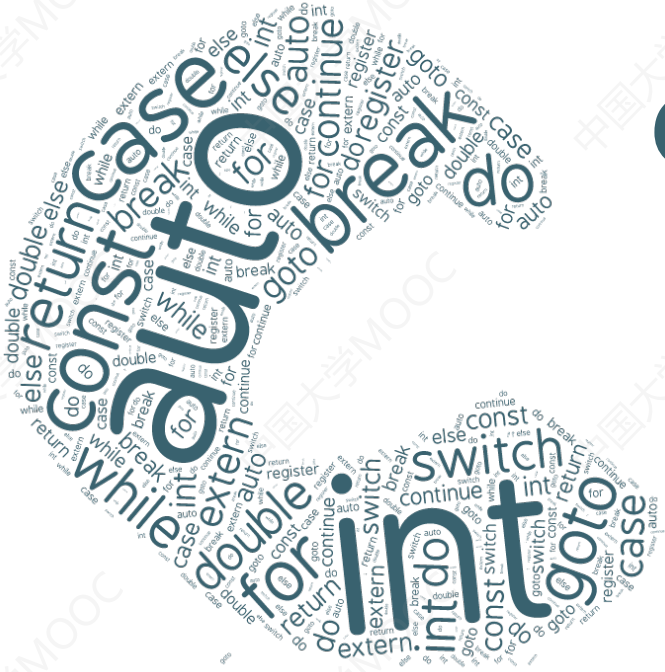


chapter 8

结构和联合

目录
content

8.1 结构的定义和使用

结构的定义

- 在标准C语言中，使用结构可以把不同类型的数据存储在一起
- 在C语言程序中先定义结构，才能进行结构变量的定义和使用
- 结构是由不同数据类型的数据组成的。组成结构的每个数据称为该结构的成员项，简称成员
- 在程序中使用结构，先要对结构的组成进行描述，称为结构的定义

```
struct 结构名
{
    数据类型    成员名1;
    数据类型    成员名2;
    ……
    数据类型    成员名n;
};
```

8.1.1 结构的定义

结构定义规则

- 结构的定义以关键字struct作为标识符，其后是定义的结构名，两者为特定结构的类型标识符。结构名命名原则与变量名相同
- 每个成员项由其数据类型和成员名组成。每个成员项后用分号“；”作为结束符。整个结构的定义也用分号作为结束符

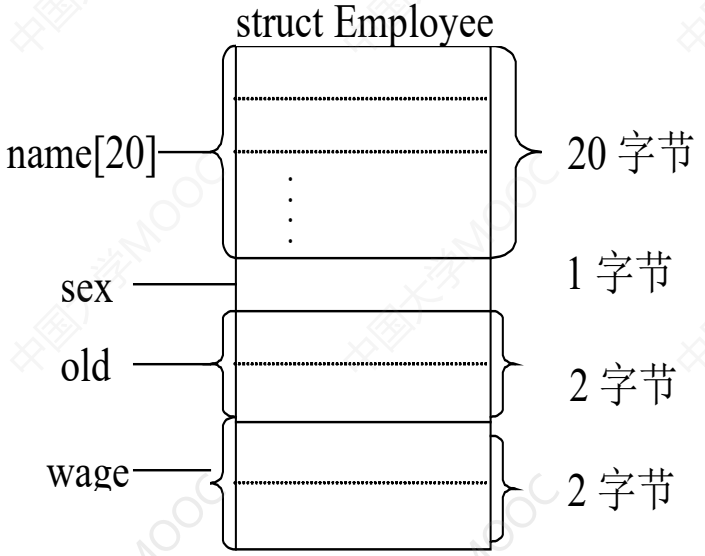
```
struct Employee
{
   char name[20];
   char sex;
   int old;
   int wage;
};
```

结构的定义

结构定义规则

- 结构的定义明确地描述了其组织形式。在程序执行时，结构的定义并不引起系统为该结构分配内存空间
- 结构Employee的定义，仅仅指定了在使用这种结构时应该按下图所示的配置情况占用内存，但这时并没有实际占用内存空间。

```
struct Employee
{
   char name[20];
   char sex;
   int old;
   int wage;
};
```



8.1.2 结构变量的定义

程序一旦定义了一个结构体，就相当于定义了一个新的结构类型，那么就可以把结构名当作像int、double等关键字一样使用，用说明语句定义该形式结构体的具体结构变量，其格式为：

```
<存储类型>  struct  结构名   结构变量名;
```

结构变量的定义在程序的数据说明部分给出：

```
struct Employee wang;
```

8.1.2 结构变量的定义

结构变量的定义将引起系统按照结构定义时制定的内存模式，为被定义的结构变量分配一定的内存空间。当多个结构变量使用结构时，它们可以在一起定义：

```
struct Employee wang, li,zhang;
```

结构变量使用内存空间，具有一定的存储类型。结构变量的存储类型概念、它的寿命、可见性及使用范围与普通变量、数组完全一致

在程序中，结构变量的定义在该结构的定义之后，对于尚未定义的结构，不能用它对任何结构进行说明

8.1.2 结构变量的定义

结构的定义和结构变量的定义也可以同时进行，被定义的结构变量直接在结构定义的大括号 } 后给出

```
struct Employee
{
   char name[20];
   char sex;
   int old;
   int wage;
} wang ,song ,zhou;
```

```
struct Employee
{
   char name[20];
   char sex;
   int old;
   int wage;
};
struct  Employee wang ,song ,zhou;
```

一个结构变量占用内存的实际大小，可以利用sizeof运算求出。

sizeof(运算量)

8.1.3 结构变量的使用形式和初始化

结构是不同数据类型的若干数据的集合体。在程序中使用结构时，一般情况下不能把结构作为一整体参加数据处理，而参加各种运算和操作的是结构的各个成员项数据。

结构的成员项用以下一般形式表示：

结构变量名.成员名

结构变量wang具有下列四个成员项：

wang.name　　wang.sex

wang.old　　wang.wage

```
struct Employee
{
   char name[20];
   char sex;
   int old;
   int wage;
} wang;
```

8.1.3 结构变量的使用形式和初始化

例8.1 结构的使用形式

```
#include <stdio.h>
//结构的定义
struct Employee
{
  char *name;
  char sex;
  int old;
  char *tel;
  char *adr;
};
void main()
{
   //结构变量的定义
   struct Employee wang,gao;
   //结构变量的成员赋值
   wang.name="wang hai";
   wang.sex='M';
   wang.old=34;
   wang.tel="010-12345678";
   wang.adr="beijing";
   gao.name="gao yang";
   gao.sex='F';
   gao.old=42;
   gao.tel="021-87654321";
   gao.adr="shanghai";
   //显示结构变量的成员内容
   printf(" name  sex  old  tel  address\n" );
   printf("                                \n");
   printf("%-14s%-4c%-4d%-10s%-20s\n",\
          wang.name,wang.sex,wang.old, \
          wang.tel,wang.adr);
   printf("%-14s%-4c%-4d%-10s%-20s\n",\
          gao.name,gao.sex,gao.old, \
          gao.tel,gao.adr);
}
```

