым открывать и закрывать файлы при каждом вызове TSR-программы. Во-вторых, нельзя быть уверенным, что в таблице прерванной задачи достаточно места для указателей файлов TSR-программы.

- Использование функций с FCB имеет два недостатка. Во-первых, файлы, открываемые с помощью FCB, должны быть в текущем подкаталоге. Пути для спецификации файла указывать нельзя. Во-вторых, все стандартные библиотечные функции Турбо Си для работы с файлами предполагают использование более гибких указателей. Для использования FCB вы должны создать функции, эквивалентные стандартным open, close, read, write, fclose, fget,

fput, fprintf и т.д.

- Не имеется документированной функции ДОС для изменения адреса PSP. Есть документированная функция для чтения текущего PSP (INT 0x21, функция 0x62), но она доступна только в ДОС версии 3.0и выше. Две функции ДОС (0x50 и 0x51), не описанные в документации, устанавливают и записывают адрес PSP, но они недоступны для использования резидентными программами в ДОС 2.0 и .2.1Они разделяют стек с некоторыми функциями ДОС, которые могут быть прерваны резидентной программой. Если TSR-программа использует функции 0x50 и 0x51 в этом случае, система зависнет. Эти функции можно использовать только с ДОС версий 3.0 и выше. Адрес PSP называется идентификатором процесса (Process ID -PID). Две скрытые функции ДОС носят название GetPID (получить идентификатор процесса) и SetPID (установить идентификатор процесса.(

Ни одно из этих решений не является удовлетворительным. Необходимы работающие заменители функций GetPID и SetPID. Одно из решений - экспериментально определить адрес, куда ДОС записывает PID. Этот адрес, конечно, зависит от версии ДОС и может быть уникальным даже при использовании одной версии ДОС. Использование таких значений является опасным решением.

Лучшим является определение адреса PID оперативно во время выполнения TSR-программы. Найти адреса, куда ДОС записывает PID, можно, используя следующую процедуру. Сначала, текущий PID восстанавливается с использованием функции GetPID. Затем по памяти, занимаемой ДОС, осуществляется поиск копии значения PID. Как только такое значение находится, его адрес запоминается и PID изменяется на какое-либо значение с помощью функции SetPID. Значение по найденному адресу анализируется на нахождение там нового значения. Если это так, то обнаружен адрес, куда в ДОС записывается PID, и он запоминается. Первоначальное значение PID восстанавливается, и поиск продолжается. Версии ДОС до 3.0 сохраняют PID в двух местах. Версии от 3.0 и выше поддерживают одно значение PID. Адреса PID и PID самой TSR-программы записываются до завершения и обьявления себя резидентной. При вызове TSR-программы один из сохраненных адресов PID используется для чтения PID прерванной программы. Это значение запоминается, устанавливается значение PID резидентной программы. Перед завершением резидентной программы она восстанавливает в системе PID прерванной программы.

Этот способ используется в функциях резидентного драйвера в главе 12.

Дисковый буфер.

Вторая часть PSP - это дисковый буфер, область, через которую ДОС пишет и читает дисковые файлы, открытые для использования с FCB. ДОС также использует эту область для функций, работающих с подкаталогами.

В библиотеке Турбо Си есть функции, читающие и устанавливающие адрес дискового буфера. Getdta возвращает адрес текущего дискового буфера, и setdta изменяет адрес. Эти функции имеют следующий формат:

```
#include <dos.h<
char far *dta;

dta = getdta;()

setdta(dta;(
```

Если TSR-программа будет использовать функции ДОС, которые пишут в дисковый буфер, она должна сохранить адрес дискового буфера прерванной программы, установить свой, и перед завершением работы восстановить предыдущий адрес.

Если TSR-программа будет писаться на Си, нельзя быть уверенным, что она не изменяет или не использует дисковый буфер. Действия функций из библиотеки Си достаточно не изучены, и лучшим решением будет самое осторожное. Резидентный драйвер, описанный в главе 12, сохраняет дисковый буфер прерванной программы.

Прерывание от клавиатуры (9.(

При нажатии пользователем "горячего ключа" резидентная программа должна прервать все, что бы ни делала система, и запустить себя. Чтобы сделать это, программа должна сканировать клавиатуру в поисках "горячего ключа". Понимание этого процесса требует понимания, как работает прерывание от клавиатуры и сканирование кодов.

Нажатие клавиши генерирует прерывание 9, и обработка передается программе по адресу вектора прерывания 9. Она должна прочитать порт данных клавиатуры и обработать это значение. Клавиатура PC генерирует код сканирования, который может быть прочитан из входного порта клавиатуры, а не значение ASCII, поступающее в вашу программу при использовании функции get_char. Каждая клавиша на клавиатуре имеет свое собственное значение. Программа должна определить по нему, какая клавиша или их комбинация нажата, и что делать с ними.

Программа обработки прерывания от клавиатуры, находящаяся в ROM-BIOS (базовой системе ввода-вывода в ПЗУ), читает коды сканирования, переводит их в коды ASCII, и пересылает их в буфер клавиатуры. Программы (включая ДОС), читающие с клавиатуры, читают коды из буфера. ROM-BIOS поддерживает байт состояния, показывающий, были ли нажаты клавиши Alt, правый Shift, левый Shift, и Ctrl. Программа может прочитать эти значения, восстановив у себя байт состояния. В главе 12 иллюстрируются коды сканирования и значения байта состояния клавиатуры.

TSR-программа может присоединить себя к прерыванию 9. Подпрограмма обработки прерывания в резидентной программе может поддерживать выполнение программы из ROM-BIOS, читая коды сканирования и байт состояния, и проверяя, не нажат ли "горячий ключ". Если нет, то обработка дальше предоставляется программе из ROM-BIOS.

Если в TSR-программе нет вызовов функций ДОС, подпрограмма обработки прерывания 9 может непосредственно выполнить свою функцию. Если же необходимо использовать функции ДОС, подпрограмма должна выставить флаг присутствия "горячего ключа" и

закончить работу. Другие подпрограммы обработки прерывания должны заметить этот флаг и выполниться, если это безопасно. Одна из подпрограмм, ждущая подходящего момента для выполнения функции- , это подпрограмма обработки прерывания от таймера.

Прерывание от таймера.

Работа PC прерывается 18.2 раза в секунду системным таймером, который использует вектор прерывания 0x1c для запуска подпрограммы обработки. В главу 12 включен пример резидентной программы "часы", использующей прерывание от таймера. TSR-программы, вызываемые по "горячему ключу", также должны использовать это прерывание, поэтому надо прикрепиться и к вектору 0x1c. Так как программа не может выполняться, пока не убедится, что использование функций ДОС безопасно, то надо проверить, установлен ли флаг "горячего ключа", как описано выше, и затем проверить безопасность. Если эти два условия выполнены, то подпрограмма обработки прерывания от таймера может запускать основную резидентную программу.

Как и в случае с прерыванием от клавиатуры, должна быть реализована возможность работы с прерыванием от таймера программе, которая до этого была обработчиком такого прерывания.

Проблема реентерабельности ДОС.

Вы уже читали о проблемах, возникающих при использовании функций ДОС из резидентных программ.В случае, когда программа прерывает ДОС и хочет использовать ее функции, ДОС не является реентерабельной. Может случиться так, что при нажатии "горячего ключа" ДОС будет находиться в состоянии ввода с клавиатуры, и пользователю не удастся быстро вызвать TSR-программу, так как до окончания ввода она не может начать выполнение.

Два стека ДОС.

ДОС поддерживает два стека.Когда ДОС выполняет одну из функций с номером от 0 до 12, используется первый стек; при выполнении других функций используется второй. ДОС сохраняет значение сегментного регистра стека и регистра смещения стека для каждой группы функций отдельно; но реентерабельных функций, однако, нет. Сохраненные значение регистров при первом вызове функции могут быть затерты при следующем вызове.

ДОС лучше определить как полуреентерабельную. При прерывании выполнения функций 0-12 безопасно вызывать остальные функции, и наоборот.

Можно легко избежать использования функций 0-12. Они предназначены для организации консольного ввода и вывода. Но есть много других способов, чтобы управлять клавиатурой и экраном, и частенько они работают лучше, чем функции ДОС. Использования остальных функций не так-то легко избежать. С их помощью можно делать многое из того, что нужно в резидентной программе, включая управление файлами.

Если вы можете избежать использование функций 0-12, но нуждаетель в остальных функциях, то ситуация с двумя стеками работает на вас. При уверенности, что прерывания по вызову вашей программы будут происходить не во время работы функций второй группы, проблемы реентерабельности не существует. Задачу можно решить установкой флажка ввода "горячего ключа" и ожиданием окончания работы небезопасных функций ДОС, если таковая имеет место. Эти функции работают быстро, так что большой задержки не получится.

Теперь остальсь узнать, каким способом получить информацию о возможности использовать ДОС. Такую информацию содержит системный флажок занятости.

