

Е.Н. ВЕСЕЛОВ

Интегрированная
система
МАСТЕР-
для
ПЭВМ



Интегрированная система МАСТЕР также относится к этому новому классу программного обеспечения. В ней реализованы большинство из тех возможностей, которые успели стать традиционными для интегрированных систем, а также некоторые новые архитектурные и технические решения. К числу традиционных относятся такие компоненты МАСТЕРа, как текстовый редактор, процессор электронных таблиц, средства деловой и иллюстративной графики, отличающиеся от широко распространенных аналогичных средств разве лишь некоторыми полезными техническими деталями. Наибольшие отличия имеются в инструментальном языке и в архитектуре системы управления базами данных (СУБД) МАСТЕРа. Инструментальные возможности системы являются более гибкими и полными по сравнению с другими интегрированными системами, что существенно упрощает создание с ее помощью прикладных информационных систем и баз данных. Важным отличием технического характера является изначальная русифицированность системы, что весьма существенно для ее отечественного применения. Следует отметить также и то, что МАСТЕР является лицензионно чистым программным продуктом, поскольку для его реализации не только не использовались какие бы то ни было пакеты, но и не существует никакого прямого зарубежного аналога.

К данной книге не следует относиться как к общему исследованию возможностей интегрированных систем. Книга представляет собой описание конкретной интегрированной системы МАСТЕР (версия 1.300) и ориентирована на ее потенциальных пользователей. Первая часть книги предназначена для тех пользователей, которые собираются применять МАСТЕР без дополнительных настроек, как готовую информационную среду. Вторая часть предназначена для прикладного программиста, собирающегося использовать МАСТЕР в качестве инструмента для разработки прикладных систем. В приложениях содержится справочная информация, облегчающая использование системы. Приложение А представляет перечень встроенных функций языка Мастер, приложение Б - перечень встроенных констант, приложение В - таблицу кодировки символов, приложение Г - таблицу нот, приложение Д - словарь терминов.

Автор глубоко признателен С. Г. Сироткину, вложившему большой и квалифицированный труд в разработку и реализацию СУБД МАСТЕРа, Д. С. Хлебникову, В. П. Мазурику, Ю. Г. Евтушенко, В. М. Брябину, А. Б. Борковскому, А. А. Чижову, влиявшим на работу своими идеями и оценками, а также Е. П. Веселовой за организационную помощь в разработке системы и написании книги.

Автор

ВВЕДЕНИЕ В ИНТЕГРИРОВАННЫЕ СИСТЕМЫ

Автоматизация учрежденческой деятельности

Характерным явлением современной информационной технологии в последнее время стало то, что прямой доступ к компьютеру получили люди самых разнообразных невычислительных специальностей, занимающиеся планированием, принятием решений, управлением, учетом документов и т.д. Существенно то, что компьютеры используются работниками учреждения без каких-либо посредников в лице персонала вычислительного центра. Это явление получило название **автоматизации учрежденческой деятельности**.

Задачи, которые при этом возлагаются на компьютеры, имеют преимущественно невычислительный характер. Некоторые из этих задач, такие, как редактирование текстов, в той или иной форме решались на компьютерах и прежде, но они использовались не конечными пользователями, а программистами и рассматривались как вспомогательные, только лишь поддерживающие основной процесс — создание алгоритмов и производство разнообразных расчетов. Теперь же редактированием текстов на компьютерах занимаются все — от машинисток до руководителей. Возникли и новые функции, такие, как построение и анализ графиков и схем, автоматизация финансовых расчетов, обработка структурных документов, внутренняя и внешняя информационная связь, хранение и сортировка больших массивов информации, автоматическая генерация отчетов.

Средства автоматизации

Функции, выполнявшиеся прежде только на бумаге, могут быть теперь перенесены на экраны персональных компьютеров. На рис. 1.1, 1.2, 1.3 показано, как выглядят на экране во время работы тексты, графики, финансовые табличные документы.

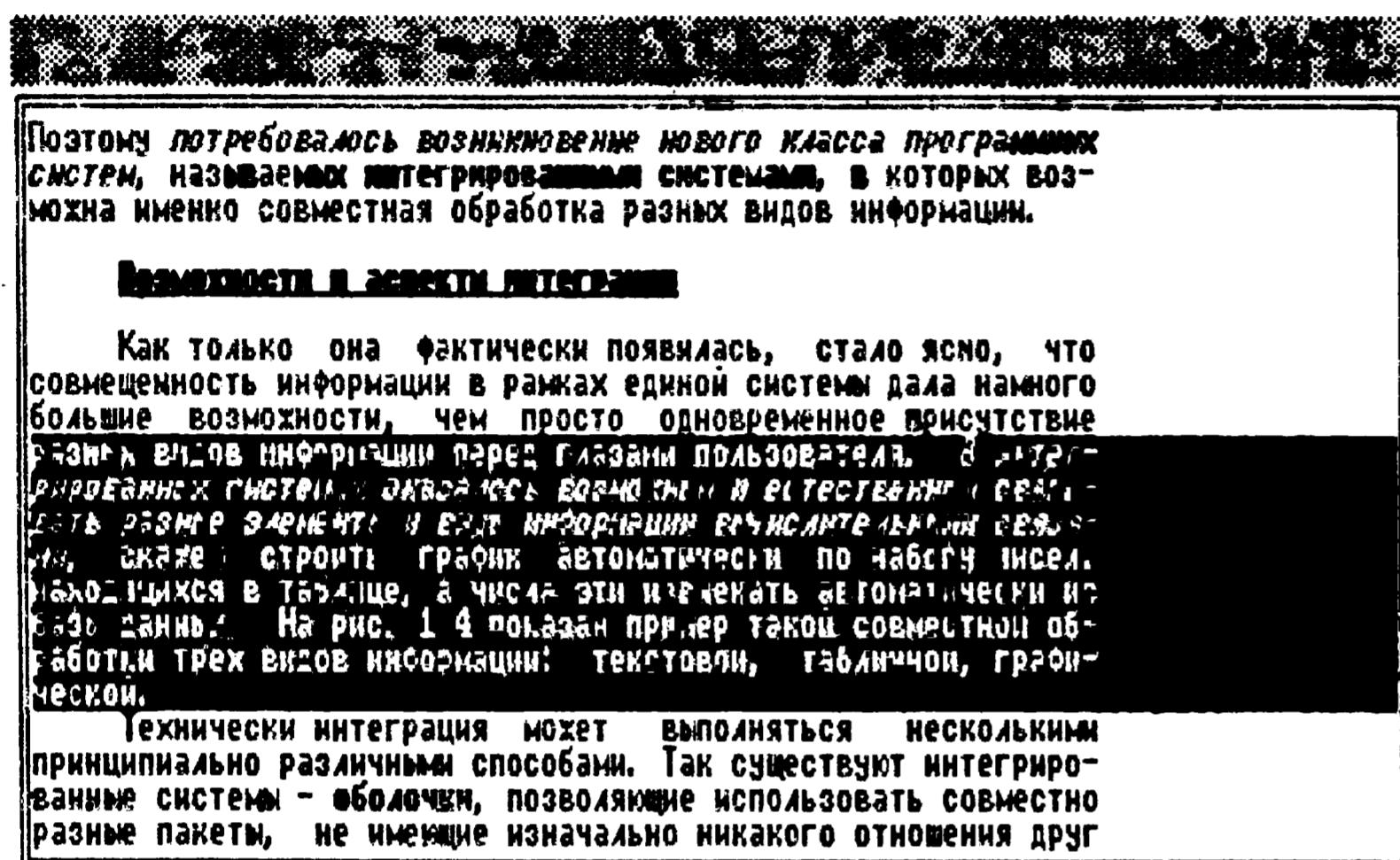


Рис. 1.1. Текстовый документ на экране дисплея

Эти функции получили очень мощную поддержку как аппаратными, так и программными средствами. Аппаратная поддержка выразилась в появлении, прежде всего, персональных компьютеров – ЭВМ нового поколения, отличающихся малыми габаритами, малой потребляемой мощностью, малой ценой, возможностью установки на обычном рабочем столе, но обладающих, в сущности, теми же возможностями в отношении памяти, скорости, выполняемых функций, что и средние и даже большие ЭВМ.

Помимо чисто количественных и габаритных показателей компьютеры нового поколения имеют принципиально новую архитектуру, в особенности в отношении средств визуализации данных и осуществления диалога с пользователем. Новые возможности ПЭВМ проявляются в использовании графических экранов, позволяющих обрабатывать одновременно и символы, и графические изображения; в мгновенном формировании и изменении изображений на экране за счет размещения памяти экрана непосредственно в оперативной памяти процессора; в развитии

компактных и точных матричных и лазерных печатающих устройств; в изобретении простого, но необыкновенно удобного манипулятора, названного "мышь", избавившего пользователя от необходимости запоминать функциональные клавиши на клавиатуре и заменившего их простым указанием экранных объектов. Здесь перечислены лишь те аппаратные возможности, с которыми непосредственно сталкивается пользователь.

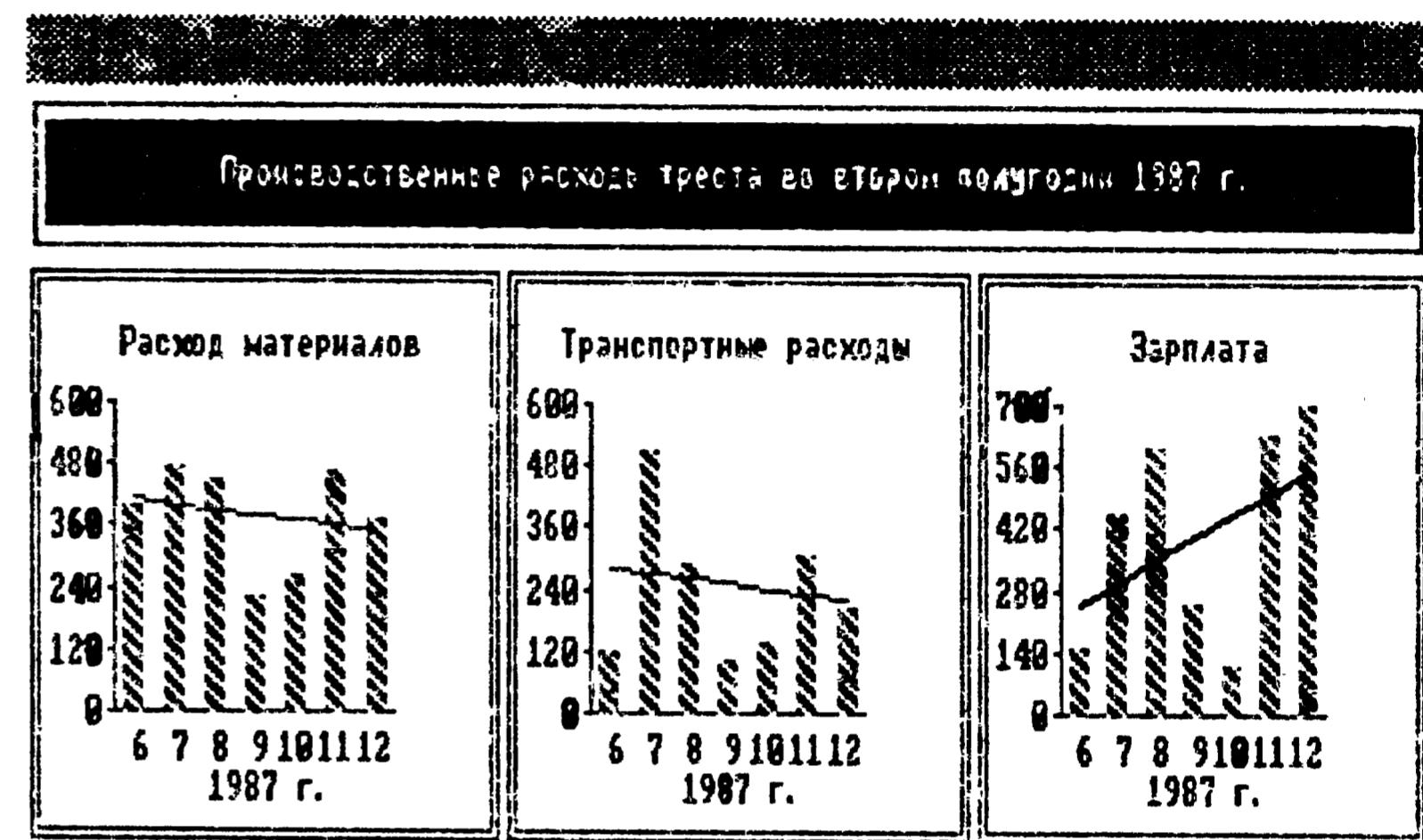


Рис. 1.2. Графическая информация на экране дисплея

Столь же бурным было развитие программных средств, отвечающих новым аппаратным возможностям и функциональным потребностям. Независимо от их назначения новые программные средства объединяет то, что все они качественно по-новому предъявляются пользователю. Это не языки программирования и не пакеты процедур для каких-то языков, это *интерактивные среды*, погружаясь в которые пользователь получает поддержку в выполнении определенной диалоговой функции, в осуществлении действий над информацией. Большинство этих программных средств по сравнению с "бумажной" технологией дают пользователю не *новый вид* информации, а всего лишь *новый способ представления* того или иного привычного вида информации: тексты, таблицы, графики, картотеки представляются в этих системах на другом носителе – не на бумажном, а на электронном. Этот электронный носитель, предназначенный для тех же самых функций, что и бумага, обладает принципиально иными свойствами, проявляющимися в работе. Например, в текстовом процессоре, у пользователя создается иллюзия, что экран –

это лист бумаги с напечатанным на нем текстом. Однако, в отличие от бумажного листа, для вставки слова в середину абзаца не нужно перепечатывать весь лист: электронная "бумага" легко раздвигается, слова автоматически переставляются внутри абзаца, точно так же автоматически переформируются страницы, легко сделать замену по всему тексту одного слова на другое и т.п.

	Январь	Февраль	Март	Апрель	
Продажи	20000.00	20200.00			
Стоимость			Март	Апрель	Май
Материалы	4000.00	4032.00			
Трудозатраты	7000.00	7000.00			
Накладные	4000.00	4032.00			
Итого	15000.00	15064.00			
Прибыль	5000.00	5136.00			

Рис. 1.3. Электронная таблица

Новые программные средства, которые часто называют функциональными пакетами программ, основываются на различных метафорах носителей информации. Для обработки текстов существует метафора бумажного листа, для графиков и рисунков – метафора холста и палитры, для таблиц – метафоры бланков и форматок. Различаясь в конкретных деталях метафор, функциональные пакеты едины в главном: они заменяют бумажный носитель информации электронным, сохраняя всю наглядность бумажного листа и придавая ему невозможную для бумаги пластичность в отношении внесения изменений. Эффект от таких новых свойств тем заметнее, чем больше доля информации, не подлежащей изменению, по отношению к добавляемой или изменяемой информации. Именно таким свойством обладает информация в учрежденческой деятельности, где приходится иметь дело с бланками стандартного вида, однотипными и повторяющимися документами. Количество различных функциональных пакетов на ранке персональных компьютеров к настоящему времени стало

очень большим. По выполняемым функциям их можно свести к следующим основным группам:

- текстовая обработка;
- обработка таблиц;
- обработка графиков;
- обработка рисунков;
- управление локальной базой данных;
- осуществление связи с удаленными абонентами или информационными системами.

Возможности и аспекты интеграции

По мере того как в новых программных средствах возникали новые возможности, все острее выступала общая проблема их интеграции: как совместно использовать различные функциональные средства, предоставляемые разными пакетами. В практической работе недостаточно, чтобы функциональные возможности всего лишь существовали. Требуется, чтобы они были интегрированы, т.е. хорошо сочленены между собой. Но интеграция не возникает сама по себе. Наоборот, информация все время стремится оказаться несовместимой либо по способу представления, либо по способу обработки, либо по способу визуализации на экране, либо по физической одновременной доступности, либо еще по каким-нибудь неожиданным причинам. Поэтому потребовалось возникновение нового класса программных систем, называемых **интегрированными системами**, в которых возможна именно совместная обработка разных видов информации.

Как только появились первые интегрированные системы, оказалось, что совмещение информации в рамках единой системы дала намного большие возможности, чем просто одновременное присутствие разных видов информации перед глазами пользователя. В интегрированных системах из-за объединения разных возможностей возникает сразу несколько новых существенных качеств.

Во-первых, интеграция в системе МАСТЕР позволяет объединить различные типы информации: числа, строки, тексты, графики, рисунки, табличные структуры, составные иерархические структуры, реляционные и сетевые структуры в базе данных. Это объединение позволяет поместить разнородную информацию в единую память и тем самым дать одновременный доступ ко всем типам. Но самое главное в этом объединении то, что оно позволяет создавать новые структурные качества информации, отражающие специфические потребности решаемой задачи. Так например, комплексный документ, включающий одновременно

текстовые, табличные и графические части, является принципиально более богатым по содержанию, нежели чистый текст, чистая таблица или отдельный рисунок. В базе данных МАСТЕРа может храниться структурированная смесь числовой, строковой, текстовой, графической информации, позволяющая всесторонне представлять описываемую прикладную область.

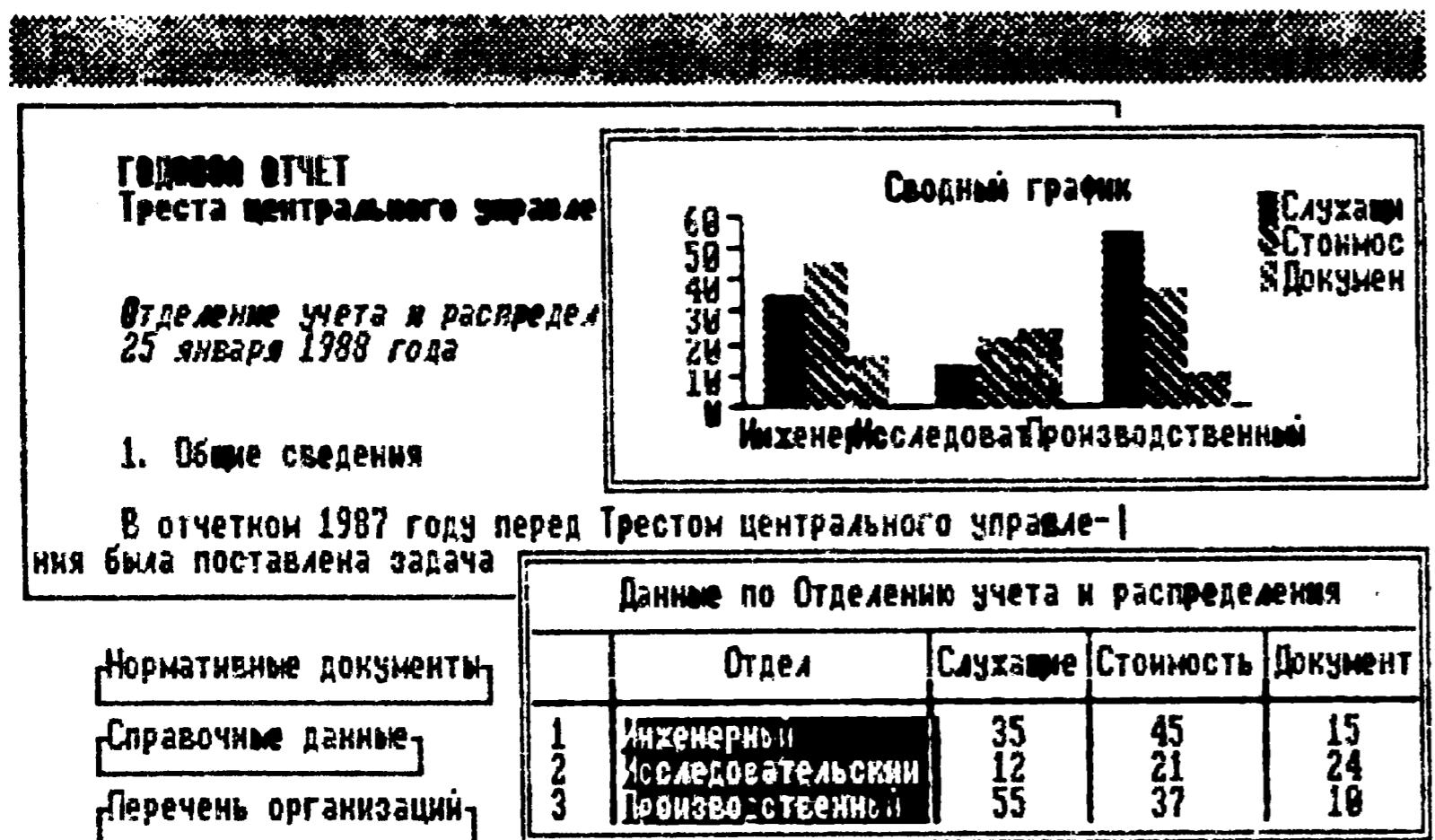


Рис. 1.4. Интегрированная обработка информации

Во-вторых, интеграция в МАСТЕРЕ проявляется в комплексной визуализации разных типов информации. Для этого используются рамки разных типов, через которые, как через окна, видна хранящаяся в них информация, и которые могут одновременно присутствовать на экране в любом взаимном расположении. На рис. 1.4 показан пример экрана, содержащего одновременно три рамки: текстовую, табличную и графическую. Одновременность визуализации данных разных типов очень важна при практической работе. В качестве примера лучше всего подходит изображение числовых массивов в табличной и графической формах. Вводя данные и выполняя вычисления над ними в числовой форме, пользователь может немедленно видеть намного более наглядное представление тех же величин в форме графиков, гистограмм, диаграмм.

В-третьих, МАСТЕР объединяет несколько диалоговых функций: редактирование текстов, работу с таблицами, построение рисунков и графиков, работу с иерархической структурой многокомпонентных документов, работу со множествами записей базы данных. Разные диалоговые функции применимы к разным ти-

пам информации, содержащимся в различных рамках. Поскольку все типы информации представлены в единой памяти и визуализированы на экране одновременно, то переключение с одной диалоговой функции на другую (например, с редактирования текста на редактирование рисунка) может быть сделано практически мгновенно: достаточно лишь переместить курсор по экрану из текстовой рамки в рисунковую.

Наконец, в-четвертых, интеграция в МАСТЕРЕ завершается использованием встроенного языка программирования, с помощью которого можно выполнить функциональную настройку созданной интегрированной структуры, т.е. задать вычислительные связи между элементами данных, доопределить или переопределить действия функциональных клавиш, сформулировать сложные запросы к базе данных, запрограммировать алгоритмы автоматизированной обработки данных. Формулы записываются на встроенном в систему языке программирования, в котором имеются все требуемые для интеграции средства: как операции для обработки каждого типа информации в отдельности, так и операции для их совместной обработки, для связывания друг с другом, для преобразования друг в друга. В языке имеются также и средства вычислительного характера, и средства организации диалога, и средства управления базой данных.

На перечисленных четырех аспектах интеграции хотелось бы особо остановить внимание читателя. Именно единство этих четырех аспектов составляет основную особенность обработки информации в интегрированных системах и, в частности, в МАСТЕРЕ. За каждым понятием интегрированной технологии следует стремиться различать:

- какой тип информации оно представляет;
- как эта информация выглядит внешне;
- какими диалоговыми функциями можно редактировать эту информацию вручную;
- какими программными средствами ее можно преобразовывать автоматически.

Уровни использования интегрированной системы

В отличие от функциональных пакетов программ, каждый из которых предполагает какую-то одну категорию пользователей, интегрированная система может использоваться несколькими принципиально различными категориями пользователей и представляется для каждой из них соответственно по-разному.

Проще всего использовать МАСТЕР как готовую интерактивную информационную среду для создания и редактирования комплекс-

ных документов. Для такого использования достаточно самых общих навыков работы с текстовыми редакторами и электронными таблицами.

Следующий уровень, требующий несколько больших усилий при обучении, позволяет создавать в МАСТЕРе персональные базы данных типа картотек, записных книжек, каталогов, реестров и т.п. При этом потребуется потратить некоторое время на описание структуры базы данных и на настройку средств интерфейса с ней. После этого можно будет занести в базу большое количество записей, делать выборки и генерировать отчеты по этим записям.

Если информация, подлежащая обработке, обладает более сложной структурой или требует более сложных алгоритмов обработки, то для конечного пользователя было бы слишком обременительно самому строить соответствующую базу данных и писать обрабатывающие программы. Поэтому в данном случае должен подключиться более квалифицированный специалист — прикладной программист. Он должен хорошо владеть инструментальным языком программирования, встроенным в систему МАСТЕР, иметь опыт разработки баз данных. Такой прикладной программист будет использовать МАСТЕР как инструмент, подобный любым другим системам программирования типа Си, Паскаль, Бейсик. База данных с соответствующим пакетом процедурной поддержки, которую он разработает с помощью этого инструмента, будет называться прикладной системой и может использоваться конечным пользователем с очень специализированной ориентацией.

Итак, для интегрированной системы МАСТЕР следует рассматривать пользователей трех категорий:

- конечные пользователи универсальной ориентации, работающие с МАСТЕРом как с готовой информационной средой;
- прикладные программисты, разрабатывающие специализированные информационные системы на языке, встроенном в систему;
- конечные пользователи, работающие со специализированными информационными системами, созданными на языке Мастер прикладными программистами.

Средо-ориентированное программирование

Независимо от того, собираетесь ли вы использовать МАСТЕР как готовую информационную среду для непосредственной работы или как инструментальную систему для разработки специализированной прикладной системы, основой для вашей работы всегда

будет совокупность тех базовых интерактивных информационных сред, которые имеются в МАСТЕРЕ изначально: текстовая, табличная, графическая, составная.

Как только вам потребуется перейти к новой части информации, вы можете создать для нее рамку соответствующего типа. Рамка служит не просто емкостью для информации — она является носителем интерактивной среды. Информация, содержащаяся в рамке, видна в ней, как через окно. Если вы "войдете" внутрь рамки, то вас начнет обслуживать диалоговый процессор, связанный с этой рамкой: он выполнит те диалоговые функции, которые требуются для обработки информации данного типа. Так, например, в рамке текстового типа содержится текстовая информация, которая визуализируется в виде текста, а диалоговая функция, доступная в этой рамке, — это экранное редактирование и форматирование текста. В рамке рисункового типа содержится графическая информация, визуализируемая в виде растрового изображения, с соответствующей диалоговой функцией — экранным рисованием.

Мощности диалоговых процессоров, встроенных в МАСТЕР для базовых интерактивных сред, вполне достаточно для выполнения самых разнообразных действий над информацией. С их помощью можно создать сложные документы, редактировать и распечатывать их. Это будет первый, самый простой уровень использования МАСТЕРа.

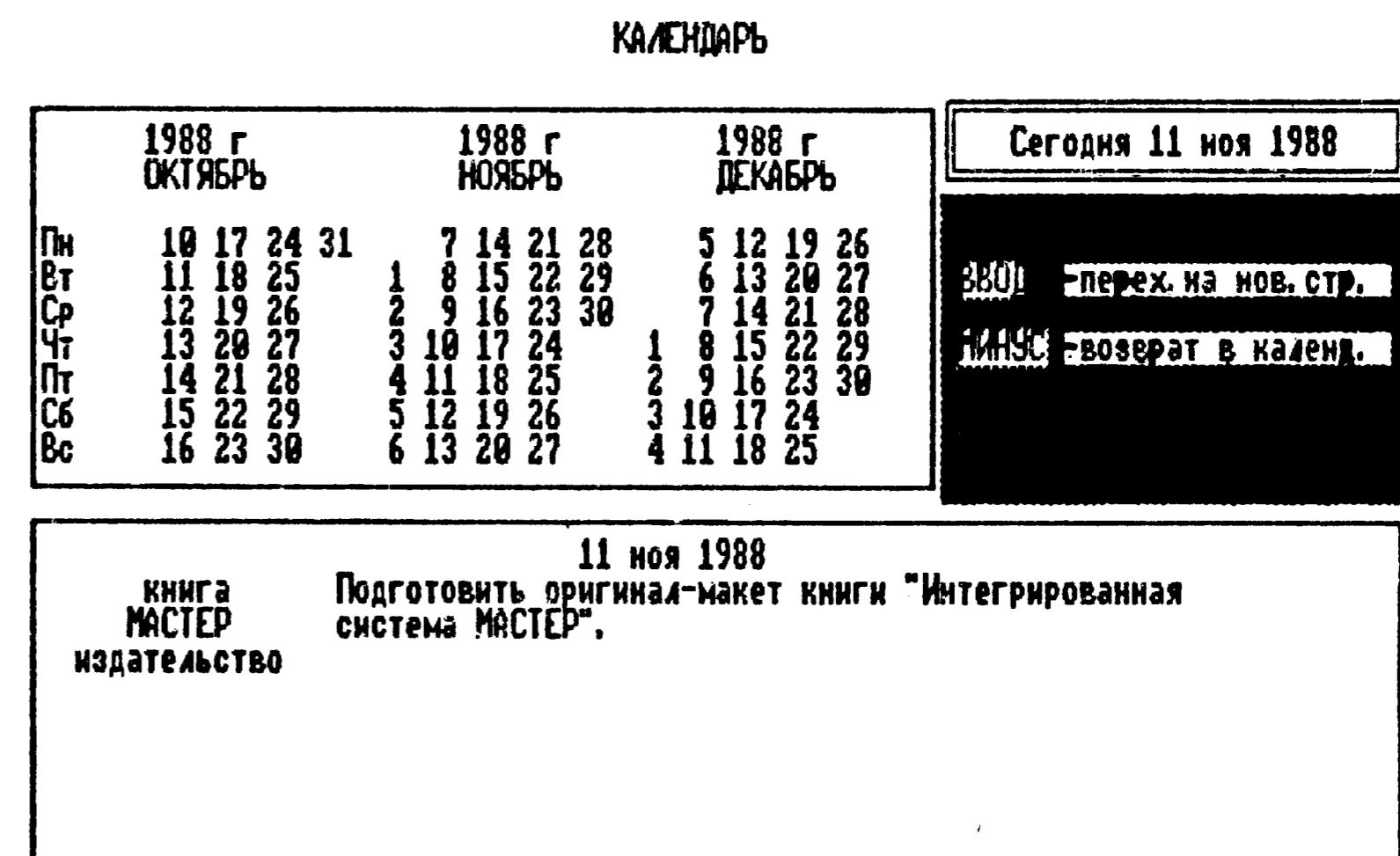


Рис. 1.5. Вид экрана при работе в персональной картотеке

Если вы обращаетесь к МАСТЕРу как прикладной программист, то ваша работа точно так же должна начинаться с создания рамок. Отличие состоит в том, что рамки вы будете создавать уже не для себя, а для своего конечного пользователя, подготавливая ему специализированную информационную среду. Одной только компоновкой соответствующих рамок, без какого бы то ни было программирования, можно создать достаточно сложную по структуре среду, ориентированную на особенности информации в конкретной прикладной области. Однако это будет лишь структурная носитель разрабатываемой интерактивной среды. Ее нужно будет дополнить программированием некоторых функциональных клавиш, добавлением процедур для автоматической обработки данных, формированием программ, реализующих сложные поисковые запросы к базе данных и преобразование записей в ней. Для всего этого придется воспользоваться встроенным языком программирования, называемым Мастер.

На рис. 1.5, например, показан вид экрана во время работы одной из таких специализированных систем — персональной картотеки. Характерной чертой этого примера является то, что экран имеет специализированный вид, в котором отсутствуют какие бы то ни было универсальные элементы МАСТЕРа (скажем, нет стандартного меню МАСТЕРа), вместо них на экране создана форматка, состоящая из текста карточки, нескольких позиций для ключевых слов, таблички-подсказки.

Такой способ создания прикладных систем характерен именно для интегрированной технологии и отличается от использования традиционных языков программирования тем, что язык здесь играет второстепенную роль, а основу разработки составляет непосредственное конструирование интерактивной среды из более простых — базовых или созданных ранее интерактивных сред. Будем называть такой способ разработки систем **средо-ориентированной технологией**. В ней, как и в любой другой технологии, при создании прикладной системы требуется решать следующие задачи:

- структурирование данных;
- визуализацию данных;
- осуществление взаимодействия с пользователем (диалог);
- выполнение конкретных вычислительных или информационно-поисковых действий.

В отличие от традиционной технологии, в которой все четыре задачи решаются только через язык программирования, в средо-ориентированной технологии язык программирования требуется только для последней задачи, в то время как три первые намного более просто и естественно реализуются через непосредственное конструирование интерактивных сред.

Возможности встроенного языка программирования

Если обратиться к самому языку Мастер, то следует отметить прежде всего его функциональную полноту и гибкость. В нем имеется достаточное количество средств для самых разных алгоритмических потребностей: и арифметические операции, и функции для обработки строк, текстов и структур данных, и разнообразные управляющие конструкции (условные операторы, операторы циклов, вызовы встроенных и определяемых функций). Этот язык, однако, отличается и от обычных языков программирования типа Си, Паскаль, Бейсик, и от инструментальных языков, встроенных в другие интегрированные системы.

От обычных процедурно-ориентированных языков программирования Мастер отличается в нескольких отношениях. Наиболее важное отличие уже было рассмотрено. Оно лежит в самой архитектуре интегрированной системы — в ее средо-ориентированном характере. Это приводит к тому, что в языке нет понятия головной программы; ее роль выполняет интерактивная среда, на которую в разных местах нанизываются те или иные формулы и процедуры, обогащающие или изменяющие стандартный операционный характер базовых сред. Второе отличие языка состоит в том, что для его программ имеется простейшая форма, применяемая в тех случаях, когда требуется написать не сложную большую программу, а всего лишь элементарную формулу, например, в ячейке электронной таблицы. Такие простейшие программы на Мастере выглядят как арифметические формулы и доступны для совершенно неподготовленного пользователя, пользующегося ятими формулами без предварительной программистской подготовки.

От языков других интегрированных систем язык Мастер отличается той ролью, которую он играет в интегрированной системе МАСТЕР. Он является не дополнением к базовым диалоговым функциональным возможностям, а основой этих возможностей. Так, например, все варианты меню, и многие функциональные клавиши не встроены фиксированно в МАСТЕР, а запрограммированы на языке Мастер. Эти программы составляют *стандартную диалоговую оболочку* системы и легко могут быть при желании дополнены или изменены. Подобно этой стандартной оболочке прикладной программист может создавать свои собственные диалоговые оболочки, ориентированные на то или иное специализированное приложение. Такая возможность основывается прежде всего на том, что в языке Мастер имеются полные наборы операций для каждого из типов информации, содержащихся в МАСТЕРЕ: не только для чисел, строк и текстов, но и для рамочных структур, табличных структур, структур записей в базах, данных.

Наиболее важными характеристиками языка Мастер представляются следующие:

- язык Мастер обрабатывается интерактивным *интерпретатором*, что позволяет оперативно отлаживать программы, быстро меняя любые подпрограммы и немедленно апробируя работоспособность измененных частей;
- язык Мастер имеет *высокий уровень примитивов*: к "элементарным" типам значений относятся, например, тексты, таблицы, рисунки, множества записей базы данных;
- в языке Мастер имеются примитивы нетрадиционного вида, принципиально отсутствующие в обычных процедурных языках, – *интерактивные среды*, на которых основываются общая организация систем, визуализация данных, диалог с пользователем;
- в язык Мастер встроены мощные и гибкие *средства конструирования и управления базами данных*.

ЧАСТЬ 1

СТРУКТУРА И ВОЗМОЖНОСТИ ДИАЛОГОВОЙ СРЕДЫ

В первой части книги МАСТЕР рассматривается как готовая информационная среда, непосредственно доступная для конечного пользователя. Здесь описываются основные структурные понятия системы, а также встроенные диалоговые процессоры, обеспечивающие работу в интерактивной среде МАСТЕРа: текстовый процессор, процессор электронных таблиц, процессор иллюстративной графики, процессор диалогового доступа к базам данных. Пользуясь одними только этими средствами, можно создавать комплексные документы, небольшие аналитические модели инженерного и экономического характера, редактировать тексты и таблицы, иллюстрируя их графиками и рисунками.

Изучение интерактивных возможностей базовых сред МАСТЕРа необходимо не только конечному пользователю, но и прикладному программисту, собирающемуся использовать МАСТЕР в качестве инструмента, поскольку основой любой прикладной системы, создаваемой на МАСТЕРе, является конструирование определенной интерактивной информационной среды, а все программирование осуществляется уже на ее основе.

Описываемые в данной части книги возможности – это различные меню и функциональные клавиши системы. Все они составляют *диалоговую оболочку* системы. Диалоговая оболочка легко перенастраивается и изменяется в соответствии со спецификой приложения. Конкретная версия МАСТЕРа, поставляемая пользователю, может быть снабжена несколько иной диалоговой оболочкой, детали которой уточняются в документации в комплекте поставки.

Глава 2

ОСНОВНЫЕ ПОНЯТИЯ И ДЕЙСТВИЯ

2.1. Сведения об операционной системе

Интегрированная система МАСТЕР – это программный пакет, работающий на ПЭВМ типа IBM PC XT/AT в среде операционной системы MS DOS. Поэтому прежде чем войти в МАСТЕР и действовать по законам интегрированной интерактивной среды, вы должны выполнить некоторые действия в командной среде операционной системы. С этой средой придется сталкиваться и впоследствии – при работе с дисками, при обращении к прикладным системам. Поэтому мы начнем с краткого ознакомления с основными понятиями среды MS DOS.

Информационная среда операционной системы

Компьютер, с точки зрения пользователя, представляет собой информационную среду, в которой он может хранить и обрабатывать информацию. Для работы необходимо уметь ориентироваться и перемещаться в этой среде, конструировать ее структуру.

Первым структурным понятием информационной среды компьютера является **дисковод** – устройство, осуществляющее доступ к дискам, на которых хранится информация. При обращении к дисководам требуется каким-то образом именовать их, и для этого используются буквы латинского алфавита с двоеточием: A:, B:, C: и т.д.

Информация на дисках хранится в **файлах**. Это слово, обычно непонятное для начинающих пользователей ЭВМ, не означает,

однако, ничего сложного. С английского языка его можно перевести как "папка для бумаг", а в информатике файл означает набор данных (как бы совокупность бумаг), хранящихся под единым именем на диске. Для наглядности представим себе, что диск – это полка, а файлы – это папки с бумагами, разложенные на этой полке.

Для идентификации файлов используются **символические имена файлов**. В операционной системе MS DOS имя файла состоит из двух частей: мнемоники и расширителя. Мнемоника содержит от 1 до 8 букв или цифр и представляет сокращенное обозначение содержимого файла. Расширитель, хотя он и не обязательен, состоит, как правило, из 3 букв, и обозначает тип файла. Например, файлы, порожденные интегрированной системой МАСТЕР, имеют расширитель MAS, а файлы, используемые системой программирования Паскаль, имеют расширитель PAS. Файлы, содержащие программы, готовые к вызову, имеют расширитель EXE или COM. При записи имени файла сначала записывается мнемоника, затем расширитель, а между ними ставится точка. Например, вы можете создать в МАСТЕРе несколько документов и разместить их в файлах с именами "REPORT.MAS", "LIST.MAS", "PASSPORT.MAS".

Диски, в особенности типа "винчестер", обладают очень большой емкостью, и поэтому на них может быть размещено огромное количество (до нескольких тысяч) файлов. Чтобы обилие этих файлов не создавало хаоса на диске, необходимо как-то группировать их подобно тому, как папки с бумагами складываются не в одну стопку, а разносятся по разным отделениям полок и по ящикам столов. Аналогом таких отделений в информационной среде компьютера являются **каталоги** – это такие отделения на диске, внутри которых можно группировать файлы по темам. Подобно тому как отделение шкафа, предназначенное для большого количества бумаг, может иметь деление на более мелкие отделения, точно так же и каталоги на дисках могут содержать не обязательно только файлы, но и в свою очередь другие отделения-каталоги. Таким образом, каталоги образуют иерархическую структуру, завершающуюся файлами с рабочей информацией. Так, например, при подготовке этой книги у автора на компьютере был заведен каталог "BOOK" для всей книги в целом, чтобы отделить ее от прочих файлов, хранящихся на диске. Внутри каталога "BOOK" были созданы два подкаталога "PART1", "PART2", соответствующих двум частям книги, и уже внутри этих каталогов хранились текстовые файлы, называемые "MASTER1.TXT", "MASTER2.TXT", ..., содержащие тексты отдельных глав.

Каталогам, подобно файлам, присваиваются мнемонические имена, обычно не имеющие расширителей. Если вы желаете ука-

зять не только имя файла, но и то место в дереве каталогов и то дисковое устройство, где он находится, то вы должны использовать **полное имя файла**. Оно начинается с указания устройства, затем следует перечень имен каталогов по цепочке от корня дерева до того подкаталога, в котором находится файл; а в конце ставится мнемоника и расширитель имени файла. Между именами каталогов и мнемоникой ставятся символы "\". Так, полным именем для файла, содержащего текст этой главы книги, будет:

C:\BOOK\PART1\MASTER2.TXT

Общение с операционной системой

Мы рассмотрели, как устроена информационная среда, в которой вы будете работать на персональном компьютере. Обратимся теперь к тому, как происходит общение пользователя с этой средой. Общение с ней обеспечивается операционной системой MS DOS, точнее, ее командным интерпретатором. Командный интерпретатор MS DOS – это такая программа, которая с самого начала находится в оперативной памяти компьютера и поддерживает ваше взаимодействие со средой. Когда командный интерпретатор готов принять от вас очередную команду, то он выводит на экран **приглашение к приему** команды. Оно может выглядеть по-разному, в зависимости от настройки операционной системы, но, как правило, имеет следующий вид:

C:\BOOK\PART1>

По своей форме это приглашение совпадает с именем подкаталога "PART1" каталога "BOOK", находящегося на диске C:. Это связано с тем, что при работе в операционной системе MS DOS один из каталогов считается текущим. Следует представлять себе, что вы находитесь в этом каталоге на соответствующем диске. Так, работая с текстами глав первой части книги, удобно войти в подкаталог C:\BOOK\PART1. Это позволит вместо полных имен файлов использовать только их мнемоники, поскольку система будет настроена на текущий каталог. Существуют команды, позволяющие переходить из одного каталога в другой, с одного диска на другой. То, что имена текущего диска и текущего каталога указываются в приглашении к приему команды, достаточно удобно, потому что вы всегда имеете перед глазами идентификацию вашего местонахождения.

Получив от командного интерпретатора приглашение к приему команды, вы можете ввести с клавиатуры текст команды. Выполнение команды начинается при нажатии клавиши {ВВОД}.

Команды операционной системы MS DOS подразделяются на две группы: встроенные и программируемые. Встроенные команды выполняют самые общие действия над файлами и каталогами. Наиболее важные из встроенных команд перечислены в табл. 2.1.

Таблица 2.1. Основные команды операционной системы

Команда	Действие
DIR	Выдает на экран перечень имен файлов и подкаталогов, содержащихся в текущем каталоге
DEL имя файла	Уничтожает файл с указанным именем
COPY откуда куда	Копирует файл из одного места (диска или каталога) в другое: "откуда" должно быть именем файла, а "куда" – именем дискового устройства или каталога
MD имя Каталога	Создает каталог с указанным именем
CD имя Каталога	Выполняет переход в указанный каталог, т.е. делает его текущим
A: B: C:	Выполняет переход на указанный дисковод, т.е. делает его текущим

Программируемые команды реализуются программами на различных языках и представляются на дисках в виде файлов с расширениями EXE или COM. Для вызова таких программ нужно ввести имя соответствующего файла. Так, например, интегрированная система МАСТЕР представлена файлом MASTER.EXE. Поэтому для того чтобы вызвать систему МАСТЕР, вы должны ввести команду

MASTER

Клавиатура

Посредником при вашем общении с компьютером является клавиатура. Устройство это простое, очень похожее на клавиатуру обычной пишущей машинки. Отличие состоит в том, что на клавиатуре, помимо буквенно-цифровых клавиш, имеется еще довольно много функциональных клавиш. Они служат для разного рода операций над данными, для управления компьютером, для указания объектов, подлежащих обработке.

При описании клавиш нам будут требоваться точные обозначения для них. В качестве таких обозначений будем использовать те названия, которые определены для клавиш внутри самой системы МАСТЕР. Эти обозначения представляют собой мнемонические названия, заключенные в фигурные скобки, например {Ф1}, {Ф2}, {ВЛЕВО}, {ВВЕРХ}, {ВВОД}, {ОТКАЗ}. На рис. 2.1 приведено изображение набора клавиш ПЭВМ типа IBM PC XT, и на клавиатах показаны их условные обозначения. Действительные обозначения на клавиатуре могут быть иными, но мы будем использовать только эти условные обозначения.

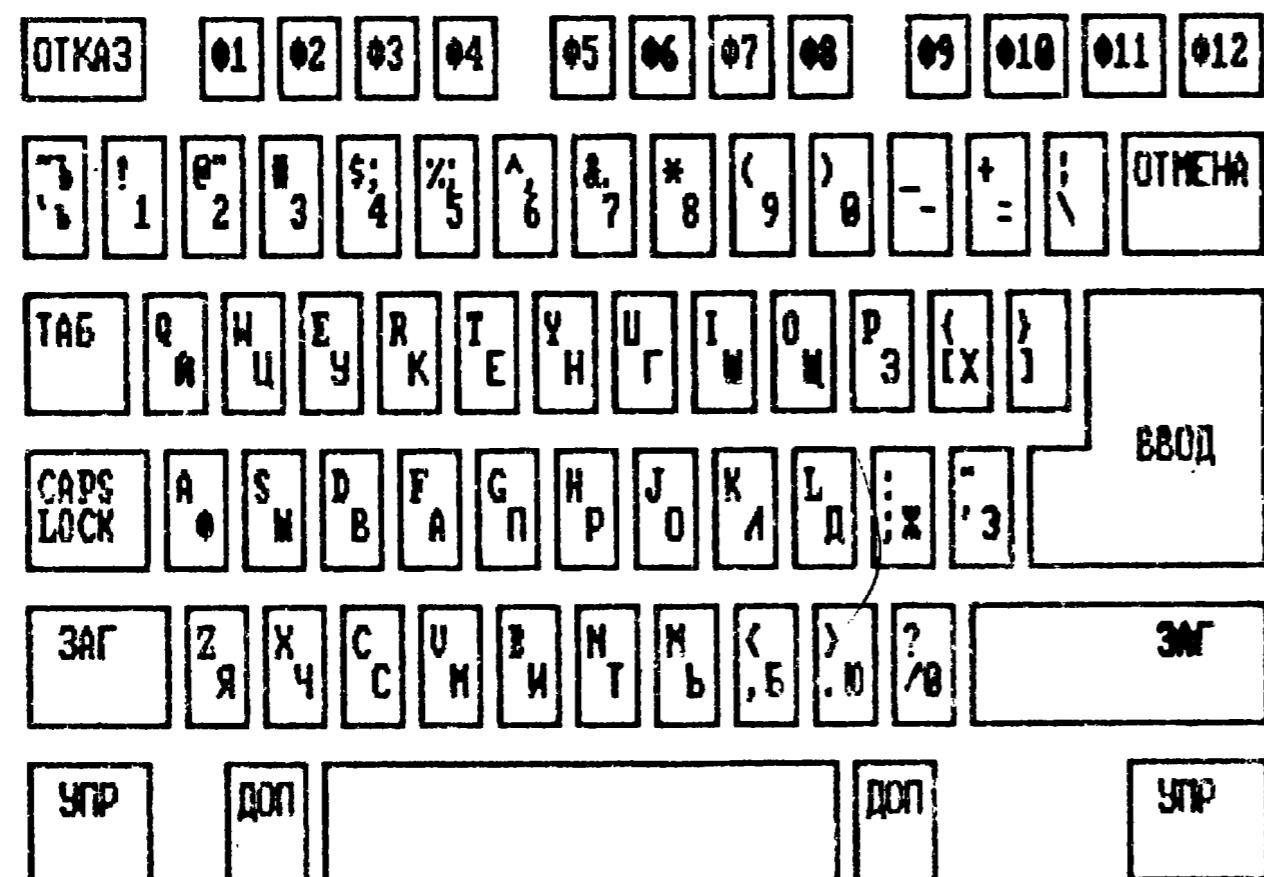


Рис. 2.1. Буквенно-цифровая часть клавиатуры

Особую роль играют три клавиши – {ЗАГ}, {УПР} и {ДОП}, называемые модификаторами. Эти клавиши никогда не употребляются сами по себе, они служат только для одновременного нажатия вместе с другими клавишами. Для обозначения таких одновременных нажатий мы будем использовать составные имена клавиш, например {УПР-Ф10}, {ДОП-Ф9}, {УПР-ВЛЕВО}. Клавиша, нажатая отдельно, и та же клавиша, нажатая с модификатором, производят различные действия в системе МАСТЕР. Таким образом, при наличии на клавиатуре десяти функциональных клавиш число разных клавишных команд оказывается значительно большим.

Обратите внимание на то, что на буквенных клавишиах нанесены сразу два обозначения букв – русское и латинское. Выбор буквы при нажатии клавиши определяется состоянием клавиатуры

русским или латинским. Для переключения этих состояний служат разные клавиши в зависимости от конфигурации компьютера и системы МАСТЕР. Это может быть одна из клавиш {Ф9} или {CAPS-LOCK}, либо клавиши {РУС} и {ЛАТ}.

Описывая те или иные действия в диалоговой среде МАСТЕРа, мы будем записывать последовательности нажатий клавиш, соединяя в цепочку обозначения отдельных нажатий. Например, строка:

{Ф10}РСТчасть 1{ВВОД}

начинает последовательные нажатия: функциональной клавиши {Ф10}, затем буквенных клавиш Р, С, Т, Ч, А, С, Т, Ъ, клавиши пробела, 1 и, наконец, клавиши {ВВОД}.

Начальные действия вне системы МАСТЕР

Для того чтобы вы могли работать с системой МАСТЕР, прежде всего необходимо привести ее в рабочее состояние, т.е. произвести ее установку на вашем компьютере. Эта установка осуществляется с помощью специальной программы, зависящей от комплекта поставки системы и не описываемой в данной книге.

Если установка системы уже произведена, то для ее запуска достаточно ввести команду

MASTER

Через несколько секунд появится заставка системы МАСТЕР, а затем изображение его интерактивной среды, в которой вы будете работать.

2.2. Структура интерактивной среды МАСТЕРа

Работа в интегрированной интерактивной среде МАСТЕРа довольно проста. Она требует предварительного изучения лишь небольшого числа новых понятий, а всему остальному можно научиться непосредственно за клавиатурой ПЭВМ. Прежде чем приступить к содержательному рассмотрению возможностей этой среды, познакомимся с ее формой и наиболее общими правилами поведения в ней.

Объектный характер интерактивной среды

Характерным свойством интегрированной системы МАСТЕР является то, что она представляется пользователю не в виде

программы, с которой нужно общаться командами, вопросами и ответами, выраженными на каком-то формальном языке, а в виде **интерактивной среды**, которая воспринимается пользователем так же предметно, как поверхность рабочего стола с разложенными на ней объектами – бумагами, папками, канцелярскими принадлежностями. На экране вы видите изображение этой среды, а клавиши дают вам операционный доступ к ней. Многие объекты, составляющие эту среду, выглядят примерно так же, как и аналогичные им объекты материального мира: например, тексты и таблицы на экране выглядят именно так, как если бы они были напечатаны на листе бумаги. Другие объекты, не имеющие привычных материальных аналогов, изображаются на экране в виде каких-то условных значков, которые довольно быстро начинают восприниматься, как предметные образы.

Предметность этой среды поддерживается не только в отношении самих обрабатываемых объектов, она обеспечивается еще и тем, что действия над объектами выглядят на экране так же предметно. Например, если объект уничтожается, то он исчезает с экрана; если объект изменяет свою структуру, то меняется его внешний вид; если объект изменяет свое положение среди других объектов, то на экране это выглядит как перемещение изображения объекта.

Для совершения большинства действий в интерактивной среде требуется предварительно указать тот объект, к которому применяется это действие. Эта возможность указания объектов является основой организации диалогового взаимодействия пользователя с интерактивной средой. **Указание** обеспечивается тем, что один из объектов, называемый текущим объектом, выделяется с помощью подсветки, которая называется курсором. Для управления курсором используются клавиши, на которых изображены стрелки. Нажатие клавиши **{ВВЕРХ}** воспринимается системой как команда перемещения курсора на одну позицию вверх, т.е. к ближайшему вышележащему объекту. Нажатия клавиш **{ВНИЗ}**, **{ВЛЕВО}**, **{ВПРАВО}** перемещают курсор на одну позицию в соответствующих направлениях.

После того, как курсор подведен к нужному объекту, можно совершить над этим объектом какое-то действие: уничтожить, передвинуть, скопировать, создать рядом еще один объект и т. п. Для выполнения этих действий служат команды меню и разные функциональные клавиши на клавиатуре, которые уже не столь унифицированы по своему значению, как клавиши-стрелки, и обладают различным смыслом применительно к разным типам объектов и в разных типах интерактивных сред МАСТЕРа. Меню – это как бы расширение клавиатуры путем вынесения ее на экран. Пункты меню – это аналоги клавиш, но находящихся не на клавиатуре, а изображенных на экране в виде мнемонических

слов. Курсор может двигаться по этим пунктам-“клавишам” с помощью стрелок, а при желании “нажать” какую-либо из таких “клавиш” нужно, поставив на нее курсор, нажать клавишу **{ВВОД}**. Выполнение пункта меню часто приводит к появлению нового меню, вспомогательного, уточняющего набор допустимых операций. Для того чтобы перевести курсор с текущего объекта в меню, в МАСТЕРе служит клавиша **{Ф10}**.

Такая организация диалога принципиально отличается от диалога командного типа, действующего на уровне операционной системы. В командном языке, основанном на принципе “подумай и напечатай команду”, каждая команда задается предложением, состоящим из слова – мнемоники команды и из списка аргументов – объектов, подлежащих обработке. Синтаксис этих команд может быть довольно сложным, в особенности если объектная среда имеет сложную структуру, требуя соответствующей сложности при идентификации объектов. Здесь же действует другой принцип – “смотри и показывай”, который предполагает визуальный выбор и указание обрабатываемых объектов, что намного облегчает работу пользователя.

Структура экрана МАСТЕРа

Поскольку наглядность информационной среды играет столь важную роль в организации диалога, то, прежде чем приступить к изучению возможностей среды, необходимо познакомиться с тем, как она выглядит на экране. Структура экрана МАСТЕРа показана на рис. 2.2, где видны основные компоненты экрана – несколько его полей, каждое из которых имеет специальное назначение.

Основную часть экрана занимает **рабочее поле**. В нем будет располагаться вся обрабатываемая информация – тексты, таблицы, рисунки. Три верхние строки экрана заняты служебной информацией. Первая из них называется **полем сообщения** и служит для вывода различных вопросов и пояснений системы. Вторая строка экрана называется **полем значения**. В это поле выводятся значения и формулы, содержащиеся в ячейках электронных таблиц, в нем происходит редактирование этих значений. В этом же поле пользователь дает ответы на запросы, задаваемые МАСТЕРОм.

Третья строка экрана называется **статус-строкой** и содержит четыре индикатора: состояния системы, местоположения, состояния клавиатуры и времени.

Индикатор состояния системы показывает, каким видом деятельности занимается в данный момент система и кому принадлежит активность – пользователю или МАСТЕРу. Когда выполня-

ется некоторая длительная операция, например пересчет электронной таблицы или ввод-вывод файлов, активность принадлежит МАСТЕРу. Это значит, что пользователю не следует в это время ничего вводить с клавиатуры, поскольку МАСТЕР все равно занят другим делом. Во всех таких состояниях индикатор состояния помимо смысловой мнемоники содержит слово ЖДИ, например ИСПОЛНЕНИЕ.ЖДИ или ЧТЕНИЕ.ЖДИ. Во всех остальных состояниях активность принадлежит пользователю, и индикатор состояния не содержит слова ЖДИ, а только мнемоническое обозначение состояния, такое, как РАМКА или ТАБЛИЦА.

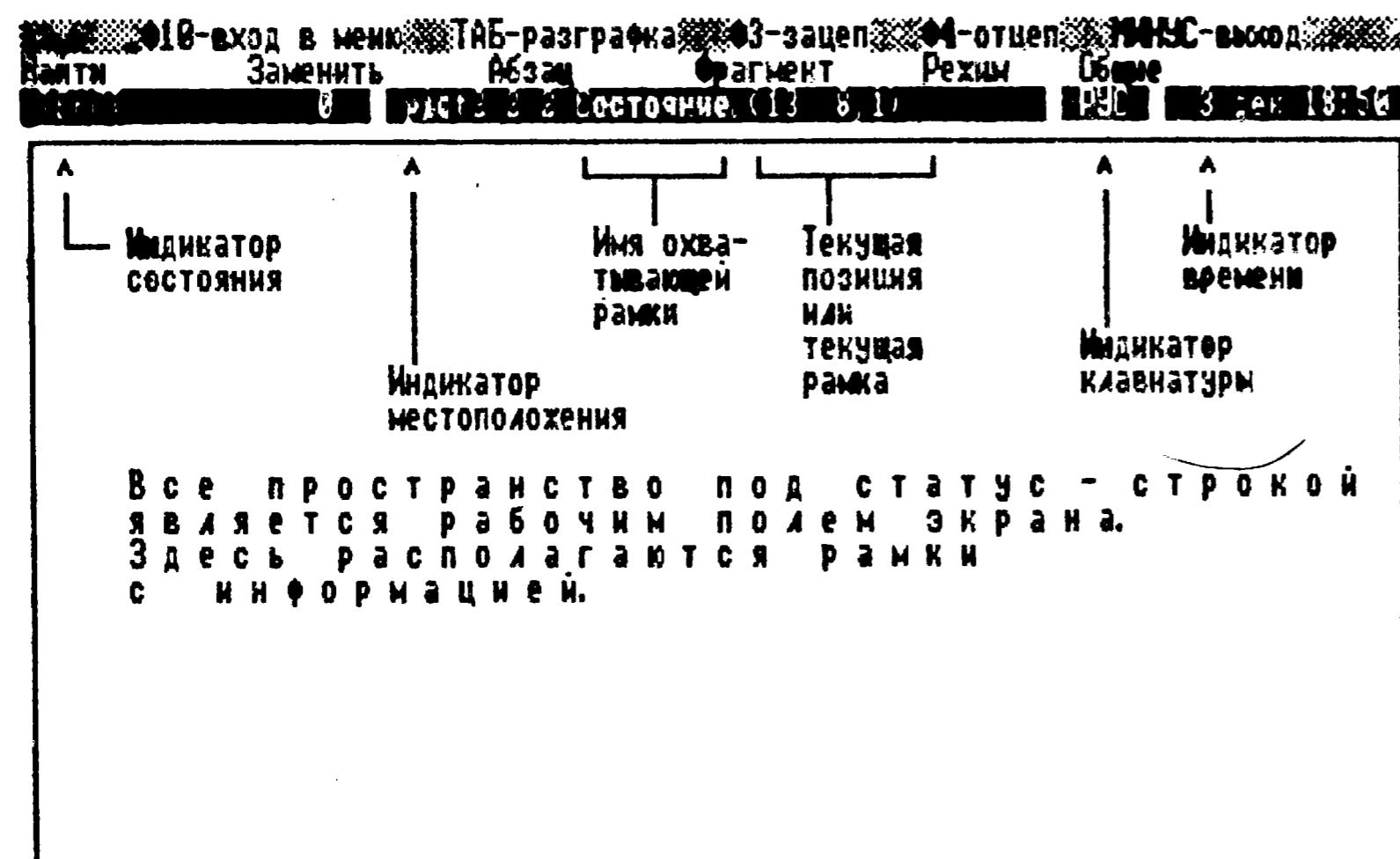


Рис. 2.2. Структура экрана в системе МАСТЕР

Индикатор местоположения показывает ваше местоположение в информационной среде МАСТЕРа. Среда МАСТЕРа является такой же многоуровневой, как дерево каталогов на диске. При работе в такой иерархической структуре желательно иметь перед глазами индикатор своего местоположения в ней. Подобно тому как на командном уровне операционной системы индикатор местоположения высвечивался в приглашении к приему команды, в МАСТЕРЕ этот индикатор высвечивается в статус-строке.

Далее в статус-строке находится **индикатор состояния клавиатуры**. В состоянии, когда клавиатура находится в русском режиме, этот индикатор изображает слово РУС, а в латинском – слово ЛАТ. Последним располагается **индикатор времени**, показывающий текущую дату и время.

Рамки

Прежде чем начать работать с какой-либо информацией, необходимо создать носитель для нее, т.е. ту емкость, в которой информация будет храниться, обрабатываться и изображаться. Такими носителями информации в МАСТЕРЕ являются **рамки**. В рамке может содержаться информация любого из следующих типов: текст, таблица, график или рисунок. В соответствии с типом содержимого рамка будет называться текстовой, табличной или рисунковой. На экране рамка выглядит как прямоугольное окно, через которое частично видно ее содержимое. На рис. 2.3 показан внешний вид рамки на экране и отмечены ее основные элементы.

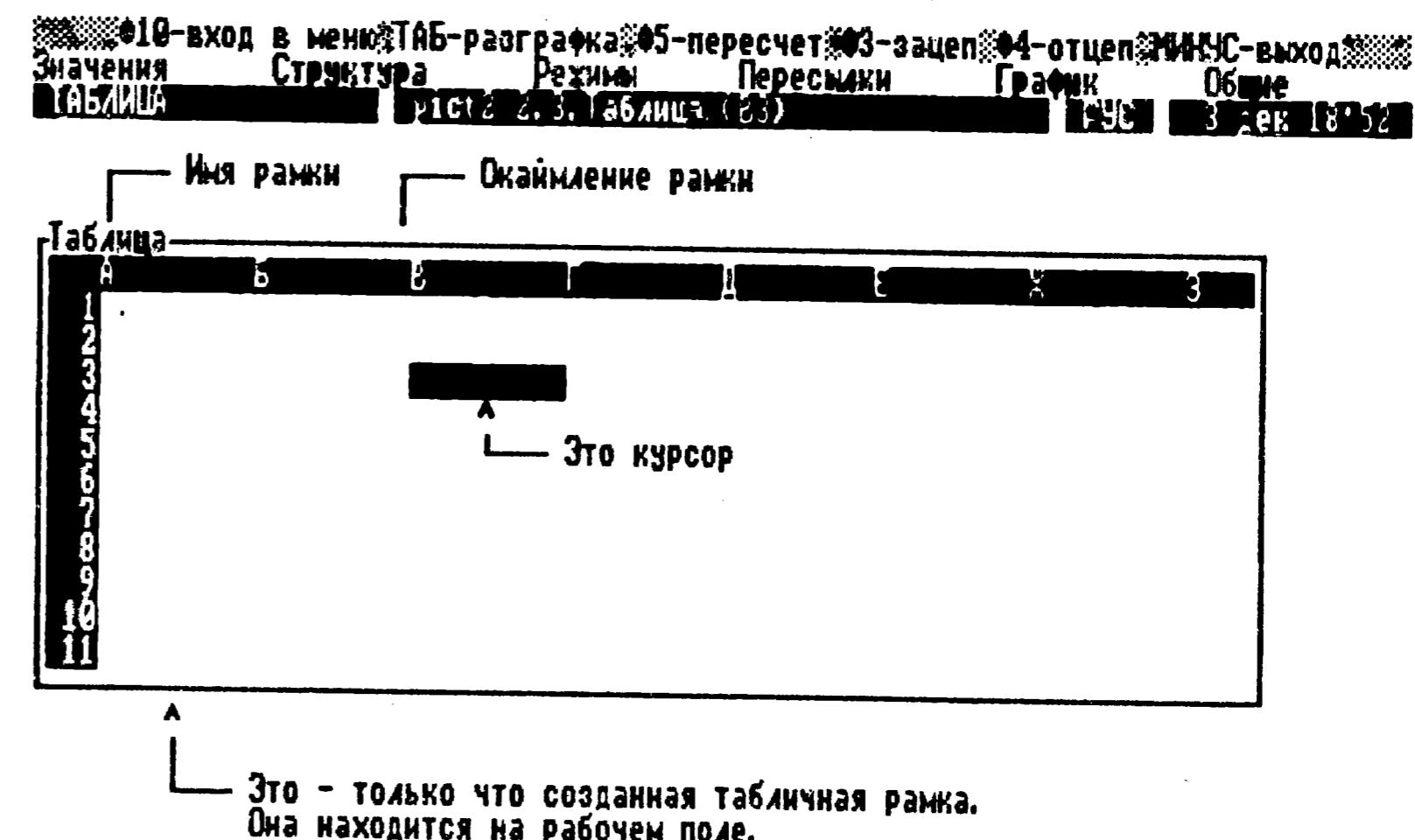


Рис. 2.3. Рамка и ее элементы

Рамка не просто пассивно хранит информацию, она представляет в то же время и интерактивную среду, в которую можно войти, с тем чтобы взаимодействовать с этой информацией. Создав много рамок, вы создадите тем самым сложную интерактивную среду, состоящую из нескольких разных подсред, каждая из которых содержит информацию своего типа и предоставляет интерактивные возможности, соответствующие этому типу информации. В рамку можно войти, и тогда станет доступна для редактирования содержащаяся в ней информация.

Содержимым текстовой рамки служит как бы длинный лист бумаги с написанными на нем символами, а сама рамка открывает доступ к некоторой части этого листа: лист как бы располагается под рамкой. При необходимости этот лист "продерги-

вается" под рамкой, открывая то одну, то другую части текста. Войдя в текстовую рамку, вы окажетесь в интерактивной среде экранного текстового процессора, который будет обслуживать вас, позволяя редактировать и форматировать текст.

Табличная рамка, подобно текстовой, содержит как бы лист бумаги, но, в отличие от текстового листа, он разграфлен вертикальными и горизонтальными графами, в пересечении которых образуются ячейки. В ячейках могут располагаться различные значения (числовые, строковые и др.), а также формулы для автоматического вычисления этих значений. Таким образом, в табличной рамке возникает активная вычислительная среда, называемая электронной таблицей, которая может применяться для перерасчета финансовой ведомости или для аналитического исследования инженерной модели.

Рамки третьего вида – **рисунковые рамки** – по существу тоже аналогичны листу бумаги, но на этом листе могут располагаться не тексты и числовые таблицы, а графические изображения: схемы, рисунки, графики.

Рамки трех перечисленных типов составляют основу для представления обрабатываемой информации – это те листы, на которых записывается сама информация. Будем называть их **информационными рамками**. На рабочем поле может быть создано несколько информационных рамок каждого типа. В каждой из них можно расположить отдельную часть обрабатываемой вами информации. Информацию можно переносить из рамки в рамку, даже если они имеют различные типы. Так, например, часть таблицы можно перенести в текстовую рамку; в этом случае, правда, перенесенная часть потеряет свою внутреннюю структуру (в частности, в ней исчезнут формулы), но внешний вид будет полностью сохранен.

Составные рамки

При создании большого комплексного документа может потребоваться довольно много таких информационных рамок. Для того чтобы объединить все их в единый документ, можно воспользоваться рамкой еще одного типа – **составной**. Содержимым **составной рамки** служат какие-то другие рамки – информационные или в свою очередь составные. Зрительным образом для содержимого составной рамки тоже является некоторая плоская поверхность, на которой в отличие от текстового листа находятся не символы, а расположены другие рамки. Через составную рамку видна часть этой поверхности с лежащими на ней рамками. При необходимости эта поверхность "продергивается" под составной рамкой точно так же, как "продергивается" тексто-

вый или табличный лист под текстовой или табличной рамкой при работе в них. В результате этого "продергивания" вы можете видеть те или иные рамки, расположенные на этой поверхности. На рис. 2.4 показан пример составной рамки. Она представляет собой документ, состоящий из текста, таблицы и графика.

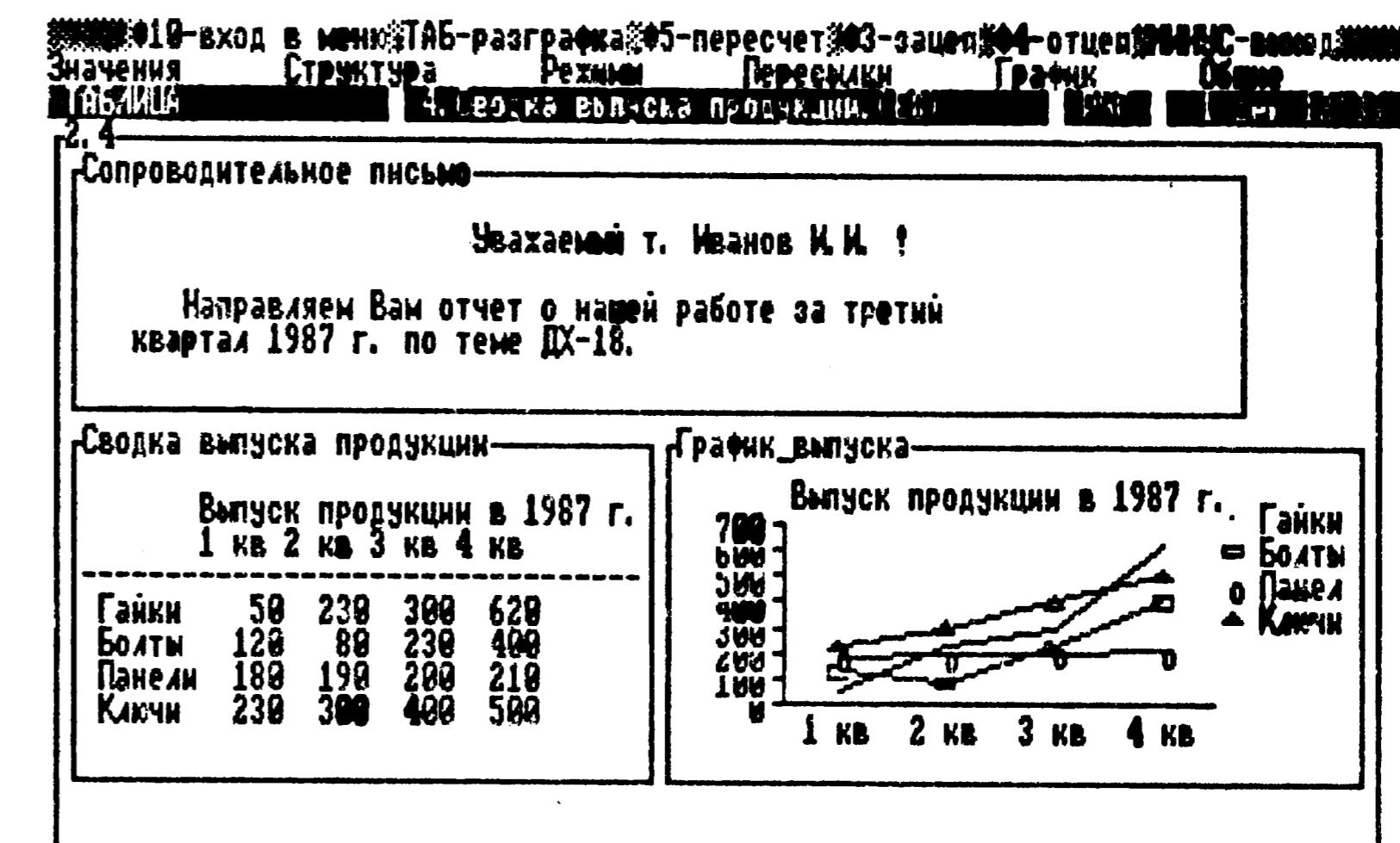


Рис. 2.4. Пример составной рамки

Составные рамки аналогичны каталогам файлов на дисках. Точно так же, как каталог позволяет объединить несколько разных файлов и других каталогов в единую группу, так и составная рамка позволяет объединить в одну логическую группу несколько информационных и составных рамок. Такое группирование рамок можно делать на нескольких уровнях, отражая многоуровневую иерархическую структуру документа или информационной модели.

Движение по рамковой структуре

Работая в информационной среде МАСТЕРа, вы входите то в одну, то в другую рамку и соответственно этому находитесь на одной из рабочих поверхностей: либо на поверхности текстового листа, либо на поверхности электронной таблицы, либо на рамковой поверхности составной рамки. На каждой поверхности в зависимости от ее типа допустимы соответствующие операции.

Так, находясь на текстовой поверхности, вы можете редактировать текст, на графической поверхности можно рисовать изображения, на рамковой поверхности можно совершать различные операции над рамками.

Иными словами, обработке командами меню и функциональными клавишами подвергаются те объекты, которые располагаются на той рабочей поверхности, где вы находитесь: на текстовой поверхности этими объектами являются символы, строки и абзацы, на табличной – ячейки и группы ячеек, на рамковой – рамки.

Прежде чем выполнять операцию, нужно переместиться к тому объекту, над которым она должна совершаться. Перемещения по информационной среде отмечаются с помощью подсветки, называемой курсором, отмечающим тот объект, который является в данный момент текущим. В текстовой рамке (т.е. на текстовой поверхности) курсор имеет вид небольшого прямоугольника, отмечающего текущий символ. В табличной рамке курсор подсвечивает текущую ячейку таблицы. В графической рамке курсор выглядит в виде скрещивающихся линий. На рамковой поверхности курсор подсвечивает одну из рамок, обрамляя ее по периметру.

Движение курсора по информационной среде возможно в двух измерениях: в плоскости одной рабочей поверхности и от одной поверхности к другой. Для движения в пределах одной плоскости действуют клавиши-стрелки {ВЛЕВО}, {ВПРАВО}, {ВВЕРХ}, {ВНИЗ} и их модификации {ЗАГ-ВЛЕВО}, {ЗАГ-ВПРАВО}, {ЗАГ-ВВЕРХ}, {ЗАГ-ВНИЗ}. Они перемещают курсор в соответствующих четырех направлениях, причем первые четыре клавиши перемещают его на одну позицию, а последние четыре перемещают его на максимально удаленную позицию.

Движение от плоскости к плоскости осуществляется с помощью двух функциональных клавиш – {ПЛЮС} и {МИНУС}. На клавиатуре они находятся одна над другой, и верхняя из них – {МИНУС} – выполняет подъем вверх по иерархии рабочих плоскостей, а нижняя – {ПЛЮС} – спуск вниз по этой иерархии.

Движение по иерархии рамок основывается на двух понятиях: охватывающая рамка и текущая рамка. Охватывающей рамкой называется та, на рабочей поверхности которой (т.е. внутри которой) находится курсор. В каком бы месте информационной среды МАСТЕРа вы ни находились, вы всегда находитесь внутри какой-то охватывающей рамки. Даже в самом начальном состоянии, когда вами не создано еще ни одной рамки, охватывающая рамка существует: этой рамкой является сама система МАСТЕР, а ее рабочей поверхностью – экран компьютера (по своему типу эта исходная рамка относится к составным: ее рабочая поверхность предназначена для размещения каких-то других рамок).

В отличие от охватывающей рамки, которая определена всегда, текущая рамка определена только тогда, когда вы находитесь на рамковой поверхности, т.е. внутри составной (а не текстовой, табличной или рисунковой) рамки. Текущая рамка – это та рамка, на которой в данный момент стоит курсор.

Если вы хотите войти в какую-то рамку, то следует, двигаясь с помощью клавиш-стрелок по рамковой поверхности, добираться до нужной рамки и нажать клавишу {ПЛЮС}. Эта клавиша выполнит вход в рамку, т.е. спуск по иерархии рамок с рамковой поверхности на ту рабочую поверхность, которая содержится в текущей рамке. Клавиша {МИНУС} выполняет движение в обратном направлении – выход из рамки, т.е. подъем с рабочей поверхности на ту рамковую поверхность, где расположена рамка, бывшая охватывающей.

ВНИМАНИЕ! Понятия текущей и охватывающей рамок являются центральными в системе МАСТЕР. Не следует смешивать эти понятия друг с другом. Охватывающей рамкой является та, внутри которой находится курсор, а текущей рамкой – та, на которой стоит курсор.

Открытие, закрытие и распахивание рамок

Сразу после создания рамка располагается в рабочем поле в открытом состоянии. Это значит, что она выглядит в виде прямоугольного окна, через которое видно содержимое рамки. Рамку можно закрыть, тогда прямоугольник окна уменьшится до некоторого минимального размера, при котором содержимое рамки уже не видно.

Открытием и закрытием рамок удобно пользоваться для упорядочения своего рабочего места: вы открываете те рамки, информация которых нужна для работы, и закрываете те, необходимости в которых в данный момент нет.

Для того чтобы открыть или закрыть рамку, нужно, чтобы курсор находился на ней, т.е. чтобы эта рамка была текущей. Для открытия и закрытия текущей рамки служит клавиша {ВВОД}: ее последовательные нажатия то открывают, то закрывают рамку.

Если вы используете в документе одновременно много рамок, объединенных в сложную структуру, то они, перекрывая друг друга, могут затруднить видимость той рабочей плоскости, в которой вы работаете. Тогда, чтобы убрать из поля зрения рамки, с которыми вы в данный момент не работаете, можно распахнуть рамку во весь экран. Такое "распахивание" выполняется нажатием клавиши {ДОП-Ф9}. Ее повторные нажатия то распахивают, то нормализуют вид рамки, внутри которой вы на-

ходится. При выходе из распахнутой рамки она автоматически примет свои нормальные размеры.

Имена рамок

Каждая рамка может быть снабжена именем. Если для файлов и каталогов на диске наличие имен обязательно, поскольку доступ к ним из командного языка операционной системы возможен только по именам, то для рамок имена не являются обязательными. Вы можете или не давать рамкам никаких имен, или давать одинаковые имена нескольким разным рамкам – это не повлияет существенно на вашу работу, поскольку доступ к рамкам все равно происходит путем их указания курсором. Тем не менее желательно давать рамкам какие-то имена. Во-первых, в закрытом состоянии от всего изображения рамки остается фактически только одно имя, и вам будет удобнее находить глазами нужные рамки, если они будут поименованы. Во-вторых, имена рамок используются при записи их на диск – тогда эти имена превращаются в имена файлов. В-третьих, при желании записать формулу, связывающую разные рамки друг с другом какой-то вычислительной зависимостью, имя будет необходимо потому, что в языке формул рамки лучше всего идентифицируются по своим именам.

Именем рамки может быть совершенно произвольная цепочка символов, но удобнее всего использовать однословные не слишком длинные имена.

Выполнение обрабатывающих операций

Подведя курсор к нужному объекту информационной среды, вы можете выполнить над этим объектом ту или иную операцию. Операции совершаются в МАСТЕРе двумя способами: с помощью функциональных клавиш и через меню.

С функциональными клавишами связаны операции, требующие наибольшей оперативности и встречающиеся наиболее часто. Такие операции имеют то преимущество, что выполняются одним нажатием клавиши. Их недостатком, однако, является то, что от пользователя требуется предварительное запоминание значений функциональных клавиш. Некоторые из функциональных клавиш уже были рассмотрены. Это клавиши общего назначения, связанные с перемещением курсора и распахиванием рамок: клавиши-стрелки, {ПЛЮС}, {МИНУС}, {ДОП-Ф9}. Эти клавиши имеют универсальное значение в любом месте информационной среды МАСТЕРа. Остальные клавиши связаны со свойствами той

рабочей поверхности, на которой находится курсор. Их описание будет дано в последующих главах книги, посвященных различным типам информационных сред: текстовой, табличной, графической, базе данных.

Другой способ выполнения операций связан с использованием меню. Меню выглядит на экране в виде перечня допустимых действий, где каждое действие представлено одним словом, своей мнемоникой обозначающим смысл действия. Из-за этой мнемоничности пользователю для выполнения операции не нужно ничего запоминать, а достаточно просто указать курсором на то или иное слово в меню.

Меню высвечено во второй строке экрана (в поле значения), причем его вид автоматически изменяется по мере передвижения по информационной среде: оно содержит всегда перечень тех операций, которые применимы в данном месте среды. Начальное меню, которое вы видите сразу после входа в МАСТЕР, называется **главным меню**.

Для того чтобы указать на какой-то пункт меню, требуется прежде всего переместить курсор из рабочего поля в поле значения, где изображены пункты меню. Для этого служит функциональная клавиша {Ф10}. Курсор при попадании в меню приобретает форму прямоугольника, подсвечивающего один из пунктов меню. Находясь в меню, вы можете перемещать курсор от пункта к пункту с помощью клавиш-стрелок {ВЛЕВО} и {ВПРАВО}. По мере передвижения от пункта к пункту вы можете видеть в верхней строке экрана (в поле сообщения) фразы, служащие объяснениями тех пунктов меню, на которые встает курсор. Подведя курсор к нужному пункту, следует нажать клавишу {ВВОД}. Это нажатие исполнит операцию, представленную выбранным пунктом. Существует более быстрая возможность выбрать пункт меню, не требующая многократных нажатий клавиш-стрелок для подвода курсора к нужному пункту и завершающего нажатия клавиши {ВВОД}. Достаточно просто нажать ту букву, с которой начинается название требуемого пункта.

Если, войдя в меню, вы передумали и захотели вернуться в рабочее поле, не выполняя никакой из предложенных в нем операций, то нужно нажать клавишу {ОТКАЗ} (обычно эта клавиша обозначена символами ESC, на клавиатуре ЕС1841 – КЛЮЧ).

Меню МАСТЕРа имеет иерархическую структуру, поэтому, выбрав какой-то пункт, вы, как правило, попадаете в очередное меню, которое высвечивается на месте первого. Такое углубление по структуре меню позволяет уточнить операцию и задать ее параметры. Так, например, при создании рамки появляющиеся вспомогательные меню позволят выбрать сначала нужное действие (создание), затем тип создаваемой рамки (текстовый, табличный, графический и др.).

2.3. Пример работы над документом

Прежде чем описывать в подробностях возможности каждой информационной среды, рассмотрим пример, в котором показано, как происходит работа с информацией в целом. В качестве примера предположим, что требуется составить аналитическую записку о развитии средств вычислительной техники за рубежом. В этом примере будем использовать тексты и количественные данные из известной книги Г.Р.Громова¹.

Создание текстовой рамки

Начнем работу над документом с создания простого текста. Для этого необходимо прежде всего создать рамку для хранения текста. Это делается через меню, войти в которое можно нажатием клавиши {Ф10}. В меню нужно выбрать пункт Рамка, затем операцию Создать. При этом появится еще одно меню, предлагающее перечень возможных типов рамок, — из него нужно выбрать тип Текст. Наконец, вам будет задан вопрос об имени создаваемой рамки — введем имя, скажем, "текст". После этого вы увидите на экране созданную рамку, а подсветка выделит ее по периметру. Это означает, что курсор стоит на рамке.