运行结果：

```
name       sex   old      tel              adr
wang hai     m    34   010-12345678   beijing
gao yang     f    42   021-87654321   shanghai
```

8.1.3 结构变量的使用形式和初始化

结构的初始化

在结构说明的时给这个结构的每个成员赋初值

struct 结构名　结构变量={初始数据};

初始数据之间用逗号分隔，初始数据的个数与结构成员项的个数应该相同。每个初始数据必须符合与其对应的成员项的数据类型。

```
struct Employee wang={"wang hai",'M',34,"123-1111","beijing"};
```

其功能等价于

```
wang.name="wang hai";
wang.sex='M';
wang.old=34;
wang.tel="010-12345678";
wang.adr="beijing";
```

结构数组

具有相同结构的结构也可以组成数组，称它们为结构数组。说明形式：

<存储类型> struct 结构名 结构数组名[元素个数] [= {初值表}];

例如：

```
struct Employee man[3];
```

说明了结构数组man[ ]，它有三个元素man[0]、man[1]和man[2]，它们都是具有Employee结构的结构变量

- 在定义结构数组的同时可以用初始化列表给它的每个元素赋初值
- 在对结构数组进行初始化时，方括号[ ]中的元素个数可以缺省
- 结构数组也具有数组的属性，结构数组名是结构数组存储首地址

结构数组

例8.2　结构数组的使用

```
#include <stdio.h>
#define STUDENT 3
//用符号常量STUDENT表示学生人数
struct Data {          //定义一个结构
  char  name[20];     // 姓名
  short age;          // 年龄
  char  adr[30];      // 地址
  long  tel;          // 电话号码
};
void main( )
{
  struct Data man[STUDENT] = {
  //定义一个结构数组并初始化
  {"王伟", 20, "东八舍416室", 87543641},
  {"李玲", 21, "南三舍219室", 87543945},
  {"张利", 19, "东八舍419室", 87543645}};
  int i;
```

```
// 输出显示表头提示信息
printf("编号\t姓名\t年龄\t地址\t电话\n\n");
//输出结构数组的数据
for(i = 0; i < STUDENT; i++)
printf("%-d\t%-s\t%-d\t%-s\t%Ld\n", \
        i+1,man[i].name, man[i].age, \
        man[i].adr,man[i].tel);
// 每个输出项都左对齐，编号从1开始
printf("\n结构类型Data的数据长度 : \
      %d字节。\n", sizeof(struct Data));
}
```

运行结果：

编号	姓名	年龄	地　址	电　话
1	王伟	20	东八舍416室	87543641
2	李玲	21	南三舍219室	87543945
3	张利	19	东八舍419室	87543645

结构类型Data的数据长度 : 56字节

结构数组

例8.3　对候选人得票进行统计：设有4个候选人，N个人参加选举，每次输入一个得票的候选人的名字，要求最后输出个人的得票结果。

```
#include <stdio.h>
#include <string.h>
//宏定义，定义参加选举人的个数
#define N 10
//结构定义
struct person
{
    char name[20];
    int count;
};
void main()
{
  //结构数组定义及初始化
    struct person leader[4]={\
        {“wang”,0},{“zhang”,0},\
        {“zhou”,0},{"gao",0}};
    char name[20],i,j;
```

```
//模拟选举过程，循环一次代表一次选举
    for(i=0;i<N;i++)
    {
      gets(name);
      //在候选人中查找匹配的人
      for(j=0;j<4;j++)
      if(strcmp(name,leader[j].name)==0)
      {
        leader[j].count++;
        break;
       }
    }
    printf("\n");
    for(j=0;j<4;j++)
      printf(“%s:%d\n”,leader[j].name,\
              leader[j].count);
}
```

运行结果：

```
//输入
wang
wang
zhang
zhang
zhang
zhang
zhou
zhou
gao
gao
//输出
wang:2
zhang:4
zhou:2
gao:2
```

8.2.2 结构指针

指向结构的指针变量称为结构指针。结构指针的运算按地址计算规则进行，定义格式为(结构名必须是已经定义过的结构类型)：

<存储类型> struct 结构名 * 结构指针名 [= 初始地址值];

例如：

```
struct Employee  * pman;
struct Employee  * pd;
```

- 存储类型是结构指针变量本身的存储类型
- 由于结构指针是指向整个结构体而不是某个成员,因此结构指针的增（减）量运算，将跳过一个结构变量的整体，指向内存中下一个结构变量或结构数组中下一个元素，结构指针本身的（物理）地址增量值取决于它所指的结构变量的数据长度

8.2.2 结构指针

- 结构体可以嵌套，即结构体成员又是一个结构变量或结构指针

例如

```
struct student {
    char    name[20];
    short   age;
    char    adr[30];
    struct  Date  BirthDay;
    struct  Date  StudyDate;
} studt[300];
```

- 在结构嵌套中，当结构体的成员项具有由该结构类型定义的结构指针时，这种结构体称为递归结构体

```
struct  Node {
    // 数据场，节点序号
    int   num;
    // 指针场，指向下一个节点的结构指针
    struct Node * next;
};
```

8.2.2 结构指针

使用结构指针对结构成员进行引用时，有两种形式：

- 使用结构成员访问运算符“.”

