第9章 建立自己的数据类型（2）

S.IST@XMU

复习回顾

- **上次课的内容：**
  - **常见的内存错误**
  - **指针小结**
  - **程序的调试**
  - **结构体的概念**
  - **一句话证明你学过结构体**

有一天，小强和小明在网聊

他俩不知为什么吵起来了

没学过结构体的小强吼了一句：

“小明你这个猪！”

学过结构体的小明不假思索地回了一句：

xiaoqiang.IsPig = TRUE

如何用递归输出n的全排列

- n的范围：1～9
- 样例输入：3
- 样例输出：

```
123
132
213
231
312
321
```

```
#include <stdio.h>

int  n;
int arr[10]; //存放全排列的数组

void make_permute(int pos);
int no_repeat(int pos, int value);

int main()
{
   scanf("%d", &n);
   make_permute(0); //递归调用
   return 0;
}
```

用递归输出n的全排列-递归函数

```
1. void make_permute(int pos)
2. {
3.     int i, j;
4.     if (pos == n)
5.     { //当安排到第n+1位即输出，不必继续递归
6.         for (i=0; i<n; i++)
7.         {
8.             printf("%d", arr[i]);
9.         }
10.        printf("\n");
11.    }
12.    else
13.    {   //j为当前位置所有可能的取值
14.        for (j=1; j<=n; j++)
15.        {   //若j与前pos位的值均不相等
16.            if (no_repeat(pos,j)==1)
17.            {   //安排j为当前位置的值
18.                arr[pos] = j;
19.                make_permute(pos+1);
20.            }
21.        }
22.    }
23.}
```

```
int no_repeat( int pos,
               int value )
{
   int i;
   for (i=0; i<pos; i++)
   {
        if (arr[i]==value)
        {
               return 0;
        }
   }
   return 1;
}
```

如何初始化结构体变量

```
struct Person
{
     char name[20];
    int height;
    int weight;
};
```

这是height

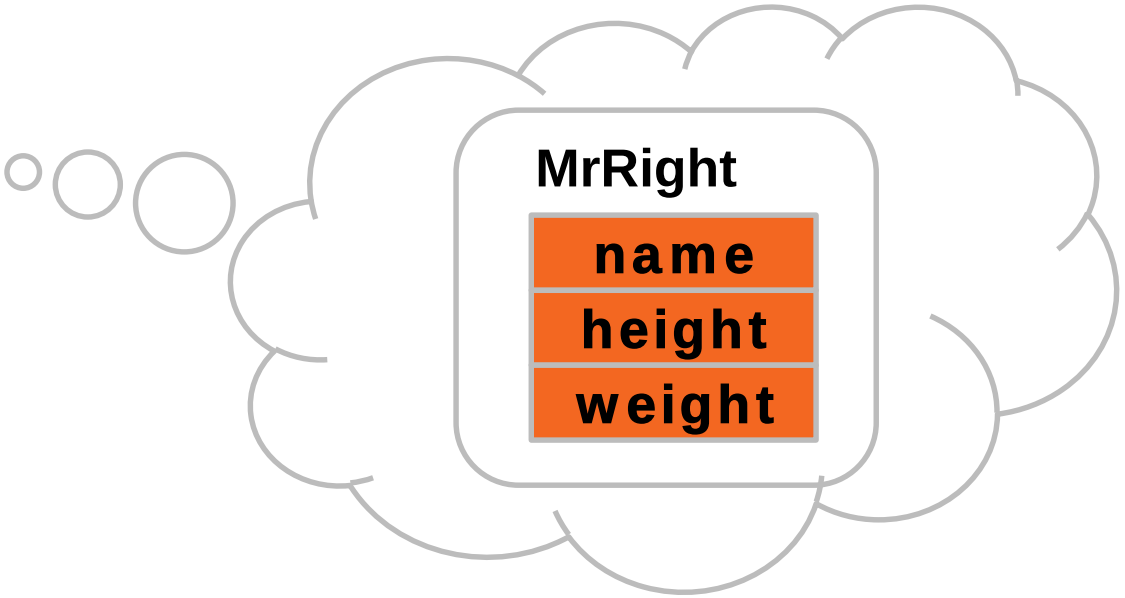
```
struct Person MrRight = {"Huang Xiaoming", 179, 65};
```

这是name

这是weight

访问结构体成员的方式1：.

```
struct Person
{
    char
   name[20];
    int height;
    int weight;
} MrRight;
```



