

规格严格 功夫到家



第12讲 结构体和共用体 及其应用

教材12.1~12.7节

MOOC第12周

哈尔滨工业大学

苏小红

sxh@hit.edu.cn

如何存储这些学生的信息？



100310121
100310122
100310123
100310124
.....

王刚
李小明
王丽红
陈莉莉
.....

'M'
'M'
'F'
'F'
.....

1991
1992
1991
1992
.....

72
88
98
87
.....

83
92
72
95
.....

90
78
89
78
.....

82
78
66
90
.....

内存分配不集中，结构零散，内存管理困难，寻址效率不高
对数组赋初值时，易发生错位



```
long  studentId[30] = {100310121, 100310122, 100310123, 100310124};  
char  studentName[30][10] = {"王刚", "李小明", "王丽红", "陈莉莉"};  
char  studentSex[30] = {'M', 'M', 'F', 'F'};  
int   yearOfBirth[30] = {1991, 1992, 1991, 1992};  
int   scoreMath[30] = {72, 88, 98, 87};  
int   scoreEnglish[30] = {83, 92, 72, 95};  
int   scoreComputer[30] = {90, 78, 89, 78};  
int   scoreProgramming[30] = {82, 78, 66, 90};
```



我们希望的内存分配图

相同类型的数据单独放在一起存储

100310121	王刚	'M'	1991
100310122	李小明	'M'	1992
100310123	王丽红	'F'	1991
100310124	陈莉莉	'F'	1992
.....

72	83	90	82
88	92	78	78
98	72	89	66
87	95	78	90
.....



逻辑相关但类型不同的数据放在一起存储

100310121	100310122	100310123	100310124
王刚	李小明	王丽红	陈莉莉
'M'	'M'	'F'	'F'
1991	1992	1991	1992
72	88	98	87
83	92	72	95
90	78	89	78
82	78	66	90



结构体类型

```
struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    int yearOfBirth;
    int score[4];
};
```

结构体——用户自定义的构造数据类型

- 不同语言定义不同的基本类型，但预定义的类型远远不够
 - * 试图定义较多类型，如数组、树、栈等，实践证明不是个好办法（如PL/1）
 - * 变形金刚之组合金刚中的每个机器人，它们都可独档一面，各有所长
- C语言的海陆空编队来了



- 有如“宝蓝兄弟”般个头小但能量大的基本数据类型
- 有如“大黑牛”般庞大而能力无限的数组
- 还有如“队长”般灵活多变、无限欢乐的指针

如何声明一个结构体类型？

```
struct student
```

```
{  
    long studentID;  
    char studentName[10];  
    char studentSex;  
    int  yearOfBirth;  
    int  score[4];  
};
```

```
struct student
```

```
{  
    long studentID;  
    char studentName[10];  
    char studentSex;  
    int  yearOfBirth;  
    int  score[4];  
};  
typedef struct student STUDENT;
```

结构体模板(Structure Template)

结构体标签(Structure Tag)

结构体成员(Structure Member)

编译器不为其分配内存

关键字**typedef**为已存在的类型定义一个别名

```
typedef unsigned short WORD;
```

```
typedef struct student  
{  
    long studentID;  
    char studentName[10];  
    char studentSex;  
    int  yearOfBirth;  
    int  score[4];  
}STUDENT;
```

结构体类型实例

```
typedef struct date
{
    int    year;
    int    month;
    int    day;
}DATE;
```

```
typedef struct date
{
    int    year;
    char   month[10];
    int    day;
}DATE;
```

```
typedef struct clock
{
    int    hour;
    int    minute;
    int    second;
}CLOCK;
```

```
typedef struct student
{
    long studentID;
    char  studentName[10];
    char  studentSex;
    DATE  birthday;
    int   score[4];
}STUDENT;
```

```
typedef struct _COORD
{
    short X; //horizontal coordinate
    short Y; //vertical coordinate
}COORD;
```

```
typedef struct rectangle
{
    COORD pt1;
    COORD pt2;
}RECTANGLE;
```

嵌套的结构体

这样就可对付更复杂的结构了

如何定义结构体变量？

```
typedef struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    int yearOfBirth;
    int score[4];
}STUDENT;
```

```
typedef struct date
{
    int year;
    int month;
    int day;
}DATE;
```

```
typedef struct date
{
    int year;
    char month[10];
    int day;
}DATE;
```

先定义类型再定义变量

```
typedef struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    DATE birthday;
    int score[4];
}STUDENT;
```

```
struct student stu1;
```

```
STUDENT stu1;
```