(*结构指针名).成员名

运算符"."的优先级比取内容运算符"*" 高，所以需要使用圆括号

- 结构指针运算符“->”（横线加大于号）。它与前一种表示方法在意义上是完全等价的。

结构指针名->成员名

结构指针

例8.4　结构指针的使用

```
#include <stdio.h>
struct Date
//定义一个结构
{
  int month;
  int day;
  int year;
};
```

```
void main()
{
    //定义一个结构变量和结构指针变量
    struct Date  today,*date_p;
    //将结构指针指向一个结构变量
    date_p =&today;
    //采用结构指针给目标变量赋值
    date_p->month =4;
    date_p->day =15;
    date_p->year =1990;
    //采用结构指针输出目标变量的数据
    printf("Today is %d/%d/%d\n", \
        date_p->month,date_p->day , \
        date_p->year );
}
```

运行结果：
Today is 4/15/1990

8.2.2 结构指针

例8.5 结构指针运算

```
#include <stdio.h>
struct Key
{
   char *keyword;
   int keyno;
};
void main()
{
  struct Key kd[]={{“are”,123},\
        {“my”,456},{"you",789}};
  struct Key *p;
  int a;
  char chr;
  p=kd;
  a=p->keyno;
  printf("p=%d,a=%d\n",p,a);
  a=++p->keyno;
  printf("p=%d,a=%d\n",p,a);
  a=(++p)->keyno;
  printf("p=%d,a=%d\n",p,a);
  a=(p++)->keyno;
  printf("p=%d,a=%d\n",p,a);
  p=kd;
  chr=*p->keyword;
  printf("p=%d,chr=%c(adr=%d)\n",\
  p,chr,p->keyword);
  chr=*p->keyword++;
  printf("p=%d,chr=%c(adr=%d)\n",\
  p,chr,p->keyword);
  chr=(*p->keyword)++;
  printf("p=%d,chr=%c(adr=%d)\n",\
  p,chr,p->keyword);
  chr=*p++->keyword;
  printf("p=%d,chr=%c(adr=%d)\n",\
  p,chr,p->keyword);
  chr=*++p->keyword;
  printf("p=%d,chr=%c(adr=%d)\n",\
  p,chr,p->keyword);
}
```

定义结构

结构指针变量

访问成员

a=++(p->keyno)

变化后指向目标的成员

变化前指向目标的成员

chr=*(p->keyword)

chr=*((p->keyword)++)

chr=(*(p->keyword))++

变化前指向目标的成员

chr=*(++(p->keyword))

运行结果：

```
p=158,a=123
p=158,a=124
p=162,a=456
p=166,a=456
```

```
p=158,chr=a(adr=170)
p=158,chr=a(adr=171)
p=158,chr=r(adr=171)
p=162,chr=s(adr=174)
p=162,chr=y(adr=175)
```

结构指针

例8.6　指向结构数组的指针

```
#include <stdio.h>
struct student   //定义一个结构
{
  int No;
  char name[20];
  char sex;
  int age;
};
void main()
{
   //定义一个结构数组并初始化
   struct student stu[3]={
     {10101,"Li Lin",'M',18},
     {10102,"Zhang fan",'M',19},
     {10104,"Wang    Min",'F',20}};
   //定义一个结构指针变量
   struct student *p;
   printf("No.  Name  Sex  Age\n");
   for( p=stu ; p<stu+3 ; p++)
   printf("%8d%-12s%6c%4d\n",\
      p->No,p->name,p->sex,p->age);
}
```

运行结果：

No.	Name	Sex	Age
10101	Li Lin	M	18
10102	Zhang fan	M	19
10104	Wang Min	F	20

8.3 结构在函数间的传递

将一个结构体变量的值传递给另一个函数，有3种方法：

- 用结构体变量的成员作参数。用法和用普通变量作实参是一样的，属于“值传递”方式
- 用结构体变量作实参。在调用函数时，把结构作为参数传递给函数，采取的是“值传递”的方式，将结构体变量所占的内存单元的内容全部顺序传递给形参
- 用结构变量的地址或结构数组的首地址作为实参。用指向相同结构类型的结构指针作为函数的形参来接受该地址值

8.3 结构在函数间的传递

例8.7 采用值传递在函数间传递结构变量，计算雇佣天数和工资。

```
#include <stdio.h>
struct  Date {
   int   day;
   int   month;
   int   year;
   int   yearday;
   char  mon_name[4];
};
int day_of_year(struct Date  pd);
void main( )
{
   struct Date HireDate;
   float  laborage;   // 存放每天的雇佣工资
   printf("请输入每天的工资：");
   scanf("%f", & laborage);
   printf("请输入年份：");
   scanf("%d", & HireDate.year);
   printf("请输入月份：");
   scanf("%d", & HireDate.month);
   printf("请输入日：");
   scanf("%d", & HireDate.day);
   HireDate.yearday = day_of_year( HireDate);
   printf("从%d年元月1日到%d年%d月%d\
       日的雇佣期限：%d天，\n应付给你的\
       工钱: %-.2f元。\n",          \
HireDate.year, HireDate.year,HireDate.month, \
       HireDate.day, HireDate.yearday,  \
       laborage *  HireDate.yearday);
}
```

```
// 计算某年某月某日是这一年的第几天
int day_of_year(struct Date  pd)
{
   int day_tab[2][13]={  {0,31,28,31,30,31,30,31,31,30,31,30,31},
                  // 非闰年的每月天数
   {0,31,29,31,30,31,30,31,31,30,31,30,31}}; // 闰年的每月天数
   /* 为了与月份一致，列号不使用为零的下标号。
   行号为0是非闰年的每月天数，为1是闰年的每月天数 */
   int i, day, leap;        // leap为0是非闰年，为1是闰年
   day = pd.day;                    // 将当月的天数加入
   leap = pd. year % 4 == 0 &&  // 能被4整除的年份基本上
                                //是闰年
   pd. year % 100 != 0 ||   // 能被100整除的年份不是闰年
   pd. year % 400 == 0;    // 能被400整除的年份又是闰年
   // 求从元月1日算起到这一天的累加天数
   for(i = 1 ; i < pd .month ; i++)
      day += day_tab[leap][i];
   return day;
}
```

运行结果：
请输入每天的工资：38.5(CR)
请输入年份：2000(CR)
请输入月份：10(CR)
请输入日：1(CR)
从2000年元月1日到2000年10月1日
的雇佣期限：275天
应付给你的工钱：10587.50元。

例8.8 通信录的建立和显示程序(采用地址方式传递结构变量)

```
#include <stdio.h>
#define MAX  3
struct Address  //定义一个通信录的结构
{
  char name[20];
  char addr[50];
  char tel[15];
};
int input(struct Address *pt); //通信录录入函数
void display(struct Address *pt,int n);
                                //通信录显示函数
void main()
{
  struct Address man[MAX]; //定义结构数组
  int i;
  for(i=0;i<MAX;i++)          //建立通信录
  if(input(&man[i])==0)break;
  display(man,i);              //显示通信录
}
```

```
int input(struct Address *pt)
{
   printf("Name?");
   gets(pt->name);
   if(pt->name[0]=='0')return 0;
   printf("Address?");
   gets(pt->addr);
   printf("Telephone?");
    gets(pt->tel);
    return 1;                          //正常返回1
}
void display(struct Address *pt,int n)
{
  int i;
  printf("  name        address              tel\n");
  printf("-----------------------------------------\n");
  for(i=0 ; i<n ; i++,pt++)
    printf("%-15s%-30s%s\n",pt->name, \
          pt->addr,pt->tel);
}
```

