Глава 10
Составные типы данных

Десятая глава освещает широкий круг составных типов данных: массивы, строки символов, записи, множества, кортежи и списки. Массивы рассматриваются самым пристальным образом. В частности, обсуждаются следующие вопросы: разновидности массивов (с позиций размещения в памяти и времени жизни), их инициализация, атрибуты и операции простых массивов и их развитых (например, в скриптовых языках) версий. Многомерные массивы подразделяются на прямоугольные и массивы массивов, поясняется специфика доступа к сечениям массивов, а также доступ по содержанию к ассоциативным массивам. Помимо обычных записей описываются объединения и вариантные записи, анализируется их надежность.

Массивы

Массив является наиболее популярной и востребованной структурой данных.

Массив — это структура данных, которая содержит последовательность элементов одинакового типа. Фундаментальное свойство массива — время доступа к любому его элементу A[i] не зависит от значения *индекса* i.

Индекс первого элемента называют *нижней границей*, а индекс последнего элемента — *верхней границей* массива.

Тип массива в языке Pascal записывается в виде:

```
type
array [<границы>] of  <тип_элемента>
```

где `[<границы>]` определяют границы массива, а `<тип_элемента>` задает тип элемента массива.

В языке Pascal границы массива задаются перечислением или поддиапазоном:

```
array [1999..2012] of  real -- индексы записаны в виде поддиапазона
array [(пон, втр, срд, чтв, птн)] of  integer -- индексы записаны в виде
                                                 перечисления
array [(char)] of  лексема -- индексы записаны в виде перечисления с именем
```

В большинстве языков программирования границы массива, а значит и индексы, записываются в квадратных скобках. Из современных языков исключением стал лишь язык Ada — в нем используются круглые скобки и объявления типов массива имеют, например, следующий вид:

```
type
array (Day) of Integer; -- индексы записаны в виде перечисления с именем Day
array (Integer range 1999..2012) of  Float; -- индексы записаны в виде диапазона
                                              целых чисел
```

Различают *компоновку* массива (layout) и *размещение* массива в памяти (allocation). При компоновке задаются относительные адреса элементов массива (относительно адреса 1-го элемента), вычисляются формулы для адресации элементов. При размещении определяются действительные машинные адреса элементов и выделяется память.

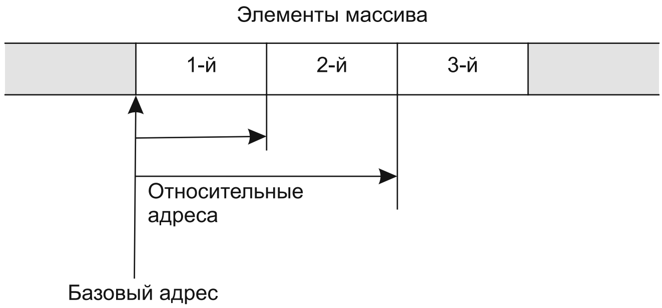

Рис. 10.1. Адресация элементов массива

Как показано на рис. 10.1, возможность использования относительных адресов обусловлена спецификой организации и размещения элементов массива: элементы (ячейки) массива имеют одинаковый размер, а в памяти размещаются плотно, примыкая друг к другу без всяких пропусков.

Объявление массива:

```
var M: array [low .. high] of T;
```

определяет размещение элементов массива в последовательности ячеек памяти. Эти ячейки имеют смежные адреса.

Если `base` — адрес начала массива, `w` — длина элемента массива, то адрес i-го элемента массива равен:

```
A [i] = (base – low × w) + i × w
```

где `(base – low × w)` вычисляется предварительно, а `(i × w)` вычисляется во время выполнения программы (поскольку i изменяется).

В языке С первый элемент массива имеет индекс `0`, поэтому адрес i-го элемента массива вычисляется по упрощенной формуле:

```
A [i] = base + i × w
```

Легко заметить, что количество команд машинной программы, необходимых для вычисления адреса, не зависит от значения i. Поэтому время доступа к элементу массива — константа.