Системный флажок занятости (0x34.(

Функция ДОС 0x34 возвращает сегментный адрес и смещение специального флага в памяти, поддерживаемого ДОС. Он называется системным флажком занятости. Этот флаг устанавливается, когда ДОС выполняет одну из своих "небезопасных" функций. После выхода из функции ДОС очищает флаг.

При первом запуске TSR-программы она использует функцию ДОС 0x34 для нахождения и запоминания адреса системного флажка занятости. При каждом прерывании от системного таймера обработчик, находящийся в TSR-программе, проверяет наличие флажка "горячего ключа", устанавливаемого обработчиком прерывания от клавиатуры, и отсутствие системного флажка занятости. При наличии этих двух условий обработчик таймера сбрасывает флажок "горячего ключа" и запускает основную часть TSR-программы.

Такая процедура временами срабатывает. Но временами ДОС остается занятой, пока пользователь занят чем-то другим, например, набирает символы в командной строке. Если при этом он нажмет "горячий ключ", то обработчик клавиатуры установит флаг "горячего ключа", но обработчик таймера никогда не запустит TSR-программу. Чтобы бороться в таких условиях, надо использовать вектор прерывания DOSOK.

Прерывание DOSOK.

Часто наступают периоды, когда ДОС занята, но можно использовать вторую группу функций; например, когда ДОС ожидает ввода строки символов. В этом случае ДОС вызывает прерывание 0x28 -DOSOK. Цель этого - известить TSR-программу (в частности- системный спулер PRINT.COM), о том, что можно использовать функции второй группы. Если никакая программа не прикреплена к этому прерыванию, оно просто указывает на команду IRET. Прерывание 0x28 - программное прерывание; обработчик, прикрепленный к нему, будет вызываться из другой программы по команде INT 0x28.

TSR-программы могут прикрепляться к прерыванию DOSOK и использовать его для обнаружения занятости ДОС. Обработчик этого прерывания из TSR-программы проверяет установку флажка "горячего

ключа" и в зависимости от его значения либо запускает основную часть TSR-программы; либо возвращает управление программе, вызвавшей это прерывание, или программе, ранее прикрепленной к этому прерыванию.

Прерывание DOSOK вызывается в случае, когда ДОС занята; однако TSR-программа, вызванная в это время, не может быть прервана другой такой же программой до тех пор, пока выполняющаяся TSR-программа не сделает вызов прерывания 0x28. Причина этого ограничения в том, что когда ДОС вызывает прерывание DOSOK, флажок занятости ДОС остается установленным. При выполнении TSR-программы он не меняет значения, и, если пользователь нажмет "горячий ключ" для другой TSR-программы, ее обработчик таймера будет ожидать, пока флажок занятости не сбросится. А это не случится до окончания текущей TSR-программы. Такая ситуация всегда случается при вызове TSR-программы в момент, когда ДОС ожидает ввода команды пользователя.

Резидентная программа, чтобы разрешить прервать свое выполнение другой такой же программе, должна вызвать прерывание DOSOK в подходящее время. Лучше всего это сделать, когда текущая программа ожидает ввода строки от пользователя. Если вы обратили внимание, функция get_char из главы 4 вызывает прерывание 0x28 во время ожидания ввода строки. Это делается для поддержки концепции резидентных программ, описанной в этой главе и используемой в главе 12.

Дисковое прерывание ROM-BIOS.(0x13(

Дисковые операции нельзя прерывать из-за возникновения ошибок в прерванной программе. Чтобы избежать этого, TSR-программа должна присоединить себя к дисковому прерыванию ROM -BIOS 0x13. Если какой-либо процесс вызывает это прерывание, TSR-программа устанавливает флаг и затем передает управление по старому адресу прерывания. Когда обработка прерывания 0x13 завершается, TSR-программа очищает флаг. Этот флаг проверяется ISR таймера и DOSOK. Если производится дисковая операция, то TSR не разрешает прерывание.

Прерывание тяжелой ошибки в ДОС.(0x24(

При возникновении тяжелой ошибки ДОС вызывает прерывание 0x24. Например, при обращении к дискете в открытом дисководе, ДОС обнаруживает состояние неготовности и вызывает прерывание тяжелой ошибки. Если программа не присоедиила себя к вектору 0x24, все критические ошибки будут обрабатываться ISR в командном процессоре ДОС. Он выдает сообщение "Abort,Retry, or Ignore" на экран. ISR возвращает в ДОС значение, определяющее порядок дальнейшей обработки.

Предположим, что ни одна программа не присоединена к вектору 0x24. Ваша TSR прерывает нерезидентную программу, и пытается прочитать дискету из открытого дисковода. Командный процессор ДОС перехватывает управленеие и запрашивает пользователя о дальнейших действиях. Пользователь отвечает "A" (Abort-прервать), и ДОС пытается прервать выполнение программы. Если ваша TSR переключила адрес PSP, то ДОС будет пытаться прервать TSR. ДОС не знает, что в памяти находится нерезидентная программа в прерванном состоянии, и не возвращает ей управление. Система ломается. Если

TSR не переключала PSP, то ДОС завершает нерезидентную программу и не возвращает управление TSR.

Предположим, что нерезидентная программа присоединилась к вектору 0x24. Дисковая ошибка в вашей программе вызовет исполнение обработчика тяжелой ошибки из прерванной нерезидентной программы.

TSR должна присоединяться к вектору 0x24 в любом случае, будет ли или не будет в ней какая-либо обработка ошибок. Вектор присоединяется при "всплытии" TSR и восстанавливается на тот, что был, при возвращении управления в прерванную программу. Никакие обстоятельства не должны заставить TSR просить ДОС прервать обработку. Большинство TSR-программ просто игнорируют ошибки и говорят ДОС также их игнорировать. Прерывание тяжелой ошибки не связывается в цепочки - это опасно. Если другая TSR-программа)например, спулер) присоединила к себе вектор 0x24 и вызвана после вашей TSR, она получит ваши ошибки. Если же эта программа была вызвана до вашей TSR, то вы будете получать ее ошибки и говорить ДОС игнорировать их, что является проблемой, которую нельзя разрешить без написания системно-ориентированного обработчика ошибок.

Прерывание Ctrl-Break в ДОС.(0x23(

При нажатии пользователем клавиш Ctrl-Break ДОС отображает на экране в текущей позиции курсора символы ^C и вызывает прерывание 0x23. Обработчик этого прерывания, имеющийся в ДОС, вызывает немедленное завершение текущей программы. TSR-программу завершать таким способом нельзя: слишком много векторов прерываний прикреплено к ней, и нерезидентная программа может находиться в памяти за ней. Если вы просто проигнорируете это прерывание, имеется риск, что другая программа, загруженная за вашей, присоединит себя к этому прерыванию и сделает что-то неподходящее при нажатии клавиш Ctrl-Break во время выполнения вашей программы.

В ДОС имеется функция (0x33), которая позволяет программе читать текущий статус (разрешено/запрещено) обработки Ctrl-Break и устанавливать его. При вызове TSR-программы она должна прочитать текущее значение статуса, и затем запретить обработку Ctrl-Break. При завершении своей работы TSR должна восстановить значение статуса обработки Ctrl-Break. Присоединяться к этому прерыванию нет необходимости.

После таких действий вашей TSR в случае нажатии Ctrl-Break во время ее выполнения, после возврата в прерванную программу она будет закончена ДОС. Если все TSR будут использовать такой способ, то завершаться по Ctrl-Break будут только нерезидентные программы.

Выполнение TSR-программы.

Выполнение TSR-программы проходит в два этапа. Первый этап- когда пользователь запускает ее из командной строки. Программа выполняет инициализационный код, сохраняет свой контекст, присоединяет себя к прерываниям, если это нужно, и завершается с использованием функции ДОС TSR, таким образом объявляя себя

резидентной.

Второй этап - когда TSR-программа запускается в результате одного из прерываний, к которому она прикрепилась. В большинстве случаев, векторы прерываний связываются в цепочки, как это было описано ранее. Программа определяет, может ли она выполняться- это зависит от состояния некоторых важных системных индикаторов, описанных выше. Если она может выполняться, она сохраняет контекст прерванной программы, восстанавливает свой собственный контекст, выполняет свою задачу, восстанавливает контекст прерванной программы, и передает ей управление.

Завершение TSR-программы.

Могут быть причины, по которым вы можете захотеть завершить TSR-программу, что отнюдь не является легкой задачей. Вспомните, ведь ДОС не знает ничего о программе, оставшейся резидентной, и вам самим надо сделать все то, что обычно делает ДОС по отношению к обычным нерезидентным программам.

Завершение TSR-программы включает в себя следующие шаги:

.1 Сообщение программе о том, что ей надо завершиться.

