Основные структуры данных

Работа с большими наборами данных автоматизируется проще, когда данные *упорядочены,* то есть образуют заданную структуру. Существует три основных типа структур данных: *линейная, иерархическая* и *табличная.* Их можно рассмотреть на примере обычной книги.

Если разобрать книгу на отдельные листы и перемешать их, книга потеряет свое назначение. Она по-прежнему будет представлять набор данных, но подобрать адекватный метод для получения из нее информации весьма непросто. (Еще хуже дело будет обстоять, если из книги вырезать каждую букву отдельно — в этом случае вряд ли вообще найдется адекватный метод для ее прочтения.)

Если же собрать все листы книги в правильной последовательности, мы получим простейшую структуру данных — *линейную.* Такую книгу уже можно читать, хотя для поиска нужных данных ее придется прочитать подряд, начиная с самого начала, что не всегда удобно.

Для быстрого поиска данных существует *иерархическая структура.* Так, например, книги разбивают на части, разделы, главы, параграфы и т. п. Элементы структуры более низкого уровня входят в элементы структуры более высокого уровня: разделы состоят из глав, главы из параграфов и т. д.

Линейные структуры (списки данных, векторы данных)

Линейные структуры — это хорошо знакомые нам списки. *Список* — это простейшая структура данных, отличающаяся тем, что каждый элемент данных однозначно определяется своим номером в массиве. Проставляя номера на отдельных страницах рассыпанной книги, мы создаем структуру списка. Обычный журнал посещаемости занятий, например, имеет структуру списка, поскольку все студенты группы зарегистрированы в нем под своими *уникальными* номерами. Мы называем номера *уникальными* потому, что в одной группе не могут быть зарегистрированы два студента с одним и тем же номером.

При создании любой структуры данных надо решить два вопроса: как разделять элементы данных между собой и как разыскивать нужные элементы. Разделителем может быть и какой-нибудь специальный символ. Нам хорошо известны разделители между словами — это пробелы. В русском и во многих европейских языках общепринятым разделителем предложений является точка.

Таким образом, *линейные структуры данных (списки) — это упорядоченные структуры, в которых адрес элемента однозначно определяется его номером.*

Табличные структуры (таблицы данных, матрицы данных)

С таблицами данных мы тоже хорошо знакомы, достаточно вспомнить всем известную таблицу умножения. Табличные структуры отличаются от списочных тем, что элементы данных определяются *адресом ячейки,* который состоит не из одного параметра, как в списках, а из нескольких. Для таблицы умножения, например, адрес ячейки, определяется номерами строки и столбца. Нужная ячейка находится на их пересечении, а элемент выбирается из ячейки.

При хранении табличных данных количество разделителей должно быть больше, чем для данных, имеющих структуру списка. Например, когда таблицы печатают в книгах, строки и столбцы разделяют графическими элементами — линиями вертикальной и горизонтальной разметки.

Если нужно сохранить таблицу в виде длинной символьной строки, используют один символ-разделитель между элементами, принадлежащими одной строке, и другой разделитель для отделения строк, например, так:

*Меркурий * 0,39 * 0,056 * 0 # Венера * 0,67 * 0,88 * 0 # Земля * 1,0 * 1,0 * 1*

Еще проще можно действовать, если все элементы таблицы имеют равную длину. Такие таблицы называют *матрицами.* В данном случае разделители не нужны, поскольку все элементы имеют равную длину и количество их известно.

Таким образом, табличные структуры данных *(матрицы)* — это упорядоченные структуры, в которых *адрес элемента определяется номером строки и номером столбца, на пересечении которых находится ячейка, содержащая искомый элемент.*

Многомерные таблицы

Выше мы рассмотрели пример таблицы, имеющей два измерения (строка и столбец), но в жизни нередко приходится иметь дело с таблицами, у которых количество измерений больше. Вот пример таблицы, с помощью которой может быть организован учет учащихся.

Номер факультета: 3

Номер курса (на факультете): 2

Номер специальности (на курсе): 2

Номер группы в потоке одной специальности: 1

Номер учащегося в группе: 19

Размерность такой таблицы равна пяти, и для однозначного отыскания данных об учащемся в подобной структуре надо знать все пять параметров (координат).