- 如何对height变量进行赋值？
  - `height = 179;` //错误！计算机提示找不到height的定义！
  - `MrRight.height = 179;` //正确！ MrRight.height是int型变量，
    // 和普通的int型变量一样可以进行
    // 赋值和取值操作

“.”运算符

- 成员运算符，一般和结构体变量名称一起使用，用来指定结构体变量的成员
- 引用结构体变量的一般形式

结构体变量名.成员名

- wusong.number--;
- scanf("%s", wusong.weapon);
- printf("%s", wusong.nickname);

结构体变量的初始化和引用

例：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：**

- ◆自己建立一个结构体类型，包括有关学生信息的各成员
- ◆用它定义结构体变量，同时赋以初值
- ◆输出该结构体变量的各成员

```
#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;
}
```

```
NO.:10101
name:Li Lin
sex:M
address:123 Beijing Road
```

```
#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;
}
```

- 初始化结构体变量时不要忘了两个大括号！

```
#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"};

    a.num=10010;  对

    printf("%s\n",a); 不对

    ……
    return 0;
}
```

- 必须先定义结构体变量，才能对其进行引用！

- 不能对结构体变量整体进行诸如输入/输出的操作！

```
#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"};

    struct Student b;

    b=a;  对

    b.num++;  对

    ……
    return 0;
}
```

- C语言允许两个相同类型的结构体变量之间进行整体赋值

- 结构体成员变量可以像普通变量一样参与各种运算和操作

```
#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"};
    scanf("%ld",&a.num);  对
    printf("%o",&a);  对
    scanf("%ld,%s,%c,%s\n",&a);错
    ……
    return 0;
}
```

- 可以引用结构体变量地址，也可以引用结构体成员变量的地址

```
#include <stdio.h>
int main()
{
    int sum;
    struct Date { int month; int day; int year; };
    struct Student
    {
        long int num;
        char name[20];
        char sex;
        int age;
        struct Date birthday;
        char addr[20];
    } a, b;
    a.birthday.month=12;  对
    a.age=10;  b.age=9; 对
    sum=a.age+b.age; 对
    ……
    return 0;
}
```

- 如果一个结构体变量的成员又是一个结构体类型，引用时要用成员运算符逐级遍历到最底层的成员。

结构体的嵌套

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

```
struct Person
{
    char name[20];
    int height;
    int weight;
    struct Date birthday;
};
```

中文名	黄晓明
外文名	Huang Xiaoming
身　高	179cm
体　重	65kg
出生日期	1977年11月13日

这是新增的成员birthday，属于Date类型

```
struct Person MrRight={"Huang Xiaoming", 179, 65, {1977,11,13}};
```

这是name　这是height　这是weight　这是birthday

结构体嵌套：结构体的成员本身可以属于某种结构体类型

想获得 MrRight 的出生年份？ `MrRight.birthday.year;`

结构体变量应用实例

例： 输入两个学生的学号、姓名和成绩，输出成绩较高学生的学号、姓名和成绩

- **解题思路：**

(1)定义两个结构相同的结构体变量student1和student2；

(2)分别输入两个学生的学号、姓名和成绩；

(3)比较两个学生的成绩，如果学生1的成绩高于学生2，就输出学生1的全部信息，如果学生2的成绩高于学生1，就输出学生2的全部信息。如果二者相等，输出2个学生的全部信息

代码实现：

```
#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 %6.2f\n",student1.num, student1.name,
               student1.score);
    else if (student1.score < student2.score)
        printf("%d %s %6.2f\n",student2.num, student2.name,
               student2.score);
    else
    {
        printf("%d %s %6.2f\n",student1.num,
            student1.name, student1.score);
        printf("%d %s %6.2f\n",student2.num,
            student2.name, student2.score);
    }
    return 0;
}
```

不必加&

```
10101 Wang 89
10103 Ling 90
The higher score is:
10103  Ling   90.00
```

结构体数组的例子

例：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，最后输出各人得票结果。

结构体数组的例子

- 解题思路：
  - 设一个结构体数组，数组中包含3个元素
  - 每个元素中的信息应包括候选人的姓名(字符型)和得票数(整型)
  - 输入被选人的姓名，然后与数组元素中的“姓名”成员比较，如果相同，就给这个元素中的“得票数”成员的值加1
  - 输出所有元素的信息

```
#include <string.h>
#include <stdio.h>
struct Person
{
    char name[20];
    int count;
}leader[3]={"Li",0,"Zhang",0,"Sun",0};
```

全局的结构体数组

leader[0]