Для этого можно использовать другой "горячий ключ", или коммуникационный вектор прерывания, указывающий на сигнатуру в программе, означающую, что программа уже в памяти. Второй метод используется в драйвере TSR-программ, описанном в главе 12. При этом пользователь, знающий, что программа уже резидентна, запускает ее второй раз, давая в командной строке параметр, означающий, что программа должна завершиться. Программа ищет свою сигнатуру способом, описанным выше. При нахождении сигнатуры она также находит свой вектор прерывания. Через этот вектор она посылает команду завершения свой резидентной копии, после чего та совершает следующие шаги завершения.

.2 Восстановление векторов прерывания в первоначальное состояние.

У вас может быть возможность сделать это, а может и не быть. Если другая TSR-программа была загружена после вашей, и она связала векторы в цепочки, то вы успешно дезактивируете ее при восстановлении векторов. Далее, если эта TSR-программа будет затем завершена, то она переустановит вектора на адреса в вашей программе, по которым уже может находиться неизвестно что. Так что до того, как восстанавливать вектора, надо убедиться, что ваша программа до сих пор владеет ими. Чтобы определить это, надо сравнить адреса, находящиеся в соответствующих векторах, с адресами соответствующих обработчиков в вашей программе. Если какой-либо вектор изменен, то ваша программа уже не владеет им, поэтому завершать ее нельза и надо просто приостановить ее.

Еще одним осложнением, возникающем при завершении вашей TSR с другой программой такого же типа, загруженной после нее, является фрагментация памяти в нерезидентной области. ДОС будет использовать область, освобожденную вашей программой, при запросах на память из выполняющихся программ, но для загрузки других программ эта область не будет использоваться. Помните, что

ДОС - однозадачная ОС - не понимает концепции фрагментированных программ, потому что она не понимает мультипрограммирование.

.3 Закрытие всех файлов, открытых в программе.

При завершении нерезидентной программы ДОС автоматически закрывает все файлы, сканируя массив указателей файлов программы в PSP и закрывая все вхождения в него, которые еще используются. Помните, что эти вхождения - это номера элементов массива, который поддерживает ДОС .При завершении и объявлении резидентной ДОС не закрывает файлы, открытые этой программой, и при завершении работы TSR-программы необходимо это сделать. Если файлы не будут закрыты, то элементы массива не будут освобождены, и будут недоступны для дальнейшего использования другими программами. Загрузка и завершение без закрытия файлов TSR-программ может привести к исчерпанию таблицы файлов ДОС и "зависанию" системы.

Элементы в таблице указателей файлов представляют файлы на уровне указателей. Это эквивалентно функциям небуферизованного ввода-вывода низкого уровня в Си. Функции Си, выполняющие буферизованный потоковый ввод-вывод, поддерживают собственные буферы и указатели и могут потребовать особого порядка закрытия, называемого потоковым. Такое закрытие - это библиотечная функция Си, отличающаяся от соответствующей функции ДОС. Такие файлы надо закрывать стандартной функцией fclose. Файлы, открытые функциями open и creat, могут быть закрыты функцией close.

.4 Возврат памяти, занимаемой программой, в ДОС.

Для этого можно использовать функцию ДОС 0x49. TSR-программой, как минимум, занимается два блока памяти. Первый –это блок системных параметров, адрес которого хранится в PSP со смещением 0x2c. Второй блок - это сам PSP. Если программе выделены оба этих блока, то они оба должны быть возвращены ДОС. В главе 12 демонстрируется техника сканирования списка управляющих

блоков памяти ДОС для нахождения и возвращения в ДОС блоков памяти, выделенных программе.

Приостановка и возобновление выполнения TSR-программы.

При приостановке TSR-программы ее не удаляют из памяти, а только дают ей команду не реагировать на прерывания путем установки флага. Команда возобновления сбрасывает этот флаг. С этой целью можно использовать коммуникационный вектор прерывания.

TSR-драйвер в главе 12 использует эту технику для завершения, приостановки, и возобновления выполнения TSR-программ.

Выводы.

В этой главе описывается операционное окружение резидентных программ. В главе 12 на примере двух таких программ описывается

весь процесс шаг за шагом. Первая программа - это оперативная программа-часы, которая поддерживает на экране постоянное отображения текущих времени и даты. Вторая программа более интересна - это TSR-драйвер. Вы связываете свою программу на Турбо Си с этим драйвером, используя некоторые соглашения для инициализации, и ваша программа на Турбо Си становится резидентной. Это значит, что программа будет вызываться по нажатию клавиши, что она может открывать, читать, пиать в и закрывать дисковые файлы, что она использует ROM-BIOS функции для чтения с клавиатуры и прямой доступ к видеопамяти для вывода на экран, и что она никогда не выходит в ДОС.

ГЛАВА 12

Построение резидентных программ

В этой главе демонстрируется, как концепции TSR-программ, описанные в главе 11, воплощаются на практике в программы на Турбо Си. Функции из этой главы представляют собой программу-драйвер, связав которую с вашей программой на Турбо Си, вы получите резидентную программу. Имеются некоторые ограничения на эту программу. Первое - это то, что в ней не должны применяться функции Турбо Си, вызывающие прерывание ДОС 0x21 с номерами функций от 0 до 12. Это означает, что весь обмен с клавиатурой и экраном должен выполняться с помощью вызовов BIOS или прямого доступа к экранной памяти. В примерах используются оконные функции из предыдущих глав, удовлетворяющие этим соглашениям. Второе - TSR-программа должна быть скомпилирована в крохотной (tiny) или малой (small) модели памяти.

В главе 10 демонстрируется интеграция всех предыдуших примеров в одну программу, выполняемую под управлением оконных меню. В этой главе та же программа превращается в TSR.

Интегрированный пример использует файловые функции ДОС и выполняет в соответствии с этим все правила, описанные в главе .11По причине сложности этой задачи и для того, чтобы облегчить ваше знакомство с программированием TSR, первый пример не требует соблюдения этих правил.

Пример TSR-программы - "часы."

На листинге 12.1 приведена программа clock.c, простая TSR-утилита, обеспечивающая постоянное отображение даты и времени в верхнем левом углу экрана. В программе после активизации не делаются вызовы ДОС, поэтому нет нужды в защите от нереентерабельности ДОС.

Превращение программы в резидентную.

Функция main производит все подготовительные действия и объявляет себя резидентной. Сначала она сохраняет свой указатель стека, что позволит затем восстановить свой собственный стек при вызове. Затем в программе используется getvect для чтения текущего вектора прерывания таймера, после чего с помощью setvect в качестве обработчика таймерного прерывания устанавливается функция newtimer. Указатель стека TSR-программы устанавливается как функция от объявленного размера программы и адреса

видеопамяти, определяемого на основе значения, возвращаемого функцией vmode. Для получение даты и времени используются функции ДОС.

Прерывание по делению на ноль.

При старте программ, написанных на Турбо Си, выполнение начинается со стартового кода. При этом устанавливается начальные величины стека и "кучи" и вызывается функция main. Стартовый код находится в файлах c0t.obj (для крохотной модели) и c0s.obj (для малой модели). Эти файлы поставляются вместе с Турбо Си. В стартовом коде находится обработчик прерывания по делению на ноль, который присоединяется к соответствующему вектору перед вызовом функции main. При выполнении return из функции main этот вектор устанавливается в предыдущее свое значение. Возврат из функции main нормальной нерезидентной программы означает, что

программа закончила свои действия и готова к завершению. Но TSR-программы не завершаются выдачей return из функции main. Они используют одну из TSR-функций ДОС, и таким образом должны сами восстанавливать предыдущее значение вектора прерывания по делению на ноль до завершения и превращения в резидентную. Если этого не будет сделано, то ошибка деления на ноль в другой программе будет обрабатываться стартовым кодом вашей TSR-программы.

Фирма Borland поставляет исходные тексты стартового кода Турбо Си. Они находятся в файлах c0.asm и rules.asi. Вам понадобится модифицировать c0.asm и ассемблировать его дважды- для крохотной и малой моделей памяти. Адрес, по которому в c0.asm сохраняется вектор прерывания по делению на ноль, назван ZeroDivVector. Эта переменная локальна в c0.asm. Чтобы ваша программа могла восстановить значение этого вектора, к нему надо получить доступ путем преобразования ZeroDivVector в public-переменную. Турбо Си добавляет символ _ в начале имен внешних переменных, и каждое вхождение переменной ZeroDivVector в c0.asm вы должны заменить на _ZeroDivVector. Затем надо заменить оператор, объявляющий ZeroDivVector в программе, на следующий:

```
PubSym@ ZeroDivVector <dd 0>,_CDECL_
```

Ассемблируйте файл дважды с помощью следующих команд:

```
C>masm c0,c0t /ML /D_TINY,_
C>masm c0,c0s /ML /D_SMALL,_
```

после чего будут созданы файлы c0t.obj и c0s.obj. Замените исходные файлы Турбо Си на эти.

Обратите внимание на объявление в clock.c указателя функции прерывания ZeroDivVector. Внешний указатель - это как раз то, что вы только что объявили public в стартовом коде. В функции main, до объявления себя резидентной, clock.c использует функцию setvect для восстановления вектора прерывания по делению на ноль. Затем clock.c завершается с объявлением себя резидентной.