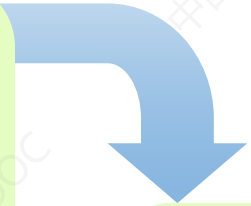
input( )函数用于以人机对话方式输入数据，它的形式参数pt是结构指针，用来接收结构地址

display( )函数显示输入结果。调用时，实参是结构数组man[ ]的首地址

8.3 结构在函数间的传递

例8.8 通信录的建立和显示程序(采用地址方式传递结构变量)

```
//输入
该程序运行结果为:
Name?王伟
Address?东八舍416室
Telephone? 87543641
Name?李玲
Address?南三舍219室
Telephone? 87543945
Name?张利
Address?东八舍419室
Telephone?87543645
```



```
//输出
name      address          tel
-----------------------------------------
王伟      东八舍416室      87543641
李玲      南三舍219室      87543945
张利      东八舍419室      87543645
```

8.3 结构在函数间的传递

例8.9 返回结构变量的函数

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct Record              //定义一个结构
{
  char str[20];
  int num;
};
//函数原型声明
struct Record str_add_int(struct Record x);
void main()
{
  struct Record p,s={“31.45”,20};
  p=str_add_int(s);
  printf("%s   %d\n",s.str,s.num);
  printf("%s   %d\n",p.str,p.num);
}
//形参x和实参s的类型一致
struct Record str_add_int(struct Record x)
{
  float e;
  e=atof(x.str);          //将字符串x.str转换为浮
                          //点数并赋给e
  e=e+x.num;              //浮点数与整数相加
  gcvt(e,5,x.str);        //将浮点数e再转换为字
                          //符串，并赋给x.str
  return x;               //将处理后的结果（结构
                          //变量）返回给调用函数
}
```

值传递，将结构变量s传递给函数，处理后再将结果通过return语句返回

运行结果：

```
31.45   20
51.45   20
```

8.3 结构在函数间的传递

例8.10 结构指针型函数

```
#include <stdio.h>
#define  NULL  0
struct Sample      //定义一个结构
{
  int num;
  char color;
  char type;
};
struct Sample *find(struct Sample *pd,int n);
void main()
{
  int num;
  struct Sample *result;  //结构指针变量定义
  struct Sample car[]={{101,'G','c'}, {210,'Y','m'},\
      {105,'R','l'},{222,'B','s'},{308,'P','b'},{0,0,0}};
  printf("Enter the number:");
  scanf("%d",&num); //输入查找样品代号
  result=find(car,num);
  if(result->num!=NULL)
  {
    printf("number :%d\n",result->num);
    printf("color :%c\n",result->color);
    printf("type :%c\n",result->type);
  }
  else
    printf("not found");//未找到，给出提示信息
}
struct Sample *find(struct Sample *pd,int n)
{
  int i;
  for(i=0 ; pd[i].num!=0 ;i++)
    if(pd[i].num==n)break;     //找到，退出循环
  if(pd[i].num!=NULL)
    return  pd+i;                  //返回查找结果
  else
    return NULL;
}
```

运行结果：

```
Enter the number:101      //输入
number:101                //输出
color:G
type:c
```

结构指针型函数是以地址传递方式向调用它的函数返回结构的数据。不仅可以返回某个结构的地址，也可以返回结构数组的地址，从而把函数中处理的若干结构的数据返回给调用它的函数中

8.4 位字段结构

计算机应用与过程控制、参数检测和数据通信领域时，要求其应用程序具有对外部设备借口硬件进行控制和管理的功能。经常使用的控制方式是向接口发送方式字或命令字，以及从接口读取状态字等。

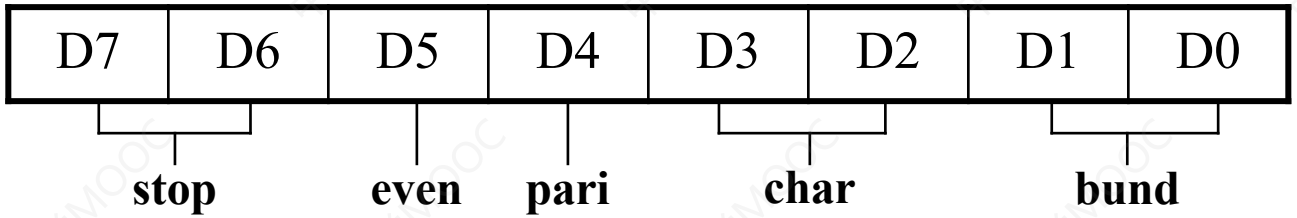
与接口有关的命令字、方式和状态字是以二进制位为单位的字段组成的数据，他们称为**位字段数据。**

位字段数据是一种压缩形式，数据整体没有具体意义。总是以组成它的位字段为处理对象。在C语言程序中，可以使用位操作（位逻辑与、或操作）对位字段进行处理。

C语言还提供了处理位字段的另一种构造类型——**位字段结构。**

8.4 位字段结构

位字段结构是一种特殊形式的结构，其成员项是二进制位字符。例如8251A使用RS-232接口进行数据通信时，方式字具的结构：



可以定义一下位字段结构：

```
struct bit
{
    unsigned bund_bit:2;
    unsigned char_bit:2;
    unsigned pari_bit:1;
    unsigned even_bit:1;
    unsigned stop_bit:2;
}
```

8.4 位字段结构

位字段结构定义中的每个成员项一般书写格式为：

unsigned 成员名：位数;

- 位字段以单个unsigned型数据为单位，其内存放着一连串相邻的二进制位
- 若某一段要从另一个字开始存放，可以使用以下形式：

```
struct test
{
    unsigned a:1;
    unsigned b:2;    //成员a和b占用一个存储单元
    unsigned :0;
    unsigned c:3;    //成员c占用另一个存储单元
}
```

在位字段结构体内若有不带名字的位字段时，冒号后面的位数应写0，表示位字段间的填充物，使得下一个成员项c分配在相邻的下一个unsigned型内

8.4 位字段结构

- 位字段结构在程序中的处理方式和表示方法等于与普通结构相同。例如，上述8251A方式字结构在定义后可以有下列说明：

```
struct bit  mod;
```

其结构变量mod的成员为：

```
mod.bund_bit
mod.stop_bit    等
```

- 一个位段必须存储在同一个存储单元中。如果一个单元空间不能容纳下一位段，则不用该空间，从另一个单元起存放该位段。位段的长度不能大于存储单元的长度，也不能定义为段数据

8.4 位字段结构

- 位字段结构的成员项可以和一个变量一样参加各种运算，但一般不能对位字段结构的成员项作取地址&运算。例如：