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

leader[j].count=leader[j].count+1;

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

```
Li
Li
Fun
Zhang
Zhabg
Fun
Li
Fun
Zhang
Li
```

```
   Li:4
Zhang:3
  Sun:3
```

关于结构体数组定义的说明

(1)定义结构体数组一般形式是

① struct 结构体名

{成员表列} 数组名[数组长度];

② 先声明一个结构体类型，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];

如：

struct Person leader[3];

关于结构体数组定义的说明

(2)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｝;

如：

```
struct Person  leader[3]=
  {"Li",0,"Zhang",0,"Fun",0};
```

结构体数组应用举例

例：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n个学生信息，采用选择法对各元素进行排序(进行比较的是各元素中的成绩)。

```
#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,"Fun",100}};
    struct Student temp;
    const int n = 30;  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;
}
```

常变量

注意temp的类型

若人数变为30

写法上与普通变量一致

指向结构体变量的指针

- 指向结构体对象的指针变量既可以指向结构体变量，也可以用来引用结构体数组中的元素。
- 指针变量的基类型必须与结构体变量的类型相同。例如：

```
struct Student *pt;
```

结构体变量指针应用实例

例：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：**

◆在已有的基础上，本题要解决两个问题

- ●怎样对结构体变量成员赋值；
- ●怎样通过指向结构体变量的指针访问结构体变量中成员。

```
#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",stu_1.num);
    printf("name:%s\n",stu_1.name);
    printf("sex:%c\n",stu_1.sex);
    printf("score:%5.1f\n",stu_1.score);
    return 0;
}
```

(*p).num

(*p).name

(*p).sex

(*p).score

关于结构体变量指针的说明

- 为了使用方便和直观，C语言允许把(*p).num用p->num来代替
- (*p).name等价于p->name
- 如果p指向一个结构体变量stu，以下等价：

① stu.成员名(如stu.num)

② (*p).成员名(如(*p).num)

p->成员名(如p->num)

指向结构体数组的指针

例：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：**用指向结构体变量的指针处理

(1)声明struct Student，并定义结构体数组、初始化

(2)定义指向struct Student类型指针p

(3)使p指向数组首元素，输出元素中各信息

(4)使p指向下一个元素，输出元素中各信息

(5)再使p指向结构体数组的下一个元素，输出它指向的元素中的有关信息

```
#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 Fun",'M',19},
                {10104,"Wang Min",'F',20} };
```

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

```
No.  Name                    sex age
```

```
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;
}
```

10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

stu[0]
stu[1]
stu[2]

```
No.   Name                    sex age
10101 Li Lin                   M   18
```

```
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;
}
```

p →

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

```
 No.  Name                 sex  age
10101 Li Lin                M    18
10102 Zhang Fun             M    19
```

```
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;
}
```

p →

10101	Li Lin	M	18	stu[0]
10102	Zhang Fang	M	19	stu[1]
10104	Wang Min	F	20	stu[2]

```
 No.  Name                 sex  age
10101 Li Lin                M    18
10102 Zhang Fun             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;
}
```

p →

10101	Li Lin	M	18
10102	Zhang Fang	M	19
10104	Wang Min	F	20

stu[0]
stu[1]
stu[2]

结构体变量作函数参数

- 将一个结构体变量的值传递给另一函数，有3个方法

(1)用结构体变量的成员作参数。

例如，用stu[1].num作函数实参，将实参值传给形参。

```
printf("%d", stu[1].num);
```

◆用法和用普通变量作实参是一样的，属于“值传递”方式

◆应当注意实参与形参的类型保持一致。

(2)用指向结构体变量（或数组元素）的指针作参数，将结构体变量（或数组元素）的地址传给形参。

结构体变量作函数参数

- 将一个结构体变量的值传递给另一函数，有3个方法

（3）用结构体变量作实参。

- ◆用结构体变量作实参时，将结构体变量所占的内存单元的内容全部按顺序传递给形参，形参也必须是同类型的结构体变量
- ◆在函数调用期间形参也要占用内存单元。这种传递方式在空间和时间上开销较大
- ◆在被调用函数期间改变形参（也是结构体变量）的值，不能返回主调函数
- ◆一般较少用这种方法

结构体做函数参数的好处（1）

```
void judge(char* name, int h, int w)
{
  ……
}
int main()
{
  judge("XiaoMing", 179, 60);
  return 0;
}
```