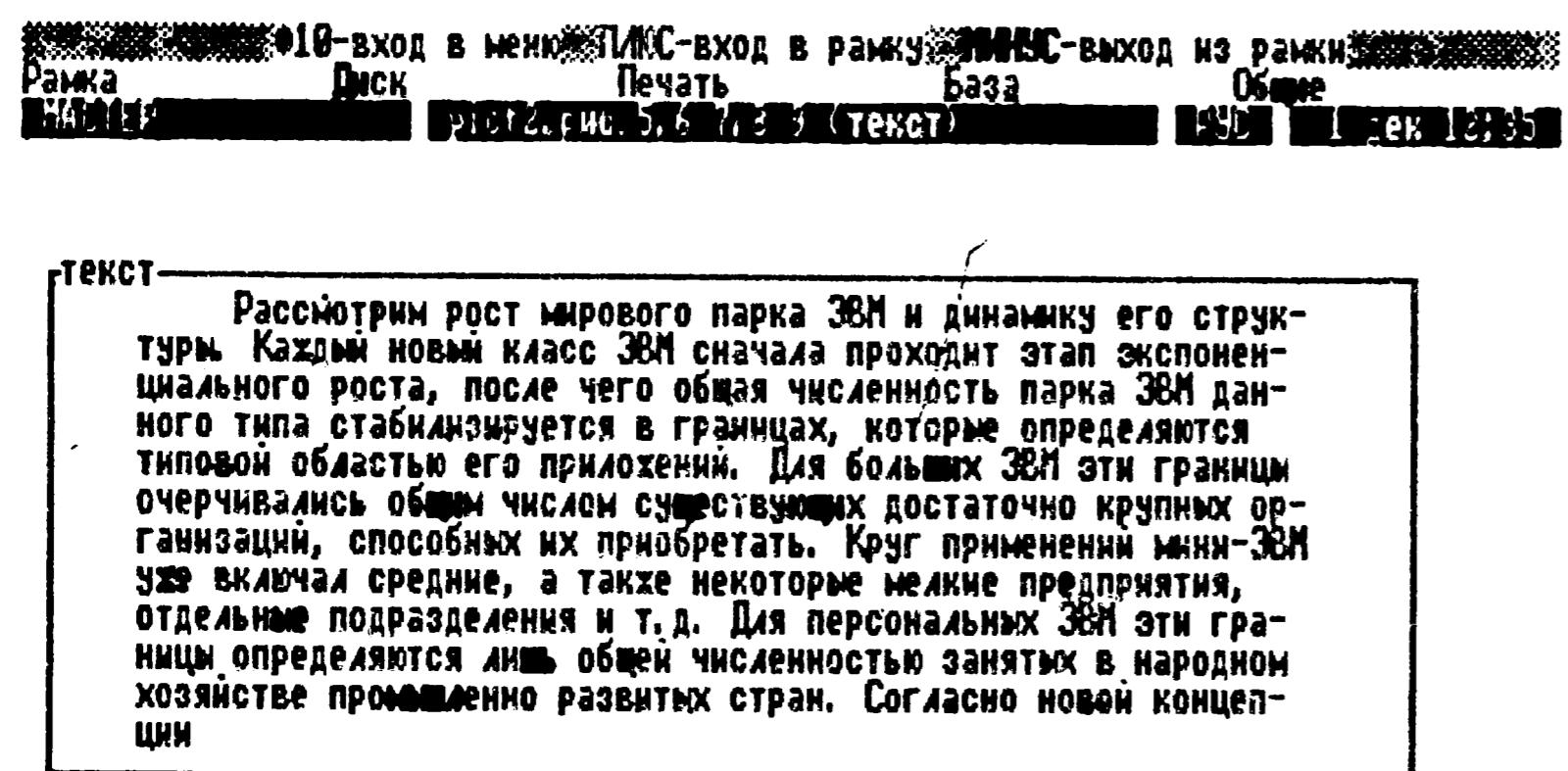


Рис. 2.5. Текстовая рамка с введенным в нее текстом

¹ Громов Г.Р. Национальные информационные ресурсы: проблемы промышленной эксплуатации. — М.: Наука, 1985.

Для того чтобы начать работать с текстом, необходимо войти в эту рамку. Для этого следует нажать клавишу {ПЛЮС} — курсор войдет в рамку и превратится в маленький прямоугольник размером в один символ. Начиная с этого момента, вы можете вводить и редактировать текст внутри рамки.

Ввод текста внутри рамки осуществляется нажатиями соответствующих букв и знаков препинания на клавиатуре. При этом вы можете не заботиться о правой границе абзаца — перенос слов и переход курсора на следующую строку будет сделан автоматически по достижении этой границы. После ввода текста экран будет иметь вид, изображенный на рис. 2.5.

Табличная рамка

Следующим этапом в создании нашего аналитического документа будет построение таблицы, представляющей по годам количество вводимой в эксплуатацию вычислительной техники разных типов — больших, малых и микроЭВМ. С таблицами в МАСТЕРЕ следует работать в рамках табличного типа. Поэтому нужно создать еще одну рамку — для таблицы. Внутри текстовой рамки, где мы сейчас находимся, никаких рамок создавать нельзя и, значит, нужно выйти из текстовой рамки, нажав клавишу {МИНУС}, чтобы оказаться опять на рамковой поверхности.

Создание табличной рамки выполняется точно так же, как и текстовой, с тем лишь отличием, что на вопрос о типе создаваемой рамки следует ответить выбором пункта "Электронная-Таблица". После создания табличной рамки в нее нужно войти, установив на нее курсор и нажав клавишу {ПЛЮС}.

Внутри таблицы курсор движется не по символам, а по ячейкам, и каждое значение располагается в таблице не в произвольном месте, а в той или иной ячейке. Значениями ячеек могут быть строки и числа. Заполним нашу таблицу числами в соответствии с имеющимися исходными числовыми данными. Предварительно заполним шапку и боковик таблицы, введя в них обозначения ее строк и столбцов. Строки в боковике обозначим названиями классов ЭВМ — Большие, Малые, Микро, а столбцы — номерами годов — 50, 55, 60, ..., 90. Поскольку эти обозначения будут использоваться при построении графика для разметки его осей, число в обозначениях годов следует вводить не в числовом, а в строковом виде, для чего перед первой цифрой каждого числа нужно нажимать двойную кавычку ("). Внутри самой таблицы введем числа, представляющие количества ЭВМ соответствующих классов, выпущенных в соответствующем году. Полученная таблица изображена на рис. 2.6.

	58	55	60	65	70	75	80	85	90
Большие	8	0.3	8.7	3.0	10.0	58.0	100.0	110.0	70.0
Малые	0	0	0	0.2	100.0	1000.0	2000.0	1300.0	1000.0
Микро	0	0	0	0	0	1.8	1000.0	3000.0	10000.0

Рис. 2.6. Табличная рамка с числовыми данными

Графическое отображение данных

Для более наглядного представления введенных данных можно изобразить их в графической форме. Операция построения графика выполняется через меню командой:

{Ф10} График

При ее выполнении вам будет задана последовательность вопросов: какие названия отложить по оси X, какие функции отложить по оси Y, какие типы графиков выбрать для каждой из функций, в какой ячейке разместить формулу, выполняющую построение графика и поддержание его в соответствии со значениями чисел на таблице. По полученным от вас ответам МАСТЕР построит формулы, реализующие построение нужных графиков. На рис. 2.7 показан график, построенный на основе данных таблицы.

Аналитическое исследование данных

По рисунку видно, что полученный график является неудачным. Из-за того что масштабы выпуска компьютеров разных типов различаются слишком сильно, мы видим на графике только

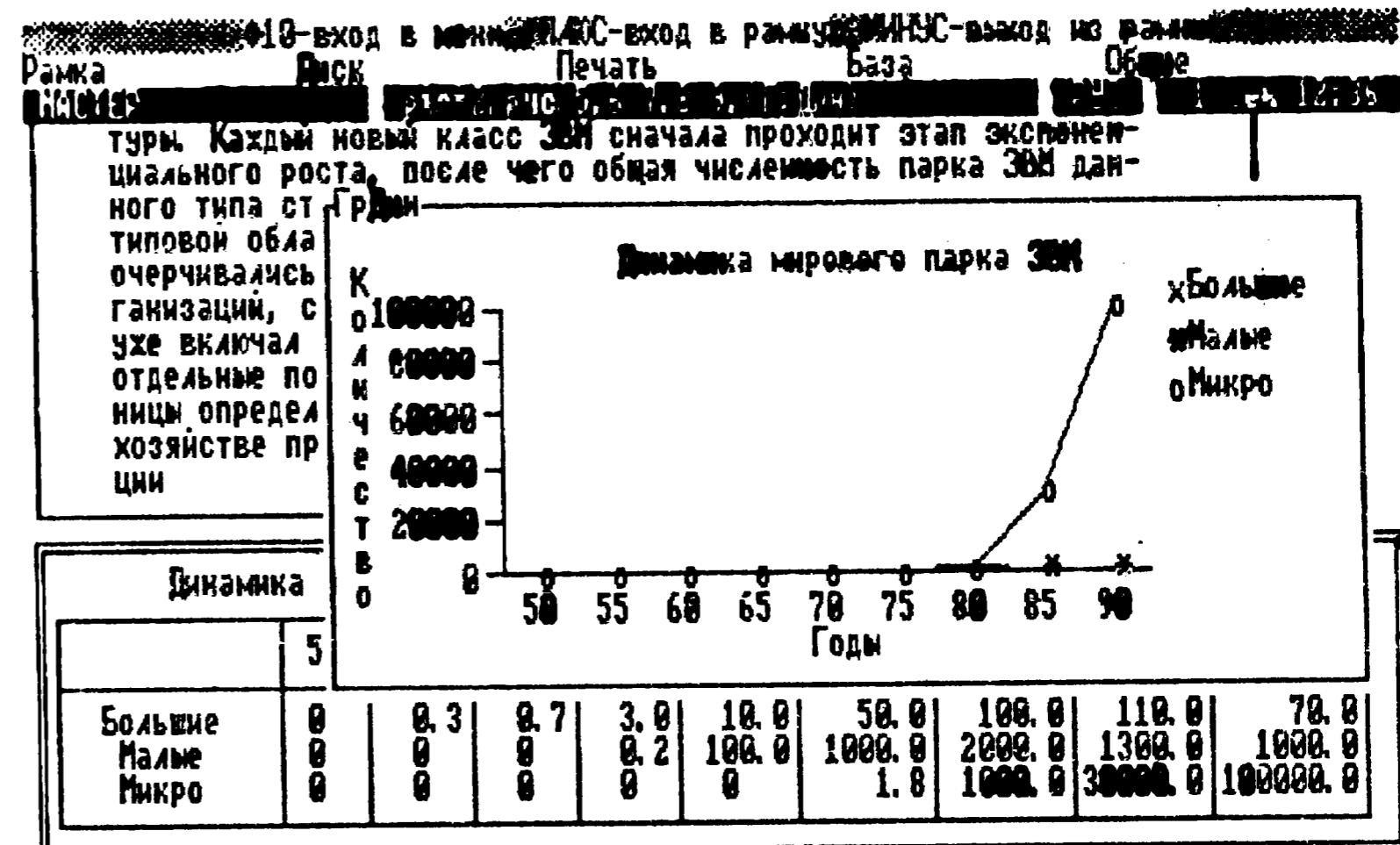


Рис. 2.7. Графическое представление числовых данных

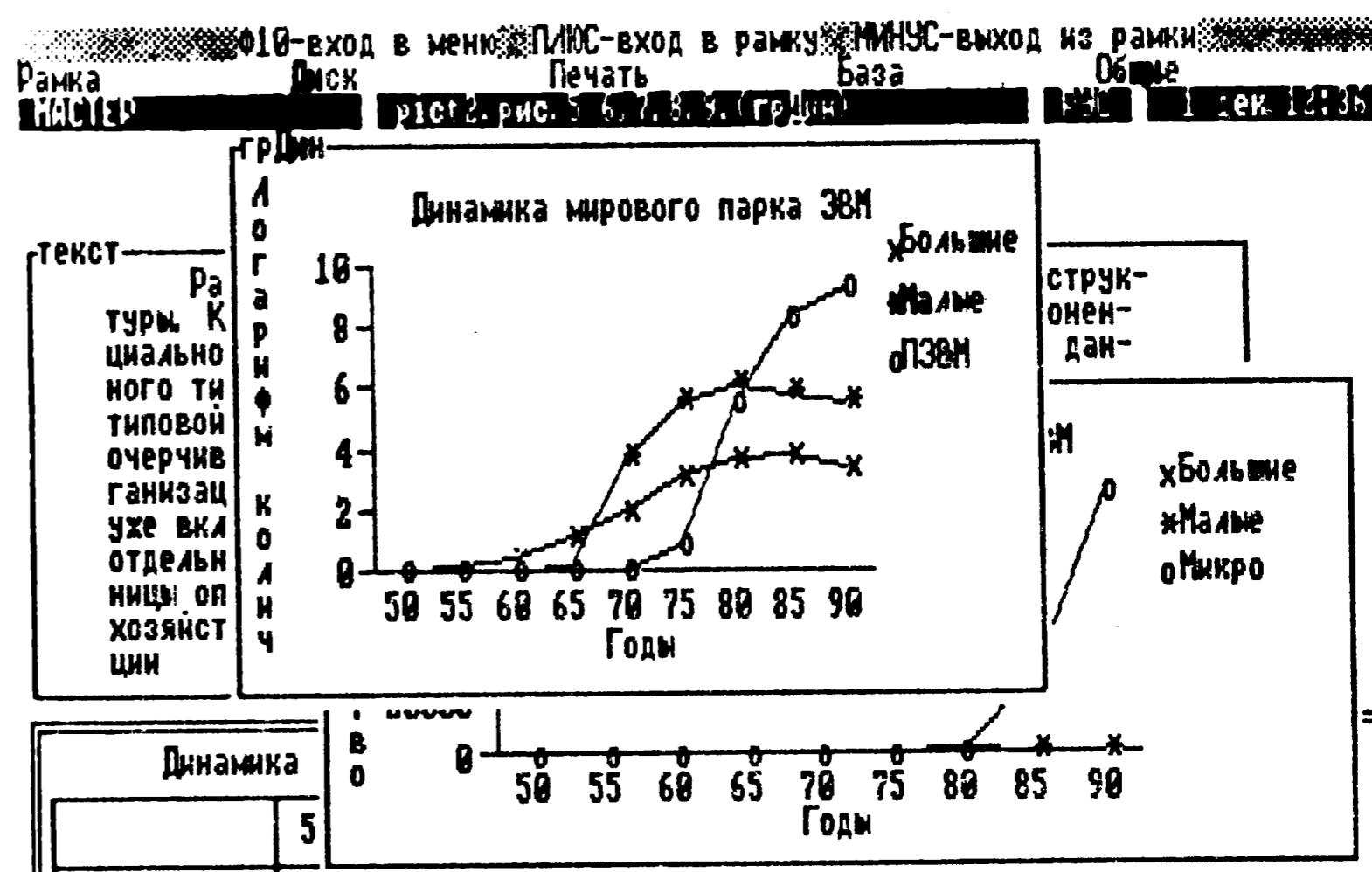


Рис. 2.8. Графики, построенные в логарифмическом масштабе

значения для персональных компьютеров, а кривые для ЭВМ малых и больших типов слились с горизонтальной осью графика. Для того чтобы при значительном различии масштабов все-таки сопоставить характер нескольких кривых, используется лога-

рифмическое масштабирование. Отобразим на графике не количества ЭВМ, а логарифмы этих количеств. Для этого создадим в таблице совокупность формул, порождающих для каждого числа его логарифм (для этого вычисления в МАСТЕРе имеется функция $\text{Ln}(x)$ – натуральный логарифм числа x). После этого построим второй график, отображающий вместо значений их логарифмы (рис. 2.8). По нему уже можно судить о тенденциях в мировом парке ЭВМ.

Аналогичным образом продолжим наше исследование далее. Построим еще одну таблицу и вычислим с помощью соответствующих формул процент ЭВМ разных типов в общем их количестве. Получим числовую таблицу и соответствующий график, изображенные на рис. 2.9.

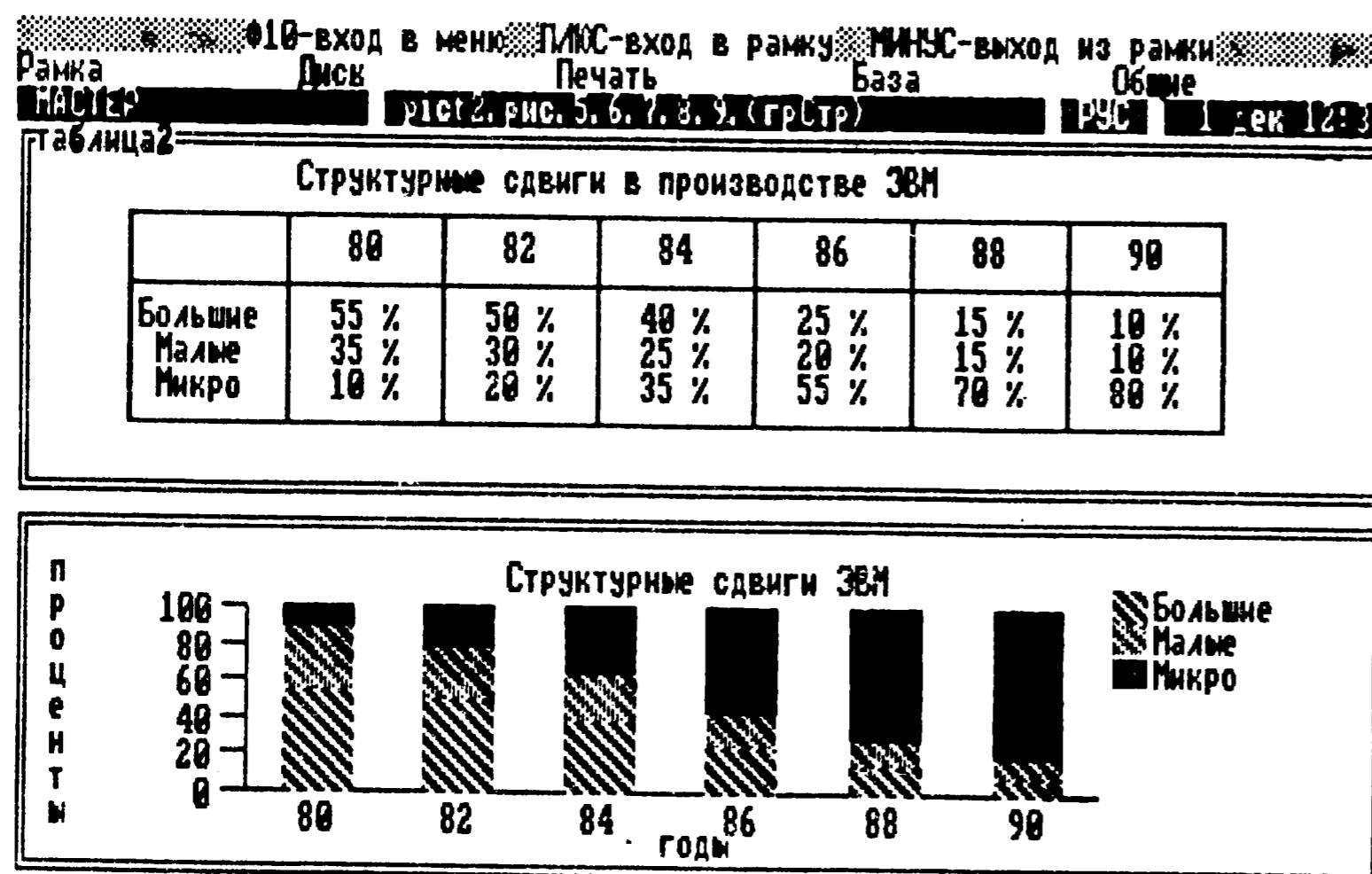


Рис. 2.9. Таблица и график долевого распределения выпуска

Создание составного документа

В результате этой работы мы получили несколько рамок – текстовую, две табличные, три графические. Все они представляют собой разные части одного составного исследования. Поэтому для дальнейшей работы в этом направлении было бы удобно объединить все рамки в один документ. Если бы мы с самого начала предполагали делать составную структуру, то лучше всего было бы вначале создать составную рамку, войти в нее и затем создавать все рассмотренные рамки внутри. Тогда бы мы

имели требуемую целостность с самого начала. Но не поздно объединить рамки и теперь. Для этого в меню имеется операция Сбор, после выполнения которой вокруг всех рамок, лежащих на рабочем поле, создается группирующая их охватывающая рамка. Получившийся составной документ представлен теперь одной рамкой, которую следует сохранить на диске для продолжения работы в следующих сессиях.

2.4. Операции над рамками

Основная работа в МАСТЕРЕ происходит внутри информационных рамок и состоит в редактировании текстов, работе с электронными таблицами, построении графиков, рисовании рисунков, обращении к базе данных. Однако все эти информационные среды располагаются в рамках, которые могут быть еще сгруппированы в составных рамках, в результате чего возникает довольно сложная рамковая структура. Прежде чем говорить об операциях в информационных рамках, необходимо рассмотреть операции над рамками как целыми, поскольку с их помощью вы сможете организовать свою рабочую среду в МАСТЕРЕ.

Операции над рамками доступны тогда, когда рабочей поверхностью является рамковая поверхность и когда курсор стоит на одной из рамок, лежащих на этой рабочей поверхности. Меню в этом состоянии содержит команду Рамка, выполнение которой вызывает вспомогательное меню, которое мы будем называть **рамковым меню**.

Таблица 2.2. Операции рамкового меню

Пункт	Выполняемое действие
Создать	Создание новой рамки
Положение	Задание положения, размеров, цвета рамки
Формат	Установка формата для простой рамки
Вставить	Изменение логической позиции рамки
Уничтожить	Уничтожение текущей рамки
Сбор	Сбор всех рамок в одну составную рамку
Имя	Редактирование имени рамки
Окаймление	Выбор варианта окаймления рамки
Действие	Определение формулы локального действия

Создание новой рамки

Для создания новой рамки служит операция, выполняемая через меню командой

{Ф10} Рамка Создать

Перед созданием рамки МАСТЕР выводит еще одно меню, предлагаю выбрать тип создаваемой рамки. Пункты этого меню – возможные типы рамок – перечислены в табл. 2.3. Последним вопросом перед созданием рамки МАСТЕР запрашивает у пользователя имя для создаваемой рамки.

Таблица 2.3. Меню типов рамок

Пункт	Назначение рамки
Текст	Редактирование текста
Электронная Таблица	Работа с электронной таблицей
Функция	Представление определяемой функции на языке программирования Мастер
Рисунок	Работа с иллюстративной графикой
Простая	Хранение одного числа или строки
Составная	Объединение нескольких рамок

Пространственное и логическое расположение рамок

Рамки, созданные внутри составной рамки, располагаются в определенных пространственных позициях в плоскости рабочей поверхности этой составной рамки. Это пространственное расположение рамок определяет внешний вид рамковой среды на экране. Кроме того, все рамки одного иерархического уровня соединяются невидимой логической цепочкой, определяемой независимо от пространственного расположения. В этом смысле какая-то из рамок является самой первой в логической цепочке, какая-то – следующей, и т.д. до логически последней рамки. Пространственное и логическое расположение рамок никак не связано друг с другом: рамки, имеющие логический порядок, например, А, Б, В, Г, могут пространственно располагаться совершенно произвольно, как это показано на рис. 2.10.

При движении курсора по рамкам можно использовать как их пространственное, так и логическое расположение. Для движения в пространственных направлениях используются клавиши-стрелки {ВПРАВО}, {ВЛЕВО}, {ВВЕРХ}, {ВНИЗ}, а для движения по логической цепочке используются две другие клавиши:

{УПР-ВПРАВО} – переход к логически следующей рамке;
{УПР-ВЛЕВО} – переход к логически предыдущей рамке.

При использовании только пространственных перемещений некоторые рамки могут оказаться недоступными. Если рамка не пересекается по горизонтали и вертикали с текущей рамкой, то курсор не сможет попасть на нее при нажатии клавиш-стрелок.

В этом случае следует использовать клавиши для логического перебора рамок. Это свойство можно использовать и для того, чтобы изолировать друг от друга группы рамок: достаточно расположить их по диагонали друг от друга, чтобы клавиши-стрелки перестали переводить курсор из одной группы рамок в другую.

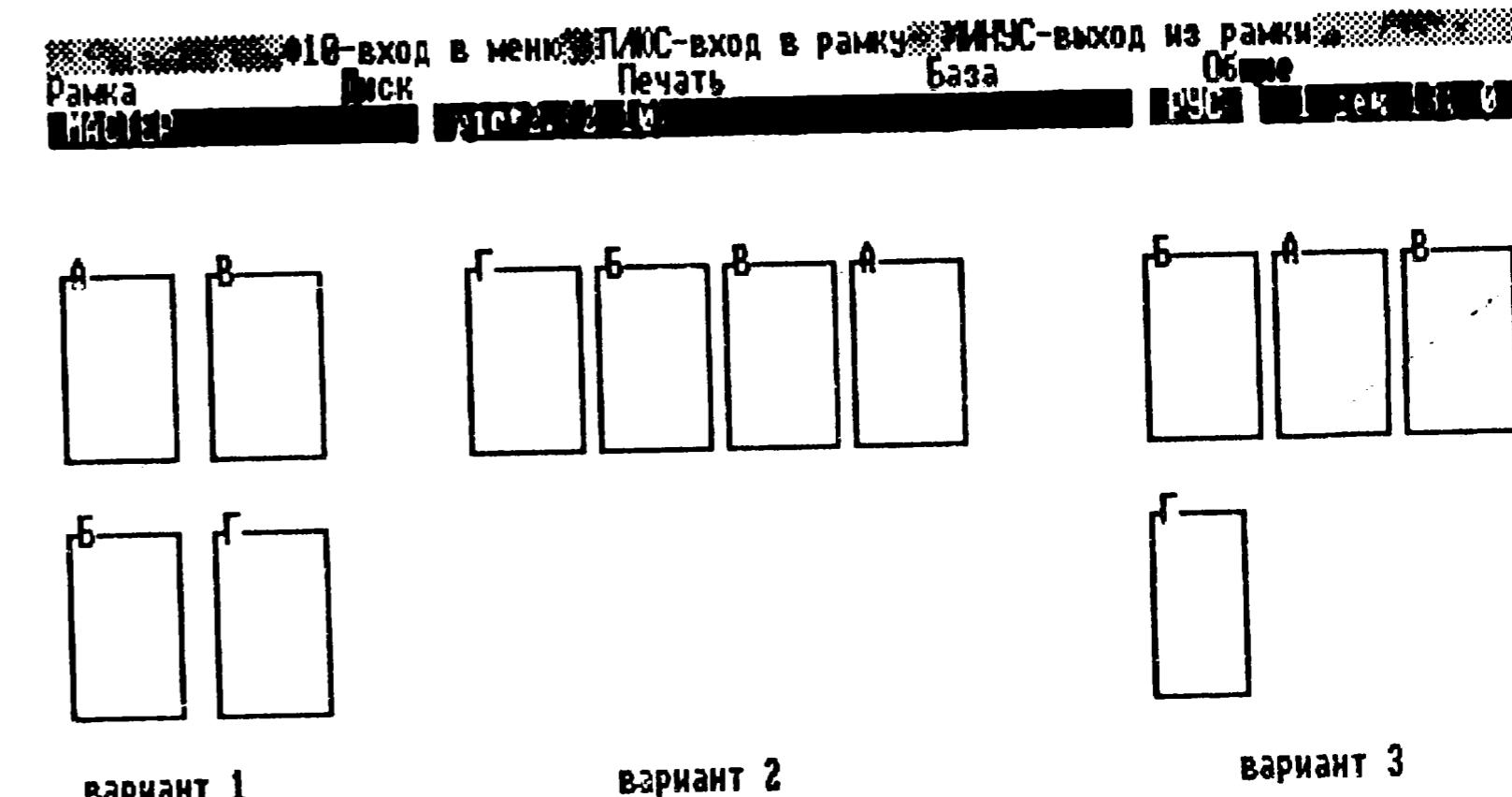


Рис. 2.10. Варианты пространственного расположения рамок

Определение пространственного и логического расположения рамок осуществляется при создании рамок, а также командами {Ф10} Рамка Положение и {Ф10} Рамка Вставить .

Вновь создаваемая рамка размещается на рабочем поле пространственно под текущей рамкой. Размеры и цвет создаваемой рамки устанавливаются автоматически. Все эти атрибуты – размеры, положение и цвет рамки – можно в дальнейшем изменять, для чего существует команда

{Ф10} Рамка Положение

Выполняя эту операцию, вы должны выбрать один из трех режимов: режим установки размеров, режим установки положения или режим установки цвета рамки. Это делается нажатием соответственно клавиш {ВСТ}, {УДЛ}, {ПЛЮС}. Между режимами можно несколько раз переключаться до завершения всей операции, устанавливая поочередно то один, то другой аспект внешнего вида рамки.

Работа в каждом из трех режимов выполняется клавишами-стрелками: они либо изменяют размер рамки, либо передви-

гают ее по рабочему полю, либо перебирают поочередно цвета фона и букв в зависимости от выбранного режима. Завершается операция нажатием клавиши {ВВОД}.

Логическое положение рамки определяется тем, на каком уровне рамковой иерархии (т.е. внутри какой составной рамки) и в какой логической позиции среди рамок этого уровня она находится. Вновь создаваемая рамка вставляется в логическую цепочку после текущей рамки. Иногда требуется изменить логическую позицию рамки, либо переместив ее на данном уровне, либо вставив в другую составную рамку. Для этого служит команда

{Ф10} Рамка Вставить

Операция применяется к текущей рамке. Место, куда эта рамка будет перенесена, определяется ответом на дополнительный запрос:

Укажите место для установки рамки

Задав этот вопрос, МАСТЕР переходит в особое состояние, называемое состоянием **указания**. В поле состояния в это время будет высвечен индикатор УКАЗАНИЕ. В этом состоянии вы можете передвигать рамковый курсор по уровням рамковой иерархии теми же клавишами, которыми он обычно двигается по рамкам. Указание места вставки делается нажатием клавиши {ВВОД}.

ВНИМАНИЕ. Для указания места вставки рамки нужно поставить курсор на некоторую рамку. Поэтому вставка в пустую составную рамку невозможна – требуется предварительно создать в ней какую-нибудь рамку.

2.5. Сохранение информации

Виды файлов в МАСТЕРе

На дисках информация может храниться в различных видах, ориентированных на разные действия над ней. В МАСТЕРЕ применяются три различные файловые структуры для представления данных на дисках: рамковые файлы, текстовые файлы и файлы базы данных.

Рамки, созданные в МАСТЕРЕ, вместе со всей содержащейся в них информацией хранятся в файлах, называемых **рамковыми файлами**. Информация в этих файлах защищена от несанкционированного доступа и не может быть прочитана другими системами (скажем, текстовые редакторы не могут обрабатывать рамковые файлы).

Вместе с тем иногда возникает необходимость информационного обмена именно с другими системами, и тогда необходимы **текстовые файлы**. В них информация хранится в более универсальной форме и доступна для считывания и записи практически любыми информационными системами.

В текстовых и рамковых файлах может храниться информация относительно небольшого объема: файлы и того и другого типов должны иметь возможность целиком разместиться в оперативной памяти, поскольку в МАСТЕРЕ они рассматриваются как содержимое некоторых рамок. Для хранения информации большого объема должна быть применена специальная информационная структура, называемая **базой данных**. Эта структура характеризуется и особыми способами хранения, и особыми операциями доступа. В данной главе, говоря о средствах сохранения информации на дисках, мы совсем не будем касаться вопроса об организации баз данных, которому посвящены отдельные главы книги.

Пользователю необходимо самому заботиться о том, чтобы информация из оперативной памяти (из рамок) своевременно передавалась в долговременную память (в файлы на дисках). Это выполняется операцией записи, имеющейся в главном меню МАСТЕРА. Если вы забудете сохранить какую-то рамку, то перед выходом МАСТЕР напомнит вам об этом. Тем не менее рекомендуется почаще делать промежуточные сохранения рамок. Это предотвратит потерю данных от аварийного выключения электрического питания, от каких-либо сбоев, порожденных ошибками, и т. п. Лучше всего взять за правило выполнять операцию сохранения данных не реже, чем раз в полчаса.

Рамковые файлы МАСТЕРа

Основными операциями для обмена информацией с диском служат операции записи и считывания в рамковом формате. Для этого служат три команды главного меню:

{Ф10} диск Записать

{Ф10} диск Считать

{Ф10} диск Выбор

Командой Записать на диске создается файл, в который записывается информация, содержащаяся в текущей рамке. В этом файле запоминаются не только содержимое, но и все атрибуты рамки: размеры, положение, цвет, вид окаймления, имя рамки, позиция курсора внутри нее. Если записываемая рамка является составной, то все составляющие ее рамки записываются в этот же файл. Если вы захотите записать в отдельный файл не целую

составную рамку, а одну из ее составляющих (какую-то внутреннюю рамку), то следует установить курсор на нее и выполнить операцию записи.

Командой Считать с диска в оперативную память считывается файл и превращается в рамку, которая располагается среди рамок текущего уровня. Считывать рамки можно либо на рабочее поле МАСТЕРа, либо внутрь составной рамки, но не в информационную рамку.

Поскольку файл создан ранее, было бы удобно не вводить его имя, а просто указать курсором, выбрав из числа существующих. Такую возможность предоставляет третья из перечисленных команд – Выбор. При ее выполнении МАСТЕР выводит на экран перечень рамковых файлов, имеющихся на диске. От вас потребуется лишь подвести курсор к нужному имени и нажать {ВВОД} (рис. 2.11).

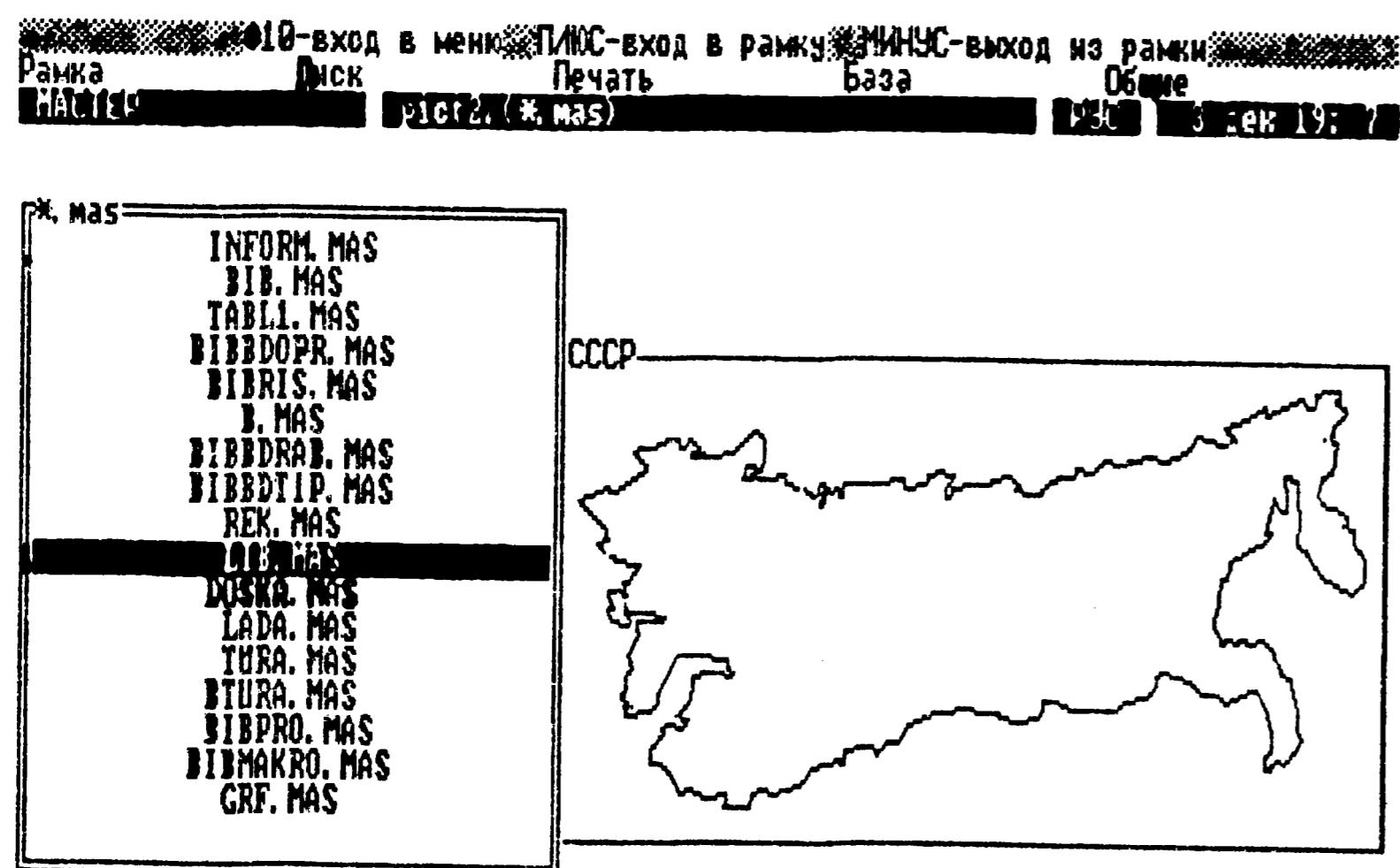


Рис. 2.11. Выбор файла для считывания

Текстовые файлы

В отличие от рамковых файлов, формат которых является закрытым для других программных систем, формат текстовых файлов доступен различным текстовым процессорам, базам данных и другим системам. Текстовый формат непосредственно доступен, в частности, и печатающим устройствам. Поэтому для выдачи информации на печать достаточно преобразовать ее к

виду стандартного текстового файла и передать этот файл на принтер. Операция преобразования рамки в текстовый файл осуществляется командой {Ф10} Печать, имеющей две разновидности: {Ф10} Печать Файл и {Ф10} Печать Принтер.

В первом варианте команды Печать

{Ф10} Печать Файл

информация из текущей рамки записывается в текстовый файл на диске. Этот вариант фактически осуществляет экспорт информации из МАСТЕРа, преобразуя ее из внутренней формы представления в универсальную текстовую форму доступную текстовым процессорам и другим информационным системам. Такие текстовые файлы доступны для считывания и самому МАСТЕРу – к ним применимы обычные операции считывания, выполняемые любой из команд {Ф10} Диск Считать и {Ф10} Диск Выбор.

Вторым вариантом команды Печать является

{Ф10} Печать Принтер

Выполнение данной команды вызывает вспомогательное меню, позволяющее установить некоторые параметры, управляющие распечаткой. Состав и назначение параметров зависят от конкретной конфигурации системы и используемого печатающего устройства. Более подробные сведения об управлении распечаткой приведены в следующей главе, посвященной текстовому процессору МАСТЕРа.

Операция распечатки может быть применена как к текстовым и табличным рамкам, так и к составным. В последнем случае формируется текст, собранный из всех рамок, входящих в эту составную рамку, причем рамки берутся в их логической (а не пространственной) последовательности.

2.6. Прочие общие возможности

Описанные операции над рамками допустимы только в тех случаях, когда курсор находится на рамковой поверхности, т.е. внутри либо составной рамки, либо на верхнем уровне рабочего поля МАСТЕРа. В этих случаях в меню видны операции Рамка, Диск и Печать, представляющие все рассмотренные операции. Во всех других состояниях, т.е. внутри текстовых, табличных, графических рамок и в рамках базы данных, эти операции недоступны.

Однако имеется ряд операций самого общего характера, вообще не связанных с рамками. Эти операции применимы в любом

состоянии и в любом месте информационной среды МАСТЕРа. В любом варианте меню МАСТЕРа они представлены пунктом **Общие**. К числу таких общих операций относятся команды, предназначенные для определения макроклавиш, вычислений формул, управления состоянием экрана, выхода из МАСТЕРа.

Использование макросредств

Макроклавишами называются буквенные клавиши в сочетании с модифицирующей клавишей {ДОП}: {ДОП-А}, {ДОП-Б}, {ДОП-В} и т.д. Вы можете связать с любой из таких клавиц символическую последовательность, представляющую нажатия каких-то других клавиш и называемую **макропоследовательностью**. Например, если вы хотите, работая в текстовом редакторе, многократно вводить какое-то длинное слово или словосочетание, то можно связать с макроклавишей это слово, и тогда для его ввода в текст будет достаточно нажать лишь одну эту макроклавишу.

Помимо буквенно-цифровых символов в макропоследовательность можно включать и функциональные клавиши, что позволит уже не просто вводить сокращения для слов и фраз, а создавать небольшие программы, пользуясь операционными действиями функциональных клавиш и меню МАСТЕРа. Для изображения функциональных клавиш внутри макропоследовательности используются условные обозначения в фигурных скобках. Перечень основных обозначений клавиш и их расположение на клавиатуре показано на рис. 2.1. Помимо этих основных обозначений можно использовать еще обозначения модифицированных нажатий.

Все знаковые клавиши изображаются в макропоследовательностях самими знаками:

А Б В Г ... а б в г ... 1 2 3

Единственное исключение составляет открывающая фигурная скобка, обозначаемая в виде {{}. Клавиши можно обозначать также и их кодами, например

- {128} – код русской буквы А;
- {-1} – код функциональной клавиши {ОТКАЗ}.

Определение макроклавиш выполняется с помощью команды

{Ф10} Общие Макро Определить

при выполнении которой сначала запрашивается, для какой буквы выполняется определение (в ответ нужно нажать буквенную

клавишу, не нажимая модифицирующей клавиши {ДОП}), а после этого запрашивается строка макропоследовательности. При определении сложной макропоследовательности полезно помнить о том, что при вводе ее текста, вы можете воспользоваться карманом, в который (с помощью клавиш {Ф3} и {УПР-Ф3}) можно предварительно забрать текст из текстовой рамки или из другого макроопределения.

Набор всех сделанных в диалоговом сеансе макроопределений можно сохранить в файле для того, чтобы затем использовать в следующих сеансах. Для этого служат команды:

{Ф10} Общие Макро Сохранить
{Ф10} Общие Макро Загрузить

Другой более удобный способ задания макроопределений состоит в использовании **режима обучения**. Этот режим включается нажатием функциональной клавиши {ЗАГ-Ф1}, и в нем МАСТЕР запоминает все те нажатия клавиш, которые вы делаете. При этом нажатие каждой клавиши сопровождается коротким звуковым сигналом, напоминающим о включенности режима обучения. Завершается режим обучения нажатием той же клавиши {ЗАГ-Ф1}. При этом задается вопрос о том, с какой макроклавишей связать запомненную последовательность действий, в ответ на который также нужно нажать какую-то буквенную клавишу без модификатора {ДОП}.

Программирование функциональных клавиш в рамках

Использование макроклавиш – это первый шаг к программированию. Макропоследовательности представляют собой простейшие программы, задающие названиями функциональных клавиш последовательности действий. Этот способ программирования, однако, сильно ограничен, поскольку сводится только к тем действиям, которые связаны с функциональными клавишами. К тому же язык макропоследовательностей слишком слаб с алгоритмической точки зрения. В МАСТЕРЕ имеется возможность не только использовать действия функциональных клавиш, но и доопределять их. Для этого существует встроенный инструментальный язык программирования Мастер.

Полное изучение языка требует некоторого времени и необходимо только тем пользователям, которые собираются использовать МАСТЕР в качестве инструмента для разработки прикладных систем. Однако и обычным пользователям полезно знать о наличии этого языка и владеть некоторыми простейшими его возможностями.

Одна из главных возможностей состоит в том, что с каждой функциональной клавишей внутри каждой рамки можно связать некоторую формулу, написанную на языке Мастер. Тогда при нажатии этой клавиши внутри данной рамки будет срабатывать связанная с клавишей формула. Тем самым функциональные возможности системы МАСТЕР могут быть произвольным образом расширены. Средства, имеющиеся в языке Мастер, достаточно полны и гибки, чтобы выразить с их помощью и простейшие арифметические вычисления, и структурные преобразования в рамковой среде, и сложные запросы к базе данных, и произвольные диалоговые эффекты.

Подробному описанию языка Мастер посвящена вся вторая часть книги. Здесь рассмотрим только то, каким образом можно связывать формулы с функциональными клавишами в рамках. Для этого существует несколько команд в рамковом меню, сгруппированных в пункте

{Ф10} Рамка Действие

Одна из этих команд

{Ф10} Рамка Действие Клавиша

позволяет связать формулу с какой-либо функциональной клавишей или посмотреть, какая формула уже связана с той или иной клавишей. Выполнение операции вызывает два запроса. Первый из них – "нажмите функциональную клавишу" просит нажать ту клавишу, которую вы собираетесь определить. Второй – "введите формулу" – просит ввести формулу, определяющую действие этой клавиши.

Связывание формулы с клавишей делается для той рамки, которая является текущей при выполнении данной операции. Действие этой функциональной клавиши ограничивается пределами одной только указанной рамки, поэтому оно называется **локальным действием клавиши**. Это действие доступно только внутри этой рамки. Поэтому если сразу после определения действия клавиши вы нажмете ее, то она не сработает запрограммированным вами образом. Требуется прежде войти в ту рамку, для которой было сделано это определение. В разных рамках с одной и той же клавишей можно связывать различные формулы.

Иногда удобнее задавать определение локального действия функциональной клавиши не снаружи, как это было описано, а изнутри той рамки, для которой делается определение. Это возможно с помощью специальной функциональной клавиши {ДОП-Ф10}. Ее нажатие вызывает те же два запроса, что и в команде {Ф10} Рамка Действие Клавиша, но определяемое действие относится уже не к текущей, а к охватывающей рамке.

Простейшим примером такого определения клавиши может быть следующий. Связем в некоторой рамке с клавишей {Ф2} формулу

Сохранить(Имя(ОхвРамка()), ОхвРамка())

Эта формула задает действие, состоящее в записи охватывающей рамки на диск. После введения этого определения вы сможете, работая внутри данной рамки, одним нажатием клавиши {Ф2} сохранять рамку в файле на диске. Для сравнения с языком макропоследовательностей можно заметить, что эта формула эквивалентна следующей макропоследовательности:

{МИНУС}{Ф10}ДЗ{ВВОД}{ПЛЮС}

Калькулятор

К числу вспомогательных средств относится еще одна операция, позволяющая делать вычисления. Эту операцию можно использовать в качестве арифметического калькулятора. Выполняется она через меню следующим образом:

{Ф10} Общие Формула

В начале операции МАСТЕР выдает запрос "введите формулу"; в ответ на него нужно ввести арифметическую формулу, которую требуется вычислить. Примерами формул могут служить следующие:

1985 – 1917
Sin (3.14)
Sqrt (3 ^ 2 + 5 ^ 2)

После ввода такой формулы МАСТЕР вычисляет ее и выдает в поле сообщения результат этого вычисления. Языком для написания формул служит инструментальный язык системы МАСТЕР, в котором имеется множество самых разнообразных возможностей помимо арифметических. Полному изложению этого языка посвящена вторая часть книги, а здесь приведем лишь простейшие арифметические возможности.

В языке Мастер имеются арифметические операции: сложение (+), вычитание (-), умножение (*), деление (/), возведение в степень (^), а также математические функции Sin(x), Cos(x), Tg(x), Ctg(x), Arcsin(x), Arccos(x), Exp(x), Ln(x), Sqr(x). При использовании этих функций следует иметь в виду, что в языке Мастер строчные буквы считаются отличными от прописных, поэтому названия функций следует записывать так, как

Что сделано в приведенном списке: первая буква каждого имени является прописной, а остальные -- строчными.

Глава 3

Переключение режимов экрана

Еще одна вспомогательная возможность сервисного характера существует в том, чтобы переключать экран в разные режимы. Здесь имеются два выбора. Первая альтернатива: экран может находиться либо в **графическом**, либо в **символьном режиме**. Вторая альтернатива: строка может быть видна или не видна на экране. Все эти альтернативные режимы экрана представлены во вспомогательном меню, вызываемом из общего меню командой Экран.

Символьный режим отличается от графического тем, что в нем допустимо использование одновременно 16 цветов, но недопустимо использование графики (графические изображения не появляются на экране). В графическом режиме, наоборот, доступны графические изображения, но зато все цвета, кроме черного, сводятся в один, и изображение получается двухцветным.

Выход из МАСТЕРа

Наконец, последняя возможность общего характера – это **выход** из системы МАСТЕР. Эта возможность представлена командой

{Ф10} Общие Выход

Выйти из МАСТЕРа на уровень операционной системы MS DOS можно двумя способами. Первый способ выхода – временный:

{Ф10} Общие Выход Временно

При этом как сама система МАСТЕР, так и все рамки остаются в оперативной памяти, и в свободную часть оперативной памяти вызывается командный интерпретатор MS DOS, который позволит вам выполнить произвольные операции уровня операционной системы. После выполнения этих команд можно вернуться в МАСТЕР с помощью команды

EXIT

Второй способ выхода из МАСТЕРа – окончательный, завершающий сеанс работы:

{Ф10} Общие Выход Конец

ДИАЛОГОВОЕ РЕДАКТИРОВАНИЕ ТЕКСТОВ

3.1. Технология обработки текстов

Текстовые процессоры

Основным видом информации, обрабатываемой в учрежденческой деятельности, является текстовая информация. Почти половина продаваемых персональных компьютеров предназначается преимущественно для обработки текстов. Возник целый класс систем программного обеспечения, называемых **текстовыми процессорами**, которые предназначены для выполнения этой обработки. Текстовые процессоры по своему количеству, возможностям, развитости технологии их производства и применения превосходят все остальные виды программного обеспечения – операционные системы, компиляторы, базы данных, вычислительные программы.

Ни одна интегрированная информационная среда не может обойтись без текстового процессора. В том числе и в МАСТЕРе встроенный текстовый процессор играет важнейшую роль в обработке информации. Текстовый процессор МАСТЕРа обслуживает все те участки интегрированной среды, где находится текстовая или строковая информация: работу в текстовых рамках, редактирование содержимого ячеек электронных таблиц, ответы на запросы, требующие строковых или текстовых ответов, и т.п.

Важность электронной обработки текстовой информации в учрежденческой деятельности обуславливается несколькими причинами. Во-первых, документы при электронной обработке оказываются избавленными от подчисток, помарок, исправлений.

Вторых, несмотря на различное содержание документов, они часто оказываются совпадающими во многих пунктах, за исключением, подчас, лишь нескольких цифр или формулировок, как, например, варианты договоров, справок и т.п. При этом возможность быстро создать новый документ из существующего старого варианта многократно повышает производительность работы. В-третьих, многие документы, такие, как официальные письма, должны репродуцироваться во многих экземплярах с отличиями лишь в обращениях.

Для входа в меню нажмите F10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перейти Шрифт ДС Выход
МК (24 8,1 0) МА 2 лин 2:48
лишь в обращениях.

Текстовый процессор **весьма** сходен обычной пишущей машинкой: вводимый текст пользователь видит непосредственно перед собой, как если бы он был напечатан на листе бумаги. Однако в отличие от пишущей машинки текстовый процессор обладает многими очень существенными преимуществами.

Для входа в меню нажмите F10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перейти Шрифт ДС Выход
МК (24 7,4 0) МА 2 лин 2:48
лишь в обращениях.

Текстовый процессор **весьма** сходен обычной пишущей машинкой: вводимый текст пользователь видит непосредственно перед собой, как если бы он был напечатан на листе бумаги. Однако в отличие от пишущей машинки текстовый процессор обладает многими очень существенными преимуществами.

Для входа в меню нажмите F10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перейти Шрифт ДС Выход
МК (24 7,3 0) МА 2 лин 2:50
лишь в обращениях.

Текстовый процессор **во многом** сходен обычной пишущей машинкой: вводимый текст пользователь видит непосредственно перед собой, как если бы он был напечатан на листе бумаги. Однако в отличие от пишущей машинки текстовый процессор обладает многими очень существенными преимуществами.

Рис. 3.1. Стирание и вставка слов в середину абзаца

Текстовый процессор во многом сходен с обычной пишущей машинкой: вводимый текст пользователь видит непосредственно перед собой, как если бы он был напечатан на листе бумаги. Однако в отличие от пишущей машинки текстовый процессор обладает многими очень существенными преимуществами. Наиболее заметные из этих отличий продемонстрированы на рис. 3.1 и 3.2. На первом рисунке показано, как стирается слово, а затем добавляется несколько новых слов в середину абзаца. На втором – как изменяется форма абзаца (сдвигаются его левая и

правая границы). Причем и в том и в другом случаях совсем не требуется повторный ввод текста этих абзацев, как это необходимо было бы делать при бумажной технологии.

Для входа в меню нажмите F10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перейти Шрифт ДС Выход
МК (24 8,1 0) МА 2 лин 2:48
ты В-третьих, многие документы, такие, как официальные Текстовый процессор во многом сходен обычной пишущей машинкой:
а) вводимый текст пользователь видит непосредственно перед собой, как если бы он был напечатан на листе бумаги. Однако в отличие от пишущей машинки текстовый процессор обладает многими очень существенными преимуществами.

Для входа в меню нажмите F10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перейти Шрифт ДС Выход
МК (24 8,1 0) МА 2 лин 2:48
ты В-третьих, многие документы, такие, как официальные Текстовый процессор во многом сходен обычной пишущей машинкой:
а) вводимый текст пользователь видит непосредственно перед собой, как если бы он был напечатан на листе бумаги. Однако в отличие от пишущей машинки текстовый процессор обладает многими очень существенными преимуществами.

Рис. 3.2. Переформатирование абзаца

В рамках интегрированной среды текстовая информация является лишь одним из видов информации, доступной для обработки. Здесь можно создавать комплексные документы, включающие не только текстовые, но еще и табличные, и графические части. В данной главе мы рассмотрим возможности обработки документов текстового характера.

Стадии обработки текста

Процесс обработки текстов в целом включает в себя несколько этапов:

сбор исходного материала. На этом этапе пользователь вводит новые тексты, подбирает и исправляет подходящие старые тексты, пользуется, быть может, базой данных или даже вычислительной программой для получения содержательных данных;

форматирование текста. Полученной текстовой информации должен быть придан вид, удовлетворяющий какие-то конкретные оформительские требования – правая и левая границы абзацев, число строк на странице, использование шрифтов и т.п. Преоб-

Изование формы текста во всех этих отношениях называется **форматированием**;

пробная распечатка. Такая распечатка делается максимально быстрым способом, не давая высокого качества печати. На этой распечатке тем не менее должны быть видны все принципиальные аспекты текстового документа – расположение на страницах, шрифтовые выделения отдельных слов и фраз. По ней проверяются возможные ошибки как в содержании, так и в оформлении документа. Эта пробная распечатка может быть использована при рассылке текста для предварительного ознакомления;

выдача окончательного текста. После всех проверок и исправлений документ распечатывается максимально качественным (но и наиболее медленным) способом.

Сбор исходного материала

Назначением текстового процессора в интегрированной системе является выполнение перечисленных технологических этапов. Первый из них – сбор материала – осуществляется за счет интегрированных связей текстового процессора с прочими видами информации и, в частности, с базой данных. Основным понятием, обеспечивающим работу на этом этапе, является **текстовая рамка**. Поскольку вообще любая информация в МАСТЕРе хранится в рамках соответствующих типов, то и для текстов имеются рамки текстового типа. Этих рамок можно создать несколько штук, располагая в них разные тексты для одновременного редактирования. Одновременное использование нескольких текстовых рамок очень удобно, поскольку позволяет сравнивать тексты, переносить информацию из одной рамки в другую. Рамки можно произвольным образом расположить на экране, открывать и закрывать, задавать цвета и размеры рамок с помощью обычных операций над рамками. Существующие в системе МАСТЕР средства интеграции позволяют обмениваться информацией между текстовыми рамками, табличными рамками, рамками базы данных.

Понятия и элементы текстовой структуры

На этапе форматирования текста используется ряд понятий, составляющих структуру текста: строки, абзацы, фрагменты, страницы, разделы. Все эти понятия имеют самое обычное содержание, но для определенности поясним каждое из них на примерах. На рис. 3.3 показан вид текстовой рамки, содержащей все перечисленные элементы.

Абзацем называется совокупность строк, начинающихся в одной и той же левой позиции, за исключением первой строки, которая немного сдвинута либо вправо, либо влево от края абзаца, образуя абзацный отступ или выступ.

Во время редактирования часто требуется выделять какие-то части текста для того, чтобы копировать, уничтожать, переформатировать их. Выделение выглядит на экране в виде подсветки, а выделенная часть текста называется **фрагментом**. Фрагменты в текстовом процессоре МАСТЕРа могут быть двух типов: строковые и прямоугольные. Строковые фрагменты выделяют целиком несколько строк, например какой-нибудь абзац или раздел текста. Прямоугольные фрагменты выделяют прямоугольные области или отдельные слова.

Еще один структурный элемент текста – **страница**. Разделение на страницы изображается горизонтальной линией между теми строками текста, где кончается одна страница и начинается следующая. Разделители страниц могут быть либо фиксированными, либо плавающими. Фиксированные разделители страниц ставятся вручную в таких местах, как начало главы или раздела, и их позиции не изменяются при изменении числа строк в тексте. Плавающие разделители страниц расставляются текстовым процессором автоматически путем подсчета строк, и поэтому они автоматически перемещаются при вставке или уничтожении строк. Фиксированные разделители изображаются на экране двойной линией, а плавающие – одинарной.

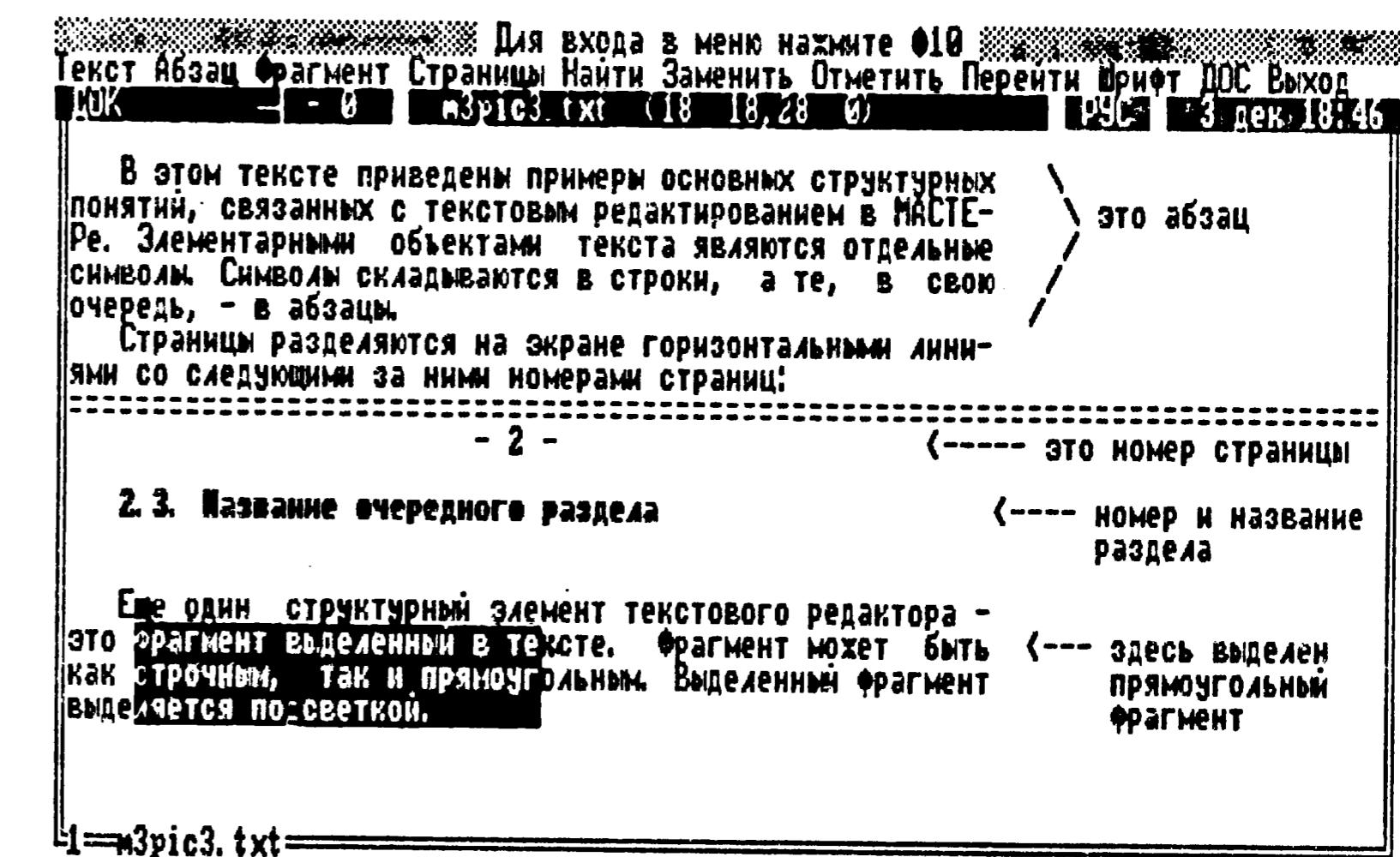


Рис. 3.3. Структурные элементы текста

Функциональные возможности текстового редактирования

Возможности текстового процессора МАСТЕРа сгруппируем в следующие пять функциональных групп:

общее управление, т.е. движение по текстовому документу, выбор частей текста, поиск определенных позиций в тексте по заданным подстрокам;

редактирование текста, т.е. ввод новых частей текста, уничтожение или изменение существующих частей, изменение шрифтов символов, контекстный поиск и замена подстрок;

форматирование абзацев, т.е. управление положением границ абзацев, переформатирование абзацев после редактирования их содержимого, управление режимами переноса слов и выравнивания абзацев;

разбивка на страницы, т.е. расстановка в тексте разделителей страниц с учетом межстрочных расстояний и физического размера листа, расстановка номеров страниц;

операции над текстами в целом, т.е. сохранение их на диске, распечатка.

3.2. Общее управление текстовым процессором

Текстовые рамки

Прежде чем начать работать с текстовым процессором в системе МАСТЕР, вам потребуется создать рамку, предназначенную для хранения и обработки текста. Создание рамки выполняется командой

{Ф10} Рамка Создать Текст

После создания рамке можно придать произвольный внешний вид: выбрать положение, размеры, цвет. Это делается командами рамкового меню. Следует напомнить, что размеры текстовой рамки никоим образом не ограничивают редактируемого в ней текста. Рамка может быть совсем небольшой и содержать тем не менее многостраничный и большой по ширине текст.

Текстовое меню

При попадании курсора в текстовую рамку изменяется вид меню МАСТЕРа. Вместо общих операций, таких, как База или Диск, появляются операции, специфичные для текстовой обработки – Найти, Заменить, Шрифт, Абзац, Фрагмент и др. Общие операции МАСТЕРа по-прежнему представлены пунктом Общие.

Этот вариант меню будем называть **текстовым меню**. Некоторые пункты, как и в главном меню МАСТЕРа, вызывают вспомогательные меню. Двухуровневая структура текстового меню показана в табл. 3.1.

Таблица 3.1. Операции текстового меню

Пункт	Действие
Абзац	Переформатирование абзаца
Формат	Переключение режима форматирования
Режим	Переформатирование всех абзацев
ГлобФорм	Включение и выключение переноса
Перенос	Переключение режима выравнивания
Выравнивание	Задание левой границы абзаца
Левая	Задание правой границы абзаца
Правая	Задание абзацного отступа
Отступ	Настройка на абзац по образцу
Образец	
Фрагмент	Выделение строкового фрагмента
Выделить	Выделение прямоугольного фрагмента
Строчный	Забрать фрагмент в карман
Прямоугольный	
Забрать	Вставка строкового фрагмента
Вставить	Вставка прямоугольного фрагмента
Строчный	Отмена выделения фрагмента
Прямоугольный	
Снять	Расстановка разделителей страниц
Выделение	Задание межстрочного расстояния
Страницы	Задание высоты страницы
Расставить	Уничтожение разделителей страниц
Страницы	Задание контекста поиска
Расстояние	Задание контекстов поиска и замены
Высота	Запоминание позиции в тексте
Убрать	Переход к запомненной позиции
Страницы	Изменение текущего шрифта
Найти	Меню общих действий МАСТЕРа
Заменить	
Отметить	
Перейти	
Шрифт	
Общие	

Индикатор местоположения в текстовой рамке

После входа в текстовую рамку обратите внимание на индикатор местоположения, находящийся в статус-строке. В его левой части виден перечень тех рамок, внутри которых вы находитесь по рамковой иерархии, а затем в круглых скобках показана текущая позиция. Эта позиция характеризуется четырьмя числами: полным числом строк в тексте, номером теку-

ней строки, номером текущего столбца, номером шрифта текущего символа. Примером показания индикатора местоположения может быть

ИмяВашейРамки. (206 1,46 0)

Этот индикатор означает, что вы находитесь внутри рамки с именем "ИмяВашейРамки" на позиции 1,46 (первая строка, 46-й столбец). Число 206 означает общее число строк, созданных в этой текстовой рамке, а число 0 – означает номер шрифта того символа, на котором стоит курсор.

Перемещение курсора

Редактирование в экранном режиме основано на текущем положении курсора. Ввод и удаление символов, создание и уничтожение строк, выполнение всех прочих операций выполняется относительно текущей позиции курсора. Поэтому основой экранного редактирования является возможность перемещения курсора по тексту. Для этого используются не только клавиши-стрелки, но и некоторые другие клавиши, расположенные на функциональной цифровой части клавиатуры и представленные в табл. 3.2.

Таблица 3.2. Клавиши перемещения курсора по тексту

Клавиши	Перемещение курсора
{ВВЕРХ} {ВНИЗ} {ВПРАВО} {ВЛЕВО}	Перемещение курсора на одну позицию в соответствующем направлении
{УПР-ВЛЕВО} {УПР-ВПРАВО}	Перемещение курсора на одно слово влево или вправо
{ПРЕД} {СЛЕД}	Перемещение курсора на несколько строк вверх или вниз
{НАЧАЛО} {КОНЕЦ}	Перемещение курсора в начало или конец строки
{ЗАГ-ВЛЕВО} {ЗАГ-ВПРАВО}	Перемещение курсора на левую или правую границы рамки
{ЗАГ-ВВЕРХ} {ЗАГ-ВНИЗ}	Перемещение курсора на самую первую или на самую последнюю строки текста
{ЗАГ-ПРЕД} {ЗАГ-СЛЕД}	Перемещение курсора на заголовок предыдущего или следующего раздела
{УПР-ПРЕД} {УПР-СЛЕД}	Перемещение курсора на начало предыдущей или следующей страницы

Переход к строке по поисковому контексту

Другой способ быстрого перехода к той или иной строке состоит в нахождении этой строки по вхождению в нее определенной комбинации символов – **поискового контекста**. Для этого в меню имеется команда Найти, с помощью которой можно задать поисковый контекст и выполнить поиск первого вхождения этой строки. Поиск осуществляется, начиная с текущей позиции курсора вниз по тексту. При повторном использовании этой операции в запросе поискового контекста предлагается тот контекст, который был использован ранее. С ним можно согласиться простым нажатием клавиши {ВВОД} или изменить любым требуемым способом.

Задав однажды поисковый контекст командой Найти, можно в дальнейшем выполнять операцию поиска одним нажатием функциональной клавиши {УПР-Ф1}. При этом уже не запрашивается поисковый контекст, а ищется контекст, заданный ранее. Таким образом, нажатиями клавиши {УПР-Ф1} можно быстро перебирать одно за другим все вхождения определенного контекста.

Сдвиг текста в рамке

Текст в рамке, как правило, не может быть виден целиком. По мере передвижения курсора происходит автоматический сдвиг текста в окне таким образом, чтобы курсор всегда находился в поле зрения. При переходе со строки на строку происходит перемещение текста по вертикали. При горизонтальном движении текст сдвигается в горизонтальном направлении.

Кроме того, имеется возможность явного сдвига текста в окне. Для этого служат клавиши {Ф5} и {Ф6}. Клавиша {Ф5} сдвигает текст влево, а клавиша {Ф6} – вправо.

3.3. Редактирование текста

Основная операция при работе с текстом – это ввод в него новых символов, что осуществляется нажатиями соответствующих знаковых клавиш. Нажатый символ вводится в ту позицию, где в момент нажатия стоит курсор. Курсор при этом автоматически сдвигается на одну позицию направо.

Режимы ввода символов: вставка и замена

Ввод символов в текст производится двумя разными способами. В одном режиме, называемом **режимом вставки**, символы, на-

ходящиеся справа от курсора, отодвигаются вправо на одну позицию при каждом нажатии символьной клавиши. Тем самым новые символы вставляются между старыми, не стирая их. В другом режиме, называемом **режимом замены**, символы, находящиеся справа от курсора, не сдвигаются, а новые символы вводятся вместо старых, заменяя их.

Как правило, более удобно работать в режиме вставки, и редактор по умолчанию находится именно в этом режиме. Режим замены удобен только в тех текстах, которые имеют вид форматных бланков или таблиц; раздвижка символов нарушила бы форматную структуру текста. Для переключения режимов служит клавиша {ВСТ}, последовательные нажатия которой переключают режим ввода символов с режима вставки на режим замены и наоборот. Состояние режима вставки-замены отображается в индикаторе состояния (в левой части статус-строки). Если текст находится в режиме замены, то в этом поле высвечивается флагок **ЗАМ**, а в режиме вставки этот флагок отсутствует.

Русские и латинские буквы

Еще один режим, определяющий характер ввода символов, – это режим состояния клавиатуры. Клавиатура может находиться либо в русском, либо в латинском состояниях. Для переключения этого состояния используется в зависимости от конфигурации компьютера и МАСТЕРа одна из клавиш: {Ф9}, {CAPS-LOCK} или {РУС} и {ЛАТ}; их последовательные нажатия переключают клавиатуру из русского состояния в латинское и обратно. Текущее состояние клавиатуры отображается флагками РУС или ЛАТ в индикаторе клавиатуры (в правой части статус-строки).

Создание новых строк

Для образования новых строк служит клавиша {ВВОД}. Действие этой клавиши аналогично действию клавиши возврата каретки на пишущих машинках и состоит в переводе курсора на начало следующей строки. При этом если текст находится в состоянии вставки, то строки текста раздвигаются за счет образования новой строки, а остаток текущей строки справа от курсора перемещается в новую строку вместе с курсором (рис. 3.4). Иными словами, действие клавиши {ВВОД} в режиме вставки состоит в том, что текущая строка разрывается на две строки, причем границей разрыва служит положение курсора. В частности, если курсор находится в самом конце строки, то выполнение клавиши {ВВОД} приводит к образованию новой пустой строки.

Для входа в меню нажмите **Ф18**
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Пронт ЛС Выход
Ф10
или словами, действие клавиши {ВВОД} в режиме вставки состоит в том, что текущая строка разрывается на две строки, причем границей разрыва служит положение курсора. В частности, если курсор находится в самом конце строки, то выполнение клавиши {ВВОД} приводит к образованию новой пустой строки.

Для входа в меню нажмите **Ф10**
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Пронт ЛС Выход
Ф10
или словами, действие клавиши {ВВОД} в режиме вставки состоит в том, что текущая строка разрывается на две строки, причем границей разрыва служит положение курсора. В частности, если курсор находится в самом конце строки, то выполнение клавиши {ВВОД} приводит к образованию новой пустой строки.

Рис. 3.4. Разделение строки клавишей {ВВОД}

В режиме замены нажатие клавиши {ВВОД} просто переводит курсор на начало очередной строки без создания новых строк.

Переход курсора на новую строку не требует явного нажатия клавиши {ВВОД}. Если текст находится в состоянии автоматического форматирования абзацев, то всякий раз при достижении курсором правой границы абзаца он автоматически переходит на следующую строку, разделяя слово, которое пересекло эту позицию, по правилам переноса. Таким образом, вводя текст, можно не заботиться о пересечении правой границы абзаца. Нужно просто нажимать символьные клавиши, а описанное правило переноса курсора обеспечит создание абзаца правильной формы. И только при достижении конца абзаца нужно нажать клавишу {ВВОД} – это послужит началом нового абзаца, и курсор встанет в позицию абзацного отступа.

Стирание символов и слияние строк

Для стирания символов служат две клавиши – {УДЛ} и {ОТМЕНА}. Первая из них стирает текущий символ, а вторая – предыдущий символ. Обе эти клавиши можно нажимать вместе с модификатором {ЗАГ}. Нажатие клавиши {ЗАГ-ОТМЕНА} означает уничтожение всех символов перед курсором от начала строки. Клавиша {ЗАГ-УДЛ} уничтожает все символы, начиная от курсора до правого конца строки.

Особым действием в режиме вставки обладает клавиша {УДЛ}, когда курсор находится в самом конце строки и никакого текущего символа уже не существует. В этом случае считается, что текущим символом является символ перевода строки, а его стирание означает, что следующая строка соединяется с текущей. Тем самым в этом состоянии клавиша {УДЛ} обеспечивает слияние строк (рис. 3.5).

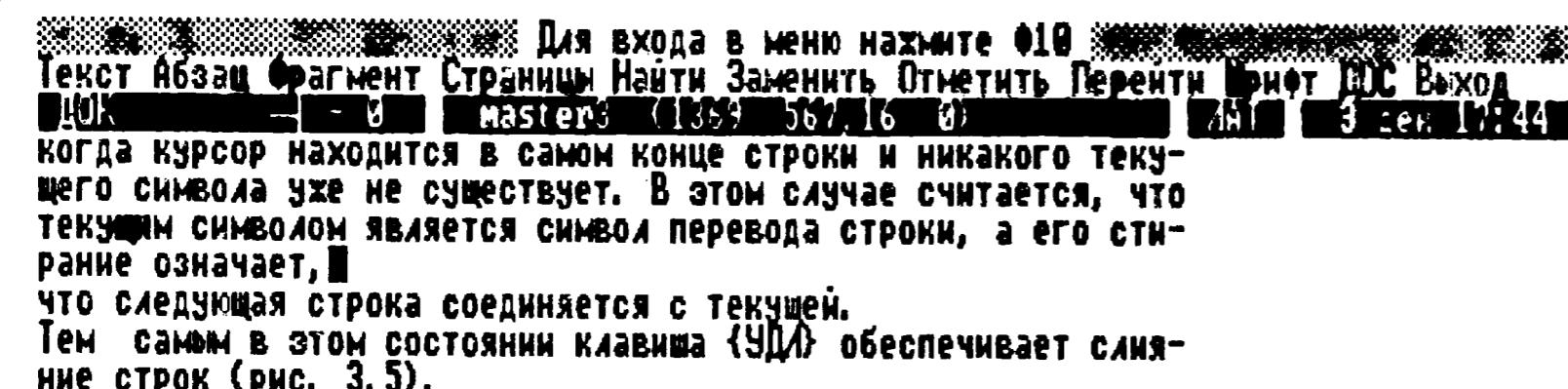


Рис. 3.5. Слияние строк клавишей {УДЛ}

Переключение шрифтов

В текстовом процессоре МАСТЕРа можно использовать несколько шрифтов, показанных на рис. 3.6. Различие шрифтов проявляется в тексте на экране по-разному в зависимости от состояния экрана. В графическом состоянии различные шрифты имеют соответствующую форму, а в символьном состоянии различным шрифтам соответствуют разные цвета символов.

Управление шрифтами основывается на понятии **текущего шрифта**. Шрифты идентифицируются номерами от 0 до 3. Номер текущего шрифта виден в поле состояния. Так, в нормальном состоянии, когда текущим является стандартный прямой шрифт, высвечен номер 0. Кроме этого номера текущего шрифта, высвечиваемого в индикаторе состояния, в индикаторе местоположения высвечивается еще и номер шрифта того символа, на котором стоит курсор. Этот номер высвечивается в индикаторе местоположения вслед за номерами строки и столбца.

Варианты шрифтов в текстовом процессоре МАСТЕРа:	
Номер шрифта	Внешний вид шрифта на экране
0	Прямой стандартный шрифт
1	Курсив (наклонный шрифт)
2	Подчеркнутый шрифт
3	Выделенный курсив

Рис. 3.6. Варианты шрифтов на экране

Для переключения шрифтов имеется несколько клавиш. Во-первых, шрифт можно включить непосредственно по его номеру. Для этого служит клавиша {ДОП-Ф1}. После ее нажатия выдается запрос "укажите номер шрифта", в ответ на который вы должны нажать одну из цифр: 0, 1, 2 или 3, обозначающую номер выбранного шрифта. Вторая возможность состоит в том, чтобы включать или выключать наклон символов независимо от их жирности либо, наоборот, включать или выключать жирность символов независимо от их наклона. Наклон символов включается и выключается последовательными нажатиями клавиши {ДОП-Ф3}. Жирность символов включается и выключается последовательными нажатиями клавиши {ДОП-Ф4}. Этими двумя клавишами можно пользоваться независимо, например, включить курсив нажатием {ДОП-Ф3}, а затем добавить жирность нажатием {ДОП-Ф4}, после чего можно в любой последовательности выключать жирность и курсив.

Помимо жирности и наклона символов можно еще подчеркивать их. Для этого служит клавиша {ДОП-Ф2}. Ее последовательные нажатия включают и выключают подчеркивание.

Все определенные в этом разделе клавиши действуют по-другому, если в тексте выделен некоторый фрагмент. В этом состоянии они изменяют не текущий шрифт, а шрифт тех символов, которые находятся в выделенном фрагменте. Например, клавиша {ДОП-Ф1} позволяет установить у всех символов выделенного фрагмента один и тот же шрифт путем явного указания его номера. Клавиша {ДОП-Ф2} выполняет подчеркивание всех неподчеркнутых символов и убирает подчеркивание у всех подчеркнутых символов. Подобным же образом клавиша {ДОП-Ф3} меняет

наклонные буквы на прямые и наоборот, а клавиша {ДОП-Ф4} меняет полужирные символы на обычные и наоборот.

Переключение шрифтов действует только на экране ПЭВМ. Для того, чтобы те же самые шрифты оказались распечатанными на бумаге, требуется специальная программа, которая входит в состав интегрированной системы МАСТЕР. Эта программа может запускаться либо непосредственно из МАСТЕРа, либо самостоятельно.

Контекстный поиск и замена

Операции, связанные с контекстными поиском и заменой, представлены в текстовом меню пунктами Найти и Заменить. Выполнение каждой из команд начинается с вопросов о вводе контекстов. Команда Найти запрашивает ввод поискового контекста, команда Заменить запрашивает ввод поискового и заменяющего контекстов.

Поисковым контекстом называется подстрока, вхождения которой затем будут отыскиваться в тексте. **Заменяющим контекстом** называется подстрока, которая будет вставляться в текст вместо найденного поискового контекста. После срабатывания любой из рассматриваемых операций контексты запоминаются в памяти редактора, а сами операции поиска и замены можно делать простым однократным нажатием функциональных клавиш. Для выполнения поиска служит клавиша {УПР-Ф1}, а для замены только что найденного контекста – клавиша {УПР-Ф2}. При очередном обращении к операциям поиска или замены через меню вам будут предлагаться заломленные ранее контексты, которые можно либо оставить неизменными, сразу нажимая {ВВОД}, либо отредактировать.

В операции Найти после задания поискового контекста выполняется поиск этого контекста, и курсор устанавливается на его начало. После этого поиск можно многократно повторять нажатием одной клавиши {УПР-Ф1}. Повторные нажатия этой клавиши будут приводить к поиску очередных вхождений этого же контекста – таким образом можно просматривать их последовательно друг за другом.

В операции Заменить после задания обоих контекстов – поискового и заменяющего – выполняется поиск первого вхождения поискового контекста. Замена в этот момент не выполняется. Собственно замена всегда производится только с помощью функциональной клавиши {УПР-Ф2}. Эта клавиша должна нажиматься после успешного нахождения поискового контекста. Если курсор будет сдвинут и в момент нажатия окажется не на начале поискового контекста, то замена не будет выполнена.

При необходимости произвести многократную замену можно нажимать последовательно клавиши {УПР-Ф1} и {УПР-Ф2}. Тогда будут одно за другим находиться и заменяться вхождения контекстов. Некоторые из вхождений можно пропускать, не нажимая клавиши замены {УПР-Ф2}. Однако если вы хотите заменить не некоторые, а все вхождения и если этих вхождений очень много, то можно воспользоваться клавишей глобальной замены – {ЗАГ-Ф2}. Эта клавиша заменяет все вхождения поискового контекста на заменяющий от текущей позиции курсора до конца текста. Если ту же клавишу {ЗАГ-Ф2} нажать в режиме выделенного строкового фрагмента, то замена будет произведена только в пределах выделенной части текста.

Работа с фрагментами текста

Помимо действий над отдельными символами текстовый процессор может выполнять некоторые операции над целыми фрагментами текста. Для выделения фрагментов служат клавиши {Ф3} и {ЗАГ-Ф3}.

Клавиша {Ф3} выделяет строковый фрагмент, т.е. фрагмент, включающий целиком несколько строк. После нажатия этой клавиши следует двигать курсор вверх или вниз, расширяя выделение от той строки, где была нажата клавиша {Ф3}, до той строки, где находится курсор (рис. 3.7).

Для входа в меню нажмите Ф10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перемычка Файл Выход
Ф3 OK master3 (1063 731,4 0) ЗАГ-Ф3 Вид Дек 14:58
текста. Если ту же клавишу {ЗАГ-Ф2} нажать в режиме выделенного строкового фрагмента, то замена будет произведена только в пределах выделенной части текста.

Работа с фрагментами текста

Помимо действий над отдельными символами текстовый процессор может выполнять некоторые операции над целыми фрагментами текста. Прежде чем выполнять любую из таких операций, нужно выделить фрагмент, для чего служат клавиши {Ф3} и {ЗАГ-Ф3}.

Клавиша {Ф3} выделяет строковый фрагмент, т.е. фрагмент, включающий целиком несколько строк. После нажатия этой клавиши следует двигать курсор вверх или вниз, расширяя выделение, которое распространяется от той строки, где была нажата клавиша {Ф3}, до той строки, где находится курсор. Все эти строки выделяются полсветкой (рис. 3.7).

Клавиша {ЗАГ-Ф3} выделяет прямоугольный фрагмент – прямоугольник, пересекающий несколько строк. После нажатия этой клавиши выделяется не целая строка, а только та сим-

Рис. 3.7. Выделение строкового фрагмента

Клавиша {ЗАГ-Ф3} выделяет **прямоугольный фрагмент** – **прямоугольник**, пересекающий несколько строк. После нажатия этой клавиши выделяется не целая строка, а только та символьная позиция, в которой стоит курсор. Расширение фрагмента выполняется не только вертикальными, но и горизонтальными стрелками (рис. 3.8).

