11 结构体与共用体

1.1 定义一个结构的一般形式

在实际问题中，一组数据往往具有不同的数据类型。例如，在学生登记表中，姓名应为字符型；学号可为整型或字符型；年龄应为整型；性别应为字符型；成绩可为整型或实型。显然不能用一个数组来存放这一组数据。因为数组中各元素的类型和长度都必须一致，以便于编译系统处理。为了解决这个问题，C 语言中给出了另一种构造数据类型——"结构（structure）"或叫"结构体"。它相当于其它高级语言中的记录。"结构"是一种构造类型，它是由若干"成员"组成的。每一个成员可以是一个基本数据类型或者又是一个构造类型。结构既是一种"构造"而成的数据类型，那么在说明和使用之前必须先定义它，也就是构造它。如同在说明和调用函数之前要先定义函数一样。

定义一个结构的一般形式为：

```
struct 结构名
{成员表列};
```

成员表列由若干个成员组成，每个成员都是该结构的一个组成部分。对每个成员也必须作类型说明，其形式为：

```
类型说明符 成员名;
```

成员名的命名应符合标识符的书写规定。例如：

```
struct stu
{
    int num;
    char name[20];
    char sex;
```

```
    float score;
};
```

在这个结构定义中，结构名为 stu，该结构由 4 个成员组成。第一个成员为 num，整型变量；第二个成员为 name，字符数组；第三个成员为 sex，字符变量；第四个成员为 score，实型变量。应注意在括号后的分号是不可少的。结构定义之后，即可进行变量说明。凡说明为结构 stu 的变量都由上述 4 个成员组成。由此可见， 结构是一种复杂的数据类型，是数目固定，类型不同的若干有序变量的集合。

1.1 结构类型变量的说明

说明结构变量有以下三种方法。以上面定义的 stu 为例来加以说明。

1. 先定义结构，再说明结构变量。

如：

```
struct stu
{
    int num;
    char name[20];
    char sex;
    float score;
};
struct stu boy1,boy2;
```

说明了两个变量 boy1 和 boy2 为 stu 结构类型。也可以用宏定义使一个符号常量来表示一个结构类型。

例如：

```
#define STU struct stu
STU
{
    int num;
    char name[20];
    char sex;
    float score;
};
STU boy1,boy2;
```

2. 在定义结构类型的同时说明结构变量。

例如：

```
struct stu
{
    int num;
    char name[20];
    char sex;
    float score;
}boy1,boy2;
```

这种形式的说明的一般形式为：

```
    struct 结构名
```

{
成员表列
}变量名表列;

3. **直接说明结构变量。**

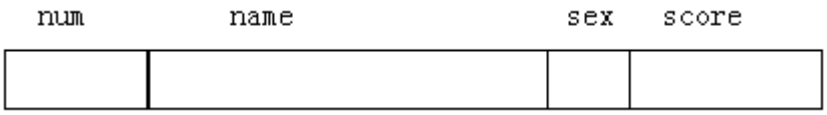
例如：

```
struct
{
    int num;
    char name[20];
    char sex;
    float score;
}boy1,boy2;
```

这种形式的说明的一般形式为：

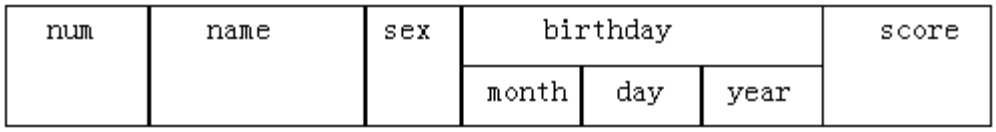
```
struct
{
  成员表列
}变量名表列;
```

第三种方法与第二种方法的区别在于第三种方法中省去了结构名，而直接给出结构变量。三种方法中说明的 boy1,boy2 变量都具有下图所示的结构。



说明了 boy1,boy2 变量为 stu 类型后，即可向这两个变量中的各个成员赋值。在上述 stu 结构定义中，所有的成员都是基本数据类型或数组类型。

成员也可以又是一个结构，即构成了嵌套的结构。例如，下图给出了另一个数据结构。



按图可给出以下结构定义：

```
struct date
{
    int month;
    int day;
    int year;
};
struct{
    int num;
    char name[20];
    char sex;
    struct date birthday;
    float score;
}boy1,boy2;
```