```
struct bit mod;
mod.bund_bit=3;        //如果赋值4就有错
mod.char_bit=2;
mod.pari_bit=0;
printf("%d,%d,%d",mod.bund_bit,mod.char_bit,m
od.pari_bit);
```

- 也可以使用%u,%o,%x等格式输出。
- 位段可以在数值表达式中引用，它会被系统自动的转换成整数型。例如: mod.bund_bit+5/mod.char_bit

8.5 联合

在C语言中，不同数据类型的数据可以**使用共同的存储区域**，这种数据结构类型称为联合体，简称联合

- 联合体在定义、说明和使用形式上与结构相似。二者本质上的不同仅在于使用内存的方式上

```
union  联合名
{
    数据类型  成员名1;
    数据类型  成员名2;
    ……
    数据类型  成员名n;
}
```

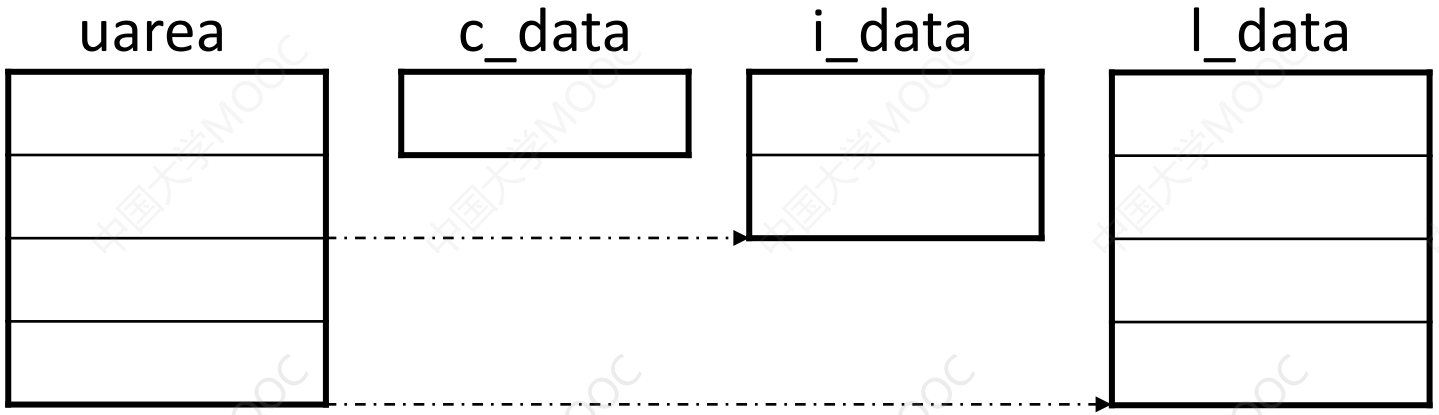
- 联合的定义制定了联合的组成形式，同时指出了组成联合的成员具体数据类型。与结构的定义相同，联合的定义并不为联合体分配具体的内存空间，它仅仅说明联合体使用内存的模式

联合

联合uarea由3个成员项组成，这3个成员项在内存中使用共同的存储空间

由于联合中各成员项的数据长度往往不同，联合体在存储时总是按照其成员中数据长度最大的成员项占用内存空间

```
union  uarea
{
    char  c_data;
    int     i_data;
    long  l_data;
}
```



8.5 联合

- 联合一经定义，就可以定义使用这种数据构造类型的具体对象，即联合变量的定义，其形式与结构变量的定义类似。

例如：

```
union uarea udata, * pud, data[10];
```

定义了使用uarea联合类型的联合变量udata、联合指针* pud和联合数组data[]。

- 由于联合体的各个成员项使用共同的内存区域，所以**联合体的内存空间中在某个时刻只能保持某个成员项的数据**
- **在程序中参加运算的必然是联合体的某个成员项**

8.5 联合

- 联合体成员项的表现形式与结构相同，用访问成员运算符.和->表示

Udata.c_data，udata.i_data，udata.l_data

pud->c_data，pud->i_data，pud->l_data

- 联合变量可以向另一个相同联合类型的联合体赋值。联合变量可以作为参数传递给函数，也可以作为返值从函数中返回
- 在程序中经常使用结构与联合体相互嵌套的形式

```
union  uarea
{
    char  c_data;
    int     i_data;
    long  l_data;
}
```

```
struct data
{
    char *p;
    int type;
    union uarea udata;
}
```

联合

例8.11　设有若干人员的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括有：姓名、号码、性别、职业、职务。现要求把它们放在同一表格中。要求先输入人员的数据，然后再输出。

- 分析：

从上面可以看出，学生和老师包含的数据是不同的。如果job项为s（学生），则第五项为class，即li是501班的。如果job项是t（教师），则第五项为position，wang是professor。显然**对第五项可以采取联合方式来处理。**

name	num	sex	job	Class/position
li	1011	f	s	501
wang	2058	m	t	professor

例8.11 输入输出学生、教师信息

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 10
union Career              //定义一个联合
{
   int myclass;
   char position[10];
};
struct Data               //定义一个结构
{
   int num;
   char name[10];
   char sex;
   char job;
   union Career category;
}
void main()
{
   int n;
   char str[20];
   struct Data person[N];//定义结构数组
   for(n=0;n<N;n++)
   {
      gets(str);
      person[n].num=atoi(str);
      gets(person[n].name);
      gets(str);
      person[n].sex=str[0];     //取字符串的第一个字符
      gets(str);
      person[n].job=str[0];     //取字符串的第一个字符
      if(person[n].job=='s')    //为学生
      {
         gets(str);
         person[n].category.myclass=atoi(str);
      }
      else if(person[n].job=='t') //为老师
         gets(person[n].category.positon);
      else                      //输入数据错误
         printf("input error");
   }
   printf("\n");
   printf("No.  Name  Sex  Job  class/position\n");
   for(n=0;n<N;n++)             //显示录入数据
   {
      if(person[b].job=='s')    //为学生
         printf("%-6d%-10s%-3c%-3c%-10d\n",\
         person[n].num,person[n].name,person[n].\
        sex,person[n].job,person[n].category.myclass);
      else if(person[n].job=='t') //为老师
         printf("%-6d%-10s%-3c%-3c%-10d\n",\
         person[n].num,person[n].name,person[n].\
        sex,person[n].job,person[n].category.position);
   }
}
```

8.6 类型定义语句typedef

- typedef：用新的类型名来代替已有的基本数据类型名和已经定义了的类型的功能。格式为：

```
typedef <类型说明> <新类型名>;
```

其中类型说明是对新类型名的描述，类型说明可以是各种基本数据类型和已定义了的结构体、联合体、指针、枚举等类型

```
typedef int     INTEGER;
typedef float   REAL;
```

```
int     i,j;
float   a,b;
```

