# 第7章　结构体与共用体

前面的课程我们学习了一些简单数据类型（整型、实型、字符型）的定义和应用，还学习了数组（一维、二维）的定义和应用，这些数据类型的特点是：当定义某一特定数据类型，就限定该类型变量的存储特性和取值范围。对简单数据类型来说，既可以定义单个的变量，也可以定义数组。而数组的全部元素都具有相同的数据类型，或者说是相同数据类型的一个集合。

在日常生活中，我们常会遇到一些需要填写的登记表，如住宿表、成绩表、通讯地址等。在这些表中，填写的数据是不能用同一种数据类型描述的，在住宿表中我们通常会登记上姓名、性别、身份证号码等项目；在通讯地址表中我们会写下姓名、邮编、邮箱地址、电话号码、E-mail等项目。这些表中集合了各种数据，无法用前面学过的任一种数据类型完全描述，因此C引入一种能集中不同数据类型于一体的数据类型 —— 结构体类型。结构体类型的变量可以拥有不同数据类型的成员，是不同数据类型成员的集合。

## 7.1　结构体类型变量的定义和引用

在上面描述的各种登记表中，让我们仔细观察一下住宿表、成绩表、通讯地址等。

住宿表由下面的项目构成：

| 姓　名<br>（字符串） | 性　别<br>（字符） | 职　业<br>（字符串） | 年　龄<br>（整型） | 身份证号码<br>（长整型或字符串） |
|---|---|---|---|---|

成绩表由下面的项目构成：

| 班　级<br>（字符串） | 学　号<br>（长整型） | 姓　名<br>（字符串） | 操作系统<br>（实型） | 数据结构<br>（实型） | 计算机网络<br>（实型） |
|---|---|---|---|---|---|

通讯地址表由下面的项目构成：

| 姓　名<br>（字符串） | 工作单位<br>（字符串） | 家庭住址<br>（字符串） | 邮　编<br>（长整型） | 电话号码<br>（字符串或长整型） | E-mail<br>（字符串） |
|---|---|---|---|---|---|

这些登记表用C提供的结构体类型描述如下：

住宿表:

```
struct accommod
{
char  name[20];  /*姓名*/
char  sex;       /*性别*/
char job[40];    /*职业*/
int   age;       /*年龄*/
long  number;    /*身份证号码*/
```

```
    };
```

**成绩表**:

```
struct score
{
    char  grade[20];    /*班级*/
    long  number;       /*学号*/
    char name[20];      /*姓名*/
    float os;           /*操作系统*/
    float  datastru;    /*数据结构*/
    float  compnet;     /*计算机网络*/
    };
```

**通讯地址表**:

```
struct addr
{
  char  name[20];
  char  department[30];/*部门*/
  char  address[30];   /*住址*/
  long   box;          /*邮编*/
  long   phone;        /*电话号码*/
  char   email[30];           /*Email*/
};
```

这一系列对不同登记表的数据结构的描述类型称为结构体类型。由于不同的问题有不同的数据成员，也就是说有不同描述的结构体类型。我们也可以理解为结构体类型根据所针对的问题其成员是不同的，可以有任意多的结构体类型描述。

下面给出C对结构体类型的定义形式：

struct 结构体名
{
成员项表列
};

有了结构体类型，我们就可以定义结构体类型变量，以对不同变量的各成员进行引用。

### 7.1.1 结构体类型变量的定义

结构体类型变量的定义与其它类型的变量的定义是一样的，但由于结构体类型需要针对问题事先自行定义，所以结构体类型变量的定义形式就增加了灵活性，共计有三种形式，分别介绍如下：

1) 先定义结构体类型，再定义结构体类型变量：

```
struct stu                      /*定义学生结构体类型*/
{
  char name[20];                /*学生姓名*/
  char  sex;                    /*性别*/
  long  num;                    /*学号*/
  float  score[3];              /*三科考试成绩*/
};
struct stu  student1,student2;/*定义结构体类型变量*/
```

```
struct stu student3,student4;
```