首先定义一个结构 date，由 month(月)、day(日)、year(年) 三个成员组成。 在定义并

说明变量 boy1 和 boy2 时，其中的成员 birthday 被说明为 data 结构类型。成员名可与程序中其它变量同名，互不干扰。

1.1 结构变量成员的表示方法

在程序中使用结构变量时，往往不把它作为一个整体来使用。在 ANSI C 中除了允许具有相同类型的结构变量相互赋值以外，一般对结构变量的使用，包括赋值、输入、输出、运算等都是通过结构变量的成员来实现的。

表示结构变量成员的一般形式是：

结构变量名.成员名

例如：

boy1.num　　即第一个人的学号

boy2.sex　　即第二个人的性别

如果成员本身又是一个结构则必须逐级找到最低级的成员才能使用。

例如：

boy1.birthday.month

即第一个人出生的月份成员可以在程序中单独使用，与普通变量完全相同。

1.1 结构变量的赋值

结构变量的赋值就是给各成员赋值。可用输入语句或赋值语句来完成。

【例 11.1】给结构变量赋值并输出其值。

```
main()
{
    struct stu
    {
      int num;
      char *name;
      char sex;
      float score;
    } boy1,boy2;
    boy1.num=102;
    boy1.name="Zhang ping";
    printf("input sex and score\n");
    scanf("%c %f",&boy1.sex,&boy1.score);
    boy2=boy1;
    printf("Number=%d\nName=%s\n",boy2.num,boy2.name);
    printf("Sex=%c\nScore=%f\n",boy2.sex,boy2.score);
}
```

本程序中用赋值语句给 num 和 name 两个成员赋值，name 是一个字符串指针变量。用 scanf 函数动态地输入 sex 和 score 成员值，然后把 boy1 的所有成员的值整体赋予 boy2。最后分

别输出 boy2 的各个成员值。本例表示了结构变量的赋值、输入和输出的方法。

1.1 结构变量的初始化

和其他类型变量一样，对结构变量可以在定义时进行初始化赋值。

【例 11.2】对结构变量初始化。

```
main()
{
    struct stu    /*定义结构*/
    {
      int num;
      char *name;
      char sex;
      float score;
    }boy2,boy1={102,"Zhang ping",'M',78.5};
 boy2=boy1;
 printf("Number=%d\nName=%s\n",boy2.num,boy2.name);
 printf("Sex=%c\nScore=%f\n",boy2.sex,boy2.score);
}
```

本例中，boy2,boy1 均被定义为外部结构变量，并对 boy1 作了初始化赋值。在 main 函数中，把 boy1 的值整体赋予 boy2，然后用两个 printf 语句输出 boy2 各成员的值。

1.1 结构数组的定义

数组的元素也可以是结构类型的。因此可以构成结构型数组。结构数组的每一个元素都是具有相同结构类型的下标结构变量。在实际应用中，经常用结构数组来表示具有相同数据结构的一个群体。如一个班的学生档案，一个车间职工的工资表等。

方法和结构变量相似，只需说明它为数组类型即可。

例如：

```
struct stu
{
    int num;
    char *name;
    char sex;
    float score;
}boy[5];
```

定义了一个结构数组 boy，共有 5 个元素，boy[0]～boy[4]。每个数组元素都具有 struct stu 的结构形式。对结构数组可以作初始化赋值。

例如：

```
struct stu
{
```

```
        int num;
        char *name;
        char sex;
        float score;
    }boy[5]={
              {101,"Li ping","M",45},
              {102,"Zhang ping","M",62.5},
              {103,"He fang","F",92.5},
              {104,"Cheng ling","F",87},
              {105,"Wang ming","M",58};
             }
```

