ГЛАВА 2

ИНФОРМАЦИОННЫЕ СТРУКТУРЫ

Нет, не увижу, это ясно,
Поэмы, дерева прекрасней.

— ДЖОЙС КИЛМЕР (JOICE KILMER) (1913) (Перевод Л. Макаровой)

Ах, я с таблицы памяти моей
Все суетные записи сотру.

— *Гамлет* (акт I, сцена 5, строка 98) (Перевод М. Лозинского)

2.1. ВВЕДЕНИЕ

Программы обычно оперируют информацией, которая содержится в таблицах. В большинстве случаев эти таблицы представляют собой не просто бесформенную массу численных значений, а элементы данных с важными *структурными взаимосвязями.*

В простейшем виде таблица может выглядеть так, как простой линейный список элементов, а сведения о ее структурных свойствах можно получить, ответив на следующие вопросы. Какой элемент располагается в списке первым? Какой последним? Какие элементы предшествуют данному элементу, а какие следуют за ним? Сколько элементов содержится в списке? Ответив на такие вопросы, даже в этом простом случае можно получить богатую информацию о структуре данных (см. раздел 2.2).

В более сложных случаях таблица может иметь структуру двумерного массива (т. е. матрицы или решетки, состоящей из строк и столбцов) или даже n-мерного массива для n больше 2, древовидную структуру (tree structure) , которая представляет иерархические или разветвляющиеся связи, либо такую сложную многосвязную структуру (multilinked structure) с большим количеством соединений, которая устроена, как человеческий мозг.

Для правильного использования компьютера необходимо хорошо знать структурные взаимосвязи между данными, основные методы представления структур внутри компьютера, а также методы работы с ними.

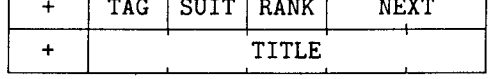
В данной главе приводятся наиболее важные сведения об информационных структурах: статические и динамические свойства структур разных типов, средства выделения памяти для хранения и представления структурированных данных, эффективные алгоритмы для создания, изменения, удаления информации о структуре данных, а также для доступа к ней. В ходе изучения материала на нескольких важных примерах будет проиллюстрировано применение этих методов для решения широкого круга задач. В примерах будут рассмотрены топологическая

сортировка, арифметика многочленов, моделирование дискретных систем, работа с разреженными матрицами, манипулирование алгебраическими формулами, а также создание компиляторов и операционных систем. Основное внимание здесь будет уделено представлению структуры *внутри* компьютера; ее преобразование между внешними и внутренними представлениями будет подробно рассмотрено в главах 9 и 10.

Большая часть представленного здесь материала часто называется обработкой списков, особенно с тех пор как было создано достаточно систем программирования, например LISP , предназначенных специально для работы со структурами общего типа под названием *Списки*. (Далее в этой главе слово "Список" с прописной начальной буквой будет использоваться для обозначения отдельного типа структуры, которая более подобно рассматривается в разделе 2.3.5.) Хотя в большинстве случаев системы обработки Списков очень полезны, они часто накладывают на работу программиста не всегда оправданные ограничения. В собственных программах обычно эффективнее непосредственно использовать предлагаемые в этой главе методы, подгоняя для конкретного приложения формат данных и алгоритмы их обработки. Многие разработчики программного обеспечения все еще считают, что методы обработки Списков очень сложны (и потому вынуждены использовать интерпретаторы сторонних разработчиков или уже готовые наборы процедур) и что обработка Списков может выполняться лишь некоторым строго определенным способом. Как будет показано ниже, ничего сверхъестественного, загадочного или трудного в работе со сложными структурами нет. Приведенные здесь методы являются частью арсенала каждого программиста, и их можно использовать при создании программ на языке ассемблера или на таких алгебраических языках, как FORTRAN, C и Java.

Методы работы с информационными структурами будут проиллюстрированы здесь на примере компьютера MIX. Читателю, которого не интересует подробная структура программ для компьютера MIX, следует изучить хотя бы способы, с помощью которых информация о структуре представляется в памяти компьютера MIX.

