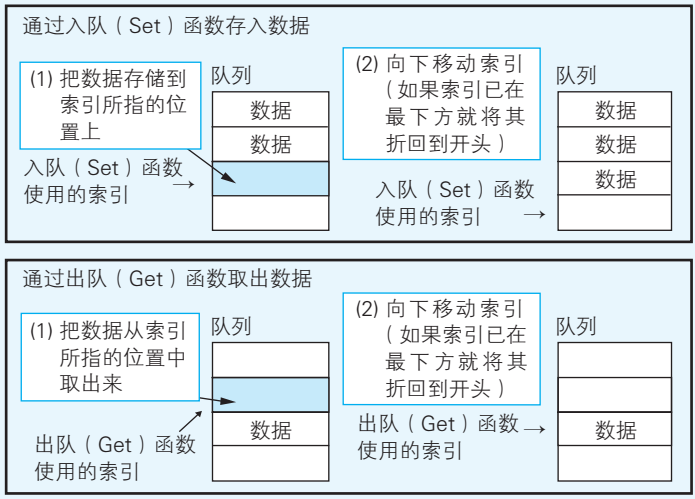

图 6.9　数组变成了“数据之环”

6.6　要点 6：了解结构体的组成

要想理解用 C 语言程序实现链表和二叉树的方法，就必须先了解何谓“结构体”。所谓结构体，就是把若干个数据项汇集到一处并赋予其名字后所形成的一个整体。例如，可以把学生的语文、数学、英语的考试成绩汇集起来，做成一个叫作 TestResult 的结构体。

在代码清单 6.8 中，定义了一个叫作 TestResult 的结构体。C 语言中结构体的定义方法是：先在 struct 这个关键词后面接上结构体的名字（也被称作是结构体的标签），然后在名字后面接上用“{”和“}”括起来的程序块，并在程序块中列出若干个数据项。

代码清单 6.8 结构体汇集了若干个数据项

```
struct TestResult{
    char Chinese;    /* 语文成绩 */
    char Math;       /* 数学成绩 */
    char English;    /* 英语成绩 */
};
```

一旦定义完结构体，就可以把结构体当作是一种数据类型，用它来定义变量。如果把结构体 TestResult 用作数据类型并定义出了一个名为 xiaoming 的变量（代表小明的成绩），那么在内存上就相应地分配出了一块空间，这块空间由用于存储 Chinese、Math、English 这三个成员（Member）数据所需的空间汇集而来。被汇集到结构体中的每个数据项都被称作“结构体的成员”。在为结构体的成员赋值或是读取成员的值时，可以使用形如 xiaoming.Chinese（表示小明的语文成绩）的表达式，即以“.”分割变量和结构体的成员（如代码清单 6.9 所示）。

代码清单 6.9 结构体的使用方法

```
struct TestResult xiaoming;  /* 把结构体作为数据类型定义变量 */
xiaoming.Chinese = 80;       /* 为成员数据 Chinese 赋值 */
xiaoming.Math = 90;          /* 为成员数据 Math 赋值 */
xiaoming.English = 100;      /* 为成员数据 English 赋值 */
```

如果要编写一个用于处理 100 名学生考试成绩的程序，就需要定义一个以 TestResult 为数据类型、包含 100 个元素的数组。通过定义，在内存上就分配出了一块空间，能够存储 100 个数据的集合，每个数据的集合中都含有 Chinese、Math、English 三个数据项（如图 6.10 所示）。接下来只要巧妙地运用结构体的数组就可以实现链表和二叉树了。