当对全部元素作初始化赋值时，也可不给出数组长度。

【例 11.3】计算学生的平均成绩和不及格的人数。

```
struct stu
{
    int num;
    char *name;
    char sex;
    float score;
}boy[5]={
          {101,"Li ping",'M',45},
          {102,"Zhang ping",'M',62.5},
          {103,"He fang",'F',92.5},
          {104,"Cheng ling",'F',87},
          {105,"Wang ming",'M',58},
        };
main()
{
    int i,c=0;
    float ave,s=0;
    for(i=0;i<5;i++)
    {
      s+=boy[i].score;
      if(boy[i].score<60) c+=1;
    }
    printf("s=%f\n",s);
    ave=s/5;
    printf("average=%f\ncount=%d\n",ave,c);
}
```

本例程序中定义了一个外部结构数组 boy，共 5 个元素，并作了初始化赋值。在 main 函数中用 for 语句逐个累加各元素的 score 成员值存于 s 之中，如 score 的值小于 60(不及格)即计数器 C 加 1，循环完毕后计算平均成绩，并输出全班总分，平均分及不及格人数。

【例 11.4】建立同学通讯录

```
#include"stdio.h"
#define NUM 3
struct mem
{
    char name[20];
    char phone[10];
};
main()
{
    struct mem man[NUM];
    int i;
    for(i=0;i<NUM;i++)
     {
      printf("input name:\n");
      gets(man[i].name);
      printf("input phone:\n");
      gets(man[i].phone);
     }
    printf("name\t\t\tphone\n\n");
    for(i=0;i<NUM;i++)
      printf("%s\t\t\t%s\n",man[i].name,man[i].phone);
}
```

本程序中定义了一个结构 mem，它有两个成员 name 和 phone 用来表示姓名和电话号码。在主函数中定义 man 为具有 mem 类型的结构数组。在 for 语句中，用 gets 函数分别输入各个元素中两个成员的值。然后又在 for 语句中用 printf 语句输出各元素中两个成员值。

1.1 结构指针变量的说明和使用

1.1.1指向结构变量的指针

一个指针变量当用来指向一个结构变量时，称之为结构指针变量。结构指针变量中的值是所指向的结构变量的首地址。通过结构指针即可访问该结构变量，这与数组指针和函数指针的情况是相同的。

结构指针变量说明的一般形式为：

```
struct 结构名 *结构指针变量名
```

例如，在前面的例题中定义了 stu 这个结构，如要说明一个指向 stu 的指针变量 pstu，可写为：

```
struct stu *pstu;
```

当然也可在定义 stu 结构时同时说明 pstu。与前面讨论的各类指针变量相同，结构指针变量也必须要先赋值后才能使用。

赋值是把结构变量的首地址赋予该指针变量，不能把结构名赋予该指针变量。如果 boy 是被说明为 stu 类型的结构变量，则：

pstu=&boy

是正确的，而：

pstu=&stu

是错误的。

结构名和结构变量是两个不同的概念，不能混淆。结构名只能表示一个结构形式，编译系统并不对它分配内存空间。只有当某变量被说明为这种类型的结构时，才对该变量分配存储空间。因此上面&stu 这种写法是错误的，不可能去取一个结构名的首地址。有了结构指针变量，就能更方便地访问结构变量的各个成员。

其访问的一般形式为：

(*结构指针变量).成员名

或为：

结构指针变量->成员名

例如：

(*pstu).num

或者：

pstu->num

应该注意(*pstu)两侧的括号不可少，因为成员符"."的优先级高于"*"。如去掉括号写作*pstu.num 则等效于*(pstu.num)，这样，意义就完全不对了。

下面通过例子来说明结构指针变量的具体说明和使用方法。

【例 11.5】

```
struct stu
    {
      int num;
      char *name;
      char sex;
      float score;
    } boy1={102,"Zhang ping",'M',78.5},*pstu;
main()
{
    pstu=&boy1;
    printf("Number=%d\nName=%s\n",boy1.num,boy1.name);
    printf("Sex=%c\nScore=%f\n\n",boy1.sex,boy1.score);
    printf("Number=%d\nName=%s\n",(*pstu).num,(*pstu).name);
    printf("Sex=%c\nScore=%f\n\n",(*pstu).sex,(*pstu).score);
    printf("Number=%d\nName=%s\n",pstu->num,pstu->name);
    printf("Sex=%c\nScore=%f\n\n",pstu->sex,pstu->score);
}
```

