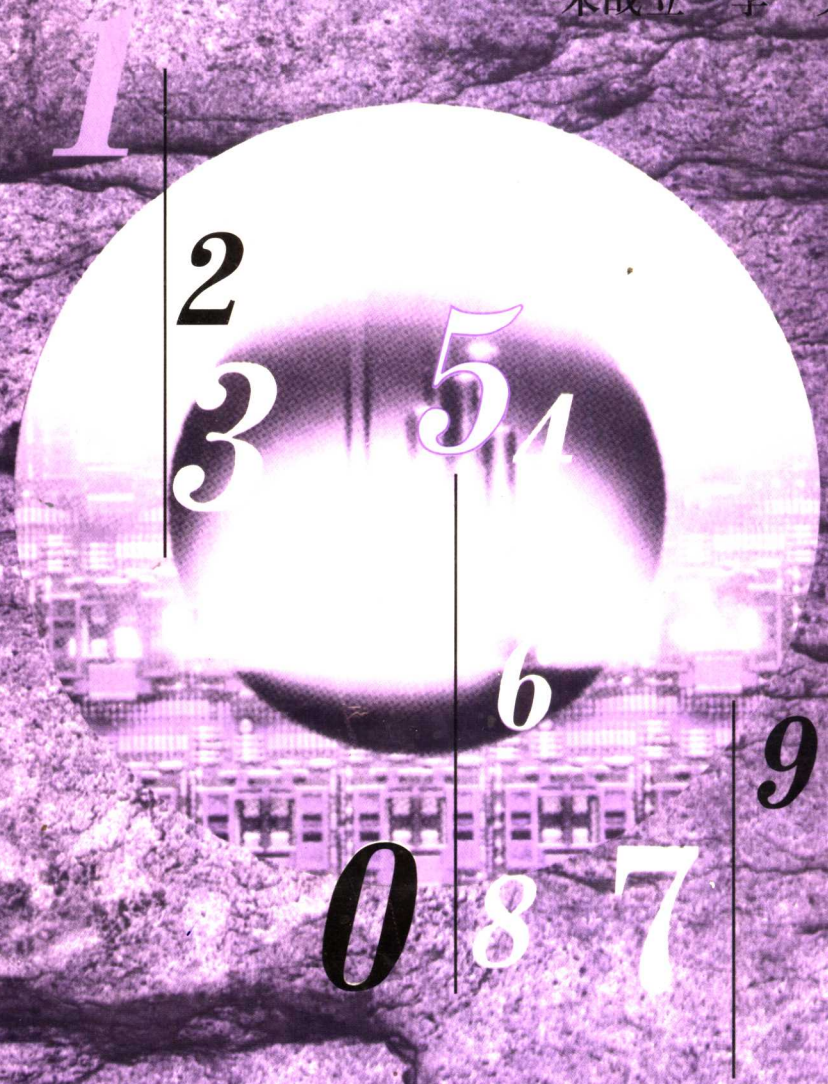
数据结构

——使用C语言

朱战立 李 文 编著



西安交通大学出版社

数 据 结 构

——使用C语言

朱战立 李 文 编著

西安交通大学出版社

内容简介

本书系统地介绍了各种类型的数据结构和查找、排序的各种方法。对于每一种类型的数据结构，都详细阐述了基本概念、各种不同的存储结构和不同存储结构上的一些操作实现算法，并列举了许多具体实例帮助读者理解。对查找和排序中的每一种方法，除给出了具体算法外，还进行了效率分析。另外，本书还介绍了递归程序设计方法和文件的组织方法。本书采用C语言作为算法描述语言。

本书概念清楚，内容丰富，深度适中，叙述简明。全书安排有大量例题帮助读者理解。每章后面配有适量习题。

本书既可作为大专院校计算机应用专业和非计算机专业的教科书，也可作为从事计算机应用的工程技术人员的自学参考书。

(陕)新登字 007 号

数 据 结 构

——使用C语言

朱战立　李　文　编著

责任编辑　孙文声　赵跃进

责任校对　郭丽芳

*

西安交通大学出版社出版发行

(西安市咸宁西路28号　邮政编码:710049　电话:(029)3268316)

西安向阳印刷厂印装

各地新华书店经销

*

开本:787mm×1092mm　1/16　印张:17.5　字数:421千字

1997年3月第1版　1999年1月第4次印刷

印数:12 001～15 500

ISBN 7-5605-0883-9/TP·142　定价:20.00元

前言

数据结构是一门计算机专业和计算机应用专业本科、专科学生必修的专业基础课。它是许多涉及数据结构设计和算法设计课程，如操作系统、编译原理、计算机图形学、人工智能等的先导课。数据结构也是非计算机专业培养学生软件设计能力重点选择的一门专业课