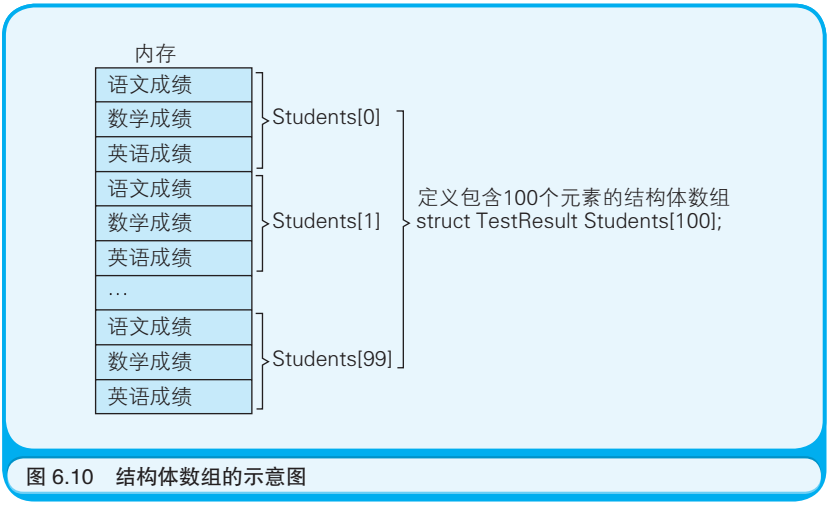

图 6.10　结构体数组的示意图

6.7　要点 7：了解链表和二叉树的实现方法

下面讲解如何使用结构体的数组实现链表。链表是一种类似数组的数据结构，这个“数组”中的每个元素和另一个元素都好像是手拉着手一样。在现有的以结构体 TestResult 为数据类型的数组 Student[100] 中，为了让各个元素“把手拉起来”，就需要在结构体中再添加一个成员（如代码清单 6.10 所示）。

代码清单 6.10　带有指向其他元素指针的自我引用结构体

```
struct TestResult{
    char Chinese;            /* 语文成绩 */
    char Math;               /* 数学成绩 */
    char English;            /* 英语成绩 */
    struct TestResult* Ptr;  /* 指向其他元素的指针 */
};
```

请诸位注意，这里在结构体 TestResult 中添加了这样一个元素。

```
struct TestResult* Ptr
```

虽然本节不会详细地分析这条语句，但是简单地说，这里的成员 Ptr 存储了数组中另一个元素的地址。在 C 语言中，把存储着地址的变量称为“指针”。这里的“*”（星号）就是指针的标志。诸位可以看到，Ptr 就是以结构体 TestResult 的指针（struct TestResult*）为数据类型的成员。这种特殊的结构体可以称为“自我引用的结构体”。之所以叫这个名字，是因为在结构体 TestResult 的成员中，含有以 TestResult 的指针为数据类型的成员，这就相当于 TestResult 引用了与自身相同的数据类型。

在结构体 TestResult（已变成了自我引用的结构体）的数组中，每个元素都含有一个学生的语文、数学、英语成绩以及成员 Ptr。Ptr 中存储着本元素接下来该与哪一个元素相连的信息，即下一个元素的地址。在链表的初始状态中，会按照元素在内存上的分布情况设定成员 Ptr 的值（如图 6.11 所示）。

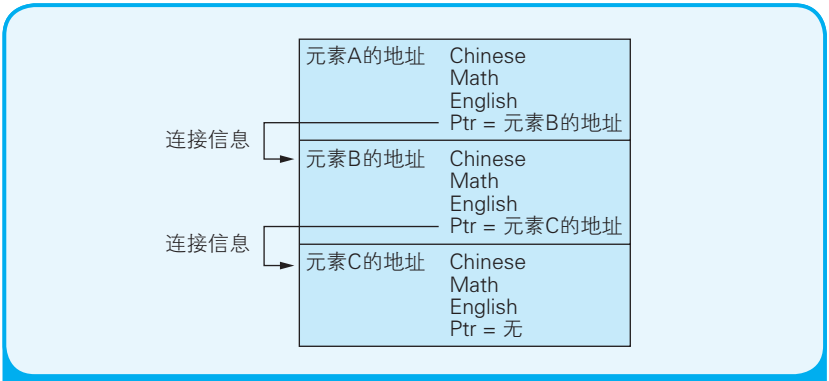

图 6.11 初始状态的链表中，元素的排列顺序与元素在内存上的物理排列顺序相同

那么，接下来就是链表的有趣之处了。因为 Ptr 中存储的是与下一个数组元素的连接信息，所以只要替换了 Ptr 的值，就可以对数组中的

元素排序，使元素的排列顺序不同于其在内存上的物理排列顺序。首先，我们来试着把数组中元素 A 的 Ptr 的值改为元素 C 的地址，然后把元素 C 的 Ptr 的值改为元素 B 的地址。通过这样一改，原有的顺序 A → B → C 就变成了 A → C → B（如图 6.12 所示）。