В конструировании типа «массив» задействуются два вспомогательных типа: один из них определяет тип элемента, а другой — тип индексов. В языках Pascal и Ada для создания индексов применяют многие элементарные типы (целый, логический, символьный и перечисления). Во всех остальных языках разрешены лишь поддиапазоны целых чисел.

При обращении к массиву желательно проверять значение текущего индекса. Значение должно находиться в пределах допустимого диапазона. Такая проверка способствует повышению надежности вычислений. Однако реализована она только в языках Pascal, Ada, Java и C#. В других языках (например, в C и C++) проверка отсутствует в силу специфики организации массивов. Эту специфику мы будем обсуждать в следующей главе.

Несколько необычно организована индексация в языке Perl. Здесь имена всех массивов должны начинаться с символа `@`, но поскольку элементы массива являются скалярными величинами, их имена предваряются знаком доллара `$`. Из-за этого в массиве `@list` , например, ко второму элементу обращаются по имени `$list[1]`.

На элемент массива в Perl можно сослаться при помощи отрицательного индекса. Отрицательное значение индекса обозначает номер позиции элемента с конца. Индекс –1 соответствует последнему элементу массива, –2 — предпоследнему и т. д. Для получения индекса, отсчитываемого от начала, нужно отрицательный индекс сложить с размером массива. Например, если массив `@list` состоит из семи элементов с индексами 0..6, то индексированное имя `$list[-2]` адресует предпоследний элемент с индексом `5`. Ссылка на несуществующий элемент массива в Perl возвращает значение `undef`, но сообщение об ошибке не формируется.

Разновидности массивов

Связывание типа индекса массива с переменной массива обычно выполняется статически, но по диапазону индексов связывание происходит иногда динамически.

В некоторых языках нижняя граница диапазона значений индексов задается по умолчанию. Например, в C-подобных языках нижняя граница всех диапазонов индексов равна нулю. В языке Fortran 95 (и выше) она зафиксирована как единица, но может принимать любое целое значение. В большинстве других языков диапазоны индексов полностью определяются программистом.

Классифицируем массивы по трем признакам:

- связывание по диапазонам индексов;
- связывание с памятью;
- категория памяти для размещения.

Всего возможны пять разновидностей массивов. В первых четырех разновидностях связывание по диапазону индексов и размещению в памяти сохраняется в течение всего времени жизни переменной. Имеется в виду, что при фиксации диапазонов индексов массив уже не может изменить размер.

Статические массивы

Статическим называют массив со статическим связыванием по диапазонам индексов и таким размещением в памяти, которое происходит до начала работы программы. Статические массивы достаточно эффективны, поскольку отсутствуют затраты времени как на динамическое размещение в памяти, так и на удаление из нее. Недостатком является то, что память ими занимается на все время выполнения программы.

Явные стековые массивы

Явным стековым называется массив со статическим связыванием по диапазонам индексов и таким размещением в памяти, которое происходит по итогам обработки объявления в ходе выполнения программы (подпрограммы). Массив удаляется из стековой памяти по завершении работы программы (подпрограммы). В сравнении со статическими явные стековые массивы повышают эффективность использования памяти: большой массив одной подпрограммы может занимать ту же область памяти, что и большой массив другой подпрограммы, причем так может продолжаться до тех пор, пока обе подпрограммы не окажутся активными одновременно. К недостатку следует отнести дополнительные затраты времени на размещение и удаление массива из памяти.

Стековые массивы

Стековым называется массив с динамическим связыванием по диапазонам индексов и динамическим размещением в памяти, которое происходит в ходе обработки объявления. Связанность диапазонов индексов и размещение массива в памяти сохраняются в течение всей жизни переменной. Главным преимуществом этой разновидности массивов по сравнению с двумя предыдущими считается гибкость — не нужно заранее знать (до момента использования) размер массива.