本例程序定义了一个结构 stu，定义了 stu 类型结构变量 boy1 并作了初始化赋值，还定义了一个指向 stu 类型结构的指针变量 pstu。在 main 函数中，pstu 被赋予 boy1 的地址，因此 pstu 指向 boy1。然后在 printf 语句内用三种形式输出 boy1 的各个成员值。从运行结果

可以看出：

结构变量.成员名

(*结构指针变量).成员名

结构指针变量->成员名

这三种用于表示结构成员的形式是完全等效的。

1.1.1指向结构数组的指针

指针变量可以指向一个结构数组，这时结构指针变量的值是整个结构数组的首地址。结构指针变量也可指向结构数组的一个元素，这时结构指针变量的值是该结构数组元素的首地址。

设 ps 为指向结构数组的指针变量，则 ps 也指向该结构数组的 0 号元素，ps+1 指向 1 号元素，ps+i 则指向 i 号元素。这与普通数组的情况是一致的。

【例 11.6】用指针变量输出结构数组。

```
struct stu
{
    int num;
    char *name;
    char sex;
    float score;
}boy[5]={
          {101,"Zhou ping",'M',45},
          {102,"Zhang ping",'M',62.5},
          {103,"Liou fang",'F',92.5},
          {104,"Cheng ling",'F',87},
          {105,"Wang ming",'M',58},
        };
main()
{
 struct stu *ps;
 printf("No\tName\t\t\tSex\tScore\t\n");
 for(ps=boy;ps<boy+5;ps++)
 printf("%d\t%s\t\t\t%c\t%f\t\n",ps->num,ps->name,ps->sex,ps->score);
}
```

在程序中，定义了 stu 结构类型的外部数组 boy 并作了初始化赋值。在 main 函数内定义 ps 为指向 stu 类型的指针。在循环语句 for 的表达式 1 中，ps 被赋予 boy 的首地址，然后循环 5 次，输出 boy 数组中各成员值。

应该注意的是，一个结构指针变量虽然可以用来访问结构变量或结构数组元素的成员，但是，不能使它指向一个成员。也就是说不允许取一个成员的地址来赋予它。因此，下面的赋值是错误的。

ps=&boy[1].sex;

而只能是：

ps=boy;(赋予数组首地址)

或者是：

ps=&boy[0];(赋予 0 号元素首地址)

1.1.1结构指针变量作函数参数

在 ANSI C 标准中允许用结构变量作函数参数进行整体传送。但是这种传送要将全部成员逐个传送，特别是成员为数组时将会使传送的时间和空间开销很大，严重地降低了程序的效率。因此最好的办法就是使用指针，即用指针变量作函数参数进行传送。这时由实参传向形参的只是地址，从而减少了时间和空间的开销。

【例 11.7】计算一组学生的平均成绩和不及格人数。用结构指针变量作函数参数编程。

```
struct stu
{
    int num;
    char *name;
    char sex;
    float score;}boy[5]={
        {101,"Li ping",'M',45},
        {102,"Zhang ping",'M',62.5},
        {103,"He fang",'F',92.5},
        {104,"Cheng ling",'F',87},
        {105,"Wang ming",'M',58},
      };
main()
{
    struct stu *ps;
    void ave(struct stu *ps);
    ps=boy;
    ave(ps);
}
void ave(struct stu *ps)
{
    int c=0,i;
    float ave,s=0;
    for(i=0;i<5;i++,ps++)
      {
        s+=ps->score;
        if(ps->score<60) c+=1;
      }
    printf("s=%f\n",s);
    ave=s/5;
    printf("average=%f\ncount=%d\n",ave,c);
}
```

本程序中定义了函数 ave，其形参为结构指针变量 ps。boy 被定义为外部结构数组，因此在整个源程序中有效。在 main 函数中定义说明了结构指针变量 ps，并把 boy 的首地址赋予它，使 ps 指向 boy 数组。然后以 ps 作实参调用函数 ave。在函数 ave 中完成计算平均成绩和统计不及格人数的工作并输出结果。