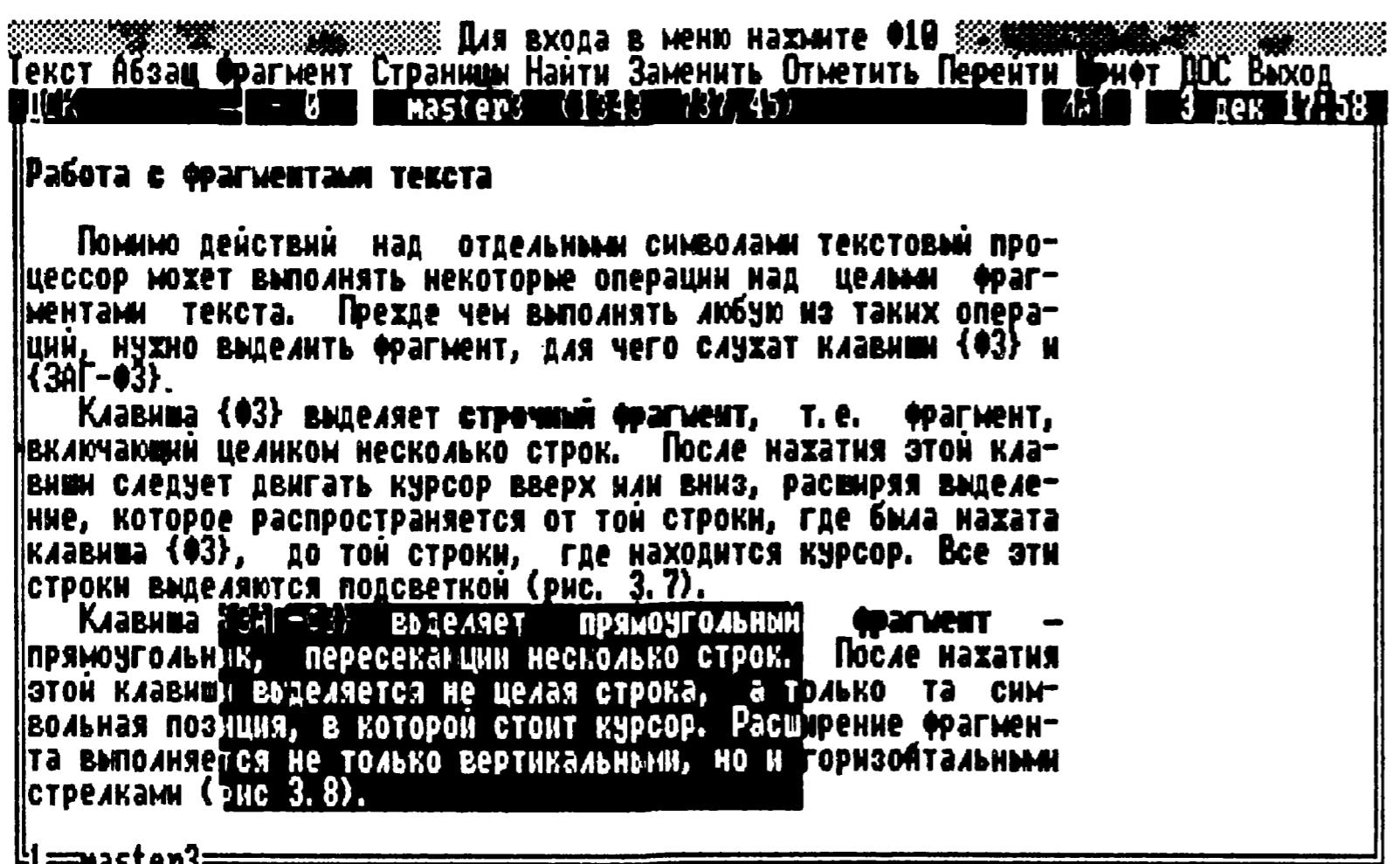


Рис. 3.8. Выделение прямоугольного фрагмента

В режиме выделенного фрагмента текстовый процессор находится в ином состоянии, нежели при обычном редактировании. В этом режиме невозможно вводить и стирать символы, а действуют только операции над фрагментами. Для того чтобы вернуться из этого состояния в состояние обычного редактирования, нужно нажать клавишу {Ф4}, которая выполняет **отмену выделения**. Она ничего не делает с выделенным фрагментом, а просто возвращает текстовый процессор в состояние редактирования.

Для работы с фрагментами в текстовом процессоре имеется специальное временное хранилище, называемое **карманом**. Главная операция над фрагментом состоит в том, что выделенный фрагмент можно "забрать" в этот карман, запомнив его там на некоторое время, а в дальнейшем несколько раз "вынимать" из кармана, вставляя копию в другие места текста или даже в другие текстовые рамки. Выделенный фрагмент забирается в карман с помощью клавиши {УПР-Ф3}. При этом фрагмент изымается из текста. Таким образом, операция {УПР-Ф3} может ис-

пользоваться и для запоминания фрагмента в кармане, и для уничтожения его из текста.

Для того чтобы вставить запомненный в кармане фрагмент в текст, следует поставить курсор на ту строку, перед которой нужно сделать вставку, и нажать клавишу {УПР-Ф4}. При этом копия содержимого кармана будет вставлена перед текущей строкой, а курсор окажется на первой строке вставленного фрагмента. Сам вставленный фрагмент окажется выделенным, так что над ним можно сразу же произвести какие-то другие фрагментные операции, например сдвинуть вправо или влево либо удалить.

Фрагмент из кармана можно вставлять не только путем образования новых строк, но и путем образования новых столбцов. Для этого служит клавиша {ЗАГ-Ф4}. При нажатии этой клавиши фрагмент вставляется в виде прямоугольника в уже существующие строки, начиная с текущей строки и пересекая несколько нижележащих строк (в соответствии с высотой фрагмента). Если текст находится в состоянии вставки, то фрагмент вставляется, отодвигая вправо символы, находящиеся правее курсора, а в состоянии замены фрагмент вставляется, накладываясь на существующие символы.

После вставки фрагмента из кармана клавишами {УПР-Ф4} или {ЗАГ-Ф4}, если вы не хотите совершить над ними какой-нибудь фрагментной операции, нужно нажать клавишу {Ф4}, чтобы отменить выделение только что вставленного фрагмента.

После выполнения команды взятия фрагмента в карман клавишей {УПР-Ф3} содержимое кармана остается неизменным до очередного выполнения этой же команды. Фрагмент, взятый таким образом в карман, можно многократно вставлять в текст в разных местах с помощью клавиш {УПР-Ф4} или {ЗАГ-Ф4}. В частности, фрагмент, взятый в карман в одной рамке, можно вставить в текст в другой рамке, осуществляя тем самым перенос информации между рамками. Точно так же можно, взяв фрагмент в карман из рамки, вставить его в текст, находясь в поле значения при ответе на какой-нибудь вопрос (скажем, при определении макроклавиши или формулы локального действия клавиши). Это позволяет обмениваться через карман информацией между любыми точками информационной среды МАСТЕРа, в которых может действовать текстовый процессор.

Еще одна операция, которую требуется выполнять над группами строк, – это сдвиг их по горизонтали. Это требуется особенно часто при работе с текстами программ, которые имеют ступенчатую структуру. В частности, часто требуется "подправить" только что вставленный строковый фрагмент, углубив или выдвинув все строки фрагмента одновременно. Для этого служат операции сдвига строкового фрагмента, реализуемые клавишами

{УПР-Ф5} и {УПР-Ф6} (сдвиги строкового фрагмента влево и вправо на одну позицию).

В пределах выделенного фрагмента возможны некоторые редактирующие операции. Например, клавиша глобальной замены {ЗАГ-Ф2} в состоянии, когда в тексте выделен фрагмент, производит замену не по всему тексту, а только в пределах выделенного фрагмента. При этом изменяется также и действие клавиш переключения шрифтов {ДОП-Ф1}, {ДОП-Ф2}, {ДОП-Ф3} и {ДОП-Ф4}. Вместо переключения текущего шрифта в этом состоянии они изменяют шрифт у всех символов выделенного фрагмента, как описано выше.

3.4. Форматирование абзацев

Очень важной функцией текстового процессора является автоматическое формирование абзацев, т.е. расстановка слов по строкам таким образом, чтобы они по возможности плотно заполняли строки, были выровнены по левому и правому краям с использованием, если надо, переносов слов по слогам. Если выполнять эту операцию вручную, то после каждой вставки или удаления слова внутри абзаца приходилось бы делать множество редактирующих операций для перестановки слов в нем. Во избежание такой ручной работы имеется возможность выполнять ее автоматически одним нажатием функциональной клавиши. Рассмотрим все связанные с этим механизмы управления.

Включение и выключение форматирующего режима

Возможность автоматического формирования включена в текстовом процессоре не всегда, поскольку для некоторых текстов (как, например, в программах на языках программирования) формирования абзацев не требуется.

Форматирующий режим помогает правильно оформлять абзацы, выравнивая их по правому и по левому краям, делая переносы слов. В этом режиме абзацы можно форматировать непосредственно при вводе текста — символы можно вводить, не заботясь о пересечении правой границы абзаца, — слова будут автоматически перенесены на новую строку по правилам переноса. При этом не требуется вообще никаких команд для форматирования абзаца — достаточно просто вводить текст. Другая возможность состоит в том, чтобы переформатировать абзац по команде с клавиатуры — это требуется после того, как абзац был "испорчен" теми или иными исправлениями в нем.

В информатирующем текстовом режиме всего этого не происходит. Этот режим ориентирован на ввод и редактирование структурных программных текстов. Единственным подспорьем, которое редактор оказывает в данном режиме, — это установка курсора в правильную структурную позицию при вводе новых строк — курсор при вводе новой строки устанавливается на текущем структурном уровне, избавляя пользователя от необходимости вводить много пробелов в начала строк.

Переключение между этими двумя режимами осуществляется клавишей {ЗАГ-Ф10}. Ее последовательные нажатия изменяют режим на противоположный. Текущее состояние режима форматирования отображено в индикаторе состояния. При включенном режиме форматирования в поле состояния виден флажок ДОК, а при выключенном режиме — индикатор ТЕКСТ.

Установка границ абзацев

Абзац представляет собой последовательность строк, начинающихся с одной и той же позиции с левой стороны. Одна только первая строка абзаца имеет особое начало — она может быть сдвинута относительно левой границы вправо или влево, образуя абзацный отступ или выступ. Кроме того, при форматировании абзаца должна быть задана его правая граница, при пересечении которой слова переносятся на следующую строку. К этой же правой границе слова выравниваются, если это требуется, путем раздвижки пробелов внутри строки.

Перед вводом текста или выполнением форматирующих операций необходимо задать значения этих трех позиций — левой и правой границы абзаца и размера абзацного отступа от левой границы. Установка этих позиций может быть сделана тремя способами.

Проще всего установить границы абзаца "по образцу". Подведя курсор к первой строке некоторого абзаца-образца и нажав клавишу {ЗАГ-Ф7}, вы настроите текстовый процессор на указанный абзац. Для определения структуры абзаца требуется, чтобы в абзаце были по крайней мере две строки. При нажатии клавиши {ЗАГ-Ф7} с первой строки абзаца считаются правая граница и абзацный отступ, а с второй строки — левая граница. Эта возможность является наиболее удобной для переключения с одного вида абзацев на другой. Документ, как правило, состоит из абзацев двух — трех видов. При желании настроиться на другой вид абзаца нужно подвести курсор к ближайшему абзацу подходящего вида и нажатием клавиши {ЗАГ-Ф7} считать его структуру.

Второй способ установки границ абзаца состоит в использовании текстового меню. Для этого в нем имеются три операции:

- {Ф10} Абзац Левая – задание левой границы;
- {Ф10} Абзац Правая – задание правой границы;
- {Ф10} Абзац Отступ – задание размера абзацного отступа.

В этих операциях все три позиции задаются числовыми значениями. Для левой и правой границ это числовое значение означает номер столбца, а для абзацного отступа это значение задает сдвиг первой строки абзаца относительно его левого края и может быть как положительным, так и отрицательным. Например, значения границ для текста в данной книге таковы: левая граница = 1, правая граница = 60, абзацный отступ = 3.

Третий способ состоит в указании каждой из трех позиций с помощью курсора и функциональных клавиш {Ф7}, {УПР-Ф7}, {ДОП-Ф7}. Установив курсор в требуемую позицию, нужно нажать одну из следующих трех клавиш, запоминающих соответствующие три границы абзаца по текущей позиции курсора:

- {Ф7} – установка левой границы;
- {ДОП-Ф7} – установка правой границы;
- {УПР-Ф7} – установка размера абзацного отступа относительно левой границы.

Установку абзацного отступа нужно производить в этом варианте обязательно после установки левой границы, поскольку отступ имеет относительный характер. Он устанавливается как разность между позицией, в которой нажата клавиша {УПР-Ф7}, и установленной ранее левой границей. Этот относительный размер отступа сохраняется при последующих изменениях левой границы.

Оформление абзацев при вводе текста

Если включен режим форматирования и текст просто вводится с клавиатуры символ за символом, то автоматическое срабатывание форматирования приводит к тому, что абзац приобретает правильную форму, выровненную справа и слева, со словами, перенесенными по слогам. Это показано на рис. 3.9. В первой половине рисунка курсор находится уже за правой границей абзаца, и нажатие очередного символа в этом состоянии приводит к автоматическому переходу курсора на следующую строку вместе с частью слова, перенесенного по правилам русского языка.

Для входа в меню нажмите Ф10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Файл ДС Выход
Блок 1990 год 18:15

Если включен режим форматирования, и текст просто вводится с клавиатуры символ за символом, то автоматическое срабатывание форматирования приводит к тому, что абзац приобретает правильную форму, выровненную справа и слева, со словами, перенесенными по слогам. Это показано на рис. 3.9. В первой половине этого рисунка курсор находится уже за правой границей.

Для входа в меню нажмите Ф10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Файл ДС Выход
Блок 1990 год 18:15

Если включен режим форматирования, и текст просто вводится с клавиатуры символ за символом, то автоматическое срабатывание форматирования приводит к тому, что абзац приобретает правильную форму, выровненную справа и слева, со словами, перенесенными по слогам. Это показано на рис. 3.9. В первой половине этого рисунка курсор находится уже за правой границей.

Рис. 3.9. Автоматический перенос слова

Когда ввод текста достигает конца абзаца, и вы хотите перейти к следующему, нужно нажать клавишу {ВВОД} – ее срабатывание переведет курсор на новую строку и установит его в позицию абзацного отступа. Тем самым будет начат очередной абзац. Таким образом, клавиша {ВВОД} в режиме форматирования играет роль окончания предыдущего абзаца и начала нового.

Обратите внимание на то, что если эта же клавиша {ВВОД} нажата не при вводе текста, а при редактировании (т.е. при нахождении курсора не справа от вводимой строки, а внутри нее), то курсор, перескакивая на следующую строку, встает уже в позицию левой границы, и тем самым абзац не разрывается абзацным отступом.

Переформатирование абзацев в готовом тексте

При первоначальном вводе текста строки автоматически форматируются текстовым процессором, что приводит к выравниванию правого и левого краев абзаца. Если после этого изменить текст с помощью тех или иных редактирующих операций, то потребуется заново переформатировать некоторые абзацы, испорченные редактированием. Для этого служит команда переформатирования, выполняемая клавишей {УПР-Ф8}.

Обычно при редактировании оказывается испорченным правый край абзаца, поскольку в результате вставки и уничтожения символов сдвигаются вправо или влево те части строк, которые находятся справа от курсора. Левые границы строк остаются выровненными и не затрагиваются редактированием. Поэтому после редактирования абзаца обычно требуется исправить только правый его край. Для того чтобы сделать это, нужно поставить курсор на первую строку абзаца и нажать клавишу {УПР-Ф8}. Действие этой клавиши состоит в переформатировании всех строк, начиная от той, где стоит курсор, до последней строки абзаца, т.е. до очередного абзацного отступа или выступа (рис. 3.10). Переформатирование абзаца выполняется в этом случае без учета текущей установки левой границы, сделанной клавишами {Ф7}, {ЗАГ-Ф7} или через меню. Левые границы строк сохраняются такими, какими они были в форматируемом абзаце. Это позволяет одним нажатием клавиши исправлять абзацы с разными левыми границами, не заботясь о перенастройке левой границы. В отличие от левой границы правая граница формируется по текущей установке, сделанной клавишами {ДОП-Ф7}, {ЗАГ-Ф7} или через меню. Как правило, для всех абзацев одного текста правая граница одинакова. В тех же редких случаях, когда это не так, вам придется перед переформатированием абзаца нужным образом настроить правую границу.

Для входа в меню нажмите Ф10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Бриф ПС Выход
Мастер (1345 1036 2 0) 3 лист 18:18

Переформатирование абзацев в готовом тексте

■ При первоначальном вводе текста строки автоматически форматируются текстовым процессором, что приводит к выравниванию правого и левого краев абзаца.

Если после этого изменить текст с помощью тех или иных редактирующих операций, то потребуется заново переформатировать некоторые абзацы, испорченные редактированием, выполняемая клавишей {УПР-Ф8}.

Для входа в меню нажмите Ф10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Бриф ПС Выход
Мастер (1345 1036 2 0) 3 лист 18:18

Переформатирование абзацев в готовом тексте

■ При первоначальном вводе текста строки автоматически форматируются текстовым процессором, что приводит к выравниванию правого и левого краев абзаца. Если после этого изменить текст с помощью тех или иных редактирующих операций, то потребуется заново переформатировать некоторые абзацы, испорченные редактированием. Для этого служит команда переформатирования, выполняемая клавишей {УПР-Ф8}.

Рис. 3.10. Исправление правого края клавишей {УПР-Ф8}

Иногда требуется переформатировать абзац таким образом, чтобы, напротив, изменить его левую границу. Это может быть нужно, например тогда, когда вы хотите объединить несколько абзацев в один или когда форматируемый текст вообще не имеет абзацной структуры. В этом случае следует воспользоваться той же клавишей {УПР-Ф8}, но предварительно нужно выделить (через клавишу {Ф3}) фрагмент тех строк, которые должны быть сформированы в единый абзац. Выделив нужный строковый фрагмент, нажмите клавишу {УПР-Ф8}, и весь этот фрагмент будет переформатирован, причем левая и правая границы, а также абзацный отступ будут установлены не по виду абзаца, как в предыдущем случае, а в соответствии с их текущими значениями, установленными клавишами {Ф7}, {ДОП-Ф7}, {УПР-Ф7}, {ЗАГ-Ф7} или через меню (рис. 3.11).

Для входа в меню нажмите Ф10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Бриф ПС Выход
Мастер (1345 1036 2 0) 3 лист 18:18

зачев одного текста правая граница одинакова. В тех же редких случаях, когда это не так, вам придется перед переформатированием абзаца нужным образом настроить правую границу.

Иногда требуется переформатировать абзац таким образом, чтобы, напротив, изменить его левую границу. Это может быть нужно, например тогда, когда вы хотите объединить несколько абзацев в один или — когда форматируемый текст вообще не имеет абзацной структуры. В этом случае следует воспользоваться той же клавишей {УПР-Ф8}, но предварительно нужно выделить (через клавишу {Ф3}) фрагмент тех строк, которые должны быть сформированы в единый абзац. Выделив нужный строковый фрагмент, нажмите клавишу {УПР-Ф8}, и весь этот фрагмент будет переформатирован.

Левая и правая границы, а также абзацный отступ будут установлены не по виду абзаца, как в предыдущем случае, а в

Для входа в меню нажмите Ф10
Текст Абзац Фрагмент Страницы Найти Заменить Отметить Перенести Бриф ПС Выход
Мастер (1345 1036 2 0) 3 лист 18:18

зачев одного текста правая граница одинакова. В тех же редких случаях, когда это не так, вам придется перед переформатированием абзаца нужным образом настроить правую границу.

Иногда требуется переформатировать абзац таким образом, чтобы, напротив, изменить его левую границу. Это может быть нужно, например тогда, — когда вы хотите объединить несколько абзацев в один или — когда форматируемый текст вообще не имеет абзацной структуры. В этом случае следует воспользоваться той же клавишей {УПР-Ф8}, но предварительно нужно выделить (через клавишу {Ф3}) фрагмент тех строк, которые должны быть сформированы в единый абзац. Выделив нужный строковый фрагмент, нажмите клавишу {УПР-Ф8}, и весь этот фрагмент будет переформатирован.

Левая и правая границы, а также абзацный отступ будут установлены не по виду абзаца, как в предыдущем случае, а в

Рис. 3.11. Переформатирование с выделением фрагмента

Управление оформлением правого края абзаца

При формировании абзацев правый край оформляется с учетом нескольких управляющих установок: требуется или не требуется перенос слов по слогам и требуется или не требуется выравнивание правого края. Выравнивание абзацев по правому краю делается за счет добавления пробелов между словами.

Включение и выключение каждой из этих двух установок делается через текстовое меню командами:

{Ф10} Абзац Выравнивание
{Ф10} Абзац Перенос

В меню команды Выравнивание и Перенос снабжены словами ДА или НЕТ, показывающими, в каком состоянии находится в данный момент соответствующий признак. Исполнение в меню любой из этих команд переключает признак на его противоположное значение.

В индикаторе состояния режим переносов слов отображается символом "-". Если этот символ виден, то режим переносов включен, в противном случае режим переносов выключен. Режим выравнивания в этом же поле отображается наличием или отсутствием условного знака вида "—|". Наличие этого знака символизирует включенность режима выравнивания абзаца по правому краю.

Центрирование строк

Строки, служащие заголовками в тексте, как правило, должны располагаться по центру. Имеется команда, которая центрирует строку автоматически – {Ф8}. Положение центра строки определяется неявно при задании правой и левой границ абзаца.

3.5. Оформление страниц

Еще одна важная задача текстового редактирования, требующая автоматизированной поддержки, состоит в разделении текста на страницы. При работе в текстовом процессоре носитель текста потенциально неограничен по длине, и текст, состоящий из многих сотен строк, может быть введен без единого разделителя страниц, как если бы вы написали его на длинном свитке бумаги. При наличии такого сплошного текста требуется перед распечаткой расставлять в нем разделители страниц.

Параметры, управляющие разбивкой страниц

Разделение текста на страницы должно выполняться в расчете на определенный физический размер страниц. Разбивка на страницы производится с учетом этого размера и расстояния между строками: чем меньше это расстояние, тем большее количество строк может разместиться на странице. Эти два параметра задаются через текстовое меню:

{Ф10} Страницы Высота – задает высоту страницы;
{Ф10} Страницы Расстояние – задает межстрочное расстояние.

Оба параметра исчисляются в "интервалах" – единицах расстояния, соответствующих минимальному шагу пишущих машинок. Таким образом, высота стандартного листа формата А4 равна 60 интервалам, а наиболее распространенным межстрочным расстоянием являются полтора или два интервала.

Команда разбивки страниц

Для автоматического разбиения текста на страницы используется команда

{Ф10} Страницы РасставитьСтраницы

При ее выполнении текстовый процессор пробегает по всему тексту от начала до конца, подсчитывает строки и через определенные промежутки расставляет плавающие разделители страниц. Эти разделители называются плавающими, потому что при очередном исполнении той же команды (после вставки новых строк или уничтожения существующих) все прежние плавающие разделители автоматически уничтожаются и появляются на новых местах.

Команда разбивки страниц действует всегда на весь текст от начала до конца независимо от текущего положения курсора. Однако ясно, что если в начальной части текста никаких изменений в отношении количества строк сделано не было, то все плавающие разделители страниц окажутся на тех же местах, где они были перед этим. Поэтому обычная технология разбивки на страницы состоит в следующем. Подготовив текст, нужно выполнить команду расстановки страниц. Затем, начиная с начала текста, следует просмотреть его, проверяя, насколько удачно распределились разделители страниц по тексту (не отсекают ли они единичные строки, заголовки и т.п.). Заметив какую-то неправильность, следует ввести в текст несколько пустых строк с целью "вытолкнуть" неуместившийся текст на следующую

страницу. После этого нужно снова выполнить расстановку страниц. Таким образом нужно действовать, просматривая текст от начала до конца.

Помимо плавающих разделителей, которые свободно перемещаются по тексту при вставке и уничтожении строк, иногда требуются такие разделители, которые жестко стоят на своих местах, независимо от количества строк. Такие разделители страниц требуются, например, перед началом каждой главы. Эти разделители называются **фиксированными разделителями страниц**. Они вставляются в текст не текстовым процессором, а самим пользователем – с помощью клавиши {ЗАГ-Ф8}. Фиксированные разделители изображаются на экране двойной горизонтальной линией в отличие от плавающих разделителей, изображающихся одинарной линией.

Управление межстрочным расстоянием

Задание значения межстрочного расстояния через меню устанавливает начальное состояние этого расстояния. Для отдельных участков текста, быть может, потребуется установить меньшее или большее расстояние, для чего нужно вставить в текст специальные управляющие строки. Сами эти управляющие строки распечатываться не будут, они влияют на межстрочное расстояние для следующих далее строк.

Управляющая строка для задания межстрочного расстояния должна начинаться со специального символа {255} (выглядящего на экране в графическом режиме в виде перевернутого вопросительного знака), после которого должна стоять буква "ш" и далее – значение расстояния, исчисляемое в "интервалах". Например, для задания расстояния в полтора интервала нужно вставить управляющую строку из пяти символов: {255}ш1.5.

Особым смыслом обладает значение межстрочного расстояния, равное нулю. В качестве значения расстояния можно указать число 0. Это значение означает не нулевое расстояние, а возврат к начальному межстрочному расстоянию. Начальным межстрочным расстоянием называется то расстояние, которое было задано для текста через меню командой

{Ф10} Страницы Шаг

Если вы желаете сделать значение этого начального межстрочного расстояния явно видимым в вашем тексте, то можно использовать управляющую строку точно такого же вида, что и для временного изменения расстояния, но вместо строчной буквы "ш" использовать прописную.

Управление нумерацией страниц

При выполнении команды расстановки разделителей страница каждой странице проставляется ее порядковый номер. Начальным номером по умолчанию является 1. Однако при составлении многокомпонентных документов некоторые части должны начинаться с произвольных номеров. Для этого в начало текста должна быть вставлена управляющая строка, задающая номер первой страницы. Эта управляющая строка состоит из специального символа {255}, затем должна идти прописная русская буква "Н" (начальный номер), после которой без пробелов должен следовать номер первой страницы.

3.6. Операции над текстами в целом

Считывание и запись текстов

Для сохранения текстовой информации на диске существуют две возможности: сохранение рамки в рамковом файле, недоступном другим программным системам, кроме МАСТЕРа, и сохранение рамки в стандартном текстовом файле. Для записи и считывания в рамковом формате следует использовать обычные команды главного меню МАСТЕРа:

{Ф10} Диск Записать
{Ф10} Диск Считать
{Ф10} Диск Выбор

Для записи рамки в текстовом формате служит команда

{Ф10} Печать Файл

Все эти команды описаны в разделе 2.4. Напомним, что перед выполнением записи рамки необходимо выйти из нее, чтобы она была исключена из охватывающей ее текущей.

Распечатка текста

Распечатка текстовой рамки выполняется через главное меню командой

{Ф10} Печать Принтер

При вызове этой операции появляется меню, позволяющее задать некоторые параметры, управляющие процессом распечатки. Это качество и плотность распечатки.

ЭЛЕКТРОННЫЕ ТАБЛИЦЫ

Качество распечатки – параметр, определяющий, насколько аккуратно нужно распечатать текст. Как правило, качество непосредственно связано со скоростью печати: чем выше качество, тем меньшей скоростью оно будет достигнуто. Уровни качества обозначаются словами ВЫСОКОЕ, СРЕДНЕЕ, НИЗКОЕ.

Плотность распечатки – параметр, задающий один из трех способов выбора ширины символов, обозначаемых словами РАВНОМЕРНАЯ, ПРОПОРЦИОНАЛЬНАЯ, УЗКАЯ. Равномерная плотность означает, что все символы на печати будут иметь одинаковую ширину. Пропорциональная плотность означает, что разные символы будут иметь индивидуальную ширину. Узкая печать означает, что печать производится максимально узкими символами.

Собственно распечатка начинается при исполнении команды

{Ф10} Печать Принтер Начать

4.1. Структура и назначение электронных таблиц

Электронные таблицы – очень мощный инструмент для редактирования и анализа информации, связанной с числовыми массивами и функциональными зависимостями данных. Они представляют собой одно из самых интересных явлений в развитии программного обеспечения персональных компьютеров 80-х годов. Идея использования персонального компьютера для имитации работы с таблицей принадлежит Д.Бриклину и Р.Франксону, разработавшим в 1978 г. первую промышленную электронную таблицу VisiCalc. В настоящее время электронные таблицы распространены очень широко и реализованы, в частности, в каждой интегрированной системе, а их приложения захватывают самые разнообразные области. Электронные таблицы основываются на весьма простых и наглядных понятиях:

- электронная таблица представляет собой прямоугольную матрицу, состоящую из ячеек;
- строки электронной таблицы пронумерованы числами, а столбцы помечены одиночными и двойными буквами; из этих букв и чисел образуются **указатели ячеек**, такие, как В2 или АГ14;
- в любую ячейку таблицы можно поместить **значение** – число, или строку, текст или рисунок;
- если значение в одной ячейке должно зависеть от значений каких-то других ячеек (например, должно быть их суммой), то вместо конкретного числа в нее можно поместить **формулу**, определяющую способ вычисления значения; например, в ячейку А5 можно ввести формулу А3+А4;
- в электронной таблице может быть сделан **пересчет**, т.е. вычисление всех входящих в нее формул, что приведет к появлению вычисленных значений в тех ячейках, где хра-

нятся формулы. Пересчет выполняется либо автоматически при изменении значений ячеек, либо по специальному указанию с клавиатуры.

Примером электронной таблицы может служить ведомость расходов, изображенная на рис. 4.1. В этой таблице каждая строка представляет запись о некотором расходе денег. В столбце в содержится фамилия получателя, в столбце Г – величина расхода, а в столбце Б – указание того фонда, из которого производится этот расход, обозначенного одним из условных обозначений "з/п", "мат", "фрп", "скб", "прем". Столбцы от Д до И содержат в верхней части эти условные обозначения и под ними в каждой строке остаток соответствующего фонда после очередной выплаты. Во второй строке воспроизведенного на рисунке экрана вы можете видеть ту формулу, которая находится ячейке Д4 и вычисляет остаток фонда "з/п" для первой выплаты. Во всех остальных ячейках диапазона Д4:И21 находятся аналогичные формулы.

Рис. 4.1. Электронная таблица для учета фондов

Дата	Фонд	Получатель	Сумма	з/п	скб	мат	фрп	прем
		Начальные фонды ----->	500.00	200.00	250.00	100.00	50.00	
4 сен	з/п	Иванов П.Н.	27.36	472.64	200.00	250.00	100.00	50.00
4 сен	скб	Козлов Г.И.	11.31	472.64	188.69	250.00	100.00	50.00
6 сен	фрп	Петряев Т.Е.	20.38	472.64	188.69	250.00	79.70	50.00
7 сен	прем	Дубов Н.К.	17.10	472.64	188.69	250.00	79.70	32.90
9 сен	з/п	Ильинский Г.Ч.	30.50	442.14	188.69	250.00	79.70	32.90
15 сен	з/п	Аленина Т.В.	11.02	431.12	188.69	250.00	79.70	32.90
19 сен	скб	Теймуров Ш.Г.	6.00	431.12	182.69	250.00	79.70	32.90
23 сен	мат	Александров А.С.	4.43	431.12	182.69	245.57	79.70	32.90
25 сен	мат	Матросов Р.Т.	46.76	431.12	182.69	198.81	79.70	32.90
29 сен	з/п	Миянов А.А.	104.21	326.91	182.69	198.81	79.70	32.90
3 окт	фрп	Загорский Д.А.	20.72	326.91	182.69	198.81	58.98	32.90
4 окт	з/п	Жеребьевский Т.Т.	150.01	176.90	182.69	198.81	58.98	32.90
6 окт	прем	Артемьев О.Н.	31.02	176.90	182.69	198.81	58.98	1.88
9 окт	з/п	Князев Г.П.	57.89	119.01	182.69	198.81	58.98	1.88
9 окт	скб	Яркадьев А.С.	74.82	119.01	107.87	198.81	58.98	1.88
9 окт	мат	Парченков С.С.	170.90	119.01	107.87	27.91	58.98	1.88
11 окт	мат	Арданов А.Г.	20.00	119.01	107.87	7.91	58.98	1.88
13 окт	з/п	Новикова А.А.	100.00	19.01	107.87	7.91	58.98	1.88

Рис. 4.2. Инженерная модель на электронной таблице

Идентификация параметров полиномиальной модели								
Координаты	Экспериментальное значение	Теоретическое значение	Разница	Пар1	Пар2	Пар3	Пар4	Пар5
1	10	9.1371911	0.7444	-1.726	-1.726	-1.726	-1.726	
2	26	25.784792	0.04631	-3.443	-1.722	-0.8688	-0.4304	
3	58	58.424299	0.18	22.91	7.637	2.546	0.8486	
4	112	112.5397	0.2913	69.08	17.27	4.318	1.079	
5	194	193.61498	0.1482	-96.25	-19.25	-3.85	-0.77	
Мера разн.								
Частные производные								
1.18756 -7.94005 1.86069 0.35938 -0.84044								
веса: 0.00002 0.00029 0.01000 0.01000								
шаг: 0.00016 -0.00037 -0.00359 0.00840								
разнос: 5756.62 -6749.08 -754.12 357.66								

На рис. 4.2 показан пример более сложной электронной таблицы, ориентированной на решение инженерной задачи и подбор оптимальных значений параметров. Формулы, находящиеся в этой таблице, выполняют вычисления отклонения между теоретическим и экспериментальным наборами данных и осуществляют пошаговые изменения значений некоторых коэффициентов теоретической модели, постепенно минимизирующие эти отклонения.

Рассматривая функциональные возможности электронных таблиц МАСТЕРа, сгруппируем их следующим образом:

- общее управление, т.е. создание и уничтожение таблицы, движение по ней, выделение ее частей;
- работа со значениями ячеек, т.е. ввод и редактирование значений в ячейках, управление форматами их визуализации, копирование содержимого ячеек внутри одной рамки и между разными рамками;
- работа со структурой и оформлением таблицы, т.е. вставка и уничтожение строк и столбцов, управление разграфкой таблицы, упорядочение строк по возрастанию или убыванию значений;
- использование формул, т.е. способы определения формульных связей между значениями ячеек;
- построение графиков, т.е. получение наглядного графического представления числовых информации, содержащейся в таблице.

4.2. Общее управление электронной таблицей

Создание табличной рамки

Для того чтобы работать в МАСТЕРЕ с электронной таблицей, вы должны прежде всего создать рамку соответствующего типа – табличную. Операция создания табличной рамки полностью ана-

югична созданию любой другой рамки, она реализуется через главное меню командой

{Ф10} Рамка Создать Электронная Таблица

Для табличных рамок действительны все те общие клавиши и команды, которые применимы для любых других рамок. В частности, для того, чтобы войти в табличную рамку, нужно нажать клавишу {ПЛЮС}, а для выхода из нее – {МИНУС}. Через главное меню можно, как обычно, изменить размеры, положение, цвет табличной рамки, уничтожить рамку или сохранить ее на диске.

Указание диапазонов ячеек

Во многих операциях МАСТЕР запрашивает указание диапазона ячеек, над которыми должно производиться действие. Диапазон – это группа ячеек, занимающих прямоугольную область на таблице. При запросе диапазона система переходит в состояние указания: в индикаторе состояния вы видите слово УКАЗАНИЕ, а в поле сообщения – фразу, объясняющую назначение запрашиваемого диапазона, например "укажите, что копировать" или "укажите, где устанавливать формат" и т.п. В этом состоянии вы можете передвигать курсор клавишами-стрелками. При этом будет происходить световое выделение прямоугольной части таблицы (рис. 4.3).

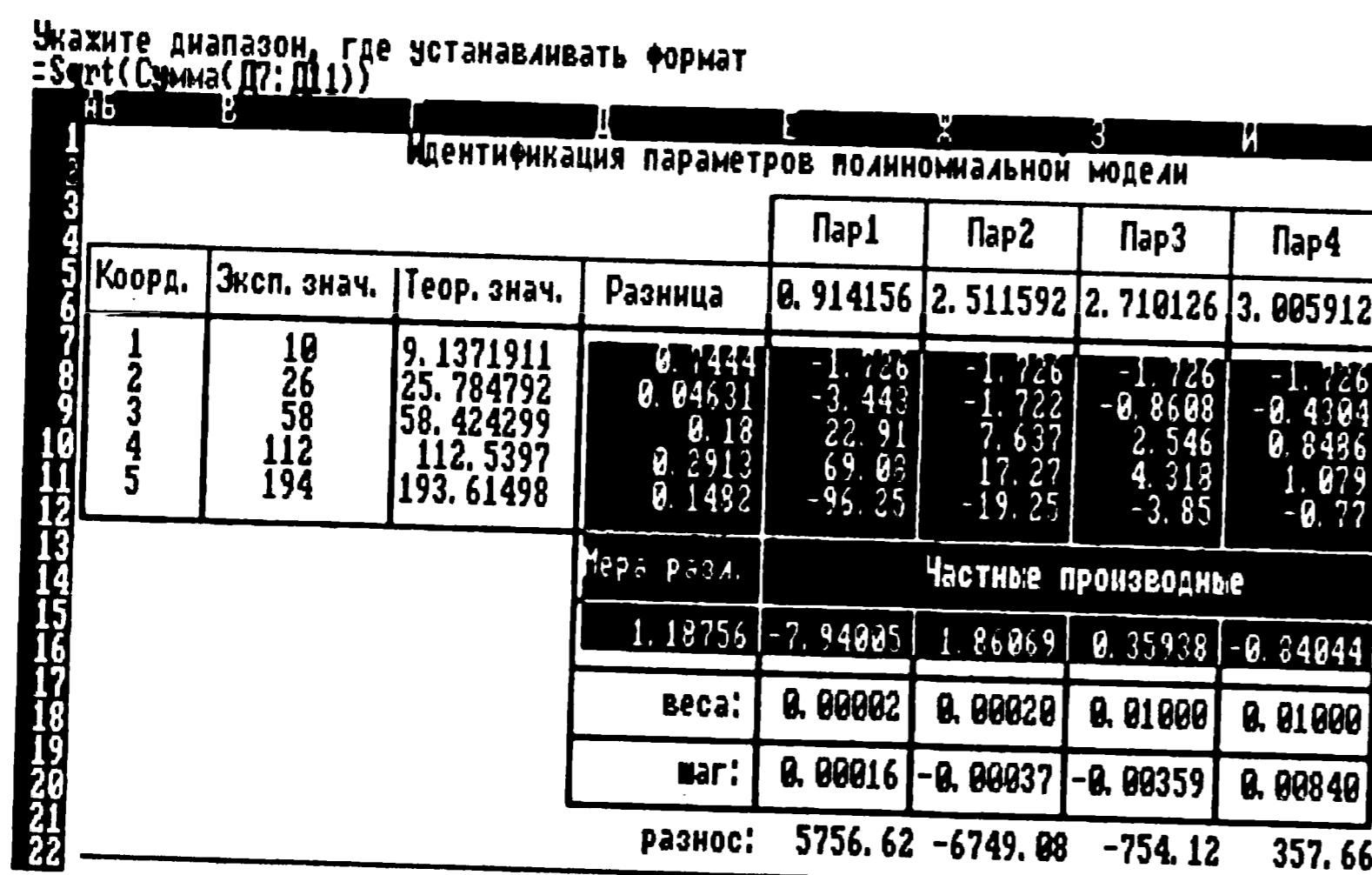


Рис. 4.3. Выделение диапазона в таблице

Может потребоваться, чтобы выделенный диапазон начинался не в той ячейке, где курсор стоял в начале выделения, а в какой-то другой. Тогда нужно "отцепить" диапазон, нажав клавишу {Ф4}. В этот момент подсветка диапазона исчезнет, и подсвеченной остается одна только текущая ячейка. После этого можно подвести курсор к любой другой ячейке и нажать клавишу {Ф3}. Это нажатие опять "зацепит" диапазон, и дальнейшие движения курсора вновь будут выделять на таблице прямоугольную область. Такие зацепки и отцепки можно делать многократно: нажатие клавиши {Ф4} отцепляет, а клавиши {Ф3} зацепляет диапазон. Когда нужный диапазон выделен подсветкой, следует нажать клавишу {ВВОД}.

После входа в табличную рамку автоматически изменяется вид меню во второй строке экрана. Вместо главного меню там появляется **табличное меню**, команды которого перечислены в табл. 4.1.

Таблица 4.1. Операции табличного меню

Операция	Действие
Значения	
Копировать	Копирование диапазонов в диапазоны
Очистить	Уничтожение содержимого ячеек
Сортировать	Упорядочение строк по значению ячеек
Заполнить	Заполнение диапазона числами
Структура	
Формат	Установка формата ячеек
Прозрачность	Установка прозрачного формата ячеек
Ширина	Установка ширины и высоты ячеек
Высота	
Вставить	Вставка строк или столбцов
Убрать	Уничтожение строк или столбцов
Переслать	
Печать	Вывод диапазона таблицы на печать
Экспорт	Вывод диапазона в текстовом формате
Импорт	Загрузка диапазона в текстовом формате
Отдать	Пересылка диапазона в текстовую рамку
Взять	Пересылка из текстовой рамки в диапазон
График	
Режимы	Создание графика в графической рамке
Активна	Включение автоматического пересчета
Пассивна	Выключение автоматического пересчета
Номера	Включение обозначения строк и столбцов
БезНомеров	Выключение обозначения строк и столбцов
Значения	Включение визуализации формул
Скрыто	Выключение визуализации формул
Шапка	Фиксация неподвижной части таблицы
Боковик	

4.3. Ввод и редактирование значений ячеек

Типы значений в ячейках

Ячейки электронных таблиц в интегрированной системе МАСТЕР могут содержать значения любого из следующих типов:

- числа — вещественные числа с точностью до 15 знаков;
- строки — символьные строки длиной до 250 знаков;
- тексты — многострочные тексты неограниченного размера;
- рисунки — графические изображения рисунков и графиков.

Если ячейка содержит только одно значение, то она называется независимой и во время пересчета таблицы не изменяет своего содержимого. Вместо независимого значения ячейка может содержать формулу, позволяющую автоматически вычислять значение ячейки по значениям других ячеек. Формула может вырабатывать значение любого из четырех перечисленных типов. При пересчетах значения таких ячеек автоматически изменяются в соответствии с проделанными вычислениями.

Для текстовых и рисунковых значений обычно бывает недостаточно односторонней высоты ячейки, поэтому для строк, содержащих такие ячейки следует увеличить высоту. Тогда ячейки станут небольшими прямоугольными окнами.

Движение табличного курсора

Движение курсора по таблице выполняется с помощью клавиш-стрелок, перечисленных в табл. 4.2.

Таблица 4.2. Команды передвижения курсора в таблице

Клавиша	Движение курсора
{ВВЕРХ}	на одну строку вверх
{ВНИЗ}	на одну строку вниз
{ВПРАВО}	на один столбец вправо
{ВЛЕВО}	на один столбец влево
{ПРЕД}	на одну страницу вверх
{СЛЕД}	на одну страницу вниз
{УПР-ВПРАВО}	на одну страницу вправо
{УПР-ВЛЕВО}	на одну страницу влево
{ЗАГ-ВВЕРХ}	на верхнюю границу таблицы
{ЗАГ-ВНИЗ}	на нижнюю границу таблицы
{ЗАГ-ВЛЕВО}	на левую границу таблицы
{ЗАГ-ВПРАВО}	на правую границу таблицы
{ВВОД}	к следующему ряду (строке или столбцу)

Ввод значения в ячейку

Для ввода значения в ячейку необходимо прежде всего сделать ее текущей, поставив на нее курсор. Ввод значения начинается с нажатия первой клавиши, представляющей это значение. После нажатия первой же буквенно-цифровой или знаковой (т.е. нефункциональной) клавиши начинается прием значения ячейки. При этом ячейка превратится в маленькое текстовое окно, в котором вы можете пользоваться любыми возможностями текстового процессора МАСТЕРа, а курсор превратится из табличного в текстовый. В поле сообщения (в верхней строке экрана) при этом выводится приглашение: "Введите значение ячейки".

Во время ввода любые значения — и числа, и строки, и формулы — представляются как обычные строки. Вы можете редактировать вводимую строку всеми теми средствами, которые предоставляет текстовый редактор МАСТЕРа: буквенно-цифровые и знаковые клавиши вводят соответствующий символ в редактируемую строку в текущую позицию курсора, стрелки {ВЛЕВО} и {ВПРАВО} переводят курсор в соответствующем направлении, клавиши {ОТМЕНА} и {УДЛ} обеспечивают стирание символов, а клавиши {ЗАГ-ВЛЕВО} и {ЗАГ-ВПРАВО} позволяют быстро попасть к левому или правому краю вводимой строки.

Ввод значения завершается нажатием клавиши {ВВОД}. В этот момент введенное значение анализируется и преобразуется к нужному типу. Введенное значение будет воспринято одним из следующих способов — как число, как время, как дата, как строка или как формула. Это определяется первым символом введенного значения. Если первый символ — цифра или знак минус, то значение будет воспринято как число. Если первый символ — прямая наклонная черта (/) или обратная наклонная черта (\), то значение будет воспринято соответственно как время или дата. Если первый символ знак равенства (=), то значение будет воспринято как формула. Во всех остальных случаях значение будет воспринято как символьная строка. Иногда бывает необходимо, чтобы строковое значение начиналось с одного из перечисленных спецсимволов. В этом случае нужно поставить перед началом строки двойную кавычку ("), которая является явным обозначением строкового значения.

Для облегчения ввода значений один из этих спецсимволов представляется автоматически в соответствии с прежним типом и форматом той ячейки, куда вводится значение. Этот символ оказывается слева от курсора и не мешает вводу. Однако его можно стереть как самый обычный символ и заменить на другой, задавая тем самым новый тип значения ячейки.

Редактирование старого значения ячейки

Часто при работе с таблицей требуется не вводить новое значение, а немного изменить то, которое уже находится в ячейке. Для этой цели имеется клавиша {Ф2}, нажатие которой приводит к такому же редактированию значения, что и описанное в предыдущем разделе, но при этом поле ввода не пусто, а содержит прежнее значение ячейки. Это значение можно редактировать теми же клавишами, что и при вводе нового значения.

Еще одна особенность состоит в том, что при нажатии {Ф2} редактирование производится не в самой ячейке, а в поле значения (во второй строке экрана) – работать здесь удобнее, если строковое значение или формула являются длинными. Клавишу {Ф2} можно использовать как для заполненных, так и для пустых ячеек.

Работа с текстовыми ячейками

Описанным образом производится ввод и редактирование простых значений ячеек, т.е. чисел, строк и формул. Для ввода или редактирования текстовых ячеек нужно предварительно войти в ячейку. Это делается, как и для входа в рамки, нажатием клавиши {ПЛЮС}. Вход возможен только в пустую или в текстовую ячейку, так что, если ячейка уже содержит строку, число или формулу, и вы хотите ввести в нее текст, то сначала нужно очистить ячейку клавишей {УДЛ}.

Войдя в ячейку, вы можете работать в ней как в обычной текстовой рамке, используя все команды текстового процессора МАСТЕРа. Выход из ячейки в таблицу делается, как и из рамок, с помощью клавиши {МИНУС}.

Установка направления ввода данных

При набивке больших массивов данных становится существенным каждое лишнее нажатие клавиши. С целью облегчения этого процесса можно установить направление ввода: одно из четырех возможных направлений, в котором будет автоматически сдвигаться табличный курсор после ввода каждого нового значения.

Для установки направления ввода данных следует воспользоваться клавишей {ЗАГ-ПЛЮС}, после которой следует нажать любую из клавиш-стрелок. Это направление станет направлением ввода. Для отмены направления нужно нажать {ЗАГ-ПЛЮС} дважды. Отмена направления ввода означает, что после ввода значения курсор остается на месте.

После заполнения очередной строки или столбца в указанном направлении ввода желательно иметь возможность одним нажатием переводить курсор к началу следующего ряда. Для этого служит клавиша {ВВОД}. Если задано горизонтальное направление ввода, то при ее нажатии курсор перейдет к началу следующей строки, а при вертикальном направлении ввода – к началу следующего столбца.

Заполнение диапазона ячеек набором чисел

Часто при формировании таблиц требуется заполнять большие се диапазоны наборами чисел (арифметическими прогрессиями или случайными числами). Для этой цели служит команда табличного меню:

{Ф10} Значения Заполнить

При этом сначала запрашивается диапазон, а затем вызывается вспомогательное меню, предлагающее два варианта заполнения указанного диапазона: Прогрессия (арифметической прогрессии) и Случайно (с помощью датчика случайных чисел). При выборе варианта Прогрессия последуют запросы двух чисел – начального значения и шага приращения прогрессии. При выборе варианта Случайно последуют запросы двух чисел, означающих границы интервала, на котором будет генерироваться равномерно распределенный набор чисел.

Очистка диапазона ячеек

Операция очистки стирает информацию в ячейках, но не уничтожает сами эти ячейки. Ячейки, располагающиеся правее и ниже очищаемых ячеек, остаются на своих местах. Операция вызывается из табличного меню командой

{Ф10} Значения Очистить

Сначала при выполнении операции запрашивается диапазон, подлежащий очистке. После этого появляется вспомогательное меню, предлагающее два варианта очистки указанного диапазона: при очистке можно всего лишь убрать из ячеек формулы, оставляя в них значения, а можно очищать ячейки полностью – убирать и формулы, и значения. Эти два варианта предлагаются в виде меню из двух пунктов: Значения и Формулы. Вариант Значения означает полную очистку ячеек указанного диапазона, а вариант Формулы – уничтожение только формул.

Копирование диапазона ячеек

Операция копирования предназначена для переноса содержимого ячеек одного диапазона в другой диапазон. Она выполняется командой

{Ф10} Значения Копировать

При этом запрашиваются два диапазона: сначала задается вопрос "укажите, что копировать", затем "укажите, куда копировать". Первый диапазон будем называть **источником**, второй – **получателем**.

Эта операция переносит содержимое ячеек целиком вместе с формулами, форматами, символами разграфки. При копировании не повторяются только размеры ячеек, которые в диапазоне-получателе остаются неизменными.

Способ исполнения операции копирования зависит от видов диапазонов источника и получателя. Если диапазон источника – одна ячейка, то эта ячейка копируется в каждую ячейку диапазона-получателя (рис. 4.4). Если диапазон-источник – одна строка, то эта строка копируется в каждую строку диапазона-получателя. При этом строки диапазона-получателя определяются по его левому краю: положение правого края диапазона-получателя несущественно (рис. 4.5). Если диапазон-источник – один столбец, то этот столбец копируется в каждый столбец диапазона-получателя. При этом столбцы диапазона-получателя определяются по его верхнему краю. Положение нижнего края диапазона-получателя несущественно (рис. 4.6). Если диапазон-источник включает более одной строки и более одного столбца, то весь прямоугольник копируется в прямоугольник такого же размера. При этом в диапазоне-получателе учитывается лишь положение левого верхнего угла, а правая и нижняя границы диапазона не используются (рис. 4.7).

Ф10-вход в меню ТАБ-разграфка		Ф5-пересчет	Ф3-зацип	Ф4-отцеп	МИНУС-выход
Изменения	Структура	Режимы	Пересылки	График	Общие
ТАБЛИЦА	стр104	стр4	стр64	стр16	стр1024
1	2	3	4	5	6
7	8	9			

Диапазон-источник

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Варианты диапазонов-получателей

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Рис. 4.4. Копирование одноячеичного диапазона

Ф10-вход в меню ТАБ-разграфка		Ф5-пересчет	Ф3-зацип	Ф4-отцеп	МИНУС-выход
Изменения	Структура	Режимы	Пересылки	График	Общие
ТАБЛИЦА	стр104	стр4	стр64	стр16	стр1024
1	2	3	4	5	6
7	8	9			

Диапазон-источник

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Варианты диапазонов-получателей

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Рис. 4.5. Копирование односторочного диапазона

Ф10-вход в меню ТАБ-разграфка		Ф5-пересчет	Ф3-зацип	Ф4-отцеп	МИНУС-выход
Изменения	Структура	Режимы	Пересылки	График	Общие
ТАБЛИЦА	стр104	стр4	стр64	стр16	стр1024
1	2	3	4	5	6
7	8	9			

Диапазон-источник

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Варианты диапазонов-получателей

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Рис. 4.6. Копирование одностолбцовогодиапазона

Ф10-вход в меню ТАБ-разграфка		Ф5-пересчет	Ф3-зацип	Ф4-отцеп	МИНУС-выход
Изменения	Структура	Режимы	Пересылки	График	Общие
ТАБЛИЦА	стр104	стр4	стр64	стр16	стр1024
1	2	3	4	5	6
7	8	9			

Диапазон-источник

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Диапазон-получатель

AAAAAA	BBBBBB	CCCCCC
BBBBBB	CCCCCC	CCCCCC
CCCCCC	CCCCCC	CCCCCC

Рис. 4.7. Копирование прямоугольного диапазона

Передача информации между рамками разных типов

Перенос информации возможен не только в пределах одной табличной рамки, но и из одной рамки в другую. Из одной табличной рамки в другую табличную же рамку информация передается с помощью уже рассмотренной операции копирования диапазонов:

{Ф10} Значения Копировать

Начинать эту операцию надо в той табличной рамке, из которой вы хотите перенести информацию. Именно в этой рамке нужно ответить на первый вопрос операции, указывая диапазон источника. Затем, отвечая на вопрос о диапазоне-получателе, следует использовать обычные клавиши-стрелки и клавиши {МИНУС} и {ПЛЮС}, чтобы перейти в любую другую табличную рамку и указать в ней диапазон-получатель.

После завершения указания каждого диапазона в другой таблице курсор автоматически возвращается в ту табличную рамку, из которой была начата операция копирования.

Другой вариант переноса информации обеспечивает связь табличных и текстовых рамок. Соответствующие операции содержатся во табличном меню:

{Ф10} Переслать Отдать – пересылка из таблицы в текст
{Ф10} Переслать Взять – пересылка из текста в таблицу

В обеих этих операциях требуется указать в табличной рамке диапазон ячеек и в текстовой рамке – строковый фрагмент между которыми производится передача информации. Для перехода из рамки в рамку здесь, как и обычно, используются клавиши {ПЛЮС} и {МИНУС}.

В операции Взять имеется еще один дополнительный вопрос. Он связан с тем, что при превращении текста в таблицу можно либо разбирать строки на отдельные элементы (числа и слова), размещая эти элементы в разные ячейки, либо не разбирать строки и помещать их целиком в левые ячейки диапазона получателя. Этот выбор предоставляется в виде вспомогательного меню, состоящего из двух пунктов Разбирать и НеРазбирать.

4.4. Структура и оформление таблицы

Вставка и уничтожение строк и столбцов

Операции вставки предназначены для раздвижки информации в таблице и образования в ней пустого места для новых строк

или столбцов. Для этой цели в табличном меню имеются две команды, первая из которых предназначена для вставки строк, вторая – столбцов:

{Ф10} Структура Вставить Горизонтали
{Ф10} Структура Вставить Вертикали

При выполнении этих операций МАСТЕР запрашивает диапазон, который указывает местоположение и размер вставки – задается вопрос "какие строки вставлять" или "какие столбцы вставлять". При вставке строк существенны положение и размер диапазона по вертикали, а при вставке столбцов существенны положение и размер диапазона по горизонтали. Положение диапазона по горизонтали или по вертикали задает место вставки, а его размер – число вставляемых строк или столбцов соответственно.

Две подобные команды существуют и для уничтожения строк и столбцов таблицы:

{Ф10} Структура Уничтожить Горизонтали
{Ф10} Структура Уничтожить Вертикали

Упорядочение строк таблицы

Имея таблицу, заполненную определенными значениями, вы можете захотеть переставить ее строки, учитывая либо алфавитное упорядочение каких-то названий, либо возрастание числовых значений в определенном столбце. Для этой цели служит команда

{Ф10} Значения Упорядочить

Перед выполнением этой операции задается запрос того диапазона-столбца, значения которого требуется упорядочить. Выполнение операции требует некоторого заметного времени, поскольку строки переставляются вместе с содержащимися в них формулами и прочими значениями. В этой связи следует отметить, что для сортировки больших массивов информации значительно более подходят структуры базы данных МАСТЕРа, в которых переупорядочение происходит практически мгновенно.

Форматы визуализации значений

Числовые значения, находящиеся в ячейках таблицы, могут визуализироваться в нескольких различных форматах. Наиболее

Простыми способами визуализации являются обычные числовые форматы: с фиксированной и с плавающей десятичной точкой. Кроме таких обычных форматов существуют еще два специальных формата: **формат даты** и **формат времени**.

При визуализации в **формате даты** число рассматривается как количество дней, отсчитываемое от базовой даты 17 сентября 1993 г., которой соответствует число 0. Отсчет ведется как положительными, так и отрицательными числами. В этом виде представимы даты, начиная с 1 января 1904 года (чему соответствует число -32767), кончая 4 июня 2083 г. (чему соответствует число 32767). С точки зрения выполнения арифметических операций число, визуализируемое в **формате даты**, является обычным целым числом. Поэтому над числами-датами можно выполнять различные вычисления, связанные с подсчетом дней.

Аналогичная возможность существует и для визуализации чисел в **формате времени**. В этом случае число, хранящееся в ячейке, интерпретируется как количество минут, прошедших с начала суток, и выводится в одной из форм, принятых для обозначения времени.

Помимо формата, имеющего смысл только для чисел, для значений всех типов можно установить еще способ выравнивания значения в ячейке. Здесь имеются четыре варианта: прижимать значение влево или вправо, располагать его по центру или же не изображать его вовсе (скрытое значение).

Для установки всех этих аспектов визуализации значений в ячейках служит команда табличного меню

{Ф10} Значения Формат

Началом выполнения этой операции является запрос того диапазона, в котором требуется задать формат. Во всех ячейках указанного диапазона формат будет задан одинаково. После указания требуемого диапазона ход операции зависит от того, для какого типа значений устанавливается формат – для чисел или для строк. Этот тип определяется по левой верхней ячейке указанного диапазона (предполагается, что во всех остальных ячейках лежат значения того же типа). Если тип этой ячейки числовой, то появляется вспомогательное меню, перечисляющее разные числовые форматы:

Десятичный Научный Общий Процент Время Дата

Выбрав вид формата, вы зададите способ визуализации всех чисел, появляющихся в указанном вами диапазоне. Примерами изображения чисел в различных форматах являются следующие:

десятичный	- 3.14	2.37981	3
научный	- 3.14e+00	0.314e+01	
общий	- 3.14e+02	2.37981	3.
процент	- 102 %	3.14 %	
время	- 10-30	6 час 30 мин	6 час
дата	- 29.08.1954	29 авг 1954	III кв. 1954

Для первых четырех форматов требуется задать еще число цифр, выводимых после десятичной точки. Это число может быть любым от 0 до 7. Число десятичных цифр, равное нулю, означает для десятичного формата выдачу в виде целого числа без десятичной точки. При выборе форматов Время и Дата возникают вспомогательные меню, предлагающие варианты формы выдачи времени или даты соответственно. Для выдачи времени возможны следующие три варианта формы:

10 час 20 мин
10-20
10 час

а для даты – следующие семь вариантов:

29 авг 1954
29.08.1954
29 авг
29
авг 1954
III кв.
III кв. 1954

Наконец, после выбора числового формата и его атрибутов появляется последнее вспомогательное меню, предлагающее четыре варианта выравнивания значения внутри ячейки.

Для ячеек строкового типа задание формата происходит короче. В этом случае требуется выбрать лишь вариант выравнивания значений: располагать значение влево, вправо, по центру ячейки или сделать его скрытым. Со строковыми и текстовыми значениями связан, однако, дополнительный аспект визуализации. Строки и тексты по ширине могут превосходить ширину ячейки. В этом случае часть значения оказывается невидимой. Однако если соседние справа ячейки являются пустыми, то их можно сделать прозрачными, и тогда область визуализации для значения соответствующим образом расширится. Для того чтобы сделать те или иные ячейки прозрачными, нужно использовать команду

{Ф10} Значения Прозрачность

Установка ширины и высоты ячеек

Столбцы и строки в электронных таблицах МАСТЕРа имеют регулируемую ширину и высоту. Для этой регулировки служит команда

{Ф10} Структура ширина/высота

Операция осуществляется с помощью клавиш-стрелок: клавиши {ВЛЕВО} и {ВПРАВО} уменьшают и увеличивают ширину столбцов, а клавиши {ВВЕРХ} и {ВНИЗ} уменьшают и увеличивают высоту строк.

Разграфка таблицы

Разграфкой таблицы называется совокупность вертикальных и горизонтальных линий, пересекающихся друг с другом таким образом, чтобы изобразить структуру табличного документа. Каждая ячейка может содержать помимо значения еще и некоторый элемент разграфки – сочетание отрезков линий. Для того чтобы ввести в ячейку один из этих элементов, нужно нажать клавишу {ТАБ}. Эта клавиша вызывает специальное меню, предлагающее на выбор любой из элементов разграфки. Пример таблицы, в которой использованы разные элементы разграфки, показан на рис. 4.2.

Фиксация шапки и боковика таблицы

Большинство таблиц должно иметь в своей структуре две фиксированные части – шапку и боковик. В шапке над каждым столбцом написаны названия столбцов, а в боковике даются названия для строк таблицы. При работе с большой таблицей удобно зафиксировать шапку и боковик таким образом, чтобы они всегда оставались в поле зрения рамки, несмотря на ее продергивание в рамке при перемещении курсора. Такую фиксацию можно сделать с помощью команды

{Ф10} Режим Шапка/Боковик

В этой операции от вас потребуется указать курсором угол подвижной части таблицы, т.е. ту ячейку, которая находится непосредственно под шапкой и справа от боковика. Все строки, находящиеся выше, и столбцы слева от указанной ячейки будут зафиксированы.

При зафиксированных шапке и боковике становится невозможным попасть курсором ни на шапку, ни на боковик. Если вам

потребуется изменить в них информацию, то нужно сначала снять фиксацию шапки и боковика, для чего служит та же операция, в которой в качестве подвижного угла нужно указать ячейку А1.

Скрытие номеров строк и столбцов таблицы

В своем исходном состоянии каждая табличная рамка имеет в верхней части буквенные обозначения столбцов, в левой части – числовые обозначения строк таблицы. Эти обозначения полезно видеть при записи формул, для того чтобы задавать буквенно-цифровые адреса ячеек в этих формулах. После того как все формулы записаны и вам нужно просто работать с числовыми и строковыми значениями в ячейках таблицы, вы можете убрать эти обозначения. Для выключения и включения этих обозначений имеются команды:

{Ф10} Режим БезНомеров – выключает обозначения;
{Ф10} Режим Номера – включает обозначения.

4.5. Использование формул

Ввод и редактирование формул

Если вы хотите, чтобы некоторая ячейка содержала не фиксированное, а вычисляемое значение, зависящее от других ячеек, то вам нужно ввести в нее формулу, по которой это значение будет вычисляться.

Формула вводится в ячейку точно так же, как и обычное строковое значение, но первым символом вводимой строки должен быть знак равенства (=). Если ячейка уже содержит формулу и вы хотите отредактировать ее, то нужно, как и для редактирования строковых значений, воспользоваться клавишей {Ф2}. Этой же клавишей удобно пользоваться и при первоначальном вводе формулы, поскольку в этом случае редактирование происходит в поле значения, т.е. во всю ширину экрана, а не в узком пространстве одной ячейки.

Формулы строятся по общепринятым правилам из знаков арифметических операций, обращений к функциям, чисел и имен ячеек. Так, например, для того, чтобы в ячейке А3 вычислялась сумма значений из ячеек А1 и А2, в нее должна быть введена формула

A1+A2

Другим примером является следующая формула, вычисляющая минимальное из тех же двух числовых значений:

МИН(А1,А2)

Некоторые функции принимают в качестве аргументов не отдельные числа, а целые наборы чисел, как, например, функции вычисления минимума, максимума, суммы или среднего арифметического. Аргументом таких функций может быть **указатель диапазона**, задаваемый парой имен ячеек через двоеточие. Так, например, для вычисления суммы чисел, лежащих в ячейках от A1 до A100, можно ввести в ячейку A101 формулу

Сумма(A1:A100)

Копирование формул

Во многих таблицах требуется выполнять вычисления над целыми столбцами или строками. Например, в финансовой ведомости, где столбец Б содержит цены товаров, а столбец В – их количества, требуется, чтобы столбец Г содержал произведение столбцов Б и В. Для реализации этой связи с помощью формул нужно в каждую из ячеек Г1, Г2, Г3, ... столбца Г поместить формулы:

B1*B1
B2*B2
B3*B3

...

Составление такой таблицы было бы очень трудоемким делом, если бы пришлось вводить каждую из этих формул вручную. Вместо этого достаточно ввести формулу только в ячейку Г1. После этого для ее распространения во все ячейки столбца Г можно воспользоваться командой

{Ф10} Значения Копировать

Очень важным свойством операции копирования формул является то, что формулы копируются не буквально, а с перенастройкой содержащихся в них имен ячеек. Так, при копировании формулы B1*B1 из ячейки Г1 в ячейку Г2 она после перенастройки превращается в формулу B2*B2. Эта перенастройка происходит из-за того, что ссылки на ячейки трактуются как **относительные имена ячеек**, определяемые относительно той ячейки, в которой эта формула находится. Так формула B1*B1, находящаяся в ячейке Г1, рассматривается как произведение двух ячеек, находящихся слева от формулы. Поэтому после копирования, сохраняющего относительность имен, в ячейке Г2 произведение двух соседних ячеек будет выглядеть уже как B2*B2.

Такая относительность адресации ячеек в формулах требуется не всегда. Иногда, наоборот, нужно, чтобы формула зависела от фиксированной ячейки независимо от своего положения в таблице. Для этого имя ячейки в формуле следует сделать аб-

солютным. Это достигается постановкой восклицательных знаков перед буквенной и числовыми частями имени ячейки, например:

!B123

Два восклицательных знака требуются потому, что абсолютной или относительной может быть каждая из частей имени ячейки независимо друг от друга:

- B23** – обе части имени относительны;
- !B23** – столбец абсолютный, строка – относительная;
- B!23** – столбец относительный, строка – абсолютная;
- !B!23** – обе части имени абсолютны.

Порядок вычисления формул

В тех случаях, когда некоторые формулы в таблице зависят от ячеек, содержащих другие формулы, результат пересчета этой таблицы будет существенно зависеть от того, в каком порядке вычисляются формулы.

В электронных таблицах МАСТЕРа пересчет формул делается в фиксированном порядке, не зависящем от фактической взаимной связи формул. При пересчете формулы перебираются по строкам: сначала слева направо вычисляются формулы первой строки, затем второй и так далее до нижней строки таблицы. Формулы следует составлять с учетом именно этого порядка вычислений, в противном случае результаты вычислений не будут соответствовать вашим ожиданиям.

Рассмотрим пример. Пусть значение ячейки В1 должно зависеть от нижележащих ячеек:

Сумма(B2:B10)

Если значения ячеек B2:B10 в свою очередь вычисляются по некоторым формулам, то, поместив формулу Сумма(B2:B10) в ячейку В1, мы получим неправильный результат, поскольку эта формула будет вычисляться не после, а перед вычислением необходимых ей значений в ячейках B2:B10. Такой зависимости (снизу вверх по таблице) рекомендуется просто избегать, но если она все же необходима, то придется воспользоваться специальной функцией Присв(), позволяющей присвоить значение в произвольную ячейку таблицы. Поместим обращение к этой функции в какую-то из ячеек, лежащих ниже диапазона B2:B10, скажем, в ячейку В11:

Присв(В1,Сумма(B2:B10))

Эта формула вычислит сумму чисел в ячейках B2:B10, а затем присвоит эту сумму в ячейку В1. Сама ячейка В1 не должна

содержать никакой формулы. Таким образом можно добиться того, чтобы ячейка **B1** содержала правильное значение при стандартном порядке пересчета.

Порядком пересчета в электронных таблицах МАСТЕРа можно управлять, если воспользоваться соответствующими средствами инструментального языка Мастер. Так, имеющаяся в нем функция **Пересчет()** позволяет выполнять пересчет не всей таблицы, а любого ее поддиапазона. При этом можно пересчитывать диапазон не только по строкам, но и по столбцам. Эта функция имеет два аргумента: первый аргумент – диапазон, подлежащий пересчету, а второй аргумент равен 1 – для пересчета по строкам и 0 – для пересчета по столбцам. Обращение (или последовательность обращений) к этой функции можно связать с какой-нибудь функциональной клавишей в таблице, пользуясь механизмом, описанным в 2.2. Тогда при нажатии этой клавиши будет делаться пересчет, порядок которого ориентирован на структуру именно этой таблицы.

Так, например, для рассмотренной зависимости ячейки **B1** от ячеек диапазона **B2:B10** можно определить специальный порядок пересчета формулой

Пересчет(B2:B10);Пересчет(B1)

Определив подобным образом специальный порядок пересчета нужно выключить стандартный механизм пересчета. Для этого имеются следующие операции табличного меню:

- {Ф10} Режим Пассивна – выключение стандартного пересчета,
- {Ф10} Режим Активна – включение стандартного пересчета.

Связь ячеек из разных таблиц

Формулы могут связывать ячейки не обязательно лишь внутри какой-то одной таблицы. Имеется функция **Врамке()**, позволяющая воспользоваться значениями ячеек, находящихся в любой другой таблице. Пусть, например, имеются две таблицы с именами "РабочаяВедомость", и "СводнаяВедомость". Положим, что в ячейке **A10** СводнойВедомости требуется разместить формулу, вычисляющую сумму ячеек диапазона **B1:B100** из РабочейВедомости. Если бы диапазон находился в той же самой таблице, где и сама формула, то она выглядела бы так:

Сумма(B1:B100)

а для того, чтобы взять диапазон из другой таблицы, формула должна быть следующей:

Сумма(Врамке(РабочаяВедомость,!B1:B100))

Обратите внимание, что в этой формуле следует использовать обязательно абсолютную адресацию ячеек, поскольку относительность имеет смысл только в пределах одной таблицы.

Функция **Врамке()** имеет два аргумента. Первый из них задает имя табличной рамки, вторым аргументом является произвольное выражение, содержащее имена каких-то ячеек этой таблицы. Эти имена будут относиться не к текущей таблице, а к той, которая указана первым аргументом функции **Врамке()**.

Описанный прием позволяет взять значения из удаленной рамки. Можно, наоборот, поместить значение в удаленную рамку, если воспользоваться уже упоминавшейся функцией **Присв()**. Например, для того, чтобы при пересчете таблицы **РабочаяВедомость** сумма ее диапазона **B1:B100** отправлялась в ячейку **A10 СводнойВедомости**, в одну из ячеек **РабочейВедомости** (например, в **A11**) следует поместить формулу

Присв(Врамке(СводнаяВедомость,!A!10),Сумма(B1:B100))

Здесь имя **!A!10** относится к таблице **СводнаяВедомость** (из-за обращения к функции **Врамке()**), хотя сама формула и имена **B1** и **B100** относятся к таблице **РабочаяВедомость**.

4.6. Более сложные формулы

Вычисления, задаваемые формулами таблиц, могут выполнять не только над числами и различными другими типами информации. В сущности, в этих формулах доступны все виды информации интегрированной системы МАСТЕР, поскольку языком написания формул является инструментальный язык Мастер, в котором имеется около 200 встроенных функций, применимых и к числам, и к строкам, и к текстам, и к рисункам, и к рамкам, и к базам данных. Программист, знающий язык Мастер, сможет создавать сколь угодно сложные формулы. Здесь же мы рассмотрим лишь те возможности языка, которые имеют прямое отношение именно к таблицам и которые полезно узнать еще до (или вместо) изучения самого языка.

Формулы с условными выражениями

Первая возможность, на которую стоит обратить внимание, связана с использованием функции условного вычисления **Если()**. Таблица, показанная на рис. 4.1, состоит в расчете затрат по нескольким фондам. Столбец Г содержит перечень затрат, а столбцы с Д по И – остатки пяти фондов, уменьшающиеся по мере производства этих затрат. Каждая из затрат от-

носится к тому или иному фонду и вычитается из него. Условное обозначение фонда ставится рядом с суммой затраты в столбце б. Формула, которая вычисляет уменьшение каждого из фондов для каждой затраты, при определении ее в ячейке д4 выглядит следующим образом:

Если(!Б4=д!1, д3-!Г4, д3)

Функция **Если()** имеет три аргумента, первый из которых задается выражением, определяющим логическое условие, по которому выбирается один из двух оставшихся аргументов. Если значение первого аргумента равно 1 (т.е. если логическое условие истинно), то вычисляется второй аргумент д3-!Г4, в противном случае вычисляется третий аргумент д3.

Смысл приведенной формулы состоит в том, что значение фонда уменьшается, если затрата относится к этому фонду, т.е. если в столбце б против данной затраты стоит то же обозначение, что и в верхней ячейке столбца данного фонда. Обратите внимание также на то, что ссылка на ячейки б4 и г4 делается с помощью смешанной адресации: столбец адресуется абсолютно, а строка – относительно. Это сделано для того, чтобы после создания формулы, ее можно было копировать как в нижележащие строки, так и в соседние столбцы. В получившихся после копирования формулах останется лишь подправить букву, обозначающую фонд.

Для того чтобы не приходилось исправлять каждую скопированную формулу, следует сначала скопировать формулу из д4 в соседние справа ячейки Е4 – и4, исправить получившиеся при этом формулы, а уж затем копировать строку формул д4 – и4 в нижележащие строки без каких-либо исправлений.

Поисковые операции в таблице

Вторая возможность более сложного характера связана с использованием функции поиска по таблице – **ТабПоиск()**. Эта функция имеет три аргумента. Первый аргумент задает диапазон, в котором должен делаться поиск. Второй аргумент задает искомое значение (число или строку). Третий аргумент задает размер отступа, который нужно сделать от найденного значения для того, чтобы взять значение, сопоставляемое найденному по таблице.

Пусть в столбце А располагаются названия товаров, а в столбце Б – их цены. Эти два столбца служат справочной таблицей цен. Пусть в ячейке В1 содержится название одного из товаров, а в ячейке Г1 – количество этого товара. Тогда в

ячейку д1 можно поместить следующую формулу, вычисляющую стоимость этого товара:

ТабПоиск(!А11:!А1100, В1, 1)*Г1

Здесь функция **ТабПоиск()** используется для того, чтобы по названию товара, заданного в ячейке В1, найти соответствующую ему цену и умножить ее на количество, лежащее в Г1. Для диапазона А1:А100 использована абсолютная адресация для того, чтобы эту формулу можно было скопировать во все ячейки столбца д. Третий аргумент функции **ТабПоиск()** равен единице, задавая отступ на одну ячейку вправо от найденного в столбце А названия товара.

Операции над строками

В качестве типичного примера использования невычислительных операций может служить функция **Размножить()**. Она имеет два аргумента: строку, которая должна быть размножена, и число, задающее количество повторений. Результатом функции является строка, полученная присоединением строки самой к себе заданное число раз. Воспользоваться этой функцией можно для того, чтобы построить график табличной функции непосредственно в ячейках таблицы, не прибегая к графическим возможностям.

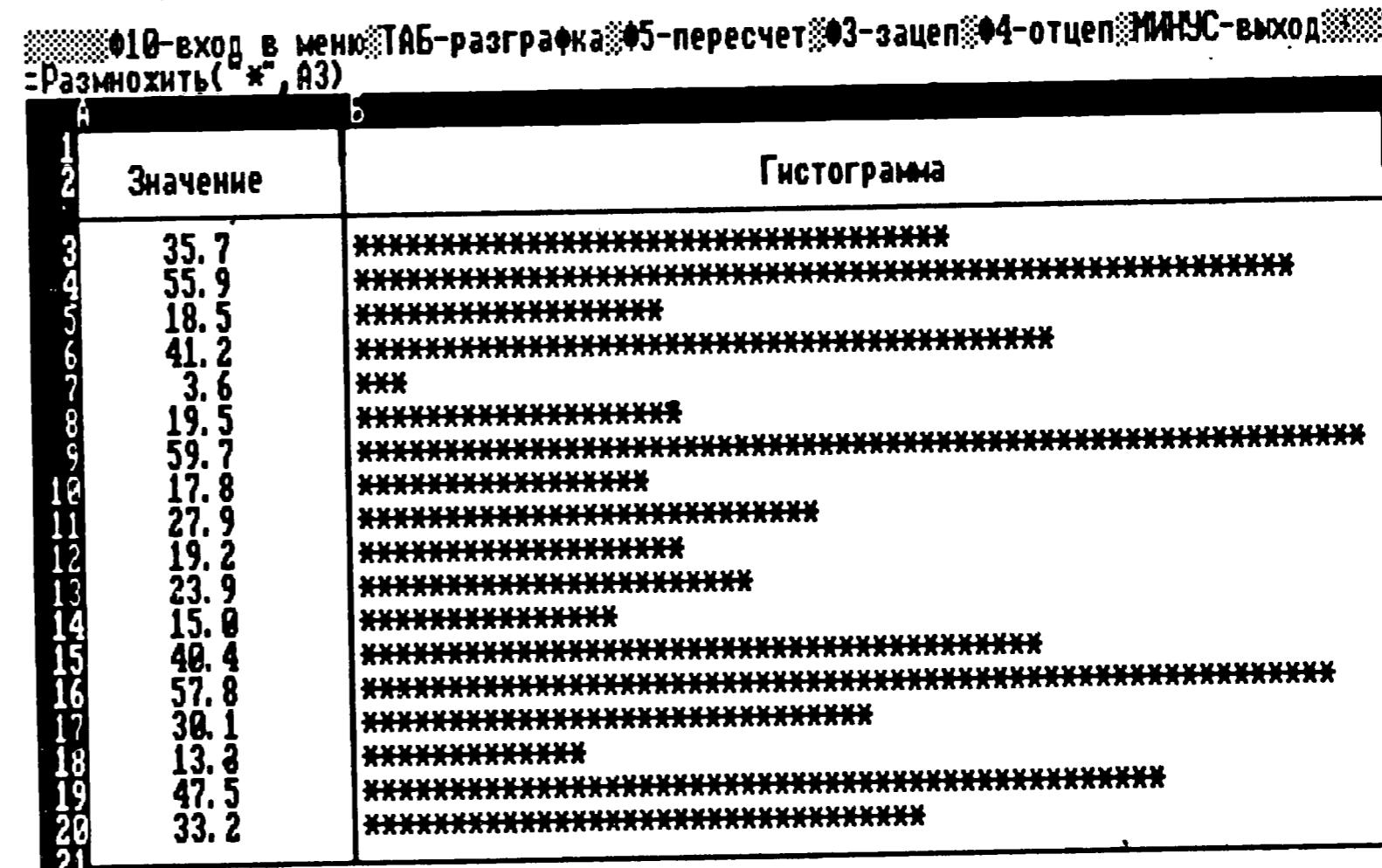


Рис. 4.8. Символьная гистограмма в таблице

Пусть в столбце А находятся числа, которые нужно отобразить в виде гистограммы. Предположим, что все эти числа находятся в пределах от 0 до 50. Установим ширину столбца Б равной 50, зададим в ячейках этого столбца формат, прижимающий значения влево, и введем в ячейку Б1 формулу

Размножить ("*", A1)

Скопировав эту формулу во все ячейки столбца Б, вы увидите в каждой из них строку из звездочек, число которых равно числу, лежащему в ячейке слева от формулы. Тем самым вы получите изображение гистограммы. Эта таблица показана на рис. 4.8.

Вообще говоря, пользуясь более сложными функциями языка Мастер, можно строить в электронных таблицах не просто расчетные финансовые ведомости, а настоящие аналитические модели. Полный набор функций, относящихся к электронным таблицам, описан в гл. 8.

4.7. Графическое отображение табличной информации

Понятия деловой графики

Электронные таблицы, как правило, представляют функциональные зависимости каких-то величин. Эти зависимости более наглядно было бы увидеть в графической форме – в виде гистограммы, круговой диаграммы или линейного графика. Для этой цели существуют специальные графические возможности, называемые средствами деловой графики.

Набор чисел, отображаемых на графике, должен занимать один столбец или одну строку. Будем называть этот диапазон **диапазоном значений графика**. Предположим для упрощения изложения, что этот диапазон – столбец. Используя математическую терминологию, можно сказать, что числа этого диапазона представляют набор значений некоторой функции, заданной таблично для набора значений ее аргумента. На графике требуется отложить по горизонтальной оси значения аргумента и над ними расположить точки графика на высоте, пропорциональной значениям функции, соответствующим этим значениям аргумента. Для формирования графика необходимо, чтобы в таблице над столбцом значений располагалась ячейка с именем этой функции, а сбоку от столбца значений располагался столбец с теми значениями аргумента, для которых вычислены соответствующие значения функции. Это показано на рис. 4.9.

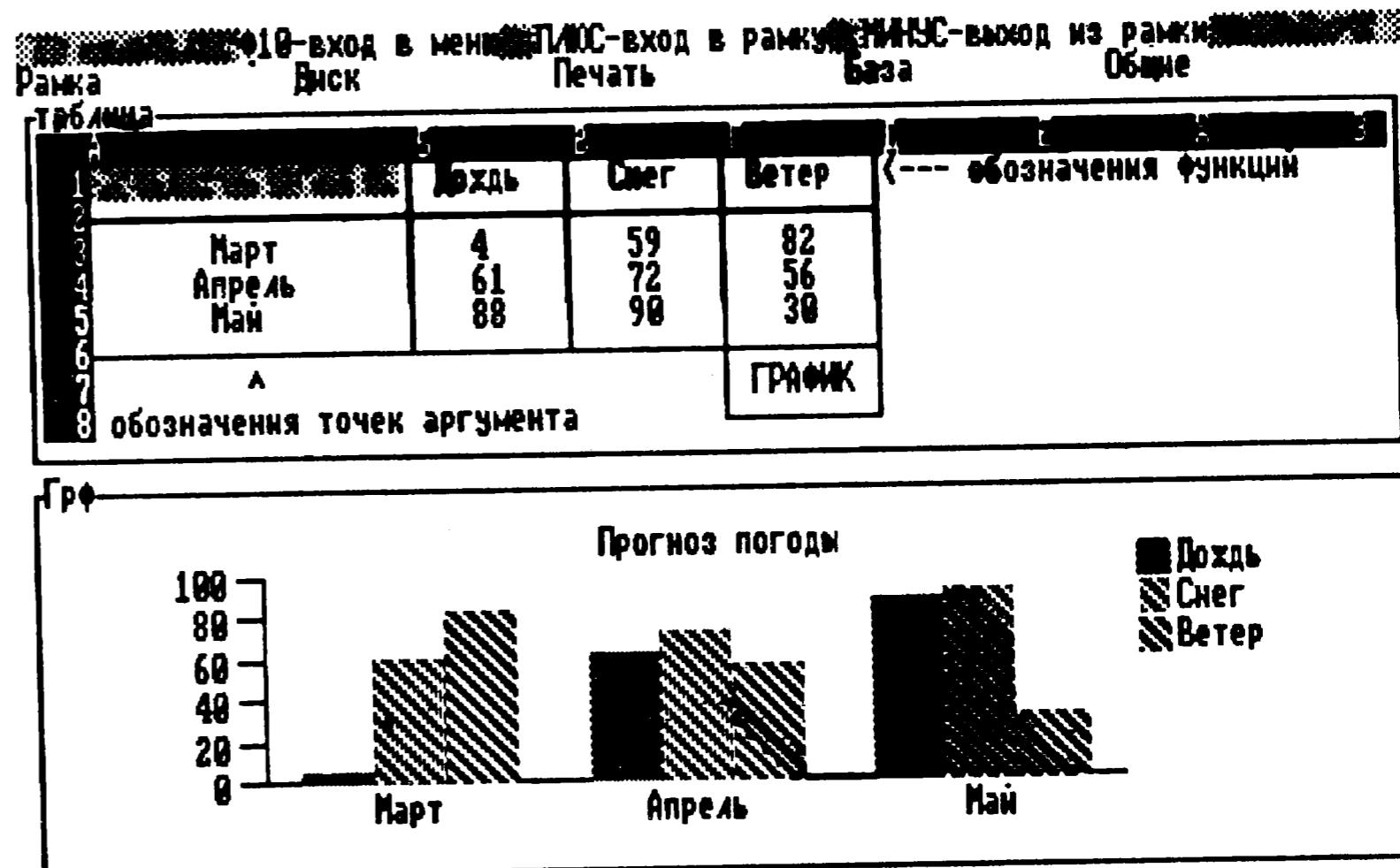


Рис. 4.9. Табличная и графическая рамки

Обозначения для аргумента могут быть представлены как числами, так и строками. Наиболее типичным является строковый вариант. Так, например, значениями аргумента для графика могут быть названия товаров, а сами значения – стоимости этих товаров. Рекомендуется в качестве значений аргументов почти всегда использовать строки, даже и в тех случаях, когда они могли бы быть числами (скажем, когда эти значения – годы). Дело в том, что в случае, когда значения аргументов являются строками, то эти строки просто равномерно откладываются на горизонтальной оси графика в том порядке, в котором они встречаются в диапазоне. Если же они оказываются числами, механизм построения графика усложняется. Числа предварительно упорядочиваются по возрастанию и располагаются на горизонтальной оси не равномерно, а пропорционально своим значениям. Если вам не требуется, чтобы работал такой механизм, то значения аргументов следует вводить в таблицу в строковом виде, для чего перед числом при вводе нужно ставить кавычку ("").