Явные динамические массивы

Явный динамический массив подобен явному стековому массиву в том, что связывание и по диапазонам индексов, и по памяти происходит после размещения в памяти. Связывание по индексам и памяти производится по запросу программы и в течение ее выполнения, но память выделяется в куче, а не в стеке. Преимуществом этой разновидности массивов является гибкость: изменение размера массива всегда остается проблемой. Недостаток — на размещение в куче тратится времени больше, чем на размещение в стеке.

Динамические массивы

Динамическим называется массив с таким динамическим связыванием по диапазонам индексов и размещению в памяти, которое повторяется многократно в течение всего жизненного цикла массива. Преимущество: максимальная гибкость. Массив по мере необходимости может расти или сжиматься прямо в ходе выполнения программы. Недостаток — на многократное размещение (удаление) в куче тратится много времени (и все оно приходится на период вычислений).

В языках C и C++ статическими являются массивы, которые объявляются в функциях со спецификатором `static`.

Массивы, объявленные в функциях C и C ++ без спецификатора `static`, считаются явными стековыми массивами.

Как показано ниже, массивы языка Ada могут быть стековыми:

```
get(Size);
declare
    Vector : array  (1 .. Size) of  Integer;
begin
…
end;
```

В этом фрагменте с помощью процедуры `get` вводится размер `Size` для массива `Vector`, который затем динамически размещается в памяти при переходе к блоку `declare`. Когда выполнение доходит до закрывающей скобки блока, массив `Vector` из памяти удаляется.

Языки С и C++ также обеспечивают явные динамические массивы. В этом случае используются стандартные библиотечные функции `malloc` и `free`, которые являются операциями для размещения в куче и удаления из нее соответственно. В языке C++ для управления кучей применяют операторы `new` и `delete`. Имя массива здесь рассматривается как указатель на набор ячеек памяти; указатель может индексироваться.

В языке Java все не родовые массивы считаются явными динамическими. После создания эти массивы сохраняют связывание по диапазонам индексов и памяти. Язык C# также обеспечивает данную разновидность массивов.

Кроме того, язык C# обеспечивает родовые динамические массивы, которые являются объектами класса `List`. Первоначально эти объекты-массивы считаются пустыми и создаются оператором:

```
List<String> stringList = new List<String>();
```

Элементы к объекту добавляются методом `Add`:

```
stringList.Add("Liza");
```

Доступ к элементам этих массивов организуется через индексирование.

Язык Java содержит родовой класс `ArrayList`, подобный классу `List` из языка C#. Разница лишь в том, что здесь индексирование не применяется — для доступа к элементам должны использоваться методы `get` и `set`.

В языке Perl массив может быть увеличен с использованием `push` (добавить один и более элементов в конец массива) и `unshift` (добавить один и более элементов в начало массива) или указанием элемента массива с использованием индекса, значение которого больше последнего индекса массива. Массив может быть сокращен до пустого за счет присвоения ему пустого списка, задаваемого символами `( )`. Длина массива вычисляется сложением значения последнего индекса с единицей.

Подобно Perl, язык JavaScript обеспечивает рост массивов с помощью методов `push` и `unshift`, а также сокращение (установкой массива в состояние пустого списка). Отрицательные индексы здесь не поддерживаются.

Массивы в JavaScript могут быть *разреженными*, в них значения индексов не образуют непрерывную последовательность. Рассмотрим массив по имени `list` из 12 элементов и с диапазоном индексов `0..11`. Выполним следующий оператор присваивания:

```
list[70] = 47;
```

Теперь массив `list` имеет 13 элементов и длину `71`. Элементы с индексами `12..69` не определены, для их хранения память не нужна. Ссылка на несуществующий элемент в массивах JavaScript считается неопределенной.

Массивы в языках Python, Ruby и Lua увеличивают или с помощью методов для добавления элементов, или путем соединения с другими массивами. Языки Ruby и Lua поддерживают отрицательные индексы, а Python — нет. В языках Python, Ruby, и Lua элемент или сечение массива можно удалить. Ссылка на несуществующий элемент в языке Python приводит к ошибке периода выполнения, а в языках Ruby и Lua считается неопределенной (донесение об ошибке не формируется).

