

The background features a dark blue grid pattern. Overlaid on it are several semi-transparent, glowing spheres in shades of blue, green, and yellow, arranged in a cluster that suggests depth and motion.

结构体数据类型

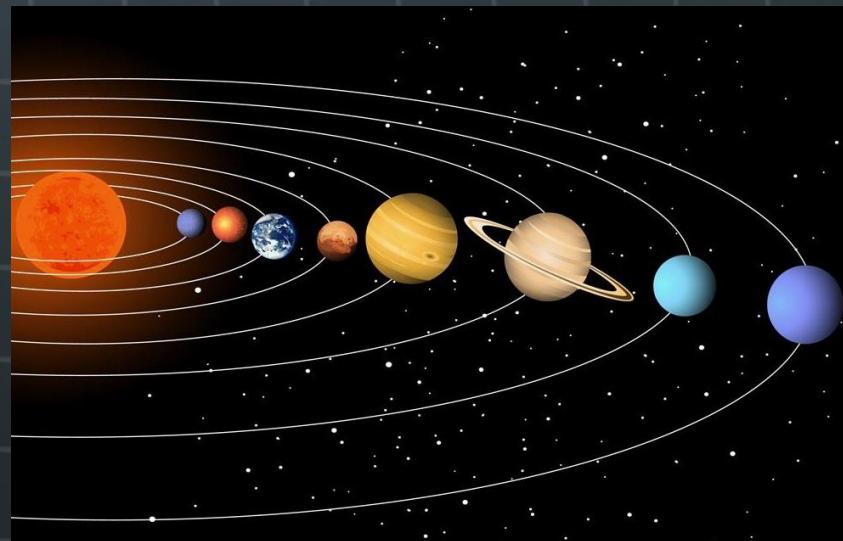
程 淼

E-mail: mew_cheng@outlook.com

集合结构

太阳系家族

太阳 the sun	水星 the mercury	金星 the venus	地球 the earth	火星 the mars	木星 the jupiter	土星 the saturn	天王星 the uranus	海王星 the neptune
 太阳是太阳系里一个有着巨大影响并占支配地位的天体，几乎占整个太阳系质量的全部。	 水星距太阳只有五百八十万公里，是太阳系内离太阳最近的一颗行星，也是最热的行星之一。	 金星表面覆盖着主要由硫酸雾组成的浓密云层，地表温度达四百多度，比水星还要热。	 地球上存在着生命，这使地球得以区别其他行星而成为太阳系中一个最特别的天体。	 近邻火星在很多方面与地球很相似，它是目前所知地球外最有可能存在生命的地方。	 木星堪称“行星之王”，是太阳系中最大的行星，其他所有行星加起来也没有它大。	 土星在许多方面和木星十分相像，它最突出的特征是环绕其赤道的壮观的土星光环。	 很久以前遭受到一次猛烈天体撞击使天王星这个太阳系中第三大的行星只能躺着旋转。	 海王星的世界是一个惊心动魄的世界，它是太阳系中大气活动最为剧烈的一个行星。



集合结构

- 定义一个学生的信息

- 性别(sex)、年龄(age)、成绩(score)、地址(addr)
- 定义为互相独立的简单变量，难以反映它们之间的内在联系

num	name	sex	age	score	addr
10010	Li Fang	M	18	87.5	Beijing

定义和使用结构体变量

- C语言允许用户自己建立由不同类型数据组成的组合型的数据结构，称为**结构体(structure)**
 - 在其他一些高级语言中称为”记录”(record)

定义和使用结构体变量

- 在程序建立结构体类型，一般形式为

- struct 结构体名**
 - { 成员表列 }**

```
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
```

定义和使用结构体变量

- 结构体类型并非只有一种，可以任意指定结构体类型
- 结构体成员可以属于另一个结构体类型

```
struct Date
{
    int month;
    int day;
    int year;
};
```

```
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    struct Date birthday;
    char addr[30];
};
```

num	name	sex	age	birthday			addr
				month	day	year	

定义结构体类型变量

- 先声明结构体类型，再定义该类型的变量
 - 建立了一个结构体类型，相当于一个模型，并没有定义变量
 - 已声明了一个结构体类型**struct Student**，可以用它来定义变量
- struct Student student1, student2**

| | |
结构体类型名 结构体 变量名

Student1:	10001	Zhang Xin	M	19	90.5	Shanghai
-----------	-------	-----------	---	----	------	----------