Если у вас нет исходных текстов стартового кода, можно найти следующий выход: позволить TSR-программе указывать на ошибку деления на ноль, как только она случилась. При возникновении такой ошибки в другой программе стартовый код вашей программы

будет выдавать сообщение об ошибке и завершать программу, в точности как и соответствующий обработчик ДОС. При завершении программы ДОС устанавливает вектор прерывания на свой обработчик деления на ноль. Ваша программа больше не будет работать. Конечно, вы должны удалить ссылки на ZeroDivVector в clock.c и resident.c.

Выполнение обработчика прерываний от таймера.

С каждым "тиканьем часов", происходящим 18.2 раза в секунду, вызывается функция newtimer, объявленная как interrupt в Турбо Си. Это объявление означает, что при вызове функции регистры сохраняются в стеке и регистр сегмента данных указывает на сегмент данных программы, с которой функция связана с помощью link. Такое объявление также гарантирует восстановление регистров из стека и выполнение машинной команды IRET при завершении работы функции. Команда IRET используется обычно для выхода из обработчика прерывания. Она восстанавливает регистры программного счетчика, флагов и сегмента кодов, сохраненные при возникновении прерывания.

Связывание старого вектора прерывания по таймеру.

При выполнении newtimer прежде всего вызывается обработчик прерывания, на который указывает oldtimer. Это действие позволяет другим программам, уже присоединенным к вектору, произвести необходимые действия. Функция newtimer проверяет флаг, который устанавливает сама же эта функция, и означающий, что она еще работает.

Сохранение и переключение контекста стека.

Функция newtimer сохраняет сегмент стека и регистры указателей - эти величины принадлежат прерванному процессу. Значения стековых регистров, сохраненные при выполнении clock.c, восстанавливаются в регистрах процессора, поэтому clock. c может использовать свой собственный стек и не портить стек прерванной программы.

Вычисление времени.

Функция newtimer подсчитывает сигналы таймера. При прохождении 18 сигналов (19 каждый пятый раз, так как сигналы приходят 18.2 раза в секунду), новая величина времени вычисляется для отображения на экран.

Затем дата и время отображаются в верхнем левом углу экрана, восстанавливаются значения регистров стека прерванной программы, и newtimer возвращает ей управление.

Заметим, что newtimer не переводит дату в полночь и не изменяет значения на экране после ввода новых даты и времени командой ДОС. Эта программа лишь иллюстрирует работу простейшей TSR-программы. Если вы не работаете после полуночи, вы можете использовать ее для отображения даты и времени на экране. Она обновляет значения каждую секунду, поэтому вывод на экран другими программами ничего не испортит. В качестве эксперимента вы можете добавить будильник в clock.c. Включите время, когда надо "звонить", как параметр, передаваемый в командной строке при первом запуске clock.exe. Затем, при каждом изменении значения часов сравнивайте его с этим временем. При равенстве времен, выдайте звуковой сигнал, избегая, естественно, использования функций ДОС. Позднее, когда вы узнаете, как использовать коммуникационный вектор прерывания, вы сможете модифицировать clock.c для установки и изменения времени звонка путем запуска clock.exe из командной строки с параметром в то время, как TSR уже резидентна. Вы можете добавить комментарии к звонку путем использования оконных функций и оконного редактора. Путем именно таких последовательных улучшений были созданы программы мировой известности.

Программа clock.c использует прерывание от таймера. Если вы загрузите ее после Sidekick, часы перестанут идти при вызове Sidekick. Так как newtimer просто считает секунды, а не читает время ДОС, такая смесь программы с Sidekick'ом сделает ее результаты неверными. Sidekick отбирает вектор прерывания от таймера у любой TSR-программы, загружаемой после него, чем и вызыает такой результат. Остерегайтесь Sidekick'а при загрузке ваших резидентных программ.

Чтобы запустить "часы", введите следующую команду:

```
C>clock
```

) Листинг 12.1.(

```
---------*/clock.c/*-----------

#include <dos.h<

void interrupt (*oldtimer;()(
void interrupt newtimer;()
extern void interrupt (*ZeroDivVector;()(
#define sizeprogram 375
unsigned intsp,intss;
unsigned myss,stack;
static union REGS rg;
struct date dat;
struct time tim;
unsigned vseg;
int running = 0;
char bf[20;[
unsigned v;
char tmsk []= " %2d-%02d-%02d %02d:%02d:%02d;"
int ticker = 0;

static struct SREGS seg;

main()
}
  segread(&seg;(
```

```
  myss = _SS;

  oldtimer = getvect(0x1c;(
  setvect(0x1c,newtimer;(
  stack = (sizeprogram - (seg.ds - seg.cs))*16-300;
  vseg = vmode() == 7 ? 0xb000 : 0xb800;

  gettime(&tim;(
  getdate(&dat;(

  setvect(0,ZeroDivVector;(

  rg.x.ax = 0x3100;
  rg.x.dx = sizeprogram;
  intdos(&rg,&rg;(
{

void interrupt newtimer()
}
*)  oldtimer;()(
  if (running ==0(
}
     running = 1;
     disable;()
     intsp = _SP;
     intss = _SS;
_     SP = stack;
_     SS = myss;
     enable;()
     if (ticker ==0(
}
        ticker = (((tim.ti_sec % 5;( 18: 19 ?(0== (
        tim.ti_sec;++
        if (tim.ti_sec == 60(
}
           tim.ti_sec =0;
           tim.ti_min;++
           if (tim.ti_min == 60(
}
               tim.ti_min=0;
               tim.ti_hour;++
               if (tim.ti_hour == 24(
                   tim.ti_hour = 0;
{
{
         sprintf(bf,tmsk,dat.da_day,dat.da_man,dat.da_year % 100,
                 tim.ti_hour,tim.ti_min,tim.ti_sec;(
{

      for (v=0;v<19;v(++
         vpoke (vseg,(60+v)*2,0x7000+bf;([]

      disable;()
_      SS = intsp;
_      SS = intss;
      enable;()
      running = 0;

{
{
```

Файл проекта для построения clock.exe с именем clock.prj имеет следующий вид:

Листинг 12.2: clock.prj

```
clock
ibmpc.obj
```

ПРОГРАММЫ TSR-ДРАЙВЕРА.

Чтобы расширить возможности TSR-программ по использованию функций ДОС при ее вызове, в этой главе приводится два исходных текста на Си. После их адаптации и связи с любой стандартной программой на Си, та становится резидентной программой. Первый текст содержит функцию main, и туда помещаются параметры, зависящие от вашей программы. Второй файл - это основной TSR-драйвер, управляющий присоединением векторов прерываний, самих прерываний, арбитраж столкновений ДОС и BIOS, определение, резидентна ли уже программа, приостановка и возобновление работы TSR-программы, и удаление TSR-программы из памяти.

Третий модуль в этом наборе - ваша прогррамма на Си, которая должна придерживаться следующих правил, чтобы правильно выполняться в этом окружении:

- программа должна быть построена в крохотной или малой моделях памяти;
- программа не должна использовать функций ДОС от 0 до 12;
- при изменении текущего дискового каталога программа должна восстанавливать его при возврате в прерванную программу;
- программа не должна использовать операции с плавающей запятой;
- программа не должна завершаться или выходить в ДОС.

Вас может заинтересовать, почему надо избегать использования операций с плавающей запятой. Дело в том, что подпрограммы с плавающей запятой Турбо Си используют ряд векторов прерываний, которые присоединяются во время выполнения стартового кода. Эти вектора не восстанавливаются до завершения программы и передачи управления при этом в стартовый код. В стартовом коде не поддерживается область сохранения для этих векторов (среди которых имеется и вектор немаскируемого прерывания 2); таким образом, при завершении работы TSR-программы и удалении ее из памяти эти векторы не будут восстанавливаться.

В этолй главе используются программа exec.c из главы 10 и все примеры оконных программ практически готовых стать резидентными.

Действия трех программных модулей.

Три программных модуля для TSR-программ - это popup.c, resident.c и ваша утилита на Турбо Си. Popup.c (листинг 12.3) и resident.c (листинг 12.4) содержат сам TSR-драйвер. Popup.c - это модуль, который надо изменить соответственно требованиям вашей

утилиты, а **resident.c** - модуль, остающийся неизменным для всех TSR-программ. В дальнейшем обсуждении мы будем переключаться между двумя этими модулями, так как они оба созданы для поддержки создания вами TSR-программ.