一个人的信息却要三个参数

感觉就像

```
void judge(struct Person * pst)
{
    ……
}
int main()
{
  struct Person cand={"XiaoMing",179,60};
  judge(&cand);
  return 0;
}
```

其实一个参数就足够了

感觉就像

结构体做函数参数的好处（2）

```
void judge(char* name, int h, int w, int y, int m, int d)
{
  ……
}
int main()
{
  judge(“XiaoMing”, 179, 60, 1977, 11, 13);
  return 0;
}
```

增加一项生日信息，又多了三个参数！

```
void judge(struct gstudent pst)
{
    ……
}
int main()
{
  struct gstudent cand={……};
  judge(cand);
  return 0;
}
```

参数不需要改变

修改一下结构体定义就可以了

```
struct gstudent
{
   char name[20];
   int height;
   int weight;
   struct date birthday;
};
```

结构体变量作函数参数的应用

例：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(包括学号、姓名、3门课程成绩和平均成绩)。

结构体变量作函数参数的应用

- **解题思路：**将n个学生的数据表示为结构体数组。按照功能函数化的思想，分别用**3个函数**来实现不同的功能：
  - 用**input函数**输入数据和求各学生平均成绩
  - 用**max函数**找平均成绩最高的学生
  - 用**print函数**输出成绩最高学生的信息
  - 在主函数中先后调用这3个函数，用指向结构体变量的指针作实参。最后得到结果。
  - 本程序假设n = 3

```
#include <stdio.h>
#define N 3
struct Student
{
    int num;
    char name[20];
    float score[3];
    float aver;
};
```

4个成员

输入前3个成员值

计算最后成员值

```
int main()
{
    void input(struct Student stu[]);
    struct Student max(struct Student stu[]);
    void print(struct Student stud);
    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[0],&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;
    }
}
```

i=0

输入第1个成员值

输入第2个成员值

输入第3个成员值

计算第4个成员值

stu

10101	Li	78	89	98	88.33	stu[0]
						stu[1]
						stu[2]

```
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[0],&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;
    }
}
```

i=1

输入第1个成员值

输入第2个成员值

输入第3个成员值

计算第4个成员值

stu

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
						stu[2]

```
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[0],&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;
    }
}
```

i=2

输入第1个成员值

输入第2个成员值

输入第3个成员值

计算第4个成员值

stu

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
10106	Sun	88	76.5	89	84.5	stu[2]

```
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];
}
```

返回

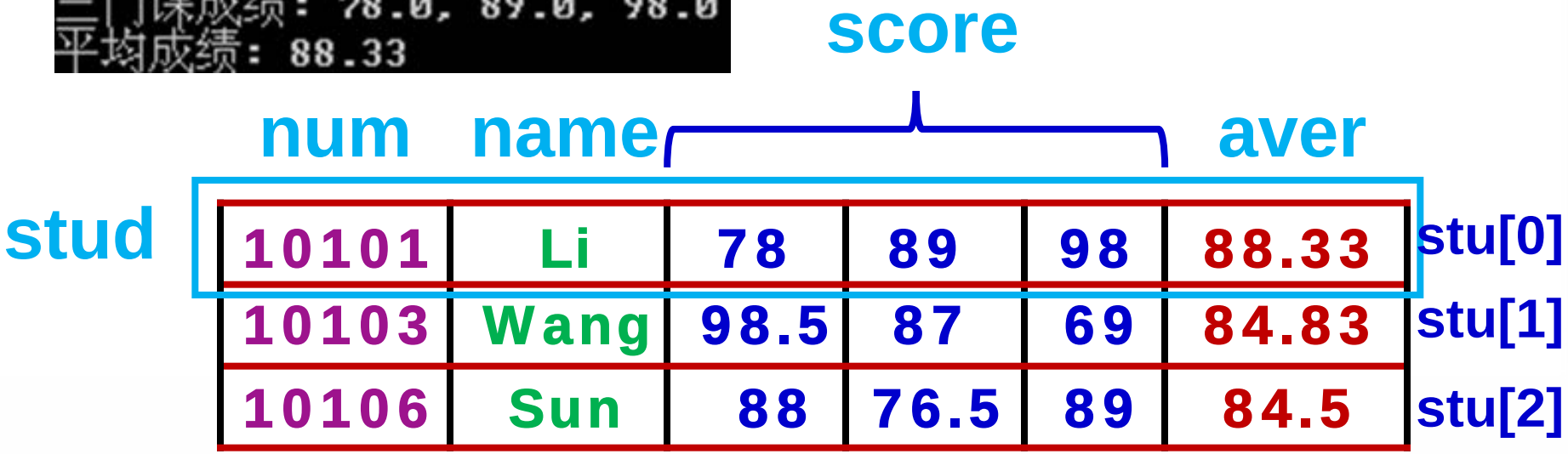
最大

stu

10101	Li	78	89	98	88.33	stu[0]
10103	Wang	98.5	87	69	84.83	stu[1]
10106	Sun	88	76.5	89	84.5	stu[2]

```
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);
}
```