Иерархические структуры данных

Нерегулярные данные, которые трудно представить в виде списка или таблицы, часто представляют в виде *иерархических структур*. С подобными структурами мы очень хорошо знакомы по обыденной жизни. Иерархическую структуру имеет система почтовых адресов.

В иерархической структуре адрес каждого элемента определяется путем доступа (маршрутом), ведущим от вершины структуры к данному элементу. Вот, например, как выглядит путь доступа к команде, запускающей программу Калькулятор (стандартная программа компьютеров, работающих в операционной системе Windows):

Пуск □ Программы □ Стандартные □ Калькулятор.

Дихотомия данных

Основным недостатком иерархических структур данных является увеличенный размер пути доступа. Очень часто бывает так, что длина маршрута оказывается больше, чем длина самих данных, к которым он ведет. Поэтому в информатике применяют методы для регуляризации иерархических структур с тем, чтобы сделать путь доступа компактным. Один из методов получил название *дихотомии.*

В иерархической структуре, построенной методом дихотомии, путь доступа к любому элементу можно представить как путь через рациональный лабиринт с поворотами налево (0) или направо (1) и, таким образом, выразить путь доступа в виде компактной двоичной записи.

Упорядочение структур данных

Списочные и табличные структуры являются простыми. Ими легко пользоваться, поскольку адрес каждого элемента задается числом (для списка), двумя числами (для двумерной таблицы) или несколькими числами для многомерной таблицы. Они также легко упорядочиваются. Основным методом упорядочения является *сортировка.* Данные можно сортировать по любому избранному критерию, например: по алфавиту, по возрастанию порядкового номера или по возрастанию какого-либо параметра.

Несмотря на многочисленные удобства, у простых структур данных есть и недостаток — их трудно обновлять. Если, например, перевести студента из одной группы в другую, изменения надо вносить сразу в два журнала посещаемости; при этом в обоих журналах будет нарушена списочная структура. Если переведенного студента вписать в конец списка группы, нарушится упорядочение по алфавиту, а если его вписать в соответствии с алфавитом, то изменятся порядковые номера всех студентов, которые следуют за ним.

Таким образом, *при добавлении произвольного элемента в упорядоченную структуру списка может происходить изменение адресных данных у других элементов.* В журналах, успеваемости это пережить нетрудно, но в системах, выполняющих автоматическую обработку данных, нужны специальные методы для решения этой проблемы.

Иерархические структуры данных по форме сложнее, чем линейные и табличные, но они не создают проблем с обновлением данных. Их легко развивать путем создания новых уровней. Даже если в учебном заведении будет создан новый факультет, это никак не отразится на пути доступа к сведениям об учащихся прочих факультетов.

Недостатком иерархических структур является относительная трудоемкость записи адреса элемента данных и сложность упорядочения. Часто методы упорядочения в таких структурах основывают на предварительной *индексации,* которая заключается в том, что каждому элементу данных присваивается свой уникальный индекс, который можно использовать при поиске, сортировке и т. п. Ранее рассмотренный принцип дихотомии на самом деле является одним из методов индексации данных в иерархических структурах. После такой индексации, данные легко разыскиваются по двоичному коду связанного с ними индекса.

Адресные данные

Если данные хранятся не как попало, а в организованной структуре (причем любой), то каждый элемент данных приобретает новое свойство (параметр), который можно назвать *адресом.* Конечно, работать с упорядоченными данными удобнее, но за это приходится платить их размножением, поскольку адреса элементов данных — это тоже данные, и их тоже надо хранить и обрабатывать.

Файлы и файловая структура

При хранении данных решаются две проблемы; как сохранить данные в наиболее компактном виде и как обеспечить к ним удобный и быстрый доступ (если доступ не обеспечен, то это не хранение). Для обеспечения доступа необходимо, чтобы данные имели упорядоченную структуру, а при этом, как мы уже знаем, образуется «паразитная нагрузка» в виде адресных данных. Без них нельзя получить доступ к нужным элементам данных, входящих в структуру.

