第 7 章 结构体

（视频讲解：3 小时）

有时需要将不同类型的数据组合成一个有机的整体，以便于引用。如：一个学生有学号/姓名/性别/年龄/地址等属性，如果针对学生的学号，姓名，年龄等都单独定义一个变量，那么当我们有很多个学生时，变量就难以分清，因此 C 语言提供结构体来管理不同类型的数据组合。通过本章，你将掌握：

- 结构体的使用
- 链表的增删查改
- 共用体与枚举的使用

7.1 结构体与结构体指针

7.1.1 结构体的定义-引用-初始化

声明一个结构体类型的一般形式为：

struct 结构体名

{成员表列}；

如：struct student

{

int num;char name[20];char sex;

int age;float score;char addr[30];

};

先声明结构体类型再定义变量名，例如：struct student student1, student2;

接下来来看【例 7.1.1-1】实例：

【例 7.1.1-1】结构体的 scanf 读取及输出

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

struct student{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};//结构体类型声明，注意最后一定要加分号

int main()
{
    struct student s={1001,"lele",'M',20,85.4,"Shenzhen"};//定义及初始化
    struct student sarr[3];
    int i;
```

```
    printf("%d %s %c %d %f %s\n", s.num, s.name, s.sex, s.age, s.score, s.addr);
    for(i=0;i<3;i++)
    {

    scanf("%d%s %c%d%f%s",&sarr[i].num,sarr[i].name,&sarr[i].sex,&sarr[i].age,&
sarr[i].score,sarr[i].addr);
    }
    for(i=0;i<3;i++)
    {

    printf("%d %s %c %d %f %s\n",sarr[i].num,sarr[i].name,sarr[i].sex,sarr[i].age,
sarr[i].score,sarr[i].addr);
    }
    system("pause");
    return 0;
    }
```

把结构体类型声明放在 main 函数之上，这样 main 函数中我们才可以使用，工作中往往把结构体声明放在头文件中，注意结构体类型声明最后一定要加分号，否则编译不通，另外定义结构体变量时，使用 struct student 来定义，不能只有 struct 或者 student，否则会编译不通，sarr 是结构体数组变量。结构体的初始化只能在一开始定义时，进行 struct student s={1001,"lele",'M',20,85.4,"Shenzhen"}，如果 struct student s 已经定义，不能在之后，写 s={1001,"lele",'M',20,85.4,"Shenzhen"},这样是编译不通的，如果结构体变量已经定义，只能对其每个成员进行单独赋值，例如 s.num=1003

通过 **结构体变量名.成员名** 来访问结构体成员，例如 s.num 拿到学号，在进行打印输出时，必须访问到成员，而且 printf 中的%类型要与各成员匹配，使用 scanf 读取标准输入时，也必须是各成员取地址，进行存入，不可以&s，即不可以直接对结构体变量取地址，整型%d，浮点型%f，字符串%s 都会忽略空格，但是字符%c 不会忽略空格，所以如果我们读取字符，注意在读取字符与读取其他数据之间，加入空格，具体操作如果不清晰，可以看视频。

可以把以下数据之间直接敲一遍放到 txt 文档中，每次运行时，直接右键，然后编辑，粘贴到控制台窗口即可。这样方便多次测试，以及我们后面会对结构体数组进行排序时也方便，记得保存你的数据在 txt，别忘记啦，执行结果如图 7.1.1-1 所示。