Здесь важно определиться с терминами и обозначениями, которые будут довольно часто использоваться. Информация в таблице состоит из набора *узлов* (*nodes*); некоторые авторы называют их записями (records) , объектами (entities) или цепочками (beads) . Иногда вместо термина "узел" будут использоваться термины "предмет" и "элемент". Каждый узел состоит из одного или нескольких последовательных слов в памяти компьютера, которые могут быть разделены на именованные части — *поля*. В простейшем случае узел представляет собой только одно слово и содержит только одно поле, охватывающее это слово целиком. В качестве другого и более интересного примера рассмотрим элементы таблицы, которая предназначена для представления игральных карт. Предположим, что узлы в ней состоят из двух слов, которые делятся на пять полей — TAG, SUIT, RANK, NEXT и TITLE:



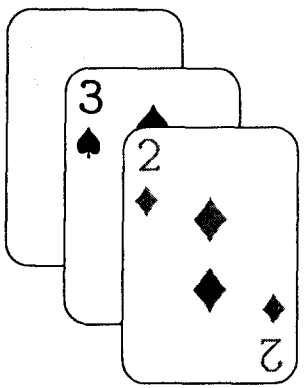
(1)

(Так выглядит формат содержимого двух слов в компьютере MIX. Следует напомнить, что слово в компьютере MIX состоит из пяти байтов и знака (см. раздел 1.3.1).)

В этом примере предполагается, что каждое слово имеет знак "+" (плюс). *Адрес* (*address*) узла, который также называется *связью* (*link*), *указателем* (*pointer*) или *ссылкой* (*reference*) на этот узел, является адресом его первого слова. Адрес часто указывается относительно некоторой базовой ячейки памяти, но в этой главе для простоты предполагается, что адрес является абсолютным.

Любое поле внутри узла может содержать числа, буквы, ссылки или нечто другое, заданное программистом. В приведенном выше примере колода карт для раскладывания пасьянса может быть представлена следующим образом: `TAG` = 1 означает, что карта обращена лицевой стороной вниз, `TAG` = 0 — лицевой стороной вверх; `SUIT` = 1, 2, 3 или 4 означает масть карты, т. е. трефы, бубны, черви или пики соответственно; `RANK` = 1, 2, ..., 13 означает ранг карты, т. е. туз, двойка, ..., король; `NEXT` является *связью* с картой, расположенной *под* данной картой в этой же колоде; `TITLE` представляет предназначенное для печати имя карты из пяти символов. Типичная колода карт может выглядеть так, как показано ниже.

Карты



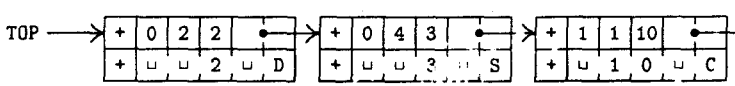
Представление в компьютере

100:	+	1	1	10	Λ
101:	+	␣	1	0	␣ C
386:	+	0	4	3	100
387:	+	␣	␣	3	␣ S
242:	+	0	2	2	386
243:	+	␣	␣	2	␣ D

(2)

Адреса карт в компьютерном представлении показаны здесь в виде чисел 100, 386 и 242, хотя в данном примере они могли бы быть любыми, поскольку каждая карта связана со следующей картой, расположенной под ней. Обратите внимание на особую связь "Λ" в узле 100. *Здесь прописная греческая буква "лямбда" обозначает пустую связь*, т. е. связь с несуществующим узлом. Пустая ссылка Λ используется в узле 100, так как десятка треф является самой нижней картой в колоде. В компьютере ссылка Λ представлена легко распознаваемым значением, которое не может быть адресом узла. Вообще, предположим, что по адресу 0 никаких узлов нет, тогда Λ практически всегда в программах для компьютера `MIX` может быть представлена как связь с нулевым значением адреса.

Введение понятия связи с другими элементами данных является чрезвычайно важной идеей в компьютерном программировании и ключевым способом представления сложных структур. При создании схемы представления узлов в компьютере связи обычно изображаются в виде стрелок. Тогда схема примера (2) будет выглядеть следующим образом:



(3)

При этом фактические адреса 242, 386 и 100 (которые в данном примере все равно не имеют большого значения) в схеме (3) не представлены. Используемое в электротехнике обозначение заземления здесь применяется для обозначения пустой связи в правой части схемы. Обратите внимание также, что в схеме (3) на самую верхнюю карту указывает стрелка TOP. Здесь TOP представляет собой *переменную связи (link variable)*, которая часто называется *указателем* (*pointer variable*), т. е. переменной, значением которой является связь. Все ссылки на узлы в программе определяются непосредственно с помощью переменных связи (или констант связи) либо косвенно с помощью полей связи в других узлах.