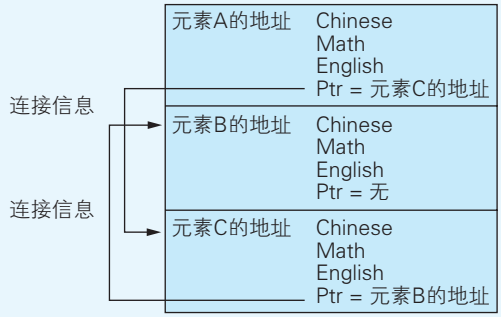

图 6.12　只要改变连接信息，元素就可以呈现出新顺序，不同于其在内存上的物理排列顺序

为什么说链表很方便呢？请思考一下不使用链表且还要对大量的数据进行排序时应该怎么处理。答案是那就必须要改变元素在内存上的物理排列顺序了。这不仅要改变大量数据的位置，而且程序的处理时间也会变长。如果是使用链表，对元素的排序就只需要变更 Ptr 的值，程序的处理时间也会缩短。这个特性也适用于对元素进行删除和插入。在实际的程序中，为了能够处理大量的数据，都会在各种各样的情景下灵活地运用链表。不使用链表的情况倒是很少见。

只要明白了链表的构造，也就明白了二叉树的实现方法。在二叉树的实现中，用的还是自我引用的结构体，只不过要改为要带有两个连接信息的成员的自我引用结构体（如代码清单 6.11 所示）。

代码清单 6.11 带有 2 个链表连接信息的自我引用结构体

```
struct TestResult{
    char Chinese;              /* 语文成绩 */
    char Math;                 /* 数学成绩 */
    char English;              /* 英语成绩 */
    struct TestResult* Ptr1; /* 指向其他元素的指针 1 */
    struct TestResult* Ptr2; /* 指向其他元素的指针 2 */
};
```

二叉树多用于实现那些用于搜索数据的算法，比如“二分查找法”。比起只使用链表，使用二叉树能够更快地找到数据。因为搜索数据时并不是像在简单数组中那样沿一条线搜索，而是寻着二叉树不断生长出来的两根树杈中的某一枝搜索，这样就能更快地找到目标数据了（如图 6.13 所示）。