Student2:	10002	Wang Li	F	20	98	Beijing
-----------	-------	---------	---	----	----	---------

定义结构体类型变量

- 在**定义struct Student类型的同时，定义变量**
- 一般形式为

```
struct 结构体名  
{  
    成员表列  
}变量名表列;
```

```
struct Student  
{  
    int num;  
    char name[20];  
    char sex;  
    int age;  
    float score;  
    char addr[30];  
}student1, student2;
```

定义结构体类型变量

- 不指定类型名而直接定义结构体类型变量
 - 不能再以此结构体类型去定义其他变量
- 一般形式为

```
struct
{
    成员列表
}变量名表列;
```

定义结构体类型变量

- 结构体类型与结构体变量是不同的概念，不要混同
- 结构体类型中的成员名可以与程序中的变量名相同，但二者不代表同一对象
- 对结构体变量中的成员(即“域”), 可以单独使用，它的作用与地位相当于普通变量

结构体变量的初始化和引用

- 在定义结构体变量时，可以对它初始化，即赋予初始值
 - 初始化列表是用花括号括起来的一些常量，这些常量依次赋给结构体变量中的各成员
 - C99标准允许对某一成员初始化，如

```
struct Student b = {.name="Zhang Fang"}; // 在成员名前有成员运算符
```

“.name” 隐含代表结构体变量**b**中的成员**b.name**

- 未被指定初始化的数值型成员被系统初始化为0，字符型成员被系统初始化为‘\0’

结构体变量的初始化和引用

- 可以引用结构体变量中成员的值，引用方式为
 - 结构体变量名.成员名
 - 例如： student1.num = 10010;
- 不能企图输出结构体变量名来达到输出结构体变量所有成员的值。
 - `printf("%s\n", student1);` X
- 只能对结构体变量中的各个成员分别进行输入和输出

结构体变量的初始化和引用

- 一个例子：把一个学生的信息(包括学号、姓名、性别、住址)放在一个结构体变量中，然后输出这个学生的信息
- 解题思路：建立一个结构体类型，包括学生信息的各成员。然后用它来定义结构体变量，同时赋予初值(学生的信息)。最后输出该结构体变量的各成员(即该学生的信息)

结构体变量的初始化和引用

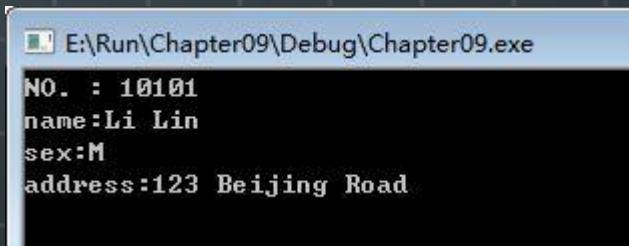
编写程序

```
#include <stdio.h>

int main()
{
    struct Student
    {
        long int num;
        char name[20];
        char sex;
        char addr[20];
    }a = {10101, "Li Lin", 'M', "123 Beijing Road"};

    printf("NO. : %ld\nname:%s\nsex:%c\naddress:%s\n", a.num, a.name, a.sex, a.addr);

    return 0;
}
```



结构体变量的初始化和引用

- 如果成员本身又属一个结构体类型，则要用若干个成员运算符
- 只能对最低级的成员进行赋值或存取以及运算
- 例如，如果在结构体**struct Student**类型的成员中包含另一个结构体**struct date**类型的成员**birthday**
 - student1.num**
 - student1.birthday.month**

结构体变量的初始化和引用

- 对结构体变量的成员可以像普通变量一样进行各种运算
- 例如
 - `student2.score = student1.score;`
 - `sum = student1.score+student2.score;`
 - `student1.age++;`

结构体变量的初始化和引用

- 同类的结构体变量可以互相赋值
 - 如， `student1 = student2;`
- 可以引用结构体变量成员的地址， 也可以引用结构体变量的地址,例如
 - `scanf("%d", &student1.num);`
 - `printf("%o", &student1);`
- 不能整体读入结构体变量
 - `scanf("%d, %s, %c, %d, %f, %s\n", &student1);`