本书共分 11 章。第 1 章介绍了数据结构的基本概念，概述了 C 语言的数据类型，说明了算法设计目标和算法效率度量；第 2 章到第 5 章分别介绍了线性结构中线性表、堆栈、队列、串和数组的基本概念，以及存储结构和一些基本操作的实现；第 6 章介绍了递归概念以及递归算法的设计方法和用非递归算法模拟递归算法的基本方法；第 7 章和第 8 章分别介绍了非线性结构中树和二叉树以及图的基本概念、存储结构、一些基本操作的实现及它们的典型应用；第 9 章和第 10 章分别介绍了排序和查找的各种常用算法，并对各种算法进行了简单的性能分析；第 11 章介绍了文件的概念和文件的几种组织方法。

本书在内容组织上力求丰富充实，结合实用；在语言描述上力求深入浅出、简洁明了。为了便于学习和理解，书中列举了大量例题，并在一些章后设有专门小节讨论较大型的综合应用问题。本书每章都配有适量习题。

本书既可作为计算机应用专业本科、专科学生的教材，也可作为非计算机专业本科学生的教材。对从事计算机应用的工程技术人员，本书也是十分有价值的参考书。

目前国内数据结构教材基本使用类 PASCAL 语言或用 PASCAL 语言作为算法描述语言，这与 C 语言的广泛使用和计算机应用专业教学的发展变化要求极不相符。本书是编著者在多年来讲授数据结构和 C 语言程序设计课程的基础上编写的。本书采用 C 语言作为算法描述语言，书中全部算法都通过了上机调试。本书第 1 章至第 6 章及第 11 章由朱战立编写，第 7 章至第 10 章由李文编写，全书由朱战立修改定稿。

本书由西安交通大学冯博琴教授审阅，西安交通大学出版社对本书的出版给予了大力支持，在此一并表示谢意。

编著者

1997.1

目录

第1章 绪 论

计算机是对各种各样数据进行处理的机器。在计算机中如何组织数据，如何处理数据，从而如何更好地利用数据是计算机科学的基本研究内容。对各种计算机应用专业的学生来说，掌握数据在计算机中的各种组织和处理方法是继续深入学习的基础。

1.1 数据结构的基本概念

早期的计算机多用于进行数值计算，数值计算的特点是数据元素间的关系简单，但计算复杂。随着计算机应用范围的扩展，目前计算机被更多地用于非数值处理，如管理、控制等领域，非数值处理问题的特点是数据元素间的关系复杂，而计算较简单。

数据元素间的结构关系（或称逻辑结构）有几种基本形式。最简单的是线性结构，这时其有关结构的性质可以归纳为下述一些问题：哪一个数据元素是线性表中的第一个数据元素？哪一个数据元素是线性表中的最后一个数据元素？某一个数据元素的前驱和后继各是哪一个数据元素等等。更为复杂的这种结构关系有树形结构——表示着等级和分枝的关系，以及图形结构——表示着更复杂的客观世界事物之间的关系。

为了有效地在计算机上解决具有各种结构关系的实际问题，我们还必须研究这种具有结构关系的数据在计算机内部的存储方法，或称存储结构，以及在计算机中处理这样的具有结构关系数据所需进行的操作和操作的实现方法。

我们先来看一个最简单的线性表关系的例子。

例 1-1 建立一个住院病人押金情况表。住院病人押金情况表包括：姓名、性别、年龄、住院押金。要求每当病人住院时，插入一条记录，每当病人出院时，删去该病人记录。

分析：每个病人的数据为一条记录，每个记录包括姓名、性别、年龄、住院押金四个数据项。所有病人的记录构成一个线性表。这构成了该问题的逻辑数据结构。

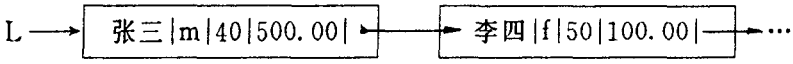
线性表上要求有插入、删除一条记录的操作。插入操作和删除操作构成了该线性表的操作集合。