При формировании графика нужно будет указывать не сам диапазон значений графика, а ту ячейку, где находится имя функции, и тот диапазон, где расположены значения аргумента. Имя функции будет использоваться для формирования списка условных обозначений графика, ячейки со значениями аргумента будут использоваться для разметки горизонтальной оси графика. Значения же для графика будут взяты из диапазона, являю-

щегося пересечением тех строк, в которых находится диапазон аргументов, и того столбца, в котором находится имя функции.

В случае, если диапазон значений графика является не столбцом, а строкой, имя функции должно располагаться в ячейке слева или справа от него, а диапазон значений аргумента – сверху или снизу.

Создание графика

Для создания графика используется команда
{Ф10} График

При выполнении этой операции сначала запрашивается имя рамки, в которой нужно разместить график. По умолчанию предлагаются стандартное имя "Грф", которое вы можете произвольным образом изменить. Рамка с указанным вами именем будет создана автоматически и расположится справа от таблицы. В дальнейшем эту рамку можно будет переместить по рабочему полю или изменить в размерах, как и любую другую рамку.

Вторым вопросом запрашивается диапазон, в котором располагаются значения аргументов отображаемой функции. Третьим вопросом следует запрос той ячейки, в которой находится имя отображаемой функции. Подчеркнем еще раз, что сами числовые значения, отображаемые на графике не должны указываться ни в том, ни в другом случае. Диапазон значений графика определяется как пересечение строк и столбцов этих двух указаний.

Вслед за диапазоном запрашивается в виде меню вид графика для отображения указанной функции. Возможными видами графиков являются следующие: гистограмма, линия, точечный график.

Последние три вопроса (имя функции, тип графика и форма точек) затем повторяются несколько раз для того, чтобы вы могли разместить на одном графике сразу несколько функций. Все эти функции должны зависеть от одного и того же набора аргументов, и поэтому вопрос о диапазоне аргументов уже не повторяется.

Формула, порождающая график

Самым последним вопросом, задаваемым перед построением графика, является запрос указания той ячейки, в которую нужно поместить формулу, вычисляющую график. Дело в том, что графики в МАСТЕРе вычисляются с помощью таких же формул, как и обычные арифметические значения. Отличие состоит только в том, что для вычисления графика используется специаль-

ная функция График(), значением которой является не число, как, скажем, у функции Сумма(), а рисунок. Этот рисунок должен быть присвоен в рамку рисункового типа с помощью той же самой функции Присв(), которую мы уже использовали для присваивания числовых значений. Структура формулы, вычисляющей график, является поэтому следующей:

Присв(Грф, График(...))

Здесь График(...) – обращение к функции График(), строящей изображение графика, а Грф – имя той рисунковой рамки, в которую присваивается это изображение.

Пользователю можно и не знать того, какими должны быть аргументы функции График(), поскольку после описанного диалога это обращение создается автоматически. Однако если вы хотите более активно управлять формой графиков, то полезно знать эти аргументы и, редактируя формулу вручную, добиваться более подходящих изображений. Редактировать формулу, построенную автоматически, можно точно так же, как и любую другую формулу, используя для этого клавишу {Ф2}.

Список аргументов функции График имеет довольно сложную структуру:

График(

"титульная надпись графика",
диапазон значений аргумента,
"надпись под осью аргумента",
минимальное значение аргумента,
максимальное значение аргумента,
количество градаций на оси аргумента,

способ комбинирования функций,
"надпись слева от оси функции",
минимальное значение функции,
максимальное значение функции,
количество градаций на оси функции,

ячейка с именем функции,
тип графика для этой функции,
тип точек или закраски гистограммы для этой функции,

...

Минимальные и максимальные значения для аргумента и функции требуются для того, чтобы определить масштаб по соответствующей оси. Если эти аргументы заданы не числами, а строками (например, пустыми), то соответствующее значение

определяется по представленным в таблице числам автоматически.

Количество градаций задает число отметок, изображаемых на соответствующей оси. Здесь тоже, если вместо числа будет задана пустая строка, то число отметок будет выбрано автоматически.

Последняя тройка аргументов повторяется столько раз, сколько функций должно быть одновременно представлено на одном графике. Каждая тройка начинается с ячейки с именем функции, затем идет аргумент, задающий тип графика, а затем – тип точек. Тип графика задается одной из мнемонических констант: ГИСТОГРАММА, ЛИНИЯ, ТОЧКИ. Типы точек задаются числами от 0 до 5.

Важным аргументом является "способ комбинирования функций". Он определяет, каким образом должны располагаться относительно друг друга те несколько функций, которые должны быть одновременно отображены на графике. Для этого аргумента допустимы три значения:

- 1 – столбцы гистограммы располагаются рядом друг с другом;
- 2 – точки располагаются друг над другом, изображая тем самым не только величину каждой функции в отдельности, но и сумму всех функций в каждой точке;
- 3 – все функции должны быть изображены не в виде графиков, а в виде круговых диаграмм.

4.8. Ввод-вывод и распечатка таблиц

Считывание и запись таблиц на диск

Для сохранения табличной информации на диске существуют три возможности:

- запись целой таблицы в виде рамкового файла;
- запись таблицы в текстовом виде с сохранением ее внешнего вида;
- запись таблицы в структурном текстовом виде DIF.

Основной операцией, которая будет требоваться в работе, является запись таблицы в рамковом файле. Эта операция выполняется через главное меню

{Ф10} Диск Записать

и делает то же, что и для рамок любого другого типа, – сохраняет в файле всю информацию, как содержательного, так и оформительского характера, в специальном внутреннем формате, защищенном от несанкционированного доступа через другие

программные системы. Для выполнения этой операции табличная рамка должна быть текущей, т.е. перед записью вам потребуется выйти из нее. Записанную таким образом рамку можно в дальнейшем считать с диска через то же главное меню

{Ф10} Диск Считать

Вторая возможность ввода-вывода таблиц – представление их в стандартном текстовом формате. Соответствующие операции, называемые экспортом и импортом, реализуются двумя способами. Первый способ обеспечивает запись таблицы в текстовом формате, при котором сохраняется ее внешний вид со всеми элементами разграфки, форматом выдачи чисел и строк, шириной и высотой ячеек. Этот способ реализуется так же, как и текстовый экспорт – через операцию распечатки, в которой в качестве получателя информации указывается не принтер, а файл. Эта операция содержится в главном меню

{Ф10} Печать Файл

Перед исполнением этой операции курсор должен быть установлен на выводимую табличную рамку, которая выводится в файл целиком. Такую же операцию можно выполнить и через табличное меню

{Ф10} Переслать Печать Файл

Но в этом случае перед выполнением операции будет сделан запрос диапазона, и выводиться будет не вся таблица, а только указанная вами часть. Здесь задается еще один вспомогательный вопрос в виде меню, состоящего из двух пунктов: Начало и Продолжение. Этот вопрос позволяет вам записать очередную порцию выводимой таблицы либо в новый файл, либо в конец уже существующего. Обычно, при выводе большой таблицы бывает удобно выводить несколько ее частей в один и тот же файл последовательно друг за другом. Тогда при записи первой части нужно выбрать ответ Начало, а для всех остальных частей – Продолжение. Естественно, что при этом нужно указывать для всех частей одно и то же имя файла.

Второй способ записи таблицы в текстовом формате ориентирован на информационный обмен с внешними базами данных в структурном текстовом формате, совместимом с dBASE III и удобном для считывания различными прикладными программами. Этот способ представлен в табличном меню командами:

{Ф10} Переслать Экспорт

{Ф10} Переслать Импорт

Распечатка таблиц

Распечатка таблиц, подобно считыванию и записи, может выполняться как изнутри, так и извне таблицы. Извне таблицы (т.е. когда курсор стоит на табличной рамке) для этого используется общая команда Печать, содержащаяся в главном меню

{Ф10} Печать Принтер

Изнутри таблицы (т.е. когда курсор находится в табличной рамке) операция распечатки доступна через табличное меню и содержится во вспомогательном меню Переслать в виде следующей команды:

{Ф10} Переслать Печать Принтер

По своему результату оба варианта эквивалентны, за исключением того, что в первом случае на распечатку выводится сразу весь диапазон таблицы, содержащейся в табличной рамке, а во втором случае пользователю задается вопрос об указании определенного диапазона, что позволяет "разрезать" большую таблицу на несколько частей и уместить на узком листе бумаги.

Иногда при распечатке протяженных таблиц требуется, чтобы шапка таблицы (несколько верхних строк) повторялась на каждой странице. Это и другие подобные особые требования легко можно учесть, пользуясь языком Мастер. Так, если наша таблица состоит из трех страниц, а ее заголовок включает 6 строк, то можно связать с клавишей, скажем, {Ф6} следующую формулу:

```
{Ф6}
ТабТекст(!A!1:!D!6,"PRN",1);
ТабТекст(!A!7:!D!26,"PRN",1);
Сообщить('Вставьте следующую страницу');
ТабТекст(!A!1:!D!6,"PRN",1);
ТабТекст(!A!27:!D!46,"PRN",1);
Сообщить('Вставьте следующую страницу');
ТабТекст(!A!1:!D!6,"PRN",1);
ТабТекст(!A!47:!D!66,"PRN",1)
```

При нажатии клавиши {Ф6} в этой таблице будет происходить распечатка последовательно трех страниц с шестистрочным заголовком в начале каждой из них и с остановкой перед каждой очередной страницей. Для того чтобы пользоваться такого рода возможностями более активно, требуется изучить язык программирования Мастер, рассматриваемый во второй части книги.

ДИАЛОГОВАЯ ИЛЛЮСТРАТИВНАЯ ГРАФИКА

5.1. Базовые понятия иллюстративной графики

Назначение иллюстративной графики в МАСТЕРе

Многочисленные и разнообразные графические пакеты, созданные за последнее время на персональных компьютерах, обладают впечатляющими возможностями. Графические возможности МАСТЕРа на этом фоне представляются более, чем скромными. Это обусловлено тем, что обработка графической информации сама по себе не входит в число основных его задач, а реализация графики в большем объеме потребовала бы значительных расходов оперативной памяти. Тем не менее, обойтись вообще без графики в современной интегрированной системе практически невозможно. Не говоря уже о деловой графике, которая, безусловно, необходима для наглядного представления табличных данных, требуется также хотя бы минимальные средства иллюстративной графики. Иллюстративной графикой называется набор таких диалоговых возможностей, которые позволяют непосредственно на экране дисплея с помощью манипулятора "мышь" или клавиатуры создавать и редактировать графические изображения. В МАСТЕРЕ такая графика требуется для нескольких целей.

Прежде всего рисунки могут потребоваться просто для иллюстрации документов при желании изобразить что-то в виде схемы, диаграммы, наглядного образа и вставить это изображение в текст выдаваемого на печать документа.

Затем, подобно тому, как деловая графика позволяет отображать числовую информацию в виде графиков функций, точно так же вы можете захотеть отображать ее в каком-то особом виде. Например, если вы занимаетесь проектированием некото-

рой конструкции, и на таблице у вас находятся ее геометрические параметры, то рисунок автоматически мог бы изображать эту конструкцию, причем видимые размеры и количество компонентов определялись бы по формуле числами из таблицы. Такого рода задача является специализированной разновидностью деловой графики, и для ее реализации требуется владение языком Мастер.

Третья потребность заключается в том, чтобы, описывая тот или иной объект (скажем, при занесении его в базу данных), иметь возможность задавать не только количественные и символические характеристики, но и давать хотя бы минимальные сведения графического характера. Например, информация о программном продукте могла бы содержать в графическом виде блок-схему его архитектуры, информация о железобетонной, металлической или любой другой технической конструкции – ее принципиальное изображение. Малая разрешающая способность графики МАСТЕРа, разумеется, не позволит использовать эту графику для изготовления чертежей или карт, но ее вполне достаточно для формирования наглядного образа объекта, помогающего пользователю при поиске и при выборе.

Еще одно возможное приложение связано с тем, чтобы использовать графическое изображение не в качестве хранимого информационного атрибута того или иного описания, а в качестве наглядного средства навигации по информации. Так, в базе данных о сельскохозяйственных угодьях можно предложить пользователю интерфейс, основанный не на названиях (полей, районов, сел), а на указании этих объектов на карте. В другом случае аналогом такой карты может служить изображение структурной схемы организации или конструкции. Во всех этих случаях непосредственное указание позиций на графическом образе может оказаться намного более простым для пользователя, нежели идентификация объектов через названия, координаты, числовые признаки.

Наконец, последнее, для чего может потребоваться графика в МАСТЕРЕ – это просто украшение экрана при создании какого-то сложного интерфейса. Наглядность экрана в современных интерактивных системах играет далеко не последнюю роль.

Таким образом, потребность в иллюстративной графике может возникнуть практически в любом месте информационной среды МАСТЕРа – и в базе данных, и в электронной таблице, и как самостоятельная задача создания отдельных иллюстраций. В данной главе описываются диалоговые возможности небольшого пакета, включенного в виде динамически загружаемого оверлея в стандартную диалоговую оболочку МАСТЕРа. Полезно знать, что все описываемые возможности графического процессора, в отличие от текстового и табличного процессоров не являются

встроенными в ядро МАСТЕРа, а реализованы на инструментальном языке Мастер.

Графическая концепция МАСТЕРа

Графическая концепция МАСТЕРа основывается на том, что изображения рассматриваются наряду со всеми прочими элементами данных как значения, допускающие вычисления, преобразования, копирование, хранение. Такая концептуальная унификация означает не только удобства при программировании, но создает дополнительные возможности обработки информации в интегрированной среде, связанные с редактированием изображений, хранением их в базе данных, в таблицах и текстах и т.п.

Работая с деловой графикой, вы уже видели, что изображение графика порождается обращением к функции График() и присваивается, как обычное число или строка, в рамку рисункового типа. Создание рисунка в диалоге имеет тот же результат: рисунковое значение оказывается присвоенным в рамку. Отличие состоит только в том, каким образом создается это значение. В данном случае оно создается с помощью специального диалогового пакета, который в конце работы с ним присвоит в рамку рисунковое значение.

После этого присваивания рамка становится носителем рисункового значения. Она может быть, как и всякая иная рамка, записана на диск, она может служить полем записи в базе данных, графической частью сложного документа, ее можно изменять в размерах и сдвигать по рабочему полю и т.п.

Объектная структура рисунков

Для того, чтобы обрабатывать какую бы то ни было информацию, требуется выделять в ней те или иные элементы, чтобы они могли служить объектами, к которым применяются обрабатывающие операции. В текстах такими объектами являются символы и строки, в таблицах – ячейки и диапазоны. В процессоре иллюстративной графики такими объектами являются некоторые части изображения, объединяемые не пространственной близостью, а самим процессом их построения. Рисунок строится из некоторых графических элементов, таких, как отрезки прямых линий, дуги и окружности, заштрихованные области, текстовые надписи. Формируя рисунок, вы вносите в него один за другим такие графические элементы. По своему усмотрению вы можете объединять наборы графических элементов в группы, каждая из которых, задавая собой более или менее сложную часть рисунка, представляет **графический объект**.

Вы можете весь рисунок представить как единый графический объект, а можете разбить его же на несколько мелких объектов. Окончательный внешний вид рисунка от этого разбиения никак не изменится, но полезность этого разбиения проявится в самом процессе диалогового создания рисунка. Дело в том, что данный пакет не позволяет изменить ни один из уже нарисованных графических элементов рисунка. Единственное доступное редактирующее действие – это уничтожение целиком некоторого объекта. Поэтому если вы будете использовать объекты из большого числа графических элементов, то при желании внести малейшее изменение придется перерисовывать заново слишком большую часть рисунка.

Во время рисования объекты никак не отделены друг от друга. На плоскости рисунка линии разных объектов могут произвольным образом пересекаться друг с другом. Идентифицировать объекты можно только в специальной операции, когда объекты поочередно начинают мерцать.

Заготовки графических объектов

Часто в рисунке требуется повторять объекты какого-то одного и того же вида, изменения только их положение и, быть может, размеры. Для этой цели могут быть созданы **заготовки графических объектов**. Каждая заготовка снабжается именем и хранится в списке заготовок в памяти графического процессора. В любой момент при создании рисунка вы можете обратиться к этому списку и, выбрав из него нужную заготовку, превратить ее в конкретный графический объект в определенном месте и с определенными размерами.

Этапы создания рисунка

Диалоговое создание рисунка должно происходить внутри рамки рисункового типа. Для этого прежде всего необходимо ее создать, а затем войти в нее. После создания рисунковая рамка является пустой, т.е. не содержит никакого рисунка. Диалоговое построение рисунков возможно только в пустых рисунковых рамках.

Оказавшись внутри этой рамки, вы можете начать рисовать курсором один за другим графические элементы (линии, дуги, символы, заштриховки). Создание первого же графического элемента переведет систему в **состояние рисования объекта**, в котором она будет находиться до того момента, пока вы не завершите объект, выполнив через графическое меню

соответствующую операцию. После этой операции вы возвратите систему в **свободное состояние**. Таким образом, создавая один объект за другим, вы будете находиться то в состоянии рисования объекта, то в свободном состоянии.

В каждом из этих двух состояний вам будут доступны два разных **графические меню**. В свободном состоянии в этом меню предлагаются операции над объектами и над заготовками, а в состоянии рисования объекта – операции над графическими элементами.

Все то время, пока вы продолжаете создавать графические объекты, находясь то в свободном, то в объектном состояниях, рисунок остается пока еще неенным. Все объекты и заготовки хранятся только в памяти графического пакета. Сама рисунковая рамка по-прежнему остается пустой, т.е. в нее не присвоено никакого значения. Тем не менее изображения всех созданных вами объектов вы видите внутри этой рамки, как если бы рисунок уже был создан и присвоен в нее. Окончательное формирование рисунка предполагает объединение всех накопленных графических объектов в одно рисунковое значение и присваивание этого значения в рамку. После такого формирования рамка перестает быть пустой, и редактирование изображения в ней становится невозможным. Эта заключительная операция формирования рисунка делается при выходе из рисунковой рамки.

Если вы не намерены пока еще фиксировать рисунок, но вам требуется выйти из рамки или даже закончить сеанс работы с МАСТЕРом, то следует выполнить операцию сохранения набора созданных объектов. Эти объекты будут записаны в отдельный рамковый файл, которым можно будет воспользоваться в дальнейшем. Точно так же можно запоминать и списки заготовок, создавая из них библиотеки стандартных изображений.

5.2. Диалоговые возможности в состоянии рисования объекта

Переход в состояние рисования

Подведя курсор к тому месту, с которого вы хотите начать рисование, нужно нажать клавишу {ВВОД}. Эта клавиша является командой фиксации точки на рисунке, и ее нажатие переведет систему в состояние рисования объекта. При этом изменится вид меню, высвеченного в верхней части экрана. Это меню будет содержать операции, которые позволяют создавать и настраивать графические элементы, являющиеся составными частями графического объекта.

Движение курсора

После входа в рисунковую рамку в ней высвечивается опорная сетка и курсор, имеющий форму крестика, размещается в центре рамки (рис. 5.1).

Курсор перемещается по рисунковой рамке, как обычно, с помощью клавиш-стрелок: {ВВЕРХ}, {ВНИЗ}, {ВПРАВО}, {ВЛЕВО}. При каждом нажатии любой из этих клавиш курсор передвигается в соответствующем направлении на один шаг, размер которого совпадает с размером самого курсора.

Размер шага курсора можно изменять, для чего имеются клавиши {ПРЕД} и {СЛЕД}. Клавиша {ПРЕД} увеличивает вдвое размер курсора, а {СЛЕД} – вдвое уменьшает.

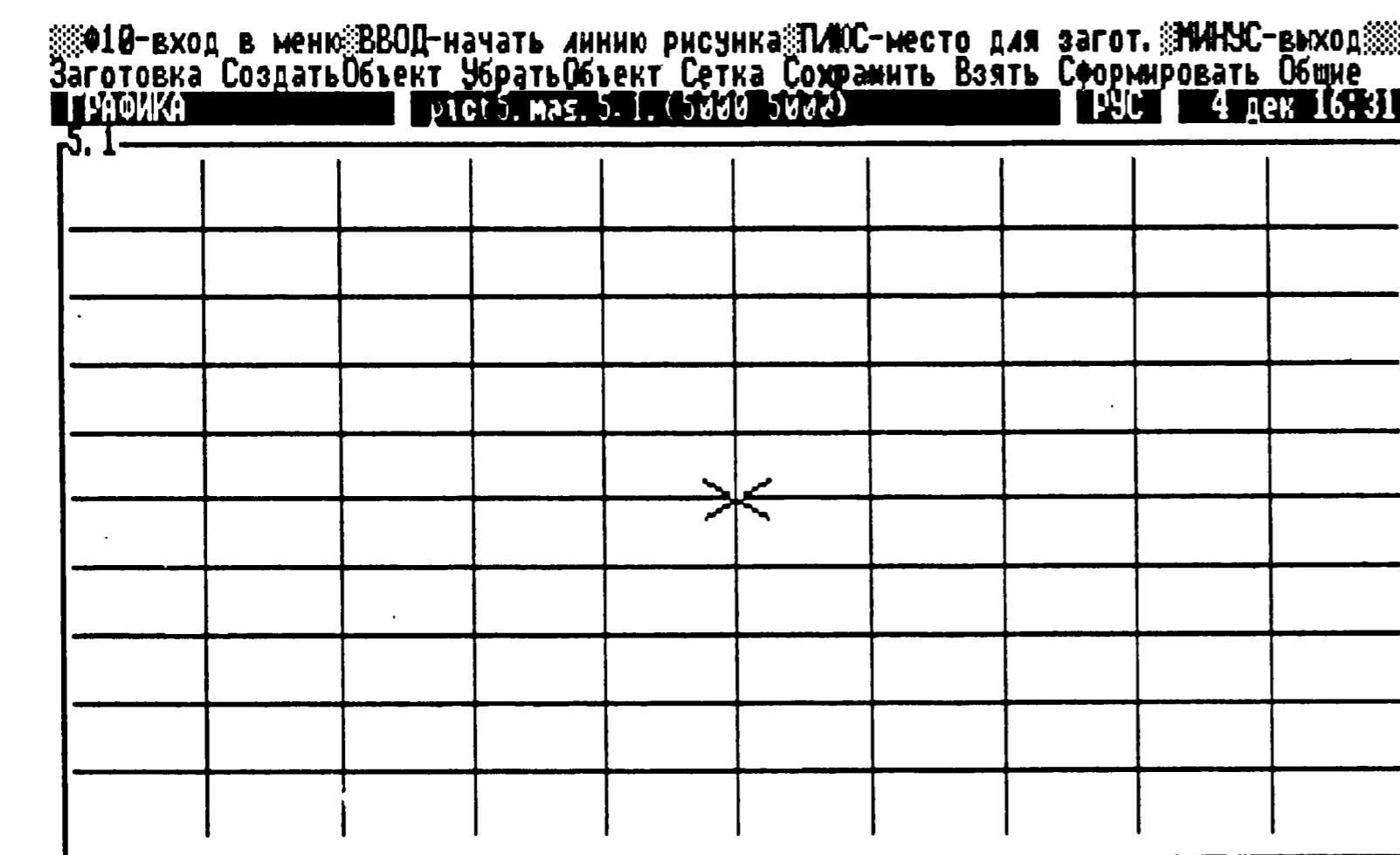


Рис. 5.1. Начальное состояние при рисовании

Рисование ломаных линий

В состоянии рисования имеются два подсостояния: курсор зацеплен линией за некоторую точку или отцеплен. Когда курсор зацеплен за точку, от него к этой точке тянется линия, которую можно зафиксировать, нажав клавишу {ВВОД}. При этом в создаваемый объект вставляется очередной графический элемент – отрезок линии, а точкой зацепа становится та точка, в которой при этом находился курсор.

Таким образом, пользуясь клавишей {ВВОД} и стрелками, легко рисовать ломаные линии. Достаточно последовательно пе-

ремещать курсор по ломаной и нажимать в каждой точке излома линии клавишу {ВВОД} (рис. 5.2).

{Ф10}-вход в меню, {ВВОД}-начать линию рис., {ПЛЮС}-место для загот. {МИНУС}-выход
Заготовка Создать Объект Убрать Объект Сетка Сохранить Взять Сформировать Общие
ГРАФИКА рис5. ма5. (5 000 5000) РУС 4 дек 16:31

5.2

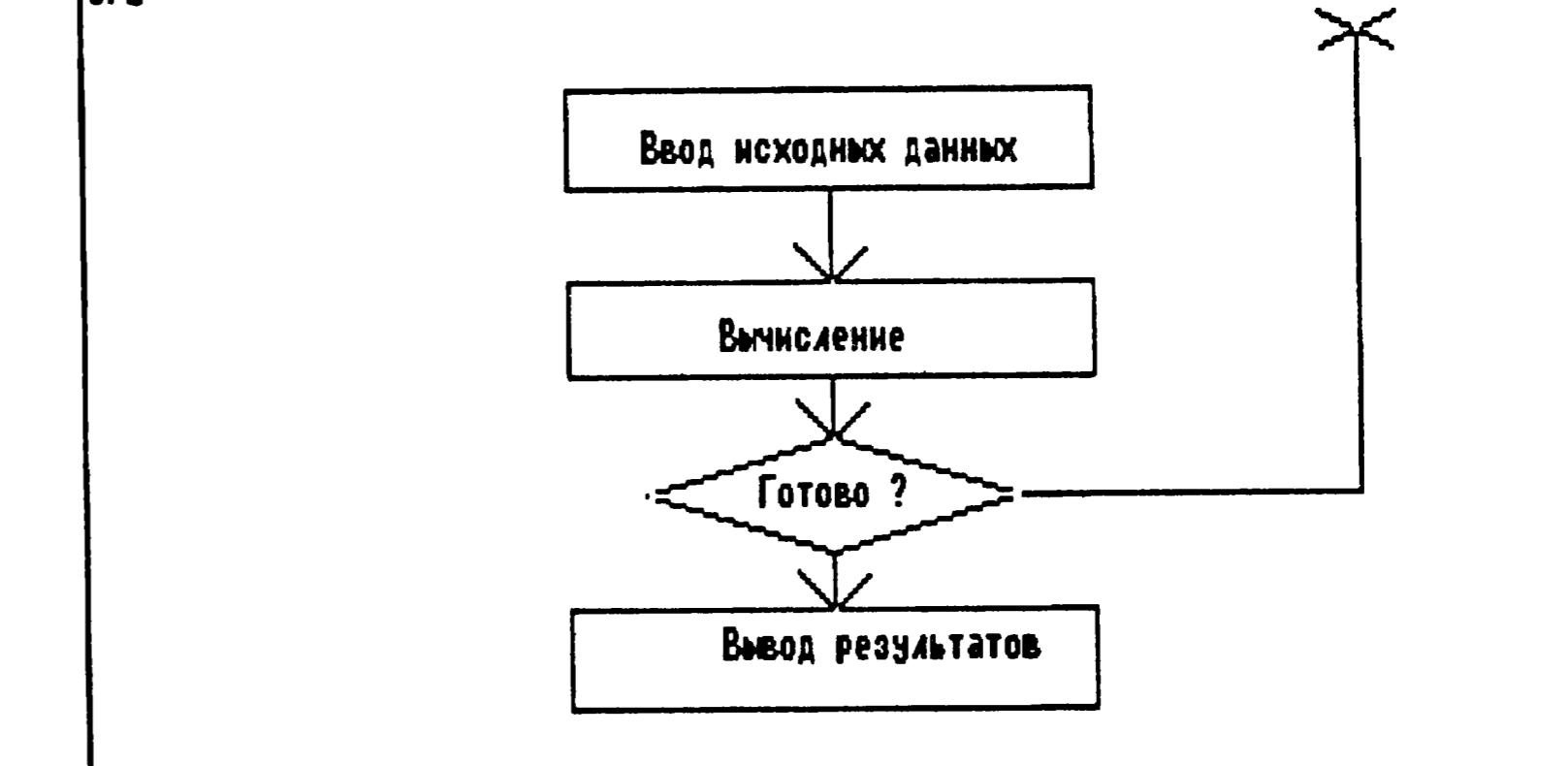


Рис. 5.2. Рисование ломаной линии

Отцепление и зацепление курсора

После того, как ломаная линия завершена, потребуется отцепить курсор, чтобы переместить его в начало другой линии. Для отцепления курсора служит клавиша {ОТМЕНА}. Для того, чтобы начать в другом месте новую ломаную линию, нужно подвести курсор в отцепленном состоянии к некоторой точке и нажать клавишу {ВВОД}. В результате курсор зацепится за эту точку, в которой было сделано это нажатие.

Отцепление курсора не следует путать с переходом в свободное состояние. Единожды войдя в состояние рисования при первом нажатии {ВВОД}, система остается в нем все время до явного завершения рисования данного объекта через графическое меню. В отцепленном состоянии курсор двигается в рамке так же свободно, как и в свободном состоянии, но начатый объект считается незавершенным.

Включение и выключение заштриховки

Если вы хотите, чтобы область, ограниченная ломаной линией, была заштрихована, то в самом начале рисования этой ломаной линии следует нажатием клавиши {Ф3} включить режим

заштриховки. Та вертикаль, в которой находился курсор в момент нажатия этой клавиши становится **базовой линией заштриховки**. В этом режиме при фиксации каждого отрезка ломаной рисуется не просто линия, но еще и закрашивается вся область, находящаяся между проведенной линией и базовой линией заштриховки (рис. 5.3). Для выключения режима заштриховки служит клавиша {Ф4}.

{Ф10}-вход в меню, {ВВОД}-начать линию рис., {ПЛЮС}-место для загот. {МИНУС}-выход
Заготовка Создать Объект Убрать Объект Сетка Сохранить Взять Сформировать Общие
МАКСИМ Рис. 5.3. mas. (5.3) РУС 4 дек 17:25
5.3

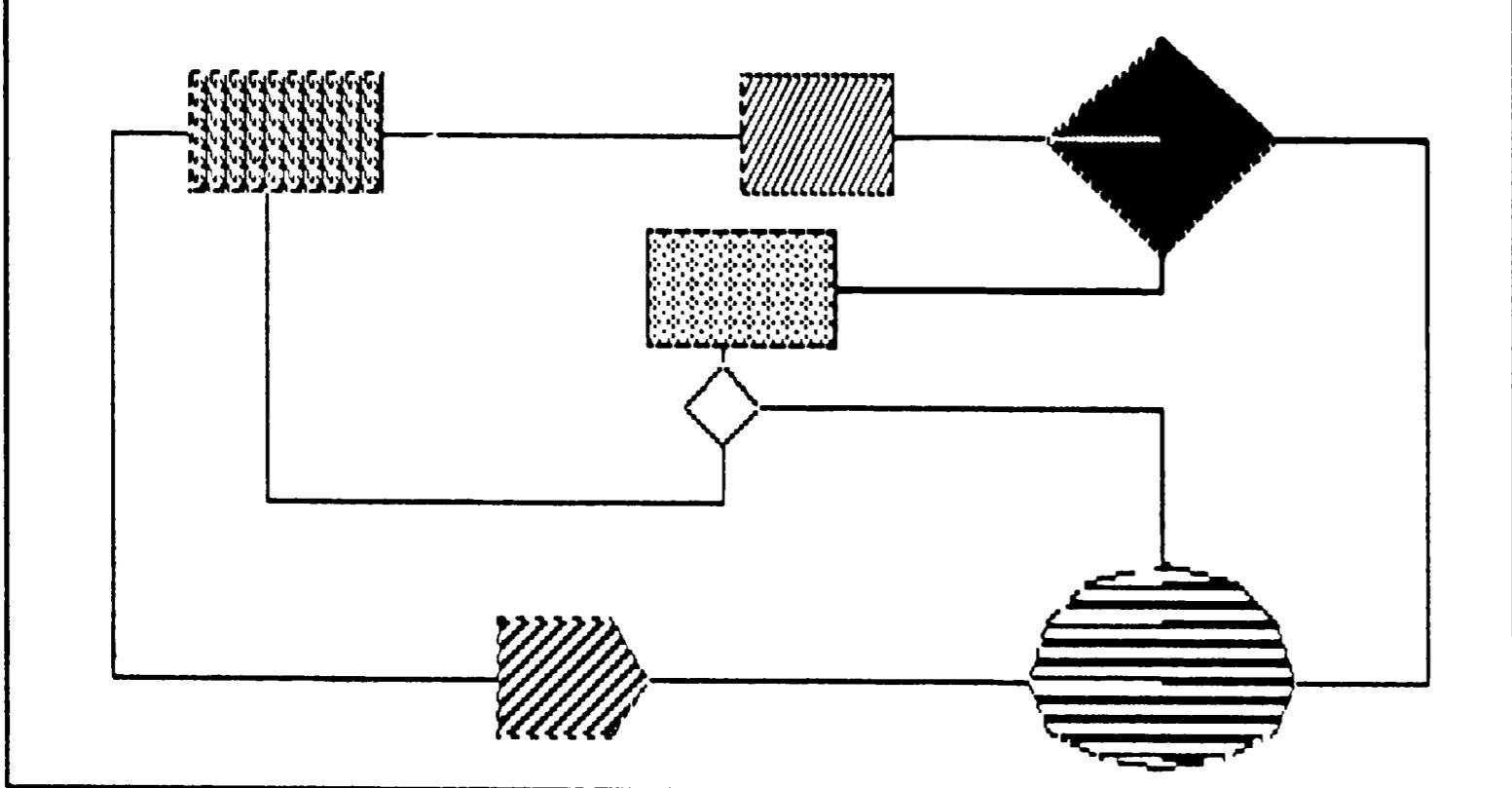


Рис. 5.3. Рисование ломаной линии с заштриховкой

Выбор маски заштриховки

Для заштриховки области используется некоторая **маска заштриховки**, которая представляет собой матрицу белых и черных точек, состоящую из 4 строк и 8 столбцов. Матрица заштриховки первоначально содержит только черные точки. Вы можете переопределить эту матрицу. Для этого служит команда

{Ф10} Штриховка

Выполнение этой команды требует задания четырех строк матрицы, вводящихся символьными строками, в которых черные точки обозначаются звездочками "*", а белые – точками ".".

Нанесение надписей на рисунок

Для того чтобы написать какие-то символы внутри рисунка, нужно вызвать то же графическое меню нажатием {УПР-Ф10} и

116

выбрать в нем пункт Текст. После этого нужно ввести желаемый текст, и он появится на рисунке, начиная с той позиции, где находится курсор. Более точно, верхний левый угол первого символа будет установлен в текущую точку, а остальные символы будут установлены правее.

При введении в рисунок текста следует иметь в виду, что в отличие от всех остальных элементов изображения символы, составляющие текст, являются немасштабируемыми. Поэтому при изменении размеров рисунковой рамки сохраняются только относительные начальные позиции текстов, а сами они не будут изменяться в размерах.

Завершение рисования графического объекта

После того как очередной графический объект полностью нарисован, его требуется завершить. Для этого нужно выполнить команду

{Ф10} Завершить

При выполнении этой операции система перейдет в свободное состояние. Вы можете создавать ваш рисунок как единый графический объект или, наоборот, разбить его на множество мелких графических объектов – в итоге вы получите одно и то же графическое изображение. На размер одного графического объекта наложено ограничение – в нем может быть не более 100 отрезков линий. Для того чтобы не приходилось самому контролировать это количество, система начинает выдавать звуковые сигналы при приближении к ограничивающему числу. Рекомендуется завершить объект, не доходя до ограничения.

5.3. Операции над объектами в свободном состоянии

Создание заготовок

Часто бывает так, что изображение составляется из нескольких графических объектов одинаковой формы, отличающихся только местоположением и, возможно, размерами. Для создания таких объектов предусмотрена возможность определения графической заготовки, из которой можно многократно создавать однотипные объекты. Заготовку нужно начинать строить из свободного состояния пакета. В графическом меню свободного состояния имеется пункт Заготовка, выполнение которого начинает создание графической заготовки.

Само по себе создание заготовки совершается в точности теми же средствами, что и создание отдельных графических

объектов. Точно так же движется курсор под действием клавиш-стрелок, точно так же фиксируются отрезки ломаных нажатиями клавиши {ВВОД}, точно так же курсор может отцепляться от линии нажатием клавиши {ОТМЕНА} и зацепляться нажатием {ВВОД}, точно так же включается и выключается режим заштриховки клавишами {Ф3} и {Ф4}.

Отличие проявляется в операции завершения рисования. Если после завершения рисования обычного объекта этот объект фиксируется в рисунке, то заготовка после завершения исчезает с поля зрения. Она запоминается внутри пакета в специальном масштабированном состоянии, позволяющем вставлять ее в дальнейшем в произвольные места изображения и изменять размеры. Каждой заготовке дается определенное имя, по которому ее можно будет использовать впоследствии.

{Ф10}-вход в меню, {ВВОД}-начать линию рис., {ПЛЮС}-место для загот. {МИНУС}-выход
Заготовка СоздатьОбъект УбратьОбъект Сетка Сохранить Взять Сформировать Общие
[MINUS] [PLUS] Рис. мас. (5.4) 1980 4 дек 17:25

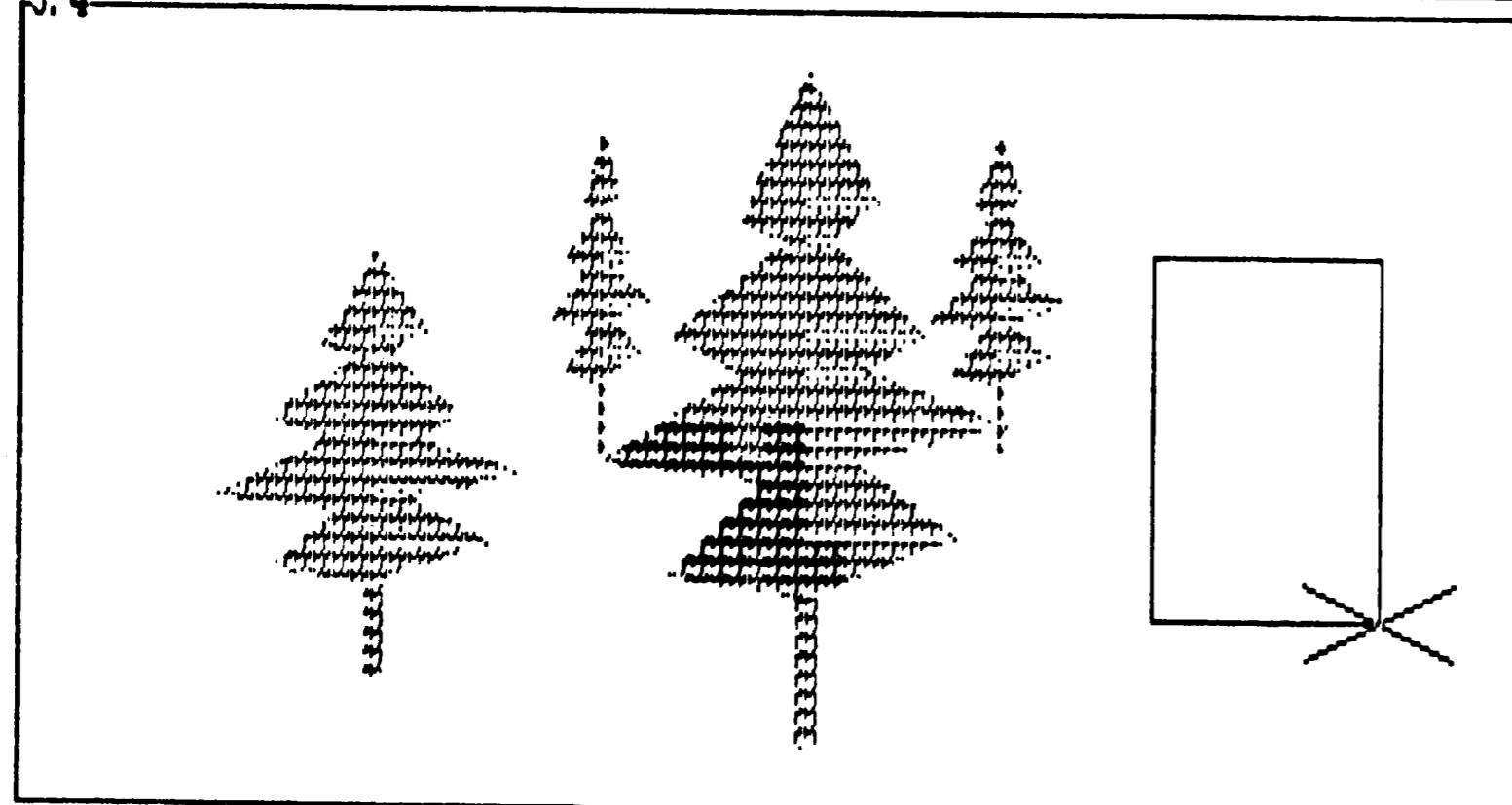


Рис. 5.4. Выделение области для вставки заготовки

Создание объектов по заготовкам

Если вы создали несколько заготовок, то можно воспользоваться любой из них для того, чтобы создать объект. Предварительно нужно выделить прямоугольную область, которая очертит то место, куда должен встать объект, создаваемый из заготовки. Для выделения прямоугольной области нужно нажать клавишу {ПЛЮС} – это зафиксирует один угол области, а затем движениями курсора переместить диагонально противоположный угол в нужную точку. Вы будете видеть область, заключенную

в прямоугольник (рис. 5.4). В таком состоянии нужно вызвать меню и выполнить в нем пункт СозданиеОбъекта. Вам будет задан вопрос об имени заготовки, по которой нужно создавать объект, а затем изображение заготовки появится внутри выделенного вами прямоугольника.

Это еще не конец операции. Объект еще не создан. Вам показаны только его начальные размер и положение. Можно изменить их нажатиями клавиш-стрелок и клавиш {ВСТ} и {УДЛ}, действующих точно так же, как они действуют при установке размеров рамки. Клавиша {ВСТ} переводит в режим установки размера объекта, и тогда стрелки изменяют размер; клавиша {УДЛ} переводит в режим установки положения, и тогда клавиши-стрелкидвигают весь объект целиком. Завершение всей операции выполняется нажатием клавиши {ВВОД}. Только теперь созданный объект фиксируется и располагается в памяти системы в ряду тех объектов, которые были нарисованы или созданы из заготовок ранее.

Просмотр и уничтожение объектов

На изображении, которое вы видите во время рисования внутри рамки, отдельные объекты не выделены никакими видимыми признаками. Они могут произвольным образом пересекаться друг с другом. Для того чтобы увидеть, из каких объектов состоит в данный момент изображение, можно воспользоваться командой

{Ф10} УничтожитьОбъект

Выполнение этой команды приводит к тому, что начинает мерцать первый объект, имеющийся в изображении. При нажатии клавиши {ВВОД} мерцать начинает следующий объект. Таким образом вы можете пройти по всем объектам. После последнего объекта мерцать вновь начинает первый объект. Во время просмотра любой из объектов может быть уничтожен. Для этого достаточно нажать клавишу {УДЛ}. Для возврата из состояния просмотра объектов в свободное состояния без совершения каких-либо действий служит клавиша {ОТКАЗ}.

Сохранение наборов объектов и заготовок

Для того чтобы иметь возможность продолжить работу с созданными объектами в следующих сеансах, необходимо сохранить набор объектов. Для этого служит команда

{Ф10} Сохранить Объекты

РАБОТА**С БАЗАМИ ДАННЫХ**

Набор объектов сохраняется в рамковом файле, имя которого запрашивается у вас при выполнении этой операции.

Аналогичным образом можно сохранить и набор заготовок для чего служит команда

{Ф10} Сохранить Заготовки

Впоследствии можно будет вновь загрузить наборы заготовок или объектов командами

{Ф10} Загрузить Объекты

{Ф10} Загрузить Заготовки

Сохраненные наборы заготовок могут представлять самостоятельный интерес, как графические библиотеки. Над этими библиотеками будет необходимо выполнять некоторые операции, такие, как объединение наборов заготовок, исключение лишних заготовок и т.п. Специальных команд для выполнения этих операций нет. Нужно просто загрузить на рабочее поле МАСТЕРа файл, в котором был сохранен набор заготовок. Набор заготовок представляет собой составную рамку, внутри которой хранятся функциональные рамки, представляющие заготовки. Эти рамки можно переносить из одного набора в другой, уничтожать, выполнять любые другие подобные операции с помощью обычных команд рамкового меню МАСТЕРа.

Завершение рисунка

Завершение рисунка, т.е. превращение набора заготовок в рисунковое значение, присвоенное в рамку, осуществляется момент выхода из рисунковой рамки. Для выхода из рамки используется, как обычно, клавиша {МИНУС}. При нажатии этой клавиши система задает вопрос "завершить формирование рисунка", и если ответить на него утвердительно, то рисунок будет сформирован. При отрицательном ответе курсор останется в рисунковой рамке.

До тех пор пока информация имеет относительно небольшой объем, ее можно обрабатывать в текстовых, табличных, графических и составных рамках. Однако любые рамки всегда располагаются только в оперативной памяти, и поэтому объем хранимой в них информации ограничен величиной около 300 Кбайт (при наличии в компьютере 640 Кбайт оперативной памяти).

В такие объемы свободно вмещаются отдельные документы, большие инженерные модели, тексты статей. Но для практической информации о деятельности учреждения, о товарах и поставках, для полных сведений о сотрудниках и т.п. этого объема недостаточно. Такая информация требует применения внешних устройств большой емкости. Но главная проблема для информации большого объема – это не столько емкость памяти, сколько потребность в специальных логических структурах ее хранения, позволяющих наглядно представлять пользователю содержание информации, быстро выбирать нужные части, корректно извлекать и модифицировать их. Если для текста в несколько десятков строк эти проблемы совершенно несущественны, то для сотен тысяч записей они приобретают ключевое значение.

Для хранения и обработки больших объемов информации в МАСТЕРЕ имеются специальные средства, называемые **средствами управления базами данных** (СУБД). С помощью этих средств вы можете создать на диске хранилище большой емкости, своей структурой ориентированное на вашу задачу и называемое **базой данных**. Это хранилище будет устроено таким образом, чтобы представленную в нем информацию можно было рассматривать с разных точек зрения, в разном порядке, чтобы можно было легко и быстро отыскивать нужные элементы информации.

Для базы данных в отличие от текстов и таблиц нет какого-то определенного типа рамок. Для визуализации и диалоговой обработки информации из базы используются комбинации рамок всех типов – и текстовые, и табличные, и рисковые, и составные. Чем более сложная структура информации содержится в базе данных, тем более сложная система рамок используется для ее визуализации.

Не следует думать, что рамки как таковые хранятся в базе, (как они действительно хранятся в рамковых файлах). Структура информации в базе строится на совсем иных элементах: записях, полях, множествах и подмножествах записей.

6.1. Основные структурные понятия базы данных

Записи

База данных МАСТЕРа образуется из **записей**. Каждая запись представляет собой описание какого-то объекта и состоит из нескольких элементов данных, называемых **полями**. Поля могут быть числовыми, строковыми, текстовыми, рисунковыми. Это позволяет дать в описании объекта и количественные характеристики (числовые поля), и названия (строковые поля), и расширенные описания (текстовые поля), и даже графические изображения (рисунковые поля). Примером структуры записи может служить запись, описывающая сведения о закупке товара, структура которой показана на рис. 6.1.

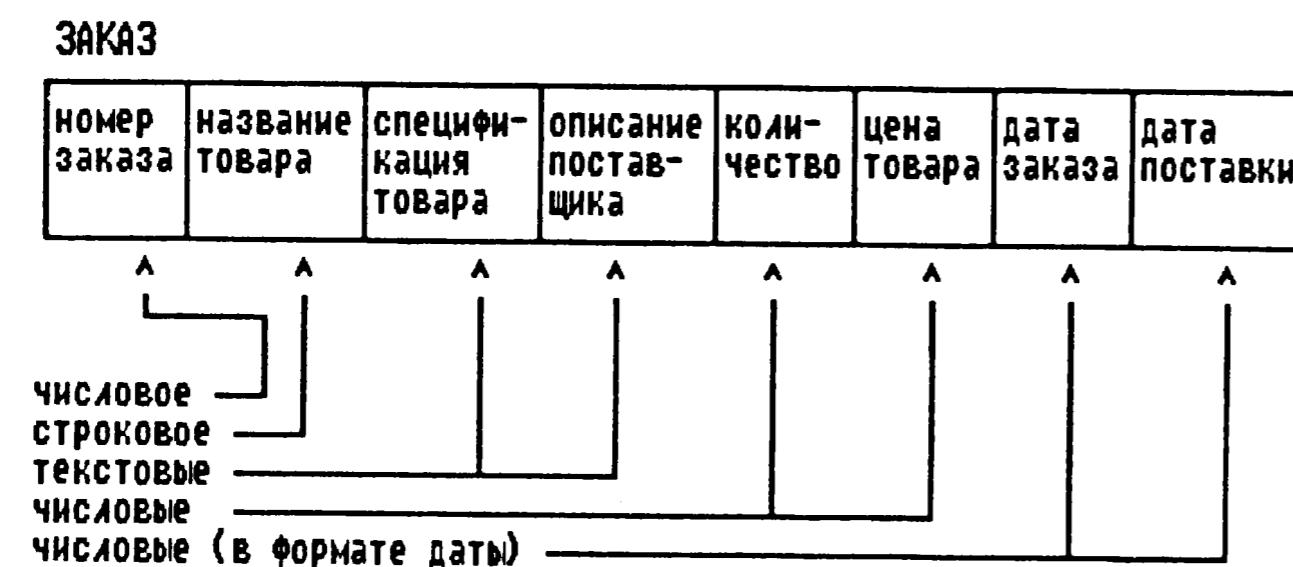


Рис. 6.1. Пример структуры записи

Диалоговая работа с записями предполагает, что записи могут быть представлены пользователю на экране и он может задавать или изменять значения их полей. Такой доступ к записи

сям осуществляется в МАСТЕРЕ либо через форматки, либо через таблицы.

Форматный доступ к записям

При форматном доступе все внимание уделяется единичной записи. Она располагается на экране в отдельной составной рамке, называемой **форматкой**, внешний вид которой отражает содержательную структуру записи. Например, для записи о закупке форматка может иметь вид, изображенный на рис. 6.2. Форматка реализуется в МАСТЕРЕ с помощью составной рамки, внутри которой располагаются рамки-поля и вспомогательные рамки, служащие для оформления ее внешнего вида.

Рис. 6.2. Форматка для доступа к записи

Табличный доступ к записям

При табличном доступе все записи выстраиваются одна за другую и располагаются в таблице, занимая по одной строке на каждую запись. Здесь каждой записи уделено существенно меньше места, но зато записи показаны в совокупности, их можно сопоставлять друг с другом, охватывая какие-то общие закономерности в представляющей ими информации. Например, для записей о закупках табличная форма представления имеет вид, показанный на рис. 6.3. Из этой таблицы видно, что имеются два заказа с номерами 2101 и 2105. Первый заказ представлен

четырьмя записями – товары ВАЗ-2101, ВАЗ-21011, ВАЗ-21013, ВАЗ-2102, а второй заказ – тремя записями о товарах ВАЗ-2105, ВАЗ-21061, ВАЗ-21063.

Номер заказа	Наименование товара	Спецификация	Описание поставщика	Коли-чество	Цена товара	Дата заказа	Дата поставки
2101	ВАЗ-2101	[]	[]	20	6500	10 янв	20 мая 1988
2101	ВАЗ-21011	[]	[]	5	7200	10 янв	20 мая 1988
2101	ВАЗ-21013	[]	[]	0	7200	10 янв	20 мая 1988
2101	ВАЗ-2102	[]	[]	0	7800	10 янв	20 мая 1988
2105	ВАЗ-2105	[]	[]	20	8300	10 янв	20 мая 1988
2105	ВАЗ-21061	[]	[]	50	9200	10 янв	20 мая 1988
2105	ВАЗ-21063	[]	[]	10	9600	10 янв	20 мая 1988

Рис. 6.3. Табличный доступ ко множеству записей

Табличная форма доступа к записям реализуется в МАСТЕРе через обычные электронные таблицы. Поэтому над записями, представленными в табличной форме, можно делать многие из тех действий, которые доступны в электронных таблицах, в частности, переставлять столбцы, изменять их ширину, высоту и форматы визуализации значений полей.

Основными действиями, из которых складывается в конечном счете вся деятельность пользователя в базе, являются следующие:

- ввод новых записей;
- модификация существующих записей;
- поиск нужных записей по значениям полей;
- последовательный перебор записей;
- упорядочение записей по значениям полей;
- получение упорядоченных таблиц записей и работа с ними.

Типы записей

Если посмотреть внимательнее на приведенную структуру записи о заказе, можно увидеть в ней серьезный недостаток. Как правило, в одном заказе содержится не один а несколько товаров – именно эта ситуация представлена в примере табличного доступа на рис. 6.3. В заказе 2101 содержатся четыре товара, а в заказе 2105 – три товара, и поэтому они должны быть представлены соответственно четырьмя и тремя записями. Можно попытаться, изменив структуру записи, включить в нее не одну, а несколько таких четверок, позволяя объединить описания

нескольких товаров в одной записи. Тогда возникнет проблема операционального расхода памяти: с одной стороны, всегда может оказаться такой заказ, в котором будет перечислено товаров больше максимально предусмотренного, а с другой стороны, в большинстве записей неиспользованные поля будут попусту занимать место в базе. Кроме того, разнесение однородных элементов данных о нескольких товарах по разным полям записи создаст затруднение при поиске заказов по наименованиям товаров и вообще при массовой обработке товаров в заказах. В структуре, представленной на рис. 6.3, мы поступили по-другому. Один заказ представлен в ней не одной записью, а несколькими – по одной на каждый из товаров, включенных в данный заказ. При этом мы вынуждены многократно дублировать значения некоторых полей, относящихся к заказу в целом: описание поставщика, дата заказа, дата поставки.

Это очень типичное затруднение в организации баз данных и означает оно то, что в схеме базы сделана ошибка: информация, которая должна быть представлена *несколькими типами* записей, сведена в записи *одного типа*. Так, в данном случае сведения о заказах в целом должны храниться в записях одного типа, а сведения о каждом из товаров этого заказа – в отдельных записях другого типа. Состав полей в записи для описания заказа должен быть один, а состав полей в записи для описания товара – другой. Разделение нашего исходного типа записей ЗАКАЗ на два новых типа записей – ЗАКАЗ и КОЛИЧЕСТВО_ТОВАРА – показано на рис. 6.4.

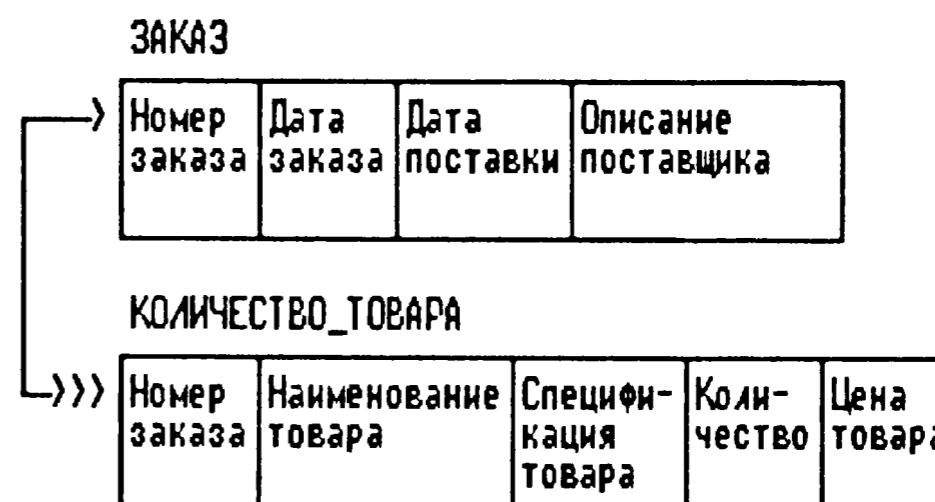


Рис. 6.4. Разбивка записи заказа на две записи

Наша база данных будет теперь состоять из записей двух типов – записей о заказах и записей о количествах товаров в этих заказах. В реальных базах данных таких типов записей может быть значительно больше. Чем большее количество разных

типов записей, тем более сложную логическую структуру имеет база, тем более богатые связи можно в ней представить.

ПРИМЕЧАНИЕ. В некоторых информационных системах (скажем, в dBASE III) вместо понятия "тип записей" используется термин "база данных". При этом, подразумевая совместную обработку нескольких типов записей, говорят об одновременной обработке нескольких баз данных. В МАСТЕРе принята более традиционная терминология; в соответствии с ней считается, что пользователь работает не с несколькими базами данных, а с одной, состоящей из многих типов записей.

Реляционные соединения записей

Продолжим анализ логической структуры базы данных. Обратим внимание на следующее обстоятельство. В записи о товаре повторено поле "номер заказа" из записи о заказе. Это имеет принципиальное значение, поскольку количество товара указывается не отвлеченно, а применительно к вполне определенному заказу. Тот же товар может появиться в другом заказе в другом количестве – это будет другая запись о количестве товара. Поле "номер заказа" относит запись о количестве товара к соответствующей записи о заказе.

Таким образом, мы разделили исходную запись о заказе на части, но эти части не стали независимыми – они соединены друг с другом посредством связующего поля "номер заказа", присутствующего и в той и в другой записи. На рис. 6.4. это соединение показано стрелкой. Совсем не обязательно, чтобы связующие поля назывались в разных записях одинаково. Главное – это то, что они идентифицируют один и тот же объект: в данном случае – номер заказа. По своему содержанию это соединение записей о заказах с записями о количествах товаров имеет характер "один ко многим", а обратное соединение – "один к одному".

Бывают соединения и более сложного характера – "многие ко многим", но они требуют для своего представления еще один тип записей-посредников. Рассмотрим эти соединения на том же примере о заказах. Можно заметить, что из-за повторяемости записей о количествах товаров приходится многократно записывать одни и те же название, спецификацию и цену (имеющие, быть может, значительный размер). В этом случае можно сделать еще одно разделение: информацию о количестве товара отделить от информации об описании самого товара как такового. Для этого закодируем все товары числовыми кодами и разделим тип записи "количество товара" на два типа записи: ТОВАР и КОЛИЧЕСТВО_ТОВАРА. Общая получившаяся структура базы

данных показана на рис. 6.5. Соединения между типами записей показаны стрелками.

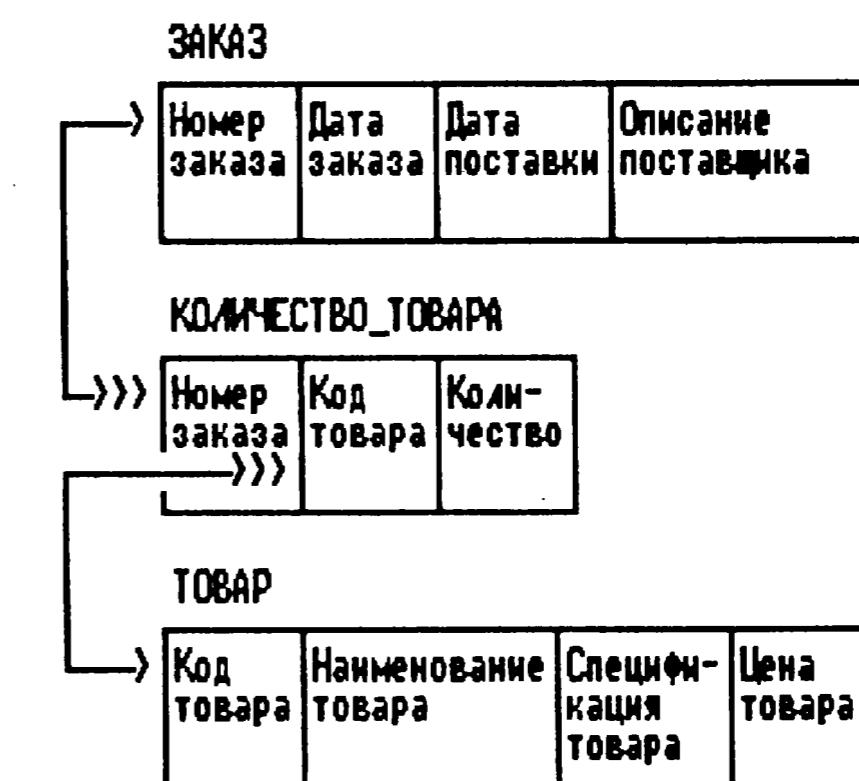


Рис. 6.5. Соединение типа "многие ко многим"

В этой схеме базы сведения о товарах уже не должны дублироваться многократно: описание каждого товара имеется лишь в одном экземпляре. Записи о количествах товаров представляют собой комбинации номера заказа и кода товара. Каждой комбинации номера заказа и кода товара ставится в соответствие число – количество этого товара в данном заказе. Эти записи описывают количественные включения товаров в заказы. Все три типа записей в целом представляют соединение "многие ко многим": каждый товар входит во многие заказы (в разных количествах), и каждый заказ включает многие товары.

В принципе трех перечисленных типов соединений достаточно для того, чтобы представлять связи самой сложной структуры. В сущности, базовых типов соединений лишь два, поскольку соединение "многие ко многим" является комбинацией соединений типа "один ко многим" и "многие к одному".

Обратите внимание на то, что, начав разрабатывать базу данных, мы создали один тип записи со многими полями, а в конце после детального анализа пришли к нескольким коротким типам записей. Это не случайно. Вообще стремление к неизбыточности и гибкости структуры базы данных диктует необходимость введения многих типов записей с небольшим числом полей, а не одного-двух огромных типов записей. Поэтому базы данных типа dBASE III, ориентированные на небольшое (до 10) число одновременно обрабатываемых типов записей, существенно проигрывают в гибкости базе данных МАСТЕРа, не накладывающей

никаких ограничений на число одновременно обрабатываемых типов записей и предоставляемой развитые средства по соединению этих записей друг с другом.

Для желающих более глубоко изучить вопрос о принципах проектирования баз данных можно порекомендовать познакомиться с теорией нормализации реляционных баз данных. Рассмотренный процесс постепенного расчленения типов записей называется **нормализацией** и имеет глубокие теоретические основы.

6.2. Технологические понятия базы данных

В предыдущем разделе были рассмотрены те понятия баз данных МАСТЕРа, которые составляют ее логическую основу. База данных, однако, не является всего лишь концептуальным объектом. Она располагается на диске, причем может занимать огромное место. Большой объем хранимой информации обуславливает повышенные требования к решению таких проблем, как надежность хранения, эффективность поиска и обработки, компактность представления. Поэтому желательно, чтобы пользователь имел представление не только о логической структуре базы, но также и о некоторых технологических аспектах связанных с ней. Понятия, требующиеся для этого, рассматриваются в настоящем разделе.

Способы хранения значений в полях записей

Рассмотрим типы данных, которые могут храниться в полях записей. К ним относятся числа, строки, тексты и рисунки. Это как раз все те типы данных (за исключением таблиц и составных рамковых структур), которые могут храниться в рамках МАСТЕРа. Однако в отличие от рамок, в которых не предполагается хранить чрезмерно большие объемы информации и для которых поэтому не требуется дополнительных уточнений о способе хранения, в базе данных такого разделения на типы недостаточно. Здесь необходимо указывать еще, какого размера память должна быть отведена под те или иные числа и строки. Так, числовые поля могут занимать один из четырех фиксированных размеров: один, два, четыре или восемь байт. Соответственно размеру представления в памяти числа могут принимать значения из следующих диапазонов:

- 1 байт – целые числа от -127 до +127 (2-3 цифры);
- 2 байта – целые числа от -32767 до +32767 (4-5 цифр);
- 4 байта – целые числа от -2x10⁹ до +2x10⁹ (9 цифр);
- 8 байт – вещественные числа от 4.19x10⁻³⁰⁷ до 1.67x10⁺³⁰⁸ (15-16 значащих цифр);

Фактически такое представление является упакованным: оно примерно в 2 раза более компактно, чем строковое представление тех же чисел.

Для строковых полей допустимы два различных способа хранения: строковые поля фиксированной и переменной длины. Более удобны для пользователя **строковые поля переменной длины**. Используя их, можно не заботиться о количестве символов в строке, поскольку в них допустимы как совсем короткие (например, односимвольные), так и длинные (до 255 символов) строки. Однако на каждое такое поле приходится дополнительный служебный расход памяти в размере около 7 байт. С целью экономии памяти можно пользоваться **строковыми полями фиксированной длины**. Такое поле занимает всегда одно и то же количество байт в записи – столько, сколько указано при определении этого типа записи. Ясно, что при этом строки большей длины не смогут уместиться в поле, а строки меньшей длины будут храниться в избыточном объеме памяти. Это представление выгодно в тех случаях, когда размеры строк в разных записях не слишком сильно отличаются друг от друга.

Для текстовых и рисунковых значений никаких дополнительных уточнений способа хранения не требуется. В теле записи каждое такое значение представлено 4 байтами, использующимися для ссылки на текст или рисунок. Само же значение упаковывается отдельно от записи в автоматически подбираемое свободное место базы.

Области базы данных на диске

Единицей хранения с этой точки зрения является файл. Файлы базы данных МАСТЕРа имеют специфическую внутреннюю организацию, ориентированную на эффективное представление и обработку всех требуемых в ней структур данных. Подчеркивая их структурное отличие от прочих файлов, используемых МАСТЕРОМ (рамковых, текстовых), будем называть эти файлы **областями базы данных**.

Одна база данных может состоять из нескольких (от 1 до 7) одновременно открываемых областей. Назначение областей определяется разработчиком базы данных произвольно. Так, например, можно разместить все используемые в базе типы записей и все вспомогательные структуры (упорядочивающие индексы) в одной области. Можно, наоборот, разные типы записей и разные вспомогательные структуры разнести по разным областям. В этом отношении имеется одно ограничение: все записи одного типа должны находиться в одной области. Разнесение информации по областям следует делать исходя из соображений модульности.

По отношению к операционной системе области являются обычными файлами. Их можно копировать, уничтожать, переносить с диска на диск и из каталога в каталог обычными файловыми командами. Каждая область обладает именем, формируемым по обычным правилам MS DOS, описанным в 1.1. С другой стороны, что касается внутреннего строения базы данных, то все составляющие ее области пронумерованы целыми числами, называемыми **кодами областей**. Код каждой области определяется при ее создании и не может быть изменен. Все одновременно открываемые области базы данных должны иметь разные коды, поскольку именно по кодам они различаются при работе с базой. Само значение кода не играет никакой роли, и поэтому, если в архитектуре базы нет каких-либо особенностей, то коды можно назначать последовательными целыми числами, начиная с нуля.

Порядки записей

Следующее важное технологическое понятие базы данных связано с упорядоченностью записей. Упорядоченность играет двойную роль. Во-первых, в содержательном отношении упорядоченность позволяет рассматривать записи в определенной последовательности; например в порядке возрастания номеров, цен, дат, в алфавитном порядке и т.п. Во-вторых, в техническом отношении явным образом хранящийся в базе порядок (в виде индекса) позволяет находить записи во много раз быстрее, чем при последовательном просмотре.

Множество записей каждого типа может быть упорядочено несколькими разными способами. Каждое упорядочение формируется по возрастанию или убыванию значений того или иного поля и называется **порядком** данного типа записей. Существует несколько способов образования порядков по значениям полей – по отдельным полям, по разным комбинациям полей, по возрастанию или по убыванию значений. Различные способы упорядочения обеспечивают разные возможности поиска записей, и от того, как спроектированы порядки в базе, могут весьма значительно зависеть ее поисковые возможности и эффективность.

Каждый порядок представляет собой некоторую структуру данных (называемую в других системах **инвертированным списком**, индексом или В-деревом). Эта структура занимает определенную память в базе данных, а также требует для своего поддержания в корректном состоянии некоторого времени при выполнении каждой модифицирующей операции, производимой над записями этого типа.

Упорядочение по слиянию полей

Рассмотрим более детально те способы, которыми могут упорядочиваться множества записей в базе данных МАСТЕРа.

Первый из этих способов называется **упорядочением по слиянию полей**. В этом упорядочении используется одно или несколько полей. Значения этих полей в определенной последовательности берутся из каждой записи, сливаются друг с другом (т.е. ставятся в строку одно за другим), и такое составное значение используется как ключ упорядочения.

Пример такого упорядочения можно привести на рассмотренном типе записей "количество товара". Для упорядочения возьмем все три поля этого типа записей и расположим их в ключе в следующей последовательности: номер заказа, код товара, количество. При упорядочении записей по такому составному ключу окажется, что в основном записи будут расположены в порядке возрастания номера заказа, так что сначала будут стоять все записи о первом заказе, затем все записи о втором заказе и т.д. В пределах одного заказа записи упорядочиваются по возрастанию кода товара, а уж для одинаковых кодов товара внутри заказа они будут упорядочены по возрастанию количества поставляемого товара.

В приведенном примере все поля использованы в ключе только по возрастанию. Точно так же можно использовать поля в ключе по убыванию их значений, а можно ставить поля в упорядочивающий ключ и с разными знаками: некоторые по возрастанию, а некоторые по убыванию значений. Так, в приведенном примере поле "количество товара" целесообразнее брать в убывающем порядке, чтобы первыми оказались более значимые заказы.

Очевидно, что при определении порядков по слиянию полей последовательность полей, и в особенности выбор самого первого поля ключа, очень существенна: при перестановке полей в ключе порядок сразу приобретает совсем иной смысл. Будем называть **рангом поля в ключе** ту позицию, которую это поле занимает в последовательности полей, образующих ключ. Наиболее значимым является первое поле (ранг 1), затем второе (ранг 2) и т.д.

Упорядочение по смешению полей

Описанный способ упорядочения – по слиянию полей – является традиционным и используется во всех системах управления базами данных. В базе данных МАСТЕРа кроме этого способа имеется еще один, ориентированный на поддержку понятия "ключевое слово", необходимого в системах типа картотек, очень

распространенных в автоматизации учрежденческой деятельности. Этот способ называется **упорядочением по смешению полей**.

В этом упорядочении тоже используется несколько полей. В отличие от упорядочения по слиянию, где поля в ключе могут быть разного типа, все смешиаемые поля обязательно должны быть однотипными – либо только строковыми, либо только числовыми. Поля не соединяются друг с другом, как при слиянии, они используются независимо: каждая запись вставляется в порядок не один раз, а несколько – по одному разу на каждое из смешиемых полей. В результате, при просмотре одна запись встретится несколько раз – в тех позициях, которые определяются разными полями.

Это отражает понятие карточки с ключевыми словами. Если на карточке написано несколько разных слов, то она может быть найдена по каждому из них. Упорядочение по смешению удобно во всех тех случаях, когда в записи имеется несколько однородных полей (фамилии исполнителей некоторого поручения, коды товаров в одном заказе, ключевые слова текстовой карточки и т.п.), и при поиске записи несущественно, в каком из полей встретится значение. Эти поля смешиваются друг с другом.

Все поля при смешении являются равноправными и никаких рангов, которые имеются в слиянии, здесь нет.

Частным случаем упорядочения по смешению является порядок, когда в смешении участвует лишь одно поле. Такой порядок полностью эквивалентен порядку по слиянию с использованием только одного поля. Таким образом, при упорядочении по одному полю вы можете выбирать любой способ.

Упорядочение по физической вставке

Оба описанных способа упорядочения записей основываются на значениях тех или иных полей. Иногда требуется располагать записи в определенном порядке, никак не связанном с какими бы то ни было полями. Такая возможность предоставляется третьим способом упорядочения, называемым **упорядочением по физической вставке** или просто **по вставке**. Записи в таком порядке стоят на тех местах, где они были созданы и никак не переставляются при изменении значений их полей.

К этому способу упорядочения обращаются весьма редко. Рекомендуется в любом порядке использовать для упорядочения то или иное поле, например номер записи. Порядки, образованные "по вставке", почти бесполезны: любой поиск в таком порядке делается только последовательным просмотром записей, что совершенно неприемлемо при больших объемах информации.

Автоматическое поддержание порядков

Существенным свойством всех порядков в базе данных МАСТЕРа является то, что они автоматически поддерживаются во время всех модифицирующих операций: при создании новых записей, при уничтожении записей, при изменении значений полей записей. Создав однажды в вашей базе данных некоторый порядок, вам не потребуется переупорядочивать записи после каких-то изменений или добавлений данных – этот порядок всегда будет находиться в корректном состоянии.

Главные порядки типов записей

Для одного типа записей может быть создано сколько угодно разных порядков. Можно упорядочивать записи разными способами: по разным полям, по разным комбинациям полей, разными методами (по слиянию или по смешению). Все такие порядки будут равноправными в том смысле, что при работе с базой вы сможете переключаться с одного порядка на другой практически мгновенно. Это будет выглядеть как мгновенное переупорядочение записей то в одной, то в другой последовательности.

Среди всех порядков выделяется **главный порядок**. Он неизбежно возникает при создании любого типа записей, даже если их вообще не требуется упорядочивать. Необходимость в главном порядке существует потому, что вообще без порядков записи существовать не могут: они доступны лишь через тот или иной порядок. Поэтому уничтожение всех порядков означает уничтожение возможности доступа к записям. Выделенность главного порядка состоит в том, что его в отличие от всех остальных порядков невозможно уничтожить, точнее, его уничтожение приведет к одновременному уничтожению всех остальных порядков и самих записей этого типа.

Еще одно отличие состоит в том, что из всех порядков только главный порядок может быть упорядочен по вставке. Остальные порядки должны быть упорядочены либо по слиянию, либо по смешению каких-то полей.

Порядки и реляционные соединения записей

Порядки имеют непосредственное отношение к реляционным соединениям записей. Это обусловлено уже упоминавшейся технической ролью порядков как структур, предназначенных для быстрого поиска записей по значениям полей. Поиск записей

путем последовательного их перебора из-за больших объемов информации представляется заведомо неприемлемым. Поэтому мы будем считать, что поиск возможен только по тем полям, по которым существуют порядки (причем поле должно иметь первый ранг в порядке по слиянию). Соединения в реляционной базе реализуются именно поиском по значению поля. Следовательно, для реализации соединения необходимо, чтобы записи были упорядочены по соединяемым полям.

Проиллюстрируем это на примере. В рассмотренной схеме с тремя типами записей возможны четыре разных соединения:

- (1) ЗАКАЗ.номер_заказа --->>>
КОЛИЧЕСТВО_ТОВАРА.номер_заказа;
- (2) КОЛИЧЕСТВО_ТОВАРА.номер_заказа --->
ЗАКАЗ.номер_заказа;
- (3) КОЛИЧЕСТВО_ТОВАРА.код_товара --->
ТОВАР.код_товара;
- (4) ТОВАР.код_товара --->>>
КОЛИЧЕСТВО_ТОВАРА.код_товара.

Стрелками обозначено направление, в котором осуществляется поиск. Например, в первом соединении мы берем поле "номер_заказа" из записи типа ЗАКАЗ и ищем все те записи типа КОЛИЧЕСТВО_ТОВАРА, в которых поле "номер_заказа" имеет такое же значение. По смыслу рассматриваемой структуры в одном заказе может быть несколько разных товаров, поэтому таких записей может быть найдено несколько. Это соединение типа "один ко многим" обозначено тройной стрелкой --->>>. Одинарной стрелкой ---> обозначаются соединения типа "один к одному". Так, с одной записью типа КОЛИЧЕСТВО_ТОВАРА связана строго одна запись типа ЗАКАЗ – это тот единственный заказ, которому принадлежит поставка товара, описываемая записью КОЛИЧЕСТВО_ТОВАРА.

Требуются ли в работе все четыре соединения или нет, зависит от того, какие поисковые операции мы собираемся разрешить в базе. Так, для определения по номеру заказа всех входящих в него товаров потребуется цепочка соединений

(1),(3): ЗАКАЗ --->>> КОЛИЧЕСТВО_ТОВАРА ---> ТОВАР

Для определения по виду товара перечня всех тех заказов, в которые входит этот товар, потребуется обратная цепочка

(4),(2): ТОВАР --->>> КОЛИЧЕСТВО_ТОВАРА ---> ЗАКАЗ

Формально в структурном отношении обе эти цепочки подобны друг другу. С учетом же практической потребности без второй

134

цепочки можно обойтись, в то время как первая цепочка явно необходима.

Для реализации соединений требуется возможность выполнения поиска по тем полям, на которые в приведенных обозначениях указывают стрелки. Так, для реализации соединения

(1) ЗАКАЗ.номер_заказа --->>>
КОЛИЧЕСТВО_ТОВАРА.номер_заказа

необходимо, чтобы записи типа КОЛИЧЕСТВО_ТОВАРА были упорядочены по полю "номер_заказа". Для реализации соединения

(3) КОЛИЧЕСТВО_ТОВАРА.код_товара ---> ТОВАР.код_товара

нужен еще один порядок – записи типа ТОВАР должны быть упорядочены по полю "код_товара". Для реализации цепочки (4),(2) потребовалось бы еще два порядка.

При определении схемы базы данных можно будет не заботиться специально об определении порядков. Достаточно указать, какие именно требуются соединения типов записей, и все нужные порядки будут созданы автоматически. Все такие автоматически порождаемые порядки будут использовать только одно поле в ключе упорядочения. Их можно "подправить", добавив по слиянию поля следующих рангов, чтобы упорядочение оказалось более естественным.

6.3. Определение схемы базы данных

Когда вы создаете обычную текстовую или табличную рамку, достаточно всего лишь выбрать ее тип из меню и определить имя – и рамка сразу будет готова к использованию. В случае с базой данных все обстоит значительно сложнее. При создании базы данных вы должны указать, какие типы записей вы хотите в ней иметь, какие поля должны входить в каждый из этих типов записей, какого типа будут эти поля и какой способ хранения следует использовать для каждого из них, какие порядки и какие соединения записей должны поддерживаться в базе, начиная с каких областях (файлах) на диске должны быть размещены все эти структуры. Только ответив на совокупность всех этих вопросов, вы определите структуру той информационной среды, в которой собираетесь хранить и обрабатывать информацию в вашей базе. Эта структура называется **схемой базы данных**, а для ее определения в МАСТЕРЕ существует специальная диалоговая поддержка, описываемая в данном разделе.

Начальное меню базы данных

Вообще всякий выход на базу данных происходит через главное меню МАСТЕРа. В нем имеется пункт База, который вызывает вспомогательное меню, содержащее три пункта: Открыть, Создать и Доопределить. Первый пункт используется для входа в саму базу данных для работы с содержащейся в ней информацией. Два других пункта обеспечивают вход в режим определения схемы базы данных: пункт Создать позволяет создать новую базу, начав ее определение с пустой схемы, а пункт Доопределить позволяет продолжить или изменить определение какой-то существующей схемы.

Выполнение пунктов Создать и Доопределить начинается с запроса: "Задайте имя базы". Ответом на него должен служить идентификатор, записываемый латинскими буквами, поскольку он будет использоваться в качестве мнемоники для тех файлов, в которых будет располагаться база данных на диске. После задания имени базы на рабочее поле считывается специальная рамка, в которой содержится описание схемы создаваемой вами базы данных. Нужно войти в эту рамку, где вы сможете, заполнив определенные поля, определить структурную схему базы данных.

Рамка схемы базы данных

Понятие схемы базы данных включает в себя очень много компонентов, количество которых возрастает тем более, чем больше разных типов записей будет в базе данных. Для структурного и наглядного представления всех этих компонентов схемы используется сложная система рамок, являющаяся интерактивной средой для создания и редактирования схемы в таком же экранном режиме, в каком редактируются тексты или таблицы. Вся эта интерактивная среда представляется в виде составной рамки, называемой **рамкой схемы базы данных**.

Рамка-схема имеет двухуровневую структуру. Войдя в нее через одну из операций Создать или Доопределить, вы оказываетесь на первом уровне – в **сводной таблице схемы** (рис. 6.6). Здесь дается в табличной форме сводка областей, из которых состоит база, типов записей, которые в нее включены, соединений, которые установлены между записями.

Второй уровень схемы базы данных дает доступ к определениям отдельных типов записей. Для каждого типа записи в схеме существует одна рамка, в которой в наглядной форме (рис. 6.7) приведены все сведения об этом типе записей: имена, типы и размеры каждого поля в отдельности и всех полей вместе, перечислены все имеющиеся для этого типа записей порядки.

Каждая такая рамка называется **таблицей определения типа записей**.

Переход с одного уровня на другой происходит точно также, как и между уровнями рамковой иерархии – с помощью клавиш {ПЛЮС} и {МИНУС}. Нажатие клавиши {ПЛЮС} в позиции, когда курсор стоит в сводной таблице на имени некоторого типа записей, приводит ко входу в таблицу определения этого типа записей. Возврат из таблицы определения типа записей в сводную таблицу происходит с помощью нажатия клавиши {МИНУС}.

В этой рамке Вы можете создавать и редактировать типы записей
#18-вызов меню ПЛЮС-вход в рамку

СХЕМА БАЗЫ ДАННЫХ				Версия 1.4
Области Базы Данных				Название базы
Имя области	institut			INSTITUT
Размер буфера				
Типы записей базы данных				
Имя типа	сотрудник	отдел	тема	
Область	institut	institut	institut	
Число полей	6	3	4	
Разм. записи	19	12	14	
Соединения				
Множество 1	сотрудник	отдел	отдел	
Поле 1	отдел	название	начальни	
Множество 2	отдел	сотрудник	сотрудни	
Поле 2	название	отдел	фамилия	
Вид связи	1:1	1:M	1:1	

Рис. 6.6. Сводная таблица схемы базы данных

Сводная таблица схемы базы

Сводная таблица схемы базы данных состоит из трех частей – списка областей, списка типов записей, списка соединений записей. Каждый из этих списков представлен рамкой, и передвижение по таблице от списка к списку осуществляется по обычным правилам МАСТЕРа с помощью клавиш-стрелок {ВНИЗ} и {ВВЕРХ}, а для входа и выхода в рамки-списки также используются обычные клавиши {ПЛЮС} и {МИНУС}.