结构体变量的初始化和引用

- 一个例子：输入两个学生的学号、姓名和成绩，输出成绩较高的学生的学号、姓名和成绩
- 解题思路
 - 定义两个结构相同的结构体变量**student1**和**student2**；
 - 分别输入两个学生的学号、姓名和成绩
 - 比较两个学生的成绩

结构体变量的初始化和引用

编写程序

```
#include <stdio.h>

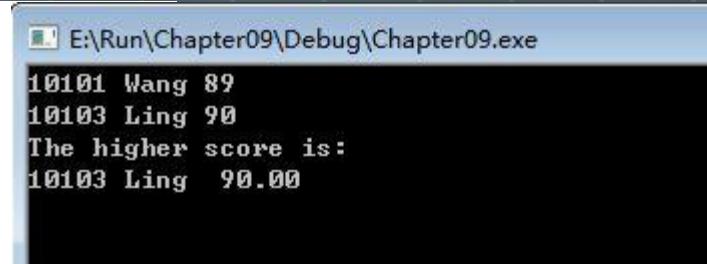
int main()
{
    struct Student
    {
        int num;
        char name[20];
        float score;
    } student1, student2;

    scanf("%d%s%f", &student1.num, student1.name, &student1.score);
    scanf("%d%s%f", &student2.num, student2.name, &student2.score);

    printf("The higher score is:\n");

    if (student1.score > student2.score)
        printf("%d %s %.2f\n", student1.num, student1.name, student1.score);
    else if (student1.score < student2.score)
        printf("%d %s %.2f\n", student2.num, student2.name, student2.score);
    else
    {
        printf("%d %s %.2f\n", student1.num, student1.name, student1.score);
        printf("%d %s %.2f\n", student2.num, student2.name, student2.score);
    }

    return 0;
}
```



使用结构体数组

结构体数组

- 一个结构体变量中可以存放一组有关联的数据
- 结构体数组与以前介绍过的数值型数组的不同之处在于**每个数组元素都是一个结构体类型的数据**

结构体数组

- 定义结构体数组一般形式
 - **struct 结构体名**
 - **{成员表列} 数组名[数组长度];**
- 先声明一个结构体类型(如**struct Person**)， 然后再用此类型定义结构体数组
 - 结构体类型 数组名[数组长度];
 - **struct Person leader[3];**
- 对结构体数组初始化的形式是在定义数组的后面加上 =
 {初值表列};
 - **struct Person leader[3] = {"Li", 0, "Zhang", 0, "Sun", 0};**

结构体数组

- 一个例子：有3个候选人，每个选民只能投票选一人，要求编一个统计选票的程序，先后输入被选人的名字，最后输出个人得票结果
- 解题思路：定义一个结构体数组，每个元素中的信息应包括候选人的姓名和得票数

结构体数组



编写程序：

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

struct Person
{
    char name[20];
    int count;
};

leader[3] = {"Li", 0, "Zhang", 0, "Sun", 0};

int main()
{
    int i, j;
    char leader_name[20];
    for (i = 1; i <= 10; i++)
    {
        scanf("%s", leader_name);
        for (j = 0; j < 3; j++)
            if (strcmp(leader_name, leader[j].name) == 0)
                leader[j].count++;
    }

    printf("\nResult:\n");
    for (i = 0; i < 3; i++)
        printf("%5s:%d\n", leader[i].name, leader[i].count);

    return 0;
}
```

```
E:\Run\Chapter09\Debug\Chapter09.exe
Li
Li
Sun
Zhang
Zhahg
Sun
Li
Sun
Zhang
Li

Result:
    Li:4
    Zhang:2
    Sun:3
```

结构体数组

- 一个例子：有n个学生的信息(包括学号、姓名、成绩)，要求按照成绩的高低顺序输出各学生的信息
- 解题思路：用结构体数组存放学生的信息，然后依照结构体的成员数据对结构体元素进行排序

结构体数组



编写程序