Поскольку адресные данные тоже имеют размер и тоже подлежат хранению, хранить данные в виде мелких единиц, таких, как байты, неудобно. Их неудобно хранить и в более крупных единицах (килобайтах, мегабайтах и т. п.), поскольку неполное заполнение одной единицы хранения приводит к неэффективности хранения.

В качестве единицы хранения данных принят объект переменной длины, называемый *файлом. Файл* — это последовательность произвольного числа байтов, обладающая уникальным собственным именем. Обычно в отдельном файле хранят данные, относящиеся к одному типу. В этом случае тип данных определяет *тип файла.*

Проще всего представить себе файл в виде безразмерного канцелярского досье, в которое можно по желанию добавлять содержимое или извлекать его оттуда. Поскольку в определении файла нет ограничений на размер, можно представить себе файл, имеющий 0 байтов *(пустой файл),* и файл, имеющий любое число байтов.

В определении файла особое внимание уделяется имени. Оно фактически несет в себе адресные данные, без которых данные, хранящиеся в файле, не станут информацией из-за отсутствия метода доступа к ним. Кроме функций, связанных с адресацией, имя файла может хранить и сведения о типе данных, заключенных в нем. Для автоматических средств работы с данными это важно, поскольку по имени файла они могут автоматически определить адекватный метод извлечения информации из файла.

Требование уникальности имени файла очевидно — без этого невозможно гарантировать однозначность доступа к данным. В средствах вычислительной техники требование уникальности имени обеспечивается автоматически — создать файл с именем, тождественным уже имеющемуся, не может ни пользователь, ни автоматика.

Каждый файл имеет параметры:

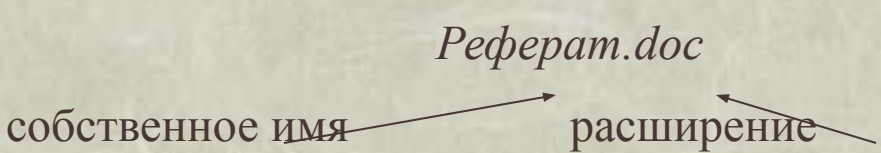
– размер (в байтах);

– дата и время создания;

– атрибуты;

– имя и расширение.

Каждый файл имеет *собственное имя* и *расширение*. Расширение представляет собой набор латинских букв и/или цифр, но его длина не более трех символов. Оно отделяется от основного имени точкой и чаще всего говорит о характере информации, хранимой в данном файле.



Далее приводится список наиболее часто используемых расширений:

.com и .exe – программы, готовые к выполнению;

.bat – командные файлы;

.bas – программы на языке Бейсик;

.pas – программы на языке Паскаль;

.bak – копия файла, делаемая перед его изменением;

.txt и .doc – текстовые файлы;

.pcx, .bmp, .jpg, .tif, .gif – графическое изображение;

.xls – табличные файлы;

.sys – системные файлы;

.avi – видеофайл;

.wav – файл, содержащий музыкальное сопровождение.

Кроме имени и расширения файлы имеют *атрибуты* – это дополнительные параметры, определяющие свойства файлов:

- *Только для чтения* – ограничивает возможности работы с файлом. Его установка означает, что файл не предназначен для внесения изменений.
- *Скрытый* – данный файл не отображается на экране при проведении операций. Это мера защиты против случайного повреждения файла.

- *Системный* – помечаются файлы, обладающие важными функциями. Его отличительная особенность в том, что средствами операционной системы его изменить нельзя. Как правило, большинство файлов, имеющих атрибут Системный, имеют также атрибут Скрытый.
- *Архивный* – для создания резервной копии файла меньшего размера.

Хранение файлов организуется в иерархической структуре, которая в данном случае называется *файловой структурой.* В качестве вершины структуры служит имя носителя, на котором сохраняются файлы. Далее файлы группируются в *каталоги (папки),* внутри которых могут быть созданы *вложенные каталоги (папки, подкаталоги).* Каталог самого верхнего уровня принято называть *корневым*. Каталог, в котором в данный момент времени происходит работа, называется *текущим*.

Путь доступа к файлу начинается с имени устройства и включает все имена каталогов (папок), через которые проходит. В качестве разделителя используйся символ «\» (обратная косая черта).