线性表可以选用不同的存储结构。具体到这个线性表可以有二种存储结构：

(1) 顺序存储结构：即把所有记录依次存储在一个数组中。如

数组下标	姓名	性别	年龄	住院押金
0	张三	m	40	500.00
1	李四	f	50	100.00
⋮	⋮	⋮	⋮	⋮

(2) 单链存储结构：即把每个记录看作一个结点，让所有结点依次勾链。如



显然，存储结构不同则实现操作的算法就不同。

由此可见，一个具体问题的软件设计通常包括三个步骤：(1)弄清该问题的逻辑数据结构；(2)设计该问题的具体存储结构；(3)设计该问题在具体存储结构下的操作实现算法。

下边我们较为准确地定义我们所引进的几个基本术语：

数据 在计算机中这个术语含义非常广泛，可以认为它是描述客观事物的数字、字符、以及所有能输入到计算机中并能为计算机所接受的符号集。

数据元素 数据的基本单位，是计算机进行输入输出操作的最小单位。大多数情况下，一个数据元素由若干个数据项组成。数据项是数据不可分割的最小单位。

数据结构 是指计算机处理的数据元素的组织形式和相互间关系。从数据元素之间的不同特性分，数据结构有三种基本类型：线性结构、树形结构和图形结构。从不同的抽象层次看，数据结构可分为逻辑数据结构和物理数据结构(或称存储结构)。存储结构主要有两种基本类型：顺序存储结构和链式存储结构。

操作 是指数据结构上需要或能够进行的处理。对一个具体问题来说，不同的存储结构实现操作的具体算法就不同；相同的存储下也可有许多种不同的操作实现算法。

我们研究数据结构，需要研究如下问题：一个具体问题的逻辑数据结构是什么？适宜选用什么样的存储结构？采用什么样的操作实现算法效率更高？由此我们可以说，数据结构是一门研究非数值处理问题的软件设计中计算机的操作对象以及它们之间的关系和操作实现等的学科。换句话说，数据结构的研究包括三个方面的内容：数据的逻辑结构、数据的物理存储结构和数据的操作实现算法。

1.2 数据类型和抽象数据类型

1. 数据类型

数据类型这个概念最早出现在高级程序设计语言的实现中。我们知道，计算机硬件系统并不能直接处理如整数、浮点数、双精度数等，需按一定规则转换成二进制码来表示和处理。从计算机硬件系统的角度看，计算机能处理的数据类型只包括位、字节和字。高级程序设计语言设计者们引入整数、浮点数、双精度数等，对计算机硬件系统来说就是一种新的数据结构。对这种数据结构的操作，需通过编译器转化成机器语言的位、字节和字这样的计算机硬件能接受的数据类型的操作来实现。从高级程序设计语言用户的角度看，数据结构实现了信息的隐蔽，即将一切用户不必了解的细节都封装在数据结构之中，从而实现了数据的抽象。例如，硬件系统直接支持的机器语言程序需考虑整数的具体机器表示和整数运算的机器实现算法，而高级程序设计语言用户在使用整数时，不需要了解整数在计算机内部是如何表示和各种运算是如何实现的。对如两整数求和这样的问题，高级程序设计语言用户只需了解其数学上求和的抽象操作含义。这样，在硬件系统机器语言的数据类型支持下，高级程序设计语言实现了整数这样的数据结构。进一步，对高级程序设计语言来说，整数就是它的数据类型，或称固有数据类型，高级程序设计语言用户可直接使用。在用高级程序设计语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。数据类型显式或隐式地规定了在程序执行期间变量、常

量或表达式所有可能的取值范围，以及允许进行的操作。例如，C 语言中的整数类型，取值范围为[－maxint，maxint]上的整数(maxint 为特定计算机所允许的最大整数)，定义在该取值区间上的一组操作为：加(＋)、减(－)、乘(＊)、整除(/)和取余(％)。

因此我们可以给数据类型作如下定义：数据类型是一个值的集合和定义在这个值的集合上的一组操作的总称。或者说，数据类型是某种程序设计语言中已实现的数据结构。

在程序设计语言提供的数据类型的支持下，我们就可根据从问题中抽象出来的各种数据模型，逐步构造出描述这些数学模型的各种新的数据结构。

2. 抽象数据类型

抽象数据类型指的是用户进行软件系统设计时从问题的数学模型中抽象出来的逻辑数据结构和逻辑数据结构上的操作，而不考虑计算机的具体存储结构和操作的具体实现算法。

一个具体问题的抽象数据类型的定义通常采用简洁、严谨的文字描述，一般包括数据元素、结构和操作三方面的内容。如本章例 1-1 问题的抽象数据类型(Abstract Date Type，简写为 ADT)可定义如下：