用此结构体类型，可以定义更多的该结构体类型变量。

2) 定义结构体类型同时定义结构体类型变量：

```
struct data
{
  int day;
  int  month;
   int year;
  }  time1,time2;
```

也可以再定义如下变量：

```
 struct data time3,time4;
```

用此结构体类型，同样可以定义更多的该结构体类型变量。

3) 直接定义结构体类型变量：

```
 struct
 {
  char name[20];          /*学生姓名*/
  char  sex;              /*性别*/
  long  num;              /*学号*/
  float  score[3];        /*三科考试成绩*/
} person1,person2;        /*定义该结构体类型变量*/
```

该定义方法由于无法记录该结构体类型，所以除直接定义外，不能再定义该结构体类型变量。

### 7.1.2 结构体类型变量的引用

学习了怎样定义结构体类型和结构体类型变量，怎样正确地引用该结构体类型变量的成员呢？C 规定引用的形式为：

<结构体类型变量名>.<成员名>

若我们定义的结构体类型及变量如下：

```
 struct data
{
   int day;
   int  month;
   int year;
  }  time1,time2;
```

则变量time1和time2各成员的引用形式为：time1.day、time1.month、time1.year及time2.day、time2.month、time2.year，如图7-1所示。

其结构体类型变量的各成员与相应的简单类型变量使用方法完全相同。

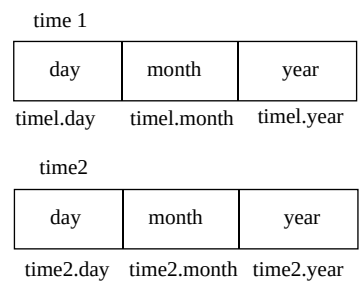



图7-1 结构体类型示例中变量各成员的引用形式

### 7.1.3 结构体类型变量的初始化

由于结构体类型变量汇集了各类不同数据类型的成员，所以结构体类型变量的初始化就

略显复杂。

结构体类型变量的定义和初始化为：

```
struct stu                  /*定义学生结构体类型*/
{
  char name[20];            /*学生姓名*/
  char  sex;                /*性别*/
  long  num;                /*学号*/
  float  score[3];          /*三科考试成绩*/
};
struct stu  student={"liping",'f',970541,98.5,97.4,95};
```

上述对结构体类型变量的三种定义形式均可在定义时初始化。结构体类型变量完成初始化后，即各成员的值分别为：student.name="liping"、student.sex='f'、student.num=970541、student.score[0]=98.5、student.score[1]=97.4、student.score[2]=95。其存储在内存的情况如图7-2所示。

| | | | | | |
|---|---|---|---|---|---|
| Liping | f | 970541 | 98.5 | 97.4 | 95 |

图7-2 结构体类型变量在内存中的存储

我们也可以通过C提供的输入输出函数完成对结构体类型变量成员的输入输出。由于结构体类型变量成员的数据类型通常是不一样的，所以要将结构体类型变量成员以字符串的形式输入，利用C的类型转换函数将其转换为所需类型。类型转换的函数是：

int atoi( char *str)；转换str所指向的字符串为整型，其函数的返回值为整型。

double atof(char *str)；转换str所指向的字符串为实型，其函数的返回值为双精度的实型。

long atol(char *str)；转换str所指向的字符串为长整型，其函数的返回值为长整型。

使用上述函数，要包含头文件"stdlib.h"。

对上述的结构体类型变量成员输入采用的一般形式：

```
 char  temp[20];
 gets(student.name);          /*输入姓名*/
 student.sex=getchar();       /*输入性别*/
 gets(temp);                  /*输入学号*/
 student.num=atol(temp);      /*转换为长整型*/
 for(i=0;i<3;i++)             /*输入三科成绩*/
{
   gets(temp);
   student.score[i]=atoi(temp);
}
```