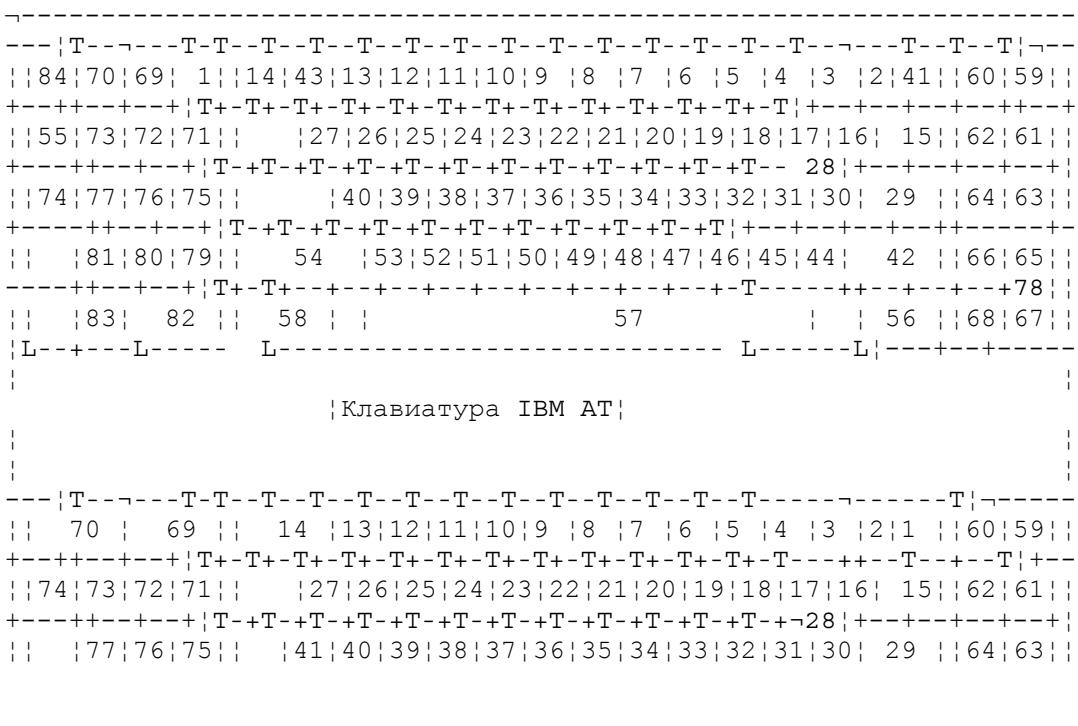
Popup.c содержит кроме любого установочного кода, необходимого вам, еще и несколько переменных, которые надо инициализировать значениями, описывающими вашу программу.

Размер TSR-программы.

Беззнаковая переменная **sizeprogram** специфицирует размер программы в параграфах по 16 байт. Вы уже прочитали в главе 11, как определять это значение. До тех пор, пока ваша программа не начнет действовать, увеличивайте это значение. Программа в крохотной модели не может быть больше, чем 64К (4096 параграфов,(а в малой модели - больше 128К (8192 параграфов.(

Присвоение "горячего ключа."

Значение клавиши "горячего ключа" TSR-программы определяется значениями беззнаковых переменных **scancode** и **keymask**. При нажатии клавиши обработчик прерывания 09 читает входной порт клавиатуры, в котором находится скан-код клавиши) ее порядковый номер.(Каждая клавиша имеет свой скан-код, которые изображены на рисунке .12.1Следовало бы присваивать такие клавиши, которые не совпадали бы с клавишами других программ, в том числе и нерезидентных. Стоит избегать функциональных клавиш, комбинаций с Alt- и Ctrl-, так как многие программы их используют. Наилучшим выбором является редкая и оригинальная комбинация клавиш, такая, как Alt-точка.



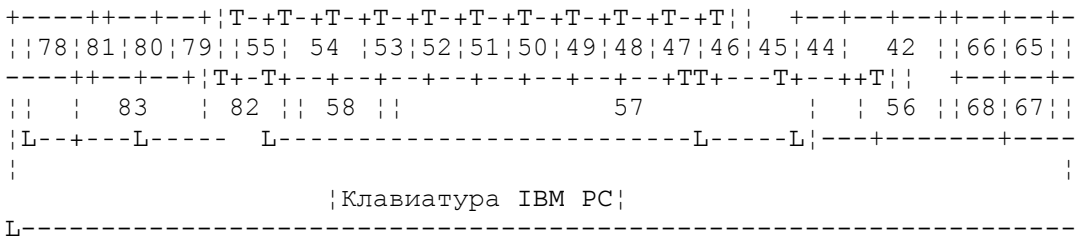


Рис.12.1. Скан-коды клавиатуры.

Обнаружение комбинации клавиш облегчается при использовании маски статуса по адресу 0:417, содержащей биты для клавиш Ctrl, Alt, Shift, Ins, Caps Lock и Scroll Lock. Эта маска показывает, какая из этих клавиш в текущий момент нажата. На рис.12.2 показано положение битов в маске.

INS	CAPS LOCK	NUM LOCK	SCROLL LOCK	ALT	CTRL	LEFT SHIFT	RIGHT SHIFT

Рис.12.2.

Если соответствующий бит в маске установлен в 1, то клавиша нажата. По скан-коду нажатой клавиши и значению бита в маске обработчик прерываний от клавиатуры может определить, нажата ли клавиша "горячего ключа". Чтобы специфицировать эти значения, вам надо присвоить значения переменным scancode и keymask в popup.c. Как показано в листинге, scancode равен 52, и keymask равен 8, что соответствует Alt-точка.

Сигнатура TSR-программы.

Программа popup.c также обеспечивает сигнатуру, которая используется для определения, не резидентна ли уже TSR-программа. Символьный массив с именем signature - это строка, заканчивающаяся \0.

При первом старте popup.c вызывает функцию resident. Она находится в resident.c, ее задачей является определение, резидентна ли уже программа, и если нет, определить и назначить коммуникационный вектор. Это делается сканированием пользовательских векторов прерываний от 0x60 до 0x67. Если какой-либо вектор содержит значение, то сегментная половина этого значения комбинируется со смещением сигнатуры в программе. Это смещение является указателем на сигнатуру, то есть смещением относительно регистра сегмента данных. Сегментная часть вектора суммируется с разницей между значениями регистров сегмента кода и сегмента данных.

Помните, что значение сегментного регистра, взятое из вектора, должно быть значением сегмента кода первоначально стартовавшей TSR-программы. Сигнатура имеется как в области данных TSR-программы, так и в области данных копии TSR-программы, которая пытается стать резидентной. Нам нужен сегментный адрес данных TSR-программы, который не сохраняется в коммуникационном векторе. Это значение должно быть вычислено.

Программа ,просматривающая векторы, является второй копией

TSR-программы (в предположении, что TSR-программа уже загружена,(поэтому разность значений регистров кода и данных у нее должна быть такой же, как и у уже резидентной программы. Пользуясь этим алгоритмом, вы сравниваете значение по соответствующим смещением, и если они одинаковы, то программа уже загружена, и функция resident вернет найденный вектор в функцию main в popup.c. Если сигнатуры не равны, сканирование продолжается, пока не будут проверены все векторы. Если определяется, что TSR-программа еще не резидентна, то первый подходящий вектор становится коммуникационным. В функцию main возвращается 0, что означает, что программа стала резидентной.

Коммуникационные прерывания.

Если функция main обнаруживает, что копия программы уже резидентна, она проверяет параметры командной строки. Используемая техника позволяет передавать при запуске программы параметры для ее резидентной копии. Напомним, что в памяти имеется в этот момент две копии TSR-программы - резидентная и только что загруженная в нерезидентную область. Нерезидентная версия может связываться с резидентной копией через коммуникационный вектор. В типичной TSR-программе используются три параметра, но вы можете добавить еще, если необходимо. Стандартные три позволяют пользователю удалять программу из памяти, приостанавливать и возобновлять ее выполнение. Если один из этих параметров присутствует в командной строке (это определяется при помощи args,argv), то генерируется программное прерывание, с установлением регистра ax в соответствующее опции значение. Прерывание генерируется функцией main нерезидентной копии TSR-программы.

Посмотрим далее на листинг popup.c. Функция прерывания с именем ifunc является обработчиком коммуникационного прерывания. Когда функция main нерезидентной копии TSR-программы инициирует прерывание, вызывается обработчик из резидентной копии. Он проверяет значение регистра ax и производит соответствующие действия. При этом вызывается одна из трех функций: terminate, restart или wait в resident.c. Вы можете вставить сюда и другую логику, чтобы управлять состоянием резидентной программы.

Функции wait и restart в resident.c просто устанавливают и очищают флаг, означающий для обработчика клавиатуры необходимость реакции на нажатие клавиши "горячего ключа."

Функция terminate в resident.c должна определять, может ли быть завершена TSR-программа, и затем, если да, производить все действия, аналогичные действиям ДОС при завершении нерезидентной программы. Этот процесс будет обсуждаться после того, как вы поймете, как программа становится резидентной.

Подготовка к резидентности.

Если функция resident обнаруживает, что программа еще не в памяти, то popup.c подготавливает себя к переходу в резидентные. ваша программа захочет открыть файлы или сделать еще что-нибудь

"по хозяйству" при ее вызове. Popup.c, взятая для примера, вызывает load_help для установки функций помощи из главы 7, устанавливает путь ДОС для использования программой notepad из главы 9, и выдает сообщение. Потом она вызывает resinit, функцию из resident.c, которая делает все остальное, чтобы программа стала резидентной.