```
1003 lili F 22 84.2 Beijing
1005 zhangsan M 25 96 Shanghai
1001 wangwu M 20 88.3 Hangzhou
```

图 7.1.1-1

7.1.2 结构体指针

一个结构体变量的指针就是该变量所占据的内存段的起始地址。可以设一个指针变量，用来指向一个结构体变量，此时该指针变量的值是结构体变量的起始地址。指针变量也可以用来指向结构体数组中的元素。

下面来看【例 7.1.2-1】实例：

【例 7.1.2-1】结构体指针使用

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

//结构体指针
struct student{
        int num;
        char name[20];
        char sex;
};

int main()
{
    struct student s={1001,"wangle",'M'};
    struct student
sarr[3]={1001,"lilei",'M',1005,"zhangsan",'M',1007,"lili",'F'};
    struct student *p;//定义结构体指针
    int num;
    p=&s;
    printf("%d %s %c\n",p->num,p->name,p->sex);
    p=sarr;
    printf("%d %s %c\n",(*p).num,(*p).name,(*p).sex);//方式一获取成员
    printf("%d %s %c\n",p->num,p->name,p->sex);//方式二获取成员
    printf("----------------------------\n");
    num=p->num++;
    printf("num=%d,p->num=%d\n",num,p->num);
    num=p++->num;
    printf("num=%d,p->num=%d\n",num,p->num);
    system("pause");
    }
```

通过代码实例【例 7.1.2-1】可以看到，p 就是一个结构体指针，可以将结构体 s 取地址赋值给 p，这样通过成员选择操作符->,就可以通过 p 访问结构体的每个成员，然后进行打印。我们知道数组名中存储着数据的首地址，所以我们可以将 sarr 赋值给 p，可以通过两种方式访问对应的成员，(*p).num 访问成员为什么要加括号呢，原因是点号成员选择的优先级高于*（即取值）运算符，所以必须加括号，通过*p 拿到 sarr[0]，然后再获取对应成员。

思考题来啦，前面我们讲过自增运算符，那么 num=p->num++之后，num 的值和 p->num

的值是多少呢，num=p++->num;之后，num 的值和 p->num 的值是多少呢？如果你一眼就可以分辨出来，说明前面的自增运算符已经掌握，如果不清楚，可以看下视频，或者编写一下代码，自己看看结果，看了结果还不明白，就可以 QQ 群里提问啦

7.1.3 typedef 的使用

前面定义结构体变量使用 struct student s，觉的有点长，每次都要写 struct student，有没有好办法呢，其实是有的，用 typedef 声明新的类型名来代替已有的类型名。请看代码实例：

【例 7.1.3-1】typedef 的使用

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

//结构体指针
typedef struct student{
        int num;
        char name[20];
        char sex;
}stu,*pstu;

typedef int INTEGER;

int main()
{
    stu s={1001,"wangle",'M'};
    pstu p;
    INTEGER i=10;
    p=&s;
    printf("i=%d,p->num=%d\n",i,p->num);
    system("pause");
}
```

通过 stu 定义结构体变量，和 struct student 定义结构体变量等价，用 INTEGER 定义变量 i 和通过 int 定义变量 i 等价。pstu 等价于 struct student*，所以 p 是结构体指针变量。

7.2 链表的增删查改

7.2.1 链表是什么

链表是一种常见的重要的数据结构,是动态地进行存储分配的一种结构。头指针存放一个地址，该地址指向一个元素节点：用户需要的实际数据和链接节点，这样我们只需定义一个头指针，即可不断的增加节点，最后一个节点内的指针存储为 NULL，用于判断到达链表尾。如图 7.2.1-1 所示，链表的头指针 head 只需存储第一个节点的首地址，通过遍历，即可访问到每一个元素。

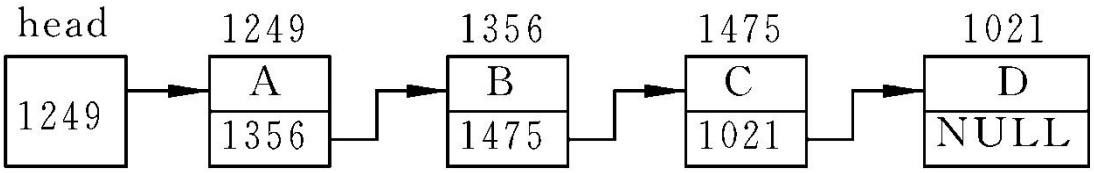

图 7.2.1-1

用结构体建立链表：

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

其中成员 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 是指针类型的成员，它指向 struct student 类型数据（这就是 next 所在的结构体类型）。如图 7.2.1-2 所示。

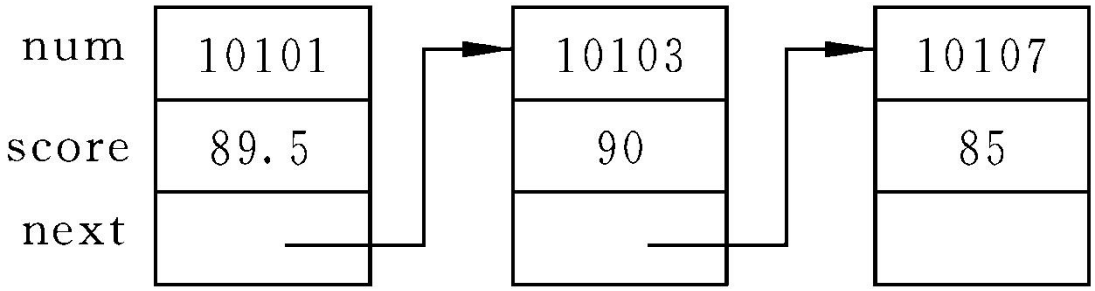

图 7.2.1-2

7.2.2 链表的增删查改

链表的新增分为尾插法，头插法和有序插入，为了高效的实现增删查改，往往会使用两个指针来指向链表，请看代码案例【例 7.2.2-1】。

【例 7.2.2-1】链表的增删查改

func.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct student{
    int num;
    float score;
    struct student* pNext;
}stu,*pstu;

//尾插法
void list_tail_insert(pstu*,stu**,int);
//头插法
void list_head_insert(pstu*,pstu*,int);
//有序插入
```

```
void list_sort_insert(pstu*,pstu*,int);
//删除
void list_delete(pstu*,pstu*,int);
//链表修改
void list_modify(pstu,int,float);
//链表打印
void list_print(pstu);
void list_print_score(pstu);
```