```
#include <stdio.h>

struct Student
{
    int num;
    char name[20];
    float score;
};

int main()
{
    struct Student stu[5] = {{10101, "Zhang", 78}, {10103, "Wang", 98.5}, {10106, "Li", 86}, {10108, "Ling", 73.5}, {10110, "Sun", 100}};
    struct Student temp;
    const int n = 5;
    int i, j, k;

    printf("The order is:\n");
    for (i = 0; i < n - 1; i++)
    {
        k = i;
        for (j = i+1; j < n; j++)
            if (stu[j].score > stu[k].score)
                k = j;

        temp = stu[k];
        stu[k] = stu[i];
        stu[i] = temp;
    }
    for (i = 0; i < n; i++)
        printf("%6d %8s %6.2f\n", stu[i].num, stu[i].name, stu[i].score);

    printf("\n");

    return 0;
}
```

```
E:\Run\Chapter09\Debug\Chapter09.exe
The order is:
10110      Sun 100.00
10103      Wang 98.50
10106      Li 86.00
10101      Zhang 78.00
10108      Ling 73.50
```

结构体指针

结构体指针

- 结构体指针就是指向结构体变量的指针
- 结构体变量的起始地址就是这个结构体变量的指针
- 指向结构体对象的指针变量既可指向结构体变量，也可指向结构体数组中的元素，比如
 - `struct Student * pt;`
 - `(*pt).num`是`pt`指向的结构体变量中的成员`num`

结构体指针

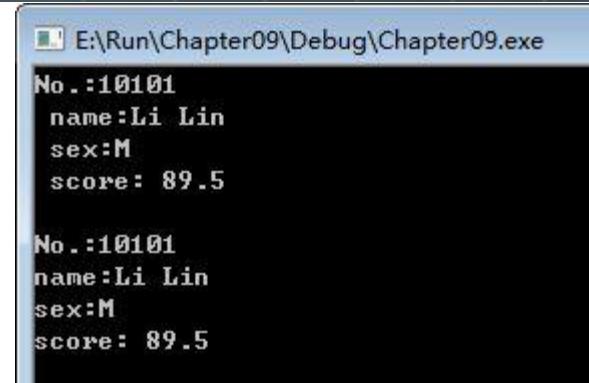
- 一个例子：通过指向结构体变量的指针变量输出结构体变量中成员的信息
- 解题思路：通过结构体变量的指针访问结构体

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

int main()
{
    struct Student
    {
        long num;
        char name[20];
        char sex;
        float score;
    };

    struct Student stu_1;
    struct Student * p;
    p = &stu_1;
    stu_1.num = 10101;
    strcpy(stu_1.name, "Li Lin");
    stu_1.sex = 'M';
    stu_1.score = 89.5;
    printf("No.:%ld\n name:%s\n sex:%c\n score:%5.1f\n", stu_1.num, stu_1.name, stu_1.sex, stu_1.score);
    printf("\nNo.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n", (*p).num, (*p).name, (*p).sex, (*p).score);

    return 0;
}
```



```
E:\Run\Chapter09\Debug\Chapter09.exe
No.:10101
name:Li Lin
sex:M
score: 89.5

No.:10101
name:Li Lin
sex:M
score: 89.5
```

结构体指针

- 使用结构体指针访问结构体成员
 - $(*p).num$ 可以用 $p->num$ 来代替，代表 p 所指向的结构体变量中的 num 成员
 - $(*p).name$ 等价于 $p->name$
- 如果 p 指向一个结构体变量 stu ，以下用法等价
 - $stu.$ 成员名(如 $stu.num$);
 - $(*p).$ 成员名(如 $(*p).num$);
 - $p->$ 成员名(如 $p->num$)

结构体指针

- 一个例子：有3个学生的信息，放在结构体数组中，要求输出全部学生的信息
- 解题思路：定义一个指向结构体的指针变量，对结构体数组进行遍历访问

结构体指针

编写程序：

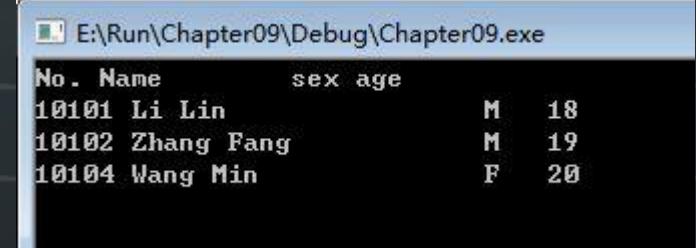
```
#include <stdio.h>

struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
};

struct Student stu[3] = {{10101, "Li Lin", 'M', 18}, {10102, "Zhang Fang", 'M', 19}, {10104, "Wang Min", 'F', 20}};

int main()
{
    struct Student * p;
    printf("No. Name      sex age\n");
    for (p = stu; p < stu+3; p++)
        printf("%5d %-20s %2c %4d\n", p->num, p->name, p->sex, p->age);

    return 0;
}
```