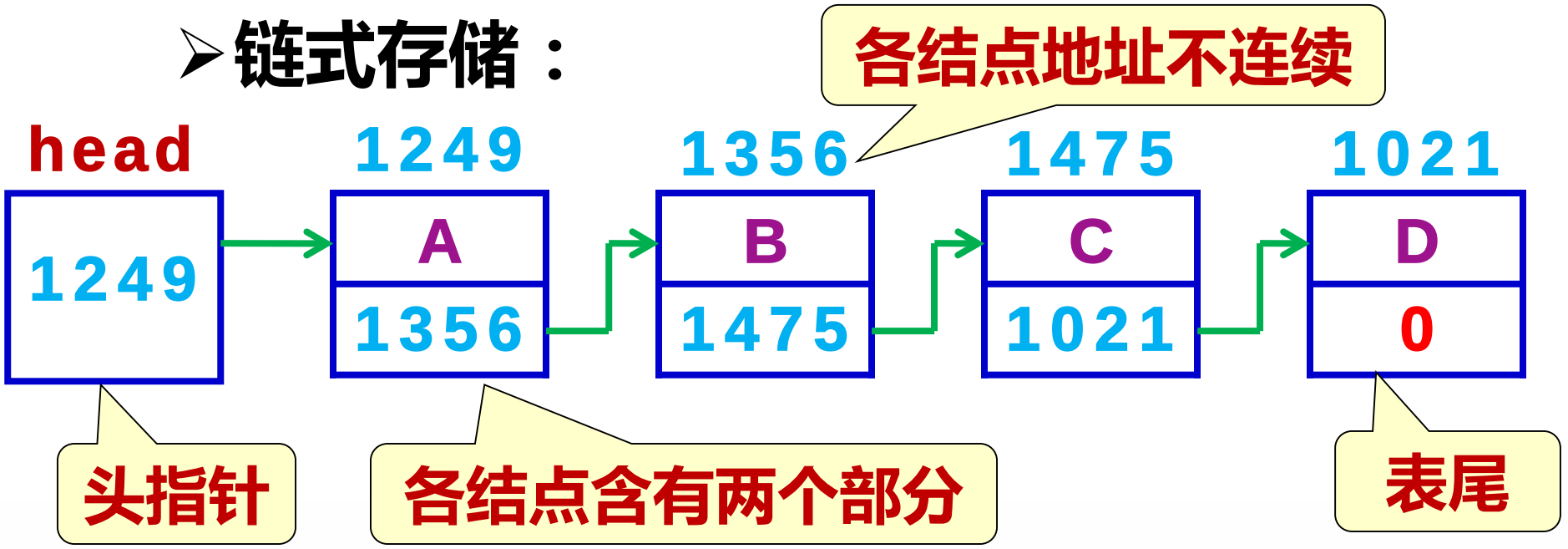
```
成绩最高的学生是:
学号:10101
姓名:Li
三门课成绩: 78.0, 89.0, 98.0
平均成绩: 88.33
```



- 以上3个函数的调用，情况各不相同：
  - ◆调用input函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数无返回值。
    - `void input(struct Student stu[]); input(p);`
  - ◆调用max函数时，实参是指针变量，形参是结构体数组，传递的是结构体元素的地址，函数的返回值是结构体类型数据。
    - `struct Student max(struct Student stu[]); max(p);`
  - ◆调用print函数时，实参是结构体变量，形参是结构体变量，传递的是结构体变量中各成员的值，函数无返回值。
    - `void print(struct Student stud); print(max(p));`