Список областей, располагающийся в верхней части сводной таблицы, представляет собой таблицу, каждый столбец которой содержит описание одной области базы данных. Столбец состоит из двух позиций: верхняя – имя файла, в котором должна рас-

полагаться область, нижняя – целое число от 0 до 6, задающее код этой области.

Список типов записей, находящийся в середине сводной таблицы, содержит по одному столбцу на каждый тип записи. В верхней позиции столбца записывается имя типа записей, во второй позиции – имя области, в которой его следует расположить. Две нижних позиции заполняются автоматически и недоступны для исправления. Они играют информационную роль и содержат: количество полей в записях этого типа и размер записи в байтах. Первоначально оба числа оказываются равными нулю. Они будут показывать ненулевые значения после того, как вы определите структуру этого типа записи.

Список соединений, расположенный внизу сводной таблицы, описывает набор реляционных связей, поддерживаемых в базе. Каждый столбец здесь описывает одно из соединений. В столбце находятся две пары имен, каждая из которых состоит из имени типа записи и имени поля в этом типе. Первая пара (верхняя) задает то поле, из которого исходит соединение, вторая пара (нижняя) задает то поле, на которое это соединение указывает. Под этими четырьмя именами в каждом столбце стоит один из знаков 1:1 или 1:M, обозначающих тип соединения – "один к одному" или "один ко многим". Так например, соединение

(1) ЗАКАЗ.номер_заказа --->>
КОЛИЧЕСТВО_ТОВАРА.номер_заказа

будет представлено в таблице соединений столбцом следующего вида:

ЗАКАЗ
номер_заказа
КОЛИЧЕСТВО_ТОВАРА
номер_заказа
1:M

а соединение

(3) КОЛИЧЕСТВО_ТОВАРА.код_товара ---> ТОВАР.код_товара

будет представлено столбцом:

КОЛИЧЕСТВО_ТОВАРА
код_товара
ТОВАР
код_товара
1:1

Таблица определения типа записей

Для каждого типа записи в рамке-схеме имеется собственная таблица, называемая **таблицей определения типа записей**, содержащая всю необходимую информацию об этом типе.

Введите имя поля (УДА-удаление поля ВСТ-вставка поля)
Это должен быть идентификатор

СТРУКТУРА ТИПА ЗАПИСИ							
Имя типа записи : сотрудник	Число полей : 6						
Имя области : institut	Размер Записи : 19 байт						
	Число порядков : 1						
Порядки основного множества :							
Имя поля	Тип	Длина	Главное	institut			
Фамилия	С	4	С	-	-	-	-
отдел	С	4	-	С	-	-	-
оклад	Ч	2	-	-	-	-	-
тема	Т	4	-	-	-	-	-
возраст	Д	2	-	-	-	-	-
партийность	И	3	-	-	-	-	-

Рис. 6.7. Таблица определения типа записей

Структура этой таблицы, хотя и содержит довольно много позиций, достаточно наглядна и понятна сама по себе. Пример такой таблицы приведен на рис. 6.7. В левой части располагаются имена полей – здесь вы можете вводить произвольные идентификаторы. Справа от каждого имени стоит буква, обозначающая тип поля:

- Ч – числовое поле;
- С – строка переменной длины (от 0 до 255 байт);
- И – строка фиксированной длины;
- Т – текстовое поле;
- Р – рисунковое поле;
- Д – поле для даты;
- В – поле для времени.

В третьем столбце располагаются числа, задающие размеры соответствующих полей. Для некоторых типов полей (текстовых, рисунковых, полей дат, времен, строк переменной длины) эти размеры фиксированы и проставляются системой автоматически, а для чисел и строк фиксированной длины должны задаваться

пользователем. Размер строки фиксированной длины может быть задан произвольным числом от 1 до 255 – это число означает максимально возможное для данной строки число символов. Для числового поля размер может быть один из четырех – 1,2,4,8. Диапазоны чисел, представляемых этими четырьмя размерами числовых полей, приведены в 6.2.

В столбцах, стоящих еще правее, описываются все те порядки, которые существуют для данного типа записей. Каждый столбец описывает очередной порядок. Описание порядка состоит в том, чтобы указать, по каким полям и каким образом упорядочить записи в этом порядке. Первый (самый левый) из этих столбцов играет особую роль – он описывает главный порядок, который создается в каждом типе записей. Остальные порядки могут не создаваться, если они не требуются.

Определение порядков

При определении порядка требуется прежде всего задать, в какой области следует разместить представляющую его информационную структуру (индекс, организованный в виде В-дерева). Это имя области запрашивается и записывается в ячейке, находящейся в верхней части столбца, определяющего порядок. По умолчанию предлагается нулевая область, по имени совпадающая с именем самой базы данных.

В определении порядков используются четыре символа, стоящих в позициях напротив имен полей:

- – прочерк означает, что поле не участвует в упорядочении;
- в – поле вставляется в ключ методом "по слиянию" и упорядочивается по возрастанию значений;
- у – поле вставляется в ключ методом "по слиянию" и упорядочивается по убыванию значений;
- с – поле участвует в упорядочении методом "по смешению".

Для полей, участвующих в упорядочении по слиянию, помеченных символами в или у, требуется еще указывать ранги полей в составном ключе. Ранги в виде цифр 1,2,3,... записываются рядом с этими буквами, например, в1, у2, в3.

В начальном состоянии таблицы определения типа записей все порядки содержат только прочерки – множество записей этого типа по умолчанию никак не упорядочено. Если оставить таблицу в таком виде, то это будет означать, что ни одного порядка, кроме главного, для этого типа записей не будет создано, а главный порядок будет упорядочен методом "по физической вставке". Рекомендуется не оставлять тип записи

совсем не упорядоченным. Желательно тем или иным образом определить упорядочение главного порядка – лучше всего выбрать для этого то поле, по которому предполагается разыскивать записи этого типа.

Довольно типичная ошибка при определении порядков состоит в том, что несколько полей, по которым желательно иметь порядки, отмечаются в одном столбце, например, в главном порядке. Это приведет к созданию не нескольких разных упорядоченностей по каждому из этих полей, а только одного порядка, формируемого по слиянию всех этих полей. В сущности, записи окажутся упорядоченными только по первому из этих полей. Для действительного создания нескольких порядков требуется отметить по одному полю в разных колонках – каждая из колонок определит очередной порядок.

Меню схемы базы

При нахождении внутри рамки схемы базы главное меню МАСТЕРа исчезает. Вместо него в двух верхних строках экрана высвечиваются подсказки о ваших возможных действиях в той позиции, где вы сейчас находитесь. Возможность войти в меню, однако, не исчезает: клавиша {F10} по-прежнему служит этой цели. Только вместо главного меню МАСТЕРа она вызывает меню схемы базы, в котором содержатся лишь те немногие операции, которые допустимы при работе в схеме. Эти операции перечислены в табл. 6.1.

Таблица 6.1. Меню схемы базы данных

Операция	Действие
Очистить	Уничтожение записей всех или одного типа
Сохранить	Сохранение состояния схемы базы данных
Возврат	Завершение работы со схемой базы данных

Операция Очистить требуется в тех случаях, когда нужно изменить структуру какого-то из типов записей путем добавления или уничтожения полей. Такое изменение возможно лишь в том случае, когда в изменяемом типе записей не существует ни одной записи. Если вы уже начали работать с данной базой и создали хотя бы одну запись некоторого типа и хотите добавить еще одно поле в этот тип, то это вам не удастся: модификация данного типа записей будет заблокирована. Тогда придется пожертвовать информацией, введенной в уже созданные записи и уничтожить их, выполнив операцию Очистить. После

такой очистки блокировка с очищенным типа записей снимается и его структуру можно редактировать.

Операция **Очистить** выполняется по-разному в зависимости от того, где находится курсор. Если он стоит на одном из типов внутри таблицы типов записей, то очищаться будет только один этот тип. Если же курсор находится вне таблицы типов, то очистке будут подлежать все имеющиеся в базе данных типы записей.

Операция **Сохранить** позволяет записать на диск наработанное к данному моменту состояние рамки-схемы базы. Эта операция, вообще говоря, необязательна, поскольку сохранение рамки-схемы всегда выполняется автоматически при завершении работы со схемой. Тем не менее рекомендуется регулярно сохранять рамку-схему во время работы в ней, поскольку, при возникновении сбоев в питании или программном обеспечении рамка-схема окажется несохраненной, может произойти рассогласование между оставшимся вариантом этой рамки-схемы и состоянием самой базы данных, в результате которого доступ к базе может быть затруднен или невозможен.

Операция **Возврат** предназначена для завершения работы над схемой базы данных и для возврата в обычное состояние **МАСТЕРа**. При выполнении этой операции будет автоматически сохранена созданная или модифицированная рамка-схема базы.

Файловая архитектура базы данных **МАСТЕРа**

В результате работы над схемой базы данных вы не только определите ее структуру, но и неявно создадите саму базу. В полном объеме всякая база данных **МАСТЕРа** состоит из трех частей:

1) **рамка-схема базы**. Хранится на диске в виде одного рамкового файла, содержащего описание схемы базы данных. Этот файл требуется только в моменты каких-либо изменений структуры базы, но не для работы с ней;

2) **информационные области данных**. Это те файлы, которые вы определили в схеме базы, и которые служат хранилищами информации базы данных. Эти файлы во время работы с базой находятся в открытом состоянии, они требуют максимально эффективного доступа и значительной емкости для хранения;

3) **рабочая рамка**. Эта рамка хранится на диске в виде одного рамкового файла, содержащего все те рамковые структуры, через которые осуществляется интерфейс пользователя с базой, а также все те программы на языке **Мастер**, которые выполняют специфические для этой базы интерфейсные и информационно-последовательные действия.

Все эти три части базы данных формируются одновременно в процессе определения вами схемы базы при работе в рамке-схеме. Вам не нужно заботиться ни о создании информационных областей, ни о создании рабочей рамки базы. Как только вы завершите работу над схемой и вернетесь в **МАСТЕР** через операцию **Возврат**, вы увидите, что все информационные области базы данных, ее рамка-схема и рабочая рамка уже созданы и находятся в текущем каталоге.

При этом используются следующие имена файлов. Рамка-схема хранится в файле, мнемоника которого совпадает с заданным вами именем базы, а расширителем имени этого файла является **SCM**. Файлы информационных областей имеют те имена, которые вы дали им при определении схемы базы. Рабочая рамка хранится в файле с мнемоникой, совпадающей с именем базы, и имеет расширитель **FRM**.

На рис. 6.8 схематически представлены все три компоненты базы данных во взаимодействии друг с другом. Над этими компонентами изображено начальное меню базы, осуществляющее вход либо в рамку-схему, либо в рабочую рамку.

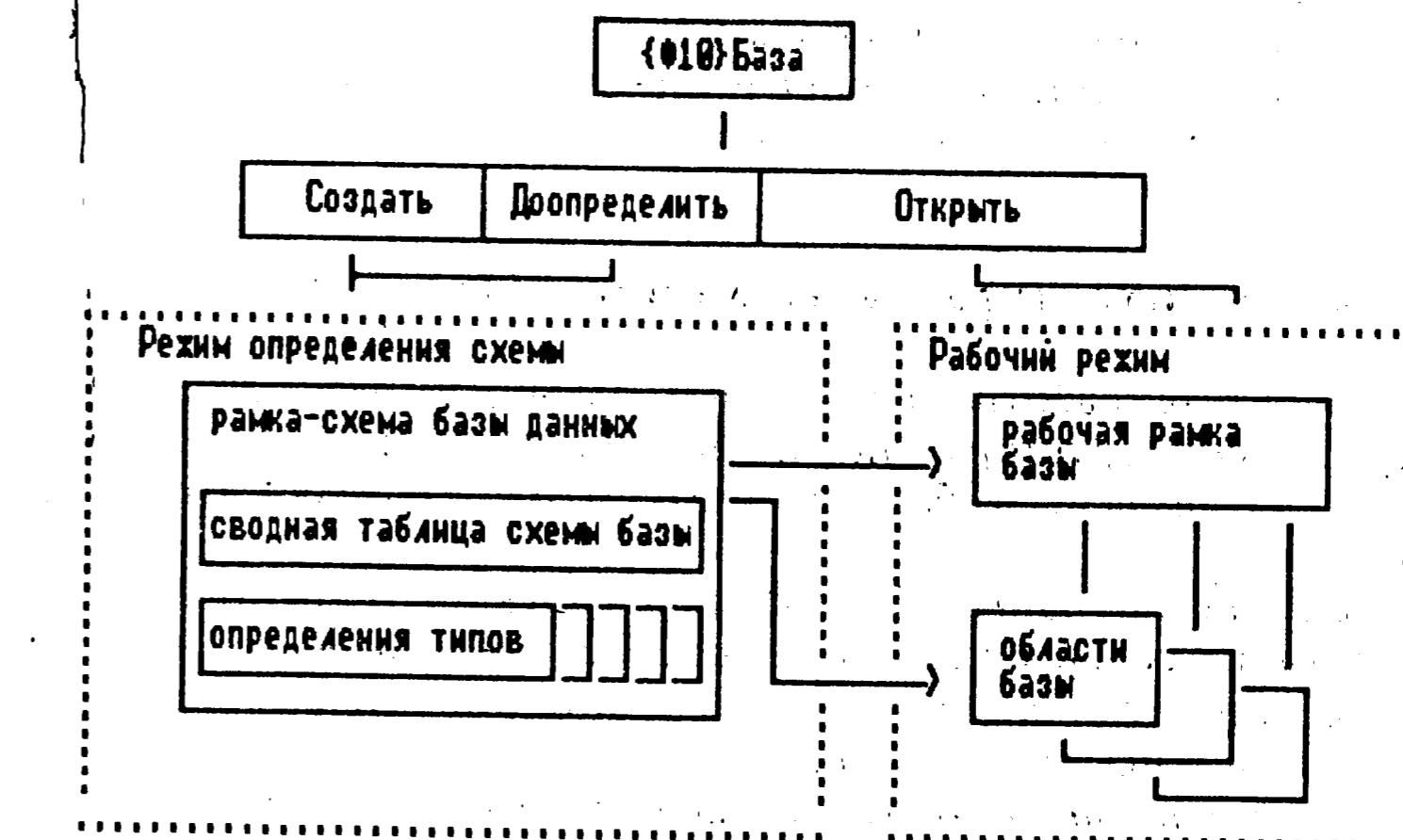


Рис. 6.8. Файловая архитектура базы данных

6.4. Рабочая интерактивная среда базы данных

Рабочая рамка базы

После того, как в рамке-схеме данных выполнены все операции по определению структуры базы, база оказывается готова к

практической работе с ней. Работа в базе, т.е. выполнение различных операций над записями, будет осуществляться через рабочую рамку базы, созданную автоматически во время описания схемы. Для начала работы требуется всего лишь вызвать ее на экран. Вообще каждый сеанс работы с базой данных должен будет начинаться с загрузки рабочей рамки базы и входа в нее. Во время загрузки рабочей рамки автоматически подзагружаются системные программы, выполняющие общие функции обслуживания баз данных, а во время входа в нее выполнится открытие информационных областей базы.

Войдя в рабочую рамку, вы окажетесь в такой интерактивной среде, которая отражает структуру вашей информации и обеспечивает диалоговое взаимодействие с этой информацией. Взаимодействие состоит в возможности создавать записи разных типов, последовательно перебирать их, находить по тем или иным поисковым ключам, отслеживать взаимосвязи между записями. Все эти возможности уже заложены в рабочей рамке вашей базы, причем ориентированы они именно на вашу структуру, ту, которую вы определили, работая в рамке-схеме.

Общее устройство рабочей рамки

Для того чтобы пользоваться этими информационно-поисковыми и диалоговыми возможностями, вы должны представлять устройство рабочей рамки и уметь перемещаться по ее структуре.

Рабочая рамка – это обычная составная рамка. Внутри нее расположены некоторые рамки, выполняющие различные функции во взаимодействии с базой данных.

Основным содержимым рабочей рамки являются рамки-записи. Каждая из них представляет один из типов записей и обеспечивает диалоговый доступ к записям этого типа. Рамки-записи расположены друг под другом и по ширине занимают весь экран (рис. 6.9).

Помимо рамок-записей внутри рабочей рамки имеются вспомогательные рамки-функции, выполняющие открытие базы и поддержку диалога. Эти вспомогательные рамки находятся в плоскости рабочей рамки по диагонали от рамок-записей и поэтому обычными клавишами-стрелками до них добраться невозможно. Но если использовать клавиши логического перебора рамок {УПР-ВЛЕВО} и {УПР-ВПРАВО}, то легко перейти в эту вспомогательную область. Это следует делать только в том случае, если вы захотите заняться процедурным развитием базы данных и дописать свои собственные программы для поиска или обработки записей либо для организации диалога с пользователем.

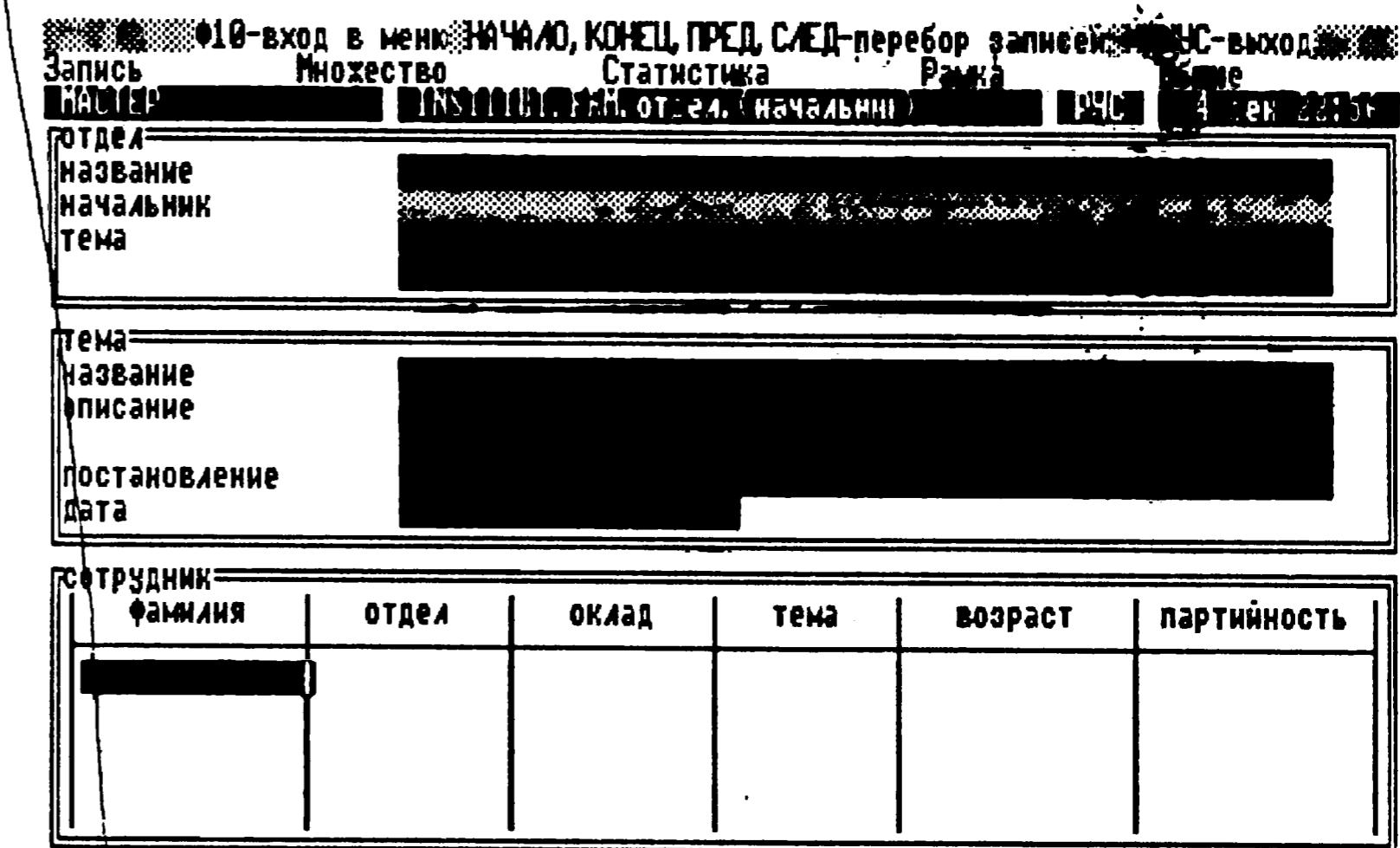


Рис. 6.9. Рабочая рамка базы данных

Устройство рамок-записей

Каждая рамка-запись представляет собой составную рамку, служащую для доступа кциальному типу записи. Внутри этой составной рамки имеются, во-первых, рамки- поля, расположенные на плоскости так, чтобы образовать форматный бланк. Во-вторых, в ней имеется табличная рамка, служащая для осуществления табличного доступа к записям. Когда эта табличная рамка открыта, доступ к записям делается через нее в табличной форме. Когда она закрыта, то доступ осуществляется в форматном режиме с использованием рамок-полей. Помимо таблицы и рамок-полей здесь есть еще одна рамка – текстовая, служащая для образования фона форматки. Она находится в логической цепочке рамок в самой первой позиции, и поэтому все рамки- поля изображаются на ее фоне.

В начальный момент фоновая рамка и рамки- поля образуют форматку некоторого стандартного вида. Вы можете произвольным образом изменить вид этой форматки, для чего нужно войти в фоновую рамку (чтобы добраться до нее, следует двигаться по логической цепочке клавишей {УПР-ВЛЕВО}), создать в ней текстовый фон требуемого вида, а затем изменить положения, размеры и цвета рамок- полей таким образом, чтобы они расположились в нужных местах на фоне созданной вами форматной структуры.

Рабочее меню базы данных

При нахождении курсора внутри любой из рамок-записей главное меню МАСТЕРа сменяется на другое меню – рабочее меню базы. Вход в это меню осуществляется, как обычно, клавишей {Ф10}. Его операции перечислены в табл. 6.2.

Таблица 6.2. Рабочее меню базы данных

Операция	Назначение
Запись	
Найти	Поиск записи по значению поля
Создать	Создание новой записи
Изменить	Занесение измененной записи в базу
Уничтожить	Уничтожение текущей записи
Табличный доступ	Переключение в табличный режим
Форматный доступ	Переключение в форматный режим
Множество	
Полное	Снятие ограничения с записей
Упорядочить	Выборка одного из порядков
Ограничить	Наложение ограничения на записи
Статистика	Перебор записей с выдачей статистики
Сохранить	Сброс буфера базы данных на диск
Рамка	Вызов рамкового меню
Таблица	Вызов табличного меню
Общие	Вызов меню общих операций МАСТЕРА

Переключение форматного и табличного режимов

В каждой рамке-записи имеется электронная таблица, предназначенная для обеспечения табличного доступа к записям. В зависимости от открытого или закрытого состояния этой рамки работа с записями осуществляется либо в форматном, либо в табличном режиме, от чего зависит не только внешний вид рамки-записи, но и способ выполнения некоторых операций. Например, в форматном режиме вы можете одной командой создать, изменить или уничтожить лишь одну запись, а в табличном – произвольное количество записей.

Для переключения из табличного режима в форматный и обратно используются соответственно следующие две команды:

{Ф10} Запись ФорматныйДоступ

{Ф10} Запись ТабличныйДоступ

6.5. Операции над записями

Создание записей

Одной из основных операций при работе с базой данных является создание новых записей. Для ее выполнения нужно сначала заполнить все поля, а затем выполнить команду

{Ф10} Запись Создать

Заполнение полей и создание новых записей возможно как в форматном, так и в табличном состояниях рамки-записи. В форматном состоянии в рамке-записи должны быть заполнены все рамки-поля, после чего выполнение операции создания записи приведет к тому, что из значений, находящихся в рамках-полях, будет сформирована новая запись и введена в базу данных. Следует иметь в виду, что до выполнения этой операции введенные значения находятся только в рамках-полях, но пока еще не в базе. Если какая-нибудь операция, вызвав на экран другую запись, сотрет эти значения, то восстановить их будет невозможно.

В табличном состоянии записи создаются следующим образом. Прежде всего, необходимо создать в таблице несколько новых пустых строк командой

{Ф10} Таблица Структура Вставить Горизонталь

Образовавшиеся пустые строки нужно заполнить значениями полей и после этого выполнить операцию создания записи. Эта операция попросит указать те строки, в которых размещены новые записи. Таким образом, в табличном состоянии одной операцией можно создавать сразу несколько новых записей. Значения для полей этих записей можно порождать с помощью обычных табличных операций, доступных в этом режиме через табличное меню:

{Ф10} Таблица

Здесь, так же, как и в форматном состоянии, следует помнить о том, что до выполнения операции Создать введенные данные хранятся только в ячейках таблицы, но еще не в базе данных, и при неосторожных действиях они могут быть стерты с таблицы какими-нибудь другими записями.

Изменение записей

Вторая основная операция над записями – это изменение значений полей в уже существующих записях. Выполняется эта

операция аналогично операции создания записей путем изменения значений на экране и отправки их в базу командой

{Ф10} Запись Изменить

Отличие состоит в том, что если при создании новой записи было несущественно, какая запись была текущей, то операция изменения воздействует именно на текущую запись. Поэтому вы должны начать с того, чтобы получить в рамке-записи именно ту запись, которую хотите изменить, затем отредактировать некоторые поля в ней и затем выполнить команду Изменить.

Операцию изменения записей тоже можно выполнять как в форматном, так и в табличном состояниях.

Уничтожение записей

Те записи, которые перестали быть нужными, вы можете уничтожить. Для этого имеется команда

{Ф10} Запись Уничтожить

В форматном режиме эта операция без дополнительных вопросов уничтожает ту запись, которая находится в форматке, а в табличном режиме задается вопрос об указании тех строк, которых находятся удаляемые записи.

Последовательный перебор записей

Пользуясь описанными операциями, вы можете создать произвольное количество записей каждого типа. Записи при создании автоматически упорядочиваются всеми теми способами, которые предписаны для каждого типа записей в схеме базы данных. При работе с базой с рамкой-записью в каждый момент времени связывается какой-то один из порядков, определенных для данного типа записей. Первоначально таким порядком является главный порядок.

В соответствии с текущей упорядоченностью записей можно делать переходы от одной записи к другой в той последовательности, которая задается этим порядком. Для таких переходов служат клавиши {ПРЕД} и {СЛЕД}. Клавиша {ПРЕД} вызывает предыдущую по порядку запись, а клавиша {СЛЕД} – следующую. В табличном состоянии эти же клавиши осуществляют переходы не по одной записи, а сразу по несколько.

К работе с порядками записей имеют отношение и еще две клавиши: {НАЧАЛО} и {КОНЕЦ}, которые перемещают указатель

текущей записи соответственно на первую и за последнюю записи. Вторая из этих операций помимо позиционирования может использоваться еще для очистки полей. Поскольку в позиции за последней записью никакой записи не существует, то при попадании в эту позицию в рамку-запись выбирается пустая запись, т.е. происходит очистка всех полей.

Поиск записей

Основным действием, которое должно эффективно выполнять всякая база данных, является поиск записей по определенным значениям полей. В базе данных МАСТЕРа поисковые операции могут выполняться многими способами. Простейший из них осуществляется в диалоге следующим образом. Желая найти запись по значению определенного поля, вы должны ввести в него искомое значение и выполнить команду

{Ф10} Запись Найти

Если в схеме вашей базы данных был предусмотрен порядок, организованный по возрастанию или по убыванию этого поля, то поиск потребует не более секунды практически независимо от общего объема информации в базе. Если же порядка по этому полю не существует, то поиск может длиться сколь угодно долго, поскольку он будет сопряжен с последовательным перебором всех записей. Поэтому в таком случае вам будет задан вопрос о том, подтверждаете ли вы свое желание выполнить поиск.

Этот способ поиска записей одинаково работает как в форматном, так и в табличном состояниях. Отличие лишь в том, что если в форматном состоянии искомое значение нужно вводить в рамку-поле, то в табличном – в одну из ячеек столбца, соответствующего этому полю.

Ассоциативный поиск записей по соединениям

Второй способ поиска записей связан с использованием соединений, определенных в схеме базы данных. Предположим, что курсор находится в рамке-записи, связанной соединением с другим типом записей. Если поставить курсор в первой рамке на связующее поле и нажать клавишу {ОТМЕНА}, то во второй рамке появится запись, связанная с текущей записью.

Соединения типа "один к одному" и типа "один ко многим" обрабатываются этой операцией несколько по-разному. В результате обработки соединения типа "один к одному" во второй рамке-записи просто проводится поиск и найденная запись

ляется текущей. В результате обработки соединения "один ко многим" во второй рамке-записи устанавливается ограничение, в которое попадают только те записи, которые связаны с текущей записью этим соединением. При последовательном просмотре записей в рамке будут выбираться только эти записи. Когда этой рамке потребуется вернуться к полному (неограниченному) множеству записей, можно выполнить команду

{Ф10} Множество Полное

Ограничение и фильтрация записей

Третий способ поиска нужных записей – еще более мощный. Он состоит в возможности наложения составного ограничения на множество записей одновременно по разным полям. Для того чтобы ограничить множество записей по некоторому полю, нужно поставить курсор на это поле и выполнить команду

{Ф10} Множество Ограничить

При выполнении этой операции задаются два вопроса о граничных значениях. Введя эти два значения, вы выделите из всех записей те, у которых данное поле заключено внутри указанных границ.

Такое ограничение можно выполнять несколько раз по различным полям. В результате будет получено составное ограничение. Это ограничение будет сказываться на действии клавиш {ПРЕД} и {СЛЕД}. В последовательный перебор будут попадать лишь те клавиши, которые удовлетворяют составному ограничению.

Для того чтобы снять наложенное таким образом ограничение, т.е. вернуться к полному множеству, нужно выполнить ту же операцию, что и для снятия ограничения, получающегося после обработки соединения

{Ф10} Множество Полное

При наложении составного ограничения следует учитывать, что наиболее эффективно (во много раз быстрее) это ограничение работает в том случае, когда отдельные ограничения накладываются на поля последовательно по возрастанию их рангов в некотором из существующих порядков и когда все ограничивающие отрезки кроме, быть может, самого последнего являются вырожденными, т.е. с одинаковыми граничными значениями. Если эти условия не выполнены, то ограничение реализуется не структурными свойствами базы, а процедурными фильтрами, работающими по принципу селекции записей путем их последова-

тельного перебора. Ясно, что при этом получение очередной записи может оказаться довольно трудоемким делом. Это обстоятельство должно быть учтено еще при проектировании базы и составлении ее схемы на этапе определения порядков.

Переключение порядков

При наличии для одного типа записей нескольких порядков вам потребуется операция переключения с одного порядка на другой. Это нужно и для того, чтобы рассматривать записи в нужной последовательности, и для того, чтобы выбрать порядок, подходящий для наложения составного ограничения. Операция, с помощью которой можно выбрать порядок выполняется командой

{Ф10} Множество Упорядочить

Перед этим курсор должен быть поставлен на то поле, по которому требуется взять порядок.

6.6. Типовые структуры в базах данных

Рассмотрим некоторые типовые структуры, возникающие во многих базах данных. Овладев ими, вы сможете строить собственные базы данных путем комбинации таких структур.

Иерархическая связь записей

Одним из очень распространенных типов связей между записями является иерархическая связь, когда записи представляют узлы некоторого дерева, а соединение отражает принадлежность (вхождение) одних узлов к другим. В базе данных МАСТЕРа такую связь можно представить следующим образом. Снабдим каждую запись двумя полями одинакового типа: Код и Включение. Поле Код служит для однозначной идентификации записей узлов, а поле Включение в каждом узле представляет код другого узла, которому первый узел принадлежит. Помимо этих двух полей запись Узел может содержать произвольные информационные поля.

При определении схемы создадим два соединения, связывающих тип записи Узел с тем же самым типом Узел. Первое соединение типа "один ко многим":

Узел.Код ---->>> Узел.Включение
позволяет по заданному узлу определить все узлы, включенные в него.

Второе соединение типа "один к одному":
Узел.Включение ---> Узел.Код
позволяет по любому узлу выйти на тот узел, в который он включен.

Клавиши {ОТМЕНА} достаточно для движения по иерархии узлов. Действительно, находясь в любом узле и желая найти все подчиненные ему узлы, нужно поставить курсор на поле Код и нажать клавишу {ОТМЕНА}. В рамке-записи Узел будет установлено ограничение, выделяющее только те записи, у которых поле Включение равно полю Код исходной записи, т.е. записи, принадлежащие этому узлу. Эти записи можно перебирать нажатиями клавиш {СЛЕД} и {ПРЕД} и даже видеть одновременно несколько из них, если включено табличное состояние доступа. При желании, наоборот, перейти от некоторого узла к тому, в который он включен, нужно поставить курсор на поле Включение и нажать ту же клавишу {ОТМЕНА}.

Поиск записей по ключевым словам

Другой типичной для многих баз данных структурой является структура, в которой записи должны регистрироваться каждая по нескольким ключевым словам, так, чтобы для ее поиска достаточно было указания любого из связанных с ней слов. Для создания такой структуры нужно определить внутри записи несколько однородных полей, предназначенных для хранения ключевых слов, и создать порядок, организованный по смешению этих полей. Кроме того, требуется определить несколько соединений для ссылок ключевых полей на самих себя. Все эти соединения должны быть типа "один ко многим" и ссылаться из каждого ключевого поля на одно и то же первое ключевое поле. Тогда при нахождении в рамке-записи можно ввести ключевое слово в любое из ключевых полей и нажать клавишу {ОТМЕНА}. В результате ее срабатывания произойдет выборка всех тех записей, у которых в каком-нибудь из ключевых полей содержится то слово, которое было введено в текущее ключевое поле. Поскольку соединение имеет характер "один ко многим", будет установлено ограничение, отсекающее все неподходящие записи. Просмотр этих записей делается, как обычно, клавишами {ПРЕД} и {СЛЕД}.

Классификаторы

Классификаторы служат для экономии памяти при хранении символьной информации и для систематизации больших наборов символьных названий объектов. Простейший классификатор

представляется записью (назовем ее для определенности Классификатор), в которой имеются два поля: Код и Название. Первое поле задает числовой или короткий символьный код, а второе – строку нефиксированной длины, содержащую кодируемое название.

В схеме базы данных для этого типа записей должны быть заведены два порядка. Первый – по полю Код, второй – по полю Название. Тем самым будут созданы возможности по коду находить название и по названию находить код.

Использоваться этот классификатор должен в других типах записей, в которых требуется ссылка на название, но из-за многократных повторов одинаковых названий желательно вместо длинного названия использовать короткий код. Пусть этот второй тип записей называется Информация, а одно из полей внутри него – Объект. Это поле будет содержать код, определяемый согласно классификатору.

Следует установить два соединения в этой базе. Первое соединение

Информация.Объект ---> Классификатор.Код

позволит расшифровывать коды объектов. Действительно, получив очередную запись типа Информация, поставив курсор на за-кодированное поле Объект и нажав клавишу {ОТМЕНА}, вы сможете увидеть в рамке Классификатор полное название, соответствующее этому коду. Второе соединение

Классификатор.Код --->>> Информация.Объект

позволяет отыскивать информационные записи по названию объекта. Для этого нужно перейти в рамку Классификатор, ввести в поле Название строку, задающую искомое название, и выполнить команду

{Ф10} Запись Найти

В силу того что в базе имеется порядок по этому полю, запись будет найдена мгновенно. При этом в записи Классификатор будет получен код данного названия. Поставив курсор на поле Код и нажав клавишу {ОТМЕНА}, вы выделите среди информационных записей те, которые относятся к заданному объекту.

Естественно, что с помощью формул или макроклавиш этот интерфейс можно несколько упростить, но и в описанном виде он позволяет довольно неплохо справляться с классифицированными кодами.

ОСНОВЫ ИНСТРУМЕНТАЛЬНОГО ЯЗЫКА

ЧАСТЬ 2

ИНСТРУМЕНТАЛЬНЫЕ ВОЗМОЖНОСТИ

В первой части книги мы рассмотрели первый уровень интегрированной технологии. Этот уровень – самый внешний, он обращен напрямую к конечным пользователям компьютера. Во второй части мы переходим на второй, более глубокий уровень технологии – инструментальный.

Инструментальные возможности излагаются в этой части в четырех главах. В гл. 7 рассматривается общая архитектура языка программирования Мастер, его синтаксис, основные структурные и семантические понятия. Затем в гл. 8 описываются возможности обработки отдельных типов информации – чисел, строк, текстов, рисунков, таблиц, рамковых структур. В гл. 9 описываются встроенные средства управления базами данных, а в гл. 10 – способы построения типичных информационных структур в базе данных и технология построения прикладных информационных систем.

7.1. Средо-ориентированный характер языка

Если оценивать инструментальный язык системы МАСТЕР с позиций традиционного процедурного программирования, то в нем будут найдены почти все черты, присущие обычным языкам: процедуры и функции, управляющие конструкции циклов и условных операторов, переменные и присваивания, структуры данных. Все это – основа функциональной полноты языка, но определяющим его свойством является все же нечто другое – его средо-ориентированный характер. Термин этот характеризует новую инструментальную технологию, возникшую совсем недавно и продолжающую быстро развиваться с распространением интегрированных систем. Суть этой технологии состоит в том, что на первое место в инструментарии ставится не язык, а интерактивная среда – экранная среда текстового редактора, электронной таблицы, графики, структурная комбинация таких сред. Вполне естественно, что такая среда выполняет функции интерактивного характера, обеспечивая взаимодействие пользователя с компьютером. Но ее роль намного существеннее: именно эта среда (а не управляющая программа, как в обычном программировании) выполняет общеорганизующую функцию в прикладной системе. Язык программирования играет второстепенную роль. С его помощью устанавливаются различные функциональные связи элементов среды, создаются дополнительные диалоговые возможности, и лишь в более сложных случаях язык начинает работать в обычном процедурно-ориентированном смысле с привлечением соответствующей процедурной технологии.

Начала программирования на Мастере

В общих чертах вы уже успели познакомиться с языком Мастер: формулы в электронных таблицах – это простейшие программы на нем. Но формулы можно вставлять не только в ячейки таблиц. Находясь в рамке произвольного типа, вы можете связать формулу с любой функциональной клавишей и нажатия этой клавиши будут выполнять данную формулу. Для такого связывания формул с клавишами служит специальная клавиша – {ДОП-Ф10}. При ее нажатии МАСТЕР задает сначала вопрос о том, с какой клавишей нужно связать формулу, а затем просит ввести эту формулу.

Простейшим экспериментом может быть такой. Нажмите клавишу {ДОП-Ф10}. На появившийся запрос:

Нажмите функциональную клавишу,
для которой определяется локальное действие
ответьте нажатием некоторой функциональной клавиши, скажем
{F4}. На следующий далее вопрос:

Введите формулу локального действия
введите следующую формулу:

```
Звук(1000,10,2000,10)
```

В этой формуле задано обращение к функции Звук(), а ее аргументы задают периоды и длительности звуков. После этого клавиша {F4} приобретет новое значение: ее нажатие будет вызывать звуковой сигнал.

Рассмотрим чуть более сложный пример. Связем с клавишей {F4} тем же самым способом небольшое меню, состоящее из двух пунктов, называющихся до и соль, имеющих пояснительные комментарии "нота до" и "нота соль" и исполняющих эти ноты. Формула для этого меню будет такой:

```
Меню(  
    "ДО", "Нота ДО", Звук(9122,100),  
    "СОЛЬ", "Нота СОЛЬ", Звук(6088,100)  
)
```

Теперь нажатие клавиши {F4} будет формировать меню во второй строке экрана и предложит выбрать курсором одну из двух нот.

Определять формулы для клавиш описанным способом удобно лишь до тех пор, пока формулы остаются короткими и умещаются в одну строку. Но в одной строке можно изображать лишь самые

156

простейшие алгоритмические действия. Даже при программировании рассмотренного нотного меню места в одной строке хватило только на две ноты. Если Вы хотите запрограммировать более сложное действие, то нужно создать для текста формулы отдельную рамку. Эта рамка должна быть функционального типа, и создавать ее следует командой

{Ф10} Рамка Создать Функция

Попробуйте создать такую рамку, назовите ее "гамма", войдите в нее и создайте в ней текст следующей программы:

Гамма

Меню(

```
"ДО", "Нота ДО", Звук(9122,100),  
"РЕ", "Нота РЕ", Звук(8127,100),  
"МИ", "Нота МИ", Звук(7240,100),  
"ФА", "Нота ФА", Звук(6834,100),  
"СОЛЬ", "Нота СОЛЬ", Звук(6088,100),  
"ЛЯ", "Нота ЛЯ", Звук(5424,100),  
"СИ", "Нота СИ", Звук(4832,100)
```

)

Постарайтесь абсолютно точно воспроизвести все символы приведенного текста, а также разбивку его на строки. Пока вы не знаете точного синтаксиса языка, легко сделать ошибку.

После написания текста нужно выйти из рамки. При этом, если во введенном тексте нет синтаксических ошибок, то рамка автоматически закроется, а в противном случае останется открытой, причем в текст будет вставлено сообщение об ошибке отдельной строкой над предполагаемым местом ошибки.

Создав первую программу на языке Мастер, названную "гамма", вы расширили язык программирования Мастер. Для того чтобы вызвать эту функцию, применим тот же прием, с которого мы начали: связем (через клавишу {ДОП-Ф10}) обращение к этой функции с какой-нибудь клавишей, скажем с той же {F4}. Формула такого обращения должна иметь вид:

```
гамма()
```

После этого каждое нажатие клавиши {F4} внутри этой рамки будет приводить к появлению меню с семью нотами.

Обратите внимание на то, что, говоря о запрограммированном действии клавиши {F4}, мы подчеркиваем, что оно будет иметь место только внутри данной рамки – той, в которой мы находились, нажимая клавишу {ДОП-Ф10}. Такой способ связы-

вания формул с клавишами называется локальным, а формулы связываемые с клавишами через {ДОП-Ф10}, называются **локальными действиями** клавиш. Если перейти в любую другую рамку как вверх, так и вниз по иерархии, то эта клавиша уже не будет связана с заданным действием.

Можно запрограммировать функциональную клавишу и по-другому – глобально. Для этого нужно создать функциональную рамку, но назвать ее не произвольным идентификатором (как "гамма"), а именем функциональной клавиши, включая фигурные скобки, например "{Ф6}". В рассматриваемом примере можно переименовать рамку, сменив ее имя "гамма" на "{Ф6}". При этом уже не требуется связывать обращение к этой функции с клавишей через {ДОП-Ф10}. Клавиша {Ф6}, совпадающая с именем функциональной рамки, вызывает эту функцию автоматически, причем связь между клавишей и функцией будет действовать не только внутри одной рамки, но и на всех более глубоких уровнях рамковой иерархии: запрограммированная таким образом формула является **глобальным действием** клавиши.

Приведенный пример показывает в миниатюре технологический процесс программирования на языке Мастер, который основывается на трех возможностях:

- создания с помощью рамок нужной информационной среды, служащей основой разрабатываемой прикладной системы;
- определения действий клавиш формулами (локальным и глобальным образом), позволяющего развивать интерактивные свойства среды;
- создания новых инструментальных возможностей в виде функциональных рамок, определяющих новые функции в языке.

7.2. Типы данных

Встроенные функции языка Мастер объединяются в группы, ориентированные на те или иные **типы значений**. Значения в системе МАСТЕР могут быть следующих типов: числа, строки, тексты, формулы, рисунки, указатели диапазонов электронных таблиц, указатели рамок, указатели записей базы данных. Описание языка начнем с краткой характеристики каждого из этих типов значений.

Числа. Для представления и обработки количественной информации используются значения числового типа, предназначенные прежде всего для количественных вычислений. Кроме того, они используются как логические значения (0 – ложь, 1 – истина), как кодовые значения (для обозначения форматов, типов и т.п.), а также для представления времени суток и дат.

Строки. Значения строкового типа представляют собой цепочки символов, не содержащих перевода строки. Цепочки эти, хотя и могут быть достаточно длинными, все же ограничены – каждая из них содержит не более 255 символов. Символы, входящие в строки, берутся из стандартной кодовой таблицы, включающей цифры, знаки препинания, русские и латинские буквы, причем буквы могут быть как строчными, так и прописными. К символам относятся еще и знаки псевдографики, т.е. отрезки линий, с помощью которых можно изображать несложные прямоугольные рисунки. Над строками определены такие операции, как слияние, вырезка подстроки, сравнения и др.

Тексты. Значения текстового типа так же, как и строки, являются символьными. Отличие текстов от строк заключается в том, что они могут состоять из нескольких строк. На количество строк в тексте не накладывается никаких ограничений. Над текстами определены дополнительные функции по сравнению со строками, которые позволяют перебирать отдельные строки текста, вставлять и уничтожать строки, выполнять контекстный поиск и замену, а также делать лексический разбор.

Формулы. Одной из важнейших особенностей языка Мастер является то, что формулы, являющиеся программами этого языка, сами по себе считаются обрабатываемыми значениями этого же языка. Таким образом, формулы, которые выполняют обработку данных, сами по себе могут служить объектом программной обработки. Вы можете, например, написать программу, которая будет автоматически заполнять формулами электронную таблицу или будет создавать формульные рамки и тут же обращаться к ним. Эта возможность чрезвычайно повышает гибкость программирования. При диалоговом редактировании и при программной обработке формулы рассматриваются как обычные тексты со всеми присущими текстам операциями. Однако после обработки каждая формула подвергается компиляции и превращается в значение особого типа – формульного. Рамка, содержащая такое значение, является определяемой функцией, расширяющей язык Мастер.

Рисунки. Значения рисункового типа предназначены для представления графической информации. Как и все прочие значения, рисунки можно вычислять, передавать в качестве аргументов, присваивать рамкам, ячейкам таблиц, переменным, полям базы данных. При присваивании рисунка рамке или ячейке он становится виден на экране. Для процедурной обработки рисунков в языке имеется несколько встроенных функций, которые могут формировать рисунки, объединять их.

Указатели диапазонов и рамок. В МАСТЕРе мы находимся в сложной иерархической среде вложенных рамок различных типов. Обрабатываемая информация некоторым образом распределена

на по этой иерархической структуре. Инструментальный язык был бы беспомощным в этой среде, если бы не имел в своем арсенале средств обработки самой этой структуры. Для этого имеются два типа указателей: указатели диапазонов таблиц и указатели рамок. Значения этих типов сами по себе не имеют информационного содержания, как, скажем, числа, строки, тексты или рисунки. Эти значения дают доступ к тем или иным объектам информационной среды. В языке имеется много функций, позволяющих совершать над указателями разнообразные структурные действия. Указатели рамок используются для создания, уничтожения рамок, передвижения по ним, открытия, закрытия, изменения формы, содержимого и других атрибутов рамок. Указатели диапазонов дают доступ к ячейкам и диапазонам электронных таблиц, позволяют извлекать и присваивать значения ячеек, вставлять и уничтожать строки и столбцы таблиц, изменять размеры, форматы и разграфку ячеек.

Указатели записей базы данных. Работа с базой данных в языке Мастер ведется с помощью значений специального типа, называемых указателями записей. Это понятие включает в себя одновременно все те аспекты, которые требуются для обработки записей: и возможность доступа к их полям, и упорядоченность множества записей по тому или иному полю, и группированием записей в определенные наборы (подмножества), и возможность их последовательного перебора. Каждый из этих аспектов реализуется соответствующими встроенными функциями, применимыми к значению типа "указатель множества записей". Набор встроенных функций, определенных над указателями записей, один из самых больших в Мастере. Эти функции составляют функционально и технологически полный набор операций над базами данных процедурно-реляционного типа.

Обозначения типов в языке

При программировании на Мастере иногда требуется как-то обозначать *сами типы значений*. Например, это может быть нужно для распознавания типа той или иной рамки. Типы обозначаются числовыми кодами, имеющими для удобства мнемонические наименования:

ЧИСЛО, СТРОКА, ТЕКСТ, ФОРМУЛА, РИСУНОК,
ДИАПАЗОН, РАМКА, МНОЖЕСТВО,
ОШИБКА, ПУСТО

Два последних типа – ОШИБКА и ПУСТО – являются вырожденными, единственные значения которых – соответственно результат ошибочного вычисления и пустое, неопределенное значение.

7.3. Синтаксис языка

Синтаксис языка Мастер является *функциональным*, т.е. основывается всего лишь на двух понятиях: обращение к функции и обращение к операции.

Обращение к функции, подобно обычной математической символике, имеет вид:

$$F(x, y, z)$$

где F – имя вызываемой функции, а x, y, z – аргументы обращения. Имя должно быть идентификатором, а аргументы – произвольными формулами, быть может, очень сложными, состоящими в свою очередь, из обращений к функциям и операциям.

Обращение к операции может быть бинарным или унарным. Обращение к бинарной операции имеет вид:

$$x * y$$

обращение к унарной операции:

$$- x$$

где "*" и "-" – знаки операций, а x и y – аргументы операции, которые так же, как и в обращении к функции, могут быть формулами произвольной сложности. Для управления порядком вычисления операций в формулах можно использовать скобки.

При написании формул их можно располагать произвольным образом либо на одной, либо на нескольких строках. Например, формулу, описывающую цикл из десяти итераций

```
Присв(счетч,1); Повтор(счетч<=10, Присв(счетч, счетч+1))
```

можно записать и на нескольких строках:

```
Присв(счетч,1);
Повтор(счетч<=10,
    Присв(счетч, счетч+1)
)
```

Возвращаемые значения и побочные действия формул

Многие функции языка Мастер, подобно математическим функциям $\text{Sin}()$, $\text{Cos}()$, $\text{Мин}()$ или $\text{Макс}()$, при обращении к ним возвращают *возвращаемые значения*. Эти возвращаемые значения мо-

гут служить аргументами для других функций или операций. Например, в формуле

`Мин(Макс(A1:A100), Макс(B1:B100))`

оба обращения к функции `Макс()` возвращают некоторые значения, которые используются в качестве аргументов в обращении к функции `Мин()`.

Другие функции возвращаемыми значениями не обладают, но зато производят какие-то изменения в информационной среде: изменяют значения рамок или ячеек таблиц, осуществляют какие-то диалоговые эффекты. Такие изменения называются **побочными действиями** функций. Самым типичным представителем функций с побочным действием является функция `Присв()`, изменяющая значение рамки или ячейки.

Многие функции обладают как возвращаемым значением, так и побочным действием. Так, например, побочным действием обращения к функции `Ширина(A1, 20)` является изменение ширины столбца A некоторой таблицы. В то же время ее возвращаемым значением является указатель ячейки A1, которым можно воспользоваться в какой-нибудь другой операции над диапазонами. Например, оформляя таблицу, можно выполнить следующее обращение:

`Высота(Ширина(A1, 20), 3)`

В этой формуле обращение к функции `Ширина()` изменяет ширину столбца A и возвращает указатель на ту же ячейку A1, которая служила ее аргументом. Получив этот указатель, функция `Высота()` изменяет высоту первой строки той же таблицы.

Многие функции не имеют возвращаемого значения, а только побочное действие. Формально считается, что и такие функции вырабатывают возвращаемое значение, но оно имеет специальный тип **пусто**, которым обозначается отсутствие значения.

7.4. Управляющие структуры

Во всяком языке программирования **управляющими структурами** называются такие операторы или синтаксические конструкции, с помощью которых организуется общая структура алгоритмов. Для языков программирования процедурного типа, к которым относится и язык Мастер, такими управляющими структурами являются: последовательное вычисление операторов, условное разветвление, циклическое выполнение действий, вызовы определяемых пользователем функций. Все такие управляющие структуры присутствуют и в языке Мастер. Синтаксически все

они оформляются в функциональном стиле. Это значит, что все управляющие структуры представляются внешне либо как обращения к операциям, либо как обращения к функциям.

Из всего набора встроенных функций языка Мастер выделяются несколько функций особого рода, называемых **управляющими функциями**. Это функции `Если()`, `Выбор()`, `Меню()`, `Повтор()`, `Блок()`, `Ловушка()`, `Ошибка()`. Синтаксически обращения к этим функциям выглядят так же, как и обращения к любым другим функциям. Существенное отличие от обычных функций состоит в том, что аргументы управляющих функций вычисляются в нестандартном порядке. Если для обычных функций аргументы вычисляются в том порядке, в котором они записаны в списке, то управляющие функции предназначены именно для того, чтобы позволить вычислять аргументы в нестандартном порядке.

В остальных отношениях управляющие функции подобны обычным функциям. Они точно так же, как любые другие функции, обладают возвращаемыми значениями и побочными действиями. Обращения к этим функциям можно вкладывать как друг в друга, так и в обращения к любым другим функциям.

Последовательное вычисление формул

При описании сложных действий, как правило, требуется применять несколько функций, вызываемых одна за другую. Для этой цели обращения к функциям записываются в нужной последовательности, а между ними ставится разделяющий знак ";" (точка с запятой). Так формула

`Ф1 ; Ф2 ; Ф3 ; . . . Фk`

означает последовательное вычисление сначала формулы `Ф1`, затем формулы `Ф2` и т.д. до формулы `Фk`. Возвращаемым значением всей этой составной формулы является то значение, которое вычисляется последней формулой в цепочке – `Фk`. Возвращаемые значения всех предыдущих формул в последовательности игнорируются, т.е. они имеют смысл только из-за совершения в них какого-то побочного действия – присваивания значений рамкам, ячейкам или переменным, поиска или создания записей в базе данных, осуществления диалога с пользователем и т.п.

Так, например, в ячейку таблицы можно вставить следующую формулу:

```
Звук(); Сообщить('вычисление завершено', 0); А1+Б1
```

которая состоит из трех последовательных формул. Первая из них – обращение к функции Звук() – выдает звуковой сигнал, вторая – подформула Сообщить('вычисление завершено', 0) – выдает в поле сообщения символьную строку, а последняя формула – А1+Б1 – вычисляет сумму ячеек А1 и Б1. Результат всей последовательности – значение этой последней формулы. Таким образом, подформулы эквивалентны по своему возвращаемому значению вся составная формула эквивалентна более короткой формуле А1+Б1, но имеет в отличие от нее побочное действие, состоящее в выдаче звукового сигнала и сообщения.

Комментарии в Мастер-Формулах

В последовательность формул можно вставлять строки-комментарии, отделяя их тем же знаком ";". Вычисление таких подформул-строк не оказывает влияния на конечный результат всей формулы, если они не находятся в самом конце последовательной цепочки, а для программиста строки-комментарии будут служить пояснениями. Таким образом, формулы

ф1 ; "строка-комментарий" ; ф2
и
ф1 ; ф2

эквивалентны по своему результату. Однако следует иметь в виду, что формулы

ф1 ; ф2 ; "строка-комментарий"
и
ф1 ; ф2

обладают разными возвращаемыми значениями: возвращаемым значением первой формулы служит строка-комментарий, а второй – результат вычисления формулы ф2.

Условные вычисления

Встроенные функции условных вычислений позволяют в процессе выполнения осуществлять выбор между альтернативными вариантами действий. Для этой цели в языке Мастер существуют три управляющие функции: Если(), Выбор() и Меню().

Функция Если() выбирает один из двух вариантов действий по логическому условию. Она имеет одну из следующих двух форм обращения:

Если (логическое условие,
действие1
,

действие2
)

или

Если (логическое условие,
действие1
)

Первым аргументом функции является **логическое условие**, с вычисления которого начинается работа функции. Если это условие истинно, то выполняется **действие1**. В противном случае выполняется **действие2** (если оно есть). Возвращаемым значением функции Если() является значение, возвращенное выбранным действием.

В качестве логических условий можно записывать произвольные числовые формулы. Числовое значение 0 рассматривается как "ложь", а значение 1 – как "истина". Обычно логические условия строятся с помощью операций отношения (равно, не равно, меньше, больше) и логических связок (логические и, или, не).

Действия, выбираемые функцией Если(), могут быть сложными и требовать для своей записи нескольких последовательных формул. Нужно соединять эти формулы знаком ";" (точка с запятой), как это описано в предыдущем разделе о последовательных вычислениях. Так, например, формула

```
Если(A1<0,  
Звук();  
Сообщить("Отрицательное значение!");  
0  
'Sqrт(A1)  
)
```

содержит в качестве первого действия не одну формулу, а последовательность из трех формул, разделенных между собой точкой с запятой. Эта формула может использоваться в электронной таблице для предотвращения извлечения квадратного корня из отрицательного числа. Если в ячейке А1 оказывается отрицательное число, то эта формула возвращает значение 0, издав предварительно звук и выдав сообщение об ошибке. Если число в ячейке А1 неотрицательно, то формула возвращает значение квадратного корня из этого числа.

Другая функция условного вычисления – **Выбор()** – обеспечивает выбор одного действия из списка альтернативных действий. Первый аргумент этой функции – целое число – задает номер выбираемого действия, а сами действия представлены следующими далее формулами. Форма обращения к этой функции такова:

```
Выбор (номер,  
действие1,  
действие2,  
действие3,  
...  
)
```

Сначала вычисляется первый аргумент функции, который должен выработать целое число. По значению этого числа выбирается и выполняется одно из действий. Возвращаемым значением обращения к функции **Выбор()** является значение выбранного действия.

Последняя функция, выполняющая выбор альтернативных действий, – **Меню()**. Эта функция выбирает действие не сама, а по указанию пользователя через меню, высвечиваемое в поле значения. Подробно эта функция объясняется в следующей главе.

Циклические вычисления

Для организации циклического повторения некоторого действия предназначена функция **Повтор()**, имеющая одну из трех форм обращения:

Повтор (условие1)

или

**Повтор (условие1,
действие
)**

или

**Повтор (условие1,
действие
условие2
)**

Выполняется эта функция циклически. На каждой итерации сначала вычисляется **условие1**, которое должно выработать чис-

ловое логическое значение (0 означает "ложь", 1 – "истина"). Если значение оказывается истинным, то выполняется **действие**, заданное вторым аргументом. После этого вычисляется **условие2**, если оно есть. "Ложное" (нулевое) значение любого из условий прекращает работу цикла. До тех пор, пока логические условия оказываются истинными, вычисления повторяются.

Возможность постановки логических условий перед выполнением действия и после его завершения упрощает программирование, поскольку в алгоритмах требуются циклы с проверкой условия как в начале, так и в конце цикла.

Возвращаемым значением функции **Повтор()** является количество сделанных итераций.

В теле цикла, как правило, требуется записать несколько последовательно выполняемых формул. Для их объединения используется, как обычно, знак ";".

Примером использования цикла является следующая формула, генерирующая последовательные обращения к функции **Звук()** с возрастающим периодом (т.е. издающая нисходящее звучание):

```
нисходящийЗвук—  
Присв(тон,100); "присвоили в рамку 'тон' число 100";  
Повтор ( "начало цикла";  
    тон<=5000, "пока период не превзошел 5000";  
    звук(тон,100); "издать звук длительностью 1 сек";  
    Присв(тон,тон+50); "увеличить период"  
    ; "конец цикла"
```

7.5. Присваивание значений

Программирование на языке Мастер предназначено для обработки процедурным способом числовых, строковых и прочих значений. Основой процедурного программирования является возможность присвоить вычисленное значение тому или иному хранилищу, чтобы позже воспользоваться им в других операциях. Таким образом, операция присваивания значений является одной из основных в семантике процедурного языка. В языке Мастер присваивание значений выполняется с помощью функции **Присв()**, имеющей следующий формат обращения:

Присв(хранилище, значение)

Значение, вычисленное вторым аргументом, присваивается хранилищу, указанному первым аргументом. **Хранилищами значений** могут быть: рамки, ячейки электронных таблиц, локальные переменные формул. Рассмотрим особенности каждого из этих трех видов хранилищ.

Присваивание значений рамкам

Для присваивания значения рамке нужно, чтобы первым аргументом функции Присв() был указатель рамки. Этот указатель можно задать либо просто именем рамки, либо обращением к какой-нибудь из функций над рамками. Например, в формуле

```
Присв(счетчик, счетчик+1)
```

рамка-хранилище, которой делается присваивание, задается именем "счетчик". В формуле

```
Присв(ТекРамка(), ТекРамка()+1)
```

рамка-хранилище задается обращением к функции ТекРамка(), возвращающей указатель на текущую рамку.

Функцией Присв() в рамку можно присвоить значение любого из следующих типов: число, строку, текст или рисунок. Примером присваивания в рамку текстового значения может служить следующая программа, реализующая диалоговый калькулятор. Эта программа создает текст некоторой формулы (запрашивая одну из ее строк у пользователя) и присваивает его в рамку "выражение". Затем компилирует этот текст, превращая рамку из текстовой в функциональную, и обращается к функции для вычисления.

```
Присв(выражение,
    Текст(
        "Врамке(СхвРамка(),",
        ЗапросСтр("Введите формулу"),
        ")"
    );
    ОпФунк(выражение);
    Сообщить("Результат=" && Вып(выражение))
```

Присваивания значений ячейкам таблиц

С помощью функции Присв() можно изменять значения отдельных ячеек электронных таблиц. Для этого в качестве хранилища должен быть задан указатель ячейки. Указатель ячейки задаст-

ся своим буквенно-числовым именем с абсолютной или относительной адресацией, например \$13 или !Г!14. Кроме того, указатель ячейки можно задать обращением к любой из функций, возвращающих указатели ячеек. Например, функция Яч() позволяет возвращать указатель ячейки, определяемой *вычисляемыми* номерами строки и столбца.

Примером, использующим присваивания в ячейки таблиц, может служить программа, которая выполняет транспонирование матрицы размером 10x10, располагающейся в электронной таблице:

```
Присв (строка, 1); "счетчик строк";
Повтор (строка <= 10,
    Присв (столбец, 1); "счетчик столбцов";
    Повтор (столбец < строка,
        Присв (служЗнач, Яч (строка, столбец));
        Присв (Яч(строка, столбец), Яч(столбец, строка));
        Присв (Яч (столбец, строка), служЗнач);
        Присв (столбец, столбец + 1)
    );
    Присв (строка, строка + 1)
)
```

В этой формуле используются три вспомогательные переменные – простые рамки "строка", "столбец", служащие счетчиками строк и столбцов, а также "служЗнач", используемая для сохранения значения ячейки.

Присваивания локальным переменным формул

Для удобства программирования бывает необходимо заводить временные хранилища для локальных потребностей той или иной формулы, которые бы автоматически исчезали вместе с завершением выполнения формулы, а не существовали постоянно как рамки и ячейки электронных таблиц. Такие временные локальные хранилища называются **локальными переменными** формул. Для создания локальных переменных служит специальная функция Блок(), имеющая два аргумента:

Блок (число локальных переменных, формула)

Эта функция создает блок локальных переменных, количество которых задается первым аргументом. Созданные переменные существуют только во время вычисления формулы, заданной вторым аргументом, и могут использоваться в этой формуле для временного хранения различных значений. По завершении работы формулы весь блок созданных переменных уничтожается.

Переменные нумеруются числами начиная с единицы, и для работы с их значениями используются две функции: Присв() и Пер(). Обращение к функции Пер(i) возвращает значение, содержащееся в переменной с номером i. Для присваивания значения в i-ю переменную используется обращение к Присв(), в котором первый аргумент должен быть числом, равным i. Рассмотрим для примера следующую формулу:

```
Блок(2,
    Присв(1,ЗапросЧис("введите период сигнала"));
    Присв(2,ЗапросЧис("введите длительность"));
    Звук(Пер(1),Пер(2),Пер(1),Пер(2),Пер(1),Пер(2))
)
```

В этой формуле с помощью функции Блок() вводятся две локальные переменные. Затем этим переменным присваиваются два числовых значения, запрошенные у пользователя через обращения к функции ЗапросЧис(). После этого значения переменных многократно используются в обращении к функции Звук() для генерации звуковых сигналов.

Обратите внимание на то, что при присваивании локальной переменной не следует писать Присв(Пер(5),...) вместо Присв(5,...), поскольку первая формула означает присваивание не в пятую переменную, а в переменную, номер которой находится в пятой переменной!

Кратные присваивания

Функция Присв(), как и всякая другая в языке Мастер, обладает возвращаемым значением, которым является присвоенное значение. Это можно использовать для совмещения присваиваний с какими-то другими вычислениями или для выполнения кратных присваиваний – т.е. присваиваний одного и того же значения в несколько разных хранилищ. Так, например, вместо последовательности формул

```
Присв (A1, 100);
Присв (A2, 100);
Присв (A3, 100)
```

можно использовать более компактную формулу:

```
Присв (A1, Присв (A2, Присв (A3, 100)))
```

в которой результат одного присваивания служит значением, используемым во втором, а затем и в третьем присваивании.

7.6. Определение собственных функций

Функциональные рамки

Определяемые функции представляются в МАСТЕРе в виде рамок функционального типа, которые создаются командой

```
{Ф10} Рамка Создать Функция
```

Войдя в такую рамку, вы можете с помощью обычного текстового процессора МАСТЕРа ввести текст формулы. Тем самым будет создана новая функция, доступная для вызова в программах на Мастере. Такие функции называются **определяемыми функциями** в отличие от **встроенных функций**, присутствующих в языке изначально.

Обращение к определяемой функции выглядит синтаксически точно так же, как и ко встроенной, и состоит из идентификатора функции (которым в данном случае является имя функциональной рамки) и списка формул-аргументов, следующих друг за другом через запятую и заключенных в круглые скобки.

Возвращаемое значение определяемой функции

Обращения к определяемым функциям, равно как и ко встроенным, должны формировать возвращаемые значения, а также обладать побочными действиями. Побочное действие определяемой функции есть совокупность всех тех побочных действий, которые совершены выполненными в ней обращениями к другим функциям. Возвращаемым значением определяемой функции является значение, выданное формулой этой функции. Поскольку функции, как правило, задаются последовательным выполнением нескольких обращений, то возвращаемым значением функции является значение последней формулы в этой последовательности. Рассмотрим, например, функцию, вычисляющую сумму значений полей в наборе записей базы данных:

```
суммарныйОклад
Блок(1,
    Присв(1,0);                                     "обнулить счетчик";
    Повтор(ВзятьЗапись(сотрудник),
        Присв(1,Пер(1)+сотрудник.оклад); "в счетчик";
        НайтиСлед(сотрудник);           "к следующей записи"
    );
    "результатом будет накопленная сумма:";
    Пер(1)
)
```

В данной функции суммарное значение полей накапливается в локальной переменной. Последним в ряду формул стоит обращение к значению переменной $\text{Пер}(1)$. Это обращение не используется ни в какой операции или функции, а нужно только для того, чтобы сформировать возвращаемое значение функции.

Кроме того, можно формировать возвращаемое значение формулы функции и завершать работу этой формулы, не достигая ее конца. Для этого служит функция Результат(), форма обращения к которой такова:

Результат (возвращаемое значение)

Аргументы определяемых функций

Любой определяемой функции требуется передача некоторых значений, над которыми она будет работать. Некоторые значения могут быть взяты функцией из глобальных объектов – из рамок, записей базы данных, от пользователя в диалоге. Другие значения должны быть переданы функции из точки ее вызова через аргументы. В обращении к функции аргументы представляются формулами, перечисляемыми в круглых скобках через запятую, а внутри функции эти аргументы доступны с помощью специальной функции Арг(), имеющей следующий формат обращения:

Арг(номерАргумента)

В качестве примера определяемой функции можно привести формулу, вычисляющую факториал целого числа, т.е. произведение всех чисел от 1 до заданного числа включительно. Число, для которого функция будет вычислять факториал, задается ее аргументом и доступно в формуле через АРГ(1).

```
Факториал
Блок(2, "переменные для счетчика и результата";
Присв(1,1); "в счетчик единицу";
Присв(2,1); "в результат единицу";
Повтор(Пер(1)<=Арг(1), "условие продолжения";
    Присв(2,Пер(2)*Пер(1)); "помножим на счетчик";
    Присв(1,Пер(1)+1); "увеличим счетчик на 1"
    );
    "результатом функции будет переменная 2";
Пер(2)
)
```

Эту же функцию можно определить через нее саму, пользуясь тем, что факториал числа N равен факториалу числа $(N-1)$, умноженному на число N . Такое определение называется рекурсивным:

172

```
факториал—  
Если(Arg(1)=1,  
      1,  
      Arg(1)*факториал(Arg(1)-1)  
)
```

Здесь слово "факториал" – имя определяемой функции, которая таким образом обращается сама к себе.

К определяемой функции можно обращаться с различным числом аргументов. Вызываемая функция должна иметь возможность определять, сколько фактических аргументов ей было передано. Для этого существует встроенная функция ЧисАрг().

Примером ее использования является следующая программа, вычисляющая суммарную длину строк, являющихся ее аргументами. Количество аргументов этой функции не фиксировано и определяется с помощью обращения к ЧисАрг().

```
суммарная длина—  
Блок(2, "счетчик аргументов и счетчик длины";  
    Присв(1,1); "в счетчик аргументов присвоим 1";  
    Присв(2,0); "в счетчик длины присвоим 0";  
    Повтор(Пер(1)<=ЧисАрг(),  
        Присв(2,Пер(2)+Длина(Арг(Пер(1))));  
        Присв(1,Пер(1)+1)  
    );  
    "результат функции - переменная 2";  
    Пер(2)
```

В данной функции следует обратить внимание на обращение к функции Арг(Пер(1)), в котором номер аргумента является вычисляемым – он берется из первой локальной переменной.

Структурирование программ на Мастере

Многие функции для реализации сложных алгоритмов нуждаются во вспомогательных информационных структурах, переменных, функциях. Если рамки для всех этих вспомогательных объектов располагать на том же уровне, что и рамку основной функции, то в большой программной системе может возникнуть хаос.

Общепринятые методы иерархического структурирования программ представляются в МАСТЕРе с помощью иерархии рамок. А именно, если функция оказывается слишком сложной и требует вспомогательных объектов, то ее следует представлять не функциональной рамкой, а составной. Внутри этой составной рамки нужно создать функциональную рамку, определяющую ос-

новной алгоритм программы, а также все те вспомогательные информационные и функциональные рамки, которые потребуются. Имя составной рамки должно совпадать с именем основной функциональной рамки. Тогда этим именем можно будет пользоваться для вызова функции, т.е. составная рамка снаружи будет выглядеть так, как если бы она была функциональной.

Этот метод обладает еще и тем достоинством, что локальные объекты, созданные внутри составной рамки, могут именоваться из текста функции своим локальным именем без указания имен рамок по всей цепочке от корневой рамки до требуемой.

Схематически типичная структура сложной функции представлена на рис. 7.1. Составная и находящаяся внутри нее основная функциональная рамки имеют одинаковые имена Функ. Помимо основной функциональной рамки Функ в составной рамке находятся вспомогательные рамки Функ1, Функ2, Функ3, Функ4, вызов которых из текста основной функции Функ (так же, как и друг из друга) осуществляется с помощью простых имен. Обращение ко всей этой составной функции также возможно с помощью простого имени: Функ().

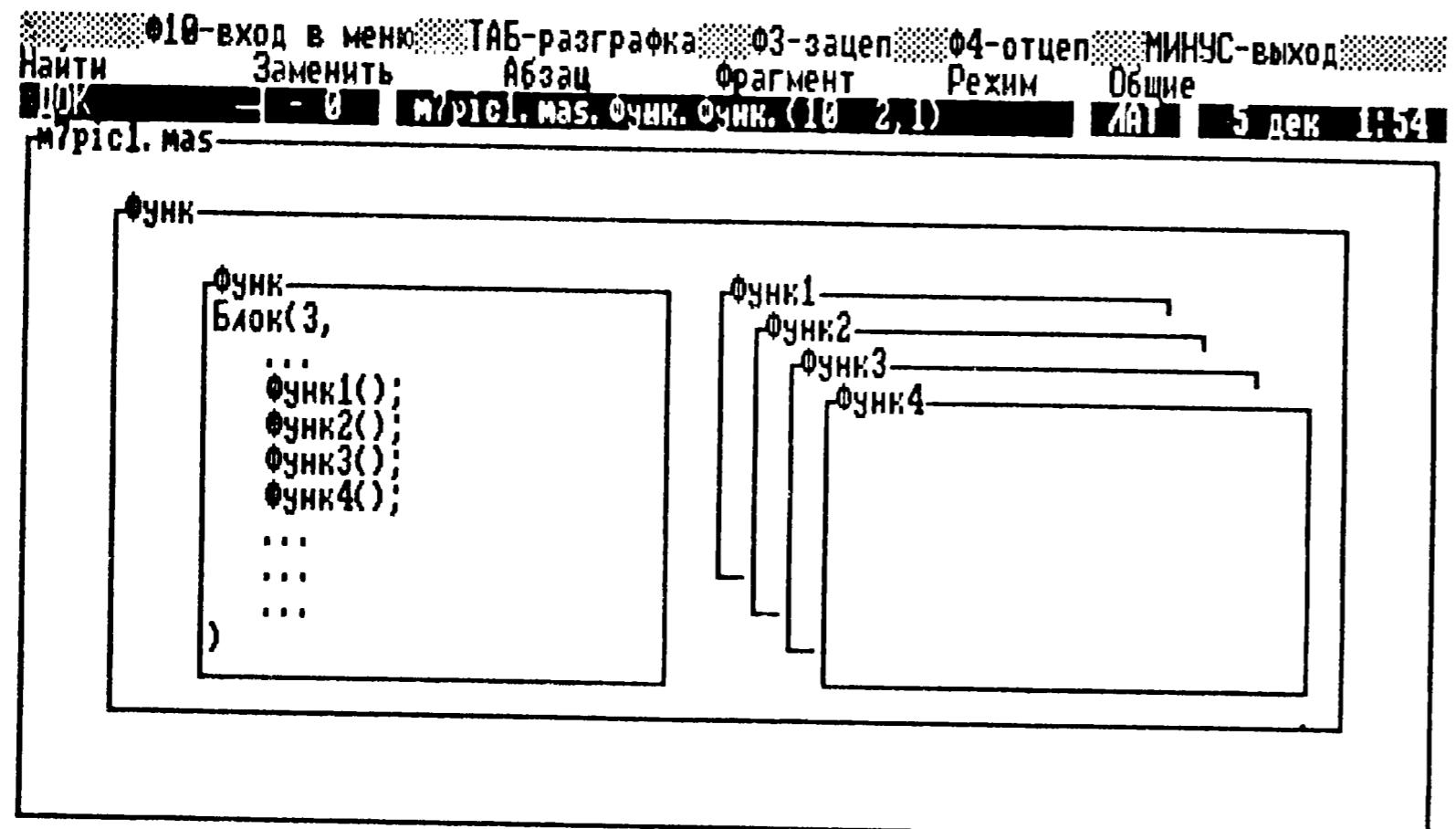


Рис. 7.1. Рамковая структура сложной функции

7.7. Именование рамок в формулах

Всякая прикладная система, разрабатываемая в МАСТЕРе, будет состоять из большого количества рамок различного назначения: определяемых функций; вспомогательных переменных и объектов, подлежащих обработке; элементов рамкового оформле-

ния экрана и т.д. Написание программы в этой рамковой структуре требует упоминания тех или иных рамок и поэтому система именования рамок существенно влияет на возможности языка программирования.

Простые и составные имена

Если взять простейший вариант рамковой структуры – одноранговую совокупность рамок, имеющих различные имена, то именование рамок будет простейшим и естественным: для указания любой рамки достаточно употребить просто ее имя. Примерами такого именования служат рассмотренные ранее программы с использованием числовых рамок "тон" или "счетч".

Однако реальные системы, как правило, представляют собой сложные иерархические структуры рамок, причем на разных уровнях иерархии могут встречаться рамки с одинаковыми именами. Например, если система создается несколькими программистами, то каждый из них может создать себе вспомогательную рамку с именем "счетчик". В этом случае для однозначности именования рамки можно воспользоваться составным именем, складывающимся из нескольких имен, начиная с рамки верхнего уровня и последующих рамок, вложенных друг в друга. Имена этих рамок отделяются друг от друга в составном имени точками. Так, у одного из программистов его рамка "счетчик" может именоваться составным именем "пакет1.счетчик", а у другого – "пакет2.часть3.счетчик".

Контекстная рамка

Этот способ именования рамок достаточно понятен и естествен. Однако записывать в программе столь длинные имена оказывается очень неудобно, да и исполнителю программы требуется лишнее время на поиск всех этих имен в процессе работы программы. Можно намного упростить именование рамок в программах, если принять во внимание способ поиска рамок по их именам в Мастере. Этот поиск основывается на понятии **контекстной рамки**. Контекстной рамкой для любой формулы является та рамка, в которой эта формула содержится. Встретив имя какой-либо рамки, интерпретатор формулы ищет рамку с таким именем сначала в контекстной рамке, затем (если не нашел ее там) в рамке, охватывающей контекстную, и т.д. вверх по иерархии.

Поэтому первому программисту, работающему внутри составной рамки "пакет1", не обязательно называть свой счетчик

полным именем "пакет1.счетчик". Достаточно написать просто "счетчик". Поскольку контекстной рамкой для его формулы служит рамка "пакет1", то поиск рамки "счетчик" будет происходить сначала именно в ней – без явного указания имени "пакет1". У другого программиста в простом имени "счетчик" точно так же будет автоматически подразумеваться контекстная рамка "пакет2.часть3".

Это правило позволяет наиболее естественным образом создавать пакеты функций на языке Мастер. Несколько функций, помещенных в одну составную рамку (пакет функций), могут для вызова друг друга и для обращения к вспомогательным объектам использовать простые имена, не заботясь о возможном совпадении этих имен с именами рамок, находящихся вне пакета. При вызове этих функций из любого места рамковой иерархии (содержащего, возможно, рамки с такими же именами) все взаимные обращения функций друг к другу останутся верными.

Если нужная рамка не может быть идентифицирована простым именем из-за того, что находится не на прямой цепочке рамок от формулы до корневой рамки, то ее можно именовать составным именем. Составное имя рамки образуется последовательностью простых имен рамок начиная с той рамки, которая может быть именована непосредственно (по описанным выше правилам), и далее углубляясь в нее по последовательным уровням иерархии.

Временное изменение контекста

Описанный механизм именования рамок удобен для тех из них, которые находятся в рамковой структуре поблизости от формулы, т.е. являются частью программного пакета. Довольно часто возникает и другая потребность – именовать рамки, находящиеся в некоторой удаленной точке рамковой структуры. Для того чтобы избежать длинных составных имен, можно воспользоваться функцией **Врамке()** для временного изменения контекста, имеющей следующий формат обращения:

Врамке (временная контекстная рамка, формула)

Первый аргумент задает указатель той рамки, которую нужно временно считать контекстной. Второй аргумент – это произвольная формула, которая будет вычисляться в контексте рамки, указанной первым аргументом. Все имена в этой формуле будут разыскиваться начиная уже не от рамки, в которой находится сама формула, а от рамки, указанной первым аргументом функции **Врамке()**.

Пользуясь изменением контекста, можно вместо формулы с составными именами:

Присв (запись.поле1, значение1);
Присв (запись.поле2, значение2);
Присв (запись.поле3, значение3)

использовать формулу с простыми именами:

Врамке(запись,
 Присв (поле1, значение1);
 Присв (поле2, значение2);
 Присв (поле3, значение3)
)

Экономия в данном примере заключается как в размере формулы, так и во времени его исполнения (поскольку мы избегаем многократного поиска рамки "запись").

Вызов рамок по именам функциональных клавиш

Особую роль играют такие имена функциональных рамок, которые совпадают с именами функциональных клавиш. Такие функции могут быть вызваны не только обращением к ним из текста какой-то другой формулы, но и непосредственно с клавиатуры – нажатием соответствующей клавиши. Например, если рамка-функция имеет имя "{Ф10}", то всяко нажатие клавиши {Ф10} будет вызывать эту функцию. При этом рамка разыскивается в иерархической структуре по обычным правилам. Именно таким образом реализованы в стандартной диалоговой оболочке МАСТЕРа многие стандартные функциональные клавиши, такие, как {Ф10}, {МИНУС}, {ПЛЮС} и др.

Библиотека диалоговой оболочки МАСТЕРа

Последнее правило, определяющее механизм поиска рамок по именам, связано с использованием специальной рамки, играющей роль стандартной диалоговой оболочки МАСТЕРа.

Все меню и большинство функциональных клавиши той интерактивной среды, которая описана в первой части книги, не являются встроенным в ядро МАСТЕРа. Они запрограммированы на инструментальном языке Мастер в виде функциональных рамок и размещены в специальной рамке, называемой **библиотекой диалоговой оболочки МАСТЕРа**. Эта рамка считывается в оперативную память при загрузке МАСТЕРа, но располагается не на ра-

бочем поле, а в особой системной области, недоступной для обычной обработки.

Используется эта специальная рамка следующим образом. Всякий раз, когда в программе на языке Мастер упоминается имя некоторой рамки, а рамка с таким именем отсутствует на рабочем поле (не может быть найдена по описанным правилам), осуществляется еще один поиск – в рамке-библиотеке диалоговой оболочки. Именно таким образом срабатывают некоторые функции, требующиеся во многих приложениях, но не встроенные в ядро системы (например, функция `данет()`). Вы можете сами расширить или модифицировать библиотеку диалоговой оболочки МАСТЕРа с учетом собственных потребностей.

Стандартная библиотека диалоговой оболочки хранится в файле с именем `W1W.MAS`. Имеется возможность вызвать МАСТЕР и с любым другим файлом в качестве диалоговой оболочки. Для этого нужно в вызове использовать ключ `"-в"`. Например, вызов МАСТЕРа с файлом `START.MAS` в качестве диалоговой оболочки выполняется следующей командой:

```
MASTER -BSTART
```

Для того чтобы создать или изменить файл диалоговой оболочки, нужно считать этот файл на рабочее поле и обрабатывать, как и всякую другую рамку.

Язык макропоследовательностей в языке Мастер

Кроме формульного языка Мастер в интегрированной системе МАСТЕР имеется еще язык макропоследовательностей, обладающий совсем иной концептуальной природой. Он имеет примитивный синтаксис, в котором нет ни условных разветвлений, ни циклов, ни присваиваний. Программа на языке макропоследовательностей записывается просто как цепочка имен клавиш. Буквенно-цифровые клавиши выглядят в этой цепочке как соответствующие буквы и цифры, а функциональные клавиши записываются своими условными мнемоническими именами, такими, как `{ВВОД}` или `{ОТКАЗ}`.

Алгоритмические возможности языка макропоследовательностей очень ограничены, и программирование сколько-нибудь сложного действия на нем крайне затруднительно. Кроме того, возможности языка макропоследовательностей полностью перекрываются формульным языком Мастер, поскольку почти все функциональные клавиши запрограммированы именно формулами на языке Мастер.

Этот язык, однако, более доступен для понимания неподготовленным пользователем, нежели язык формул. Если нужно

действие легко выражается функциональными клавишами, то для его задания проще воспользоваться макропоследовательностью, чем формулой. Это тем более упрощается из-за наличия в системе МАСТЕР перехода в режим обучения, в котором система запоминает все нажимаемые пользователем клавиши и составляет протокол этих нажатий – макропоследовательность. Эту макропоследовательность можно связать с произвольной функциональной клавишей и многократно использовать.