В функциональном языке ML массивы не определены, но широко используются в такой реализации языка, как SML/NJ. Массивы функционального языка F# подобны массивам в C#. Для обработки массивов в язык F# включен оператор `foreach`.

Инициализация массива

Некоторые языки предусматривают средства для инициализации массивов во время их размещения в памяти.

В языке Fortran 95 (и выше) массив можно инициализировать, определив в его объявлении агрегат массива. Для одномерного массива агрегат является списком литералов, ограниченных круглыми скобками и слешем (прямой косой чертой). Например, можно написать

```
Integer, Dimension (3) :: List = (/0, 4, 7/)
```

Языки C, C++, Java и C# также обеспечивают инициализацию массивов, но с одной новой особенностью: в языке C объявление

```
int list [] = {3, 4, 8, 79};
```

заставляет компилятор самостоятельно установить длину массива. Это удобно, но вносит ограничения: такая инициализация лишает систему возможности обнаруживать ошибки выхода индексов за пределы корректного диапазона. Ведь диапазон индексов явно не объявляется!

Символьные строки в языках C и C++ реализованы как массивы из символов типа `char`. Эти массивы могут быть инициализированы строковыми литералами:

```
char surname [] = "orlov";
```

Массив `surname` будет содержать шесть элементов, так как все строки завершаются символом `'\0'`, неявно добавляемым системой к строковым константам.

Массивы строк в языках C и C++ также могут инициализироваться наборами строковых литералов. В этом случае речь идет о массиве указателей на символы. Например:

```
char *towns [] = {"Riga", "Petersburg", "Kiev"};
```

Этот пример высвечивает суть символьных литералов в языках C и C++. Для инициализации символьного массива `surname` мы применили строковый литерал, являющийся массивом типа `char`. Однако в массиве `towns` литералы представляются указателями на их символы, поэтому и сам массив является массивом указателей на символы. Например, `towns[0]` — это указатель на букву `'R'` в массиве литеральных символов, содержащем символы `'R'`, `'i'`, `'g'`, `'a'` и символ нуля `'\0'`.

В языке Pascal инициализация массивов (в разделе объявлений программы) не предусмотрена.

Язык Ada предлагает две разновидности агрегатов для инициализации массивов в операторе объявления:

- *позиционный агрегат*, в котором значения элементов перечисляются в порядке их появления в массиве;

- *именной агрегат*, где с помощью операции-стрелки `=>` показываются непосредственные присваивания значений соответствующим индексным позициям массива.

Приведем примеры:

```
Buffer: array (1..5) of Integer := (1, 5, 6, 8, 9);
Container : array (1..5) of Integer := (1 => 21, 4 => 55, others => 0);
```

В первом объявлении все элементы массива `Buffer` инициализируются значениями, перечисленными в порядке следования элементов в массиве. Во втором объявлении первый и четвертый элементы массива `Container` инициализируются с помощью прямых присваиваний, а имя `others` обозначает все остальные элементы, которые инициализируются нулевыми значениями.

Атрибуты и операции простого массива

Простыми будем называть одномерные массивы фиксированного размера, создаваемые с помощью императивных языков (C, Pascal) или императивных средств таких языков, как C++ и Ada.

Такие массивы однородны, то есть размер и структура каждого их элемента одинаковы. Фиксированная размерность массива означает неизменность количества элементов и их местоположения в течение всего периода его жизни. В объект данных типа массив включается *дескриптор*, описывающий некоторые или все атрибуты массива. Верхняя и нижняя границы диапазона индексов (которые не требуются для доступа к элементам) сохраняются для того, чтобы обеспечить проверку соответствия индекса объявленному диапазону. Другие атрибуты обычно в дескрипторе не хранятся (во время вычислений); они нужны только во время компиляции для контроля типов и определения способа хранения массива в памяти.