Первоначально, resinit сохраняет значение регистра сегмента стека для переключения контекста стеков при вызове TSR-программы. Затем читается флаг занятости ДОС с помощью функции ДОС 0x34. Функция getdta получает а□□ьідрес дискового буфера для TSR-программы. Этот адрес также используется при дальнейшем переключении контекстов. Адрес идентификатора процесса (PID) восстанавливается и сохраняется. Эта техника объяснялась в главе 11. Векторы прерывания для таймера, клавиатуры, диска и DOSOK читаются и запоминаются для связывания прерываний в цепочки, и собственные обработчики присоединяются к этим прерываниям. Указатель стека TSR-программы вычисляется так, чтобы быть на 300 байт меньше, чем размер программы, и вектор обработки деления на ноль восстанавливается на то значение, которое он имел до того , как стартовый код присоединил его. Затем TSR-функция ДОС используется для завершения программы, оставляя ее резидентной.

Обработчик обращения к диску.

В то время, как компьютер выполняет программу, и программа вызывает ДОС, ДОС использует дисковое прерывание BIOS (0x13) для чтения и записи секторов данных. Напомним, что в главе 11 вас уже предупреждали о том, что дисковые операции прерывать нельзя. Обработчику диска в resident.c дано имя newdisk, и он защищает дисковые операции от прерывания вашей TSR-программой .При возникновении прерывания 0x13 устанавливается флаг, а после этого управление передается на обработку дисковых операций. После окончания обработки флаг очищается. Если этот флаг установлен, когда программа вызывается по нажатию "горячего ключа", то производится задержка выполнения до очистки этого флага.

Обратите внимание на трюк в newdisk. Функция, описанная как interrupt, сохраняет все регистры в стеке, устанавливает регистр ds на сегмент данных программы, содержащей эту функцию. Затем начинается выполнение кода функции. При завершении функции регистры восстанавливаются, и выполняется машинная команда IRET. Эта команда восстанавливает регистры программного счетчика, сегмента кода и флажков. Таким образом, выполняется полное восстановление регистров прерванной программы.

Все работает до тех пор, пока вы не имеете прерывания, которое возвращает условие в флаге переноса. Команда IRET восстанавливает все флаги в состояние, которое было при возникновении прерывания. Некоторые программные прерывания ДОС и BIOS используют этот флаг для индикации результата. В их числе и прерывание 0x13 BIOS; однако программа обработчика обращений к диску сохраняет значение флажка переноса. Когда заканчивает свои действия присоединенный обработчик olddisk, необходимо вернуть вызывавшему процессу два значения: регистр ax и флажок переноса. Значение ax берется из псевдопеременной _AX и помещается в целую переменную ax, являющуюся одним из параметров функции прерывания. Турбо Си использует это средство, чтобы изменять значения регистров, которые будут восстановлены из стека при возврате из

функции. Для регистра флагов псевдопеременной нет, и, чтобы избежать программирования на ассемблере, делается следующий хэккерский трюк: за функцией newdisk может быть сразу же вызвана функция newcrit, и она запишет флаговый регистр, сохраненный в стеке при ее вызове, во внешнюю переменную cflag. При возврате из newcrit cflag записывается в стек, где было сохранено прошлое значение регистра флагов.При возврате из функции результирующие флаги из olddisk будут восстановлены в регистр флагов.

Эти действия базируются на понимании того, как Турбо Си использует регистры и производится вызов и возврат из функций interrupt. Таким образом, эти программы не переносимы на другой компилятор, и могут быть несовместимы даже с будущими версиями Турбо Си, если Borland изменит свои соглашения. Программисты на ассемблере, возможно, скажут, что подобные дествия можно было бы легко проделать на их любимом языке. Эта критика верна, но этот программный трюк является примером, как достигнуть пределов возможностей Турбо Си. Автор благодарен Турбо Си за тот сервис, который предоставляется для разрешения всех проблем, возникающих при создании TSR-программ.

Обработчик критических ситуаций.

Функция interrupt с именем newcrit является присоединенным обработчиком критических ситуаций. Она не присоединяется к прерыванию, когда программа объявляет себя резидентной, а делает это лишь временно, при "всплытии" TSR-программы. Его задача- обезопасить TSR-программу от возникновения критических ошибок в то время, как она переключила контекст на себя. Обработчик возвращает ноль в регистре ax, что означает для ДОС игнорировать ошибку.

Обработчик клавиатуры.

Функция interrupt с именем newkb является обработчиком клавиатуры для TSR-программы. Она читает порт данных клавиатуры и проверяет на соответствие определенному скан-коду, означающему нажатие клавиши "горячего ключа". При равенстве и если TSR-программа не приостановлена, "горячий ключ" активизируется. Функция сбрасывает клавиатуру, чтобы не было будущих прерываний, и затем проверяет, а не вызвана ли уже TSR-программа. Если нет, то устанавливается флаг, означающий, что нажат "горячий ключ", и выполнение функции заканчивается.

Если скан-код и маска статуса не равны "горячему ключу", то управление передается старой программе обработки прерываний от клавиатуры.

Обработчик таймера.

Каждый импульс таймера вызывает выполнение функции newtimer в resident.c. Прежде всего она вызывает старый обработчик

таймера. Затем она проверяет, не нажат ли "горячий ключ". Если да, то проверяется флажок занятости ДОС. Если ДОС не занята, то newtimer проверяет, не производится ли в этот момент дисковая операция. Если нет, то сбрасывается прерывание от таймера, очищается флажок "горячего ключа", и вызывается функция dores, начинающая выполнение TSR-программы.

Обработчик DOSOK.

Прерывание DOSOK обслуживается обработчиком с именем new28 в resident.c. Он присоединяется к старому обработчику этого прерывания, и проверяет флажок "горячего ключа". Если он установлен, то проверяется, занята ли ДОС, и затем очищается флажок "горячего ключа" и вызывается dores.

Выполнение TSR-программы.

Функция dores вызывается лишь после того, как программа убедилась в безопасности своего выполнения. Dores прежде всего устанавливает флажок, означающий, что она выполняется. Эта установка предохраняет от повторного вызова путем вторичного нажатия "горячего ключа". Затем сохраняется регистр стека прерванной программы, и указатель стека устанавливается на собственный стек TSR-программы.

Сохраняется вектор прерывания по критической ошибке, затем соответствующий обработчик присоединяется к этому вектору. Текущий статус Ctrl-Break сохраняется, и прерывания по Ctrl-Break запрещаются.

Адрес дискового буфера прерванной программы сохраняется, и устанавливается на соответствующий текущему контексту. То же производится и с идентификатором процесса. Затем вызывается утилита popup из popup.c. Функция popup сохраняет текущее положение курсора, вызывает вашу программу, после ее выполнения восстанавливает курсор и заканчивается. В листинге popup.c вызывается функция exec, вы можете подставить туда имя вашей программы.

При завершении popup адреса идентификатора процесса, дискового буфера, вектор прерывания по критической ошибке, статус Ctrl-Break и указатель стека восстанавливаются в те значения, которые они имели до вызова TSR-программы, и выполняется возврат в прерванную программу.

Удаление TSR-программы.

При удалении пользователем TSR-программы путем запуска ее копии с соответствующим параметром в командной строке, нерезидентная копия вызывает резидентную через коммуникационное прерывание. Функция terminate в resident.c проверяет, может ли быть снята программа путем просмотра, не изменились ли значения векторов прерываний. Если изменились, выполнение TSR-программы приостанавливается. Если нет, она может быть снята.

Для удаления TSR-программы необходимо проделать три процедуры. Сначала все файлы должны быть закрыты. Когда ДОС завершает программу, она проверяет все 20 элементов в массиве указателей файлов в PSP. Эта процедура закрывает все файлы на уровне указателей и не затрагивает потоковых файлов. Так как эти файлы должны быть закрыты, то terminate вызывает функцию closefiles в popup.c, закрывающую все открытые файлы.

Вторая процедура - восстановление всех векторов прерываний в значение, которое они имели до присоединения их к себе TSR-программой.

Завершающим шагом является освобождение всех блоков памяти, распределенных под TSR-программу. Память распределяется двумя способами - из ДОС и из программы через вызов функций ДОС. Эти два типа блоков памяти должны освобождаться тем же путем, которым и выделялись.

Блоки памяти и управляющие блоки памяти.

Выделяемый ДОС блок памяти содержит 16-байтный блок управления памятью (БУП), следующий сразу за соответствующим распределяемым блоком памяти. Он содержит следующие поля:

- однобайтный маркер, идентифицирующий БУП. Все, кроме последнего БУП в списке, имеют значение маркера 0x4d. Последний БУП имеет маркер 0x5a.

- двубайтный идентификатор процесса, которому принадлежит блок памяти. Если блок свободен, это поле содержит 0.