Оба языка взаимодействуют между собой. Клавиши, составляющие команды языка макропоследовательностей, запрограммированы, как правило, на языке формул, и тем самым осуществляется вызов языка формул из языка макропоследовательностей. А для обратного вызова – макропоследовательностей из формул – существует функция `Исполнить()`, позволяющая внутри формулы обратиться к интерпретатору макропоследовательностей. Кроме того, с помощью специальных функций в языке формул можно связывать макропоследовательности с теми или иными макроклавишами, так что потом их нажатия будут вызывать эти макропоследовательности.

ОСНОВНЫЕ ВСТРОЕННЫЕ ФУНКЦИИ

В данной главе приводится описание встроенных функций, составляющих базис языка Мастер и всей интегрированной системы. Давая определения функций, мы будем приводить формат обращений к ним в виде строк следующего вида:

Строка(числовое значение [, формат])
--> строковое значение

В круглых скобках после имени функции перечисляются мнемонические обозначения ее аргументов – **числовое значение** и **формат**, записываемые курсивом. В квадратных скобках приводятся необязательные аргументы. Вслед за обращением после стрелки --> приводится обозначение возвращаемого значения функции. Если такого обозначения нет, то возвращаемым значением является значение типа ПУСТО.

Названия всех встроенных функций языка Мастер начинаются с прописных букв. Во избежание случайного совпадения имен рамок с именами встроенных функций рекомендуется все названия рамок начинать со строчных букв. Это избавит вас от неприятностей при программировании. Этого же правила мы будем придерживаться и в тексте данной книги.

8.1. Обработка числовой информации

Арифметические операции и математические функции

Средства обработки числовой информации в языке Мастер имеют традиционный характер – это обычные арифметические

180

операции, операции сравнения, логические связки, математические элементарные функции. Ниже приводится полный перечень этих операций и функций.

Арифметические операции

- | | |
|---|--------------------------|
| + | - унарный плюс; |
| - | - унарный минус; |
| + | - сложение; |
| - | - вычитание; |
| * | - умножение; |
| / | - деление; |
| ^ | - возвведение в степень. |

Операции сравнения

- | | |
|----|---------------------|
| = | - равенство; |
| # | - неравенство; |
| < | - меньше; |
| <= | - меньше или равно; |
| > | - больше; |
| >= | - больше или равно. |

Операция соединения

- && – соединение двух строк или рисунков.

Логические связки

- | | |
|---|--------------------------|
| & | - бинарная операция И; |
| | - бинарная операция ИЛИ; |
| ~ | - унарная операция НЕ. |

Математические функции

- | | |
|---------------------|--|
| $\text{Sin}(x)$ | - синус числа x; |
| $\text{Cos}(x)$ | - косинус числа x; |
| $\text{Tg}(x)$ | - тангенс числа x; |
| $\text{Ctg}(x)$ | - котангенс числа x; |
| $\text{Arcsin}(x)$ | - арксинус числа x; |
| $\text{Arccos}(x)$ | - арккосинус числа x; |
| $\text{Arctg}(x)$ | - арктангенс числа x; |
| $\text{Sh}(x)$ | - гиперболический синус числа x; |
| $\text{Ch}(x)$ | - гиперболический косинус числа x; |
| $\text{Ln}(x)$ | - натуральный логарифм числа x; |
| $\text{Exp}(x)$ | - экспонента числа x; |
| $\text{Sqrt}(x)$ | - квадратный корень из числа x; |
| $\text{Abs}(x)$ | - абсолютное значение числа x; |
| $\text{Оkr}(x)$ | - округление x до ближайшего целого числа; |
| $\text{Цел}(x)$ | - округление x до меньшего целого числа; |
| $\text{БольЦел}(x)$ | - округление x до большего целого числа; |
| $\text{Ост}(x, y)$ | - остаток от деления числа x на y. |

Для обработки чисел в электронных таблицах существует набор функций, дающих возможность массовой обработки чисел в диапазонах ячеек. Все эти функции имеют нефиксированное число аргументов, и каждый из аргументов может быть как отдельным числом, так и диапазоном. Для удобства применения этих функций диапазоны не обязаны быть целиком числовыми. Если в диапазоне встречается пустая ячейка или ячейка с нечисловым значением, то она просто не участвует в операции. К функциям такого рода относятся следующие:

- | | |
|---------------------|---|
| Сумма(...) | - вычисление суммы набора чисел; |
| Средн(...) | - вычисление среднего арифметического; |
| дисперс(...) | - вычисление дисперсии; |
| Колич(...) | - подсчет количества чисел в диапазоне; |
| Мин(...) | - вычисление минимального значения; |
| Макс(...) | - вычисление максимального значения. |

Обработка чисел, представляющих время и даты

Для информационных систем управленческого и финансового характера важную роль играют возможности обработки значений дат и времени суток. Для представления этих значений в МАСТЕРЕ применяются обычные целые числа. Даты представляются числами, отсчитывающими количество дней, а время – числами, отсчитывающими количество минут. Отсчет дней идет от 1 января 1904 года, причем этой дате соответствует отрицательное число -32767; максимально возможная дата – 4 июня 2083, ей соответствует положительное число +32767. Время суток исчисляется в минутах, причем полуночи (т.е. 0 часов 0 минут) соответствует число 0.

Для обработки этих значений в Мастере имеется несколько встроенных функций:

- Сегодня()** --> текущая дата
- Сейчас()** --> текущее время суток
- Строка(дата, форматдаты)** --> строковое изображение даты
- Число(строковоеизображениедаты, форматдаты)** --> дата
- Строка(время, форматвремени)**
 - > строковое изображение времени
- Число(строковое изображение времени, формат времени)**
 - > время

Функции **Сегодня()** и **Сейчас()** считывают с машинных часов и выдают текущие значения даты и времени суток в соответствии с определенным выше способом их кодирования.

Функция **Строка()** позволяет получить из закодированного числового значения строковое изображение даты или времени суток. Этим преобразованием управляет второй аргумент функции, задающий требуемый формат даты или времени. Форматы преобразований сами по себе представляются в языке Мастер числовыми кодами, но для наглядности эти коды имеют mnemonicические обозначения, облегчающие их запоминание. Формат складывается из двух составляющих – вида и варианта формата. Видом формата может быть в данном случае одно из двух mnemonicических обозначений – **ДАТА** или **ВРЕМЯ**. Вариант формата выбирает конкретную форму изображения даты или времени и может быть одним из следующих. Возможные варианты формата даты:

ДД_ММ_ГГГГ	- 29.08.1954
ДД_МММ_ГГГГ	- 29 авг 1954
ДД_МММ	- 29 авг
ДД	- 29
МММ_ГГГГ	- авг 1954
КВАРТАЛ_ГГГГ	- III кв. 1954
КВАРТАЛ	- III кв.

а варианты времени:

ЧЧ_ММ	- 16-30
ЧАС_МИН	- 16 час 30 мин
ЧАС	- 16 час

Комбинация вида и варианта формата задается арифметической суммой соответствующих mnemonicических обозначений.

Функция **Число()** позволяет совершить обратное преобразование: из строкового изображения даты или времени получить числовое закодированное значение. Преобразование и в этой функции определяется ее вторым аргументом, который должен задаваться одной из mnemonicических констант **ДАТА** или **ВРЕМЯ**. Добавочной константой варианта формата в этом случае не требуется.

Приведем пример работы с временными значениями. Пусть в программе требуется запросить у пользователя некоторую дату, причем в качестве ответа по умолчанию желательно предложить сегодняшнюю дату. Это можно сделать с помощью обращения:

```
запросдаты
Число(
    Запросстр(
        "Введите дату",
        Странка(Сегодня(), ДАТА+ДД_МММ_ГГГГ
    ),
    ДАТА
)
```

Здесь для диалогового запроса используется встроенная функция **Запросстр()**, имеющая два аргумента: первый задает текст вопроса, а второй – значение, предлагаемое для ответа по умолчанию. Для формирования ответа по умолчанию в этом примере используется функция **Сегодня()**, которая выдает сегодняшнюю дату в виде числа. Функция **Строка()** преобразует эту дату в строковую форму в формате вида 29 авг 1954. Пользователь вводит ответ, возвращаемый функцией **Запросстр()** в виде строки, и эта строка преобразуется в числовую форму с помощью обращения к функции **Число()** с указанием формата даты.

8.2. Строковые операции и функции

Операции синтеза и анализа строк

Для обработки строк в языке Мастер имеются следующие бинарные операции:

- = - равенство строк;
- # - неравенство строк;
- < - проверка отношения меньше;
- <= - проверка отношения меньше или равно;
- > - проверка отношения больше;
- >= - проверка отношения больше или равно;
- && - соединение (конкатенация) строк.

В этих операциях строки сравниваются на основе лексикографического упорядочения с учетом различия прописных строчных букв.

Для обработки строковых значений, кроме того, имеются следующие встроенные функции:

- Строчная(строка)** --> строка из строчных букв
- Прописная(строка)** --> строка из прописных букв
- Длина(строка)** --> длина строки
- Подстрока(строка, начальная позиция, длина)**
 - > подстрока
- Размножить(строка, число повторений)**
 - > размноженная строка
- Строка(число, формат)**
 - > строковое изображение числа
- Число(строковое изображение числа [, формат])**
 - > число
- КодКлавиши(строковое имя клавиши)**
 - > числовой код клавиши
- ИмяКлавиши(числовой код клавиши)**
 - > строковое имя клавиши

Функции **Строчная()** и **Прописная()** позволяют преобразовать буквы внутри строки из прописных в строчные и наоборот. Такое преобразование может быть полезно, скажем, в тех случаях, когда требуется сравнение строк, и его нужно сделать без учета различия строчных и прописных букв.

Преобразование чисел и строк друг в друга

Числовую и символьную информацию часто требуется преобразовывать друг в друга. Это делается двумя способами. Первый способ состоит в рассмотрении чисел как кодов символов и основан на использовании функций **КодКлавиши()** и **ИмяКлавиши()**. Второй способ состоит в рассмотрении строк как цифровых изображений чисел и основан на использовании функций **Строка()** и **Число()**.

Для функций **КодКлавиши()** и **ИмяКлавиши()** можно привести два типичных примера использования. Первый пример связан с обработкой функциональных клавиш. Приводимая ниже функция обеспечивает считывание с клавиатуры одной из двух клавиш {МИНУС} или {ПЛЮС} и возвращает числовое значение 0 или 1 соответственно. Считывание с клавиатуры выполняется в этой программе с помощью встроенной функции **Клавиша()**, ожидающей от пользователя нажатия клавиши и возвращающей код нажатой клавиши.

```
данет
Блок(1,
    Повтор(
        Присв(1, Клавиша());
        Пер(1) #КодКлавиши(" {МИНУС} ") &
        Пер(1) #КодКлавиши(" {ПЛЮС} ")
    );
    ИмяКлавиши(Пер(1)) = " {ПЛЮС} "
)
```

Второй пример показывает использование тех же функций для преобразования символьных кодов. В этом примере приведено определение функции, аргументом которой является указатель диапазона, а возвращаемым значением – строковое изображение этого диапазона вида "A10:Д30". В ее определении используются встроенные функции **ПерВВер()**, **ПослВер()**, **ПерВГор()** и **ПослГор()**, возвращающие номера первой и последней вертикальной и горизонтальной границ диапазона-аргумента. Преобразование символов в числа и обратно требуется здесь для формирования буквенных обозначений имен ячеек. Формирование строкового

имени диапазона составляется из отдельных частей с помощью операции соединения строк `&&`.

имядиапазона

ИмяКлавиши(

КодКлавиши("A") + ПервVer(Арг(1))-1) && ПервГор(Арг(1))
)&&"&&"

ИмяКлавиши(

КодКлавиши("A") + ПослVer(Арг(1))-1) && ПослГор(Арг(1))
)

Другой способ преобразования чисел и строк друг в друга основан на том, что строка рассматривается как изображение числа, составленное из цифр, десятичной точки, знака числа, порядка числа в показательной форме. Например, в обращении `Число("123.456")`

аргументом является строка, а возвращаемым значением – число, равное 123.456. При обратном преобразовании (из числа в строку) требуется указание того, в каком символьном формате должно быть получено изображение числа. Формат при этом должен задаваться числовым кодом, складывающимся из двух частей: вида формата и варианта формата. Видом формата может быть одно из следующих мнемонических обозначений:

ДЕСЯТИЧНЫЙ	- 31.456
НАУЧНЫЙ	- 3.14e+001
ОБЩИЙ	- либо 31.456, либо 3.14e+001
ПРОЦЕНТ	- 31.456 %
ДАТА	- один из форматов даты;
ВРЕМЯ	- один из форматов времени.

Любой из перечисленных видов форматов может дополняться (путем сложения) параметром, называемым вариантом формата. Этот параметр для видов форматов ДЕСЯТИЧНЫЙ, НАУЧНЫЙ, ОБЩИЙ и ПРОЦЕНТ означает количество значащих цифр, выводимых в изображении числа. В этом случае параметр задается просто числом от 0 до 7. Этот же параметр для форматов ДАТА и ВРЕМЯ выбирает один из вариантов представления даты и времени, и в этом случае он задается мнемонической константой. Варианты форматов для даты и времени перечислены в 8.1.

8.3. Обработка текстов

Понятие указателя текста

Тексты, подобно строкам, являются значениями символьного характера. Однако в отличие от строк символы в тексте располагаются не в одну линию, а переносятся со строки на строку.

186

Текст – это набор из нескольких строк. Разбиение на строки добавляет к обработке символьной информации несколько новых структурных возможностей, состоящих в выборке отдельных строк, их последовательном переборе, вставке и уничтожении. Эти операции основываются на понятии текущей строки. Текущей строкой является та, на которой при редактировании стоит курсор, а для программной обработки это понятие означает некоторую выделенную строку, к которой могут применяться операции, модифицирующие текст или выбирающие из него информацию. Есть и такие функции, которые позволяют перемещать указатель с одной строки на другую.

В языке Мастер среди прочих типов имеется тип значений, называемый **указателем текста**. Значение такого типа идентифицирует некоторый текст в целом и в то же время текущую строку в этом тексте. В описании текстовых функций словом **текст** мы будем обозначать аргументы этого типа.

Указатель текущей позиции в тексте может располагаться на любой из строк либо за последней из них. Если указатель расположен за последней строкой, то текущей строки не существует. Это состояние имеет место, например, в пустом тексте, в котором нет ни одной строки. Для обработки текстов имеется несколько встроенных функций:

ВзятьСтр(текст)	--> текущая строка
ВзятьТекст(текст, число строк, не уничтожать)	--> текст
СледСтр(текст [, число строк])	--> числом строк
Текст([текст,] строка1, строка2, строка3, ...)	--> текст
НайтиСтр(текст, подстрока поиска)	--> 0/1
ЗаменитьСтр(текст, подстрока поиска, подстрока замены)	--> 0/1
Подстрока(текст)	--> строка – очередная лексема текста
Формат(текст, операция)	
Параметр(текст, номер параметра, значение параметра)	
Параметр(текст, номер параметра)	--> значение параметра

Выборка и последовательный просмотр строк текста

Одной из главных функций при обработке текста является функция, выбирающая текущую строку – **ВзятьСтр()**. Возвращаемым значением этой функции является текущая строка тек-

та-аргумента. Сама текущая строка остается после срабатывания этой функции неизменной.

Функция **ВзятьТекст()** позволяет выбрать из текста несколько строк и, возможно, уничтожить их. Количество выбираемых строк задается вторым аргументом обращения – **число строк**, а третий аргумент **не уничтожать** означает, нужно ли уничтожить или сохранить выбранные строки в исходном тексте. Возвращаемым значением функции является текст, содержащий выбранные строки. Аргумент **число строк** может быть как положительным, так и отрицательным, что означает соответственно отсчет строк вперед или назад от текущей строки. Если же он равен нулю, то изыматься из текста должны те строки, которые были предварительно выделены в этом тексте нажатием клавиши **{F3}**.

Функция **СледСтр()** обеспечивает возможность последовательного просмотра текста. С ее помощью можно переместить указатель текущей строки на любое число строк вперед или назад по тексту. При отсутствии второго аргумента производится смещение указателя на одну строку вперед.

Конструирование текстов

Функция **Текст()** служит для конструирования текстов из отдельных строк. К ней можно обращаться двумя способами: с наличием или отсутствием в первом аргументе указателя текста. Если такой указатель в обращении присутствует, то строки, заданные остальными аргументами, добавляются к этому тексту в позиции текущей строки. Это позволяет вставлять в существующий текст новые строки. Если же указателя текста в первом аргументе нет, то обращение к функции создает новый текст, который возвращается в качестве результата функции. В частности, при отсутствии аргументов функция **текст()** порождает пустой текст без единой строки.

Комплексным примером использования функций последовательного просмотра, анализа и конструирования текстов является следующая пара функций. Первая из этих функций – **сохрМакро()** – позволяет автоматически сформировать текст, содержащий все заданные в сеансе макроопределения клавиш, а вторая – **разборМакро()** – наоборот, разбирает такой текст на отдельные макроопределения. Основой для работы этих функций является встроенная функция **ОпрМакро()**, которая при обращении к ней с одним аргументом – строкой из одной буквы – возвращает текст макроопределения для этой буквы, а при обращении с двумя аргументами – буквой и текстом макропоследовательности – связывает этот текст с указанной буквой в качестве макроопределения.

сохрМакро
Блок(3,

```
    Присв(1,КодКлавиши('A'));
    Присв(3,Текст());
    Повтор(Пер(1)<=КодКлавиши('Я'),
        Присв(2,ОпрМакро(ИмяКлавиши(Пер(1))));
        СледСтр(Пер(2),-1000);
        Если(ВзятьСтр(Пер(2))# '',
            Текст(Пер(3),'МАКРО'&&
                ИмяКлавиши(Пер(1)),Пер(2))
        );
        Присв(1,Пер(1)+1)
    );
    Записать('KEYMACRO.MAS',Пер(3));
    "запись текста в файл"
)
```

Первое обращение к функции **Текст()**, сделанное без аргументов, формирует пустой текст, к которому в дальнейшем будут добавляться тексты макроопределений, а второе обращение добавляет к этому тексту очередную порцию, состоящую из строки вида "МАКРО А" и текста макропоследовательности, заданной для этой буквы. Заметим, что текст макропоследовательности может состоять из нескольких строк. Функция **Текст()** допускает в качестве вставляемых порций не только единичные строки, но и целые тексты. В конце сформированный текст записывается этой функцией в файл **KEYMACRO.MAS**.

Обратное преобразование – из составного текста в отдельные макропоследовательности – выполняется функцией:

разборМакро

```
Блок(4,
    Присв(4,Считать('KEYMACRO.MAS',ОхвРамка()));
    Присв(1,' ');
    Присв(2,Текст());
    Повтор(
        Присв(3,ВзятьСтр(Пер(4)));
        Если(Подстрока(Пер(3),1,6)='МАКРО ',
            Если(Пер(1)#' ',ОпрМакро(Пер(1),Пер(2)));
            Присв(1,Подстрока(Пер(3),7,1));
            Присв(2,Текст())
        );
        Текст(Пер(2),Пер(3))
    );
    СледСтр(Пер(4))
);
Если(Пер(1)#' ',ОпрМакро(Пер(1),Пер(2)));
Уничтожить(Пер(4));
)
```

Поясним эту программу. Обращение к функции Считать() на начале программы считывает текст из файла KEYMACRO.MAS, созданного ранее функцией сохрмакро(). Переменная 4 – указатель текста, из которого выбираются определения макроплавиш. Переменная 1 – та буква, для которой делается очередное макроопределение. Макроопределение может оказаться многострочным, поэтому приходится запоминать эту букву и держать ее в переменной до достижения начала очередного макроопределения. В переменной 2 накапливается текст очередного макроопределения. Наконец, переменная 3 служит для выбора очередной строки текста и для ее анализа.

С целью инициализации переменной 1 присваивается пробел, что означает, что никакое макроопределение пока еще не начато. Основу программы составляет цикл Повтор(), вызываемый одним аргументом. Условие продолжения этого цикла задает стоящее в конце его тела обращение к функции Следстр(Пер(4)), переводящее указатель текущей строки тексте Пер(4) на одну строку вперед. В конце текста это очередное перемещение не удастся, функция Следстр() возвратит 0, и цикл остановится. Внутри цикла сначала выберем очередную строку текста во вспомогательную переменную 3 и, сравнивая ее со словом МАКРО, определим, не является ли она началом очередного макроопределения. Если это так, то сформируем определение предыдущей буквы, хранящейся в переменной 1, текст макроопределения для которой накопился в переменной 2, затем присвоим переменной 2 пустой текст, а в переменной 1 – запомним букву начавшегося макроопределения. Если же текущая строка – не начало очередного макроопределения, то просто добавим ее в переменную 2. По выходе из цикла нужно не забыть обработать последнее макроопределение, накопившееся в переменной 2.

Форматирующие операции над текстами

Для того чтобы завершить обработку текста, сформированного автоматически некоторой программой, требуется сформировать его абзацы, расставить разделители страниц, сформировать оглавление разделов. Для выполнения любой из этих операций служит функция Формат(). Первым аргументом этой функции является **текст**, подлежащий форматированию. Второй аргумент **операция** задает числовым кодом ту операцию, которую нужно выполнить. Возможными кодами являются следующие:

- 0 – форматирование абзаца, начинающегося в текущей строке, и установка курсора за его концом;
- 1 – форматирование абзаца, начинающегося в текущей строке, и установка курсора на его начало;

- 2 – форматирование абзацев во всем тексте, начиная с текущей строки;
- 3 – расстановка разделителей страниц;
- 3 – уничтожение разделителей страниц;
- 4 – формирование оглавления (текст сформированного оглавления возвращается в качестве значения функции).

Каждая из этих операций выполняется с учетом определенных параметров форматируемого текста – левой и правой границ абзаца, межстрочного интервала и размера страницы и т.п. Для установки значений подобных параметров текста имеется функция Параметр(). Эта функция позволяет как задать, так и запросить значение параметра. Для установки значения используется обращение с тремя аргументами: текстом, кодом параметра и значением параметра, а для запроса значения – с двумя: текстом и кодом параметра. Параметры текста в этой функции идентифицируются числовыми кодами:

- 0 – номер текущего столбца;
- 1 – номер текущей строки в текстовой рамке;
- 2 – номер текущей строки в тексте;
- 3 – режим форматирования
(0–не форматировать, 1–форматировать);
- 4 – высота страницы (в интервалах);
- 5 – режим переноса (1–включен, 0–выключен);
- 6 – режим выравнивания правого края
(1–включен, 0–выключен);
- 7 – положение левой границы абзаца;
- 8 – положение правой границы абзаца;
- 9 – относительный размер абзацного отступа;
- 10 – номер первой страницы текста;
- 11 – межстрочное расстояние (в интервалах);
- 13 – номер текущего шрифта;
- 1 – поисковый контекст для клавиши {УПР-Ф1};
- 2 – заменяющий контекст для клавиши {УПР-Ф2}.

Контекстный поиск и замена

Функции НайтиСтр() и ЗаменитьСтр() обеспечивают контекстный поиск и замену подстрок в тексте. Работа обеих функций основана на том, что в тексте помимо понятия текущей строки имеется еще и текущая позиция внутри строки – та, в которой стоит курсор в текстовой рамке. Функции контекстного поиска и замены перемещают не только текущую строку, но и позицию внутри строки. Возвращаемым значением обеих функций является признак успешности поиска строки.

Выполнение функции **НайтиСтр()** состоит в том, чтобы, найдя в тексте указанную подстроку поиска, сделать текущей позицию в начале этой подстроки. Возвращаемым значением при этом будет 1. Если же подстрока не найдена, то текущая позиция останется на месте, а возвращаемым значением будет 0.

Для выполнения функции **ЗаменитьСтр()** необходимо, чтобы текущей позицией в тексте была позиция в начале подстроки поиска, заданной первым аргументом. Тогда эта подстрока заменяется на подстроку замены, заданную вторым аргументом.

Обычным способом использования этих функций является следующий цикл:

```
Повтор(НайтиСтр(текст, подстрока поиска),
        ЗаменитьСтр(
            текст,
            подстрока поиска,
            подстрока замены
        )
    )
```

Обращение к функции **НайтиСтр()** в этом цикле служит одновременно и для поиска очередного вхождения подстроки поиска, и в качестве условия продолжения цикла. После каждого успешного поиска срабатывает обращение к **ЗаменитьСтр()**, заменяющее найденную подстроку на подстроку замены.

Рассмотрим следующую программу:

```
формированиеParameterизованногоТекста
Блок(1,
    Присв(1,1); "счетчик строк в таблице параметров";
    Повтор(Пер(1)<=ПослГор(таблицаПараметров),
            СледСтр(заготовкаТекста,-1000);
            Повтор(
                НайтиСтр(заготовкаТекста,
                    Врамке(таблицаПараметров, яч(Пер(1),1))
                ),
                ЗаменитьСтр(заготовкаТекста,
                    Врамке(таблицаПараметров, яч(Пер(1),1)),
                    Врамке(таблицаПараметров, яч(Пер(1),2))
                );
                Присв(1,Пер(1)+1)
            );
            Формат(заготовкаТекста,2); "форматирование абзацев"
            Формат(заготовкаТекста,3); "расстановка страниц"
        )
    )
```

Эта программа служит примером использования функций текстового поиска и замены и может быть весьма полезна в

автоматизации учрежденческой деятельности. Она автоматически генерирует текст некоторого стандартного документа, например договора. Исходными данными для этого служит текстовая заготовка договора и таблица параметров. Текстовая рамка содержит шаблон договора, в котором вместо конкретных адресов, названий и чисел стоят условные обозначения, скажем слова с символом # в начале (обозначения параметров могут быть произвольными, лишь бы они не совпадали случайно с другими словами в тексте).

Таблица параметров (представленная электронной таблицей) состоит из двух столбцов, в первом из которых находятся условные обозначения параметров, а во втором – их конкретные значения. Пользователю для подготовки договора остается лишь заполнить столбец значений и вызвать функцию, которая автоматически сформирует полный текст договора. Рассматриваемая программа делает контекстную замену каждого из обозначений параметров на соответствующее значение. После этих замен в тексте форматируются абзацы и расставляются страницы.

Лексический разбор текстов

Для процедурной обработки текстов очень важной является возможность их лексического разбора. Лексемы – это элементарные части текста, такие, как числа, слова, знаки препинания. В языке Мастер имеется встроенная функция **Подстрока()**, позволяющая выбирать лексемы последовательно одну за другой. Это та же функция, что используется и для выборки частей строк, но в данном случае она вызывается с одним аргументом – **текст**. Выбранная лексема является возвращаемым значением функции. После выборки лексемы текущая позиция в тексте автоматически смещается в конец выбранной лексемы.

Перед началом лексического разбора текста требуется произвести инициализацию состояния лексического анализатора и настроить его на тот текст, из которого будут выбираться лексемы. Для этого служит особое обращение к функции **Подстрока()** с передачей ей четырех аргументов:

```
Подстрока(
    текст,
    уничтожать строки,
    выбирать из фрагмента,
    убирать переносы
)
```

Первый аргумент задает тот текст, из которого будут выбираться лексемы. Второй аргумент, если он равен 1, означает,

что по мере выбора лексем следует уничтожать разобранные строки текста; если же он равен 0, то строки остаются в тексте. Третий аргумент может быть равен 0 или 1. Когда он равен 1, то выборка лексем производится только из выделенного фрагмента, в противном случае она делается до самого конца текста. Четвертый аргумент определяет, каким образом побегать с переносами. Если он равен 1, то слова, разделенные символом переноса восстанавливаются и воспринимаются как единые лексемы. Если же он равен 0, то символ переноса и правой границе абзаца рассматривается как отдельный знак минус.

8.4. Обработка графической информации

Рисунковые значения предназначены в языке Мастер для обработки графической информации. При процедурной обработке рисунковые значения являются обычными значениями языка: их можно присваивать в переменные, передавать через аргументы, возвращать в качестве значений функций. Только при помещении рисункового значения в рамку оно становится видимым на экране в виде графического изображения. Присваивание рисунковых значений рамкам производится, как и любых других значений, встроенной функцией Присв(). Ее первым аргументом должна быть указатель той рамки, куда нужно присвоить рисунковое значение, а вторым – само это присваиваемое значение.

Операции рисункового кода

Для построения рисунковых значений в языке Мастер имеется встроенная функция Рисунок(). Эта функция, выдающая в результате значение рисункового типа, принимает в качестве аргументов список числовых значений, кодирующих формируемое изображение. Например, рисунок, изображающий отрезок линии из точки (5000,5000) в точку (7000,7000), формируется обращением:

Рисунок(П, АА, 5000, 5000, В, АА, 7000, 7000)

Числа 5000 и 7000 в этом обращении задают координаты точек, а идентификаторы П, АА, В являются мнемоническими обозначениями рисунковых операций. Последовательность этих кодов выражает следующие действия: поставить перо в точку (5000,5000), затем провести линию в точку (7000,7000).

В основу системы вычисляемой графики положено специальное кодирование, в котором рисунок представляется как последователь-

тельность элементарных изобразительных операций, таких, как перенос пера, проведение линии, постановка точки, заштриховка области. Каждая такая операция представляется целочисленным кодом, который в программе может изображаться тем или иным мнемоническим именем, таким, как П, В, О и т.п. Помимо кода операция сопровождается еще и координатами той точки, к которой она применяется. Для того чтобы программа была независимой от физических свойств экрана и от размеров той рамки, в которой будет храниться изображение, координаты при рисовании задаются в абстрактных единицах в диапазоне от 0 до 10000, а в реальные точки экрана изображение преобразуется автоматически встроенным драйвером дисплея.

Задание координат в операциях может осуществляться либо абсолютным, либо относительным способом. Относительная адресация полезна в тех случаях, когда элемент изображения формируется независимо от своего местоположения и многократно повторяется в разных местах рисунка.

Основной конструкцией рисункового кода является указание адреса. Конструкция имеет один из следующих четырех видов:

АА, x, y АО, x, y ОА, x, y ОО, x, y

В каждой из конструкций сначала стоит мнемоническое обозначение кода, а за ним – координаты. Код определяет относительность или абсолютность каждой из координат: буква А соответствует абсолютной, а О – относительной адресации. Таким образом, ОА означает, что первая координата (x) задана относительно, а вторая (y) абсолютно. В последующем описании будем обозначать любую из этих четырех конструкций словом **адрес**.

Операции, управляющие рисованием и движением рисующего пера, имеют следующую структуру:

- | | |
|------------------------|---|
| П, адрес | - перенос пера в указанную точку без рисования; |
| Т, адрес | - постановка точки по указанному адресу; |
| В, адрес, адрес, ... | - проведение ломаной линии из текущей точки; |
| О, адрес | - изображение окружности с центром в текущей точке; |
| Д, адрес, угол1, угол2 | - изображение дуги окружности. |

Процесс построения изображения производится в одном из двух режимов, отличающихся способом наложения новых линий и точек на уже существующие. Первый режим называется заменяющим, второй – исключающим. В заменяющем режиме новые точки

заменяют собой существующие. В исключающем режиме новые точки накладываются на существующие с помощью операции "исключающее ИЛИ". Для включения того или иного режима служат рисунковые операции:

- ЗАМ - включение заменяющего режима;
ДОП - включение исключающего режима.

Еще одна возможность рисункового кода связана с закрашиванием областей, для чего следует включить закрашивание и определить маску закрашивания. Это включение воздействует на последующие операции проведения ломаной линии: закрашивается область, находящаяся между проводимой линией и некоторой вертикальной линией, установленной в качестве базы заштриховки. База заштриховки и маска заштриховки устанавливаются операциями:

- ЗГА, координата_x - абсолютная база заштриховки;
ЗГО, координата_x - относительная база заштриховки;
ЗНЕТ - выключение режима закрашивания;
ЗАКР, ряд1, ряд2, ряд3, ряд4 - задание четырех рядов маски.

Все описанные конструкции должны следовать друг за другом через запятую в качестве групп аргументов функции Рисунок().

Соединение рисунков

Имея два рисунковых значения, сформированных, например, обращениями к функции Рисунок(), можно объединить их в единый рисунок. Это делается применением операции соединения:

Рисунок(...) & Рисунок(...)

Данная операция, применимая также и для строковых значений, выполняет, вообще говоря, соединение двух значений. Для рисунков соединение означает, что рисующее перо выполнит последовательно сначала рисунковые операции первого рисунка, а потом операции второго рисунка.

Построение графиков функций

Функция График() предназначена для построения графиков функций, заданных поточечно на электронной таблице. Она выдает значение рисункового типа точно так же, как и функции

Рисунок(). Обращение к функции График() описано в первой части книги – в главе об электронных таблицах.

8.5. Электронные таблицы

В первой части книги мы познакомились с электронными таблицами как с диалоговым средством для редактирования табличной информации. В языке Мастер имеются встроенные функции, позволяющие выполнять над таблицами различные редактирующие операции, и эти функции могут использоваться для расширения или специализации средств диалоговой работы с таблицами. Но таблицы играют и другую важную роль. Они могут рассматриваться как средство структурирования информации, аналогичное массивам (векторам и матрицам) в других языках программирования.

Указатели ячеек и диапазонов ячеек

Основой для выполнения всех операций над таблицами служит специальный тип данных языка Мастер, называемый **указателем диапазона**. Значения такого типа представляют ссылки на прямоугольные области в табличных рамках. В частном случае прямоугольная область может содержать всего лишь одну ячейку, и такой указатель диапазона мы будем называть **указателем ячейки**.

Простейший способ записать указатель диапазона в программе на Мастере состоит в том, чтобы написать имена его граничных ячеек в буквенно-числовой нотации с разделительным двоеточием. Например, А10:Б20 задает указатель диапазона, находящийся на пересечении столбцов А и Б со строками с 10-й по 20-ю. Простое имя ячейки А10 тоже является указателем диапазона, состоящего из одной ячейки. Оно эквивалентно диапазону А10:А10.

Такие буквенно-числовые обозначения достаточны только для указания фиксированных диапазонов в таблице. Если же нужно обработать ячейки в цикле или обратиться к ячейке, адрес которой зависит от вычисляемых данных, то требуются другие средства. Для этой цели служит встроенная функция ЯЧ(), имеющая два аргумента:

ЯЧ(номер строки, номер столбца)

Функция возвращает указатель ячейки с номерами строки и столбца, которые задаются произвольными числовыми выражениями.

Как бы ни задавался адрес ячейки – либо буквенно-числовым именем, либо функцией ЯЧ() – в нем не указывается та табличная рамка, к которой он относится. В этом отношении действует то же правило, что и для имен рамок: имя ячейки относится к контекстной рамке. Напомним, что контекстной рамкой для каждой формулы является та, в которой расположена эта формула, либо та, которая назначена контекстной временно с помощью обращения к функции Врамке().

Так, например, для сложения значений двух ячеек Б10 из рамок РМК1 и РМК2 можно воспользоваться следующей формулой:

Врамке(РМК1, !Б!10)+Врамке(РМК2, !Б!10)

Для работы с указателями диапазонов требуется определять их граничные индексы. Это нужно, например, для организации циклической обработки строк и столбцов, образующих диапазон. Для этой цели служат четыре встроенных функции:

ПервГор(диапазон) --> номер первой строки
ПервВер(диапазон) --> номер первого столбца
ПослГор(диапазон) --> номер последней строки
ПослВер(диапазон) --> номер последнего столбца

Присваивание значений ячейкам

Для того чтобы присвоить ячейке значение, нужно воспользоваться функцией Присв(), в которой в качестве первого аргумента должен стоять указатель ячейки. Рассмотрим для примера программу, выполняющую транспонирование матрицы.

```
транспонироватьМатрицу—  
блок(1,  
    Присв(строка,1);  
    Повтор(строка<=10,  
        Присв(столбец,1);  
        Повтор(столбец<строка,  
            Присв(1,Извлечь(Яч(строка,столбец)));  
            Присв(Яч(строка,столбец),Яч(столбец,строка));  
            Присв(Яч(столбец,строка),Пер(1));  
            Присв(столбец,столбец+1)  
        );  
        Присв(строка,строка+1)  
    )  
)
```

В этой программе следует обратить внимание на одно важное обстоятельство. В качестве временного хранилища для значений

ячейки в ней использована локальная переменная, которой выполняется присваивание в строке:

Присв(1,Извлечь(Яч(строка,столбец)))

В этом присваивании к указателю ячейки, задаваемому функцией ЯЧ(), применяется функция Извлечь(). Это необходимо для того, чтобы присвоить локальной переменной не указатель на ячейку, а содержимое этой ячейки. В противном случае после присваивания, следующего за этой строкой, значение данной ячейки было бы потеряно.

Статистические функции

Имеется набор операций, предназначенных для вычислений числовых величин статистического характера. Все эти функции имеют нефиксированное число аргументов, каждый из которых может быть либо простым числом, либо указателем диапазона. Действие любой из следующих функций распространяется только на числовые ячейки диапазонов, заданных в их аргументах, а все остальные ячейки игнорируются.

Колич()	– количество числовых значений;
Сумма()	– сумма числовых значений;
Средн()	– среднее арифметическое числовых значений;
Дисперс()	– дисперсия числовых значений;
Макс()	– максимальное из числовых значений;
Мин()	– минимальное из числовых значений.

Преобразование структуры таблиц

Имеется ряд функций, которые работают с таблицей, преобразуя так или иначе ее структуру: вставляют и уничтожают строки или столбцы, копируют информацию целыми диапазонами и т.п. Именно через эти функции реализованы соответствующие команды табличного меню. Они имеют следующие форматы обращения:

Копировать(диапазон-источник, диапазон-получатель)
ВставитьГор(диапазон строк)
ВставитьВер(диапазон столбцов)
УничтожитьГор(диапазон строк)
УничтожитьВер(диапазон столбцов)

Аргумент **диапазон строк** задается указателем диапазона, в котором существенны только строки – именно эти строки унич-

тожаются или вставляются. В аргументе **диапазон столбцов**, наоборот, существенны только столбцы.

При вставке и уничтожении строк и столбцов все формулы в данной таблице автоматически изменяются таким образом, чтобы ячейки зависели от тех же ячеек, что и до выполнения операции. Возвращаемым значением во всех этих функциях является указатель диапазона, служивший аргументом функции.

Поиск значений в таблицах

Весьма полезной операцией при работе с электронными таблицами является функция поиска значений. С ее помощью можно и упорядочивать строки, и реализовывать таблично заданные функции, и выбирать атрибуты табличных записей. Обращение к этой функции имеет следующий формат:

ТабПоиск(диапазон, искомое значение [, отступ])
--> указатель ячейки с отступом от найденной
или
ТабПоиск(диапазон, искомое значение,
отступ, упорядоченность
) --> указатель ячейки с отступом от найденной

Эта функция выполняет поиск значения, заданного аргументом **искомое значение** в указанном **диапазоне**, и выдает в результате указатель найденной ячейки. Как правило, найдя значение в таблице, нужно взять не само это значение (оно и так известно), а соседнее справа или слева значение. Тем самым реализуется табличное задание функций. Для этого задается третий аргумент – **отступ**, который указывает, на сколько столбцов или строк нужно отступить от найденной ячейки (если диапазон является столбцом, то отступ делается вправо, а при поиске в строке отступ делается вниз). При отсутствии третьего аргумента отступ равен единице, и поэтому при поиске в столбце берется соседняя справа ячейка, а при поиске в строке – соседняя снизу ячейка.

Если значение не найдено, то возвращаемое значение имеет тип ПУСТО. Этим можно пользоваться для выяснения того, найдено значение или нет. Удобно присваивать возвращаемое значение функции ТабПоиск() локальной переменной, чтобы сначала проанализировать успешность поиска, а затем воспользоваться его результатом. Это показано в следующей функции, которая выполняет поиск значения, заданного ее аргументом в диапазоне A1:A100 таблицы ТАБ1, и выводит в поле сообщения значение, лежащее рядом с найденной ячейкой.

Блок(1,

```
Присв(1, ТабПоиск(Врамке(ТАБ1, А1:А100), Арг(1), 1));
Если(Тип(Пер(1), 0)=пусто,
Сообщить('Не найдено число '&&Арг(1)),
Сообщить('Числу '&&Арг(1)&&
' соответствует '&&Извлечь(Пер(1))
)
)
```

При обращении к ТабПоиск() с двумя или тремя аргументами поиск значения выполняется путем последовательного просмотра всех ячеек. Если известно, что значения в **диапазоне** упорядочены, то поиск может быть выполнен более эффективно. Для этого служит второй вариант обращения, в котором аргумент **упорядоченность** задает способ упорядочения значений в диапазоне: +1 – по возрастанию, -1 – по убыванию. Функция не проверяет того, действительно ли диапазон является упорядоченным указанным образом. Она просто применяет соответствующий алгоритм поиска. Если аргумент **упорядоченность** не соответствует действительному упорядочению диапазона, то результат поиска будет неправильным. В этом варианте обращения функция ТабПоиск() никогда не возвращает значения типа ПУСТО. Если искомого значения в диапазоне нет, то возвращается указатель на то место, где следовало бы стоять искомому значению. Это дает возможность реализовывать упорядочение диапазонов.

Рассмотрим пример программы, которая упорядочивает строки таблицы ТАБ1, располагая их в таблице ТАБ2 по возрастанию значения самой левой ячейки каждой строки:

упорядочениедиапазона

Блок(2,
Очистить(ТАБ2); "уничтожим содержимое ТАБ2";
Присв(1, 1); "счетчик строк в ТАБ1";
Повтор(Пер(1)<=ПослГор(ТАБ1), "цикл по ТАБ1";
Присв(2,
ТабПоиск(ТАБ2, Врамке(ТАБ1, Яч(Пер(1), 1)), 0, 1)
);
Врамке(ТАБ2, ВставитьГор(Пер(2)));
Копировать(
Врамке(ТАБ1,
Яч(Пер(1), 1):Яч(Пер(1), ПослВер(ТАБ1))
),
Пер(2)
);
Присв(1, Пер(1)+1)
)
)

Работа с формулами в электронных таблицах

Существенной чертой электронных таблиц является наличие в них расчетных формул. Формулы также доступны для программной обработки на языке Мастер, для чего имеются следующие встроенные функции:

УстФормула(ячейка, текст формулы)
Формула(ячейка) --> текст формулы
ЕстьФормула(ячейка) --> 0/1
Пересчет(диапазон [, по строкам])

Первая функция служит для вставки формулы в ячейку. Сама формула задается значением текстового типа и поэтому может быть сформирована с помощью функций строкового и текстового характера. С помощью этой же функции можно уничтожить формулу в ячейке. Для этого нужно задать в обращении пустой текст, например, следующим обращением:

УстФормула(A1, Текст())

Функция Формула() позволяет запросить текст формулы из ячейки. Этот текст является возвращаемым значением функции в виде обычного текстового значения, которое точно так же можно обрабатывать текстовыми и строковыми функциями Мастера.

Наконец, функция ЕстьФормула() позволяет узнать, имеется ли какая-нибудь формула в ячейке. Возвращаемое значение 1 означает, что формула есть, значение 0 означает отсутствие формулы.

Еще одна функция – Пересчет(), относящаяся к работе с формулами в электронных таблицах, предназначена для вызова этих формул на вычисление. Пересчет делается не во всей таблице, а только в том диапазоне, который задан первым аргументом обращения. Второй аргумент задает способ пересчета: 1 означает пересчет по строкам, 0 – по столбцам. При отсутствии второго аргумента пересчет делается по строкам. С помощью этой функции можно реализовывать в таблицах какие-либо специфические последовательности пересчета формул, отражающие структуру их функциональных зависимостей.

Преобразование информации между таблицами и текстами

Для преобразования информации между текстовой и табличной формами служат две встроенные функции ТабТекст() и ТекстТаб(), имеющие следующие варианты обращения:

ТабТекст(диапазон) --> текст

ТабТекст(диапазон, имя файла, признак продолжения файла)
ТекстТаб(текст, диапазон, делать разбор строк)

Функция ТабТекст() позволяет сформировать текстовое изображение табличной информации. При этом сохраняется весь внешний вид таблицы, включая размеры ячеек, их разграфку, форматы визуализации чисел. Функция может применяться для двух целей: либо для передачи текста в какую-нибудь текстовую рамку, либо для записи его во внешний текстовый файл. В первом случае используется обращение с одним аргументом, и сформированный текст оказывается возвращаемым значением функции. Во втором случае текст формируется не в оперативной памяти, а сразу на диске – в файле, имя которого задано вторым аргументом. Третий аргумент признак продолжения файла задает способ открытия этого файла. Если он равен 0, то файл создается заново, а если 1, то производится дозапись текста в конец существующего файла. Это позволяет формировать текстовое изображение большой таблицы несколькими порциями.

Функция ТекстТаб() производит обратное преобразование: текст, заданный первым аргументом, разбирается на отдельные значения, которые размещаются в диапазоне, заданном вторым аргументом. Третий аргумент делать разбор строк определяет, нужно ли разделять строки на отдельные значения или следует помещать строки целиком в левые ячейки диапазона. Если аргумент делать разбор строк равен 1, то разбор выполняется; если же он равен 0, то разбор не выполняется.

Электронные таблицы как средство визуализации

Электронные таблицы – не только удобные структуры для обработки регулярной информации. Они столь же полезны для визуализации информации. Самые сложные выходные формы документов лучше всего генерировать на электронных таблицах. Для этого следует заранее подготовить заготовку такой выходной формы в виде некоторой таблицы-пустографики, расположив в ней вручную все требуемые форматы, разграфку, ширину столбцов, высоту строк и т.д. Эта таблица должна храниться в вашей программе как вспомогательная структура. При формировании выходного документа программе нужно будет только разослать определенные числовые и строковые значения в нужные ячейки этой пустографики. После подготовки документа в табличном виде его можно преобразовать в текстовый вид с помощью функции ТабТекст().

Если таблица, используемая для отчета, не подготовлена заранее вручную, а формируется автоматически программой, то

в этой программе потребуются функции, выполняющие настройку таблицы, ее форматов, ширины столбцов и т.д. Каждая из этих функций имеет два варианта обращения: с одним аргументом – диапазоном и с двумя аргументами – диапазоном и числовым значением. В первом случае функция возвращает значение формата или ширины, во втором случае – устанавливает в ячейке это значение. Функции имеют следующие форматы обращения:

```
Формат(диапазон [, код формата])
--> диапазон или код формата
Ширина(диапазонстолбцов [, ширина])
--> диапазон или суммарная ширина
Разграфка(диапазон [, код разграфки])
--> диапазон или код разграфки
```

В обращении к функции **Разграфка()** в качестве кодов разграфки служат следующие числа (эти коды не совпадают с кодами символов псевдографики, они используются только в функции **Разграфка()**):

Элемент разграфки	Код разграфки
-	80
+	16
9	96
1	11
3	32
2	12
T	14
J	16
L	17
L	64
G	48
ничего	0

8.6. Обработка рамок

Рамка является основным структурным понятием системы МАСТЕР и языка Мастер. Рамки обеспечивают и хранение информации, и ее визуализацию и являются интерактивной средой для диалоговой обработки информации. В языке Мастер имеется большой и технологически полный набор встроенных функций, выполняющих над рамками действия, связанные с любым из этих трех аспектов. Заметим, что наличие этих трех функций существенно отличает язык Мастер от встроенных языков других интегрированных систем (подобных Symphony, Framework), в ко-

торых действия над их основными структурными единицами (окнами, фреймами) доступны только косвенно – через макропоследовательности, имитирующие нажатия функциональных клавиш и вызовы команд из меню. В МАСТЕРе, наоборот, все команды меню, в том числе и над рамками, реализованы через встроенные функции языка, а сам язык намного более удобен и гибок для программирования обработки рамковых структур.

Указатели рамок

Во всяком языке программирования одним из центральных вопросов семантики является вопрос о способах именования обрабатываемых в языке информационных объектов. Многие вычислительные возможности языка находятся в прямой зависимости от гибкости и эффективности этой системы именования объектов. Точно так же и в семантике Мастера этот вопрос является одним из определяющих.

Именование рамок осуществляется двумя способами: через имена (статически) и через ассоциативные структурные связи рамок друг с другом (структурно). Имена дают доступ к фиксированным рамкам, а структурные связи позволяют устраивать циклические переборы рамок, осуществлять вычисляемую адресацию рамок, что совершенно аналогично именованию ячеек: буквенно-числовые обозначения ячеек являются их статическими именами, а обращение к функции **яч()** обеспечивает вычисляемую адресацию ячеек.

Доступ к рамкам основывается на понятии **указателя рамки**. Подобно указателю диапазона или указателю текста указатель рамки сам по себе не является содержательным значением. Его назначение состоит лишь в том, чтобы дать доступ к рамке, а этим доступом можно воспользоваться, чтобы либо взять значение из рамки, либо занести в нее новое значение, либо модифицировать ее значение, либо получить доступ к любой из рамок, иерархически связанных с данной.

Во всех описываемых ниже функциях подразумевается, что аргументы, обозначаемые такими словами, как **рамка**, **охватывающая рамка**, **составная рамка** и т.п., являются указателями рамок. Эти аргументы могут быть сформированы любым из способов, дающих указатели рамок, а не обязательно именами или какими-то простыми выражениями.

Статическое именование рамок

Простейшим выражением в языке Мастер, дающим доступ к рамке (т.е. обладающим значением типа "указатель рамки"), яв-

ляется идентификатор рамки. По этому идентификатору во время выполнения формулы разыскивается рамка, причем поиск идет от той рамки, в которой находится формула, содержащая идентификатор, путем постепенного подъема вверх по иерархии рамок. Результатом этого поиска является указатель найденной рамки, который и становится значением идентификатора.

Подобным же образом можно именовать рамку составным именем, в котором идентификаторы следуют друг за другом через точку. Тогда первый из идентификаторов разыскивается в рамковой иерархии по описанному правилу, а остальные рамки берутся с последовательно вложенных уровней иерархии.

На этот поиск может воздействовать специальная встроенная функция **Врамке()**, временно изменяющая контекст поиска, т.е. ту рамку, с которой начинается поиск. Эта функция имеет следующий формат обращения:

Врамке(контекстная рамка, формула)
--> значение формулы

Первый аргумент – **контекстная рамка** – задает указатель той рамки, которую следует временно считать контекстной. Новый контекст распространяется только на те имена рамок, которые находятся внутри формулы, служащей вторым аргументом. Возвращаемым значением этого обращения служит то значение, которое выработала эта формула.

Обратите внимание на то, что во время вызовов определяемых функций друг из друга контекстная рамка автоматически переключается: во время работы любой определяемой функции контекстной является та рамка, в которой находится рамка-функция, а при завершении ее работы контекстной рамкой опять становится та, которая была контекстной перед вызовом этой функции. Здесь действует то же самое общее правило: контекстной рамкой для любой формулы является та, в которой расположена эта формула.

Некоторые рамки по тем или иным причинам имеют такие имена, которые не являются правильными идентификаторами. Например, если в имени содержатся пробелы или знаки препинания, то оно не является идентификатором и не может быть записано в формуле непосредственно. Для именования такой рамки нужно заключить нестандартное имя в кавычки и обратиться к функции

Рамка([составная рамка,] строковое имя рамки)
--> указатель рамки

Если первого аргумента не задано, то рамка будет разыскиваться в рамковой структуре по тем же правилам, что и рамки

с нормальными именами-идентификаторами. Если же в первом аргументе задана некоторая составная рамка, то имя будет найдено внутри этой составной рамки.

Структурное именование рамок

Описанный способ именования рамок является статическим, поскольку каждая рамка требует фиксированного имени. Таким способом невозможно, например, перебрать все рамки определенного уровня иерархии или адресовать рамку, которая является текущей, независимо от ее имени.

Альтернативный способ называется структурным именованием рамок. Он реализуется не через имена, а через обращения к специальным встроенным функциям, основанным на иерархической структуре рамок. Напомним, что иерархическая структура рамок образуется связями двух типов. Включение рамок друг в друга определяет для каждой рамки охватывающую рамку, т.е. ту, внутри которой первая рамка находится. Это позволяет подниматься вверх по иерархии. Внутри каждой составной рамки одна из рамок является текущей, что позволяет спускаться вниз по иерархии. Вторым типом связи рамок является логическая цепочка рамок, объединяющая в последовательность все рамки, находящиеся внутри одной составной рамки. Эта цепочка позволяет перемещаться от рамки к рамке по горизонтали рамковой иерархии.

Функции структурного доступа к рамкам имеют следующие форматы обращения:

ОхвРамка()
--> охватывающая рамка
ОхвРамка(рамка)
--> составная рамка, содержащая данную рамку
ТекРамка()
--> текущая рамка
ТекРамка(составная рамка)
--> текущая рамка внутри заданной
СледРамка(рамка)
--> рамка, следующая за данной
СледРамка(рамка, 0)
--> самая последняя рамка уровня
ПредРамка(рамка)
--> рамка, предыдущая перед данной
ПредРамка(рамка, 0)
--> самая первая рамка уровня
ВнешРамка()
--> корневая рамка

Приведенный набор функций является полным для того, чтобы обеспечить доступ до любой рамки в рамковой иерархии. Функции СледРамка() и ПредРамка() позволяют двигаться в пределах одной составной рамки на заданном иерархическом уровне, а функции ОхвРамка() и ТекРамка() – между уровнями иерархии. Во всех этих функциях в случае, если требуемой рамки (следующей, предыдущей или текущей) не существует, возвращается значение типа пусто. Подчеркнем, что эти функции никак не влияют на положение курсора в рамковой иерархии. Они позволяют, оставаясь на своем месте, получать доступ к любой рамке. Рассмотрим для примера следующую программу, которая опрашивает все рамки верхнего уровня, выясняя, была ли модифицирована какая-нибудь из них, и сохраняя таковые на диске. Условием продолжения цикла в этой программе является проверка того, что переменная 1 содержит указатель на рамку, а не значение типа пусто.

```
Блок(1,
    Присв(1,ПредРамка(ТекРамка(ВнешРамка()),0));
    Повтор(Тип(Пер(1),0)=РАМКА,
        Если(Модифицирована(Пер(1)),
            Запись(Имя(Пер(1)),Пер(1))
        );
        Присв(1,СледРамка(Пер(1)))
    )
)
```

Рамки как носители информации и структуры

В качестве носителя информации рамковая структура предстает как дерево, состоящее из узлов – рамок. Терминальные узлы этого дерева содержат некоторую информацию – значения числового, строкового, текстового, табличного, рисункового типов. Значение может быть извлечено из рамки для использования в той или иной обрабатывающей операции или функции. Для этого существует специальная функция:

Извлечь(рамка)

--> значение, содержающееся в рамке

Например, если у нас имеется простая рамка "счетчик", содержащая некоторое число, то этим числовым значением можно воспользоваться с помощью функции Извлечь() следующим образом:

Присв(счетчик, Извлечь(счетчик)+1)

Здесь к значению, извлеченому из рамки "счетчик", прибавляется единица и полученное число присваивается в ту же рамку "счетчик".

При работе с рамками (в особенности числовыми и строковыми) извлечение значений требуется постоянно. Для упрощения формул можно опускать явное обращение к функции Извлечь() во всех тех случаях, когда это не приводит к неоднозначности. Таким образом, вместо рассмотренной формулы можно применить следующую:

Присв(счетчик, счетчик+1)

Здесь идентификатор "счетчик" в выражении "счетчик+1" обладает, как и всякий идентификатор в языке Мастер, значением типа "указатель рамки". Но поскольку по контексту (в операции сложения) указатель не имеет смысла, а требуется число, то к указателю неявно применяется операция Извлечь(), которая и заменяет его на требуемое число – значение, содержащееся в рамке счетчик.

ВНИМАНИЕ. В некоторых формулах операция может быть применена как к значению рамки, так и к самому указателю этой рамки. Наиболее важными случаями такого рода является присваивание указателя рамки локальной переменной (это единственное хранилище, способное содержать указатели рамок, диапазонов и текстов). Следует понимать принципиальную разницу между двумя формулами:

Присв(1,счетчик)

и

Присв(1,Извлечь(счетчик))

В первом случае переменной 1 присваивается указатель на рамку счетчик, а во втором – значение, содержащееся в этой рамке. Это различие проявляется, например, в следующей формуле:

```
Блок(1,
    Присв(1,счетчик);
    Присв(счетчик,счетчик+1);
    Сообщить(Пер(1))
)
```

Здесь переменной 1 был присвоен указатель на рамку "счетчик", а содержимое этой рамки было изменено, затем мы воспользовались в функции Сообщить() значением той рамки, на которую указывает указатель в переменной 1. Эта программа выведет в поле сообщения не старое значение рамки счетчик, а новое. Если же в первом присваивании вместо счетчик напи-

сать Извлечь(счетчик), то в переменной сохранится не указатель на рамку, а старое значение счетчика, и будет распечатано именно оно, несмотря на произошедшее изменение содержимого рамки счетчик.

В работе с информацией нас интересуют не только отдельные элементы – числа и строки, но еще и структура, объединяющая эти элементы в единое целое. Рамки являются как носителями отдельных значений, так и средством образования структуры. Это делается за счет составных рамок, содержащих наборы других рамок. Операции передвижения по этой структуре уже описаны в предыдущем разделе: функции ОхвРамка(), ТекРамка(), СледРамка() и ПредРамка() позволяют, отталкиваясь от одной рамки, достигать любой другой рамки. Кроме того, работа со структурой требует возможностей порождения и модификации самой этой структуры, создания и уничтожения новых элементов, изменения ее топологии. Для этой цели в языке Мастер имеются следующие функции:

```
Создать(
    составная рамка,
    имя рамки,
    тип значения,
    тип окаймления рамки,
    горизонтальная позиция рамки,
    вертикальная позиция рамки,
    высота рамки,
    ширина рамки,
    цветовой атрибут рамки
) --> указатель созданной рамки
Уничтожить(рамка)
Очистить(рамка)
Переставить(переставляемая рамка, рамка-позиция)
Модифицирована(рамка) --> 0/1
```

Функция Создать() позволяет создать новую рамку в произвольном месте рамковой иерархии. Аргумент составная рамка задает указатель на ту составную рамку, внутри которой должна быть создана новая рамка. Остальные аргументы задают основные атрибуты рамки: имя, тип ее значения, тип окаймления, положение, размеры и цвет. Новая рамка вставляется в логическую цепочку вслед за той рамкой, которая является текущей в заданной составной рамке.

Функция Переставить() изменяет логическое положение рамки в иерархической структуре. Рамка, заданная первым аргументом – переставляемая рамка, убирается из того места, где она находилась, и вставляется в другое место – перед той рамкой,

которая указана вторым аргументом – рамка-позиция. Таким образом рамку можно переставить как в пределах одного иерархического уровня, так с уровня на уровень.

В качестве примера можно рассмотреть программу, которая переставляет рамки, входящие в охватывающую рамку, и расположает их в порядке возрастания вертикальной координаты рамки. Эта программа может быть полезна для того, чтобы привести логическое расположение рамок в соответствие с их пространственным расположением.

```
Блок(1,
    УстТек(ПредРамка(ТекРамка(),0));
    Повтор(Тип(СледРамка(ТекРамка()),0)=РАМКА,
        Присв(1,1);
        Повтор(Тип(СледРамка(ТекРамка(),Пер(1)),0)=РАМКА,
            Если(ПозСтр(ТекРамка())>
                ПозСтр(СледРамка(ТекРамка(),Пер(1))),
                Переставить(ТекРамка(),
                    СледРамка(ТекРамка(),Пер(1)))
            );
            Присв(1,Пер(1)+1)
        );
        УстТек(СледРамка(ТекРамка())))
    )
```

Рамки как средство визуализации

Обратимся теперь ко второму свойству рамки, состоящему в том, что она является средством визуализации информации. На нем основывается организация диалога в МАСТЕРе. Из-за того, что рамки имеют определенный внешний вид, программисту не приходится процедурными средствами обеспечивать появление на экране каждого символа, каждого элемента изображения. Достаточно вручную нужным образом расположить рамки друг относительно друга, расцветить их, задать определенное окаймление, а затем при работе делать текущей то одну, то другую рамку, и на экране автоматически будут появляться нужные картины. Программно можно лишь изменять атрибуты внешнего вида рамок, отражая ими состояние информации.

Это определяет технологию программирования в МАСТЕРЕ; значительное место в ней отводится конструированию той среды, в которой будет работать пользователь. Это конструирование аналогично описанию структур данных при традиционном

программировании, но здесь вы не описываете информационную среду, а строите ее, находясь непосредственно в ней. Тем самым сразу видны и все визуальные и все структурные свойства, и на долю программирования остается только доопределение каких-то специфических операционных свойств.

Но сам внешний вид рамковой среды не оказывается зафиксированным. Программа на Мастере может изменять не только структуру или информационное содержание, но и внешний вид. Для этого служат следующие функции:

```
Имя(рамка [, имя рамки]) --> имя рамки
ТипКаймы(рамка [, тип каймы]) --> тип каймы
ПозСтр(рамка [, номер строки]) --> номер строки
ПозСтЛв(рамка [, номер столбца]) --> номер столбца
Высота(рамка [, высота рамки]) --> высота рамки
Ширина(рамка [, ширина рамки]) --> ширина рамки
УстРамку()
Цвет(рамка [, цвет рамки]) --> цвет рамки
Открыта(рамка) --> 0/1
Открыть(рамка)
Закрыть(рамка)
Распахнуть()
Формат(простая рамка [, формат простого значения])
--> код формата
```

Любая из этих функций, допускающих два аргумента, работает по-разному в зависимости от числа аргументов. Если задан только один аргумент (рамка), то функция ничего не изменяет в этой рамке, а просто возвращает в качестве своего значения соответствующий атрибут рамки. Если же задан второй аргумент, то его значением заменяется соответствующий атрибут рамки и тем самым она меняет цвет или имя, положение или размеры и т.п.

Отдельного пояснения здесь требуют лишь некоторые функции. В функции ТипКаймы() во втором аргументе используется код, который задает тип окаймления рамки. Этот код может складываться из нескольких слагаемых. Первое слагаемое задает вид линии, а второе – наличие имени в изображении рамки. Первое слагаемое может быть либо одной из констант ДВОЙНАЯ или ОДИНАРНАЯ, либо нулем (последнее означает отсутствие какой-либо линии). Второе слагаемое может быть либо мнемонической константой имя, либо нулем (что соответствует отсутствию изображения имени).

Подобным же образом формируется код формата в функции Формат(). Этот код задается из трех слагаемых. Первое задает вид формата, второе – вариант формата, а третий – способ вы-

равнивания числового или строкового значения по ширине рамки. Вид и вариант формата задаются так же, как и в функции Стока(), описанной в п. 8.2, а способ выравнивания задается одной из мнемонических констант ВЛЕВО, ВПРАВО, ВЦЕНТР, СКРЫТО.

Изменение атрибутов рамки может быть сделано программой на Мастере не только через явное задание значений этих атрибутов, но также и запросом их у пользователя, причем не в виде чисел, а непосредственным указанием. Для этого служит функция УстРамку(). В момент обращения к данной функции возникает диалог с пользователем, позволяющий ему с помощью стрелок установить желаемые внешние атрибуты рамки – ее положение, размеры, цвета фона и букв. Возвращает значение типа ПУСТО.

8.7. Интерфейс с файловой системой

Взаимодействие с файловой системой представляет для МАСТЕРа интерес в двух отношениях: ввод-вывод информации в файлы и вызов прикладных программ.

Ввод-вывод информации

Для ввода-вывода информации в файлы существуют две встроенные функции Записать() и Считать(), имеющие несколько вариантов обращения:

```
Записать(имя файла, записываемая рамка)
Записать(имя файла, указатель текста)
ТабТекст(диапазон, имя файла, признак продолжения файла)
Считать(имя файла, составная рамка) --> указатель рамки
```

В случае, когда вторым аргументом функции Записать() является указатель рамки, эта рамка записывается в рамковый файл в специальном внутреннем представлении, в котором сохраняется не только содержащаяся в рамке информация, но и размеры, цвет, формулы локальных действий клавиш и т.д. В случае, когда вторым аргументом функции Записать() является указатель текста, запись в файл производится в стандартном текстовом формате, доступном для считывания другими информационными системами, например текстовыми процессорами.

Функция Считать() в качестве второго аргумента имеет указатель составной рамки. В этой составной рамке функция создает новую рамку и считывает в нее информацию из файла, имя

которого задано первым аргументом. Функция сама определяет в каком формате был записан этот файл – в рамковом или текстовом.

Функция **ТабТекст()** записывает табличную информацию в текстовом виде, сохраняя внешний вид этого диапазона, но без формул и прочих атрибутов ячеек. Аргумент **признак продолжения файла** может быть равен 0 – тогда файл создается заново, а если он равен 1, тогда информация дозаписывается в конец существующего файла.

Вызов внешних прикладных программ

При организации сложных прикладных информационных систем встроенных возможностей МАСТЕРа может оказаться недостаточно либо из-за их недостаточной эффективности, либо по иным причинам. В таких случаях можно воспользоваться какой-нибудь прикладной программой, созданной вне МАСТЕРа и существующей в виде EXE- или СОМ-файла. Для такого вызова служит встроенная функция **вызвать()**, имеющая следующий формат обращения:

вызвать(имя вызываемой программы, строка параметров)

Информационное взаимодействие с вызываемой программой может осуществляться либо через текстовые файлы, либо через строку параметров, задаваемую вторым аргументом обращения.

Частным случаем вызова внешней операции является обращение не к загрузочному файлу (т.е. к файлу типа СОМ или EXE), а ко встроенной команде операционной системы (типа COPY или REN) или к пакетному файлу (типа BAT). Для этого обращение имеет специальную форму:

вызвать('COMMAND.COM', '/C команда')

Например, для удаления файлов вы можете выполнить следующее обращение:

вызвать('COMMAND.COM', '/C DEL *.*')

Работа с каталогами файлов

Доступ к файловой системе требует возможностей управления каталогами файлов. Эти возможности представлены в языке МАСТЕР следующими встроенными функциями:

Файлы(шаблон имен файлов) --> текст

Каталог() --> имя текущего каталога

Каталог(имя каталога)

Каталог(имя устройства)

Среда(имя параметра) --> строковое значение параметра

Функция **Файлы()** возвращает в качестве значения текст, в строках которого перечисляются имена файлов, совпадающих с шаблоном, заданным аргументом **шаблон имен файлов**. Шаблон имен задается по обычным правилам операционной системы.

Функция **Каталог()** в зависимости от вида своего аргумента изменяет либо текущий каталог (если аргумент имеет вид A:), либо текущий дисковод (если аргумент имеет вид A:\DIR), а при отсутствии аргумента эта функция выдает строку – имя текущего каталога.

Функция **Среда()** позволяет запросить значение любого из параметров, установленных в среде операционной системы командой **SET**.

8.8. Организация диалога и интерактивной среды

Как уже говорилось, рамки обладают тремя нераздельными свойствами. Во-первых, они являются *носителями* (хранилищами) информации, т.е. выступают в роли переменных и информационных структур в традиционной технологии программирования. Во-вторых, рамки имеют готовый внешний вид, т.е. обеспечивают *визуализацию* данных, не требуя от программ почти никаких действий по формированию внешнего изображения этих данных. В то же время программа может простыми операциями гибко управлять этим внешним видом. Наконец, в-третьих, каждая рамка связана определенным диалоговым процессором, и тем самым их совокупность представляет собой *активную диалоговую среду*. Таким образом, не приступая к программированию, одной только комбинацией рамок можно создать информационную структуру с достаточно мощными интерактивными возможностями.

Стандартные, программируемые глобальные и программируемые локальные действия клавиш

Поскольку пользователь взаимодействует с компьютером в системе МАСТЕР только через клавиатуру, то центральным вопросом инструментальной технологии системы является вопрос о семантике клавиш: т.е. о способе реагирования системы на нажатия клавиш. Этот способ определяется в МАСТЕРЕ с помощью

трех понятий: стандартное действие клавиши, программируемое глобальное действие клавиши и программируемое локальное действие клавиши.

Стандартным действием клавиши называется та операция, которая встроена в диалоговый процессор (текстовый, табличный, графический, рамковый) в ядре системы МАСТЕР. Так, например, клавиша {ВВЕРХ} во всех диалоговых процессорах обладает одинаковым стандартным действием – перемещает курсор на одну позицию вверх. Это действие называется стандартным, поскольку оно запрограммировано именно в ядре системы МАСТЕР, а не на языке программирования Мастер.

Программируемым глобальным действием клавиши называется программа на языке Мастер, располагающаяся в функциональной рамке, имя которой совпадает с обозначением клавиши. Такое действие может быть связано не со всякой клавишей, а только с функциональной. Срабатывание этой программы происходит в те моменты, когда нажимается соответствующая функциональная клавиша. Зоной действия такого программного определения клавиши является то поддерево рамок, в корне которого находится функциональная рамка. Это происходит потому, что поиск рамки по имени нажатой функциональной клавиши производится по общим правилам поиска имен. Следовательно, если клавиша нажата в некоторой рамке, поднимаясь от которой из корневой рамке, можно найти функциональную рамку с именем этой клавиши, то функция будет найдена и сработает. В противном случае клавиша не будет иметь программируемого глобального действия. Таким образом, можно в разных частях рамковой структуры создавать различные определения глобальных действий клавиш. Поскольку поиск любого имени всегда завершается в специальной рамке – библиотеке диалоговой оболочки – функциональные рамки, размещенные в этой библиотеке, будут доступны из любого места рамковой структуры. Такие определения будут действительно глобальными. Именно таким образом запрограммирована, скажем, клавиша {Ф10}, обеспечивающая вход в меню МАСТЕРа.

Программируемым локальным действием клавиши называется такая операция, которая программируется на языке Мастер и размещается не в отдельной рамке, а связывается с данной клавишей непосредственно в заданной рамке. Эта формула срабатывает локально, т.е. при нажатии клавиши только в той рамке в которой установлена связь. В другой рамке с той же самой клавишей можно связать совсем другое локальное действие. Локальные действия можно связывать как с отдельными клавишами, так и с группами клавиши. Например, можно запрограммировать единую реакцию на нажатие любой буквенно-цифровой клавиши или любой функциональной клавиши, или вообще ли-

бой клавиши. Это связывание формулы с группой клавиш также действует только в пределах одной рамки.

Для работы с формулами, служащими локальными действиями клавиши, в языке Мастер имеется набор встроенных функций. Это те же самые функции, что и для формул в электронных таблицах, но формат обращения к ним в данном случае несколько иной:

УстФормулу(рамка, код клавиши, текст формулы)

Формула(рамка, код клавиши) --> текст формулы

ЕстьФормула(рамка, код клавиши) --> 0/1

Аргумент **код клавиши** в этих функциях должен задаваться числовым кодом, который удобнее всего получать обращением к функции КодКлавиши(). Например, для того чтобы связать с клавишей {ОТКАЗ} формулу, выполняющую выход из рамки, следует выполнить обращение:

УстФормулу(рамка, КодКлавиши("{ОТКАЗ}"), "Выйти()")

В случае, когда нужно связать формулу не с отдельной клавишей, а с целой группой клавищ, в качестве кода должна использоваться одна из трех мнемонических констант: **ЗНАКОВЫЕ**, **ФУНКЦИОНАЛЬНЫЕ**, **ВСЕ**.

Общая семантика нажатия клавиши в системе МАСТЕР определяется следующим образом. При нажатии клавиши внутри некоторой рамки прежде всего проверяется, нет ли для этой клавиши в данной рамке локального действия. Если есть индивидуальное локальное действие, то выбирается именно оно, в противном случае срабатывает групповое локальное действие. Срабатывание локального действия рассматривается как фильтр для дальнейшей работы клавиши. Если локальное действие отсутствует или если оно выработало значение 1 (в качестве возвращаемого значения), то считается, что фильтр разрешает действие клавиши. После этого выбирается либо стандартное действие, если оно имеется, либо программируемое глобальное действие. Обратите внимание, что срабатывают не оба, а только одно из этих действий, причем приоритет принадлежит стандартному действию перед программируемым глобальным.

Таким образом, для строгого понимания семантики работы клавиши требуется знание того, какие клавиши обладают стандартными действиями, а какие – нет. В первой части книги при описании диалоговых возможностей МАСТЕРа различия между ними не проводилось, хотя некоторые из описанных там действий клавищ являются стандартными (встроеными в ядро МАСТЕРа), а другие запрограммированы на языке Мастер и размещены в библиотеке диалоговой оболочки. Для конечного

пользователя это различие совершенно несущественно, но для программиста будет полезен перечень тех клавиш, которые обладают именно стандартными действиями. Этот перечень приведен в табл. 8.1.

Таблица 8.1. Перечень встроенных клавиш МАСТЕРа

диалоговый процессор	Перечень клавиш, обладающих в этом процессоре стандартными действиями
Рамковый	{ВВЕРХ} {ВНИЗ} {ВЛЕВО} {ВПРАВО} {УПР-ВЛЕВО} {УПР-ВПРАВО} {ЗАГ-ВВЕРХ} {ЗАГ-ВНИЗ} {ЗАГ-ВЛЕВО} {ЗАГ-ВПРАВО} {ДОП-Ф10} {Ф9} {Ф1} {Ф2}
Табличный	{ВВЕРХ} {ВНИЗ} {ВЛЕВО} {ВПРАВО} {ПРЕД} {СЛЕД} {ВВОД} {УПР-ВЛЕВО} {УПР-ВПРАВО} {ЗАГ-ВВЕРХ} {ЗАГ-ВНИЗ} {ЗАГ-ВЛЕВО} {ЗАГ-ВПРАВО} {знаки} {ДОП-Ф10} {Ф9} {Ф1} {Ф2} {Ф3} {Ф4}
Текстовый	{ВВЕРХ} {ВНИЗ} {ВЛЕВО} {ВПРАВО} {ПРЕД} {СЛЕД} {ВВОД} {НАЧАЛО} {КОНЕЦ} {ВСТ} {ЗАГ-ВВЕРХ} {ЗАГ-ВНИЗ} {ЗАГ-ВЛЕВО} {ЗАГ-ВПРАВО} {УПР-ВЛЕВО} {УПР-ВПРАВО} {УДЛ} {ЗАГ-УДЛ} {ОТМЕНА} {ЗАГ-ОТМЕНА} {Ф1} {Ф2} {Ф3} {Ф4} {Ф5} {Ф6} {Ф7} {Ф8} {Ф9} {УПР-Ф1} {УПР-Ф2} {УПР-Ф3} {УПР-Ф4} {УПР-Ф5} {УПР-Ф6} {УПР-Ф7} {УПР-Ф8} {ЗАГ-Ф2} {ЗАГ-Ф3} {ЗАГ-Ф4} {ЗАГ-Ф7} {ЗАГ-Ф8} {ЗАГ-Ф10} {ДОП-Ф1} {ДОП-Ф2} {ДОП-Ф3} {ДОП-Ф4} {знаки} {ДОП-Ф10}
Графический	{ВВЕРХ} {ВНИЗ} {ВЛЕВО} {ВПРАВО} {ПРЕД} {СЛЕД} {ВВОД} {ДОП-Ф10} {Ф9} {Ф1}

Фильтры встроенных функций

Некоторые программируемые действия бывает удобно определять в рамках, связывая их не с нажатиями тех или иных клавиш, а с выполнениями каких-то действий над рамкой: входом в рамку и выходом из нее, уничтожением рамки, считыванием ее с диска и т.п. Такую возможность дает та же встроенная функция УстФормулу(), которая задает формулы локальных действий клавиш в рамках. Формат обращения к функции УстФормулу() в данном случае остается тем же самым, но в качестве кода клавиши (второго аргумента) должен быть задан код одной из фильтруемых встроенных функций.

Фильтруемыми являются далеко не все функции языка Мастер, а только те из них, которые выполняют наиболее важные действия над рамками. Каждой из таких фильтруемых функций сопоставлено мнемоническое обозначение, которое и должно использоваться в обращении к функции УстФормулу() во втором аргументе (табл. 8.2).

Таблица 8.2. Мнемонические константы для фильтрации функций

Встроенная функция	Мнемоническое обозначение кода для установления фильтра
Войти()	ВХОД
Выйти()	ВЫХОД
Открыть()	ОТКРЫТИЕ
Закрыть()	ЗАКРЫТИЕ
Считать()	СЧИТЫВАНИЕ
Создать()	СОЗДАНИЕ
Очистить()	ОЧИСТКА
Присв()	ПРИСВАИВАНИЕ

Особое значение представляют собой фильтры встроенных функций Войти() и Выйти(). Передвижение по информационной среде МАСТЕРа происходит исключительно на основе этих двух функций. Независимо от того, каким образом выглядит на экране это передвижение, оно происходит сбалансированными обращениями к этим двум функциям. Невозможно перейти изнутри

рамки вовне ее, миновав функцию Выйти(). Даже если этот выход делается неявно – обращением к функции УстТек() или Войти(), уводящим сразу к некоторой удаленной рамке среды, система МАСТЕР в действительности отрабатывает последовательно все выходы из рамок и входы в них, требующиеся для достижения нужной точки. Таким образом, фильтры на функции Войти() и Выйти() могут рассматриваться как своеобразные "сторожи", контролирующие передвижение пользователя по информационной среде.

Заметим, в частности, в простые (строковые и числовые) рамки войти невозможно. Тем не менее считается, что в моменты ввода с клавиатуры значений в эти рамки происходит неявный вход (перед присваиванием) и неявный выход (после присваивания).

Еще одной важной с этой точки зрения функцией является функция Считать(). Фильтр такой функции срабатывает сразу после считывания рамки с диска. Эти фильтры можно использовать для того, например, чтобы сразу после считывания рамки запускать какую-то программу, связанную с ней, либо загружать какие-то дополнительные рамки, требующиеся для ее использования. Отметим, что фильтр функции Считать() срабатывает независимо от того, на какой глубине по рамковой иерархии находится считываемая рамка.

Возможность определения фильтров на встроенные функции имеется не только из языка Мастер. Она представлена также в рамковом меню МАСТЕРа. Для того чтобы воспользоваться ею нужно установить курсор на ту рамку, для которой определяется фильтр, и выполнить команду

{Ф10} Рамка Действие Функция

Эта команда вызовет специальное меню, перечисляющее все встроенные функции, которые подлежат фильтрации. Выбрав одну из функций, вы должны задать формулу фильтра точно также, как и при определении локального действия клавиши.

Программирование макроклавиш

Помимо функциональных и буквенно-цифровых клавиш, значения которых определяются согласно описанной семантике через стандартные, глобальные и локальные действия, на клавиатуре МАСТЕРа имеются еще клавиши с совершенно иным значением – так называемые **макроклавиши**. Макроклавиши – это нажатия буквенных клавиш с модификатором {ДОП}. Действия этих клавиш определяются совсем иным механизмом – макропоследовательнос-

тями. **Макропоследовательностью** может служить произвольный текст. Если связать такую макропоследовательность с некоторой макроклавишей, то нажатие этой макроклавиши МАСТЕР воспринимает так, как если бы были нажаты те клавиши, изображения которых находятся в макропоследовательности. При этом буквенно-цифровые и знаковые клавиши изображаются **именами** по этическими буквами, цифрами и знаками, а функциональные клавиши изображаются своими мнемоническими именами, заключенными в фигурные скобки. Разбиение текста макропоследовательности на строки не имеет никакого значения, и если для работы в текстовом процессоре вам потребуется ввести в макропоследовательность переход на новую строку, то нужно вставить в этом месте имя клавиши {ВВОД}.

Для работы с макроклавишами через программы на Мастере имеются следующие функции:

ОпрМакро(буква) --> текст макропоследовательности
ОпрМакро(буква, текст макропоследовательности)
Исполнить(текст макропоследовательности)

Функция ОпрМакро() имеет один или два аргумента. Подобно остальным функциям такого рода при обращении с одним аргументом она возвращает значение – текст макропоследовательности, связанной с указанной буквой, а при обращении с двумя аргументами она задает для буквы новый текст макропоследовательности. Аргумент **буква** должен задаваться строкой, состоящей из одной буквы, причем клавиша-модификатор {ДОП} в этих обращениях явно не указывается, но подразумевается, что макропоследовательность определяется именно для клавиши {ДОП-буква}.

Функция Исполнить() позволяет выполнить текст макропоследовательности, не связывая его ни с какой макроклавишей. Ее использование может быть удобно в тех случаях, когда требуемое для программы действие легче выразить цепочкой нажатий клавиш, нежели обращениями к функциям на языке Мастер. Вследствие достаточной мощности языка Мастер такая потребность встречается крайне редко, однако иногда без нее обойтись невозможно. Например, в языке нет функции, позволяющей установить для электронной таблицы направление ввода данных. Это можно сделать только через клавишу {ЗАГ-ПЛЮС}, и поэтому программа должна будет прибегнуть к функции Исполнить(), например, следующим образом:

Войти(таблица);
Исполнить(" {ЗАГ-ПЛЮС} {ВПРАВО} ");
Войти(назад)

Перемещение курсора

При программировании операций, связанных с диалогом, требуется возможность программно перемещать курсор по информационной среде. Для этого существуют функции нескольких групп. Первая группа функций связана с перемещением курсора по рамкам:

УстТек(рамка)
УстТек(составная рамка, имя рамки)
Войти(рамка)
Выйти()

Функция УстТек() в первом варианте обращения устанавливает курсор на указанную рамку. Во втором варианте она не изменяет положения курсора, а воздействует на некоторую удаленную составную рамку, делая в ней текущей рамку с указанным именем, что скажется либо на последующем входе в эту составную рамку, либо на логических позициях создаваемых в ней новых рамок, либо на результате обращения к функции ТекРамка(составнаяРамка).

Функция Войти() переводит курсор внутрь указанной рамки. Функция Выйти() выводит курсор из охватывающей рамки и делает ее текущей.

Еще одна функция – ТабКурсор() – связана с перемещением курсора по ячейкам электронной таблицы. Она имеет три разных варианта обращения:

ТабКурсор() --> указатель ячейки
ТабКурсор(табличная рамка) --> указатель ячейки
ТабКурсор(указатель ячейки)

В первом варианте обращения возвращаемым значением является указатель той ячейки охватывающей рамки, на которой находится курсор. Если таблица находится в состоянии выделения диапазона, сделанного через клавишу {Ф3}, то возвращается указатель всего выделенного диапазона.

Второй вариант обращения производит то же самое действие, но относится не к охватывающей, а к произвольной табличной рамке, заданной своим указателем. В этом варианте возвращается положение табличного курсора для указанной рамки.

Третий вариант обращения не возвращает значения, а устанавливает табличный курсор на указанную ячейку. Следует обратить внимание на то, что от предыдущего варианта обращения этот отличается только типом аргумента. Если при запросе положения курсора аргумент должен быть типа указателя рамки

го при установке положения курсора – типа указателя диапазона.