等价于

```
INTEGER     i,j;
REAL        a,b;
```

8.6 类型定义语句typedef

- 例如，在stdio.h头文件中有如下定义：

```
typedef unsigned int size_t;
```

由于经常要用无符号整形变量记录一个内存空间的大小或者是某种数据类型的数据块大小，用新类型名”size_t”，不仅书写方便，且见名知意，在标准函数库中经常使用新类型名“size_t”

- 移植方便

```
int a,b,c;
```
修改
```
long a,b,c;
```

```
typedef int INT;
```
修改
```
typedef long INT
```

类型定义语句typedef

typedef与#define的区别

```
typedef int COUNT;
#define COUNT int
```

- 作用都是用COUNT代替int，但#define语句是在预处理操作时进行字符串的替换，而typedef则是在编译时进行处理的，它并不是进行简单的字符串替换。

```
typedef char[81]  STRING;
```

- 定义了一个STRING类型，它是具有81个字符的数组，以后就可用STRING类型定义类型的字符型数组

```
STRING text,line;
```

等价于

```
char text[81],line[81];
```

8.6 类型定义语句typedef

- 使用typedef语句给已定义的结构类型赋予新的类型名，大大简化了对结构变量的说明，例如：

```
typedef struct
{
    double real;
    double imag;
} COMPLEX;
```

其中COMPLEX就是struct{}的新类型名，即struct{}与COMPLEX对其结构变量的说明作用一样，COMPLEX也是结构名

```
COMPLEX     c1,c2;
COMPLEX     * r, * op1, * op2;
```

COMPLEX是结构类型而不是结构变量，因此不能用COMPLEX进行访问成员的运算。”COMPLEX.real”是非法的

例8.12 复数的加法运算

```
#include <stdio.h>
typedef struct
{
   double real;
   double imag;
} COMPLEX;
COMPLEX cadd(COMPLEX * op1,COMPLEX * op2);
COMPLEX cadd(COMPLEX * op1,COMPLEX * op2)
{
   COMPLEX result;
   result.real=op1->real+op2->real;   //实数部分相加
   result.imag=op1->imag+op2->imag;   //虚数部分相加
   return result;
}
void main()
{
   COMPLEX a={6.0,8.0},b={2.0,8.0},c;
   printf("(1)COMPLEX a(%.21f,%.21f)\n"),a.real,a.imag);
   printf("(2)COMPLEX b(%.21f,%.21f)\n"),b.real,b.imag);
   c=caad(&a,&b);   //调用cadd()求复数a和b之和
   printf("(3)COMPLEX c(%.21f,%.21f)\n"),c.real,c.imag);
}
```

运行结果：
(1)a(6.00,8.00)
(2)COMPLEX b(2.00,8.00)
(3)COMPLEX c(8.00,16.00)

8.7 枚举类型

枚举类型：把一组整型符号常量按顺序集合成一种数据类型

其中整型符号常量叫做该枚举类型的元素，简称枚举元素，每个枚举元素都有确定的整数值。用符号常量名来一次列举他的每一个可能值

枚举定义：

enum 枚举类型名 {枚举元素1,枚举元素2,...,枚举元素n};

```
enum WEEKDAY {sun,mon,tue,wed,thu,fri,sat};
```

```
enum COLOR {BLACK,BLUE,GREEN,RED=4,YELLOW=14,WHITE};
```

<存储类> enum 枚举类型名 枚举变量名;

```
static enum COLOR backdrop, frame;
```

8.7 枚举类型

- 枚举变量具有变量的属性，可以读取它的值和对它赋值。其存储类也有auto型、static型和extern型等
- 枚举变量只能取枚举元素表内所列的可能值。赋值时一般使用枚举元素名，避免把非枚举值赋给了枚举变量。

```
frame = RED;
backdrop=BLUE;
```

- 与结构类型一样，枚举类型的定义和枚举变量的说明即可以加上所述那样分开进行，又可以同时进行

```
enum MONTH{January=1,February,March,\
                April,May,June,July,August,\
            September,October,November,\
                        December}month;
```

8.7 枚举类型

- 枚举元素均为常量，不能用赋值语句对他们赋值

```
sun=1;     //出错
RED=2;     //出错
```

枚举元素一经初始化后，就只能使用它而不能改变

- 枚举元素作为常量，编译系统按定义式的排列顺序使它们的数值为0,1,2,...如枚举类型WEEKDAY的元素sun的值为0，mon的值为1，...sat的值为6，这称为隐式初始化操作
- 枚举元素的值也可以在定义的同时由编程者自行指定

```
printf("BLACK=%d,BLUE=%d,GREEN=%d,RED=%d,YELLOW=%d,\
        WHITE=%d\n",BLACK,BLUE,GREEN,RED,YELLOW,WHITE);
```

运行结果：
BLACK=0,BLUE=1,GREEN=2,RED=4,YELLOW=14,WHITE=15

例8.13 枚举变量用于switch结构

```
  //定义一个结构指针变量
#include <stdio.h>
void main()
{
   enum MONTH {January=1,February,March,April, \
         May,June, July,August, September, \
        October,November,  December} month;
   int year,day,days;
   printf("请输入年份：");
   scanf("%d",&year);
   printf("请输入月份：");
   scanf("%d",&month);
  printf("请输入日：");
   scanf("%d",&day);
   days=day;  //加入本月的天数
   //计算（month-1）月的累加天数，
   //该switch语句为链形结构
   switch(month-1)
   {
   case December: days+=31;
   case November: days+=30;
   case October: days+=31;
   case September: days+=30;
   case August: days+=31;
   case July: days+=31;
   case June: days+=30;
   case May: days+=31;
   case April: days+=30;
   case March: days+=31;
   case February:
      if(year % 4 ==0 && year % 100 != 0|| \
          year % 400 ==0)
         days += 29;   //闰年
      else
         days += 28;   //非闰年
   case January: days + = 31;
   }
   printf("从%d年元月1日到%d年%d月%d \
                日共有%d天！\n\n",\
               year,year,month,day,days);
}
```

运行结果：
请输入年份：2000(CR)
请输入月份：11(CR)
请输入日：11(CR)
从2000年元月1日到2000年11月11日共有316天！

8.8 综合举例——链表

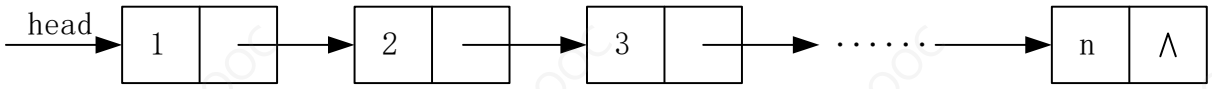
- 数组虽然结构简单、访问元素方便、执行速度快
  - 但是数组是静态数据结构，即必须要预先确定它的大小
  - 只能用顺序存储的存储结构，且要求其存储单元是连续的