对该结构体类型变量成员的输出也必须采用各成员独立输出，而不能将结构体类型变量以整体的形式输入输出。

C允许针对具体问题定义各种各样的结构体类型，甚至是嵌套的结构体类型。

```
 struct data
{
   int day;
```

```
    int mouth;
    int year;
};
  struct  stu
{
    char name[20];
    struct data birthday;   出生年月，嵌套的结构体类型*/
    long num;
} person;
```

该结构体类型变量成员的引用形式：person.name 、person.birthday.day、person. birthday. month、person. birthday.year、person.num 。

## 7.2 结构体数组的定义和引用

单个的结构体类型变量在解决实际问题时作用不大，一般是以结构体类型数组的形式出现。结构体类型数组的定义形式为：

```
 struct stu              /*定义学生结构体类型*/
{
  char name[20];         /*学生姓名*/
  char  sex;             /*性别*/
  long  num;             /*学号*/
  float  score[3];       /*三科考试成绩*/
};
 struct stu stud[20];    定义结构体类型数组stud ，*/
                         /*该数组有20个结构体类型元素*/
```

其数组元素各成员的引用形式为：

```
stud[0].name 、stud[0].sex 、stud[0].score[i];
stud[1].name 、stud[1].sex 、stud[1].score[i];
...
...
stud[19].name 、stud[19].sex 、stud[19].score[i];
```

[例7-1] 设某组有4 个人，填写如下的登记表，除姓名、学号外，还有三科成绩，编程实现对表格的计算，求解出每个人的三科平均成绩，求出四个学生的单科平均，并按平均成绩由高分到低分输出。

| Number | Name | English | Mathema | Physics | Average |
|---|---|---|---|---|---|
| 1 | Liping | 78 | 98 | 76 | |
| 2 | Wangling | 66 | 90 | 86 | |
| 3 | Jiangbo | 89 | 70 | 76 | |
| 4 | Yangming | 90 | 100 | 67 | |

题目要求的问题多，采用模块化编程方式，将问题进行分解如下：

1) 结构体类型数组的输入。

2) 求解各学生的三科平均成绩。

3) 按学生的平均成绩排序。

4) 按表格要求输出。

5) 求解组内学生单科平均成绩并输出。

6) 定义main()函数，调用各子程序。

第一步，根据具体情况定义结构体类型。

```
struct stu
{
 char name[20];  /姓名*/
 long number;    /学号*/
 float score[4]; /*数组依此存放English 、Mathema 、Physics ，及Average*/

 };
```

由于该结构体类型会提供给每个子程序使用，是共用的，所以将其定义为外部的结构体类型，放在程序的最前面。

第二步，定义结构体类型数组的输入模块。

```
  void input(arr,n)  /输入结构体类型数组arr 的n个元素*/
  struct stu arr[];
  int n;
  { int i,j;
      char temp[30];
    for (i=0;i<n;i++)
    {
        printf("\ninput name,number,English,mathema,physic\n"); /*打印提示信息*/
        gets(arr[i].name);   输入姓名*/
        gets(temp);          输入学号*/
        arr[i].number=atol(temp);
        for(j=0;j<3;j++)
                {
                 gets(temp); /输入三科成绩*/
                 arr[i].score[j]=atoi(temp);
                 };
        }
  }
```

第三步，求解各学生的三科平均成绩。

在结构体类型数组中第i个元素arr[i]的成员score的前三个元素为已知，第四个Average需计算得到。

```
void aver(arr,n)
 struct stu arr[];
 int n;
 {
  int i,j;
  for(i=0;i<n;i++) /*个学生*/
  {
     arr[i].score[3]=0;
     for(j=0;j<3;j++)
      arr[i].score[3]=arr[i].score[3]+arr[i].score[j];求和*/
```

```
    arr[i].score[3]=arr[i].score[3] /3;  平均成绩*/
    }
  }
```

**第四步，按平均成绩排序，排序算法采用冒泡法。**

```
 void order(arr,n)
 struct stu arr[];
 int n;
 { struct stu temp;
  int i,j,x,y;
  for(i=0;i<n-1;i++)
  for(j=0;j<n-1-i;j++)
   if (arr[j].score[3]>arr[j+1].score[3])
   { temp=arr[j];       结构体类型变量不允许以整体输入或输出，但允许相互赋值 */
     arr[j]=arr[j+1];  /进行交换*/
     arr[j+1]=temp;
     }
   }
```