Теперь рассмотрим самую важную часть системы обозначений — обозначение ссылки на поле внутри некоторого узла. Оно выглядит достаточно просто, поскольку для этого нужно лишь привести имя данного поля и указать вслед за ним в скобках связь с нужным узлом. Например, для случаев (1)–(3) это можно сделать так, как показано ниже:

$$\begin{aligned} \texttt{RANK(100)} &= 10; \qquad \texttt{SUIT(TOP)} = 2; \\ \texttt{TITLE(TOP)} &= \text{“}\texttt{␣␣2␣D}\text{”}; \qquad \texttt{RANK(NEXT(TOP))} = 3. \end{aligned} \tag{4}$$

Читателю следует внимательно изучить эти примеры, поскольку такие обозначения полей будут применяться во многих других алгоритмах в настоящей и последующих главах. Чтобы пояснить смысл этой идеи, рассмотрим простой алгоритм, предназначенный для размещения с верхней стороны колоды новой карты с лицевой стороной, обращенной вверх. При этом допустим, что значение переменной связи NEWCARD содержит связь с новой картой.

A1. Установить NEXT(NEWCARD) $\leftarrow$ TOP. (Таким образом, значение поля связи в новой карте будет указывать на верхнюю карту колоды.)

A2. Установить TOP $\leftarrow$ NEWCARD. (TOP по-прежнему будет указывать на верхнюю карту колоды.)

A3. Установить TAG(TOP) $\leftarrow$ 0. (Карта обращена лицевой стороной вверх.) ▮

В другом примере рассмотрим алгоритм для вычисления текущего количества карт в колоде.

B1. Установить N $\leftarrow$ 0, X $\leftarrow$ TOP. (Здесь N является целочисленной переменной, а X — переменной связи.)

B2. Если X $= \Lambda$, прекратить выполнение алгоритма; при этом значение N будет равно количеству карт в колоде.

B3. Установить N $\leftarrow$ N + 1, X $\leftarrow$ NEXT(X) и вернуться к шагу B2. ▮

Обратите внимание на то, что в этих алгоритмах символьные имена используются для двух совершенно разных объектов: как имена *переменных* (TOP, NEWCARD, N, X) и как имена *полей* (TAG, NEXT). Их не следует путать. Если F — это имя поля и L $\neq \Lambda$ — связь, то F(L) — это переменная. Но сам по себе символ F не является переменной, поскольку не обладает никаким значением, если только оно не является непустой связью.

При обсуждении подробностей работы компьютера на низком уровне будут использоваться приведенные далее обозначения, которые применяются для преобразования хранимых значений и их адресов.

a) CONTENTS всегда обозначает поле длиной в одно слово в узле длиной в одно слово. Таким образом, CONTENTS(1000) — это значение, хранимое в памяти по адресу 1000, т. е. переменная с таким значением. Если V является переменной связи, то CONTENTS(V) — значение, на которое указывает V (а не само значение переменной связи V).

b) Если V является именем переменной, значение которой хранится в некоторой ячейке памяти, то LOC(V) обозначает адрес этой ячейки. Соответственно, если V является переменной, значение которой хранится в памяти в виде полного слова, то CONTENTS(LOC(V)) = V.

Эти обозначения можно легко преобразовать в код языка ассемблера MIXAL, хотя обозначения MIXAL выглядят несколько иначе. Значения переменных связи помещаются в индексные регистры, и, чтобы сослаться на нужное поле, необходимо использовать средства MIX для доступа к части поля. Например, приведенный выше алгоритм A можно записать следующим образом.

```
NEXT EQU  4:5          Определение полей NEXT
TAG  EQU  1:1              и TAG для ассемблера.
     LD1  NEWCARD      A1. rI1 ← NEWCARD.
     LDA  TOP              rA ← TOP.                (5)
     STA  0,1(NEXT)        NEXT(rI1) ← rA.
     ST1  TOP          A2. TOP ← rI1.
     STZ  0,1(TAG)     A3. TAG(rI1) ← 0.  ∎
```

Простота и эффективность выполнения этих операций на компьютере — вот основные причины использования концепции "связанной памяти".