链表：非结构体不易实现

- 链表是一种常见的重要的数据结构
- 作为存储方式，它与顺序存储优劣互补
- 链式存储：



顺序存储的优点

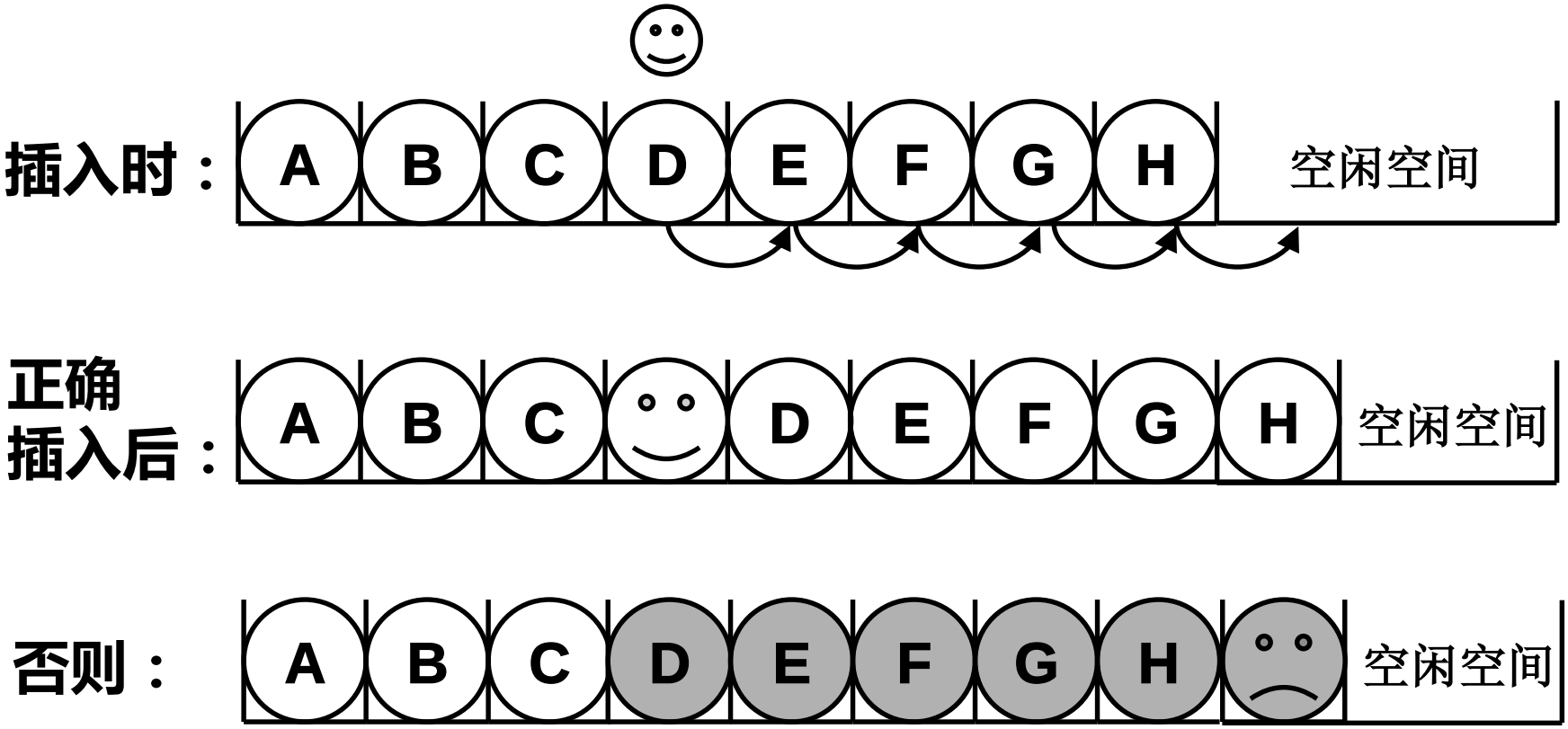
- 优点1：快速访问，知道元素的逻辑位置即可访问其内容
  - 例如，数组a，若从起始地址起每个位置都存有元素，则第(i+1)个元素的地址必然是 a+sizeof(a[i])*i
- 优点2：存储效率高，不需要额外空间表示元素之间的逻辑关系