ADT Patient _ list

数据元素　a_i 是一个记录，每个记录由数据项姓名、性别、年龄和住院押金的一组值组成，$i=1,2,\cdots,n(n>0)$

结构　线性表结构

操作　设 P 为 Patient _ list 类型的线性表，其操作包括：

(1) INITIATE(P)　初始化。设定一个空的线性表 P。

(2) INSERT(P,i,x)　前插。把给定的 Patient _ list 类型的姓名项为 x 的数据元素插入到线性表 P 中第 i 个数据元素之前，操作之前长度为 n 的线性表变成长度为n＋1的线性表。

(3) DELETE(P,x)　删除。在线性表 P 中若寻找到某一个数据元素 $a_i(i=1,2,\cdots,n$ $n>0)$，满足 a_i 的姓名项值等于 x，则将 a_i 删除，使操作之前长度为 n 的线性表变成长度为n－1的线性表；若找不到满足条件的 a_i 则不作任何操作。

抽象数据类型是软件设计者的一个十分有用的工具。软件工程学和结构化程序设计方法都告诉我们，软件系统的设计应采用自顶向下、逐步细化的方法，这样才能使规模庞大的软件系统设计具有工程化按部就班一步步实现的基础。抽象数据类型就是这样的一种工具。通过前边的讨论我们已经看到，对于一个已确定的抽象数据类型，其实现者据此完成存储结构设计和具体操作的算法设计，再进一步完成程序编码和调试测试；对于这种抽象数据类型的使用者来说，可以象使用高级程序设计语言中的数据类型一样使用它，而不用去顾及其具体的存储结构形式和操作实现细节。当然，对不同的高级程序设计语言、不同的机器所实现的抽象数据类型是有一定差别的，例如不同高级程序设计语言或不同机器所实现的整数的取值范围和一些运算符号都有一些差别，但其抽象的数学含义均相同。

1.3　C 语言的数据类型

本节我们简要回顾 C 语言的数据类型。

1. C 语言的基本数据类型

C 语言有三种由硬件系统直接支持实现的基本数据类型：int 型、float 型和 char 型。int 型

可以有三个限定词：short、long 和 unsigned。short 或 long 缩小或扩大了 int 型整数的存储空间和数值表示范围。但精确的 short int、int 和 long int 型存储空间和数值表示范围由具体 C 版本和具体机器确定。unsigned int 表示其变量只取正整数。

float 型有三种形式：float、double 和 long double。存储空间分别为 4 个字节、8 个字节和 16 个字节。

char 型存储空间为一个字符。由于字符在机器内以整数形式表示，所以 C 语言也允许用 char 型定义占 1 个字节、其数值范围为－127～128 的变量，但是为避免混淆，本书将不这样使用。

2. C 语言的指针类型

C 语言允许直接对存放变量的地址进行操作。如定义 int a，则 &a 表示变量 a 的地址，也称作指向变量 a 的指针。存放地址的变量称作指针变量。如再定义 int *pa，则语句 pa=&a 即为把变量 a 的地址赋给指针变量 pa；而语句 a=*pa 则为把指针 pa 所指地址中的值赋给变量 a。另外，这里 pa 的含义不只是指针，而且是确定的整型类型指针。这是因为不同数据类型变量在内存中分配的空间大小不同。假设 int 型占 isize 个字节，并设系统为 pa 分配的内存地址为 1000，则 pa+1 为 1000+isize；而 *(pa+1)表示从 1000+isize 开始的 isize 个字节中的整型数。

C 语言中指针的一个重要作用是在函数间传递数值。对一个形参为简单变量的函数，参数传递是值传送，即把实参的值拷贝给形参，这样函数内形参的值被修改时实参的值不会跟着变化。若函数采用指针作形参，则参数传递采用地址传送，即把实参的地址传送给形参，或者说实参、形参共用一个存储单元。这时若函数内形参的值改变，则实参中的值将跟着变化。利用指针形参可设计函数的输出参数和变参。

例 1-2 值传送和地址传送。

(1) 值传送

```
x=5;
printf("%d\n",x);
funct(x);
printf("%d\n",x);
void funct(int y)
{ y++;
  printf("%d\n",y);
}
```

输出结果为：

5(第 2 行的输出)

6(第 7 行的输出)

5(第 4 行的输出，不随形参改变)