由于本程序全部采用指针变量作运算和处理，故速度更快，程序效率更高。

1.1 动态存储分配

在数组一章中，曾介绍过数组的长度是预先定义好的，在整个程序中固定不变。C 语言中不允许动态数组类型。

例如：

```
int n;
scanf("%d",&n);
int a[n];
```

用变量表示长度，想对数组的大小作动态说明，这是错误的。但是在实际的编程中，往往会发生这种情况，即所需的内存空间取决于实际输入的数据，而无法预先确定。对于这种问题，用数组的办法很难解决。为了解决上述问题，C 语言提供了一些内存管理函数，这些内存管理函数可以按需要动态地分配内存空间，也可把不再使用的空间回收待用，为有效地利用内存资源提供了手段。

常用的内存管理函数有以下三个：

1. 分配内存空间函数 malloc

调用形式：

(类型说明符*)malloc(size)

功能：在内存的动态存储区中分配一块长度为"size"字节的连续区域。函数的返回值为该区域的首地址。

“类型说明符”表示把该区域用于何种数据类型。

(类型说明符*)表示把返回值强制转换为该类型指针。

“size"是一个无符号数。

例如：

```
pc=(char *)malloc(100);
```

表示分配 100 个字节的内存空间，并强制转换为字符数组类型，函数的返回值为指向该字符数组的指针，把该指针赋予指针变量 pc。

2. 分配内存空间函数 calloc

calloc 也用于分配内存空间。

调用形式：

(类型说明符*)calloc(n,size)

功能：在内存动态存储区中分配 n 块长度为"size"字节的连续区域。函数的返回值为该区域的首地址。

(类型说明符*)用于强制类型转换。

calloc 函数与 malloc 函数的区别仅在于一次可以分配 n 块区域。

例如：

```
ps=(struet stu*)calloc(2,sizeof(struct stu));
```

其中的 sizeof(struct stu)是求 stu 的结构长度。因此该语句的意思是：按 stu 的长度分配 2 块连续区域，强制转换为 stu 类型，并把其首地址赋予指针变量 ps。

2. 释放内存空间函数 free

调用形式：

```
free(void*ptr);
```

功能：释放 ptr 所指向的一块内存空间，ptr 是一个任意类型的指针变量，它指向被释放区域的首地址。被释放区应是由 malloc 或 calloc 函数所分配的区域。

【例 11.8】分配一块区域，输入一个学生数据。

```
main()
{
    struct stu
    {
      int num;
      char *name;
      char sex;
      float score;
    }  *ps;
    ps=(struct stu*)malloc(sizeof(struct stu));
    ps->num=102;
    ps->name="Zhang ping";
    ps->sex='M';
    ps->score=62.5;
    printf("Number=%d\nName=%s\n",ps->num,ps->name);
    printf("Sex=%c\nScore=%f\n",ps->sex,ps->score);
    free(ps);
}
```

本例中，定义了结构 stu，定义了 stu 类型指针变量 ps。然后分配一块 stu 大内存区，并把首地址赋予 ps，使 ps 指向该区域。再以 ps 为指向结构的指针变量对各成员赋值，并用 printf 输出各成员值。最后用 free 函数释放 ps 指向的内存空间。整个程序包含了申请内存空间、使用内存空间、释放内存空间三个步骤，实现存储空间的动态分配。

1.1 链表的概念

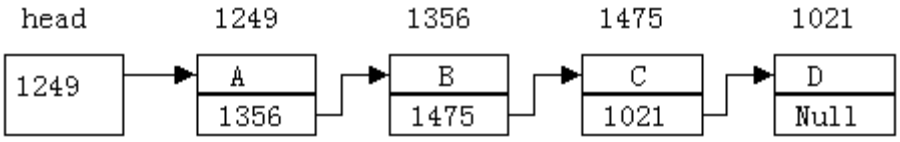
在例 7.8 中采用了动态分配的办法为一个结构分配内存空间。每一次分配一块空间可用来存放一个学生的数据，我们可称之为一个结点。有多少个学生就应该申请分配多少块内存空间，也就是说要建立多少个结点。当然用结构数组也可以完成上述工作，但如果预先不能准确把握学生人数，也就无法确定数组大小。而且当学生留级、退学之后也不能把该元素占用的空间从数组中释放出来。