**第五步，按表格要求输出。**

```
void output(arr,n)  /以表格形式输出有n个元素的结构体类型数组各成员*/
 int n;
 struct stu arr[];
 {int i,j;
  printf("********************TABLE********************\n"打印表头*/
  printf("-----------------------------------------------------\n");
   /*输出一条水平线*/
printf("|%10s|%8s|%7s|%7s|%7s|%7s|\n","Name","Number","English","Mathema",
"physics","average");
   /*输出效果为：| Name| Number|English|Mathema|Physics|Average|*/
  printf("-----------------------------------------------\n");
  for (i=0;i<n;i++)
  {
   printf("|%10s|%8ld|",arr[i].name,arr[i].number);/*        输出姓名、学号*/
   for(j=0;j<4;j++)
   printf("%7.2f|",arr[i].score[j]);/*      输出三科成绩及三科的平均*/
   printf("\n");
   printf("-----------------------------------------------\n");
   }
}
```

**第六步，求解组内学生单科平均成绩并输出。在输出表格的最后一行，输出单科平均成绩及总平均。**

```
void out_row(arr,n)    /*对n个元素的结构体类型数组求单项平均*/
 int n;
 struct stu arr[];
 {
 float row[4]={0,0,0,0};/*定义存放单项平均的一维数组*/
 int i,j;
```

```
 for(i=0;i<4;i++)
  {
   for(j=0;j<n;j++)
   row[i]=row[i]+arr[j].score[i];/*      计算单项总和*/
   row[i]=row[i]/n;                     计算单项平均*/
  }
 printf("|%19c|",' ');                  按表格形式输出*/
 for (i=0;i<4;i++)
   printf("%7.2f|",row[i]);
 printf("\n-----------------------------------------\n");
   }
```

**第七步，定义main()函数，列出完整的程序清单。**

```
#include <stdlib.h>
#include <stdio.h>

struct stu
{
 char name[20];
 long number;
 float score[4];
 };
 main()
 {
  void input();          /*函数声明*/
  void aver();
  void order();
  void output();
  void out_row();
  struct stu stud[4];    /*定义结构体数组*/
  float row[3];
  input(stud,4);         /*依此调用自定义函数*/
  aver(stud,4);
  order(stud,4);
  output(stud,4);
  out_row(stud,4);
  }
 /*************************/
  void input(arr,n)
  struct stu arr[];
  int n;
  { int i,j;
   char temp[30];
  for (i=0;i<n;i++)
    {
     printf("\nInput Name,Number,English,Mathema,Physic\n");
     gets(arr[i].name);
     gets(temp);
     arr[i].number=atol(temp);
```

```
    for(j=0;j<3;j++)
     {
      gets(temp);
      arr[i].score[j]=atoi(temp);
      };
    }
 }
 /**********************/
void aver(arr,n)
struct stu arr[];
int n;
{
 int i,j;
 for(i=0;i<n;i++)
 {
 arr[i].score[3]=0;
 for(j=0;j<3;j++)
  arr[i].score[3]=arr[i].score[3]+arr[i].score[j];
  arr[i].score[3]=arr[i].score[3] /3;
  }
}
/**********************/
void order(arr,n)
struct stu arr[];
int n;
{ struct stu temp;
 int i,j,x,y;
 for(i=0;i<n-1;i++)
 for(j=0;j<n-1-i;j++)
  if (arr[j].score[3]>arr[j+1].score[3])
  { temp=arr[j];
    arr[j]=arr[j+1];
    arr[j+1]=temp;
  }
}
/*****************/
void output(arr,n)
int n;
struct stu arr[];
{int i,j;
 printf("********************TABLE********************\n");
 printf("-----------------------------------------------------\n");
 printf("|%10s|%8s|%7s|%7s|%7s|%7s|\n","Name","Number","English","mathema",
 "physics","average");
 printf("-----------------------------------------------------\n");
 for (i=0;i<n;i++)
 {
  printf("|%10s|%8ld|",arr[i].name,arr[i].number);
```

```
    for(j=0;j<4;j++)
    printf("%7.2f|",arr[i].score[j]);
    printf("\n");
    printf("-------------------------------------------------\n");
    }
 }
 /**********************************/
 void out_row(arr,n)
 int n;
 struct stu arr[];
 {
 float row[4]={0,0,0,0};
 int i,j;
 for(i=0;i<4;i++)
  {
    for(j=0;j<n;j++)
    row[i]=row[i]+arr[j].score[i];
    row[i]=row[i]/n;
  }
 printf("|%19c|",' ');
 for (i=0;i<4;i++)
    printf("%7.2f|",row[i]);
 printf("\n---------------------------------------\n");
}
```