顺序存储的缺陷

- 为了维持顺序存储快速访问的优点，需保证：

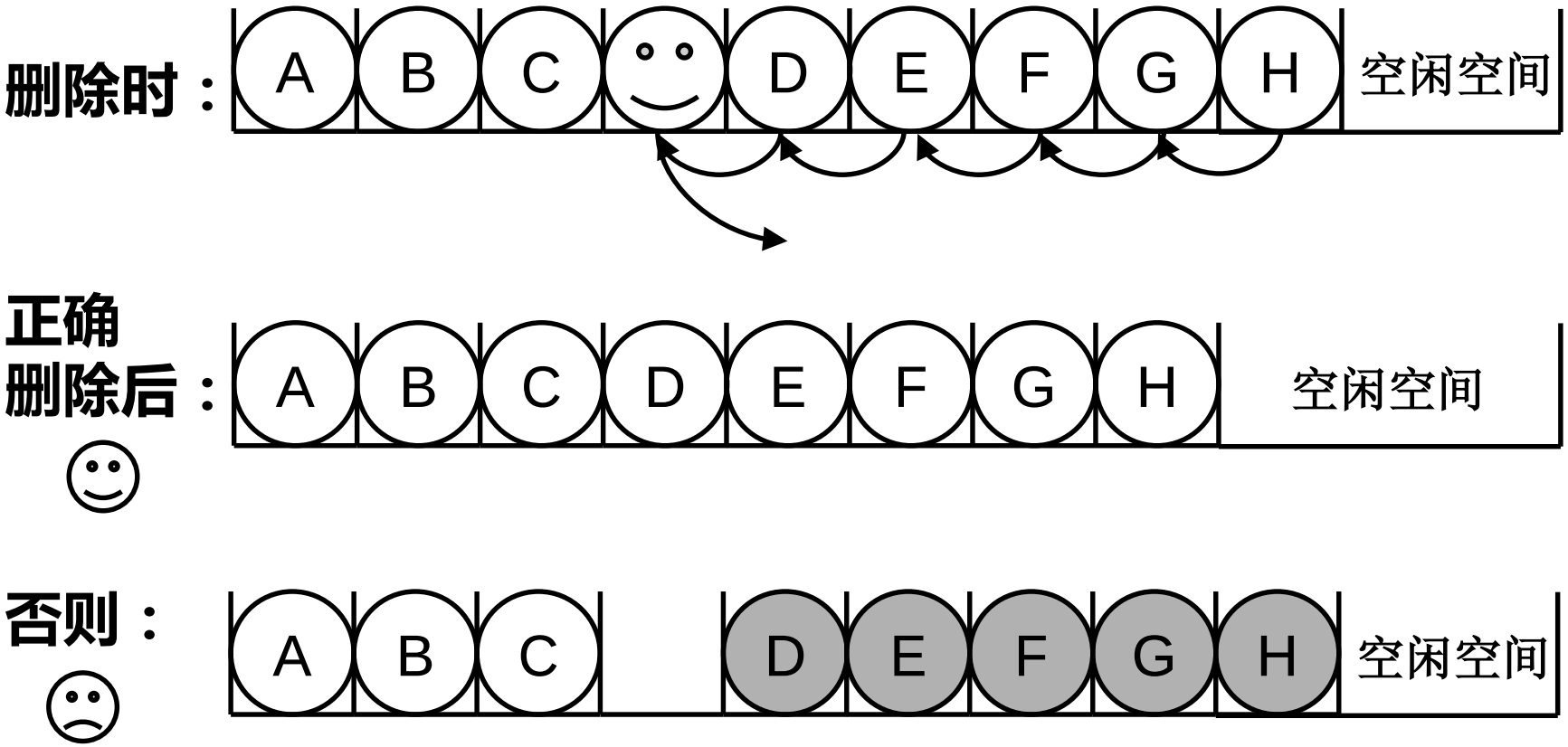
1. 连续存储。
   - 由此引发缺陷1：没有足够的连续空闲空间时怎么办？
2. 进行插入和删除操作时可能需移动大量元素以维持连续存储和正确的逻辑位置和存储位置的对应关系。
   - 由此引发缺陷2：插入和删除操作的时间开销很大。

顺序存储的插入操作



从i=4开始，第i个元素d的地址不再是a+sizeof(a[i])*(i-1)