- двубайтный размер блока памяти в параграфах. Размер БУП не учитывается в этом значении. БУП следует в памяти непосредственно за блоком памяти, который он представляет. Связки БУП-блок памяти располагаются в памяти смежно. Сегментный адрес следующего БУП равен адресу предыдущего БУП + размер блока памяти + 1. Если у вас есть адрес первого БУП в памяти, то можете проследить всю цепочку.

В ДОС имеется функция (естественно, недокументированная,(которая может быть использована для определения адреса первого БУП в цепочке. Эта функция 0x52 возвращает сегментный адрес первого БУП в регистре es и смещение в регистре bx. Эффективный адрес этой пары сегмент: смещение, уменьшенный на 2, дает адрес слова, содержащего сегментный адрес первого БУП в цепочке распределенных ДОС блоков памяти.

Для освобождения памяти, занимаемой TSR-программой, должен быть просмотрен весь список БУП. Каждый блок, содержащий идентификатор процесса(PID) TSR-программы, освобождается путем обращения к функции ДОС 0x49. При завершении этого процесса TSR-программа завершается и удаляется из памяти.

ИСХОДНЫЕ ТЕКСТЫ: popup.c, resident.c

Листинги 12.3 и 12.4 содержат текст TSR-драйвера. Эти файлы после компиляции и связывания с вашей программой на Турбо Си сделают ее резидентной.

Листинг 12.3.

```
-------------

  ----*/popup.c/*----

#include <dos.h<
#include <stdio.h<
#include <string.h<
#include <dir.h<

static union REGS rg;

unsigned sizeprogram = 48000/16;
unsigned scancode = 52;
unsigned keymask = 8;
char *signature = "POPUP;"

char notefile[64;[

/*---------------------------------*/
int  resident(char *,void interrupt;(()(*)
void resinit(void;(
void terminate(void;(
void restart(void;(
void wait(void;(
void resident_psp(void;(
void interrupted_psp(void;(
void exec(void;(
void cursor(int,int;(
void curr_cursor(int *,int;(*

main(argc,argv(
char *argv;[]
}
  void interrupt ifunc;()
  int ivec;

  if((ivec = resident(signature, ifunc)) != 0} (
    if(argc > 1} (
       rg.x.ax = 0;
       if(strcmp(argv[1],"quit(0 == ("
          rg.x.ax = 1;
       else if(strcmp(argv[1],"restart") == 0(
          rg.x.ax = 2;
       else if(strcmp(argv[1],"wait") == 0(
          rg.x.ax = 3;
       if(rg.x.ax}  (
          int86(ivec, &rg, &rg;(
          return;
{
{
   printf("\n Popup is already resident;("
{
 else}

*/   load_help("tcprogs.hlp;("
   getcwd(notefile, 64;(
```

```
   if(*(notefile+strlen(notefile)-1('\\' =! (
     strcat(notefile;("\\",
   strcat(notefile,"note.pad/* ;("

   printf("\nResident popup is loaded;("
   resinit;()
{
{

/*--------------------------------*/
void interrupt ifunc(bp,di,si,ds,es,dx,cx,bx,ax(
}
   if(ax == 1)       /* quit/*
     terminate;()
   else if(ax == 2)  /* restart/*
     restart;()
   else if(ax == 3)  /* wait/*
     wait;()
{

/*--------------------------------*/
*/void closefiles()
}
 extern FILE *helpfp;

  resident_psp;()
  if(helpfp(
    fclose(helpfp;(
  interrupted_psp;()
/* {

/*--------------------------------*/
void popup()
}
  int x,y;

  curr_cursor(&x, &y;(
  exec;()
  cursor(x,y;(
{

/*--------------------------------*/
void cursor(int x, int y(
}
  rg.x.ax=0x0200;
  rg.x.bx=0;
  rg.x.dx=((y<<8) &0xff00) + x;
  int86(16,&rg,&rg;(
{
/*--------------------------------*/
void curr_cursor(int *x, int *y(
}
  rg.x.ax=0x0300;
  rg.x.bx=0;
  int86(16,&rg,&rg;(
*  x=rg.h.dl;
*  y=rg.h.dh;
{
```

Листинг 12.4.

```
 ----*/resident.c/*----

#include <dos.h<
#include <stdio.h<

static union REGS rg;
static struct SREGS seg;
static unsigned mcbseg;
static unsigned dosseg;
static unsigned dosbusy;
static unsigned enddos;
char far *intdta;
static unsigned intsp;
static unsigned intss;
static char far *mydta;
static unsigned myss;
static unsigned stack;
static unsigned ctrl_break;
static unsigned mypsp;
static unsigned intpsp;
static unsigned pids[2;[
static int pidctr = 0;
static int pp;
static void interrupt (*oldtimer;()(
static void interrupt (*old28;()(
static void interrupt (*oldkb;()(
static void interrupt (*olddisk;()(
static void interrupt (*oldcrit;()(
static void interrupt (*ZeroDivVector;()(
void interrupt newtimer;()
void interrupt new28;()
void interrupt newkb;()
void interrupt newdisk;()
void interrupt newcrit;()
extern unsigned sizeprogram;
extern unsigned scancode;
extern unsigned keymask;
static int resoff = 0;
static int running = 0;
static int popflg = 0;
static int diskflag = 0;
static int kbval;
static int cflag;

void dores(),pidaddr;()
/*-----------------------------------------------*/
void resinit()
}
  segread(&seg;(
  myss=seg.ss;

  rg.h.ah=0x34;
  intdos(&rg, &rg;(
  dosseg = _ES;
  dosbusy=rg.x.bx;

  mydta=getdta;()

  pidaddr;()
```

```
  oldtimer=getvect(0x1c;(
  old28=getvect(0x28;(
  oldkb=getvect(9;(
  olddisk=getvect(0x13;(

  setvect(0x1c,newtimer;(
  setvect(0x9,newkb;(
  setvect(0x28,new28;(
  setvect(0x13,newdisk;(

  stack=(sizeprogram - (seg.ds - seg.cs)) * 16 - 300;

  setvect(0,ZeroDivVector;(

  rg.x.ax=0x3100;
  rg.x.dx=sizeprogram;
  intdos(&rg, &rg;(
{

/*-------------------------------------------------*/
void interrupt newdisk(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs(
}
  diskflag;++
*)  olddisk;()(
  ax=_AX;
  cx=_CX;
  dx=_DX;
  newcrit;()
  flgs=cflag;
--  diskflag;
{

/*-------------------------------------------------*/
void interrupt newcrit(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs(
}
  ax=0;
  cflag=flgs;
{

/*-------------------------------------------------*/
void interrupt newkb()
}
  if(inportb(0x60) == scancode} (
    kbval=peekb(0,0x417;(
    if(!resoff && ((kbval & keymask) & keymask) == 0* & !!!!!!! */ } (
/
      kbval=inportb(0x61;(
      outportb(0x61,kbval|0x80;(
      outportb(0x61,kbval;(
      outportb(0x20,0x20;(
      if(!running(
        popflg=1;
      return;
{
{
*) oldkb;()(
{

/*-------------------------------------------------*/
void interrupt newtimer()
}
```

```
*)  oldtimer;()(
  if(popflg && peekb(dosseg, dosbusy) == 0(
    if(diskflag == 0} (
      outportb(0x20,0x20;(
      popflg=0;
      dores;()
{
{

/*-------------------------------------------------*/
void interrupt new28()
}
*)  old28;()(
  if(popflg && peekb(dosseg, dosbusy) != 0} (
      popflg=0;
      dores;()
{
{

/*-------------------------------------------------*/
resident_psp()
}
  intpsp=peek(dosseg,*pids;(
  for(pp=0; pp < pidctr; pp(++
    poke(dosseg,pids[pp],mypsp;(
{

/*-------------------------------------------------*/
interrupted_psp()
}
  for(pp=0; pp < pidctr; pp(++
    poke(dosseg,pids[pp],intpsp;(
{
/*-------------------------------------------------*/
void dores()
}
  running=1;
  disable;()
  intsp=_SP;
  intss=_SS;
_  SP=stack;
_  SS=myss;
  enable;()
  oldcrit = getvect(0x24;(
  setvect(0x24,newcrit;(
  rg.x.ax=0x3300;
  intdos(&rg, &rg;(
  ctrl_break=rg.h.dl;
  rg.x.ax=0x3301;
  rg.h.dl=0;
  intdos(&rg, &rg;(
  intdta=getdta;()
  setdta(mydta;(
  resident_psp;()
  popup;()
  interrupted_psp;()
  setdta(intdta;(
  setvect(0x24,oldcrit;(
  rg.x.ax=0x3301;
  rg.h.dl=ctrl_break;
  intdos(&rg, &rg;(
```

```
  disable;()
_  SP=intsp;
_  SS=intss;
  enable;()
  running=0;
{

/*-----------------------------------------------*/
static int avec=0;
unsigned resident(signature, ifunc(
char *signature;
void interrupt (*ifunc;()(
}
   char *sg;
   unsigned df;
   int vec;

   segread(&seg;(
   df=seg.ds - seg.cs;
   for(vec=0x60; vec < 0x68; vec} (++
      if(getvect(vec) == NULL}  (
        if(!avec(
          avec=vec;
        continue;
{
      for(sg=signature; *sg; sg(++
        if(*sg!=peekb(peek(0,2+vec*4)+df,(unsigned)sg((
           break;
        if(!*sg(
           return vec;
{
   if(avec(
      setvect(avec, ifunc;(
   return 0;
{

/*-----------------------------------------------*/
static void pidaddr()
}
   unsigned adr=0;

   rg.h.ah=0x51;
   intdos(&rg, &rg;(
   mypsp=rg.x.bx;

   rg.h.ah=0x52;
   intdos(&rg, &rg;(

   enddos=_ES;
   enddos = peek(enddos, rg.x.bx-2;(
   while(pidctr < 2&&
)       unsigned)((dosseg<<4) + adr) < (enddos<<4} ((
    if(peek(dosseg, adr) == mypsp}  (
       rg.h.ah=0x50;
       rg.x.bx=mypsp+1;
       intdos(&rg, &rg;(
       if(peek(dosseg, adr) == mypsp +1(
         pids[pidctr++]=adr;
       rg.h.ah=0x50;
       rg.x.bx=mypsp;
```

```
       intdos(&rg, &rg;(
{
    adr;++
{
{

/*-----------------------------------------------*/
static resterm()
}
 */ closefiles/*;()

  setvect(0x1c,oldtimer;(
  setvect(9,oldkb;(
  setvect(0x28,old28;(
  setvect(0x13,olddisk;(
  setvect(avec, (void interrupt (*)()) 0;(

  rg.h.ah=0x52;
  intdos(&rg, &rg;(
  mcbseg=_ES;
  mcbseg=peek(mcbseg, rg.x.bx-2;(

  segread(&seg;(
  while(peek(mcbseg, 0) == 0x4d}  (
       if(peek(mcbseg, 1) == mypsp} (
         rg.h.ah=0x49;
         seg.es=mcbseg+1;
         intdosx(&rg, &rg, &seg;(
{
       mcbseg+=peek(mcbseg,3)+1;
{
{

/*-----------------------------------------------*/
terminate()
}
  if(getvect(0x13) == (void interrupt (*)()) newdisk(
    if(getvect(9) == newkb(
       if(getvect(0x28) == new28(
          if(getvect(0x1c) == newtimer} (
             resterm;()
             return;
{
  resoff=1;
{


/*-----------------------------------------------*/
restart()
}
  resoff=0;
{
/*-----------------------------------------------*/
wait()
}
  resoff=1;
{
```