main.c

```
#include "func.h"

int main()
{
    pstu p;
    pstu phead=NULL,ptail=NULL;//代表链表
    int i;
    float f;
    while(scanf("%d",&i)!=EOF)//链表的头插，尾插，有序插入
    {
        list_tail_insert(&phead,&ptail,i);
        //list_head_insert(&phead,&ptail,i);
        //list_sort_insert(&phead,&ptail,i);
    }
    list_print(phead);
    //while(printf("please input delete num:"),scanf("%d",&i)!=EOF)//链表删除
    //{
    //  list_delete(&phead,&ptail,i);
    //  list_print(phead);
    //}
    //while(printf("please input modify num:"),scanf("%d%f",&i,&f)!=EOF)//链表修
改
    //{
    //  list_modify(phead,i,f);
    //  list_print_score(phead);
    //}
    system("pause");
}
```

func.c

```
#include "func.h"

void list_tail_insert(pstu* pphead,stu** pptail,int i)
{
    pstu pnew;
    pnew=(pstu)malloc(sizeof(stu));
```

```
    memset(pnew,0,sizeof(stu));
    pnew->num=i;
    if(NULL==*pptail)//判断链表是否为空,为空时，pnew即为头，也为尾
    {
        *pphead=pnew;
        *pptail=pnew;
    }else{
        (*pptail)->pNext=pnew;//将新节点的地址赋值为尾节点的pnext
        *pptail=pnew;//新节点变为尾指针
    }
}
//只打印学号
void list_print(pstu phead)
{
    while(phead!=NULL)
    {
        printf("%3d ",phead->num);
        phead=phead->pNext;
    }
    printf("\n");
}
//打印学号和分数
void list_print_score(pstu phead)
{
    while(phead!=NULL)
    {
        printf("%3d %5.2f ",phead->num,phead->score);
        phead=phead->pNext;
    }
    printf("\n");
}

void list_head_insert(pstu* pphead,pstu* pptail,int i)
{
    pstu pnew;
    pnew=(pstu)malloc(sizeof(stu));
    memset(pnew,0,sizeof(stu));
    pnew->num=i;
    if(NULL==*pphead)//如果链表为空，pnew赋值给phead,ptail
    {
        *pphead=pnew;
        *pptail=pnew;
    }else{
        pnew->pNext=*pphead;//原有链表的头指针赋值给新节点的pNext
```

```
        *pphead=pnew;//新节点变为链表头部
    }
}

void list_sort_insert(pstu* pphead,pstu* pptail,int i)
{
    pstu pcur;
    pstu ppre;
    pstu pnew;
    pnew=(pstu)malloc(sizeof(stu));
    memset(pnew,0,sizeof(stu));
    pnew->num=i;
    pcur=*pphead; //让大哥，小弟都指向链表头
    ppre=*pphead;
    if(NULL==pcur)//判断链表是否为空
    {
        *pphead=pnew;
        *pptail=pnew;
    }else if(i<pcur->num)//判断i小于头部数据，插入头部
    {
        pnew->pNext=pcur;//原有链表的头指针赋值给新节点的pNext
        *pphead=pnew;//新节点变为链表头部
    }else{
        while(pcur!=NULL)//插入中间
        {
            if(pcur->num>i)