**运行程序：**

```
   RUN
   Input Name,Number,English,Mathema,Physic
   Liping
   1
   78
   98
   76
   Input Name,Number,English,Mathema,Physic
   Wangling
   2
   66
   90
   86
   Input Name,Number,English,Mathema,Physic
   Jiangbo
   3
   89
   70
   76
   Input Name,Number,English,Mathema,Physic
   Yangming
   4
   90
   100
```

67

```
********************TABLE*********************
-----------------------------------------------------
|    Number|     Name|English| Mathema| Physics| Average|
-----------------------------------------------------
|  Yangming|        4|   90.00|  100.00|    67.00|   85.67|
-----------------------------------------------------
|    Liping|        1|   78.00|   98.00|    76.00|    84.00|
-----------------------------------------------------
|  Wangling|        2|   66.00|   90.00|    86.00|    80.72|
-----------------------------------------------------
|   Jiangbo|        3|   89.00|   70.00|    76.00|    78.33|
-----------------------------------------------------
|                     |   80.75|   89.50|    76.25|    82.18|
-----------------------------------------------------
```

程序中要谨慎处理以数组名作函数的参数。由于数组名作为数组的首地址，在形参和实参结合时，传递给子程序的就是数组的首地址。形参数组的大小最好不定义，以表示与调用函数的数组保持一致。在定义的结构体内，成员 score[3]用于表示计算的平均成绩，也是我们用于排序的依据。我们无法用数组元素进行相互比较，而只能用数组元素的成员 score[3]进行比较。在需要交换的时候，用数组元素的整体包括姓名、学号、三科成绩及平均成绩进行交换。在程序 order（）函数中，比较采用： arr[j].score[3]>arr[j+1].score[3]，而交换则采用：arr[j]↔arr[j+1]

## 7.3 结构体指针的定义和引用

指针变量非常灵活方便，可以指向任一类型的变量，若定义指针变量指向结构体类型变量，则可以通过指针来引用结构体类型变量。

### 7.3.1 指向结构体类型变量的使用

首先让我们定义结构体：

```
 struct stu
{
 char name[20];
 long number;
 float score[4];
 };
```

再定义指向结构体类型变量的指针变量：

```
struct stu *p1, *p2 ;
```

定义指针变量p1、p2，分别指向结构体类型变量。引用形式为：指针变量→成员；

[例7-2] 对指向结构体类型变量的正确使用。输入一个结构体类型变量的成员，并输出。

```
  #include <stdlib.h>     /*使用malloc() 需要*/
 struct data             /*定义结构体*/
 {
```

```
    int day,month,year;
    };
struct stu    /*定义结构体*/
 {
    char name[20];
    long num;
    struct data birthday;     嵌套的结构体类型成员*/

    } ;
main()   /*定义main() 函数*/
{
  struct stu *student;        定义结构体类型指针*/
  student=malloc(sizeof(struct stu));/*为指针变量分配安全的地址*/
  printf("Input name,number,year,month,day:\n");
  scanf("%s",student->name); 输入学生姓名、学号、出生年月日*/
  scanf("%ld",&student->num);
  scanf("%d%d%d",&student->birthday.year,&student->birthday.month,
                          &student->birthday.day);
  printf("\nOutput name,number,year,month,day\n" );
  /*打印输出各成员项的值*/
  printf("%20s%10ld%10d//%d//%d\n",student->name,student->num,
          student->birthday.year,student->birthday.month,
          student->birthday.day);
  }
```