No.	Name	sex	age
10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

结构体指针

- 用结构体变量和结构体变量的指针作函数参数
 - 用结构体变量的成员作参数
 - 用结构体变量作实参
 - 用指向结构体变量(或数组元素)的指针作实参，将结构体变量的地址传给形参

结构体指针

- 一个例子：有n个结构体变量，内含学生学号、姓名和3门课程的成绩。要求输出平均成绩最高的学生的信息
- 解题思路：将n个学生的信息表示为结构体数组，利用指针对其进行访问，用不同的函数来对结构体进行处理，得到各自的平均成绩并进行比较

结构体指针



编写程序：

```
#include <stdio.h>
#define N 3

struct Student
{
    int num;
    char name[20];
    float score[3];
    float aver;
};

int main()
{
    void input(struct Student stu[]);
    struct Student max(struct Student stu[]);
    void print(struct Student stu);
    struct Student stu[N], *p = stu;
    input(p);
    print(max(p));

    return 0;
}

void input(struct Student stu[])
{
    int i;
    printf("请输入各学生的信息：学号、姓名、三门课成绩：\n");
    for (i = 0; i < N; i++)
    {
        scanf("%d %s %f %f %f", &stu[i].num, stu[i].name, &stu[i].score, &stu[i].score[1], &stu[i].score[2]);
        stu[i].aver = (stu[i].score[0]+stu[i].score[1]+stu[i].score[2])/3.0;
    }
}

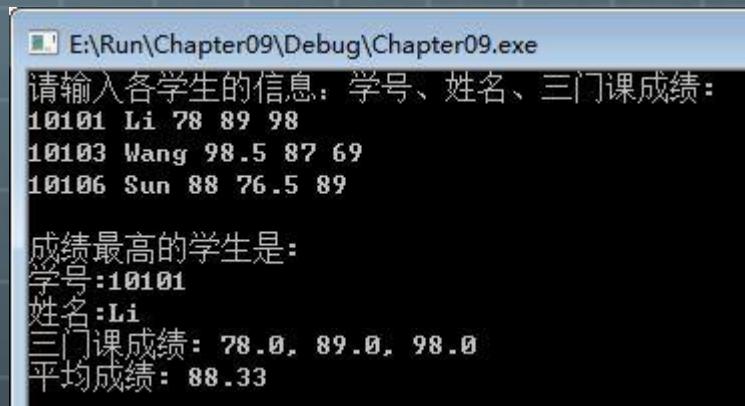
struct Student max(struct Student stu[])
{
    int i, m = 0;
    for (i = 0; i < N; i++)
        if (stu[i].aver > stu[m].aver) m = i;

    return stu[m];
}

void print(struct Student stud)
{
    printf("\n成绩最高的学生是：\n");
    printf("学号：%d\n姓名：%s\n三门课成绩：%5.1f,%5.1f,%5.1f\n平均成绩：%6.2f\n",
           stud.num, stud.name, stud.score[0], stud.score[1], stud.score[2], stud.aver);
}
```

结构体指针

编写程序



```
E:\Run\Chapter09\Debug\Chapter09.exe
请输入各学生的信息：学号、姓名、三门课成绩：
10101 Li 78 89 98
10103 Wang 98.5 87 69
10106 Sun 88 76.5 89

成绩最高的学生是：
学号：10101
姓名：Li
三门课成绩：78.0, 89.0, 98.0
平均成绩：88.33
```

用指针处理链表

链表结构

- 链表是一种常见的数据结构
 - 动态进行存储分配的一种结构
- 无法确定存储大小，则必须把数组定得足够大



链表结构

用结构体建立链表

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



链表结构

- 一个例子：建立一个简单链表，由3个学生数据的结点组成，要求输出各结点中的数据
- 解题思路：声明一个结构体类型，其成员包括num(学号)，score(成绩)，next(指针变量)