Иногда приходится иметь дело с простой переменной, которая обозначает целый узел, а потому ее значение представляет не одно поле, а последовательность полей. В таком случае можно записать

$$\texttt{CARD} \leftarrow \texttt{NODE(TOP)}, \tag{6}$$

где NODE — такая же спецификация поля, как и CONTENTS, но относится она к целому узлу, а CARD является переменной, которая имеет структуру наподобие (1). Если узел имеет размер c слов, то обозначение (6) является сокращенной записью для c операций присвоения, выполняемых на низком уровне:

$$\texttt{CONTENTS(LOC(CARD)} + j\texttt{)} \leftarrow \texttt{CONTENTS(TOP} + j\texttt{)}, \quad 0 \le j < c. \tag{7}$$

Между языком ассемблера и обозначениями, используемыми в алгоритмах, есть важное отличие. Так как язык ассемблера близок к внутреннему языку компьютера, то используемые в программах MIXAL символы обозначают адреса, а не значения. Поэтому в левых столбцах кода (5) символ TOP на самом деле обозначает *адрес* в памяти, по которому находится указатель на самую верхнюю карту в колоде, а в выражениях (6) и (7) и комментариях в (5) справа он является *значением* TOP, а именно — адресом узла самой верхней карты. Такое различие между языком ассемблера и языком высокого уровня часто является источником ошибок в работе начинающих программистов, поэтому читателю настоятельно рекомендуется выполнить упр. 7, а также другие упражнения из данного раздела для закрепления навыков работы с введенными здесь обозначениями.

УПРАЖНЕНИЯ

1. [*04*] В ситуации, показанной на схеме (3), каким будет значение выражения (a) SUIT(NEXT(TOP)); (b) NEXT(NEXT(NEXT(TOP)))?

2. [*10*] В приведенном выше разделе сказано, что во многих случаях CONTENTS(LOC(V)) = V. При каких условиях LOC(CONTENTS(V)) = V?

3. [*11*] Предложите алгоритм, обратный алгоритму A: он должен удалить самую верхнюю карту колоды (если колода не пуста) и задать ссылку NEWCARD для адреса этой карты.

4. [*18*] Предложите алгоритм, аналогичный алгоритму A, но новая карта должна располагаться *лицевой стороной вниз* в *нижней части* колоды. (Колода может быть пустой.)

▶ **5.** [*21*] Предложите алгоритм, обратный алгоритму из упр. 4. Допустим, что колода не пуста и самая нижняя карта в ней расположена лицевой стороной вниз. Предложенный вами алгоритм должен удалить эту карту и указать ее адрес в переменной связи NEWCARD. (Данный алгоритм при раскладывании пасьянсов иногда называется мошенничеством.)

6. [*06*] В примере с игральными картами предположим, что CARD — это имя переменной, значением которой является весь узел, как показано в (6). Операция CARD ← NODE(TOP) устанавливает поля CARD равными соответствующим полям самой верхней карты колоды. Какое из перечисленных ниже обозначений будет соответствовать масти самой верхней карты после выполнения этой операции:

(a) SUIT(CARD); (b) SUIT(LOC(CARD)); (c) SUIT(CONTENTS(CARD)); (d) SUIT(TOP)?

▶ **7.** [*04*] В приведенном выше примере программы (5) для компьютера MIX переменная связи с верхней картой хранится в слове компьютера MIX, которое на языке ассемблера называется TOP. Какой из приведенных вариантов кода для заданной структуры поля (1) позволит занести значение NEXT(TOP) в регистр A? Объясните, почему другой вариант неверен.

a)
```
LDA  TOP(NEXT)
```

b)
```
LD1  TOP
LDA  0,1(NEXT)
```

▶ **8.** [*18*] Напишите программу для компьютера MIX, соответствующую алгоритму B.

9. [*23*] Напишите программу для компьютера MIX, которая печатает названия карт из данной колоды, начиная с самой верхней карты и размещая их по одной в каждой строке со скобками вокруг карт, повернутых лицевой стороной вниз.