            {
                ppre->pNext=pnew;
                pnew->pNext=pcur;
                break;
            }
            ppre=pcur;//大哥先赋给小弟
            pcur=pcur->pNext;
        }
        if(NULL==pcur)//没插到中间，就是插入尾部
        {
            (*pptail)->pNext=pnew;//将新节点的地址赋值为尾节点的pnext
            *pptail=pnew;//新节点变为尾指针
        }
    }
}

void list_delete(pstu* pphead,pstu* pptail,int delete_num)
{
```

```
    pstu pcur,ppre;
    pcur=*pphead;
    ppre=pcur;
    if(pcur!=NULL)
    {
        if(pcur->num==delete_num)//如果要删除的num和头节点的值相等
        {
            *pphead=pcur->pNext;//改变头指针，指向下一个节点
            if(NULL==*pphead)//删除后链表为空
            {
                *pptail=NULL;
            }
            free(pcur);//释放空间
        }else{//删除中间节点及尾节点
            while(pcur!=NULL)
            {
                if(pcur->num==delete_num)
                {
                    ppre->pNext=pcur->pNext;
                    free(pcur);
                    break;
                }
                ppre=pcur;
                pcur=pcur->pNext;
            }
            if(NULL==ppre->pNext)
            {
                *pptail=ppre;
            }
            if(NULL==pcur)
            {
                printf("no this node\n");
            }
        }
    }else{
        printf("list is NULL\n");
    }
}

void list_modify(pstu phead,int i,float f)
{
    while(phead!=NULL)
    {
        if(phead->num==i)
```

```
        {
            phead->score=f;
            break;
        }
        phead=phead->pNext;
    }
    if(NULL==phead)
    {
        printf("no this node\n");
    }
}
```

把每个节点看成一个学生的信息，主要存储学生的学号信息，成绩信息，为了快速新建链表，在演示新增链表节点和删除链表节点时，我们不输入学生成绩。

list_tail_insert 为尾插法，首先我们初始化链表的头尾指针都为 NULL，头指针指向头节点，尾指针指向尾节点，因此通过子函数实现尾插法时，头指针，尾指针都会被改变，所以头尾指针均需取地址传入 list_tail_insert，对于尾插法，当链表为空时，新增的第一个节点既是头节点也是尾节点，当链表不为空时，新节点放在尾节点之后。

list_head_insert 是头插法，对于头插法，当链表为空时，新增的第一个节点既是头节点也是尾节点，当链表不为空时，新节点放在头节点之前。

list_sort_insert 是有序插入，对于有序插入，当链表不为空时，有序插入需要考虑新节点插入在头部，还是中间，还是尾部。

list_delete 对链表删除时，要考虑链表只剩余一个元素，删除后链表为空的情况，因为链表为空需要将头尾指针设为 NULL，同时删除元素需要考虑删除的是头节点，中间节点，还是尾部节点。

list_modify 对链表进行修改，通过找到对于的 num 值，然后修改其成绩，与查询链表非常相似。并不会改变链表的头尾指针值。

上面所使用的增删查改为不带头节点的增删差异，与王道数据结构上的伪代码不同，不带头节点的难度更高，掌握后，自然可以编写带头节点的增删查改，曾经有同学复试，导师要求其写不带头节点的增删查改，为什么会这样，因为实际在内核及各种中间件，使用的就是不带头节点的增删查改，因此大家需要掌握这种难度稍微高一些的增删查改。