用动态存储的方法可以很好地解决这些问题。有一个学生就分配一个结点，无须预先确定学生的准确人数，某学生退学，可删去该结点，并释放该结点占用的存储空间。从而节约了宝贵的内存资源。另一方面，用数组的方法必须占用一块连续的内存区域。而使用动态分配时，每个结点之间可以是不连续的(结点内是连续的)。结点之间的联系可以用指针实现。即

在结点结构中定义一个成员项用来存放下一结点的首地址，这个用于存放地址的成员，常把它称为指针域。

可在第一个结点的指针域内存入第二个结点的首地址，在第二个结点的指针域内又存放第三个结点的首地址，如此串连下去直到最后一个结点。最后一个结点因无后续结点连接，其指针域可赋为0。这样一种连接方式，在数据结构中称为"链表"。

下图为最一简单链表的示意图。



图中，第0个结点称为头结点，它存放有第一个结点的首地址，它没有数据，只是一个指针变量。以下的每个结点都分为两个域，一个是数据域，存放各种实际的数据，如学号num，姓名name，性别sex和成绩score等。另一个域为指针域，存放下一结点的首地址。链表中的每一个结点都是同一种结构类型。

例如，一个存放学生学号和成绩的结点应为以下结构：

```
struct stu
{ int num;
  int score;
  struct stu *next;
}
```

前两个成员项组成数据域，后一个成员项next构成指针域，它是一个指向stu类型结构的指针变量。

链表的基本操作对链表的主要操作有以下几种：

1. 建立链表；
2. 结构的查找与输出；
3. 插入一个结点；
4. 删除一个结点；

下面通过例题来说明这些操作。

【例11.9】建立一个三个结点的链表，存放学生数据。为简单起见， 我们假定学生数据结构中只有学号和年龄两项。可编写一个建立链表的函数creat。程序如下：

```
#define NULL 0
#define TYPE struct stu
#define LEN sizeof (struct stu)
struct stu
    {
      int num;
      int age;
      struct stu *next;
    };
TYPE *creat(int n)
{
    struct stu *head,*pf,*pb;
    int i;
    for(i=0;i<n;i++)
```

```
    {
      pb=(TYPE*) malloc(LEN);
      printf("input Number and  Age\n");
      scanf("%d%d",&pb->num,&pb->age);
      if(i==0)
      pf=head=pb;
      else pf->next=pb;
      pb->next=NULL;
      pf=pb;
    }
    return(head);
}
```

在函数外首先用宏定义对三个符号常量作了定义。这里用 TYPE 表示 struct stu，用 LEN 表示 sizeof(struct stu)主要的目的是为了在以下程序内减少书写并使阅读更加方便。结构 stu 定义为外部类型，程序中的各个函数均可使用该定义。

creat 函数用于建立一个有 n 个结点的链表，它是一个指针函数，它返回的指针指向 stu 结构。在 creat 函数内定义了三个 stu 结构的指针变量。head 为头指针，pf 为指向两相邻结点的前一结点的指针变量。pb 为后一结点的指针变量。

1.1 枚举类型

在实际问题中，有些变量的取值被限定在一个有限的范围内。例如，一个星期内只有七天，一年只有十二个月，一个班每周有六门课程等等。如果把这些量说明为整型，字符型或其它类型显然是不妥当的。为此，C 语言提供了一种称为“枚举”的类型。在“枚举”类型的定义中列举出所有可能的取值，被说明为该“枚举”类型的变量取值不能超过定义的范围。应该说明的是，枚举类型是一种基本数据类型，而不是一种构造类型，因为它不能再分解为任何基本类型。

1.1.1枚举类型的定义和枚举变量的说明

1. 枚举的定义枚举类型定义的一般形式为：

   enum 枚举名{ 枚举值表 };