В состав атрибутов для объекта данных типа «простой массив» входят:

- *Количество элементов* — указывается косвенно, путем задания диапазона изменения индексов.
- *Тип данных для каждого элемента* — в данном случае он одинаков для всех элементов.
- *Список значений индексов*, применяемых для выбора элементов, — задается в виде набора целых чисел, первое из которых соответствует первому элементу, второе — второму элементу и т. д. Он может представляться в виде диапазона значений, например `[-7..25]`, или определяться только верхней границей диапазона, если нижняя задается по умолчанию, например `[100]`.

Базовой считают операцию выбора элемента массива. Она называется *индексацией* и обозначается в виде имени массива с присоединенным индексом искомого элемента: `B[2]` или `ExamsMark[student]`. В общем случае индекс может записываться как вычисляемое выражение: `ExamsMark[`i` + 2]`. В левой части оператора присваивания операция индексации возвращает l-значение искомого элемента (его адрес), а в правой части — r-значение искомого элемента (его величину).

Индексация позволяет применять к элементам массивов все операции, которые разрешены для типов этих элементов.

В языках Pascal и Ada возможны присваивания массивов целиком. Дополнительно в языке Ada разрешены операции сравнения и конкатенации простых массивов, а также логические операции над простыми булевыми массивами. При конкатенации (составлении) длина результирующего массива равна сумме длин массивов-операндов.

В языке Fortran 95 (и выше) предусмотрены разнообразные операции над массивами, предназначенные для *поэлементной* обработки пар массивов. К ним относятся присваивания, операции отношений, арифметические и логические операции. При выполнении, например, сложения двух массивов одинаковой размерности попарно складываются значения их элементов.

В С-подобных языках отсутствуют операции, при выполнении которых массив считается единым целым. Исключением являются объектно-ориентированные методы языков Java, C++ и C#.

Операции над массивами в скриптовых языках

В языке Python массивы называются списками, хотя они имеют все характеристики динамических массивов. Поскольку объекты могут иметь любые типы, массивы здесь неоднородны. Python обеспечивает присваивание массива, хотя оно заключается лишь в изменении ссылок. Python также имеет операции для конкатенации массивов (`+`) и определения вхождения (членства) элемента (`in`). Он содержит две различные операции сравнения: одна определяет, ссылаются ли две переменные на один и тот же объект (`is`); другая проверяет равенство всех объектов, входящих в указанные объекты, независимо от глубины вложенности (`= =`).

Подобно языку Python, элементы в массивах Ruby являются ссылками на объекты. И здесь операция `= =` над двумя массивами возвращает `true`, если массивы имеют одинаковую длину, а соответствующие элементы эквивалентны. Массивы в Ruby могут подвергаться конкатенации с помощью метода класса `Array`.

Язык Perl обеспечивает присваивания массивов, но не поддерживает сравнения.

Прямоугольные массивы и массивы массивов

Прямоугольный массив — это многомерный массив, в котором все строки и столбцы имеют одинаковое количество элементов. Прямоугольные массивы точно моделируют прямоугольные таблицы.

Объявление двухмерного прямоугольного массива

```
var M: array [1 .. 3, 1 .. 2] of integer;
```

можно рассматривать как три двухэлементных подмассива `M[1]`, `M[2]`, `M[3]`, которые являются строками в следующей матрице элементов:

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \\ M_{31} & M_{32} \end{bmatrix}.$$

Возможны два варианта размещения двухмерного массива в памяти: по строкам и по столбцам.

Размещение массива *по строкам* проиллюстрировано на рис. 10.2.

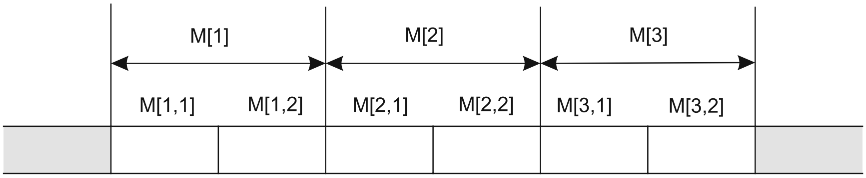

Рис. 10.2. Размещение двухмерного массива по строкам

В этом случае быстрее всего изменяется последний индекс j каждого элемента `M`[i, j].