7.3 共用体与枚举

7.3.1 共用体

使几个不同的变量共占同一段内存的结构称为“共用体”类型的结构。共用体变量所占的内存长度等于最长的成员的长度（如图 7.3.1-1）。

定义共用体类型变量的一般形式为：

```
union   共用体名
 {
      成员表列
  } 变量表列;
```

定义共用体变量方法，例如：

```
union data                          union data
   { int i;                            { int i;
```

```
        char ch;   或                    char ch;
         float f;                          float f;
    } a,b,c;                          };union data a,b,c;
```

如果共用体变量 a 的起始地址为 1000，那么公共体内成员 i，ch，f 的起始地址均为 1000。

注意：

(1)同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一种，而不是同时存放几种。

(2) 共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员后原有的成员就失去作用。

(3) 共用体变量的地址和它的各成员的地址都是同一地址。

(4) 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值。

(5) 共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

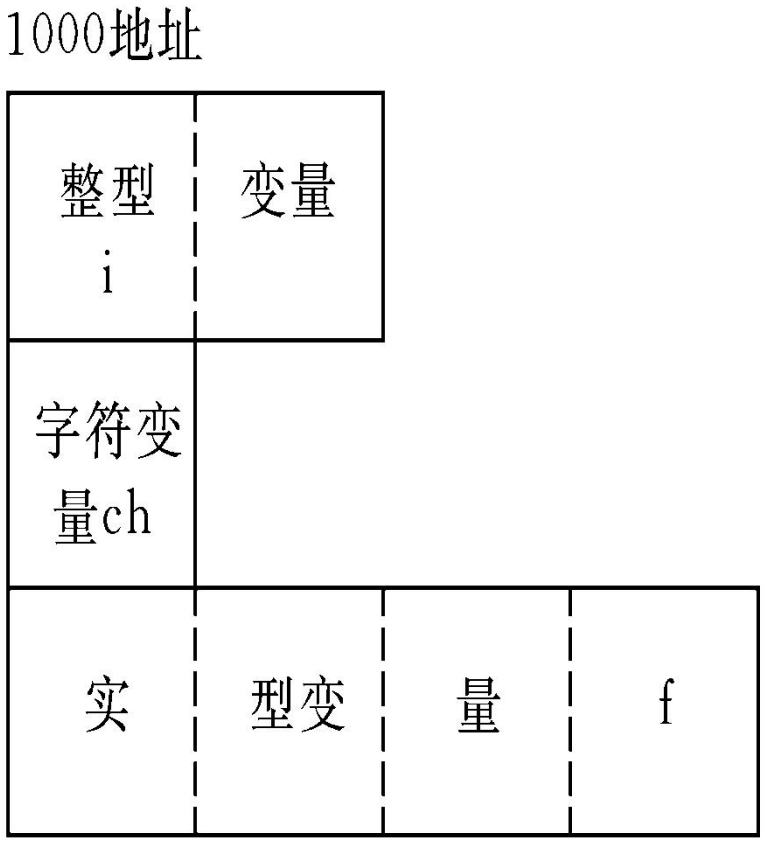

图 7.3.1-1

7.3.2 枚举

枚举：将变量的值一一列举出来，变量的值只限于列举出来的值的范围内。

申明枚举类型用 enum

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

定义变量：

```
enum weekday workday，week-day;
```

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

变量值只能是 sun 到 sat 之一 。

(1)在 C 编译中，对枚举元素按常量处理，故称枚举常量。它们不是变量，不能对它们赋值。

(2) 枚举元素作为常量，它们是有值的， C 语言编译按定义时的顺序使它们的值为 0，1， 2…

(3) 枚举值可以用来作判断比较。

(4) 一个整数一般不直接赋给一个枚举变量。(通过把枚举常量赋给枚举变量)