链表结构

编写程序：

```
#include <stdio.h>

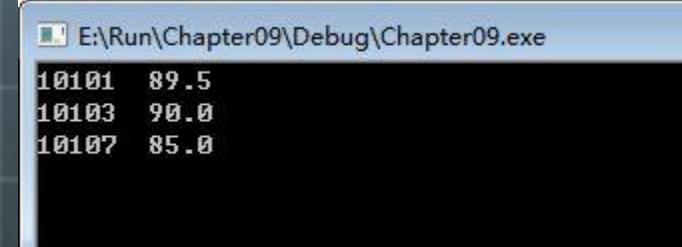
struct Student
{
    int num;
    float score;
    struct Student * next;
};

int main()
{
    struct Student a, b, c, * head, * p;
    a.num = 10101;
    a.score = 89.5;
    b.num = 10103;
    b.score = 90;
    c.num = 10107;
    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);

    return 0;
}
```



E:\Run\Chapter09\Debug\Chapter09.exe

num	score
10101	89.5
10103	90.0
10107	85.0

链表结构

- 所谓**动态链表**是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系

链表结构

- 一个例子：写一函数建立一个有3名学生数据的单向动态链表



编写程序

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

#define LEN sizeof(struct Student)

struct Student
{
    long num;
    float score;
    struct Student * next;
};

int n;

struct Student * creat(void)
{
    struct Student * head;
    struct Student * p1, * p2;
    n = 0;
    p1 = p2 = (struct Student *) malloc(LEN);
    scanf("%ld, %f", &p1->num, &p1->score);
    head = NULL;
    while(p1->num != 0)
    {
        n = n+1;
        if (n == 1) head = p1;
        else p2->next = p1;
        p2 = p1;
        p1 = (struct Student *)malloc(LEN);
        scanf("%ld, %f", &p1->num, &p1->score);
    }
    p2->next = NULL;
}
return(head);
}

int main()
{
    struct Student * pt;
    pt = creat();
    printf("\nnum:%ld\nscore:%5.1f\n", pt->num, pt->score);

    return 0;
}
```

```
E:\Run\Chapter09\Debug\Chapter09.exe
1001, 76.5
1003, 87
1004, 99.5
0, 0

num:1001
score: 76.5
```

链表结构

- 输出链表的例子：编写一个输出链表的函数print

链表结构



编写程序

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

#define LEN sizeof(struct Student)

struct Student
{
    long num;
    float score;
    struct Student * next;
};

int n;

void print(struct Student * head)
{
    struct Student * p;
    printf("\nNow, These %d records are:\n", n);
    p = head;
    if (head != NULL)
        do{
            printf("%ld %5.1f\n", p->num, p->score);
            p = p->next;
        }while(p != NULL);
}
```

E:\Run\Chapter09\Debug\Chapter09.exe

```
1001, 67.5
1003, 87
1005, 99
0, 0

Now, These 3 records are:
1001 67.5
1003 87.0
1005 99.0
```

```
struct Student * creat(void)
{
    struct Student * head;
    struct Student * p1, * p2;
    n = 0;
    p1 = p2 = (struct Student *) malloc(LEN);
    scanf("%ld, %f", &p1->num, &p1->score);
    head = NULL;
    while(p1->num != 0)
    {
        n = n+1;
        if (n == 1) head = p1;
        else p2->next = p1;
        p2 = p1;
        p1 = (struct Student *)malloc(LEN);
        scanf("%ld, %f", &p1->num, &p1->score);
    }
    p2->next = NULL;
    return(head);
}

int main()
{
    struct Student * head;
    head = creat();
    print(head);

    return 0;
}
```

共用体类型

共用体类型

- 用一段内存单元存放不同类型的变量
- 结构体变量所占内存长度是各成员占的内存长度之和。每个成员分别占用其自己的内存单元
- 共用体变量所占的内存长度等于最长的成员的长度

```
union 共用体名  
{ 成员表列;  
}变量表列;
```

共用体类型

- 例如，把一个短整型变量、一个字符型变量和一个实型变量放在同一个地址开始的内存单元中

```
union Data
{
    int i;
    char ch;
    float f;
}a, b, c;
```

1000地址



共用体类型

- 只有先定义了共用体变量才能引用它，不能引用共用体变量，而只能引用共用体变量中的成员。例如，
 - a.i
 - a.ch
 - a.f

共用体类型的特点

- 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个成员，而不是同时存放几个
- 可以对共用体变量初始化，但初始化表中只能有一个常量
- 共用体变量中起作用的成员是最后一次被赋值的成员，在对共用体变量中的一个成员赋值后，原有变量存储单元中的值就取代