Адрес элемента можно определить по формуле:

$$A\ [i_1]\ [i_2] = (base - low_1 \times w_1 - low_2 \times w_2) + i_1 \times w_1 + i_2 \times w_2$$

где w_1 — длина строки $M\ [i_1]$,
w_2 — длина элемента $M\ [i_1]\ [i_2]$,
low_1 — нижняя граница строки,
low_2 — нижняя граница элемента строки.

Очевидно, что

$$w_1 = n_2 \times w_2$$

где n_2 — количество элементов в строке,

$$n_2 = high_2 - low_2 + 1$$

При размещении массива *по столбцам* быстрее всего изменяется первый индекс i каждого элемента `M`[i, j]:

```
M[1,1],   M[2,1],   M[3,1],   M[1,2],   M[2,2],   M[3,2]
```

Прямоугольные массивы можно создавать в языках Fortran, Pascal, Ada, F# и C#.

В этих случаях все индексные выражения в ссылках на элемент помещаются в единую пару скобок (круглых — в языках Fortran и Ada, квадратных — во всех остальных языках). Например, на языке C# можно записать:

```
myArray[3, 7]
```

Массив массивов — это многомерный массив, в котором не требуется, чтобы длины у строк были одинаковыми. Массивы массивов иногда называют *не выровненными* массивами. Например, не выровненная матрица может состоять из трех строк, одна из которых содержит пять элементов, другая — семь элементов, а третья — двенадцать элементов. Все эти соображения применимы и к массивам более высоких размерностей. Так, для трехмерного массива в третьем измерении (измерении уровней) каждый уровень может иметь различное количество элементов. Не выровненные массивы становятся возможны, когда многомерные массивы, по сути, являются массивами, состоящими из массивов. Например, матрица строится как массив из одномерных массивов.

Рассмотрим пример построения не выровненного массива средствами языка C#, в котором элементы подобного массива имеют ссылочный тип и инициализируются значением `null`.

Объявим одномерный массив из трех элементов, каждый из которых является одномерным массивом целых чисел:

```
int[][] jaggedArray = new int[3][];
```

Перед использованием `jaggedArray` его элементы нужно инициализировать. Сделать это можно следующим образом:

```
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[4];
jaggedArray[2] = new int[2];
```

Каждый элемент представляет собой одномерный массив целых чисел. Первый элемент массива состоит из пяти целых чисел, второй — из четырех и третий — из двух.

Для заполнения элементов массива значениями можно выполнить инициализацию, при этом размер массива знать не требуется:

```
jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };
jaggedArray[1] = new int[] { 0, 2, 4, 6 };
jaggedArray[2] = new int[] { 11, 22 };
```

Также массив можно инициализировать в объявлениях:

```
    int[][] jaggedArray2 = new int[][]
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

В языке C# можно использовать и сокращенную форму. Следует помнить, что при инициализации элементов оператор `new` опускать нельзя, так как инициализация по умолчанию не предусмотрена:

```
int[][] jaggedArray3 =
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

Напомним, что не выровненный массив является массивом массивов, поэтому его элементы имеют ссылочные типы и инициализируются значением `null`.

Доступ к отдельным элементам массива организуется следующим образом:

```
// Присвоить 77 второму элементу ([1]) первого массива ([0]):
jaggedArray3[0][1] = 77;
// Присвоить 88 второму элементу ([1]) третьего массива([2]):
jaggedArray3[2][1] = 88;
```

Массивы массивов можно смешивать с прямоугольными массивами. Покажем объявление и инициализацию одномерного массива массивов, состоящего из трех двухмерных элементов различного размера:

```
int[][,] jaggedArray4 = new int[3][,]
{
    new int[,] { {1,3}, {5,7} },
    new int[,] { {0,2}, {4,6}, {8,10} },
    new int[,] { {11,22}, {99,88}, {0,9} }
};
```