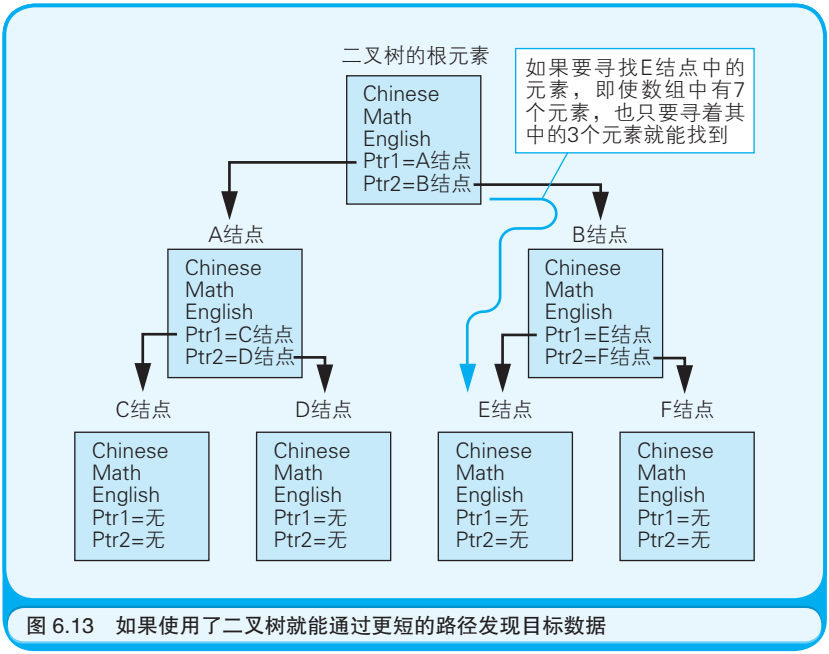

图 6.13 如果使用了二叉树就能通过更短的路径发现目标数据

在 C 语言的教科书等资料中，都会把结构体、指针、自我引用的结构体这些概念放到最后讲解，它们被认为是在 C 语言的应用中最难理解的部分。而诸位却通过学习本章，一下子触及到了这些概念。如果诸位有偏爱的编程语言，也请想一想用那门语言该如何实现栈、队列、链表和二叉树。无论是在哪种编程语言中，数据结构的基础都是数组，因此设法灵活地运用数组才是关键。

☆　　☆　　☆

通过学习第 5 章和第 6 章，诸位就相当于上完了算法和数据结构基础这门课程。虽然一路讲解了各种各样的要点，但是在最后还是请允许笔者再提醒诸位一点：即便是有了由睿智的学者们提出的那些了不起的算法和数据结构，也不能 100% 地依赖它们。希望诸位要经常自己动脑思考算法和数据结构。在了解了典型的算法和数据结构（也就是基础）之后，请不要忘记还要灵活地去运用它们。只要诸位灵活地去运用典型算法和数据结构，就能创造出出色的原创作品，而能够创造出原创作品的程序员才是真正的技术者。

在接下来的第 7 章中，笔者将从各个角度介绍面向对象编程。敬请期待！

第 7 章 成为会使用面向对象编程的程序员吧

热身问答

在阅读本章内容前，让我们先回答下面的几个问题来热热身吧。

初级问题

Object 翻译成中文是什么?

中级问题

OOP 是什么的缩略语?

高级问题

哪种编程语言在 C 语言的基础上增加了对 OOP 的支持?

怎么样？被这么一问，是不是发现有一些问题无法简单地解释清楚呢？下面，笔者就公布答案并解释。

答案

初级问题：Object 翻译成中文是“对象”。

中级问题：OOP 是 Object Oriented Programming（面向对象编程）的缩略语。

高级问题：C++ 语言。

解释

初级问题：对象（Object）是表示事物的抽象名词。

中级问题：面向对象也可以简称为 OO（Object Oriented）。

高级问题：++ 是表示自增（每次只将变量的值增加 1）的 C 语言运算符。之所以被命名为 C++，是因为 C++ 在 C 语言的基础上增加了面向对象的机制这一点。另外，将 C++ 进一步改良的编程语言就是 Java 和 C# 语言。

本章重点

在本章笔者想让诸位掌握的是有关面向对象编程的概念。理解面向对象编程有着各种各样的方法，程序员们对它的观点也会因人而异。本章会将笔者至今为止遇到过的多名程序员的观点综合起来，对面向对象编程进行介绍。哪种观点才是正确的呢？这并不重要，重要的是把各个角度的观点整合起来，而后形成适合自己的理解方法。在读完本章后，请诸位一定要和朋友或是前辈就什么是面向对象编程展开讨论。

7.1 面向对象编程

面向对象编程（OOP，Object Oriented Programming）是一种编写程序的方法，旨在提升开发大型程序的效率，使程序易于维护[①]。因此在企业中，特别是管理层的领导们都青睐于在开发中使用面向对象编程。因为如果开发效率得以提高、代码易于维护，那么就意味着企业可以大幅度地削减成本（开发费用＋维护费用）。甚至可以这样说，即使管理者们并不十分清楚面向对象编程到底是什么，他们也还是会相信“面向对象编程是个好东西”。

但是在实际的开发工作中，程序员们却有一种对面向对象编程敬而远之的倾向。原因在于他们不得不重新学习很多知识，还会被新学到的知识束缚自己的想法，导致无法按照习惯的思维开发。以笔者写书的经验来看，如果是讲解传统的编程方法，那么只需要写一本书就够了，而讲解面向对象编程则需要写两本书。直说的话就是面向对象编程太麻烦了。甚至还曾听到过这样的传言：若是在面向开发人员的杂

① 这里所说的维护指的是对程序功能的修改和扩展。