- **链表是一种动态地进行存储分配的数据结构。**非常适合处理数据个数预先无法确定且数据记录频繁变化的场合，使用动态存储技术和递归结构体建立链表，可通过指针增加或删除结点（Node）

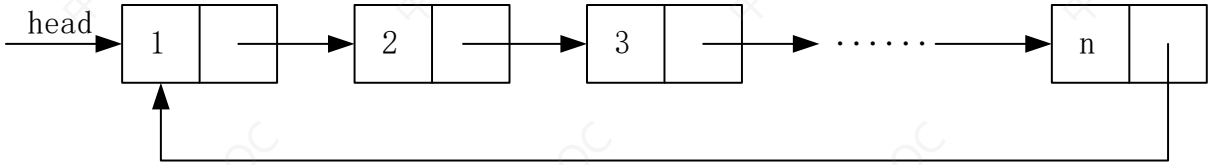
综合举例——链表

链表结构

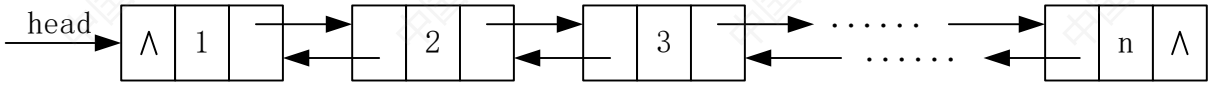
常用的链表有单向链表和双向链表，单向链表中又包含循环单向链表，双向链表也包含循环双向链表



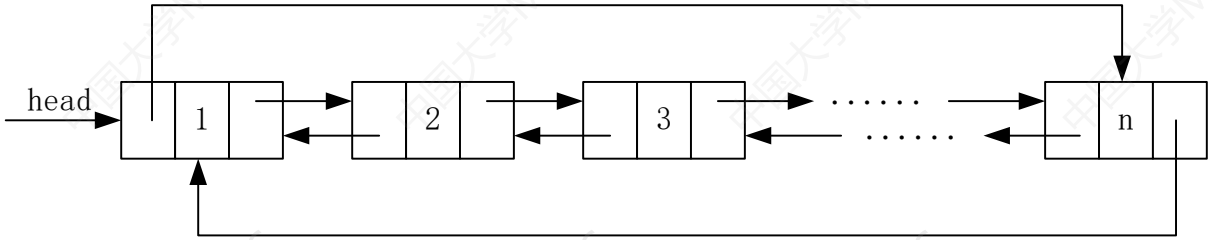
一般单向链表



循环单向链表



一般双向链表

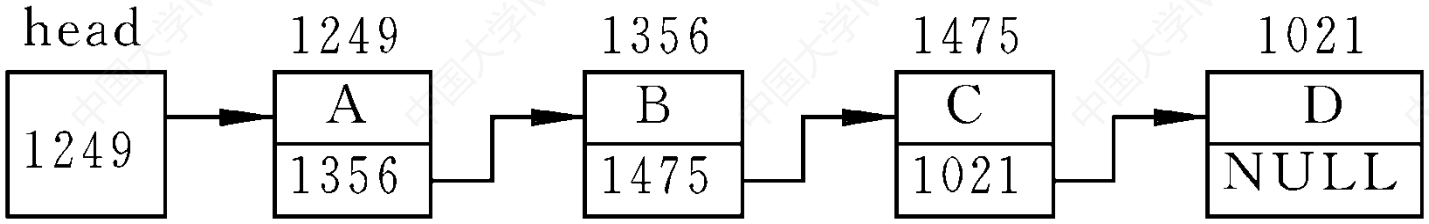


循环双向链表

综合举例——链表

单向链表的结构

- 头指针：存放一个地址，该地址指向一个元素
- 结点：用户需要的实际数据和链接下一个结点的指针



可以不连续存放

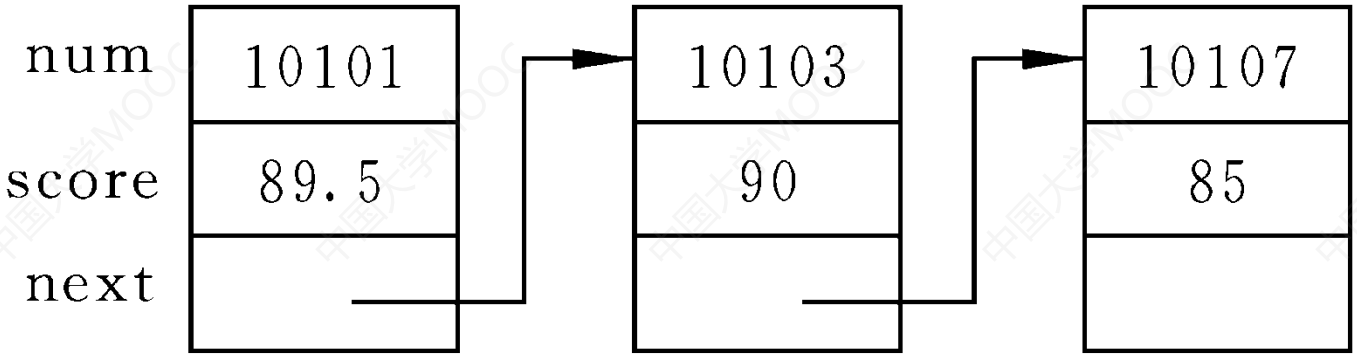
不提供“头指针”（head），则整个链表都无法访问

综合举例——链表

单向链表的结构

- 成员num和score用来存放结点中的有用数据（用户需要用到的数据）
- next是指针类型成员,指向struct student类型数据（next所在的结构体类型）

```
struct student
    {
      int num;
      float score:
      struct student * next;
};
```



8.8 综合举例——链表

简单举例

```
#include <stdio.h>
#define NULL 0
struct student
{
        long num;
        float score;
        struct student *next;
};
```

```
void main()
{
        struct student a,b,c,*head,*p;
        a. num=99101; a.score=89.5;
        b. num=99103; b.score=90;
        c. num=99107; c.score=85;
        head=&a;
        a.next=&b;
        b.next=&c;
        c.next=NULL;
        p=head;
      do
      {
              printf("%ld %5.1f\n",p->num,p->score);
              p=p->next;
        } while(p!=NULL);
}
```