   在枚举值表中应罗列出所有可用值。这些值也称为枚举元素。

   例如：

该枚举名为 weekday，枚举值共有 7 个，即一周中的七天。凡被说明为 weekday 类型变量的取值只能是七天中的某一天。

2. 枚举变量的说明

   如同结构和联合一样，枚举变量也可用不同的方式说明，即先定义后说明，同时定义说明或直接说明。

   设有变量 a,b,c 被说明为上述的 weekday，可采用下述任一种方式：

   ```
   enum weekday{ sun,mou,tue,wed,thu,fri,sat };
   enum weekday a,b,c;
   ```

或者为：

```
enum weekday{ sun,mou,tue,wed,thu,fri,sat }a,b,c;
```

或者为：

```
enum { sun,mou,tue,wed,thu,fri,sat }a,b,c;
```

1.1.1枚举类型变量的赋值和使用

枚举类型在使用中有以下规定：

1. 枚举值是常量，不是变量。不能在程序中用赋值语句再对它赋值。

   例如对枚举 weekday 的元素再作以下赋值：

   ```
   sun=5;
   mon=2;
   sun=mon;
   ```

   都是错误的。

2. 枚举元素本身由系统定义了一个表示序号的数值，从 0 开始顺序定义为 0，1，2…。

   如在 weekday 中，sun 值为 0，mon 值为 1，…,sat 值为 6。

【例 11.10】

```
main(){
    enum weekday
    { sun,mon,tue,wed,thu,fri,sat } a,b,c;
    a=sun;
    b=mon;
    c=tue;
    printf("%d,%d,%d",a,b,c);
}
```

说明：

只能把枚举值赋予枚举变量，不能把元素的数值直接赋予枚举变量。如：

```
a=sum;
b=mon;
```

是正确的。而：

```
a=0;
b=1;
```

是错误的。如一定要把数值赋予枚举变量，则必须用强制类型转换。

如：

```
a=(enum weekday)2;
```

其意义是将顺序号为 2 的枚举元素赋予枚举变量 a，相当于：

```
a=tue;
```

还应该说明的是枚举元素不是字符常量也不是字符串常量，使用时不要加单、双引号。

【例 11.11】

```
main(){
    enum body
    { a,b,c,d } month[31],j;
```

```
        int i;
        j=a;
        for(i=1;i<=30;i++){
          month[i]=j;
          j++;
          if (j>d) j=a;
        }
        for(i=1;i<=30;i++){
          switch(month[i])
          {
            case a:printf(" %2d  %c\t",i,'a'); break;
            case b:printf(" %2d  %c\t",i,'b'); break;
            case c:printf(" %2d  %c\t",i,'c'); break;
            case d:printf(" %2d  %c\t",i,'d'); break;
            default:break;
          }
        }
        printf("\n");
    }
```

1.1 类型定义符 typedef

C语言不仅提供了丰富的数据类型，而且还允许由用户自己定义类型说明符，也就是说允许由用户为数据类型取“别名”。类型定义符 typedef 即可用来完成此功能。例如，有整型量 a,b,其说明如下：

```
int a,b;
```

其中 int 是整型变量的类型说明符。int 的完整写法为 integer，为了增加程序的可读性，可把整型说明符用 typedef 定义为：

```
typedef int INTEGER
```

这以后就可用 INTEGER 来代替 int 作整型变量的类型说明了。

例如：

```
INTEGER a,b;
```

它等效于：

```
int a,b;
```

用 typedef 定义数组、指针、结构等类型将带来很大的方便，不仅使程序书写简单而且使意义更为明确，因而增强了可读性。

例如：

```
typedef char NAME[20];
```

表示 NAME 是字符数组类型，数组长度为 20。然后可用 NAME 说明变量，如：

```
NAME a1,a2,s1,s2;
```

完全等效于：

```
char a1[20],a2[20],s1[20],s2[20]
```

又如：

```
typedef struct stu
{ char name[20];
  int age;
  char sex;
       } STU;
```

定义 STU 表示 stu 的结构类型，然后可用 STU 来说明结构变量：

```
STU body1,body2;
```

typedef 定义的一般形式为：

```
typedef 原类型名  新类型名
```

其中原类型名中含有定义部分，新类型名一般用大写表示，以便于区别。

有时也可用宏定义来代替 typedef 的功能，但是宏定义是由预处理完成的，而 typedef 则是在编译时完成的，后者更为灵活方便。