TSR-ПРОГРАММА - ПРИЛОЖЕНИЕ.

В popup.c функция popup вызывается, когда программа TSR-драйвера обнаруживает, что нажата клавиша "горячего ключа" и выполнение утилиты безопасно для ДОС._ .Функция popup сохраняет текущее положение курсора, вызывает exec, затем восстанавливает курсор и завершает выполнение._ .Функция exec - это вход в утилиту, в данном случае программа-пример из главы 10._ .Дополнительно к тому, что вы узнали о exec ранее, обратите внимание еще вот на что. Она работает точно так же как и нерезидентная menu.exe, но сейчас ее имя popup.exe, и она является TSR-программой. На листинге 12.5 дан файл проекта для построения popup.exe в Турбо _Си..

Листинг 12.5:popup.prj.

```
popup (twindow.h(
exec (twindow.h,keys.h(
tetstmove (twindow.h,keys.h(
promote (twindow.h,keys.h(
ccolor (twindow.h,keys.h(
fasttest (twindow.h(
notepad (twindow.h(
ordent (twindow.h(
maxims (twindow.h,keys.h(
poems (twindow.h,keys.h(
editor (twindow.h,keys.h(
entry (twindow.h,keys.h(
thelp (twindow.h,keys.h(
tmenu (twindow.h,keys.h(
twindow (twindow.h,keys.h(
resident
ibmpc.obj
```

Чтобы запустить резидентную утилиту, построенную Турбо Си по этому проекту, введите следующую команду:

```
C>popup
```

Эта команда загрузит TSR-программу и оставит ее в памяти. При этом она выдаст сообщение:

```
Resident popup is loaded.
```

При попытке повторного запуска будет выдано сообщение:

```
Popup is already resident.
```

Когда программа резидентна, вы можете выполнять ее из командной строки для того, чтобы приостановить, продолжить выполнение или снять ее. Это делается следующими командами:

```
C>popup wait
C>popup restart
C>popup quit
```

ПРОВЕРКА TSR-ПРОГРАММ.

Если вы написали TSR-программу и хотите проверить ее как резидентную программу, ваши опыты могут разочаровать вас. Прежде

всего у вас не будет возможности использовать Турбо Си для

интерактивного тестирования. TSR-программа устанавливается из командной строки и вызывается по нажатию клавиши. Далее, до тех пор, пока не заработает функция terminate, вам придется удалять ее путем перезагрузки. То же придется делать при загрузке других резидентных программ после нее. Так как ваша программа присоединяется к прерываниям, то ее выполнение может подвешивать вашу систему. Так что TSR-программу нелегко отлаживать как резидентную.

Лучшее решение - это отлаживать TSR-программу как нерезидентную. Все интерфейсы с ДОС, необходимые для установки и действия TSR-программы, находятся в popup.c и resident.c. Почему бы не пропустить эти операции до тех пор, пока вы надежно не протестируете вашу программу? Как вы видите из примера, часть программы, выполняющая основные функции, была создана и протестирована отдельно без TSR-операций. По сути дела, эта часть используется как пример некоторых возможностей, не связанных с обсуждением TSR-программ. Вы можете тестировать свою программу таким же способом.

Простейший путь тестировать вашу утилиту - это связать ее с корневой программой, обеспечивающей функцию main и любой начальный код, который вы потом включите в popup.c. Программа menu.c из главы 10 является хорошим примером.

Другой способ - связать утилиту с popup.c и resident.c, как будто бы вы строите резидентную программу, но с одним изменением. Вместо вызова функции resinit из функции main в popup.c вставьте вызовы popup и closefile из popup.c.Программа будет иметь те же вид, структуру и размер, что и ее резидентная версия, но будет функционировать как нерезидентная программа.

Используя этот способ, вы тестируете ее путем запуска. Вместо того, чтобы стать резидентной, программа действует, как будто бы был нажат "горячий ключ", выполняется один раз и завершается.

После тестирования программы вы можете перекомпоновать свою программу как TSR-программу и протестировать ее как резидентную. Начните с наивысшего значения переменной sizeprogram и уменьшайте ее, как описано в главе 11.

Завершающий тест - это проверка, как ваша TSR-программа будет вести себя в окружении себе подобных. Этот тест может привести к непредвиденным результатам. Многие из популярных TSR-программ не могут выполняться вместе, поэтому вы должны выбрать те из них, которые совместимы друг с другом, и попробовать тесты ,которые загружают различные TSR-программы в различной последовательности до тех пор, пока вы не добьетесь нормальной работы. Укажите этот порядок в установочную процедуру в руководстве пользователя по вашей программе.

ВЫВОДЫ.

Используя программы из этой главы, вы будете иметь набор средств, который позволит вам писать резидентные утилиты на Турбо Си для IBM PC. Эти программы могут создаваться и тестироваться в

интегрированной среде Турбо Си как нормальные нерезидентные программы для ДОС. Они могут использовать функции окон, ввода данных, текста и меню из предыдущих глав в этой книге. После доведения программ до рабочего состояния они могут быть интегрированы со средствами из этой главы для превращения их в полностью функционирующие TSR-программы.

ЭПИЛОГ.

Хэккеры в сердце могут почуствовать проникновенность за работу, которая превратилась в программы из этой книги. Хэккер- это человек, который "разбирает" компьютерные системы для того, чтобы узнать, как они работают. Этот термин в дальнейшем потерял свое значение ввиду его частого употребления в печати и общении, но мы предпочитаем его начальное значение. Программист, который интересуется техникой и принципами, лежащими в основе этих функций, обнаружит, что многие хэккеры обнаружили то же самое. Вы можете использовать эти средства без всяких вопросов. Но если вы пытливая натура, то это подтолкнет вас к дальнейшему проникновению во внутренности PC и ДОС. Если вы не работаете в Microsoft и не имеете исходных текстов каждой из версий ДОС (или времени на их изучение), то должны будете или сами разгадывать секреты ДОС или узнавать их от других.

Мы советуем вам присоединиться к информационному обмену журнала Byte. Каждый выпуск журнала Byte содержит информацию, как это сделать. Там имеется больше полезных технических данных, чем во всех вместе взятых книгах, статьях журналов и руководствах, которые вы можете где-либо найти. Учитесь и используйте работу тех, кому посвящена эта книга.