Уникальность имени файла обеспечивается тем, что *полным именем файла считается собственное имя файла вместе с путем доступа к нему.* Понятно, что в этом случае на одном носителе не может быть двух файлов с тождественными полными именами.

- Пример записи полного имени файла:

<имя носителя>\<имя каталога-1>\...\<имя каталога-N>\ <собственное имя файла>

Вот пример записи двух файлов, имеющих одинаковое собственное имя и размещенных на одном носителе, но отличающихся путем доступа, то есть полным именем. Для наглядности имена каталогов (папок) напечатаны прописными буквами.

C:\АППАРАТЫ\ВЕНЕРА\АТМОСФЕРА\ Результаты исследований

Файловая система содержит и файлы и каталоги.

Функции ФС:

•обеспечение считывания и записи информации на диск

•формирование каталогов

•обеспечение выполнения операций с файлами и каталогами

•группа команд работы с файлами (копирование, удаление, просмотр, переименование)

•доступ ФС к прикладным программам

•обмен информации с внешними устройствами, если они рассматриваются как файлы.

Файловая структура

Одноуровневая

Диск: имя.расширение

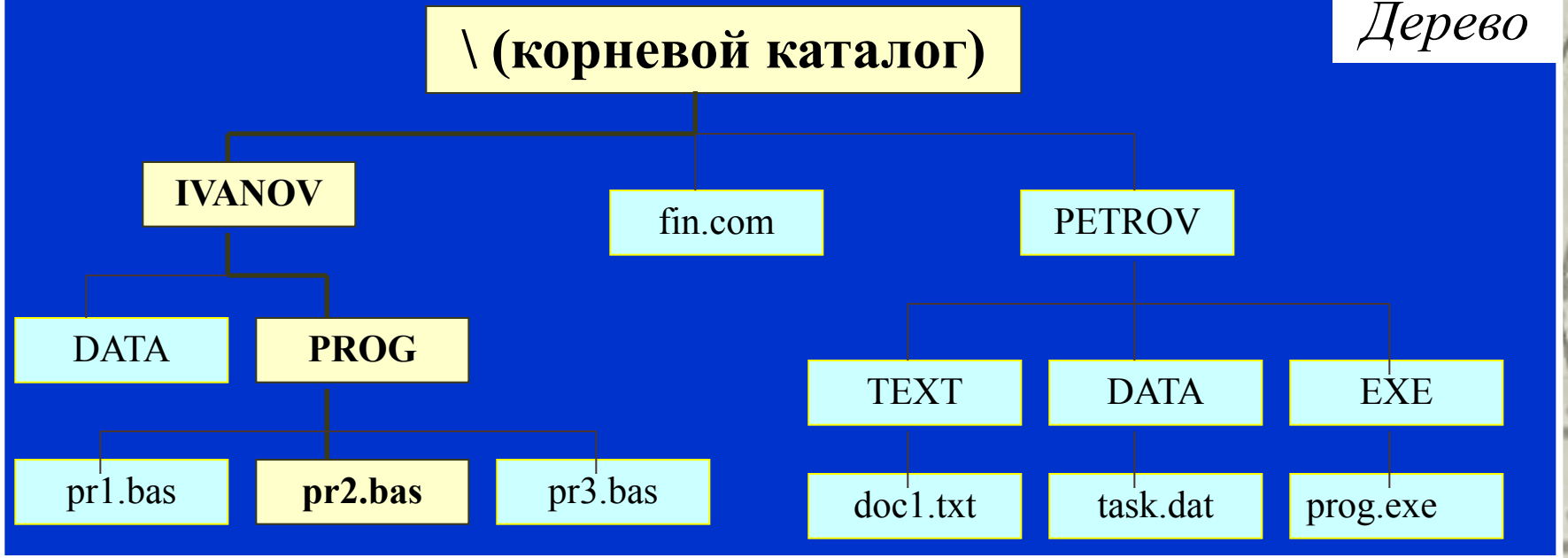
Пример:

A: tetris.exe

Многоуровневая

Диск: (корневой каталог) \ каталог…\ каталог \ имя.расширение

Дерево



Пример полного имени файла: **C:\IVANOV\PROG\pr2.bas**

Диск | Путь | Имя файла