Для управления курсором внутри текстовой рамки можно использовать функцию Параметр(). Она позволяет запрашивать и устанавливать значения различных параметров текста. Два параметра – с номерами 0 и 1 – управляют позицией курсора в текстовой рамке. Параметр 0 – это положение курсора по горизонтали, а параметр 1 – положение по вертикали. Таким образом, если вы хотите узнать, в какой строке находится курсор в текстовой рамке, то следует воспользоваться обращением:

Параметр(текст, 1)

а для установки курсора в крайнюю левую позицию текущей строки нужно выполнить обращение

Параметр(текст, 0, 1)

Еще одна функция позволяет запросить у пользователя указания того или иного объекта в информационной среде МАСТЕРа. Эта функция имеет следующий формат обращения:

указать(строка запроса, тип объекта) --> указатель

Первый аргумент этой функции задается строкой, содержащей вопрос для пользователя, отвечая на который он должен указать некоторый объект. Второй аргумент – тип объекта – задает мнемонической константой тип того объекта, который должен быть указан. Допустимыми здесь мнемоническими константами являются следующие: РАМКА, ДИАПАЗОН, ТЕКСТ. Поэтому указание (т.е. нажатие клавиши {ВВОД}) будет воспринято только в том случае, когда курсор стоит соответственно на рамке, на ячейке таблицы или на строке текста. Если же пользователь нажмет {ВВОД} на объекте иного типа, то система будет оставаться в режиме указания (это иногда вводит пользователя в заблуждение, поскольку система в состоянии указания не реагирует на обычные клавиши редактирования и на многие из функциональных клавиш; возникает впечатление, система как бы "зависает").

Визуализация информационной среды при исполнении программ

При работе Мастер-программ не производится непосредственного показа вызванных ею изменений в информационной среде

д. До тех пор пока программа не закончила своей работы, экране остается неизменное изображение, которое уже не соответствует текущему состоянию информационной среды. Только при завершении работы программы и передаче управления пользователю экран будет автоматически обновлен в соответствии полученным состоянием среды. Однако в некоторых программах требуется уже по ходу работы отображать изменения на экране. Так, например, если производится сложная выборка записей базы данных с постепенным заполнением строк таблицы, то хватительно показывать состояние таблицы после выборки каждой очередной строки — тогда пользователю будет видно, что программа работает, а не зависла и не зациклилась.

Для того чтобы показать на экране истинное текущее состояние информационной среды, имеется функция

Показать()

Вызывается эта функция без аргументов, а действием ее является просто обновление экрана. В случае, если никаких изменений в рамках, видимых на экране, не произошло, функция не производит никаких действий. Поэтому ее срабатывание в этом случае никак не задерживает работу программы.

Еще одна особенность визуализации рабочего поля на экране связана с интерпретацией макропоследовательностей клавиш, которая происходит либо при нажатии пользователем макроклавиши, либо при обращении к функции Исполнить() из формулы на языке Мастер. По умолчанию эта интерпретация осуществляется "вслепую", т.е. без непосредственной визуализации промежуточных результатов. Однако если предварительно было выполнено обращение к функции Эхо(1), то это переведет МАСТЕР в режим, в котором интерпретация макропоследовательности будет выполняться с полным отображением всех действий. Естественно, что перед началом работы функции Исполнить() в таком режиме неявно будет совершена операция Показать() для приведения экрана в соответствие с текущим состоянием рабочего поля.

Обращение к функции Эхо(0) позволяет выключить режим такого непосредственного отображения, а обращение к функции Эхо() без аргументов позволяет запросить текущее значение режима отображения.

Функции диалога "вопрос-ответ"

При работе всякой программы довольно часто требуется задать пользователю какой-нибудь вопрос или вывести то или

иное сообщение. В МАСТЕРе для такого рода диалога используются две верхние строки экрана — поле сообщения и поле значения. Набор встроенных функций, обеспечивающих такой диалог, включает следующие:

```
Сообщить(строка сообщения)
СообщитьЗнач(строка сообщения)
ЗапросЧис(строка запроса [, значение по умолчанию])
    --> число
ЗапросСтр(строкаЗапроса [, значение по умолчанию])
    --> строка
ЗапросТекст(строкаЗапроса [, значение по умолчанию])
    --> текст
Звук(тон, длительность, ...)
Меню(пункт, объяснение, действие, ...)
    --> значение действия
Клавиша([ждать])
    --> код клавиши
```

Функция Меню() имеет нефиксированный набор аргументов, состоящий из групп по три аргумента в каждой. Такая тройка аргументов задает один из пунктов меню, причем первый аргумент в тройке задает имя пункта, второй аргумент — его пояснение, а третий — то действие, которое должно быть выполнено при выборе пользователем этого пункта. Возвращаемым значением обращения к Меню() является значение того действия, которое было выбрано.

Средства отладки программ на Мастерс

Система программирования на языке Мастер имеет интерпретирующий характер. Каждая формула при вводе подвергается некоторой предварительной обработке — компиляции, в процессе которой, однако, могут быть обнаружены только самые простейшие ошибки: излишние или отсутствующие скобки, неправильные операции. Все остальные ошибки могут быть вскрыты только при запуске программ.

Диалоговый и интерпретирующий характер языка сам по себе в значительной степени облегчает отладку. Так, простейшим средством отладки является вставка в программу операций распечатки тех или иных значений с помощью обращения к функции Сообщить(). Поскольку изменения вносятся в программы очень легко и не требуется никаких затрат времени на компиляцию и сборку программы, то такие отладочные действия оказываются весьма эффективными.

СИСТЕМА**УПРАВЛЕНИЯ****БАЗАМИ ДАННЫХ**

Имеется встроенная функция, предназначенная для более глубокого изучения хода работы программ. Эта функция имеет следующие варианты обращения:

Трассировка(уровень трассировки [, текстовая рамка])
--> уровень трассировки

Аргумент **уровень трассировки** может принимать значения 0, 1 и 2. Уровень 0 означает отсутствие трассировки. Уровень 2 означает отладочный уровень трассировки, при котором прослеживаются обращения ко всем встроенным и определяемым функциям, а также возвращаемые значения этих обращений. Отслеживание этих обращений реализуется либо в двух верхних строках экрана, либо запоминается в текстовой рамке, указанной вторым аргументом.

Уровень трассировки, равный 1, означает, что запоминаться в текстовой рамке должны только нажатия клавиш. Этот уровень трассировки нужен не для отладки, а для реализации режимов обучения. С помощью трассировки можно запоминать действия пользователя, чтобы в дальнейшем повторять их в виде макросов последовательностей.

9.1. Основные понятия СУБД МАСТЕРа**Записи и множества записей**

Основным понятием СУБД является **запись**. Запись состоит из полей, которые могут содержать значения числового, строкового, рисункового и текстового типов. Одна база данных может содержать записи разных типов, отличающихся количеством, именами и типами полей. Так, например, создавая базу данных о научном институте, следует завести в ней такие типы записей: **сотрудник**, **отдел**, **тема**. Каждый из этих типов записей будет характеризоваться собственными атрибутами (полями). Скажем, записи типа **сотрудник** могут содержать поля **фамилия**, **ученаяСтепень**, **оклад**, **номерОтдела**, а полями записей типа **отдел** будут совсем другие – **номерОтдела**, **название**, **руководитель**.

Таким образом, в соответствии со своими типами записи естественным образом объединяются в совокупности – **множества записей** разных типов. Поскольку каждую запись можно представить себе в виде строки, то их совокупности – множества записей – представляются как таблицы. В реляционных СУБД множества записей так и называются таблицами. В терминологии системы МАСТЕР во избежание смешения с электронными таблицами мы будем называть их **множествами**.

Множества записей характеризуются двумя аспектами: во-первых, какие именно записи данного типа в них включены, и во-вторых, в каком порядке взяты эти записи. Особую роль играют такие множества, которые включают *все* записи данного типа. Они будут называться **полными множествами** или **порядка-**

ми. Остальные множества, включающие какие-либо выборки записей данного типа, будут называться **подмножествами**.

В языке Мастер имеются почти 40 встроенных функций, составляющих функционально полный базис для обработки записей по порядковым, подмножеством. Функциональная полнота означает в данном случае то, что комбинациями этих функций можно выполнить самые разнообразные действия с базой данных, начиная с обработки любых ее элементов и кончая управлением всей структурой. Так, с помощью этих функций можно и создать, модифицировать, и проанализировать базу данных, можно выполнить доступ к информации в базе, создавая, изменения и уничтожая записи, их поля, множества и подмножества записей, и т.д.

О концепции СУБД МАСТЕРа

Концепция СУБД МАСТЕРа является нетрадиционной в сравнении с общезвестными моделями. Она содержит в себе определенные черты реляционной идеологии, но отнюдь не совпадает ни с реляционной алгеброй, ни с реляционным исчислением. Если попытаться двумя словами обозначить сущность этой концепции, то ее, пожалуй, следует назвать *процедурно-реляционной*. Это означает, что, хотя в основе структурных связей данных лежит обычный реляционный принцип сопоставления значений полей, но обработка этих связей представлена не крупномасштабными операциями над целыми таблицами типа реляционных проекций, соединений, произведений и пересечений, а операциями поэлементной последовательной обработки. Если потребуется, из этих элементарных операций можно легко собрать обычные крупномасштабные реляционные операции, но можно сконструировать и любые другие операции сетевого, иерархического, теоретико-множественного и других типов.

Попробуем в общих чертах сопоставить концепцию СУБД МАСТЕРа с общезвестной реляционной концепцией. Основой реляционного механизма является понятие таблицы, столбцы которых (домены) соответствуют полям записей МАСТЕРа, а строки – единичным записям. Реляционная алгебра предлагает для работы с таблицами набор операций, определенных над таблицами как целыми. Поисковый запрос в реляционной алгебре представляется суперпозицией операций, преобразующих исходный набор таблиц в некоторую новую таблицу, являющуюся выборкой требуемой информации.

В СУБД МАСТЕРа аналогом таблиц являются множества записей, а инструментальный язык предоставляет программисту набор операций над этими множествами. В отличие от реляционных операций Мастера рассматривают таблицы не как неделимые об-

ъекты, а как многострочные структуры, состоящие из отдельных записей, и обеспечивают передвижение от строки к строке, индивидуальную вставку, уничтожение и модификацию строк. Тем самым их следует рассматривать как относительно низкоуровневые операции, которые вследствие этого являются более гибкими и эффективными, но менее аналитичными и абстрактными.

Эффективность этих операций по сравнению с реляционными обусловливается тем, что последние из-за своей крупномасштабности вынуждены подчас порождать промежуточные структуры большого размера только потому, что они возникают в реляционной формальной семантике запроса. Практически же всегда можно обойтись без явного порождения таких структур за счет выполнения запроса процедурными вычислениями, создающими только требуемый конечный результат без явного порождения промежуточных реляционных таблиц.

Таблицы с указателем текущей строки

Если единственным основным понятием чистой реляционной идеологии является просто *таблица*, то таким же единственным понятием процедурно-реляционной идеологии является *таблица с указателем текущей строки*. Если операции реляционной алгебры преобразуют несколько целых таблиц в новую целую таблицу, то операции процедурно-реляционной идеологии, как правило, не порождают новой таблицы, а либо перемещают указатель текущей строки в этой таблице, либо делают локальное изменение в ней: вставляют новую строку, уничтожают или модифицируют существующую. Есть, впрочем, и такие операции, которые подобно реляционным порождают целые таблицы. Полный набор процедурно-реляционных операций подразделяется на операции следующих категорий:

навигационные операции – перемещают указатель текущей строки по таблицам без модификации содержимого таблиц;

модифицирующие операции – выполняют локальные изменения таблицы в текущей позиции указателя;

множественные операции – выполняют подобно реляционным операциям действия сразу над целыми множествами записей.

Операции процедурно-реляционного базиса

Не вдаваясь в детали, опишем те функции языка Мастер, которые выполняют основные операции перечисленных категорий.

1) Для *навигации* по таблицам служат функции *Найтиначало()*, *НайтиКонец()*, *НайтиСлед()*, *НайтиПред()*, *Найти()*.

Первые две функции устанавливают указатель на начало и конец таблицы, две другие позволяют перемещать указатель на следующую и предыдущую строки таблицы, последняя функция обеспечивает прямой поиск записи по заданному значению поля.

2) *Модифицирующие операции* представлены функциями СоздатьЗапись(), ИзменитьЗапись(), УничтожитьЗапись(), которые выполняют соответственно вставку новой записи во множество записей, изменение содержимого текущей записи, удаление текущей записи. К этой же группе следует отнести функции ВзятьЗапись() и Поле(), которые позволяет выбрать запись или поле записи из базы данных в рамку или электронную таблицу МАСТЕРа, чтобы обрабатывать там информацию обычными процедурными средствами языка Мастер (числовыми, строковыми, рисунковыми, текстовыми функциями и операциями).

3) Для создания новых множеств (таблиц) служит функция СоздатьМнож(). Ее аргументом является рамка, содержащая описание создаваемой таблицы, т.е. имена и типы ее полей. Другая функция – ВзятьПорядок() – позволяет создать новую таблицу по уже существующей, которая будет не количеством, а содержимым записей, а порядком их расположения в таблице (во множестве). Еще один способ образования новых таблиц – это функция Ограничить(), которая вырезает из существующей таблицы некоторый набор строк, соответствующий диапазону значений того поля, по которому эта таблица упорядочена. Кроме того, имеется набор функций для работы с подмножествами: функция ПустоМнож() позволяет создать пустое подмножество определенного типа, а функции ВключитьЗапись() и ИсключитьЗапись() позволяют изменять состав подмножеств.

Этот набор функций представляет собой концептуальную основу СУБД МАСТЕРа. В действительности имеется еще много дополнительных функций, требующихся по соображениям уже функциональной, а технологической полноты. Рассмотрим внимание на схематических примерах принципы использования основных функций.

Последовательный перебор записей

Начнем с наиболее важного действия – последовательного перебора записей во множестве, который будет требовать постоянно в самых разнообразных действиях.

ПРИМЕЧАНИЕ. Можно сказать, что именно полный последовательный перебор таблиц позволяет из элементарных процедур реляционных операций сделать крупномасштабные операции реляционного типа. Заметим также, что понятие "полный перебор", за которым, вообще говоря, скрываются значительные затра-

времени, вовсе не означает полного перебора всех записей базы данных. Речь идет о переборе записей некоторых множеств, которые при правильной организации работы являются небольшими выборками из полных множеств записей.

Процедурная схема последовательного перебора, выраженная на языке Мастер, такова:

```
НайтиНачало(во множестве M);  
Повтор(ВзятьЗапись(из множества M в рамку),  
    обработать информацию, выбранную в рамку;  
    НайтиСлед(во множестве M)  
)
```

Выполнение подобной формулы начинается с установки указателя текущей строки на начало множества с помощью функции НайтиНачало(). Перебор записей выполняется с помощью функции НайтиСлед(), причем выборка очередной полученной записи в рамку делается с помощью функции ВзятьЗапись(). Эта же функция служит и для того, чтобы определить, когда заканчивать цикл перебора: при достижении конца таблицы она выдаст значение 0, чем и завершит работу цикла, в котором обращение к ней является условием продолжения.

Реляционное соединение таблиц

Основной идеей реляционной идеологии является то, что для представления связей между данными не требуется никаких специальных технологических структур (таких, как "наборы" из предложений комитета КОДАСИЛ). Связи между таблицами реализуются просто совпадением значений определенных полей записей (столбцов таблиц). Так, например, если одна таблица содержит фамилии сотрудников и коды отделов, в которых они работают, а вторая таблица – коды отделов и их названия, то в результате соединения этих таблиц по полям, содержащим коды отделов, будет получена новая таблица, содержащая фамилии сотрудников и соответствующие им названия отделов.

Рассмотрим процедурную схему на Мастере, которая реализует такую реляционную операцию. Пусть даны две таблицы сотрудник и отдел, содержащие пары полей (фамилия, кодОтдела) и (кодОтдела, названиеОтдела) соответственно, и требуется получить таблицу сотрудникОтдел, содержащую пару полей (фамилия, названиеОтдела). Это может быть сделано следующей процедурой:

```

СоздатьМнож(сотрудникОтдел);
НайтиНачало(сотрудник);
Повтор(ВзятьЗапись(сотрудник),
    Найти(такой Отдел, в котором поле кодОтдела равно
        полю кодОтдела во взятой записи Сотрудник
    );
ВзятьЗапись(отдел);
Присв(сотрудникОтдел.фамилия, сотрудник.фамилия);
Присв(сотрудникОтдел.названиеОтдела,
    отдел.названиеОтдела
);
СоздатьЗапись(сотрудникОтдел);
НайтиСлед(сотрудник)
)

```

Принципиальными моментами этой процедурной схемы являются следующие:

а) идею целостности множества реализует последовательный перебор записей во множестве **сотрудник**, осуществляемый в обычной схеме перебора с помощью функций **НайтиНачало()**, **ВзятьЗапись()**, **НайтиСлед()**;

б) реляционная связь множества **сотрудник** со множеством **отдел** осуществляется операцией поиска **Найти()**;

в) слияние информации из записей разных типов для образования строки в выходной таблице и создание этой строки выполняют обращения к **Присв()** и к **СоздатьЗапись()**.

Нетрудно реализовать на языке Мастер обобщенные функции, выполняющие все реляционные операции над множествами записей. Однако более эффективным подходом при создании прикладных систем является такой, при котором эти операции реализуются программистом каждый раз заново – в соответствии с приведенными общими схемами, но с учетом конкретных особенностей структуры информации. Это позволит избавиться от создания ненужных промежуточных таблиц. Кроме того, при таком подходе можно и не ограничивать себя только реляционной идеологией.

9.2. Технологические понятия СУБД МАСТЕРа

До сих пор мы рассматривали абстрактные понятия, которые составляют концептуальную идеологию СУБД МАСТЕРа. Имеются, кроме того, понятия и другого рода – технологические. Они связывают абстрактные понятия с такими практическими задачами, как выбор файловой структуры, буферизация, упаковка данных и др.

Буферизация

Информационный обмен между базой данных и рамковой средой осуществляется через специальную системную область оперативной памяти, называемую **буфером базы данных**. Само по себе содержимое буфера недоступно программисту. В него МАСТЕР выбирает с диска некоторую часть базы данных, к которой в данный момент делаются обращения. Ни одна из операций над базой данных не может быть выполнена, если буфер еще не создан. Поэтому операция создания буфера называется **открытием базы данных**, а уничтожение буфера – ее **закрытием**. Вообще над буфером в языке Мастер определены следующие операции:

```

ОткрытьБазу(размер буфера)
ЗакрытьБазу()
БазаОткрыта() --> 0/1
СохранитьБазу()

```

Размер буфера задается вторым аргументом обращения к функции **ОткрытьБазу()**, измеряющимся в килобайтах, и может быть установлен по вашему усмотрению от 5 до 200 Кбайт. Чем больше этот размер, тем большая часть базы может выбираться в него и тем быстрее будет осуществляться работа с ней, но тем меньше места останется для обычных рамок и содержащейся в них информации.

Очень важной является функция **СохранитьБазу()**. Она выполняет передачу всей модифицированной информации из буфера в базу данных. Формально эта операция не обязательна, поскольку передача информации из базы в буфер и обратно осуществляется системой автоматически: при необходимости считать в буфер новую информацию из него вытесняется какая-то другая информация и записывается в базу; кроме того, при закрытии базы данных все содержимое буфера автоматически возвращается в базу. Однако при возникновении каких-либо сбоев (скажем, сбросе питания компьютера) информация может остаться в буфере. В таких случаях происходит потеря информации, а иногда даже нарушаются структурные связи внутри базы данных, и тогда она оказывается непригодной для дальнейшего использования (требует восстановления с помощью специальной программы). Во избежание этого ваша программа должна выполнять запись буфера в базу после каждой модифицирующей операции. Этого можно не делать только внутри циклов, чтобы не замедлять работу программы. Следует иметь в виду, что функция **СохранитьБазу()** записывает на диск только те блоки буфера, которые подверглись модификации. Поэтому лишние обращения к функции не повлияют на время работы.

Информационные области базы данных

Вся информация базы данных хранится на диске в файлах, имеющих специальную структурную организацию, которые называются **информационными областями** базы данных.

С каждой информационной областью связывается числовой код, использующийся во всех операциях, относящихся к этой области, — **код области**. Этот код используется, например, в операциях, создающих новые множества записей: им определяется, в какой области создавать множество. Кроме того, в информационных областях хранятся внутрисистемные ссылки на информацию этой области (уникальные ключи записей, индексов и разнообразных дескрипторов), будут содержать код области как часть адреса информации. Поэтому коды областей играют важную роль в организации баз данных — они служат внутренними идентификаторами областей в базе данных. Код области задается в момент ее создания и может быть изменен в дальнейшем. Для работы с областями в языке Мастер имеется следующий набор встроенных функций:

```
СоздатьОбласть(код области, имя файла)
ОткрытьОбласть(имя файла [, разрешение записи])
    --> код области
ЗакрытьОбласть(код области)
ОбластьОткрыта(код области) --> 0/1
```

Аргумент **код области** во всех этих функциях должен быть целым числом от 0 до 10. Аргумент **разрешение записи** может быть равен 0 — тогда любые модифицирующие операции в открываемой области будут запрещены; 1 — разрешается модификация области; при отсутствии аргумента модификации разрешено. Функция открытия области может сработать неуспешно, если области с указанным именем не существует. Тогда ее возвращаемым значением будет отрицательное число.

Способы упаковки полей в записях

При создании каждого типа записей можно определить не только типы полей, но и способы упаковки полей в теле записи. Способ упаковки задается для полей простых типов — числовых и строковых. Текстовые и рисковые поля хранятся всегда одинаковым способом.

Каждая запись представляет собой на физическом уровне блок памяти фиксированной длины (зависящей от набора полей). Этот блок называется **телом записи**. Значения полей располагаются либо непосредственно в этом блоке, либо вне его, и тогда в блоке размещается 4-байтовый указатель значения.

Числовые поля всегда располагаются только в теле записи и занимают в нем один из четырех возможных размеров: 1 байт, 2 байта, 4 байта или 8 байт. Этим размерам соответствуют стандартные представления машинных чисел различной точности. Кроме того, имеются числовые поля, хранящиеся в форматах даты и времени, которые занимают по два байта.

Строковые поля могут располагаться либо вне, либо внутри тела записи. Если строковое поле располагается внутри тела записи, ему выделяется там фиксированное число байтов, задаваемое при определении поля. В противном случае строковое значение поля может занимать произвольный размер от 0 до 255 байт и располагаться вне тела записи. Такие строковые поля называются полями соответственно фиксированной и нефиксированной длины.

Текстовые и рисковые поля всегда хранятся вне тела записи, и на их размер не накладывается никакого ограничения.

Указатель записи и рамка-запись

Для доступа к записям в языке Мастер имеются значения специального типа — **указатели записей**. Они служат двум важным целям. Во-первых, указатель записи дает доступ к содержимому записей. Через указатель можно загрузить значения полей из базы в рамку или записать их из рамки в базу. Это делается функциями **ВзятьЗапись()** и **ИзменитьЗапись()**. Во-вторых, указатели записей связаны с теми или иными множествами записей. Они могут перемещаться от записи к записи по этому множеству с помощью функций **НайтиСлед()**, **НайтиПред()** или **Найти()** в соответствии с упорядоченностью множеств записей. Доступность множества по указателю позволяет применять к нему и такие операции, которые выполняются над множеством как целым. Поэтому в некоторых случаях указатель записи удобнее называть **указателем множества**. Формально же эти понятия эквивалентны.

В дальнейшем описании функций мы будем использовать для указателей записей три эквивалентные термины: **указатель записи**, **указатель множества**, **указатель типа записей**.

В отношении процедурной обработки указатели записей являются обычными значениями языка, подобными указателям рамок или диапазонов. Их точно так же можно присваивать в локальные переменные, передавать через аргументы функций, получать в качестве возвращаемых значений функций.

Особенность указателей записей проявляется при присваивании их рамкам. Рамка, в которую присвоен указатель записи, становится **рамкой-записью**. Прежде всего это означает, что

рамка становится составной и внутри нее автоматически создается набор рамок, соответствующих полям записи. С этого момента рамка может служить для выборки записей из базы в рамковую среду МАСТЕРа.

В отношении своей структуры такая рамка полностью аналогична обычной составной рамке, в которую можно входить и выходить, открывать и закрывать, брать внутри нее составляющие рамки-поля, совершать над этими рамками-полями любые рамковые операции и, главное, пользоваться их значениями — они-то и есть значения полей.

В то же время (помимо имеющейся в ней рамковой структуры) рамка-запись обладает значением типа "указатель записи". Это отличает рамку-запись от обычной составной, поскольку составные рамки никакими значениями не обладают. Наличие такого значения в рамке-записи означает, что к ней можно применить любую из тех функций, которые требуют в качестве аргумента указатель записи.

Некоторым функциям, таким, как `НайтиСлед()` или `НайтиПред()`, требуется только лишь сам указатель записи, и они перемещают этот указатель по множеству записей. Другие функции, такие, как `ВзятьЗапись()` или `ИзменитьЗапись()`, передают информацию между базой данных и рамковой средой МАСТЕРа. Для этого требуется не только указатель текущей записи в базе, но и рамка для размещения информации в рамковой среде. Рамка-запись как раз и выполняет обе функции. Будучи составной рамкой, она служит емкостью для информации в оперативной памяти, а через содержащийся в ней указатель записи она предоставляет доступ ко внешней памяти. Тем самым через рамку-запись обеспечивается связь между двумя информационными средами: оперативной памятью и базой данных.

Подобно тому, как указатель записи присваивается рамке, его можно присвоить и локальной переменной. Но локальная переменная содержит лишь указатель, но не имеет внутренней емкости, способной принять в себя запись. Поэтому к локальным переменным применимы только те функции, которые работают либо с позицией, либо с целым множеством, но не с содержимым записей.

9.3. Определение типа записей

Создание типа записей и его главного порядка

Для того чтобы создать в базе данных новый тип записи, нужно определить, какие поля должны быть в этом типе: и

названия, типы, способы упаковки. Кроме того, при создании типа записей требуется определить, каким образом должно быть упорядочено множество записей этого типа. Записи каждого типа могут быть упорядочены многими разными способами, но в момент создания важно задать один из этих порядков, который будет называться **главным порядком** типа.

Спецификация всех этих аспектов создаваемого типа записей дается в составной рамке, структура которой является прототипом рамок-записей данного типа. Внутри этой составной рамки должны быть созданы рамки, представляющие поля записи. Имена этих рамок будут служить именами полей, а типы значений полей и способы их упаковки задаются следующим образом.

Для простых полей — строковых, числовых, полей дат и времени — рамки-поля должны создаваться как простые рамки и в них должны быть введены строковые обозначения, задающие типы и способы упаковки. Текстовые и рисунковые поля должны быть представлены пустыми рамками, соответствующих типов — текстовыми и рисунковыми.

Спецификации простых полей задаются строками, в которых указывается, во-первых, должно ли поле участвовать в ключе упорядочения главного порядка, и, во-вторых, тип и способ упаковки значения поля.

Таблица 9.1. Обозначения типов полей

Обозначение	Тип значения поля и способ его упаковки
	Числовые поля
М	Малое число в диапазоне от -127 до +127 размером 1 байт
Ц	Число в диапазоне от -32767 до +32767 размером 2 байта
Б	Число с 9 значащими цифрами размером 4 байта
В	Вещественное число с 15 значащими цифрами размером 8 байт
Д	Число в формате даты размером 2 байта
Ч	Число в формате времени размером 2 байта
	Строковые поля
С	Строка нефиксированной длины
И20	Строка фиксированной длины размером 20 байт

Тип и способ упаковки поля задаются буквенными обозначениями, приведенными в табл. 9.1. Участие поля в ключе главного порядка указывается символом, ставящимся перед спецификацией типа поля. Эти символы перечислены в табл. 9.2.

Таблица 9.2. Символы спецификации ключевых полей

Обозначение	Участие поля в ключе упорядочения
-	Поле не является ключевым
<	Поле входит в ключ по слиянию и упорядочивается по возрастанию
>	Поле входит в ключ по слиянию и упорядочивается по убыванию
#	Поле является ключевым в упорядочении по смешению

Создав рамку, служащую прототипом рамки-записи, и введя в каждую из рамок-полей соответствующую спецификацию, нужно обратиться к функции **СоздатьМнож()**, имеющей следующий формат обращения:

СоздатьМнож(код области, рамка-прототип)

Первый аргумент этой функции задает ту информационную область, в которой должен быть размещен новый тип записей, а второй аргумент представляет рамку-прототип. После выполнения этой операции рамка-прототип превращается из составной рамки в рамку-запись, т.е. ее значением оказывается указатель записи. В начальный момент никакой записи этого типа еще не существует, указатель ссылается на пустую запись, поэтому все рамки- поля автоматически очищаются. В дальнейшем, когда будут созданы записи этого типа, рамки будут содержать значения полей текущей записи.

Примером спецификации полей может служить следующая программа на языке Мастер, которая автоматически расставляет спецификации полей перед обращением к функции **СоздатьМнож()**:

```
Врамке(сотрудник,
    Присв(отдел, "<Ц");
    Присв(фамилия, "<И20");
    Присв(тема, "-С");
    Присв(оклад, "-Б")
);
СоздатьМнож(1, сотрудник)
```

В этой программе создается новый тип записей с четырьмя полями и задается спецификация главного порядка данного типа записей. Этот главный порядок будет упорядочен по слиянию двух полей – отдел и фамилия, причем оба поля берутся по возрастанию.

Создание дополнительных порядков

То поле, по которому упорядочены записи, называется ключевым, потому что по значению этого поля, как по ключу, запись может быть найдена практически мгновенно! Как правило, требуется, чтобы в записях каждого типа было несколько ключевых полей, поскольку поиск может потребоваться по любому из них. Поэтому помимо главного порядка требуется создание в базе данных нескольких дополнительных порядков, делающих ключевыми различные поля.

Каждый дополнительный порядок, созданный для заданного типа записей, включает в себя все записи этого типа, но в каждом порядке эти записи перечисляются в разных последовательностях. Корректность порядков автоматически поддерживается системой при любых модификациях записей: и при их создании, и при изменении, и при уничтожении.

Дополнительные порядки создаются функцией **ВзятьПорядок()**. С ее помощью можно и создать новый порядок, и взять из базы какой-нибудь из существующих. Эта функция имеет следующий формат обращения:

```
ВзятьПорядок(
    код области,
    указатель множества,
    тип упорядочения,
    имя поля, возрастание/убывание,
    ...
) --> указатель множества
```

Первым аргументом задается код той области, в которой нужно создать новый порядок. Этот аргумент может быть равен -1, тогда обращение к функции должно не создавать новый порядок, а взять существующий. Второй аргумент задает тот тип записей, для которого должен быть создан или взят порядок. Аргумент **тип упорядочения** может быть равен 1 или 2, что соответствует соответственно упорядочение по слиянию или по смешению полей. Далее следует несколько пар аргументов, описывающих поля, входящие в ключ упорядочения данного порядка. В каждой паре сначала задается имя поля, затем способ упорядочения этого поля: +1 – по возрастанию, -1 – по убыванию.

Возвращаемым значением этой функции является указатель записи, идентифицирующий тот порядок, который был затребован функцией. (Напомним, что по указателю записи доступна не только текущая запись, но и один из порядков записей этого типа. Именно в этом качестве – как указатель порядка – используется указатель записи в данной функции.) Если требовалось найти существующий порядок (а не создавать его), но его не оказалось в базе, то возвращаемым значением является значение типа ПУСТО. Таким образом можно узнавать, имеется ли в базе упорядоченность записей по типу или иному полю.

Выборка порядков записей из базы

Две рассмотренные функции – СоздатьМнож() и ВзятьПорядок() – используются на стадии создания базы данных. Так, функция СоздатьМнож() создает новый тип записи и связывает указатель записи этого типа с рамкой-прототипом. После ее срабатывания эта рамка становится рамкой-записью. Однако рамка связана с указателем записи только в пределах одного сеанса. Когда рамковая структура будет сохранена в рамковом файле, все рамки-записи потеряют свои указательные значения и станут обычными составными. В очередном сеансе работы после считывания рамковой структуры из файла требуется вновь связать рамки-прототипы с указателями соответствующих записей. Для этого нужно выбрать из базы требуемые типы записей и присвоить соответствующие указатели записей в рамки-прототипы. Это вновь превратит их в рамки-записи и позволит через них работать с записями соответствующих типов. Для выборки из базы данных существующих в ней типов записей имеется следующая функция:

ВзятьМнож(код области, имя типа)
--> указатель множества

Она позволяет выбрать из заданной информационной области нужный тип записи и возвращает указатель записи этого типа. В качестве второго аргумента должно задаваться имя той рамки, которая служила рамкой-прототипом при создании этого типа записей. Операция открытия базы данных, в сущности, состоит как раз в присваивании указателей записей всем рамкам-записям.

С помощью этого обращения выбирается главный порядок данного типа записей. Если же потребуется любой из дополнительных порядков, то нужно воспользоваться функцией ВзятьПорядок(), описанной в предыдущем разделе. При обращении к ней с

тими же аргументами, что и при создании порядка, она не будет создавать второй такой порядок, а просто выберет ранее созданный и возвратит указатель записи, связанный с этим порядком. Для надежности рекомендуется в этом случае ставить в обращении к ВзятьПорядок() отрицательное число -1 в качестве кода области, что служит явным запретом создания порядка.

Типичным примером использования этих функций является следующая процедура открытия базы данных:

```
открытиеБазыДанных
"--- подготавка буфера и открытие области ----";
ОткрытьБазу(20);
Если(ОткрытьОбласть('заказ.тем')#0,
    Ошибка('Не могу открыть базу')
);
"--- выборка типа записей 'заказ' ----";
Присв(заказ, ВзятьМнож(0, 'заказ'))
"--- взятие нужного порядка ----";
Присв(заказ, ВзятьПорядок(-1, заказ, 1, 'поставщик', 1))
```

9.4. Операции над записями и полями

Форматный доступ к записям

Работа с содержимым записей через рамку-запись осуществляется набором следующих функций:

ВзятьЗапись(рамка-запись) --> 0/1
СоздатьЗапись(рамка-запись)
ИзменитьЗапись(рамка-запись)
УничтожитьЗапись(указатель записи)

Эти функции (кроме последней) осуществляют передачу информации между базой данных и рамками-записями. Так, функция ВзятьЗапись() выбирает значения полей из текущей записи и загружает ими рамки-поля. Функция СоздатьЗапись() создает новую запись данного типа и заполняет ее поля значениями рамок-полей. Функция ИзменитьЗапись() точно так же заполняет поля записи, но не новой, а существующей – той, которая является текущей. Функция УничтожитьЗапись() уничтожает текущую запись.

Функция ВзятьЗапись() обладает еще и возвращаемым значением 1 или 0 в зависимости от того, существует текущая запись или нет. Текущей записи может не существовать в любой из следующих ситуаций: множество записей не содержит ни одной записи, указатель записи сдвинут за последнюю запись

данного порядка, текущая запись вышла за ограничения, установленные для данного указателя функцией `ограничить()`. Это возвращаемое значение может использоваться, например, при последовательном переборе записей для проверки окончания цикла.

Табличный доступ к записям

Для табличного доступа к записям используются те же четыре функции, но с другим форматом обращения:

```
ВзятьЗапись(указатель записи, диапазон записей)
    --> число записей
СоздатьЗапись(указатель записи, диапазон записей)
    --> число записей
ИзменитьЗапись(указатель записи, диапазон записей)
    --> число записей
УничтожитьЗапись(указатель записи, диапазон записей)
    --> число записей
```

Первый аргумент в любом из обращений – это указатель записи. Он может быть задан как рамкой-записью, так и локальной переменной, поскольку рамки-поля здесь не потребуются. Вторым аргументом должен быть указатель диапазона электронной таблицы, задающий те строки таблицы, которые предназначены для выборки, создания, изменения или уничтожения записей. Двумя строками выше этого диапазона в таблице должны располагаться строка, содержащая имена полей. Ячейки этой строки определяют, какие поля должны располагаться в соответствующих столбцах. В одной из ячеек нужно разместить вместо имени символ "*". В столбец `диапазона записей` будут выбираться не значения какого-либо поля, а специальные системные значения – уникальные ключи записей, выбираемых в диапазон. Сами по себе эти ключи программисту бесполезны, они требуются модифицирующим операциям – `ИзменитьЗапись()`, `СоздатьЗапись()`, `УничтожитьЗапись()`. В таблице эти уникальные ключи выглядят в виде знака вопроса. Для удобства пользователя можно разместить их в самом левом столбце, скрыв вертикальной разграфкой и заблокировать доступ в этот столбец соответствующей установкой боковика таблицы. Стока имен располагается двумя строками выше диапазона записей, а непосредственно над ним для того, чтобы между именами полей и их значениями можно было провести разделяющую линию разграфки.

Возвращаемым значением функции `ВзятьЗапись()` в этом варианте обращения является количество взятых записей (кото-

всегда равно или меньше числа строк, указанных в `диапазоне записей`). Второй особенностью этого варианта обращения является то, что указатель текущей записи не остается на месте, а перемещается вперед на то количество записей, которое было взято в таблицу. Таким образом, последовательными обращениями в этой функции можно выбирать последовательные порции записей.

Для работы остальных функций существенно наличие столбца, помеченного звездочкой. Вместо имени поля в этом столбце хранятся уникальные ключи записей, взятых в таблицу. Так, функция `УничтожитьЗапись()` уничтожает те записи, которые соответствуют строкам, содержащим уникальные ключи, а вновь вставленные строки не участвуют в операции. То же самое делает и функция `ИзменитьЗапись()`. Функция `СоздатьЗапись()`, наоборот, создает только записи, соответствующие вставленным строкам таблицы, в которых нет уникального ключа.

Перенос информации из записи одного типа в другой

Две из рассмотренных функций имеют и еще один вариант обращения:

```
ВзятьЗапись(указатель записи, рамка-запись)
СоздатьЗапись(указатель записи, рамка-запись)
```

В информационном обмене здесь участвуют текущая запись, заданная первым аргументом, и рамка-запись, заданная вторым аргументом. Существенно то, что рамка-запись может содержать указатель записи типа, отличного от типа записей первого аргумента. Выборка информации производится по совпадению имен полей этих двух типов, участвующих в операции. Тем самым происходит передача информации из записей одного типа в другой. Того же эффекта можно было бы добиться и обычными присваиваниями, но это потребовало бы и более сложной программы, и было бы более медленным при исполнении.

Рассмотрим пример использования функции `ВзятьЗапись()`, который представляет способ преобразования записей одного типа в записи другого, близкого по структуре, но отличающегося наличием или отсутствием нескольких полей. Потребность в таком преобразовании возникает, например, при добавлении (или уничтожении) полей в существующий тип записей. В этом примере используются две рамки-записи: `запись1` и `запись2`. Перенос информации производится из записей типа `запись1` в запись типа `запись2`.

```

переносЗаписей—
НайтиНачало(запись1);
Повтор(ВзятьЗапись(запись1, запись2),
СоздатьЗапись(запись2);
НайтиСлед(запись1)
)

```

Взятие значений отдельных полей

Иногда при программировании для доступа к записям проще пользоваться не рамкой-записью, а локальной переменной присвоенным в нее значением типа указателя записи. Однако при отсутствии рамки-записи становятся недоступными все рассмотренные операции, связанные с выборкой или изменением значений полей записей. Это частично компенсируется наличием функции Поле(), с помощью которой можно выбирать значения полей, не пользуясь никакими рамками. Обращение к этой функции имеет следующий формат:

```

Поле(указатель множества, имя поля)
--> значение поля

```

Еще одна проблема, связанная с полями, возникает при использовании порядков, организованных по смещению полей. Взять из некоторой позиции такого множества текущую запись, вы не можете по значениям полей определить, какому из полей соответствует данное вхождение записи во множество – этим полем может оказаться любое из смешиемых полей. Для определения текущего значения ключа в упорядочении по смещению полей служит функция Ключ(). Ее аргументом является указатель множества, упорядоченного по смещению полей, а возвращаемым значением является строка или число, задающее текущее значение упорядочивающего поля. Обращение к функции

```

Ключ(указатель множества) --> значение поля

```

возвращает значение того из смешиемых полей, по значению которого запись зарегистрирована в текущей позиции множества.

9.5. Поиск и последовательный перебор записей

Последовательный перебор записей

Последовательный перебор записей основывается на том, что доступ к ним производится только через те или иные порядко-

Для организации последовательного перебора требуются следующие четыре операции:

```

НайтиНачало(указатель записи)
НайтиКонец(указатель записи)
НайтиСлед(указатель записи) --> 0/1
НайтиПред(указатель записи) --> 0/1

```

Аргументами этих функций могут быть как рамки-записи, содержащие в качестве значений указатели записей, так и локальные переменные с такими значениями. Любая из этих функций работает только с указателем, даже если аргументом является рамка-запись. Так, например, после срабатывания функции НайтиСлед() указатель записи перемещается к следующей записи, но рамка-запись продолжает содержать в рамках-полях те же значения, что были в них перед обращением. В этот момент содержимое рамок-полей не соответствует значениям полей текущей записи. Для восстановления этого соответствия требуется явное обращение к функции ВзятьЗапись().

Возвращаемым значением функций НайтиСлед() и НайтиПред() является число 1 или 0, означающее, произошел ли переход к следующей или предыдущей записи. Тем самым можно использовать это возвращаемое значение в качестве условия продолжения цикла при последовательном просмотре записей, хотя более удобно для этой же цели использовать возвращаемое значение функции ВзятьЗапись().

Типичным примером, использующим эти функции для выполнения последовательного просмотра записей, является следующая программа, суммирующая значения поля стоимость во всех записях множества заказ.

```

подсчетСуммарнойСтоимости—
Блок(1,
Присв(1,0); "Пер(1) – накопитель для стоимости";
НайтиНачало(заказ);
Повтор(ВзятьЗапись(заказ),
Присв(1,Пер(1)+заказ.стоимость);
НайтиСлед(заказ)
);
Сообщить('Суммарная стоимость = '&&Пер(1)&&'руб.')
)

```

Иногда возникает необходимость выполнить переход вперед или назад по последовательности записей сразу на несколько позиций. В этом случае в обращение к функциям НайтиСлед() и НайтиПред() нужно добавлять второй аргумент – число, задающее количество позиций, на которое нужно сдвинуть указатель.

Форма обращения к этим функциям будет в этом случае следующей:

```
НайтиСлед(указательЗаписи, числоПозиций)
--> числоПозиций
НайтиПред(указательЗаписи, числоПозиций)
--> числоПозиций
```

Значение второго аргумента в обоих случаях должно задаваться положительным числом.

Примером использования функции НайтиПред() с такой формой обращения служит реализация действия клавиши {ПРЕД} при работе с табличным доступом к базе данных. Эта клавиша должна взять предыдущую порцию записей. Поскольку после взятия текущей порции с помощью обращения к функции ВзятьЗапись() (в табличном варианте обращения) указатель текущей записи переместился за последнюю взятую в таблицу запись, необходимо сначала переместить указатель назад на число взятых ранее записей и еще на размер выбираемой порции. После этого нужно обратиться к функции ВзятьЗапись(). Предположим, что результат обращения к ВзятьЗапись(), т.е. число взятых записей, мы сохраним в ячейке А2 той же таблицы. Тогда программа, реализующая действие клавиши {ПРЕД}, будет следующей:

```
{ПРЕД}—
Врамке(таблица,
    НайтиПред(запись, !А!2+20);
    Присв(!А!2, ВзятьЗапись(запись, !А!3: !Д!22))
)
```

Поиск записей последовательным перебором

Перейдем теперь к рассмотрению функциональных возможностей, обеспечивающих поиск записей во множествах по значениям их полей.

Наиболее общим способом поиска является способ, основанный на последовательном просмотре записей, при котором на каждой записи проверяется некоторое логическое условие. Примером такой программы является следующая:

```
НайтиНачало(заказ);
Повтор(ВзятьЗапись(заказ) &
        заказ.заказчик#заказ.поставщик,
    НайтиСлед(заказ)
)
```

На реальных объемах, однако, этот способ практически неприемлем из-за того, что время выполнения такого поиска пропорционально общему количеству записей и может затянуться до нескольких десятков минут.

Следующим шагом повышения эффективности поиска является отказ от интерпретации поискового цикла на уровне языка Мастер и перевод этого поиска внутрь встроенной функции НайтиСлед() или НайтиПред(). Здесь поиск реализован на машинном уровне более эффективно, хотя и остается по-прежнему последовательным полным перебором, не задействующим никаких специальных структур типа индексов. Эта возможность обеспечивается теми же функциями путем добавления в обращение к ним дополнительных аргументов. Каждое из дополнительных поисковых условий представляется тройкой аргументов, первый из которых задает имя одного из полей, а два других – граничные значения для этого поля. Функция отыскивает такую запись, в которой все указанные в обращении поля обладают значениями, укладывающимися в заданные ограничения (включительно). Форматы обращения к функциям в этом варианте следующие:

```
НайтиСлед(
    указатель записи,
    имя поля1, меньшее значение1, большее значение1,
    имя поля2, меньшее значение2, большее значение2,
    ...
)
```

и

```
НайтиПред(
    указатель записи,
    имя поля1, меньшее значение1, большее значение1,
    имя поля2, меньшее значение2, большее значение2,
    ...
)
```

У этих обращений, как и у простейшего обращения к функциям НайтиСлед() и НайтиПред(), может быть использован дополнительный аргумент (в самом конце списка аргументов), задающий количество записей, на которые нужно сдвинуться. При этом считаются только подходящие записи. Особое значение имеет случай, когда задано количество записей, равное нулю. Это означает, что проверку пригодности нужно выполнять с текущей записью, не делая сдвига. Первая же подходящая запись должна быть сделана текущей. Этот частный случай необходим при организации циклических поисков для нахождения самой первой записи. Например, если мы хотим вычислить суммарную стоимость заказов, но не по всем записям, а только по тем,

которые относятся к определенному заказчику, то рассмотренный пример видоизменится следующим образом:

```
Блок(1,
    Присв(1,0); "Пер(1) - накопитель для стоимости";
    НайтиНачало(заказ);
    НайтиСлед(заказ, 'заказчик', 'кооператив СПРАВКА', 0);
    Повтор(ВзятьЗапись(заказ),
        Присв(1,Пер(1)+заказ.стоимость);
        НайтиСлед(заказ, 'заказчик', 'кооператив СПРАВКА'
    );
    Сообщить('Суммарная стоимость = '&&Пер(1)&&' руб.'
)
```

Первое из двух обращений к `НайтиСлед()` представляет как раз упомянутый частный случай – в конце списка аргументов стоит число 0, определяющее, на сколько специфицированных записей нужно сдвинуться. Это обращение находит первую запись требуемого типа и работает правильно, в частности и в том случае, когда подходящая запись стоит в самом начале множества.

Ключевой поиск записей

Рассмотренная форма обращения обладает большей эффективностью за счет ограничения гибкости запроса (здесь уже невозможны произвольные предикаты над полями записей), но по-прежнему не обеспечивает максимальной эффективности, поскольку ее реализация основана на последовательном переборе записей.

Основное средство для выполнения быстрых поисковых операций обеспечивается другой функцией – `Найти()`, работа которой основывается на использовании упорядоченности множеств. С помощью этой функции можно искать записи по значениям лишь тех полей, которые входят в ключи упорядочения тех или иных множеств.

Наиболее проста форма использования функции `Найти()` для поиска по значению первичного поля ключа (поля, имеющего ранг 1 в порядке по слиянию или по первому из полей, взятых по смешению). Обращение к ней в этой форме имеет следующий вид:

```
Найти(
    указатель множества,
    имя поля,
    значение,
    локальность поиска
) --> 0/1
```

Действие функции заключается в следующем. Во множестве, задаваемом аргументом `указатель множества`, производится поиск записи, в которой поле `имя поля` имеет заданное `значение`. Аргумент `локальность поиска` управляет выбором того порядка, в котором следует делать поиск. Возвращаемым значением функции является число 1 или 0, означающее успех или неуспех поиска.

Для поиска необходимо полное множество, упорядоченное по заданному полю, и это множество автоматически выбирается из базы и подставляется в первый аргумент взамен того порядка, который в нем был. Таким образом, помимо возвращаемого значения, функция обладает важным побочным действием – переходом во множество с одного порядка на другой.

Важным свойством функции является то, как она работает при отсутствии записи с указанным значением поля. Если записи с таким значением поля не существует, то указатель будет остановлен на записи с ближайшим большим значением. При этом функция возвратит значение 0. Это позволяет искать названия и фамилии, не вводя их целиком. Достаточно указать несколько первых букв, однозначно определяющих фамилию, чтобы функция `Найти()` установила указатель на записи с требуемой фамилией.

Второе важное свойство этой функции проявляется тогда, когда порядка, требуемого для осуществления поиска, в базе не существует, т.е. когда искомое поле ни в одном порядке не является ключевым. В этом случае будет взят главный порядок данного типа и поиск сделан по нему полным перебором.

Четвертый аргумент функции – `локальность поиска` – может заблокировать выбор подходящего порядка и побочное изменение первого аргумента. Если этот признак равен 1, то поиск записи ведется именно в том множестве или подмножестве, которое представлено первым аргументом, даже если это множество и не упорядочено по искомому полю (в последнем случае поиск будет вестись полным перебором). Если же четвертый аргумент равен 0, то выбор подходящего множества не блокируется и функция работает, как это описано выше. Обычно функция используется с четвертым аргументом, равным 0.

Описанный вариант поиска позволяет использовать поля только первого ранга в составных ключах упорядочения. Можно задействовать и более высокие ранги. Для этого существует второй вариант обращения к той же функции:

```
Найти(указатель множества, ранг поля, значение)
--> 0/1
```

В этом варианте обращения подразумевается, что указанное первым аргументом множество упорядочено по слиянию нескольки-

ких полей. Второй аргумент *ранг поля* задает номер того поля, по которому ведется поиск, причем поле задается не именем, а своим рангом в ключе упорядочения. Третий аргумент задает искомое значение этого поля.

Поиск производится именно в том множестве, которое предъявлено первым аргументом, без предварительного выбора подходящего порядка – об этом выборе должен позаботиться сам программист (обратившись, например, к функции *ВзятьПорядок()* или выполнив поиск по первичному ключу первой формой обращения к функции *Найти()*). При обращении с полем ранга 1 поиск ведется, как обычно – находится самая первая запись с указанным значением поля. При обращении с полем ранга 2 поиск ведется уже не с самого начала множества, а с его текущей позиции, и притом только среди записей, имеющих в первичном поле фиксированное значение – текущее значение первичного поля. При поиске по полю третьего ранга фиксированными являются значения первичного и вторичного полей и т.д.

Этот вариант обращения используется обычно последовательным образом для выполнения сложного поискового запроса. Сначала выполняется поиск по полю первого ранга, затем из текущей позиции по полю второго ранга и т.д. Например, если множество "сотрудник" упорядочено по слиянию трех полей – номер отдела (первичное поле), номер лаборатории (вторичное), фамилия (третичное), то для поиска тов. Иванова, работающего в первой лаборатории седьмого отдела, можно использовать следующую формулу:

```
Присв(сотрудник,
    ВзятьПорядок(-1, сотрудник, 1,
        "отдел", 1,
        "лаборатория", 1,
        "фамилия", 1
    )
);
Найти (сотрудник, 1, 7);
Найти (сотрудник, 2, 1);
Найти (сотрудник, 3, "Иванов")
```

Следует отметить, что работа функции *Найти()* вследствие использования специальных поисковых структур, хранящихся в базе данных, выполняется во много раз быстрее, чем любой вариант поиска, основанный на последовательном переборе записей. Поэтому при проектировании структуры вашей базы данных вы должны тщательно продумывать, какие порядки включать в базу для того, чтобы были возможны нужные вам поисковые опе-

рации.

103

9.6. Операции над подмножествами

Ограничение множеств

При работе с записями требуется включать в рассмотрение, как правило, не все записи данного типа, а лишь какое-то их подмножество, удовлетворяющее тому или иному критерию. Подмножества можно формировать несколькими разными способами, например явным перечислением некоторых записей и включением их в списковое подмножество. Однако наиболее эффективным способом является формирование подмножества путем ограничения существующего множества или подмножества по значению какого-то поля.

Ограничение множества возможно только по тому полю, по которому это множество упорядочено. Каждый указатель множества в рамке-записи или в локальной переменной Мастер-программы может быть снабжен двумя ограничительными значениями, влияющими на работу функций последовательного перебора – *НайтиНачало()*, *НайтиКонец()*, *НайтиСлед()*, *НайтиПред()*. Все эти функции при наличии ограничителей начинают работать только в пределах, задаваемых этими ограничителями.

Первоначально всякий указатель множества является неограниченным, т.е. ограничители на нем отсутствуют. Они устанавливаются на указатель множества с помощью функции *Ограничить()*, которая по своему интерфейсу весьма сходна с функцией *Найти()*:

```
Ограничить(
    указатель множества,
    имя поля, меньшее значение, большее значение,
    локальность ограничения
) --> 0/1
```

Первый аргумент *указательМножества* является входной и выходной переменной. Он задает тот указатель, который должен быть в результате работы функции снабжен ограничителями. Аргумент *имя поля* задает то поле, по которому должно быть наложено ограничение. Аргументы *меньшее значение* и *большее значение* задают два граничных значения для этого поля, которые и будут служить ограничителями. Последний аргумент *локальность ограничения* полностью аналогичен аргументу *локальность поиска* в функции *Найти()* – он определяет (если равен 1), можно ли для установки ограничения переключиться на другой порядок, или (если равен 0) нельзя этого делать. Переключение потребуется в том случае, если предъявленное множество упорядочено не по указанному полю.

После срабатывания функции **Ограничить()** для некоторого указателя множества функция **НайтиНачало()**, примененная к этому указателю, будет устанавливать текущую позицию не на самую первую запись множества, а на первую запись, удовлетворяющую ограничению. Точно так же функция **НайтиКонец()** будет устанавливать текущую позицию не за самый конец множества, а за последнюю запись, удовлетворяющую ограничению. Пользуясь этим, можно ограничить массовую обработку записей нужным вам подмножеством.

Так, например, для того чтобы подсчитать суммарную стоимость всех заказов, относящихся к одному заказчику, можно вместо рассмотренного цикла использовать следующий:

```
Блок(1,
    Присв(1,0);
    Ограничить(заказ,
        'заказчик',
        'кооператив СПРАВКА',
        'кооператив СПРАВКА',
        0
    );
    Повтор(ВзятьЗапись(заказ),
        Присв(1,Пер(1)+заказ.стоимость);
        НайтиСлед(заказ)
    );
    Сообщить('Суммарная стоимость = '&&Пер(1)&&' руб.')
)
```

Обратите внимание на то, что перед началом цикла здесь не делалось обращения к функции **НайтиНачало()**. Это неслучайно, поскольку функция **Ограничить()** после выполнения ограничения сама устанавливает текущую позицию на начало ограниченного множества. Приведенная программа является типичной в отношении обработки множеств записей в базе данных МАСТЕРа. С помощью такой программной конструкции могут быть запрограммированы всевозможные информационные структуры – сети деревья, наборы и т.п.

В рассмотренном варианте обращения к функции **Ограничить()** использовалось поле, имеющее первый ранг в ключе упорядочения. Подобно тому как в функции **Найти()** можно производить поиск также и по вторичным полям, здесь можно устанавливать ограничения по вторичным полям. В обращении к функции **Ограничить()** для этого случая вместо имени поля должен задаваться его ранг в ключе упорядочения, а последний **локальностьОграничения** должен отсутствовать (при этом предполагается, что множество упорядочено по соответствующему слиянию полей):

Ограничить(
 множество,
 ранг поля,
 меньшее значение,
 большее значение
)

В данном случае ограничивающее поле задается не именем, а своим рангом в ключе упорядочения, причем предполагается, что предъявленное множество упорядочено по соответствующему слиянию полей, о чем нужно позаботиться своевременным переключением на нужный порядок.

Ограничение по вторичному (третичному и т.д.) полю имеет несколько иной смысл, чем по первичному. Если при выделении от X1 до X2 по первичному полю отбираются все записи с такими значениями, то выделение от Y1 до Y2 по вторичному полю при текущем первичном поле, равном X, означает выбор таких записей, у которых первичное поле равно X, а вторичное поле заключено в отрезке от Y1 до Y2. Выделение по третичному полю от Z1 до Z2 означает выделение таких записей, у которых первичное поле равно текущему значению X, вторичное поле равно текущему значению Y, а третичное поле заключено в отрезок от Z1 до Z2. То же самое повторяется для ключевых полей любого ранга.

После того как ограничение, установленное на указатель, становится ненужным, вы можете снять с указателя ограничители. Для этого существует функция:

СнятьОграничение(указатель множества)

Кроме того, имеется возможность запросить, в каком состоянии находится в данный момент указатель множества – с ограничителями или без них. Для этого существует предикат:

Ограничено(указатель множества) --> 0/1

Множественные операции над записями

Описанные в предыдущем разделе операции перебора и поиска позволяют реализовывать произвольные поисковые запросы в базе данных МАСТЕРа. При этом вся идеология организации запросов ориентирована на единичные записи. Во многих случаях бывает более наглядно и более эффективно основывать реализацию запросов на понятиях множеств как целых. Так, например, если требуется реализовать запрос, имеющий структуру: "выбрать

все записи, удовлетворяющие условию А и условию Б одновременно", где каждое из условий А и Б достаточно сложно, то естественно выполнить его в три этапа: найти множество всех записей, удовлетворяющих условию А, затем найти другое множество записей, удовлетворяющих условию Б, и после этого сделать пересечение этих двух множеств, т.е. выделить лишь те записи, которые входят и в первое, и во второе множества.

Для выполнения такого рода действий требуются структуры, называемые **списковыми подмножествами**, т.е. подмножествами, образованными не простым ограничением по значению поля, а произвольной логической выборкой записей. Для работы с списковыми подмножествами существует набор специальных функций: ПустоМнож(), ВключитьЗапись(), ИсключитьЗапись(), ВзятьПорядок(), ПрисоединитьПересечение(), УничтожитьМнож(), ОчиститьОбласть(), которые позволяют создавать списковые множества, изменять произвольным образом их состав, уничтожать такие множества.

С помощью функции ПустоМнож() можно создавать новые списковые подмножества, которые в начальный момент являются пустыми и подлежат заполнению в дальнейшем с помощью остальных функций. Создаваемое множество должно быть отнесено к тому или иному типу записей и упорядочено – все это указывается аргументами в обращении к функции ПустоМнож().

Структура аргументов функции ПустоМнож() в принципе повторяет структуру аргументов функции ВзятьПорядок(). Они точно так же задают, во-первых, тип записей, над которым создается подмножество (какой совокупности записей это подмножество принадлежит), и, во-вторых, поля, участвующие в упорядочении, и способ этого упорядочения.

```
ПустоМнож(  
    код области,  
    указатель множества,  
    вид упорядочения,  
    имя поля1, упорядочение поля1,  
    имя поля2, упорядочение поля2,  
    . . .  
) --> указатель пустого подмножества
```

Результатом обращения к этой функции является создание требуемого пустого множества в указанной области, указатель на которое оказывается возвращаемым значением функции.

Следующие две функции служат для изменения элементного состава подмножеств путем включения и исключения отдельных записей в подмножества. Эти функции имеют следующий формат обращения:

```
ВключитьЗапись(  
    указатель подмножества,  
    указатель записи  
)
```

```
ИсключитьЗапись(  
    указатель подмножества  
)
```

Функция ВключитьЗапись() включает в подмножество, заданное указателем подмножества, текущую запись, заданную вторым аргументом указатель записи. Функция ИсключитьЗапись() исключает текущую запись из подмножества. Сама запись не уничтожается, она просто перестает принадлежать данному списковому подмножеству, но продолжает присутствовать во всех полных множествах данного типа, и, быть может, в каких-то других списковых подмножествах.

Списковое подмножество можно образовать и более простым способом – за одно обращение к функции ВзятьПорядок(), если записи, которые должны принадлежать этому подмножеству, могут быть выделены с помощью ограничения по некоторому ключевому полю. Функция ВзятьПорядок() – это та самая функция, которая создает или выбирает из базы полные упорядоченные множества, позволяет создавать и подмножества. Это происходит в тех случаях, когда аргументом, задающим множество в этой функции, оказывается ограниченное множество (снабженное ограничителями функцией Ограничить()). Наличие ограничителей означает, что требуется создать списковое подмножество и включить в него все записи, удовлетворяющие ограничению.

Наконец, последний способ формирования списковых подмножеств основан на теоретико-множественных операциях объединения и пересечения множеств. В Мастере обе эти операции могут выполняться с помощью универсальной функции ПрисоединитьПересечение(). Обращение к ней выглядит следующим образом:

```
ПрисоединитьПересечение(  
    указатель подмножества,  
    подмножество1, подмножество2, ...  
)
```

Эта функция добавляет в подмножество, заданное указателем подмножества, те записи, которые входят в каждое из подмножеств подмножество1, подмножество2, и т.д. В этой функции сочетаются сразу две главные теоретико-множественные операции – объединение и пересечение: она выполняет пересечение подмножеств подмножество1, подмножество2, и т.д. и затем

объединяет получившееся подмножество с подмножеством, заданным первым аргументом. Приведем важные частные случаи функции **ПрисоединитьПересечение()**.

а) Подмножество, заданное **указателем подмножества**, – пустое. Результат функции – пересечение множеств **подмножество1**, **подмножество2** и т.д.

б) У функции только два аргумента – **указатель подмножества** и **подмножество1**. Результат функции – объединение этих множеств.

в) Общий случай обращения реализует один терм дизъюнктивной формы логического высказывания.

Рассмотрим для примера следующую задачу. Пусть требуется создать список записей, включающий лишь те заказы, которые выполняются для заказчика "кооператив СПРАВКА" и имеют стоимость больше 1000 руб. каждый. Можно создать сначала списковое подмножество, включающее только заказы для "кооператива СПРАВКА", а затем исключить из него все заказы с маленьким значением стоимости:

```
Ограничить(заказ,'заказчик',
    'кооператив СПРАВКА','кооператив СПРАВКА',0
);
Присв(заказ,ВзятьПорядок(0,заказ,1,'количество',1));
Повтор(ВзятьЗапись(заказ),
    Если(заказ.стоимость < 1000,
        ИсключитьЗапись(заказ),
        НайтиСлед(заказ)
    )
)
```

Здесь использовано свойство функции **ВзятьПорядок()**, состоящее в том, что при предъявлении ограниченного множества она создает новое списковое подмножество и включает в него все записи, удовлетворяющие ограничению. Обратите внимание также на организацию цикла: внутри его тела происходит либо исключение записи (и текущей записью оказывается следующая), либо просто переход к следующей записи.

СТРУКТУРНЫЕ МОДЕЛИ ИНФОРМАЦИИ В БАЗЕ ДАННЫХ

Рассмотренные функции языка Мастер представляют собой основные элементы системы управления базами данных в МАСТЕРе. Комбинируя эти элементы, можно строить самые разнообразные структуры информации в базах данных. В этой главе мы рассмотрим некоторые наиболее типичные структуры, встречающиеся практически во всех базах данных. Каждая типовая структура представляется определенной процедурной схемой, требующей применительно к конкретной информации наполнения конкретными обрабатывающими действиями.

10.1. Организация прикладных систем в МАСТЕРЕ

Всякая база данных – это не просто информация на диске, это еще и определенная интерактивная среда, обеспечивающая интерфейс с этой информацией, т.е. доступ к записям, исполнение поисковых запросов, генерацию отчетов, вычислительную обработку записей. Реализация такого интерфейса в МАСТЕРЕ требует построения системы рамок, включающей рамки-записи, рамки-функции, вспомогательные рамки, требующиеся для визуализации данных или представления программных структур. Вся эта система рамок может быть устроена, разумеется, самым произвольным способом, однако, как и во всяком программировании, желательно придерживаться той или иной дисциплины, определенной технологий. Здесь мы рассмотрим одну из возможных технологий такого рода, которая уже используется в стандартной диалоговой оболочке МАСТЕРа.

Каждая интерактивная система представляется рамкой, как правило, составной, внутри которой сформирована интерактивная среда, обеспечивающая работу в системе. Эту рамку будем называть рабочей рамкой системы. Для функционирования этой среды могут требоваться какие-то наборы процедур, которые бывает удобно оформлять в виде отдельных рамок (например, из-за того, что эти же процедуры используются в других системах). Следует позаботиться о том, чтобы процедурные пакеты загружались в память автоматически, как только в них возникает потребность. Это будет происходить, если операцию загрузки связать в качестве фильтра с операцией Считать() в рабочей рамке системы, причем удобнее всего, чтобы пакет загружался не на рабочее поле МАСТЕРа, где он мешал бы пользователю в его работе, а в невидимую рамку Биб библиотеки диалоговой оболочки. Такое действие запрограммировано в процедуре, написанной на Мастере и расположенной в библиотеке Биб. Обращение к ней имеет формат:

ЗагрузитьПакет(имя пакета)

Напомним, что для связывания формулы с функцией в определенной рамке необходимо, установив курсор на эту рамку, выполнить команду меню:

{Ф10} Рамка Действие Функция Считывание

В таком разделении системы на рабочую рамку и процедурный пакет, а также в таком способе автоматической загрузки пакета при считывании рабочей рамки состоит *первый элемент* описанной организации прикладных систем в МАСТЕРЕ.

Второй элемент состоит в том, что действия клавиш в рабочей рамке программируются обращениями к процедурам загруженных пакетов в виде:

имяПакета.имяПроцедуры(аргументы)

В этом обращении имя вызываемой процедуры является не простым идентификатором, а составным, явно задающим тот пакет, в котором она определена. Такие обращения можно встраивать в рабочую рамку либо как локальные, либо как глобальные программируемые действия клавиш.

Процедуры, требующиеся для обслуживания рабочей рамки, можно располагать либо в самой этой рабочей рамке, либо в отдельной рамке-пакете, загружаемой описанным способом. Вынесение процедур в отдельную рамку-пакет может быть полезно не только в тех случаях, когда они имеют общий характер

могут использоваться в рабочих рамках других систем. Это желательно и тогда, когда вы предполагаете, что состояние рабочей рамки придется записывать на диск, сохраняя в ней навработанную информацию или состояние решаемой задачи. Если же вся информация хранится только в базе данных и рабочая рамка не будет многократно записываться на диск, то процедуры проще всего сосредоточить внутри нее.

Третий элемент организации рабочей рамки определяет, каким образом начинается работа с системой. Начало работы требует либо открытия базы, либо выполнения каких-то настроек, и бывает запрограммировано в некоторой инициализирующей процедуре. Вызов такой инициализирующей процедуры удобно связывать в качестве фильтра функции Войти() для рабочей рамки.

Именно на этих трех перечисленных элементах строится рабочая рамка базы данных при определении схемы базы через стандартное меню МАСТЕРа. Фильтром считывания для каждой такой рабочей рамки является формула

ЗагрузитьПакет("БибБдраб")

которая загружает пакет общих процедур, выполняющих различные операции над записями базы данных. Фильтром функции Считать() для рабочей рамки базы данных является обращение к функции

открытиеБазы()

В отличие от функции ЗагрузитьПакет(), являющейся универсальной и располагающейся в библиотеке диалоговой оболочки, эта функция является уникальной для каждой базы данных, поскольку должна открыть те области, которые имеются именно в конкретной базе. Эта функция располагается внутри рабочей рамки базы.

Разобраться в устройстве рабочей рамки базы рекомендуется тому программисту, который собирается либо развивать возможности, созданные в этой рамке автоматически, либо строить рабочие рамки независимо, не пользуясь стандартным диалоговым интерфейсом.

10.2. Организация рабочей рамки базы данных

Описанная дисциплина относится к организации произвольных интерактивных систем в МАСТЕРЕ, а не только к базам данных. Специфика баз данных добавляет собственные элементы к этой дисциплине. Из возможных связанных с базами данных вопросов в систематическом решении нуждаются прежде всего вопросы инициализации и открытия баз данных.

Процедура инициализации базы данных

Каждая база данных состоит из нескольких информационных областей на диске, внутри которых находятся записи и порядки. Если вы строили базу данных через стандартный интерфейс МАСТЕРа, то все эти информационные области и пустые упорядоченные множества записей в них были созданы автоматически во время определения схемы базы.

База данных любого прикладного характера создается, как правило, в нескольких экземплярах, например в разных организациях, для разных пользователей. Невозможно было бы потребовать повторения самой процедуры создания схемы базы только ради того, чтобы создать очередной пустой экземпляр базы. Этого, конечно же, можно легко избежать и притом несколькими различными способами. Так, например, можно после создания схемы базы данных (а значит, и самой базы) сразу же сохранить ее пустые информационные области на диске. Тогда для создания нового экземпляра базы будет достаточно переписать в новое место эти файлы.

Другое решение проблемы состоит в том, чтобы разместить непосредственно внутри рабочей рамки базы процедуру, выполняющую полную инициализацию базы. Эта процедура может заодно служить исчерпывающим формальным описанием схемы базы, независимым от диалоговой оболочки. **Процедура инициализации** базы данных должна создать на диске все ее информационные области и внутри областей создать все требуемые типы записей. Типичный вид этой процедуры может быть следующим:

```
созданиеБазы
"-- подготовка буфера базы данных";
Если(БазаОткрыта(), ЗакрытьБазу());
ОткрытьБазу(20);

"-- создание информационных областей";
СоздатьОбласть(0, "DATA0.МЕМ");
СоздатьОбласть(1, "DATA1.МЕМ");
СоздатьОбласть(2, "DATA2.МЕМ");
"-- создание типов записей и дополнительных порядков";
Врамке(запись1,
    Присв(поле1, "<С");
    Присв(поле2, "-Б");
    Присв(поле3, "-И25")
);
СоздатьМнож(0, запись1);
ВзятьПорядок(0, запись1, 1, "поле2", 1);
...
"-- закрытие созданной базы данных, сброс буфера";
ЗакрытьБазу()
```

Процедура открытия базы данных

Если информационная система организована, как описано в 10.1, то работа с ней должна начинаться с того, чтобы загрузить в МАСТЕР рабочую рамку и войти в эту рамку. При считывании рабочей рамки автоматически загружаются те процедурные пакеты, которые в ней используются. При входе в рабочую рамку сработает фильтр функции Войти(), который вызовет процедуру открытия базы данных. Вообще говоря, процедура открытия может выполнять самые разнообразные подготовительные действия, но основным ее назначением является создание буфера, открытие информационных областей и связывание всех рамок-записей с соответствующими типами записей путем присваивания им указателей множеств. Типичная процедура открытия базы данных имеет следующий вид:

```
открытиеБазы
"-- подготовка буфера базы данных";
Если(БазаОткрыта(), ЗакрытьБазу());
ОткрытьБазу(20);

"-- открытие информационных областей";
Если(ОткрытьОбласть("DATA0.МЕМ")#0,
    Ошибка("Нет области DATA0.МЕМ")
);
Если(ОткрытьОбласть("DATA1.МЕМ")#1,
    Ошибка("Нет области DATA1.МЕМ")
);
Если(ОткрытьОбласть("DATA2.МЕМ")#2,
    Ошибка("Нет области DATA2.МЕМ")
);

"-- связывание рамок-записей с указателями множеств";
Присв(запись1, ВзятьМнож(0, "запись1"));
...
...
```

10.3. Форматно-табличные рамки

Доступ к записям может осуществляться либо в форматном, либо в табличном режимах. Для форматного доступа требуется рамка-запись, т.е. составная рамка, компоненты которой — рамки- поля — представляют отдельные поля записи. Для табличного доступа требуется рамка с электронной таблицей. Желательно создать такой интерфейс, при котором легко можно переключиться с одного режима на другой без явного перехода из рамки-записи в табличную и обратно. В стандартной диалоговой

оболочке МАСТЕРа рамки-записи строятся таким образом, чтобы совместить возможности обоих режимов. Для этого они помимо рамок-полей включают в себя и табличные рамки, а интерфейсные процедуры работают так, чтобы поддерживать метафору то форматного, то табличного доступов. Такую организацию рамки-записи тоже можно считать одним из элементов дисциплины организации базы данных, а сами рамки-записи, объединяющие два режима доступа, будем называть **таблично-форматными** рамками.

Рассматриваемые в данном разделе рамковые структуры и обслуживающие их процедуры представляют часть реализации того самого диалогового интерфейса, который имеется в стандартной диалоговой оболочке МАСТЕРа и который описан в гл. 6.

Структура форматного режима доступа

Перед созданием рамок-полей в рамке-записи должна быть создана текстовая рамка, служащая фоном форматки. Эта фоновая рамка должна быть без окаймления и по размерам занимать все поле зрения рамки-записи. Текст, содержащийся в ней, может содержать произвольные надписи и линии псевдографики, придающие ему вид какого-то бланка или учетной формы, принятой при обработке информации в данной проблемной области. Операции над записями, применяемые к рамке-записи, такие, как **СоздатьМнож()** или **ВзятьЗапись()**, будут игнорировать фоновую рамку, поскольку она не является полем записи. Рамки-поля создаются после фоновой рамки и располагаются в таких пространственных позициях, которые соответствуют заполняемым информационным полям бланка. Рамки-поля тоже желательно оставлять без окаймления: тогда они воспринимаются как ячейки на текстовом фоне.

Из-за того что фоновая рамка располагается пространственно вокруг, а не слева или сверху от рамок-полей, нажатие любой из клавиш-стрелок не переводит курсор с полей на эту фоновую рамку, она оказывается как бы недоступной для курсора, и поэтому пользователь будет воспринимать ее не как рамку с информацией, а как структурное оформление экрана. Если же разработчику базы данных потребуется установить курсор на эту рамку, то клавиши передвижения по логической цепочке {УПР-ВЛЕВО} и {УПР-ВПРАВО} всегда позволят сделать это. Логическое положение фоновой рамки – перед рамками- полями – тоже существенно, поскольку из-за этого она изображается первой, а рамки-поля располагаются уже на ее фоне.

Желательно дать фоновой рамке какое-то фиксированное название (единое для всех рамок-записей), что упростит программирование интерфейсов. В стандартной диалоговой оболочке для

этого принято имя **бдфон**, содержащее латинские и русские буквы для того, чтобы снизить вероятность случайного совпадения этого имени с именами полей.

Структура табличного доступа

Электронная таблица, обеспечивающая табличный доступ к записям, размещается внутри рамки-записи и пространственно расположена под фоновой текстовой рамкой. Эта рамка, подобно фоновой, не имеет окаймления и по размерам совпадает с полем зрения рамки-записи. Поэтому у пользователя, когда он попадает внутрь этой таблицы, сохраняется впечатление, что он остается в той же самой рамке-записи, которая как бы изменила свою внутреннюю структуру. Так же как и фоновая рамка, табличная имеет единое для всех рамок-записей имя **bdtab**.

Верхняя строка этой таблицы содержит имена полей в произвольном порядке и не обязательно в полном составе. Кроме того, один из столбцов – А – используется в этой рамке для служебных целей. В нем располагаются уникальные ключи записей, выбираемых в таблицу. Для того чтобы этот столбец приобрел такой служебный статус, в ячейке А1 вместо имени поля располагается символ ******. Вторая строка таблицы используется только для горизонтальной линии разграфки, отделяющей имена полей от значений, и лишь ячейка А2 будет использоваться в служебных целях – для запоминания количества записей, выбранных в таблицу.

Переключение между табличным и форматным режимами

Поле зрения рамки-записи может быть сдвинуто либо на фоновую, либо на табличную рамки. В зависимости от этого сдвига рамка-запись выглядит для пользователя как форматный бланк или как таблица. Программам, работающим с рамкой-записью, требуется возможность определять, в каком из двух состояний она находится, чтобы по-разному выполнять соответствующие интерфейсные функции. Будем отмечать различие этих состояний открытостью или закрытостью табличной рамки. Если табличная рамка **bdtab** открыта, то будем считать, что рамка-запись находится в табличном состоянии, если же **bdtab** закрыта, то рамка-запись находится в форматном состоянии. Такая интерпретация нуждается еще в том, чтобы поддерживать правильное положение поля зрения рамки-записи – на табличной или на фоновой рамке соответственно. Эти соглашения удобно поддерживать, пользуясь набором трех функций, тексты которых приводятся ниже. Эти функции имеются в стандартной диалого-

вой оболочке МАСТЕРа и поэтому ими можно пользоваться, как если бы они были встроенными (если, конечно, диалоговая оболочка не была изменена вами). Форматы обращения к этим функциям таковы:

```
следПорция(рамка-запись)
предПорция(рамка-запись)
вТабличномСостоянии(рамка-запись) --> 0/1
вФорматку(рамка-запись)
вТаблицу(рамка-запись)
```

Первые две функции служат для того, чтобы выбирать информацию в рамку-запись независимо от того состояния (форматного или табличного), в котором она находится. Их можно рассматривать как обобщения встроенных функций НайтиСлед() и НайтиПред() с простых рамок-записей на таблично-форматные рамки-записи. Эти функции выбирают в рамку-запись следующую или предыдущую порцию данных. Под порцией понимается либо просто следующая или предыдущая запись (когда рамка-запись находится в форматном состоянии), либо следующие или предыдущие 20 записей (когда рамка-запись находится в табличном состоянии).

```
следПорция
Блок(1,
    Присв(1, Рамка(Арг(1), 'бдтаб'));
    Если(Открыта(Пер(1)),
        Врамке(Пер(1),
            Присв(!A!2, ВзятьЗапись(Арг(1), !A!3: !A!22))
        )
    ,
    НайтиСлед(Арг(1))
);
ВзятьЗапись(Арг(1))
)
```

```
предПорция
Блок(1,
    Присв(1, Рамка(Арг(1), 'бдтаб'));
    Если(Открыта(Пер(1)),
        НайтиПред(Арг(1), !A!2+20);
        Врамке(Пер(1),
            Присв(!A!2, ВзятьЗапись(Арг(1), !A!3: !A!22))
        )
    ,
    НайтиПред(Арг(1))
);
ВзятьЗапись(Арг(1))
)
```

Функция вТабличномСостоянии() позволяет определить, в каком состоянии – в табличном или форматном – находится заданная рамка-запись. Функция возвращает 1, если рамка-запись находится в табличном состоянии, и 0 – если в форматном.

вТабличномСостоянии

```
Открыта(Рамка(Арг(1), 'бдтаб'))
```

Две последние функции – вФорматку() и вТаблицу() – позволяют переводить рамку-запись соответственно в форматное или в табличное состояние. При каждом из этих переводов на всякий случай корректируются размеры как фоновой, так и табличной рамок для того, чтобы они обязательно совпадали с размерами поля зрения рамки-записи.

10.4. Работа с записями одного типа

Основой для обработки информации в любых базах данных является набор действий над отдельным типом записей. В пределах одного типа записей необходимо уметь создавать новые записи, изменять и уничтожать существующие, последовательно просматривать записи, устанавливая разные порядки просмотра, искать записи по значениям тех или иных полей или по логическим условиям. Все эти действия вы умеете выполнять в диалоговом режиме – через тот стандартный интерфейс, который реализован в диалоговой оболочке МАСТЕРа. Здесь мы рассмотрим те же самые возможности, но с другой точки зрения: с позиции инструментального языка программирования.

Последовательный просмотр записей

Наиболее простыми операциями над записями являются операции, связанные с их последовательным просмотром. Процедурные возможности для организации последовательного просмотра создаются такими встроенными функциями, как НайтиСлед(), НайтиПред(), НайтиНачало(), НайтиКонец(). Типовая схема процедурного последовательного просмотра записей была приведена в предыдущей главе. Здесь мы рассмотрим те же возможности применительно к диалоговому доступу к записям. А именно, рассмотрим способ реализации клавиш {СЛЕД}, {ПРЕД}, {НАЧАЛО} и {КОНЕЦ}, имеющихся в стандартной диалоговой оболочке МАСТЕРа. Для их реализации достаточно обратиться к уже рассмотренным в главе 10.

ренным обобщенным функциям выборки данных, написанных для таблично-форматных рамок-записей.

{СЛЕД}

```
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    следПорция(ОхвРамка())),
Если(Имя(ОхвРамка())='bdтаб',
    следПорция(ОхвРамка(ОхвРамка())))
)
```

{ПРЕД}

```
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    предПорция(ОхвРамка())),
Если(Имя(ОхвРамка())='bdтаб',
    предПорция(ОхвРамка(ОхвРамка())))
)
```

{НАЧАЛО}

```
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    НайтиНачало(ОхвРамка()));
    следПорция(ОхвРамка()),
Если(Имя(ОхвРамка())='bdтаб',
    НайтиНачало(ОхвРамка(ОхвРамка())));
    предПорция(ОхвРамка(ОхвРамка())))
)
```

{КОНЕЦ}

```
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    НайтиКонец(ОхвРамка()));
    следПорция(ОхвРамка()),
Если(Имя(ОхвРамка())='bdтаб',
    НайтиКонец(ОхвРамка(ОхвРамка())));
    предПорция(ОхвРамка(ОхвРамка())))
)
```

Создание, изменение и уничтожение записей

Записи можно создавать как в диалоговом режиме, заполняя соответствующие поля в форматке или в таблице и затем выполнения команду создания через меню, так и программно, когда значения получаются путем алгоритмического вычисления и поля заполняются программой с помощью операций присваивания.

Рассмотрим сначала пример такого программного создания записей. Таким примером может служить процедура тестового заполнения базы данных с целью проверки ее вместимости и эффективности. Перед началом разработки любой серьезной инфор-

мационной системы желательно проверить, сколько места займут реальные объемы данных, которые возникнут в этой базе, какой окажется скорость доступа к записям на этих реальных объемах. Для этого следует еще перед разработкой интерфейсов и обрабатывающих программ запустить процедуру заполнения базы записями со случайно генерируемыми данными. Эта процедура может иметь следующий вид:

```
случайноеЗаполнение
Блок(1,"Пер(1)-счетчик создаваемых записей";
    Присв(1,ЗапросЧис("Сколько записей создавать"));
    Повтор(Пер(1)>0,
        Врамке(запись,
            Присв(поле1,"ЗНАЧ"&&Цел(Случай()*100));
            Присв(поле2,Случай()*20000);
            Присв(поле3,
                Текст(
                    Размножить("*",Случай()*80),
                    Размножить("*",Случай()*80),
                    Размножить("*",Случай()*80)
                )
            );
            СоздатьЗапись(запись);
            Присв(1,Пер(1)-1)
        );
        СохранитьБазу()
    )
);
```

Данная функция на каждой итерации цикла сначала заполняет случайными числами, строками и текстами три поля рамки-записи, а затем обращается к функции СоздатьЗапись(), отправляющей эти значения в базу данных.

Подобным же образом работает и пользователь в диалоге, с той лишь разницей, что поля заполняются не путем вычисляемого присваивания, а вручную непосредственно с клавиатуры. После заполнения полей должна быть выполнена операция СоздатьЗапись(), для чего пользователь должен либо вызвать меню, либо нажать какую-нибудь функциональную клавишу. В стандартной диалоговой оболочке команда создания записи помещена в меню, которое вызывается при нажатии {F10} изнутри любых рамок-записей. В вашей информационной системе вы можете связать это действие с любой другой клавишей, например с {ВСТ}.