共用体类型的特点

- 共用体变量的地址和它的各成员的地址都是同一地址
- 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
- 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

共用体类型

- 例子：有若干个人员的数据，其中有学生和教师。学生的数据其中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。要求用同一个表格来处理



编写程序

```
#include <stdio.h>

struct
{
    int num;
    char name[10];
    char sex;
    char job;
    union
    {
        int clas;
        char position[10];
    }category;
}person[2];

int main()
{
    int i;
    for (i = 0; i < 2; i++)
    {
        printf("please enter the data of person:\n");
        scanf("%d %s %c %c", &person[i].num, &person[i].name, &person[i].sex, &person[i].job);

        if (person[i].job == 's')
            scanf("%d", &person[i].category.clas);
        else if (person[i].job == 't')
            scanf("%s", person[i].category.position);
        else
            printf("Input error!");
    }
    printf("\n");
    printf("No.      name      sex      job      class/positon\n");
    for (i = 0; i < 2; i++)
    {
        if (person[i].job == 's')
            printf("%-6d%-10s%-4c%-4c%-10d\n", person[i].num, person[i].name,
                   person[i].sex, person[i].job, person[i].category.clas);
        else
            printf("%-6d%-10s%-4c%-4c%-10s\n", person[i].num, person[i].name,
                   person[i].sex, person[i].job, person[i].category.position);
    }
}

return 0;
}
```

```
E:\Run\Chapter09\Debug\Chapter09.exe
please enter the data of person:
101 Li f s 501
please enter the data of person:
102 Wang m t prof

No.      name      sex      job      class/positon
101      Li          f          s      501
102      Wang        m          t      prof
```

枚举类型

枚举类型

枚举类型：把可能的值一一列举出来，**变量的值只限于列举出来的值的范围内**

声明枚举类型用**enum**开头。

enum [枚举名] {枚举元素列表};

例如，

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

枚举元素或枚举常量

enum Weekday workday, weekend;

枚举类型

枚举变量

枚举类型

- 也可以不声明有名字的枚举类型，而直接定义枚举变量
 - `enum{sun, mon, tue, wed, thu, fri, sat} workday, weekend;`

枚举类型

- C编译对枚举类型的枚举元素按常量处理，故称枚举常量
 - 不能对它们赋值。
 - 例如， `sun = 0; mon = 1; X`
- 每一个枚举元素都代表一个整数，C语言编译按定义时的顺序默认它们的值为`0, 1, 2, 3, 4, 5...`。
 - 例如， `workday = mon;` 相当于 `workday = 1;`
- 也可以人为地指定枚举元素的数值，在定义枚举类型时显式地指定
 - 例如， `enum Weekday{sun=7, mon=1, tue, wed, thu, fri, sat}workday, week_end;`

枚举类型

- 枚举常量是可以引用和输出的
 - 例如，`printf("%d", workday);` 将输出整数1
- 枚举元素可以用来判断比较
 - 例如
 - `if (workday == mon) ...`
 - `if (workday > sun) ...`

枚举类型

用typedef声明新类型名

typedef

- 可以用**typedef**指定新的类型名来代替已有的类型名
 - 简单地用一个新的类型名代替原有的类型名。例如，
 - typedef int Integer;** int i, j; float a, b;
 - typedef float Real;** Integer i, j; Real a, b;
- 命名一个简单的类型名代替复杂的类型表示方法

typedef

- 命名一个新的类型名代表结构体类型

```
typedef struct
{
    int month;
    int day;
    int year;
}Date;
```

```
Date birthday;
Date * p;
```

typedef

- 命名一个新的类型名代表数组类型

```
typedef int Num[100];  
Num a;
```

- 命名一个新的类型名代表指针类型

```
typedef char * String;  
String p, s[10];
```

- 命名一个新的类型名代表指向函数的指针类型

```
typedef int (* Pointer) ( );  
Pointer p1, p2;
```

typedef

- 声明一个新类型名的方法：
 - 先按定义数组变量形式书写： int a[100];
 - 将变量名a换成自己命名的类型名： int Num[100];
 - 在前面加上**typedef**， 得到**typedef int Num[100];**
 - 用来定义变量： **Num a;**

typedef

- **typedef**与**#define**的区别：
 - **#define**是在预编译时处理，它只能作简单的字符串替换
 - **typedef**是在编译阶段处理的