程序中使用结构体类型指针引用结构体变量的成员，需要通过 C提供的函数malloc()来为指针分配安全的地址。函数 sizeof()返回值是计算给定数据类型所占内存的字节数。指针所指各成员形式为：

```
student->name
student->num
student->birthday.year
student->birthday.month
student->birthday.day
```

运行程序：

```
RUN
Input name,number,year,month,day:
Wangjian  34   1987  5  23
Wangjian  34   1987//5//23
```

### 7.3.2 指向结构体类型数组的指针的使用

定义一个结构体类型数组，其数组名是数组的首地址，这一点前面的课程介绍得很清楚。定义结构体类型的指针，既可以指向数组的元素，也可以指向数组，在使用时要加以区分。

[例7-3] 在例7-2中定义了结构体类型，根据此类型再定义结构体数组及指向结构体类型的指针。

```
struct data
```

```
    {
      int day,month,year;
    };
  struct stu    /*定义结构体*/
    {
      char name[20];
      long num;
      struct data birthday;    /*嵌套的结构体类型成员*/

      } ;
  struct stu  student[4],*p;  /*定义结构体数组及指向结构体类型的指针*/
```

作 p=student，此时指针 p就指向了结构体数组 student。

p是指向一维结构体数组的指针，对数组元素的引用可采用三种方法。

1) 地址法

student+i和p+i均表示数组第i个元素的地址，数组元素各成员的引用形式为：

（student+i）->name、(student+i)->num和(p+i)->name、（p+i）->num等。student+i和p+i与&student[i]意义相同。

2) 指针法

若p指向数组的某一个元素，则 p++就指向其后续元素。

3) 指针的数组表示法

若p=student，我们说指针 p指向数组 student，p[i]表示数组的第 i个元素，其效果与student[i]等同。对数组成员的引用描述为 :p[i].name、p[i].num等。

[例7-4] 指向结构体数组的指针变量的使用。

```
   struct data  /*定义结构体类型*/
    {
      int day,month,year;
    };
  struct stu    /*定义结构体类型*/
    {
      char name[20];
      long num;
      struct data birthday;
    } ;
  main()
  { int i;
    struct stu *p,student[4]={{"liying",1,1978,5,23},{"wangping",2,1979,3,14},
    {"libo",3,1980,5,6},{"xuyan",4,1980,4,21}};
    /*定义结构体数组并初始化*/
  p=student;            /*将数组的首地址赋值给指针p,p 指向了一维数组student*/
    printf("\n1----Output name,number,year,month,day\n" );
    for(i=0;i<4;i++)    /*采用指针法输出数组元素的各成员*/
      printf("%20s%10ld%10d//%d//%d\n",(p+i)->name,(p+i)->num,
             (p+i)->birthday.year,(p+i)->birthday.month,
             (p+i)->birthday.day);
```

```
 printf("\n2----Output name,number,year,month,day\n" );
   for(i=0;i<4;i++,p++)     /* 采用指针法输出数组元素的各成员 */
    printf("%20s%10ld%10d//%d//%d\n",p->name,p->num,
           p->birthday.year,p->birthday.month,
           p->birthday.day);

printf("\n3-----Output name,number,year,month,day\n" );
for(i=0;i<4;i++)    /* 采用地址法输出数组元素的各成员 */
    printf("%20s%10ld%10d//%d//%d\n",(student+i)->name,(student+i)->num,
           (student+i)->birthday.year,(student+i)->birthday.month,
           (student+i)->birthday.day);
p=student;
printf("\n4-----Output name,number,year,month,day\n" );
  for(i=0;i<4;i++)          /* 采用指针的数组描述法输出数组元素的各成员 */
    printf("%20s%10ld%10d//%d//%d\n",p[i].name,p[i].num,
           p[i].birthday.year,p[i].birthday.month,
           p[i].birthday.day);
  }
```