{ВСТ}

```
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    СоздатьЗапись(ОхвРамка()));
    СохранитьБазу()
)
```

Аргументом обращения к функции `СоздатьЗапись()` здесь является не какое-то конкретное имя рамки, а указатель охватывающей рамки, выдаваемый функцией `ОхвРамка()`. Этим достигается то, что клавиша `{ВСТ}` будет пригодна в любой рамке-записи, имеющейся в вашей базе данных. Обращение к функции `СохранитьБазу()` стоит здесь для того, чтобы обеспечить надежность базы. Эта функция, выполняющая сброс буфера из оперативной памяти на диск, должна вызываться после любой модифицирующей операции, за исключением разве только операций, находящихся внутри программных циклов, как в приведенной программе `случайное Заполнение()`.

Довольно часто требуется, чтобы записи некоторого типа обязательно различались между собой по значению какого-либо поля, например поля "код". Такую уникальность следует поддерживать при всех модифицирующих операциях – при создании или изменении записей. Проверка уникальности создаваемой записи может быть сделана в рассмотренной функции следующим образом:

```
{ВСТ}
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    Если(Найти(ОхвРамка(),'код',
        Рамка(ОхвРамка()),'код'),0
    ),
    Ошибка('Такая запись уже есть'),
    СоздатьЗапись(ОхвРамка());
    СохранитьБазу()
)
)
```

Совершенно аналогичным образом реализуются остальные операции над записью, действующие через рамку-запись. Так, уничтожение записи, связываемое с нажатием клавиши `{УДЛ}`, можно реализовать в виде функции с именем `{УДЛ}`:

```
{УДЛ}
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    УничтожитьЗапись(ОхвРамка());
    СохранитьБазу();
    ВзятьЗапись(ОхвРамка())
)
)
```

В данной операции требуется дополнительное обращение к функции `ВзятьЗапись()`. Это нужно для того, чтобы значения полей уничтоженной записи не оставались лежать в рамках-

лях, а заменялись на значения полей той записи, которая окажется текущей после уничтожения записи.

Изменение записи выполняется аналогично созданию: пользователь сначала вызывает некоторую интересующую его запись, изменяет значения полей и нажимает клавишу, которая отправляет сделанные изменения в базу данных. Пусть такой клавишей будет `{ЗАГ-ВСТ}`:

```
{ЗАГ-ВСТ}
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    ИзменитьЗапись(ОхвРамка());
    СохранитьБазу()
)
)
```

Эта операция реализована без контроля уникальности какого-либо поля. Если вы хотите, чтобы все записи различались по какому-то полю (скажем, полю "код"), то приведенную функцию можно изменить, одновременно объединив ее с функцией создания новых записей `{ВСТ}`. Новая функция `{ВСТ}` будет анализировать поле "код" и при отсутствии в базе данных записи с заданным значением будет создавать запись, а при наличии – находить и модифицировать существующую запись. Процедура, реализующая такую логику, должна быть следующей:

```
{ВСТ}
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    Если(Найти(ОхвРамка(),'код',
        Рамка(ОхвРамка()),'код'),0
    ),
    ИзменитьЗапись(ОхвРамка())
    ,
    СоздатьЗапись(ОхвРамка())
);
СохранитьБазу()
)
)
```

Совсем другая логика создания или изменения записей может быть реализована в том случае, когда требуется минимизировать число нажатий клавиш при интенсивном заполнении базы. Определим для каждой рамки-поля фильтр функции `Выйти()`. Этот фильтр будет срабатывать при завершении ввода значения в соответствующую рамку. Обратите внимание, что даже для простых рамок, в которые явного входа никогда не делается, этот фильтр имеет смысл, поскольку считается, что вход и выход в рамку производится неявно соответственно в начале и в конце ввода значения. Итак, если в вашей записи имеются рамки-поля

поле1, поле2, поле3, то соответствующие фильтры функции `выйти()` для этих рамок могут быть следующими:

для рамки поле1: `устTek(поле2)`
для рамки поле2: `устTek(поле3)`
для рамки поле3: `{ВСТ}();устTek(поле1)`

В результате пользователю не потребуется нажатий даже для перевода курсора от поля к полю: с первого поля курсор будет автоматически переходить на второе, со второго – на третье, а после заполнения третьего поля сработает обращение к функции `{ВСТ}`, и курсор возвратится на первое поле.

Те же самые диалоговые операции можно реализовать и для табличного состояния рамок-записей.

Для этого потребуется одна вспомогательная функция, выполняющая уничтожение уникального ключа в текущей строке для того, чтобы пометить эту строку как новую запись. Это можно реализовать в виде функции `{ЗАГ-ВСТ}`:

```
{ЗАГ-ВСТ}
Если(Имя(ОхвРамка())='бдтаб',
    Очистить(Яч(ПервГор(ТабКурсор()),1))
)
```

Для выполнения создания и изменения записей в табличном состоянии можно обобщить уже написанную функцию `{ВСТ}`, распространив ее действие на табличные рамки. В таком обобщенном виде функция будет выглядеть следующим образом:

```
{ВСТ}
Если(Тип(ОхвРамка())=МНОЖЕСТВО,
    Если(Найти(ОхвРамка(), 'код',
        Рамка(ОхвРамка()), 'код'), 0
    ),
    ИзменитьЗапись(ОхвРамка())
    ,
    СоздатьЗапись(ОхвРамка())
)
,
Если(Имя(ОхвРамка())='бдтаб',
    СоздатьЗапись(ОхвРамка(ОхвРамка()), !А13:!АЯ!22);
    ИзменитьЗапись(ОхвРамка(ОхвРамка()), !А13:!АЯ!22);
    Присв(!А!2,
        ВзятьЗапись(ОхвРамка(ОхвРамка()), !А13:!АЯ!22)
    )
);
СохранитьБазу()
```

В этой функции добавлена еще одна логическая ветвь, отрабатывающая в случае, если курсор находится в табличной рамке `бдтаб`. Действие этой ветви состоит в том, что к полному диапазону таблицы применяется сначала функция `СоздатьЗапись()`, затем `ИзменитьЗапись()`. Первая функция найдет все те строки, в которых были вставлены новые записи, и создаст по этим строкам новые записи.

Упорядочение множества записей

В реализации диалога требуются операции настройки рамки-записи на тот или иной порядок. Нам потребуется операция, автоматически выбирающая порядок, в котором ключевым является текущее поле. Текущее поле в форматном состоянии рамки-записи определяется той рамкой, которая является текущей, а в табличном состоянии – столбцом, в котором стоит курсор. Функция `переключитьПорядок()` подыскивает подходящий порядок и присваивает рамке-записи указатель этого порядка. Возвращаемым значением при этом является 1. Если же подходящего порядка не найдено (это случится тогда, когда в базе данных нет порядка с текущим полем в качестве ключевого), то состояние рамки-записи не изменяется, а функция возвращает значение 0. Аргументом у функции служит указатель рамки-записи, в которой выполняется переключение порядка.

```
переключитьПорядок
Блок(3,
    "Определим имя текущего поля";
    Если(вТабличномСостоянии(Арг(1)),
        Присв(3,Рамка(Арг(1), 'бдтаб'));
        Присв(1,Врамке(Пер(3),
            Извлечь(Яч(1,ПервВер(ТабКурсор(Пер(3))))))
        )),
        Присв(1,Имя(ТекРамка(Арг(1))))
    );
    "Попытаемся найти порядок по возрастанию поля";
    Присв(2,ВзятьПорядок(-1,Арг(1),1,Пер(1),1);
    Если(Тип(Пер(2))=ПУСТО,
        "Попытаемся найти порядок по убыванию поля";
        Присв(2,ВзятьПорядок(-1,Арг(1),1,Пер(1),-1)
    );
    "Переключим порядок, если он был найден";
    Если(Тип(Пер(2))#ПУСТО,
        Присв(Арг(1),Пер(2));
        СледПорядка(Арг(1));
        1,
        0
    )
)
```

При наличии такой функции легко запрограммировать переключение на тот порядок, который упорядочен по текущему полю. Это переключение можно связать, скажем, с клавишей {ДОП-Ф1}:

```
{ДОП-Ф1}—  
переключитьПорядок(  
    Если(Имя(ОхвРамка())='бдтаб',  
        ОхвРамка(ОхвРамка()),  
        ОхвРамка())  
    )  
)
```

Поиск записей

Следующая типичная задача, связанная с одиночными записями, состоит в том, чтобы находить запись по заданному пользователем значению какого-либо поля. Операцию поиска можно связать с некоторой функциональной клавишей, например, как в текстовом процессоре, с {УПР-Ф1}. Функцию для клавиши {УПР-Ф1} можно запрограммировать следующим образом:

```
{УПР-Ф1}—  
Найти(ОхвРамка(), Имя(ТекРамка()), ТекРамка(), 0);  
ВзятьЗапись(ОхвРамка())
```

Реализация этой функции ориентирована на то, что курсор находится на рамке-поле, по которому нужно найти запись. Искомое значение должно быть введено в эту рамку перед нажатием {УПР-Ф1}. Последний аргумент функции Найти(), равный нулю, означает, что для поиска необязательно использовать тот порядок, который является текущим, и в результате порядок может замениться на какой-то другой – упорядоченный по тому полю, по которому ведется поиск. Второй оператор – ВзятьЗапись(ОхвРамка()) – требуется для того, чтобы найденная запись не только стала текущей, но еще и была выбрана в рамку-форматку (если не сделать этого, то возникнет рассогласование между текущей позицией и содержимым рамок-полей).

У этой реализации есть один недостаток. Он состоит в том, что поиск начинается независимо от того, существует ли в базе порядок, упорядочивающий записи по искомому полю, или не существует. Если окажется, что такого порядка нет, поиск будет выполняться путем полного перебора, что приведет, быть может, к многоминутному ожиданию. Для того чтобы этого не происходило, функцию можно видоизменить, используя запрограммированную ранее функцию НайтиПорядок():

272

{УПР-Ф1}—

```
Если(переключитьПорядок(ОхвРамка(), Имя(ТекРамка())),  
    Ошибка('Нет порядка по этому полю')  
,  
    Найти(ОхвРамка(), Имя(ТекРамка()), ТекРамка(), 0);  
    ВзятьЗапись(ОхвРамка())  
)
```

Еще одна особенность этой операции связана с тем, что в приведенных формулах не делалась проверка того, была ли найдена запрошенная запись. Это приведет к тому, что при отсутствии точного искомого значения будет найдена ближайшая по значению запись. Можно изменить логику поиска так, чтобы при неуспехе поиска в рамке-записи никаких изменений не происходило и оставалась текущей та же запись, которая была перед поиском. Это нужно запрограммировать так:

{УПР-Ф1}—

```
Блок(1,  
    Присв(1, Извлечь(ОхвРамка()));  
    Если(~Найти(Пер(1), Имя(ТекРамка()), ТекРамка(), 0),  
        Ошибка('Нет такой записи')  
,  
        Присв(ОхвРамка(), Пер(1));  
        ВзятьЗапись(ОхвРамка())  
)  
)
```

Отличие этой реализации функции от предыдущей состоит в том, что поиск в функции Найти() выполняется не сразу в рамке-записи, а в локальной переменной Пер(1). Поэтому при неуспешном поиске текущий порядок и текущая запись в рамке-записи останутся неизменными, и только в случае успеха порядок с найденной в нем позицией записи будет присвоен рамке-записи.

10.5. Ссылки от записей к записям

Реляционные ссылки

Основным понятием, используемым для образования структурных связей между записями, является понятие **ссылки**. Пользуясь реляционным принципом образования связей, будем считать, что запись ЗАП1 ссылается своим полем П1 на поле П2 записи ЗАП2, если значения этих полей совпадают по смыслу и содержат одинаковые значения.

Например, если в записях типа Сотрудник имеется поле кодОтдела и в записях типа Отдел имеется поле кодОтдела, то можно считать, что каждая запись о сотруднике полем кодОтдела ссылается на некоторую запись Отдел (содержащую в поле кодОтдела то же значение, что и поле кодОтдела в записи о сотруднике). И наоборот, каждая запись типа Отдел своим полем кодОтдела ссылается на несколько записей типа Сотрудник (на те, которые содержат в поле кодОтдела то же значение, что и поле кодОтдела в записи об отделе).

По своему смыслу ссылки могут быть либо однозначными, либо многозначными. Это определяется только смысловым содержанием полей. Так, первая из названных ссылок

`Сотрудник.кодОтдела --> Отдел.кодОтдела`

является однозначной, поскольку каждому сотруднику соответствует лишь один отдел, в котором он работает. Вторая ссылка

`Отдел.кодОтдела -->>> Сотрудник.кодОтдела`

является многозначной, поскольку в одном отделе работают несколько сотрудников.

Процедурные схемы реализации ссылок

Процедурной схемой для реализации однозначной ссылки является следующая условная программа:

```
однозначнаяСсылка—  
ВзятьЗапись(первый тип записи);  
Найти(  
    второй тип записи,  
    "поле второго типа записи",  
    первый тип записи.поле первого типа записи,  
    0  
);  
ВзятьЗапись(второй тип записи)
```

Этой процедурной схемой описывается то, как по заданной текущей записи первого типа, используя значение одного из полей в качестве ссылки, находится запись второго типа. Так происходит переход по ссылкам от записи к записи. Вторая процедурная схема реализует многозначную ссылку. Эта схема в принципе похожа на предыдущую, но в ней вместо поиска применяется функция Ограничить(). Кроме того, одним только поис-

ком дело не завершается. Требуется еще перебрать те записи, на которые ссылается данная многозначная ссылка, для того, чтобы выполнить над каждой из них какое-то действие. Перебор записей осуществляется обычной процедурной схемой последовательного просмотра.

многозначнаяСсылка—

```
ВзятьЗапись(первый тип записи);  
Ограничить(
```

```
    второй тип записи,  
    "поле второго типа записи",  
    первый тип записи.поле первого типа записи,  
    первый тип записи.поле первого типа записи,  
    0
```

)

```
Повтор(ВзятьЗапись(второй тип записи),  
    обработка очередной записи второго типа;  
    НайтиСлед(второй тип записи)  
)
```

Ссылки в таблично-форматных рамках

Применим рассмотренные процедурные схемы к тому, чтобы связи между записями отображались в рамках с учетом их форматного или табличного состояния. Срабатывание однозначной ссылки должно автоматически переводить рамку-запись, на которую делается ссылка, в форматное состояние и показывать в этой рамке ту запись, которая найдена по ссылке. Срабатывание многозначной ссылки должно переводить рамку-запись в табличное состояние и показывать в ней первую порцию тех записей, которые выделены по ссылке.

Функции, обрабатывающие ссылки таким образом, можно запрограммировать обобщенным способом, независимым от конкретного типа записи. Эти функции должны иметь следующие форматы обращения:

```
однозначнаяСсылка(рамка-запись, имя поля, значение)  
многозначнаяСсылка(рамка-запись, имя поля, значение)
```

В качестве первого аргумента обеим функциям передается та рамка-запись, на которую делается ссылка, вторым аргументом является имя искомого поля, а третьим аргументом – искомое значение.

однозначнаяСсылка—

```
вФорматку(Арг(1));
```

```
Найти(Арг(1), Арг(2), Арг(3), 0);
```

```
ВзятьЗапись(Арг(1))
```

многозначнаяСсылка
вТаблицу(Арг(1));
Ограничить(Арг(1), Арг(2), Арг(3), Арг(3), 0);
НайтиНачало(Арг(1))
следПорция(Арг(1))

10.6. Типовые связи сетевой модели данных

Одной из очень распространенных концепций СУБД является так называемая **сетевая модель**. Создание баз данных в МАСТЕРе вполне можно выполнять, руководствуясь концепциями этой модели. Рассмотрим, каким образом инструментальными средствами Мастера можно представлять основные понятия сетевой модели и в каких приложениях их можно использовать.

Наборы

Основным понятием, на котором строятся структуры сетевых баз данных, является понятие набора. **Набор** – это такая связь между записями, которая выражает отношения типа "отец-сыновья", "целое-части", "начальник-подчиненные" и т.п. В каждом наборе есть одна запись, которая называется его **владельцем** (для приведенных примеров это: отец, целое, начальник) и несколько записей, называющихся его членами (сыновья, части, подчиненные). Записи-владельцы и записи-члены могут быть как разных, так и совпадающих типов.

В СУБД МАСТЕРа набор проще всего реализовывать комбинацией взаимообратных многозначной и однозначной ссылок: связь записи-владельца с записями-членами реализуется многозначной ссылкой, а обратная связь – от записи-члена к соответствующей записи-владельцу – однозначной ссылкой.

Так, например, можно создать в базе данных следующую структуру. В записях-владельцах разместим поле "код", служащее для однозначной идентификации этих записей, а в записях-членах разместим поле "владелец", в котором будем размещать код той записи-владельца, которой данная запись-член принадлежит. Набор представляется здесь парой ссылок:

владелец.код <-->> член.владелец

Многозначные соотношения

С абстрактной точки зрения наборы представляют отношения типов "один ко многим" и "многие к одному". Часто встречаются и более сложные отношения типа "многие ко многим". Рас-

мотрим, например, два типа записей: сотрудник и тема. Будем считать, что в каждом из них имеется поле "код", предназначено для однозначной идентификации соответствующих записей. Каждый сотрудник занимается несколькими темами. Поэтому данную связь следует рассматривать как набор, в котором владельцем является сотрудник, а членами – темы. Однако, с другой стороны, каждой темой занимается несколько сотрудников. Так что возникает обратный набор с темами в качестве владельцев и сотрудниками в качестве членов. При таком соотношении невозможно использовать рассмотренную структуру данных. Действительно, если для представления набора

сотрудник -->> тема

ввести, как и ранее, поле "сотрудник" в запись типа "тема", то записать, что данной темой занимаются несколько сотрудников, окажется невозможным. По подобной причине невозможно и в запись "сотрудник" ввести поле "тема", поскольку одного такого поля и здесь будет недостаточно. Существуют два способа решения этой проблемы.

Первый способ состоит в том, чтобы ввести запись-посредник. Назовем этот тип "задание". Записи этого типа будут иметь только два поля: "сотрудник" и "тема", в которых будут располагаться коды соответственно сотрудников и тем. Каждая запись такого типа будет представлять тот факт, что некоторый сотрудник занимается определенной темой. В сущности, мы получили два набора:

сотрудник.код <---->>> задание.сотрудник
тема.код <---->>> задание.тема

Рассмотрим процедурные схемы работы с таким многозначным отношением. Здесь требуется возможность по заданному сотруднику выделять все порученные ему темы и по заданной теме выделять всех сотрудников, занимающихся ею. Процедуры совершенно симметричны, поэтому мы рассмотрим лишь первую из них.

выделитьИ обработать Темы
Ограничить(задание, "сотрудник",
 сотрудник.код, сотрудник.код, 0
);
Повтор(ВзятьЗапись(задание),
 Найти(тема, "код", задание.тема, 0);
 ВзятьЗапись(тема);
 обработать очередную тему;
 НайтиСлед(задание)
)

Нетрудно видеть, что в этой функции мы воспользовались комбинацией двух уже рассмотренных процедурных схем. Вся конструкция в целом реализует схему многозначной ссылки

сотрудник.код -->>> задание.сотрудник

Внутренние строки, выделенные курсивом, реализуют другую процедурную схему, в данном случае однозначную ссылку

задание.тема --> тема.код

Это вполне соответствует формальному содержанию многозначной связи как комбинации двух встречных наборов.

Существует и другой способ реализации такого многозначного отношения. Он более прост в отношении как структурного представления, так и процедурной реализации. Однако, его можно использовать только в тех случаях, когда в одном из наборов может быть наложено ограничение на максимальное число членов. Например, в рассмотренном примере можно считать, что каждый сотрудник занимается одновременно не более чем тремя темами. Тогда ссылки на эти темы можно разместить в качестве полей непосредственно внутри записи сотрудник. Поэтому потребности в записях-посредниках не возникнет.

Итак, введем в запись типа **сотрудник** три поля: **тема1**, **тема2**, **тема3**, в которые будем помещать коды тем. База данных МАСТЕРа не требует обязательного наличия в каждом поле какого-то значения. Некоторые поля могут оставаться неопределенными (т.е. будут содержать значения типа пусто). Это свойство полезно в данном случае, поскольку не каждый сотрудник занимается тремя темами. У большинства из них тема будет всего одна либо две.

Рассмотрим, какими процедурными схемами будет теперь реализовываться то же многозначное отношение. Сначала рассмотрим набор тем, заданных одному сотруднику. Перебор этих тем выполняется следующей процедурой:

обработатьТемы
блок(1,
 Присв(1,1);
 Повтор(Пер(1)<=3,
 Найти(тема,"код",
 Рамка(сотрудник,"тема"&&Пер(1)),0
);
 ВзятьЗапись(тема);
 обработать очередную тему;
 Присв(1,Пер(1)+1)
)
)

Приведенный пример показателен в том отношении, что имена полей тема1, тема2 и тема3 задаются не явно, а вычисляются путем соединения подстроки тема и номера темы, изменяющейся в цикле. Это является аналогом индексации векторного поля.

Реализация обратного набора – сотрудников, занятых заданной темой, может быть осуществлена похожим способом: троекратным ограничением множества сотрудников по полям тема1, тема2, тема3. Это может выполнить, например, следующая процедура:

обработатьСотрудников
блок(1,
 Присв(1,1);
 Повтор(Пер(1)<=3,
 Ограничить(сотрудник,"тема"&&Пер(1),
 тема.код,тема.код,0
);
 Повтор(ВзятьЗапись(сотрудник),
 обработать очередного сотрудника;
 НайтиСлед(сотрудник)
);
 Присв(1,Пер(1)+1)
)
)

Однако существует и более простая возможность. Она связана со специальным способом упорядочения, имеющимся в МАСТЕРЕ: по смешению полей. В данном случае нужно создавать для записей "сотрудник" не три порядка с ключевыми словами тема1, тема2, тема3, а один порядок – по смешению этих же полей. Тогда каждая запись о сотруднике будет зарегистрирована в этом порядке столько раз, сколько непустых тем задано в этой записи. При наличии такого порядка предыдущую функцию можно реализовать намного проще:

обработатьСотрудников
Ограничить(сотрудник,"тема1",тема.код,тема.код,0);
Повтор(ВзятьЗапись(сотрудник),
 обработать очередного сотрудника;
 НайтиСлед(сотрудник)
)

Здесь в операции ограничения указано только одно поле тема1. Однако из-за того, что наряду с этим полем в смешении участвуют поля тема2 и тема3, ограничение выделит те записи, в которых заданный код темы содержится либо в поле тема1, либо в поле тема2, либо в поле тема3.

Табличный интерфейс для работы с наборами

При работе с наборами особенно наглядным является табличный режим доступа к записям. Проиллюстрируем это на примере базы данных, содержащих записи трех типов: отдел, сотрудник и тема. Будем считать, что каждый тип записей имеет поле код для однозначной идентификации и что между записями отдел и сотрудник установлено соотношение "один ко многим" с помощью ссылки

отдел.код <-->>> сотрудник.отдел

а между сотрудниками и темами установлено многозначное отношение с помощью порядка по смешению трех полей тема1, тема2, тема3:

сотрудник.тема1
сотрудник.тема2 <<<-->>> тема.код
сотрудник.тема3

Интерфейс к этой базе данных можно реализовать в виде трех таблиц, расположенных рядом друг с другом. В левой таблице располагаются названия отделов, в средней – фамилии сотрудников, а в правой – названия тем. При необходимости каждая из таблиц может быть увеличена по ширине, чтобы дать доступ к остальным полям записей. В этой базе данных нам потребуется форматный доступ к записям, поэтому мы не будем реализовывать не только таблично-форматные рамки, но даже простую процедурную обработку форматного режима доступа. Рамки-записи, необходимые для программирования, расположены где-нибудь сбоку, в невидимой для пользователя области, пользователю предъявим три табличные рамки, с именами отделы, сотрудники, темы, устроенные так же, как и те табличные рамки, которые располагались в таблично-форматных рамках.

Диалоговый интерфейс должен обеспечить возможность получать в таблицах выборки записей, связанных друг с другом соответствующими наборами. Так, в начальном состоянии в таблице отделов должны быть видны названия всех отделов, остальные две таблицы должны быть пустыми. При нажатии клавиши {ПЛЮС} должен "раскрываться" тот объект, на котором стоит курсор. Так, если курсор стоит на названии некоторого отдела, то в рамке справа должны появиться фамилии сотрудников этого отдела и курсор должен перейти в рамку с сотрудниками. Точно так же при нажатии клавиши {ПЛЮС} на фамилию сотрудника должен появиться список тем, которыми он занимается. Клавиша {МИНУС} обеспечивает обратное движение по этой структуре.

При программировании функций будем полагать, что в столбце Б любой из таблиц всегда расположено поле код.

выбрать Сотрудников

"вызывается по нажатию {ПЛЮС} из рамки отделы";
Блок(1,
Присв(1, Извлечь(Яч(ПервГор(ТабКурсор()), 2)));
Ограничить(сотрудник, "отдел", Пер(1), Пер(1), 0);
НайтиНачало(сотрудник);
ВзятьЗапись(сотрудник, !А!3: !АЯ!100);
Войти(сотрудник)
)

выбрать Темы

"вызывается по нажатию {ПЛЮС} из рамки сотрудники";
Блок(2,
Присв(1, ПервГор(ТабКурсор()));
Если(Тип(Яч(Пер(1), 4))=ЧИСЛО, "столбец 4-поле тема1";
Найти(тема, "код", Яч(Пер(1), 4), 0)
);
ВзятьЗапись(тема, !А!3);
Если(Тип(Яч(Пер(1), 5))=ЧИСЛО, "столбец 5-поле тема2";
Найти(тема, "код", Яч(Пер(1), 5), 0)
);
ВзятьЗапись(тема, !А!4, !А!1: !АЯ!1);
Если(Тип(Яч(Пер(1), 6))=ЧИСЛО, "столбец 6-поле тема3";
Найти(тема, "код", Яч(Пер(1), 6), 0)
);
ВзятьЗапись(тема, !А!3, !А!1: !АЯ!1);
Войти(темы)
)

возврат

"вызывается по нажатию {МИНУС}";
"из таблиц сотрудник и тема";
Очистить(!А!2: !АЯ!100);
Войти(ПредРамка(ОхвРамка()))

Иерархическая структура данных

Рассмотренная пара записей "отдел-сотрудник" представляет собой простейший пример иерархической структуры. Другим типичным примером является рекурсия, образуемая на одном типе записей. Такую рекурсию можно встретить в базе данных, представляющей иерархическую структуру устройства сложной технической конструкции. Записи базы данных относятся к от-

~~дельным узлам и агрегатам устройства, а иерархия представлена в виде конструктивных включений одних узлов в другие.~~

Для представления иерархии в записи узел заведем два поля: **код** и **включение**. Поле **код** обеспечивает однозначную идентификацию узлов, а поле **включение** содержит код **того узла**, в который включен данный узел. Множество записей узел должно быть упорядочено как по полю **код**, так и по полю **включение**.

Организация интерфейса для этой базы данных может использовать тот же принцип соседнего расположения таблиц. В данном случае, однако, число иерархических уровней нефиксировано, и поэтому таблицы лучше порождать и уничтожать автоматически при переходе с уровня на уровень.

10.7. Специальные модели данных

Рассмотрим еще несколько полезных моделей данных, бытых, может, не имеющих общепринятого терминологического обозначения, но столь же полезных в практических информационных системах, как и все рассмотренные до сих пор.

Классификаторы

Весьма важной структурой в базах данных является структура, обеспечивающая кодирование длинных строковых названий. Для такого кодирования используются классификаторы. Классификатор сопоставляет с каждым строковым названием некоторый числовой или символьный код, занимающий значительно меньше места, чем исходное название.

В МАСТЕРе для представления классификаторов можно использовать записи с двумя полями: **код** и **название**. Упорядочение этих записей должны быть как по первому, так и по второму полям. Во всех прочих записях каждый раз, когда требуется значение, представляющее название, можно использовать соответствующий короткий код.

Рассмотрим, например, базу данных, в которой во многих записях должны использоваться поля, обозначающие какие-то организации. Если каждый раз записывать в таких полях полные названия организаций, то, во-первых, будет затрачено большое количество памяти и, во-вторых, в длинном названии какой-либо организации могут быть допущены ошибки и оно в разных записях будет читаться по-разному, что нарушит целостность данных.

Заведем классификатор названий организаций, для чего создадим тип записей **организация** с полями **код** и **название**. Предположим также, что среди прочих типов записей имеется

тип записей **контракт**, описывающий связи между организациями. Положим, что этот тип записей имеет поля **заказчик** и **подрядчик**, обозначающие организации. Будем хранить в этих полях не названия, а коды организаций.

В связи с использованием такой структуры возникает несколько проблем. Во-первых, при занесении в базу новых организаций желательно автоматически генерировать для них уникальные коды. Это может делать, например, следующая функция:

```
новыйКодОрганизации  
Присв(организация,  
      ВзятьПорядок(-1, организация, 1, "код", 1)  
);  
НайтиКонец(организация);  
НайтиПред(организация);  
Если(ВзятьЗапись(организация),  
      организация.код+1,  
      0  
)
```

Эта функция формирует в качестве возвращаемого значения новый код организации, равный числу, на единицу большему, чем максимальный из существующих кодов. Если же ни одной организации в базе еще не существует, то возвращается код 0.

Вторая проблема связана с тем, что для пользователя работа с кодами окажется неестественной, ему требуется предоставить возможность вводить и получать на экране названия, а не коды организаций. Для этого нужно реализовать в прикладной системе функции кодировки и раскодировки названий

```
кодОрганизации(название)  
названиеОрганизации(код)
```

Функция раскодировки – **названиеОрганизации()** – весьма проста. Она реализуется по самой обычной схеме однозначной ссылки.

```
названиеОрганизации  
Найти(организация, "код", Арг(1), 0);  
ВзятьЗапись(организация);  
Извлечь(организация.название)
```

Намного более сложной оказывается функция, вычисляющая по заданному названию организации ее код. Сложность обусловлена тем, что пользователь при вводе длинного названия может ошибиться, а кроме того, у него должна быть возможность вводить название не полностью, а сокращенно.

Функция помещает искомое название в локальную переменную, поскольку во время поиска это название, возможно, придется уточнять. Функция пытается найти организацию с заданным названием. Как только это удается, извлекается значение соот-

вествующего кода и возвращается в качестве результата функции. Если же название не найдено, то берется то название, которое оказалось ближайшим к заданному (функция Найти() при неуспехе поиска устанавливает указатель именно на лексикографическом соседе искомого значения). Найденное таким образом название показывается пользователю, и с помощью функции **Данет()** запрашивается подтверждение его правильности. Если пользователь подтвердит найденное название, то оно присваивается переменной 1, и поэтому на очередной итерации цикла функция **Найти()** сработает успешно, что и завершит цикл. Если же пользователь не подтвердит правильность найденного названия, то тогда задается вопрос о том, не является ли это название новым. При согласии пользователя генерируется новый код и создается очередная строка классификатора. Переменная 1 не изменяет своего значения, но из-за появления нового названия она будет найдена на очередной итерации цикла, что опять же завершит его. Наконец, если пользователь не согласится вводить новое название, то ему дается попытка ввести уточненное название, которое присваивается переменной 1, вся процедура повторяется. Эта процедура оказывается универсальной и решает все задачи, связанные с символическим введением кодированных названий.

```

кодОрганизации
Блок(1,
    Присв(1,Арг(1));
    Повтор(~Найти(организация,"название",Пер(1),1),
        ВзятьЗапись(организация);
        Если(Данет("Верно ли название &&
            организация.название
        ),
        Присв(1,Извлечь(организация.название)),
        Если(Данет("Ввести новое название &&Пер(1))
            Присв(организация.код,
                новыйКодОрганизации())
        );
        Присв(организация.название,Пер(1));
        СоздатьЗапись(организация)
        ,
        Присв(1,ЗапросСтр(
            "Задайте более точное название для &&
            Пер(1)
        ))
    )
);
    ВзятьЗапись(организация);
    Извлечь(организация.код)
)
)
```

При наличии данных функций мы можем видоизменить операции взятия, создания и изменения записей таким образом, чтобы пользователь был избавлен от необходимости вводить коды. Рассмотрим их только для случая форматного интерфейса. Читатель без труда распространит их и на табличный интерфейс.

Для взятия записей типа контракт в форматку потребуется функция, которая помимо выборки записи произведет еще и раскодировку полей **заказчик** и **подрядчик**. Назовем эту функцию **взятьКонтракт()**:

```

взятьКонтракт
Врамке(контракт,
    ВзятьЗапись(контракт);
    Присв(заказчик,названиеОрганизации(заказчик));
    Присв(подрядчик,названиеОрганизации(подрядчик));
)
)
```

Операции создания или модификации контракта, связываемые нами с клавишей {ВСТ}, могут быть реализованы следующим образом. Положим, что пользователь, заполняя запись о контракте, вводит в поля **заказчик** и **подрядчик** не коды, а названия организаций. Тогда перед созданием или изменением записи символьные названия нужно будет превратить в соответствующие коды, а потом опять ввести в эти поля названия. Это приводит к следующей модификации процедуры для клавиши {ВСТ}:

```

{ВСТ}
Если(Имя(ОхвРамка())="контракт",
    Врамке(контракт,
        Присв(заказчик,кодОрганизации(заказчик));
        Присв(подрядчик,кодОрганизации(подрядчик));
        Если(Найти(контракт,"номерКонтракта",
            номерКонтракта,0
        ),
        ИзменитьЗапись(контракт),
        СоздатьЗапись(контракт)
        );
        взятьКонтракт();
        СохранитьБазу()
    )
)
)
```

Многомерная модель данных

Еще одна типичная модель, которая часто встречается на практике, может быть названа **многомерной моделью**. Она возникает в тех случаях, когда данные связываются одновременно

с несколькими независимыми атрибутами, а запросы состоят в формировании тех или иных срезов в многомерном пространстве атрибутов.

Рассмотрим в качестве примера следующую производственную модель. Пусть имеются данные об уровне производства нескольких продуктов (первый атрибут – название продукта) по некоторым годам (второй атрибут – год) по некоторым районам (третий атрибут – район). Запрос может состоять в том, чтобы получить таблицу для фиксированного года, в которой столбцы будут соответствовать названиям продуктов, а строки – районам. Другой запрос может состоять в том, чтобы зафиксировать не год, а район, и получить таблицу с годами по столбцам и продуктами по строкам. А вообще говоря, желательно уметь получать произвольные срезы в этом трехмерном пространстве.

Для реализации такой базы данных удобно использовать порядки, организованные по слиянию полей. А именно, заведем порядки, ключами которых будут служить следующие наборы полей:

продукт, район, год
продукт, год, район
район, продукт, год
район, год, продукт
год, продукт, район
год, район, продукт

Заметим, что при таком количестве порядков весьма желательно использование классификаторов во всех трех полях, поскольку в противном случае порядки займут много дополнительного места.

При наличии этих порядков легко реализовать универсальную процедуру, формирующую произвольный срез в данном трехмерном пространстве атрибутов. Пусть формат обращения к этой процедуре будет следующим:

```
выдатьСрез(  
    имя фиксированного поля,  
    значение фиксированного поля,  
    имя поля для строк, значение11, значение12,  
    имя поля для столбцов, значение21, значение22  
)
```

Аргументы этого обращения задают, какое поле должно быть зафиксировано, какое отложено по строкам, а какое – по столбцам таблицы. Для фиксированного поля задается фиксирующее значение, а для двух других полей – по паре ограничивающих значений. На первый взгляд может показаться непонятным,

какой смысл будет иметь диапазон значений для продуктов или для районов. Однако в случае, если кодирование названий проведено по группам, то в диапазон 100–200 могут попасть молочные продукты, а в диапазон 200–300 – мясные. Аналогичным образом можно задавать коды и для районов. Тогда диапазоны будут весьма полезными. Реализована эта функция может быть следующим образом:

```
выдатьСрез—  
Присв(выпуск,  
    ВзятьПорядок(-1, выпуск, 1, Арг(1), 1, Арг(3), 1, Арг(6), 1)  
) ;  
Блок(3, "Пер(1)-строка, (2)-столбец, (3)-значение поля";  
    Присв(1,1);  
    Присв(3,Арг(4));  
    Ограничить(выпуск,1,Арг(2),Арг(2));  
    Повтор(Пер(3)<=Арг(5),  
        Ограничить(выпуск,2,Пер(3),Пер(3));  
        Ограничить(выпуск,3,Арг(7),Арг(8));  
        Присв(2,1);  
        Повтор(ВзятьЗапись(выпуск),  
            Присв(яч(Пер(1),Пер(2)),выпуск.количество);  
            НайтиСлед(выпуск))  
    );  
    Присв(3,Пер(3)+1)  
)
```

Рекомендуется обратить внимание на реализацию этой процедуры. Это единственный пример в книге, в котором, по существу, используются сложные ключи упорядочения. В связи с этим в процедуре используются такие обращения к функции Ограничить(), в которых вместо имени поля задается его ранг в ключе упорядочения. Сначала выполняется ограничение по фиксированному полю, затем по тому полю, которое откладывается по строкам таблицы, и наконец, по последнему полю. Существенно то, что только последнее ограничение может задавать диапазон с отличающимися граничными значениями, а все промежуточные ограничения обязаны быть точечными.

Разумеется, в реальной системе эта процедура должна не только выбирать данные в таблицу, но еще и должным образом оформлять ее, вводя соответствующие названия для строк и столбцов и формируя шапку таблицы. Строки и столбцы должны содержать раскодированные названия значений второго и третьего полей, а в шапке должно находиться раскодированное название значения фиксированного (первого) поля. Все эти добавления легко сможет сделать сам пользователь.

ЗАКЛЮЧЕНИЕ

Вы закончили изучение основных возможностей интегрированной системы МАСТЕР. Эта книга должна была ввести вас и в идеологию интегрированной системы, и в технологию разработки интегрированных баз данных, и в технику программирования на языке нового концептуального уровня. Следующим вашим шагом в этом направлении должна стать практическая работа на компьютере, собственные усилия в проектировании и реализации прикладных информационных систем.

Ограниченный объем и массовая ориентация книги не позволили дать в ней исчерпывающего описания системы. Детальная техническая документация поставляется в комплекте с системой и может содержать некоторые отличия от возможностей, описанных в книге. Так, например, в книге опущены описания некоторых вариантов обращения к функциям, некоторых дополнительных аргументов.

в

Приложения

А. Перечень встроенных функций

Операции и символы языка Мастер

- , - запятая для разделения аргументов в функциях
- ; - точка с запятой – разделяет последовательно вычисляемые формулы
- | - логическая связка ИЛИ
- & - логическая связка И
- логическое отрицание НЕ
- = - равенство
- # - неравенство
- < - меньше
- <= - меньше или равно
- > - больше
- >= - больше или равно
- && - соединение строк или рисунковых значений
- +
- сложение чисел
- вычитание чисел
- *
- умножение чисел
- /
- деление чисел
- ^
- возвведение чисел в степень
-
- унарный минус
- +
- унарный плюс
- :
- операция формирования указателя диапазона из двух других указателей диапазонов
- (
- скобки для группирования операций и для обращений к функциям
-)

Управляющие функции

Если (условие, формула [, формула])

--> значение выбранной формулы

Выбор (номер варианта, вариант1, вариант2, . . .)

--> значение выбранного варианта

Блок (число локальных переменных, формула)

--> значение вычисленной формулы

Пер (номер локальной переменной)
 --> значение локальной переменной
 Присв (номер локальной переменной, формула)
 --> присвоенное значение
 Присв (указатель диапазона, формула)
 --> присвоенное значение
 Присв (рамка, формула)
 --> присвоенное значение
 Повтор (условие1 [, формула [, условие2]])
 --> число сделанных итераций
 Меню ([1/0-запрет клавиши {ОТКАЗ}]
 пункт1, объяснение1, формула1,
 ...
)
 --> значение выбранной формулы
 Врамке (контекстная рамка, формула)
 --> значение формулы
 Ошибка (номер ошибки)
 не делает возврата в вызывающую формулу

Функции над Мастер-формулами

Формула (указатель ячейки)
 --> текст формулы
 Формула (рамка, код клавиши)
 --> текст формулы
 УстФормулу (указатель ячейки, текст формулы)
 --> 1/0-успех компиляции формулы
 УстФормулу (рамка, код клавиши, текст формулы)
 --> 1/0-успех компиляции формулы
 ЕстьФормула (указатель ячейки)
 --> 1/0-наличие формулы
 ЕстьФормула (рамка, код клавиши)
 --> 1/0-наличие формулы

Определение собственных функций

ОпрФунк (рамка)
 --> 1/0-успех компиляции функции в рамке
 ОтмФунк (рамка)
 --> нет значения
 Арг (номер аргумента)
 --> значение аргумента
 ЧисАрг ()
 --> число фактических аргументов обращения

Тип (значение [, 1/0-делать извлечение из указателя])
 --> код типа значения
 Вып (рамка, формула1, формула2, ...)
 --> значение, выработанное вызванной функцией
 Результат (формула)
 --> значение ее аргумента становится возвращаемым
 значением той формулы, в которой стоит обращение
 к Результат()
 Трассировка (0/1/2 [, текстовая рамка])
 --> текущий уровень трассировки

Функции над электронными таблицами

Пересчет (диапазон [, 1/0-по строкам/по столбцам])
 --> нет значения
 ВставитьГор (диапазон)
 --> диапазон
 ВставитьВер (диапазон)
 --> диапазон
 УничтожитьГор (диапазон)
 --> диапазон
 УничтожитьВер (диапазон)
 --> диапазон
 Копировать (диапазон-источник, диапазон-получатель)
 --> диапазон
 Заполнить (диапазон, начальное значение, приращение)
 --> диапазон
 Очистить (диапазон)
 --> диапазон
 Ширина (диапазон)
 --> суммарная ширина диапазона
 Ширина (диапазон, ширина)
 --> диапазон
 Формат (диапазон)
 --> код формата верхней левой ячейки диапазона
 Формат (диапазон, код формата)
 --> диапазон
 Разграфка (диапазон)
 --> код разграфки верхней левой ячейки диапазона
 Разграфка (диапазон, код разграфки)
 --> диапазон
 ПервГор (диапазон)
 --> номер первой строки диапазона
 ПервВер (диапазон)
 --> номер первого столбца диапазона

ПослГор (диапазон)
 --> номер последней строки диапазона
 ПослВер (диапазон)
 --> номер последнего столбца диапазона
 Сумма (диапазон/рамка/число, . . .)
 --> сумма всех числовых аргументов
 Колич (диапазон/рамка/число, . . .)
 --> количество всех числовых аргументов
 Средн (диапазон/рамка/число, . . .)
 --> среднее арифметическое всех числовых аргументов
 Дисперс (диапазон/рамка/число, . . .)
 --> дисперсия всех числовых аргументов
 Макс (диапазон/рамка/число, . . .)
 --> максимальное из всех числовых значений
 Мин (диапазон/рамка/число, . . .)
 --> минимальное из всех числовых значений
 ТабПоиск (диапазон, искомое значение
 [, отступ [, упорядоченность]]
)
 --> диапазон или нет значения
 ШапкаБоковик (табличная рамка)
 --> диапазон
 ШапкаБоковик (диапазон)
 --> нет значения
 ТабТекст (диапазон)
 --> текст
 ТабТекст (диапазон, имя файла, 1/0-делать дозапись)
 --> нет значения
 ТекстТаб (текст, диапазон, 1/0-делать разбор строк)
 --> нет значения
 Яч (номер строки, номер столбца)
 --> диапазон
 Извлечь (диапазон)
 --> значение, содержащееся в верхней левой ячейке
 ТабКурсор ()
 --> диапазон: ячейка, на которой стоит курсор
 ТабКурсор (табличная рамка)
 --> диапазон: ячейка, на которой стоит курсор в этой
 рамке
 ТабКурсор (диапазон)
 --> нет значения
 Указать (строка-приглашение, ДИАПАЗОН [, 1/0-зацеплен])
 --> диапазон
 РежимТаблицы (табличная рамка [, код режима])
 --> код режима

График (
 заглавие графика,
 диапазон имен области определения,
 1/2/3-способ комбинации нескольких функций,
 надпись вдоль оси области значений (или '''),
 минимальная точка области значений (или '''),
 максимальная точка области значений (или '''),
 число отсечек на оси значений (или '''),
 надпись вдоль оси области определения (или '''),
 минимальная точка области определения (или '''),
 максимальная точка области определения (или '''),
 число отсечек на оси определения (или '''),
 диапазон имен областей значения,
 тип графика,
 параметр типа графика,
 . . .
 . . .
 . . .
)
 --> рисунок

Числовые функции

Абс (число) --> абсолютное значение числа
 Цел (число) --> ближайшее меньшее целое число
 БольЦел (число) --> ближайшее большее целое число
 Окр (число) --> ближайшее целое число
 Ост (число1, число2)
 --> остаток от деления число1/число2
 Случай () --> случайное число в интервале 0 – 1.
 Сейчас () --> число минут от начала суток
 Сегодня () --> число дней, отсчитываемых от 17.09.1993
 Sin (число) --> синус числа
 Cos (число) --> косинус числа
 Tg (число) --> тангенс числа
 Ctg (число) --> котангенс числа
 Arcsin (число) --> арксинус числа
 Arccos (число) --> арккосинус числа
 Arctg (число) --> арктангенс числа
 ArcCtg (число) --> арккотангенс числа
 Sh (число) --> гиперболический синус числа
 Ch (число) --> гиперболический косинус числа
 Sqrt (число) --> квадратный корень числа
 Ln (число) --> натуральный логарифм числа
 Exp (число) --> экспонента числа

Функции над строками

Строка (число [, код формата])
--> строковое изображение числа
Число (строка [, код формата])
--> числовое значение, считанное из строки
Строчная (строка)
--> строка, преобразованная к строчным буквам
Прописная (строка)
--> строка, преобразованная к прописным буквам
Длина (строка)
--> число символов в строке
Подстрока (строка , начальная позиция , длина подстроки)
--> подстрока, выбранная из строки
Размножить (строка, число повторений)
--> размноженная строка
КодКлавиши (строковое имя клавиши)
--> код клавиши
ИмяКлавиши (код клавиши)
--> строковое имя клавиши

Функции над текстами

Параметр (текст, номер параметра [, значение параметра])
--> значение параметра
Подстрока (текст для лексического разбора,
1/0-убирать переносы, 1/0-во фрагменте, 1/0-уничтожать)
--> нет результата
Подстрока (текст)
--> строка - очередная выбранная лексема
Текст ([текст-получатель,] строка/текст, . . .)
--> текст (этот же текст остается в первом аргументе)
СледСтр (текст [, число строк])
--> число строк
НайтиСтр (текст, строка поиска)
--> 1/0-успех поиска
ЗаменитьСтр(текст, строка поиска, строка замены)
--> 1/0-успех замены (наличие в текущей позиции
строки поиска)
ВзятьСтр (текст)
--> текущая строка текста
ВзятьТекст (текст, число строк, 1/0-уничтожать)
--> текст фрагмента

Графические функции

Рисунок (число, . . .)
--> рисунок
Нарисовать (рисунок)
--> рисунок
График() - см. выше в разделе табличных функций

Функции организации диалога

Экран ([режим экрана, цвет экрана])
--> режим экрана
Статус ([1/0-наличие статус строки])
--> 1/0-наличие статус строки
Показать ()
--> нет значения
Показать (рамка, 1/0-признак необходимости показа)
--> нет значения
ЗапросЧис (приглашение [, значение по умолчанию])
--> число
ЗапросСтр (приглашение [, значение по умолчанию])
--> число
ЗапросТекст (приглашение [, значение по умолчанию])
--> текст
Клавиша ([1/0-ждать нажатия])
--> код нажатой клавиши
Клавиатура ([1/0-латинское состояние клавиатуры])
--> латинское состояние клавиатуры
Сообщить (сообщение [, 0=не ждать отклика])
--> строка сообщения
СообщитьЗнач (сообщение [, 0=не ждать отклика])
--> строка сообщения
Звук ([период, длительность в 0.01 сек, . . .])
--> нет значения

Функции над рамками

Создать (
охватывающая рамка,
имя рамки, тип рамки, код окаймления и открытости,
позиция по вертикали, позиция по горизонтали,
высота, ширина, цветовой атрибут
)
--> созданная рамка

Уничтожить (рамка)
 --> нет значения
Очистить (рамка)
 --> рамка
Переставить (переставляемая рамка, рамка-позиция)
 --> нет значения
Извлечь (рамка)
 --> значение, содержащееся в рамке
Записать (имя файла, рамка)
 --> нет значения
Записать (имя файла, текст [, 1/0-дозапись])
 --> нет значения
Записать (имя файла, диапазон [, 1/0-дозапись])
 --> нет значения
Считать (имя файла, охватывающая рамка)
 --> рамка или нет значения
Считать (имя файла, диапазон)
 --> 1/0-успех считывания
ОхвРамка ([рамка])
 --> охватывающая рамка
ТекРамка ([рамка])
 --> текущая рамка или нет значения
УстТек (рамка)
 --> рамка
УстТек (охватывающая рамка, строковое имя рамки)
 --> рамка
ВнешРамка ()
 --> корневая рамка - рабочее поле МАСТЕРа
СледРамка (рамка [, число рамок])
 --> рамка или нет значения
ПредРамка (рамка [, число рамок])
 --> рамка или нет значения
Рамка ([охватывающая рамка,] имя рамки)
 --> рамка или нет значения
Указать (приглашение, РАМКА)
 --> рамка
Имя (рамка [, имя рамки])
 --> имя рамки
Кайма (рамка [, код окаймления рамки])
 --> код окаймления рамки
Открыть (рамка)
 --> рамка
Закрыть (рамка)
 --> рамка
Распахнуть ()
 --> нет значения

ПозСтр (рамка [, позиция рамки по вертикали])
 --> позиция рамки по вертикали
ПозСтЛБ (рамка [, позиция рамки по горизонтали])
 --> позиция рамки по горизонтали
Высота (рамка [, высота рамки])
 --> высота рамки
Ширина (рамка [, ширина рамки])
 --> ширина рамки
Цвет (рамка [, цветовой атрибут рамки])
 --> цветовой атрибут рамки
УстРамку (приглашение)
 --> нет значения (применяется к текущей рамке)
Войти (рамка)
 --> рамка
Выйти ()
 --> рамка
Модифицирована (рамка [, 1/0-признак модифицированности])
 --> 1/0-признак модифицированности

Функции над макропоследовательностями

ОпрМакро (строка-буква [, текст макропоследовательности])
 --> текст макропоследовательности
Исполнить (текст макропоследовательности)
 --> нет значения
Эхо ([1/0-состояние отображения])
 --> 1/0-состояние отображения

Взаимодействие с операционной системой

Конец ()
 Нет возврата. Завершает работу в системе МАСТЕР
СвобПамять ([ЧИСЛО])
 --> ЧИСЛО
Среда (имя параметра среды)
 --> значение параметра среды
Каталог (устройство:)
Каталог (каталог)
Каталог ()
 --> текущее устройство и каталог
Файлы (шаблон имен)
 --> текст-перечень имен, подходящих под шаблон
Вызвать (имя_программы [, командная строка])
 --> 1/0-возможность вызова программы

Функции над базой данных

БазаОткрыта ()

--> 1/0-база находится в открытом состоянии

ОткрытьБазу (число буферных блоков)

--> нет значения

ЗакрытьБазу ()

--> нет значения

СохранитьБазу ()

--> нет значения

СоздатьОбласть (код области, имя области)

--> код области

ОткрытьОбласть (имя области)

--> код области

ЗакрытьОбласть (код области)

--> нет значения

ОбластьОткрыта (код области)

--> 1/0-область с данным кодом открыта

Область (множество)

--> код области, которой принадлежит данное множество

ОчиститьОбласть (код области)

--> нет значения

Множества (код области, вид множеств)

--> текст - перечень имен множеств данного типа

СоздатьМнож (рамка-прототип записи)

--> множество (оно же присваивается в рамку)

ТипПоля (множество, имя поля)

--> строковая спецификация типа и упаковки поля

ИзменитьРамку (рамка-запись)

--> нет значения

ИмяМнож (множество [, имя множества])

--> имя множества

ГлавноеМнож (множество)

--> главное множество того же типа, что и заданное

Полное (множество)

--> 1/0-множество является полным или подмножеством

УничтожитьМнож (множество)

--> нет значения

ВзятьМнож (код области, имя множества)

--> множество

ПолеПорядка (множество [, ранг поля])

--> имя поля

ВзятьПорядок (

код области для создания порядка или -1,

множество нужного типа,

способ упорядочения (1-по слиянию, 2-по смешению),

имя поля,

вид упорядочения (-1-по убыванию, +1-по возрастанию)

. . .

) --> множество или нет значения

ПустоеМнож (

код области для создания подмножества,

множество нужного типа,

способ упорядочения

(0-по вставке, 1-по слиянию, 2-по смешению),

имя поля,

вид упорядочения (-1-по убыванию, +1-по возрастанию)

. . .

) --> множество

НайтиНачало (множество)

--> нет значения

НайтиКонец (множество)

--> нет значения

НайтиСлед (множество [, число пропускаемых записей])

--> число пропущенных записей

НайтиСлед (множество,

имя поля, значение1, значение2,

. . .

[, число пропускаемых записей]

) --> число пропущенных записей

НайтиПред (множество [, число пропускаемых записей])

--> число пропущенных записей

НайтиПред (множество,

имя поля, значение1, значение2,

. . .

[, число пропускаемых записей]

) --> число пропущенных записей

Найти (

множество, имя поля, значение, 1/0-локальность поиска

) --> 1/0-успех поиска

Ключ (множество-порядок по смешению)

--> значение (какого-то из сливаемых полей)

Ограничить (

множество, имя поля, значение1, значение2,

1/0-локальность поиска

) --> 1/0-наличие подходящего порядка

СнятьОграничение (множество)
 --> нет значения
Ограничено (множество)
 --> 1/0-ограничено ли множество
ПрисоединитьПересечение (
 подмножество-получатель,
 множество1, множество2, ...
)
 --> подмножество-получатель
ВзятьЗапись (рамка-запись)
 --> 1/0-наличие текущей записи
ВзятьЗапись (множество)
 --> 1/0-наличие текущей записи
ВзятьЗапись (
 множество, диапазон записей [, диапазон имен]
)
 --> количество записей, взятых в диапазон
ИзменитьЗапись (рамка-запись)
 --> нет значения
ИзменитьЗапись (
 множество, диапазон записей [, диапазон имен]
)
 --> количество измененных записей
СоздатьЗапись (рамка-запись [, множество])
 --> нет значения
СоздатьЗапись (
 множество, диапазон записей [, диапазон имен]
)
 --> количество созданных записей
УничтожитьЗапись (множество)
 --> нет значения
УничтожитьЗапись (
 множество, диапазон записей [, диапазон имен]
)
 --> количество уничтоженных записей
ВключитьЗапись (множество, подмножество)
 --> нет значения
ИсключитьЗапись (подмножество)
 --> нет значения
Поле (множество, имя поля)
 --> значение поля
Поле (рамка-поле)
 --> значение поля

Б. Перечень мнемонических констант

Коды типов значений

ОШИБКА	ПУСТО	ЧИСЛО	СТРОКА	РИСУНОК	ТЕКСТ
ФОРМУЛА	МНОЖЕСТВО	ДИАПАЗОН	РАМКА		

Логические значения

ИСТИНА	= 1
ЛОЖЬ	= 0

Коды типов каймы

ОДИНАРНАЯ	ДВОЙНАЯ
ИМЯ	
ОТКРЫТАЯ	ЗАКРЫТАЯ

Коды форматов ячеек и рамок-полей

ВПРАВО	ВЛЕВО	ВЦЕНТР
СКРЫТЫЙ		

ДЕСЯТИЧНЫЙ	НАУЧНЫЙ	УНИВЕРСАЛЬНЫЙ	ВРЕМЯ
ДАТА	ПРОЦЕНТ	ДЕНЬГИ	

ДД_ММ_ГГ	ДД_МММ_ГГГГ	ДД_МММ МММ_ГГГГ
КВАРТАЛ	КВАРТАЛ_ГГГГ	

ЧЧ_ММ	ЧАС_МИН
-------	---------

Коды графических операций для функции Рисунок()

П	- позиционировать перо
Т	- нарисовать точку
В	- провести ломаную
О	- провести окружность
Д	- провести дугу
Н	- отметка начала замкнутой области
К	- отметка конца замкнутой области
АА	- абсолютное позиционирование
АО	- позиционирование абсолютное по X; относительное по Y
ОА	- позиционирование относительное по X; абсолютное по Y
ОО	- относительное позиционирование

ГРЕСТЬ - рисовать границу закрашиваемой области
 ГРНЕТ - не рисовать границу закрашиваемой области
 ЗГА - база горизонтальной закраски абсолютным адресом
 ЗГО - база горизонтальной закраски относительным адресом
 ЗВА - база вертикальной закраски абсолютным адресом
 ЗВО - база вертикальной закраски относительным адресом
 ЗНЕТ - выключение закраски
 ЗАКР - установка маски закраски
 ДОП - включение дополнительного режима изображения
 ЗАМ - включение заменяющего режима изображения
 СИМВ - вывод теста

Коды фильтров для функций

ОТКРЫТИЕ	- фильтр на функцию Открыть()
ЗАКРЫТИЕ	- фильтр на функцию Закрыть()
ВХОД	- фильтр на функцию Войти()
ВЫХОД	- фильтр на функцию Выйти()
СОЗДАНИЕ	- фильтр на функцию Создать()
УНИЧТОЖЕНИЕ	- фильтр на функцию Уничтожить()
ОЧИСТКА	- фильтр на функцию Очистить()
ПРИСВАИВАНИЕ	- фильтр на функцию Присв()
ВЫБОР	- фильтр на нажатие клавиши {ВВОД}

Коды групп клавиш

ФУНКЦИОНАЛЬНЫЕ	- фильтр на все функциональные клавиши
ЗНАКОВЫЕ	- фильтр на все буквенно-цифровые клавиши
ВСЕ	- фильтр на все клавиши

В. Таблица кодировки символов (альтернативная)

32	64 ө	96 '	128 А	160 а	192 л	224 Р
33 !	65 А	97 а	129 Б	161 б	193 Л	225 С
34 "	66 В	98 б	130 В	162 в	194 Т	226 Т
35 #	67 С	99 с	131 Г	163 г	195 Ъ	227 У
36 \$	68 Д	100 д	132 Д	164 д	196 —	228 Ф
37 %	69 Е	101 е	133 Е	165 е	197 +	229 Х
38 &	70 Ф	102 ф	134 Ж	166 ж	198 Ф	230 Ц
39 '	71 Г	103 г	135 З	167 з	199 Г	231 Ч
40 (72 Н	104 н	136 И	168 и	200 Г	232 Ш
41)	73 И	105 и	137 Й	169 й	201 І	233 Щ
42 *	74 Ј	106 ј	138 К	170 к	202 Ђ	234 Њ
43 +	75 К	107 к	139 Л	171 л	203 Џ	235 Ѝ
44 ,	76 Л	108 л	140 М	172 м	204 Џ	236 Ѐ
45 —	77 М	109 м	141 Н	173 н	205 =	237 Э
46 .	78 Н	110 н	142 О	174 о	206 ±	238 Ю
47 /	79 О	111 о	143 П	175 п	207 ±	239 Я
48 0	80 Р	112 р	144 Р	176	208 ±	240
49 1	81 Q	113 q	145 С	177	209 ±	241
50 2	82 Р	114 г	146 Т	178	210 Г	
51 3	82 С	115 с	147 У	179	211 У	
52 4	84 Т	116 т	148 Ф	180	212 Ф	
53 5	85 У	117 ц	149 Х	181	213 Х	
54 6	86 В	118 в	150 Ц	182	214 Ц	
55 7	87 Ш	119 ш	151 Ч	183	215 Ч	
56 8	88 Х	120 х	152 Щ	184	216 Щ	
57 9	89 Ў	121 ў	153 Щ	185	217 Щ	
58 :	90 З	122 з	154 Ъ	186	218 Ъ	
59 ;	91 [123 {	155 п	187	219 п	
60 <	92 \	124 —	156 Ъ	188	220 Ъ	
61 =	93]	125 }	157 Э	189	221 Э	
62 >	94 ^	126 —	158 Ю	190	222 Ю	
63 ?	95 —	127 д	159 Я	191	223 Я	

Г. Таблица нот

В данной таблице указаны периоды звуковых колебаний, которые можно использовать в обращении к функции Звук().

Октава -3	Октава -2	Октава -1	Октава 0
До 72983	До 36492	До 18243	До 9122
До # 68896	До # 34438	До # 17219	До # 8610
Ре 65029	Ре 32505	Ре 16253	Ре 8127
Ре # 61351	Ре # 30683	Ре # 15342	Ре # 7671
Ми 57926	Ми 28963	Ми 14480	Ми 7240
Фа 54662	Фа 27337	Фа 13667	Фа 6834
Фа # 51612	Фа # 25801	Фа # 12900	Фа # 6450
Соль 48705	Соль 24353	Соль 12176	Соль 6088
Соль # 45966	Соль # 22987	Соль # 11493	Соль # 5747
Ля 43392	Ля 21696	Ля 10848	Ля 5424
Ля # 40950	Ля # 20478	Ля # 10239	Ля # 5120
Си 38655	Си 19328	Си 9664	Си 4832

Октава 1	Октава 2	Октава 3	Октава 4
До 4561	До 2280	До 1140	До 570
До # 4305	До # 2152	До # 1076	До # 538
Ре 4063	Ре 2032	Ре 1016	Ре 508
Ре # 3835	Ре # 1918	Ре # 959	Ре # 479
Ми 3620	Ми 1810	Ми 898	Ми 452
Фа 3417	Фа 1708	Фа 871	Фа 427
Фа # 3225	Фа # 1613	Фа # 806	Фа # 403
Соль 3044	Соль 1522	Соль 761	Соль 380
Соль # 2873	Соль # 1437	Соль # 718	Соль # 359
Ля 2712	Ля 1356	Ля 678	Ля 339
Ля # 2560	Ля # 1280	Ля # 640	Ля # 320
Си 2416	Си 1208	Си 604	Си 302

Д. Словарь терминов

База данных

База данных в интегрированной системе МАСТЕР представляет собой набор файлов, называемых информационными областями, внутри которых располагаются записи различных типов, порядки этих записей. Управление базой данных осуществляется с помощью функций языка Мастер. В диалоговой оболочке МАСТЕРа имеется интерфейс, обеспечивающий возможность диалогового построения схемы базы и дальнейшую работу с ней. В этом интерфейсе реализованы лишь самые универсальные операции и структуры. При желании построить специализированную прикладную информационную систему можно, программируя на языке Мастер, определить произвольные диалоговые интерфейсы и информационные запросы произвольной сложности. Базы данных МАСТЕРа состоят прежде всего из информационных областей - файлов, содержащих записи этой базы. Помимо информационных областей каждая база включает, как правило, рабочую рамку (рамковый файл), представляющую диалоговую среду для общения с базой. В базах данных, созданных через стандартную диалоговую оболочку, имеются еще рамки, описывающие в наглядном виде структурные схемы баз данных.

Библиотека диалоговой оболочки

Основой системы МАСТЕР является ее ядро - совокупность встроенных возможностей: диалоговые процессоры, компилятор и интерпретатор языка Мастер, встроенные функции языка Мастер. На основе этих встроенных возможностей могут быть созданы различные диалоговые интерфейсы, предоставляющие пользователям те или иные команды меню, функциональные клавиши. Диалоговая оболочка представляется совокупностью функций, запрограммированных на языке Мастер и расположенных в составной рамке Биб. Эта рамка должна храниться в файле BIB.MAS. Она считывается автоматически при загрузке МАСТЕРа.

Буфер базы данных

Система управления базами данных в МАСТЕРЕ, как и во многих других СУБД, работает с использованием буферной области, в которую выбираются те записи, к которым чаще всего делаются обращения. Пользователю буфер непосредственно недоступен, но ему желательно знать о его существовании, чтобы понимать, что операции модификации данных отправляют свои результаты сначала в буфер, а уж затем - на диск. Поэтому при сбое питания возможна потеря тех данных, которые остались несохраненными в буфере. В языке Мастер имеется функция Сохранить-Базу(), которая сбрасывает содержимое буфера.

Возвращаемое значение

Возвращаемое значение - это то значение, которое является результатом обращения к функции и может быть использовано для передачи в качестве аргумента в другие функции и операции. Помимо возвращаемого значения каждая функция в языке Мастер обладает еще и побочным действием.

Главный порядок

Множества записей базы данных упорядочиваются несколькими разными способами по возрастанию или убыванию значений полей. Каждое такое упорядочение называется порядком, а один из порядков называется главным. От прочих порядков главный отличается тем, что он не может быть уничтожен сам по себе. При его уничтожении уничтожаются также и все остальные порядки и сами записи. Таким образом, главный порядок является как бы основным представителем данного типа записей.

Глобальное действие клавиши

Значение клавиш в системе МАСТЕР определяется четырьмя понятиями: стандартное, локальное, глобальное действия клавищ и макроопределения клавищ. Глобальным действием клавиши называется действие, запрограммированное в виде функциональной рамки на языке Мастер, имеющей в качестве имени мнемоническое название функциональной клавиши. Глобальное действие клавиши срабатывает только в том случае, если клавиша не имеет стандартного действия и если локального действия нет либо оно вырабатывает разрешающее значение равное единице. Функциональная рамка, задающая глобальное действие, разыскивается в рамковой структуре по обычному правилу: начиная от охватывающей рамки вверх до корневой и затем в библиотеке диалоговой оболочки.

Диапазон

Диапазоном называется прямоугольная область на электронной таблице. В диалоге диапазоны могут указываться с помощью подсветки, для чего используются клавиши-стрелки, а также клавиши зацепки и отцепки {Ф3} и {Ф4}. В языке Мастер для работы с электронными таблицами имеется специальный тип значений, называемый указателем диапазона. Указатели диапазона формируются в программах либо именами ячеек, либо обращением к функции ЯЧ() и могут затем обрабатываться разнообразными встроенными функциями.

Запись

Запись - это основное понятие системы управления базами данных МАСТЕРа. Записи являются элементарными носителями

информации в базе. Записи составляются из полей и упорядочиваются несколькими разными способами.

Импорт

Импортом называется запись информации на диск в универсальном текстовом виде, доступном для считывания другими программными системами и текстовыми редакторами.

Информационная рамка

Рамки в МАСТЕРе могут быть нескольких разных типов: текстовые, табличные, рисунковые, строковые, числовые, составные, рамки-записи. Информационными рамками называются те, которые являются непосредственными носителями информации: текстовые, табличные, рисунковые, строковые и числовые.

Карман

Карманом называется вспомогательный буфер, предназначенный для переноса и временного сохранения текстовой информации. Выборка строкового фрагмента выполняется с помощью функциональной клавиши {УПР-Ф3}. Предварительно фрагмент должен быть выделен с помощью клавиш {Ф3} или {ЗАГ-Ф3}. Для вставки фрагмента из кармана служат клавиши {УПР-Ф4} - вставка строкового фрагмента или {ЗАГ-Ф4} - вставка прямоугольного фрагмента.

Код области

Информационные области базы данных идентифицируются в программах на языке Мастер с помощью числовых кодов, называемых кодами областей. Кодом области может быть целое число от 0 до 10. Код связывается с областью в момент ее создания и не может быть изменен в дальнейшем. При работе с базой данных в ней не могут быть одновременно открытыми две области с одинаковыми кодами, однако после закрытия области с каким-то кодом можно открыть другую область с тем же кодом.

Контекстная рамка

Понятие контекстной рамки служит в языке Мастер для поиска рамок по именам. Контекстной рамкой для некоторой формулы является та рамка, в которой находится данная формула (формула может находиться в этой рамке либо в ячейке электронной таблицы, либо в качестве формулы локального действия клавиши). Имя рамки, встретившееся в формуле, разыскивается сначала в контекстной рамке, затем в той рамке, которая является охватывающей для контекстной, затем в еще более охватывающей и т.д. до корневой рамки, после чего делается еще один поиск в невидимой рамке Биб диалоговой оболочки МАСТЕРа. Для вре-

менного переключения с контекстной рамки на какую-то другую служит встроенная функция **Врамке()**.

Логическое упорядочение рамок

Рамки, располагаясь на рабочей поверхности МАСТЕРа или внутри составной рамки, имеют помимо непосредственно видимого пространственного размещения еще и логическое упорядочение. Логическое упорядочение доступно либо с помощью клавиш {УПР-ВЛЕВО} и {УПР-ВПРАВО}, либо с помощью встроенных функций **ПредРамка()** и **СледРамка()**.

Локальное действие клавиши

Значение клавиш в системе МАСТЕР определяется четырьмя понятиями: стандартное, локальное, глобальное действия клавиши и макроопределения клавиши. Локальным действием клавиши называется действие, запрограммированное в виде формулы на языке Мастер, связанной с клавишей индивидуально в этой рамке. Связывание формул с клавишами может быть сделано либо через меню командой {Ф10} Рамка Действие Клавиша (снаружи рамки, для которой определяется клавиша), либо с помощью функциональной клавиши {ДОП-Ф10} (изнутри этой рамки), либо с помощью встроенной функции **УстФормула()**. Локальное действие имеет приоритет перед стандартным и глобальным действием той же клавиши и срабатывает первым. Возвращаемое значение, вырабатываемое формулой локального действия, определяет, должно ли после локального действия выполняться стандартное или глобальное действие. А именно, если результат равен единице, то стандартное действие (если оно есть) либо глобальное действие выполняется. Любой другой результат блокирует выполнение этих действий.

Локальная переменная

Локальные переменные служат в языке Мастер такими же хранилищами информации, как и рамки. В них можно присваивать числа, строки, тексты, рисунки, указатели множеств записей. Локальные переменные образуются в формуле с помощью функции **Блок()** и существуют временно только во время обращения к этой функции. Есть и существенное отличие локальных переменных от рамок: в них можно присваивать указатели рамок и указатели диапазонов.

Макроклавиша

Макроклавишами называются буквенные клавиши, нажимаемые с модификатором {ДОП}. Первоначально эти клавиши не определены. С любой из них можно связать действие, определяемое последовательностью каких-то других клавиш. Это

определение задается текстом, состоящим, быть может, из нескольких строк, в котором знаковые клавиши обозначаются просто соответствующими знаками, а функциональные клавиши - мнемоническими обозначениями в фигурных скобках.

Множество записей

Множество записей - это основное структурное понятие СУБД МАСТЕРа. Записи объединяются во множества в соответствии со своими типами. Множества являются упорядоченными по значениям полей. Множество может быть либо полным (если оно включает все записи данного типа) и тогда оно называется порядком, либо неполным и тогда оно называется подмножеством. Подмножества образуются либо на порядках - путем их ограничения значениями упорядочивающего поля, либо путем явного включения нужных записей с помощью функций **ПустоМнож()**, **ВключитьЗапись()** и **ИсключитьЗапись()**.

Область базы данных

Информационными областями в МАСТЕРЕ называются файлы специального вида, предназначенные для хранения информации базы данных - записей. Каждая область снабжается при ее создании числовым кодом, по которому она в дальнейшем идентифицируется в программах на языке Мастер.

Охватывающая рамка

Охватывающая рамка - это та рамка, внутри которой в данный момент находится курсор. Для любой рамки определена ее охватывающая рамка - та внутри которой она находится. Для рамок, расположенных на верхнем уровне, охватывающей рамкой является корневая рамка - рабочее поле системы МАСТЕР.

Рамка

Рамка - основное структурное понятие интегрированной системы МАСТЕР. Рамки являются носителями информации. Они же используются и для визуализации информации на экране. Кроме того, с рамками связываются диалоговые процессоры, обеспечивающие диалоговую обработку информации.

Рамка-запись

Рамкой-записью называется такая составная рамка, которой было присвоено значение типа "указатель множества записей". Если в момент присваивания составная рамка была пустой, то в ней автоматически создаются все рамки-поля, требующиеся для этого типа записей. Если же она непуста, то поля не создаются. После такого присваивания рамка-запись может использоваться для доступа к базе данных. В диалоговой оболочке МАС-

ТЕРа для рамок-записей запрограммирован такой интерфейс, который позволяет обращаться к записям не только в форматном, но и в табличном режиме. Поэтому рамки-записи можно называть еще и таблично-форматными рамками.

Рамковый файл

МАСТЕР записывает информацию на диск в одном из трех видов: рамковый файл, текстовый файл и область базы данных. В рамковом файле представляется полная информация о рамке: ее размеры, цвет, окаймление, все формулы, связанные с рамкой, информационное содержимое рамки, включая внутренние рамки, если данная рамка является составной. Представление информации в рамковом файле имеет особый формат, недоступный для считывания другими программными системами (например, текстовыми процессорами).

Стандартное действие клавиши

Значение клавиш в системе МАСТЕР определяется четырьмя понятиями: стандартное, локальное, глобальное действия клавиши и макроопределения клавиши. Стандартным действием клавиши называется то действие, которое запрограммировано для этой клавиши в ядре системы МАСТЕР. Стандартное действие клавиши срабатывает только в том случае, если у клавиши в данной рамке либо нет локального действия, либо оно вырабатывает разрешающее значение равное единице.

Указатель рамки

Указатели рамок служат в языке Мастер для обработки рамковых структур. Формируется указатель рамки с помощью ее имени: считается, что имя, записанное в формуле, обладает значением типа "указатель рамки". Рамка разыскивается, основываясь на понятии контекстной рамки. Получив указатель рамки, к нему можно применить одну из многочисленных рамковых функций. Так, например, по указателю рамки можно получить указатель ее охватывающей рамки, следующей или предыдущей. Указатели требуются для идентификации рамок при присваивании им значений, преобразовании информации в них, при создании и уничтожении, считывании и записи рамок.

Предметный указатель

А

абзац 55, 68
абсолютность имен ячеек 96
автоматизация учрежденческой деятельности 5

Б

база данных 43, 121
инициализация 260
информационная область 129, 234
рабочая рамка 143, 258
рамка схемы 136
схема базы данных 135
библиотека диалоговой оболочки 177
боковик таблицы 94
буфер базы данных 233

В

возвращаемое значение функции или операции 161
вход:
 в МАСТЕР 28
 в рамку 31
выход:
 из МАСТЕРа 50
 из рамки 31

Г

глобальная замена 65
графика:
 деловая 108
 иллюстративная 109

Д

действие клавиши:
 глобальное 158, 216
 локальное 48, 158, 216
 стандартное 216
диалоговая оболочка 17, 177
диапазон 82
 значений графика 102
 источник 88
 получатель 88
 указатель диапазона 96
дисковод 18

З

заготовка графического объекта 112
закрыть рамку 31
заменяющий контекст 64

запись базы данных 122 41, 118, 227

значение:

возвращаемое 161, 171

определенной функции 171

присваивание 167

ячейки 79

И

имя:

рамки 32, 174, 205

простое 175

составное 175

статическое именование 205

структурное именование 207

файла 19

индикатор:

времени 26

клавиатуры 26

местоположения 26

состояния 25

интегрированная система 9

информационная рамка 28

К

калькулятор 49

карман 66

каталог файлов 19, 215

классификатор 152, 282

код области 130, 234

контекст:

замены 64

поиска 59, 64

копирование:

текста 202

формул 96

курсор 30

табличный 84

Л

лексический разбор текста 193

локальная переменная формулы 169

М

макроклавиша 46, 220

макропоследовательность 46, 178, 221

межстрочное расстояние 76

начальное 76

маска заштриховки 116

меню:

базы данных

начальное 136

рабочее 146

схемы 141

главное 33

графическое 113

рамковое 39

табличное 83

текстовое 57

множество записей 227

модификатор клавиши 22

Н

набор 276

нормализация 128

нумерация страниц 77

О

область базы данных 129, 234

обращение:

к операции 161

к функции 161

ограничение множества записей 251

операция 161

обращение к операции 161

открыть рамку 31

отладка программ на Мастере 225

отмена выделения фрагмента 66

относительность имен ячеек 96

П

перебор записей 230, 244, 265

пересчет электронной таблицы 79, 73, 97

печать 77

качество 78

плотность 78

побочное действие функции 162

подмножество 228, 254

поиск

записей 243, 272

значений в таблице 100, 200

контекстный 59, 191

поле:

записи базы данных 122

значения 25

рабочее 25

сообщения 25

строковое

переменной длины 129

фиксированной длины 129

порядок записей в базе 130, 228

главный 133

по слиянию полей 131

по смешению полей 131

по физической вставке 132
по вставке 132
приглашение к приему 20
присваивание значения 167
 кратные 170
 локальным переменным формул 169
 рамкам 168, 208
 ячейкам таблиц 168, 198
прозрачность ячейки таблицы 93

P

разделитель страниц 75
 плавающий 75
 фиксированный 76
разграфка таблицы 94
рамка 27
 информационная 28
 контекстная 175
 охватывающая 30
 рабочая рамка базы данных 143, 258
рамка-запись 235
составная 28
схемы базы данных 136
таблично-форматная 262
табличная 28, 81
текстовая 27, 54
текущая 31
функциональная 171
ранг поля в ключе 131
распахнуть 31
режим:
 вставки 59
 графический 50
 замены 60
 латинской клавиатуры 60
 обучения 47
 русской клавиатуры 60
 символьный 50
реляционное соединение записей 133, 230, 273
рисунок 159

C

сводная таблица схемы базы данных 136
сетевая модель 276
слияние полей 131
смещение полей 131
состояние:
 рисования объекта 112
 свободное 113
способ хранения записей 128
средо-ориентированная технология 14
средства управления базами данных (СУБД) 121
ссылка реляционная 273

страница 55, 74
 нумерация 77
структурирование программ 173

T

таблица определения типа записей 137
текстовый процессор 51
текущий шрифт 62
тип:
 значения 158
 ячеек 84
 рисунковый 159
 строковый 159
 текстовый 159
 указательный
 диапазона 159
 записи 160
 рамки 159, 205
 формульный 159
 числовой 158

U

указание 24
указатель:
 диапазона 96, 159, 197
 записи базы данных 160, 235
 множества записей 235
 рамки 159, 205
 текста 186, 197
 ячейки 197
упаковка полей в записи 234
управляющая структура 162
условные вычисления 164

F

файл 18
 рамковый 42
 текстовый 43
фильтр встроенной функции 219
формат 92
форматка 123
формула 159, 79, 202
фрагмент 55
 строковый 65
 прямоугольный 66
 отмена выделения 66
функция:
 аргумент 172
 встроенная 171
 возвращаемое значение 171
 обращение к функции 161
 определяемая 171

управляющая 163
фильтр встроенной функции 219

Ц
циклическое вычисление 166

Ч
числовой тип 158

Ш
шапка таблицы 94

Э
экспорт 45
электронная таблица 79

Я
язык макропоследовательностей 178
ячейка 79
указатель ячейки 79

Оглавление

Предисловие	3
Глава 1. Введение в интегрированные системы	5
ЧАСТЬ 1. Структура и возможности диалоговой среды	17
Глава 2. Основные понятия и действия	18
2.1. Сведения об операционной системе	18
2.2. Структура интерактивной среды МАСТЕРа	23
2.3. Пример работы над документом	34
2.4. Операции над рамками	39
2.5. Сохранение информации	42
2.6. Прочие общие возможности	45
Глава 3. Диалоговое редактирование текстов	51
3.1. Технология обработки текстов	51
3.2. Общее управление текстовым процессором	56
3.3. Редактирование текста	59
3.4. Форматирование абзацев	68
3.5. Оформление страниц	74
3.6. Операции над текстами в целом	77
Глава 4. Электронные таблицы	79
4.1. Структура и назначение электронных таблиц	79
4.2. Общее управление электронной таблицей	81
4.3. Ввод и редактирование значений ячеек	84
4.4. Структура и оформление таблицы	90
4.5. Использование формул	95
4.6. Более сложные формулы	99
4.7. Графическое отображение табличной информации	102
4.8. Ввод-вывод и распечатка таблиц	106
Глава 5. Диалоговая иллюстративная графика	109
5.1. Базовые понятия иллюстративной графики	109
5.2. Диалоговые возможности в состоянии рисования объекта	113
5.3. Операции над объектами в свободном состоянии	117

Глава 6. Работа с базами данных	121	Заключение	288
6.1. Основные структурные понятия базы данных	122		
6.2. Технологические понятия базы данных	128		
6.3. Определение схемы базы данных	135		
6.4. Рабочая интерактивная среда базы данных	143		
6.5. Операции над записями	147		
6.6. Типовые структуры в базах данных	151		
Часть 2. Инструментальные возможности	154	Приложения	
Глава 7. Основы инструментального языка	155	А. Перечень встроенных функций	289
7.1. Средо-ориентированный характер языка	155	Б. Перечень мнемонических констант	301
7.2. Типы данных	158	В. Таблица кодировки символов	303
7.3. Синтаксис языка	161	Г. Таблица нот	304
7.4. Управляющие структуры	162	Д. Словарь терминов	305
7.5. Присваивание значений	167		
7.6. Определение собственных функций	171		
7.7. Именование рамок в формулах	174	Предметный указатель	311
Глава 8. Основные встроенные функции	180		
8.1. Обработка числовой информации	180		
8.2. Строковые операции и функции	184		
8.3. Обработка текстов	186		
8.4. Обработка графической информации	194		
8.5. Электронные таблицы	197		
8.6. Обработка рамок	204		
8.7. Интерфейс с файловой системой	213		
8.8. Организация диалога и интерактивной среды	215		
Глава 9. Система управления базами данных	227		
9.1. Основные понятия СУБД МАСТЕРа	227		
9.2. Технологические понятия СУБД МАСТЕРа	232		
9.3. Определение типа записей	236		
9.4. Операции над записями и полями	241		
9.5. Поиск и последовательный перебор записей	244		
9.6. Операции над подмножествами	251		
Глава 10. Структурные модели информации в базе данных	257		
10.1. Организация прикладных систем в МАСТЕРЕ	257		
10.2. Организация рабочей рамки базы данных	259		
10.3. Форматно-табличные рамки	261		
10.4. Работа с записями одного типа	265		
10.5. Ссылки от записей к записям	273		
10.6. Типовые связи сетевой модели данных	276		
10.7. Специальные модели данных	282		

Производственное (практическое) издание

ВЕСЕЛОВ Евгений Николаевич

Интегрированная система „Мастер” для ПЭВМ

Зав. редакцией И. Г. Дмитриева

Редактор Л. В. Речицкая

Худож. редактор С. Л. Витте

Техн. редактор Г. А. Полякова

Корректоры Г. А. Башарина, С. Г. Мазурина

Переплет художника Л. Е. Безрученко

ИБ № 2367

Оригинал-макет книги изготовлен автором с помощью текстового
процессора ЛЕКСИКОН
и интегрированной системы МАСТЕР

Подписано в печать 19.12.88. А 11321. Формат 60x88¹/16.

Бум. офсетная. Гарнитура „Литературная”. Печать офсетная.

Усл. печ. л. 19,6. Усл. кр.-отт. 19,6. Уч.-изд. л. 17,89.

Тираж 50 000 экз. Заказ 1063. Цена 1р. 30 к.

Издательство „Финансы и статистика”,
101000, Москва, ул. Чернышевского, 7.

Отпечатано в типографии им. Котлякова издательства
„Финансы и статистика” Государственного комитета СССР
по делам издательств, полиграфии и книжной торговли.

195273, Ленинград, ул. Руставели, 13.