运行结果：
10101 89.5
10103 90.0
10107 85.0

8.8 综合举例——链表

动态链表——动态地开辟和释放存储单元

- （1）malloc函数

```
void * malloc(unsigned int size);
```

在内存的动态存储区中分配一个长度为size的连续空间。

- （2）calloc函数

```
void * calloc(unsigned n,unsigned size);
```

在内存的动态区存储中分配n个长度为size的连续空间

- （3）free函数

```
void free(void * p);
```

释放由p指向的内存区

8.8 综合举例——链表

```
typedef  int  DataType;
typedef struct linknode
{
        DataType data;
        struct linknode *next;
} Node;
```

单向链表操作——查找结点1

```
int     node_find(Node  * head, Datatype x)
{
        Node *p;
        p=head;
        while (p!=NULL)
        {
                if (p->data ==x) break;
                p=p->next;
        }
        if(p==NULL)         return 0;
        else return 1;
}
```

单向链表操作——查找结点2

```
Node  * node_find(Node  *head, Datatype x)
{
        Node *p;
        p=head;
        while (p!=NULL)
        {
                if ( p->data == x) break;
                p = p->next ;
        }
        return p;
}
```

综合举例——链表

单向链表操作——插入结点

```
void insert (Node * p, Datatype x)
{
        Node *q, *t ;
        q=(Node *)malloc(sizeof(Node));
        if(q==NULL)
        {
                printf("no enough memory");
                exit(1);
        }
        q->data = x ;
        t = p->next ;
        p->next = q;
        q->next = t;
}
```

8.8 综合举例——应用举例

例8.14 编程处理某班N个学生4门课的成绩，它们是数学、物理、英语和计算机，按编号从小到大的顺序依次输入学生的姓名、性别和四门课的成绩。计算每个学生的平均分，并以清晰的打印格式从高分到低分的顺序打印平均分高于全班总平均成绩的男生的成绩单。

问题分析与设计：

模块划分：四个功能块：输入数据、排序，求总平均成绩和输出

数据结构：由于学生的姓名、性别、各门功课的成绩及平均成绩具有一定的关联性，作为集合数据，将其规划为一个结构体：

```
struct student
{
    char name[20];  //姓名
    char sex;       //性别，'m'代表男，'f'代表女
    float score[4]; //学生的各科成绩
    float aver;     //平均成绩
};
```

8.8 综合举例——应用举例

例8.14 编程处理某班N个学生4门课的成绩，它们是数学、物理、英语和计算机，按编号从小到大的顺序依次输入学生的姓名、性别和四门课的成绩。计算每个学生的平均分，并以清晰的打印格式从高分到低分的顺序打印平均分高于全班总平均成绩的男生的成绩单。

函数原型

(1) void input(struct student *p,int n);

功能：输入学生信息

形参：第一个参数用来接收存储学生数据的结构数组首地址，第二个参数接收学生的个数

(2) void sort(struct student *p,int n);

功能：按照学生成绩排序

形参：第一个参数用来接收存储学生数据的结构数组首地址，第二个参数接收学生的个数

(3) void output(struct student *p,int n,float aver);

功能：输出学生信息

形参：第一个参数用来接收存储学生数据的结构数组首地址，第二个参数接收学生的个数
第三个参数接收总平均成绩

(4) float average(struct student *p,int n);

功能：计算平均成绩（函数返回总平均成绩）

形参：第一个参数用来接收存储学生数据的结构数组首地址，第二个参数接收学生的个数

8.8 综合举例——应用举例

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define  N   10          //符号常量代表学生人数
struct student           //结构定义
{
  char name[20];
  char sex;
  float score[4];
  float aver;
};
void input(struct student *p,int n);
void sort(struct student *p,int n);
float average(struct student *p,int n);
void output(struct student *p,int n,float aver);
void main()
{
    struct student stu[N];  //定义结构数组，存储学生的相关信息
    float t_aver;           //总平均成绩
    input(stu,N);           //录入学生信息及成绩，计算每人的平均成绩
    sort(stu,N);            //按成绩排序
    t_aver=average(stu,N);      //计算总平均成绩
    output(stu,N,t_aver);   //输出高于总平均成绩的男生的成绩单
}
```

函数声明

函数调用

8.8 综合举例——应用举例

输入学生信息

```
void input(struct student *p,int n)
{
   int i,j;
   float per_aver;
   char str[20];
   for(i=0 ;i<n ;i++,p++)
  {
      printf("input student name:\n");
      gets(p->name); 输入姓名
      printf("input student sex:\n");
      gets(str);
      p->sex=str[0];
      printf("input student score:\n");
      for(per_aver=0,j=0 ;j<4 ;j++)
      {
        gets(str);
        p->score[j]=atof(str);
        per_aver+= p->score[j];
       }
       p->aver=per_aver/4;
  }
   return ;
}
```

8.8 综合举例——应用举例

排序函数

```
void sort(struct student *p,int n)
{
  struct student temp;  //中间变量，用于交换
  int i,j;
  for(i=0;i<n-1;i++)      //选择法排序
  for(j=i+1;j<n;j++)
  if(p[i].aver<p[j].aver)      //降序排序
  {
    temp=p[i];          //结构变量交换数据
    p[i]=p[j];
    p[j]=temp;
  }
}
```

求平均成绩函数

```
float average(struct student *p,int n)
{
  int i;
  float temp;
  for(i=0,temp=0;i<n;i++)
  temp=temp+p[i].aver;
  return temp /n;
}
```

输出成绩函数

```
void output(struct student *p,int n, float aver)
{
  int i;
  printf("Name  Sex maths physics english \
                    computer  \ average\n");
  printf("---------------------------------------- \
          ----------------------------------");
  for(i=0 ;i<n ;i++)
    if(p[i].aver>aver)   printf("%10s%10c%8.2f \
      %8.2f%8.2f%8.2f %8.2f\n", \
      p[i].name,p[i].sex, p[i].score[0],\
      p[i].score[1], p[i].score[2],\
      p[i].score[3], p[i].aver);
}
```

本章小结

本章内容是C语言数据类型的进一步扩展，特别是结构类型在软件编制中有广泛的用途

- 重点掌握结构类型的定义、结构变量的定义及初始化、结构变量成员的引用方法，结构变量的输入、输出与赋值方法
- 掌握结构变量在函数间传递的方法，对应函数定义形式和调用方法
- 掌握结构变量、结构数组、结构指针的定义形式，初始化及成员引用方法
- 熟悉联合的定义及引用、枚举变量的概念、枚举变量的定义及使用
- 掌握链表的基本操作