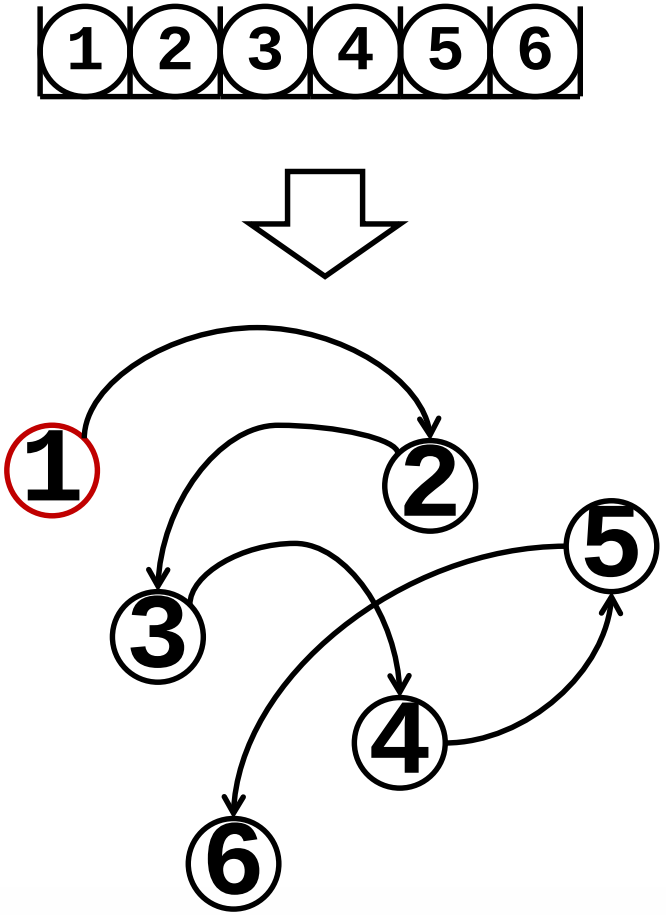
顺序存储的删除操作



从i=4开始，第i个元素d的地址不再是a+sizeof(a[i])*(i-1)

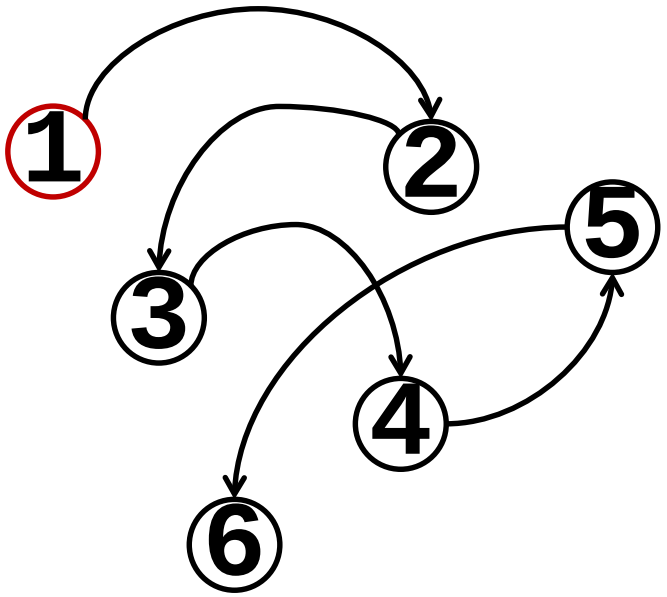
链式存储的思路

- 元素可以散落在任何位置，不必相邻
- 让每个元素知道它的下一个元素在哪里，我们只需要知道第一个元素的位置
- 插入删除操作不再需要移动元素而是需要修改元素间的关系



链式存储的优点

- 化整为零，不需要大段的连续空闲空间
- 插入删除操作没有移动元素的压力，效率很高



链式存储的缺陷

- 存放位置没有规律，元素之间单线联系，访问元素效率很低

当然，访问效率低不总是坏事...

- 每个元素都需要额外的存储空间存放下一个元素的地址

链表概述

- 链表是一种常见的线性数据结构。
  - 定义为一个结点序列，其中除了最后一个结点外的每个结点都包含有下一个结点的地址。
- 链表的变种有：单向链表、双向链表、循环链表等
- 我们仅以最简单的单向链表介绍链表的基本知识（此后链表均指单向链表）

单向链表与老鹰抓小鸡

- “母鸡”和“小鸡”的连接关系
  - “母鸡”冲在队伍最前面
  - 有只“小鸡”紧紧拉住“母鸡”
  - 后面的“小鸡”依次拉住前面的

- 链表结点的连接关系
  - 一个表头指针指向表头结点（“母鸡”）
  - 表头结点存有下一个结点地址（包含指针）并由此相连
  - 每个结点和它的下一个结点相连，直至表尾

作业 2017/12/18

- 按下列要求编写程序，提交手写源代码

1. 设有结构体struct dialing {char county[20], int code};用来记录国家名及其对应的代号（比如China对应86）。请实现（1）输入一个整数n，建立一个大小为n的struct dialing类型的数组，并依次输入n个国家的英文名和对应的代码；（2）实现一个函数，允许输入国家名，并在（1）建立的数组中查找该国家名（若仅大小写不同视为匹配）对应的代码，若找不到，提示”Not found”。