**运行程序：**

```
   RUN
1----Output name,number,year,month,day
  liying   1      1978//5//23
wangping   2      1979//3//14
    libo   3      1980//5//6
   xuyan   4      1980//4//21

2----Output name,number,year,month,day
 liying    1      1978//5//23
wangping   2      1979//3//14
    libo   3      1980//5//6
   xuyan   4      1980//4//21

3----Output name,number,year,month,day
 liying    1      1978//5//23
wangping   2      1979//3//14
    libo   3      1980//5//6
   xuyan   4      1980//4//21

4----Output name,number,year,month,day
 liying    1      1978//5//23
wangping   2      1979//3//14
    libo   3      1980//5//6
   xuyan   4      1980//4//21
```

**对二维或多维数组的指针，有兴趣的同学可课后讨论，总结出来。**

## 7.4 链表的建立、插入和删除

**数组作为存放同类数据的集合，给我们在程序设计时带来很多的方便，增加了灵活性。**

但数组也同样存在一些弊病。如数组的大小在定义时要事先规定，不能在程序中进行调整，这样一来，在程序设计中针对不同问题有时需要 30个大小的数组，有时需要50个数组的大小，难于统一。我们只能够根据可能的最大需求来定义数组，常常会造成一定存储空间的浪费。

我们希望构造动态的数组，随时可以调整数组的大小，以满足不同问题的需要。链表就是我们需要的动态数组。它是在程序的执行过程中根据需要有数据存储就向系统要求申请存储空间，决不构成对存储区的浪费。

链表是一种复杂的数据结构，其数据之间的相互关系使链表分成三种：单链表、循环链表、双向链表，下面将逐一介绍。

### 7.4.1 单链表

图7-3是单链表的结构。

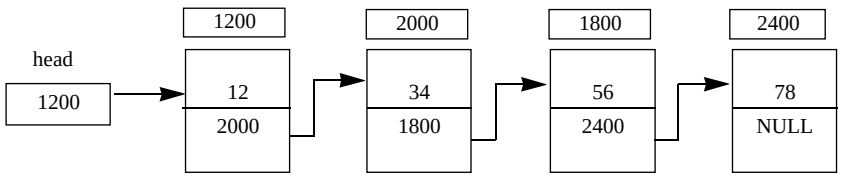



图7-3 单链表

单链表有一个头节点 head，指向链表在内存的首地址。链表中的每一个节点的数据类型为结构体类型，节点有两个成员：整型成员（实际需要保存的数据）和指向下一个结构体类型节点的指针即下一个节点的地址（事实上，此单链表是用于存放整型数据的动态数组）。链表按此结构对各节点的访问需从链表的头找起，后续节点的地址由当前节点给出。无论在表中访问那一个节点，都需要从链表的头开始，顺序向后查找。链表的尾节点由于无后续节点，其指针域为空，写作为 NULL。

图7-3还给出这样一层含义，链表中的各节点在内存的存储地址不是连续的，其各节点的地址是在需要时向系统申请分配的，系统根据内存的当前情况，既可以连续分配地址，也可以跳跃式分配地址。

看一下链表节点的数据结构定义：

```
 struct node
{
 int num;
 struct node *p;
};
```

在链表节点的定义中，除一个整型的成员外，成员 p是指向与节点类型完全相同的指针。在链表节点的数据结构中，非常特殊的一点就是结构体内的指针域的数据类型使用了未定义成功的数据类型。这是在C中唯一规定可以先使用后定义的数据结构。