(2) 地址传送

```
x=5;
printf("%d\n",x);
funct(&x);
```

```
printf("%d\n",x);
void funct(int *py)
{   (*py)++;
    printf("%d\n",*py);
}
```

输出结果为：

5(第 2 行的输出)

6(第 7 行的输出)

6(第 4 行的输出，随着形参一起变化)

3. C 语言的数组类型

数组是同一类型的一组有序数据的集合。数组名标识了这一组数，数组元素下标指示了数组元素在该数组中的顺序位置。可以有一维数组和多维数组。数组的两个最基本操作是存和取。例如有定义 int a[100],x;则语句 a[i]=x 表示把 x 的值赋给 a[i];语句 x=a[i]表示把a[i]的值赋给 x。

数组下标的最小值称为下界，这在 C 语言中总是 0。数组下标的最大值称为上界，C 语言中数组上界为数组定义值减 1，如上例数组 a 的上界即为 99。数组的上下界范围等于上界减下界加 1，如上例数组 a 的上下界范围为 100。

一个数组名代表了这一组数的起始地址，也就是说，数组名是一个指针变量。若 a 是如上定义的一维数组，则有：

&a[0]=a,&a[1]=a+1,……,&a[i]=a+i

a[0]=*a,a[1]=*(a+1),……,a[i]=*(a+i)

假如还有定义 int *p,则语句 p=a 表示把数组 a 的首地址赋给指针变量 p。语句 p+1 等价于语句 a+1，但 p 是指针变量，本身可改变，a 是地址常量，本身不可改变。

当数组作为函数参数时，其传递方法和指针作为函数参数的传递方法相同，即采用地址传送法。因此，对函数中的一维数组形参，函数定义时仅需定义形参数组名，而不必定义数组上下界范围，因为系统并不实际为形参数组分配内存。函数中的数组形参也可用指针变量表示，因为当无需定义数组范围时，数组形参名和指针变量都是指针类型，两者没有差别。

例 1-3 计算数组元素平均值的函数。

```
float avg(float a[],int size)
{
    int i;
    float sum;
    sum=0.0;
    for (i=0;i<size;i++)
        sum=sum+a[i];
    return (sum/size);
} /* end of avg */
```

主函数可以如下描述：

```
# define MAXNUM 100
```

```
main ()
{
  float a[MAXNUM];
  int n;
  scanf("%d",&n);
  ...
  avg (a,n);
}
```

一个两维数组可看作其每一个数组元素都是一个一维数组的一维数组。假如有定义

```
int b[3][4];
```

则要存取两维数组 b 需要给出行下标(第一维)和列下标(第二维)。但这实际上是一个逻辑数据结构,因为计算机的内存是一维的,只能一个一个元素顺序存放。C 语言中其物理存储结构是行主序存放方法实现的,即第一行的所有元素存完后,再顺序存放第二行的元素,以此类推。假设两维数组 b 的起始地址为 base(b),整型类型的字节数为 isize,则二维整型数组元素 b[i][j](0≤i<3,0≤j<4)的内存地址为:

base(b)+(i * 4+j) * isize

由此可知,两维数组有两个指针,一个是行指针,一个是列指针。两维数组 b 有图 1-1 所示的指针关系。

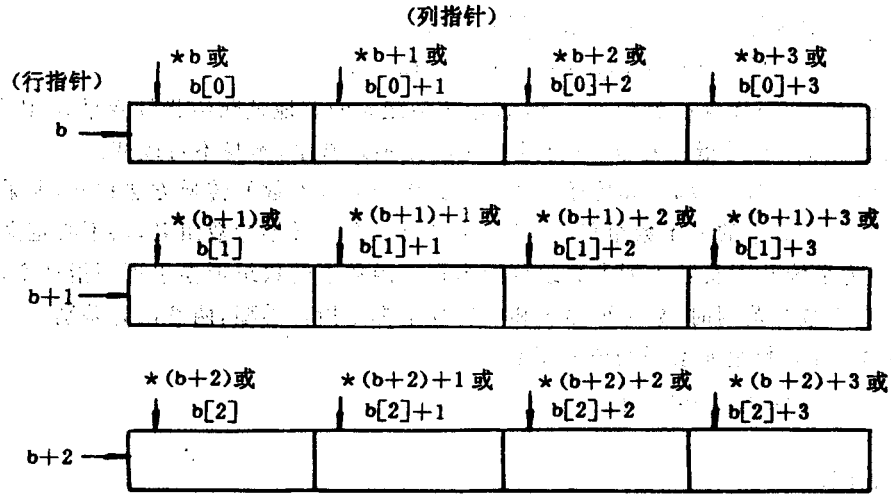


图 1-1 二维数组的行指针和列指针

其中行指针加 1 表示加 4 * isize 个字节,列指针加 1 表示加 isize 个字节。

多维数组的定义方法和内存存储顺序可以类推。

需要注意的是,C 语言中系统并不检查数组下标是否越界,需要用户自己注意。另外,C 语言不允许数组整体进行存取操作。

4. C语言的字符串

C语言中没有单独的字符串类型、字符串定义成字符数组。每个字符串由转义符'\0'指示其结束。一个字符串常量由一对双引号指示。一个字符串常量被存入内存时，系统自动在其末尾添加字符串结束标志'\0'。假设有如下初始化定义：

```
char string [80]="Data structure";
```

则 char 型的一维数组 string 的数组范围是 80，字符串长度是 14。字符串长度不包括'\0'字符，但'\0'字符需占用一个字符空间。因此必须满足：字符串长度≤数组范围－1。

C语言中字符串的操作都有系统函数，如字符串长度函数 strlen(string)，字符串的拷贝函数 strcpy(str1,str2)，字符串的连接函数 strcat(str1,str2)，等等。这些函数都包含在头文件 string.h 中，若调用这些函数需有如下文件包含：

```
#include "string.h"
```

关于字符串函数的较详细资料请参阅附录Ⅲ或相关C语言资料。

5. C语言的结构体类型

结构体由一组称为结构体成员的项组成，每个结构体成员都有自己的标识符。在有些高级程序设计语言中，结构体和结构体成员也分别称作记录和字段。

定义一个结构体类型变量由两步组成。第一步，定义结构体类型；第二步，定义结构体类型变量。例如，

```
struct student
{ char name[20];
  int age;
  float score;
};     /* 结构体类型定义 */
struct student stu1,stu2[100], *p;      /* 结构体类型变量定义 */
```

或简写作：

```
struct
{ char name[20];
  int age;
  float score;
} stu1,stu2[100], *p;
```

设 x 为 float 型变量，则可有如下操作：

```
p=&stu1;        /* p 指向 stu1 首地址 */
stu1.score=x;
```

或

```
p->score=x;
x=stu2[0].score;
```

结构体变量还可整体赋值，如可有：

```
stu1=stu2[1];
```

一个结构体的成员还可是另一个已定义的结构体类型。例如，

```
struct studnode
  { struct student stud;
```

```
    struct studnode *p;
  };          */结构体类型定义*/
struct studnode st;  /*结构体类型变量定义*/
```

此时，成员 score 应书写成 st.stud.score。

结构体变量可以作为函数参数和函数返回值。当结构体变量作为函数参数时，其参数传递方法和简单变量形参一样，即采用值传递。

6. C 语言的共用体类型

共用体是把不同的成员项组织为一个整体，它们在内存中共用一段存储单元，但不同的成员项以不同的方式被解释。在有些高级程序设计语言(如 PASCAL)中，共用体类型也称作记录的变体。共用体类型变量的定义和操作方法与结构体类型变量的相同。例如，

```
union exam
{     int class;
      char group[20];
};      /*共用体类型定义*/
union exam category;     /*共用体类型变量定义*/
```

设有如下定义：

```
int i;
char a[20];
```

则可有如下操作：

```
i=category.class;
category.class=i;
strcpy (a, category.group);
```

需要注意的是，一个共用体变量不能同时存放多个成员项的值，而只能存放其中最后赋予它的值。

7. C 语言的枚举类型

枚举类型是指这种类型变量的取值只能是指定的若干个值之一。定义一个枚举类型变量也由两步组成，即枚举类型定义和枚举类型变量定义。例如，

```
enum colorname
{red, yellow, blue, white, black};     /*枚举类型定义*/
enum colorname color;     /*枚举类型变量定义*/
```

此时，枚举变量 color 只能取枚举类型定义中的 5 个值之一。需注意的是这些值只是标识符，不是字符串，系统实现时自动把这些值依次表示为由 0 开始的数值。即有 red=0，yellow=1，…，black=4。

C 语言的枚举类型主要用于提高程序的可读性。

8. C 语言的定义类型

C 语言允许用 typedef 关键字来定义同名数据类型名，这可增强程序的抽象性、简洁性和可读性。例如可定义：

```
typedef char elemtype;
```

或

```
typedef struct plist
```