• 单链表的创建过程有以下几步：

1) 定义链表的数据结构。

2) 创建一个空表。

3) 利用malloc()函数向系统申请分配一个节点。

4) 将新节点的指针成员赋值为空。若是空表，将新节点连接到表头；若是非空表，将新节点接到表尾。

5) 判断一下是否有后续节点要接入链表，若有转到3)，否则结束。

• 单链表的输出过程有以下几步

1) 找到表头。

2) 若是非空表，输出节点的值成员，是空表则退出。

3) 跟踪链表的增长，即找到下一个节点的地址。

4) 转到2)。

[例7-5] 创建一个存放正整数（输入-999做结束标志）的单链表，并打印输出。

```
#include <stdlib.h>    包含malloc() 的头文件*/
#include <stdio.h>
struct node      /链表节点的结构*/
{
     int num;
     struct node *next;
};
   main()
   {
     struct node *creat();     /*函数声明*/
     void print();
      struct node *head;       /*定义头指针*/
      head=NULL;               /*建一个空表*/
     head=creat(head);         /*创建单链表*/
     print(head);              /*打印单链表*/
   }
/****************************************/
  struct node *creat(struct node *head) 函数返回的是与节点相同类型的指针*/
   {
     struct  node *p1,*p2;
     p1=p2=(struct node*) malloc(sizeof(struct node));申请新节点*/
     scanf("%d",&p1->num);         /*输入节点的值*/
     p1->next=NULL;                /*将新节点的指针置为空*/
    while(p1->num>0)               /*输入节点的数值大于0*/
     {
            if (head==NULL) head=p1; /*空表，接入表头*/
             else p2->next=p1;       /*非空表，接到表尾*/
             p2=p1;
             p1=(struct node *)malloc(sizeof(struct node));申请下一个新节点*/
             scanf("%d",&p1->num);         /*输入节点的值*/
     }
             return head;  /返回链表的头指针*/
  }
/****************************************/
  void print(struct node *head)  输出以head 为头的链表各节点的值*/
  {
```

```
    struct node *temp;
    temp=head;                          /* 取得链表的头指针 */
    while (temp!=NULL)                  /* 只要是非空表 */
    {
     printf("%6d",temp->num);           /* 输出链表节点的值 */
     temp=temp->next;                   /* 跟踪链表增长 */
     }
}
```

在链表的创建过程中，链表的头指针是非常重要的参数。因为对链表的输出和查找都要从链表的头开始，所以链表创建成功后，要返回一个链表头节点的地址，即头指针。

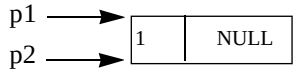
运行程序：RUN

```
  1    2    3    4    5    6    7    -999
  1    2    3    4    5    6    7
```

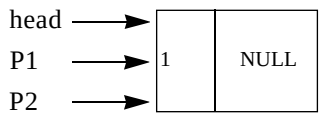
链表的创建过程用图示如下：

第一步，创建空表： head ⟶ NULL

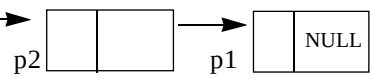
第二步，申请新节点：




第三步，若是空表，将新节点接到表头：

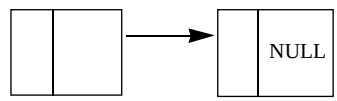



若是非空表，head ⟶ ... ⟶

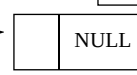



P2->next=p1。

第四步，p2=p1: head ⟶ ⟶




第五步，申请新节点：p1 ⟶




若数值为负，则结束；否则转到第三步。

### 7.4.2 单链表的插入与删除

在链表这种特殊的数据结构中，链表的长短需要根据具体情况来设定，当需要保存数据时向系统申请存储空间，并将数据接入链表中。对链表而言，表中的数据可以依此接到表尾或连结到表头，也可以视情况插入表中；对不再需要的数据，将其从表中删除并释放其所占空间，但不能破坏链表的结构。这就是下面将介绍的链表的插入与删除。

1. 链表的删除

在链表中删除一个节点，用图 7-4描述如下：

[例7-6] 创建一个学生学号及姓名的单链表，即节点包括学生学号、姓名及指向下一个节点的指针，链表按学生的学号排列。再从键盘输入某一学生姓名，将其从链表中删除。

首先定义链表的结构：

```
  struct
```