初始化列表中的**成员顺序**必须和结构体类型定义的**顺序一致**

```
STUDENT stu1 = {100310121, "王刚", 'M', 1991, {72,83,90,82}};
```

```
STUDENT stu1 = {100310121, "王刚", 'M', {1991,5,19}, {72,83,90,82}};
```

```
STUDENT stu1 = {100310121, "王刚", 'M', {1991,"May",19}, {72,83,90,82}};
```

```
STUDENT stu[30] = {{100310121, "王刚", 'M', {1991,5,19}, {72,83,90,82}},
                   {100310122, "李小明", 'M', {1992,8,20}, {88,92,78,78}},
                   {100310123, "王丽红", 'F', {1991,9,19}, {98,72,89,66}},
                   {100310124, "陈莉莉", 'F', {1992,3,22}, {87,95,78,90}}
};
```

让我们来创建一个更大的数据结构

//BMP图像信息头，大小40Bytes

typedef struct _tagBMP_INFOHEADER

{

unsigned long biSize;//结构总字节数4Bytes

long biWidth;//图像宽度4Bytes

long biHeight;//图像高度4Bytes

unsigned short biPlanes;//图像颜色平面数，始终为1， 2Bytes

unsigned short biBitCount;//图像像素位数，24表示真彩色， 2Bytes

unsigned long biCompression;//压缩方式，0表示不压缩， 4Bytes

unsigned long biSizeImage;//4字节对齐的图像大小，可以设置为0， 4Bytes

long biXPelsPerMeter;//水平分辨率， 4Bytes

long biYPelsPerMeter;//竖直分辨率， 4Bytes

unsigned long biClrUsed;//实际使用的调色板索引数（真彩色不需要）， 4Bytes

unsigned long biClrImportant;//重要的调色板索引数， 4Bytes

} BMP_INFOHEADER;

典型的BMP图像文件
由四部分组成：

1: 文件头

2: 图像参数

3: 调色板

4: 位图数据

结构就相当于其他语言中的类（class），但是添加不了方法

如何访问结构体成员？

■ 访问数组的元素

- 通过下标（位置）

```
int a[5];
```



a[0] a[1] a[2] a[3] a[4]

```
typedef struct _COORD
{
    short X;
    short Y;
}COORD;
COORD pos = {x, y};
//等价于
COORD pos;
pos.X = x;
pos.Y = y;
```

■ 访问结构体变量的成员

- 通过成员名和成员选择运算符

```
typedef struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    DATE birthday;
    int score[4];
}STUDENT;
STUDENT stu;
```



studentID studentName studentSex birthday score

```
stu.studentID = 100310121;
stu.studentName = "王刚";//???
strcpy(stu.studentName, "王刚");
```

讨论

- 两个结构体变量的成员同名是否会冲突？

```
struct student
{
    long studentID;           //学号
    char studentName[10];    //姓名
    char studentSex;          //性别
    int yearOfBirth;         //出生年
    int score[4];             //4门课的成绩
}stu1;
```

```
struct employee
{
    long employeeID;          //职工号
    char studentName[10];    //姓名
    char studentSex;          //性别
    int yearOfBirth;         //出生年
    int salary;               //工资
}ep1;
```



结构体变量能直接赋值吗？

```
typedef struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    DATE birthday;
    int score[4];
}STUDENT;

STUDENT stu1 = {100310121,"王刚",
                'M',{1991,5,19},
                {72,83,90,82}};

STUDENT stu2;
stu2 = stu1; ✓
```

```
stu2.studentID = stu1.studentID;
strcpy(stu2.studentName,stu1.studentName);
stu2.studentSex = stu1.studentSex;
stu2.birthday.year = stu1.birthday.year;
stu2.birthday.month = stu1.birthday.month;
stu2.birthday.day = stu1.birthday.day;
for (i=0; i<4; i++)
{
    stu2.score[i] = stu1.score[i];
}
```

对嵌套的成员，必须以**级联方式**访问

只能在**相同类型**的结构体变量之间进行赋值



数组能直接赋值吗？

```
int a[5] = {1,2,3,4,5};
```

```
int b[5];
```

```
b = a; ❌
```

```
typedef struct
```

```
{
```

```
    int member[5];
```

```
}ARRAY;
```

```
ARRAY a = {1,2,3,4,5};
```

```
ARRAY b;
```

```
b = a; ✅
```



把数组放到一个“空”的结构体内
封装以后，就可以直接复制数组了

结构体变量能进行比较操作吗？

```
COORD pos1, pos2;
```

```
.....
```

```
if (pos1 == pos2)
```

```
{
```

```
.....
```

```
}
```



为什么？

指向结构体变量的指针

- 如何定义指向结构体变量的指针？

```
STUDENT    stu1;  
STUDENT    *pt = &stu1;
```

- 通过成员选择运算符访问

```
stu1.studentID = 1;  
(*pt).studentID = 1;
```

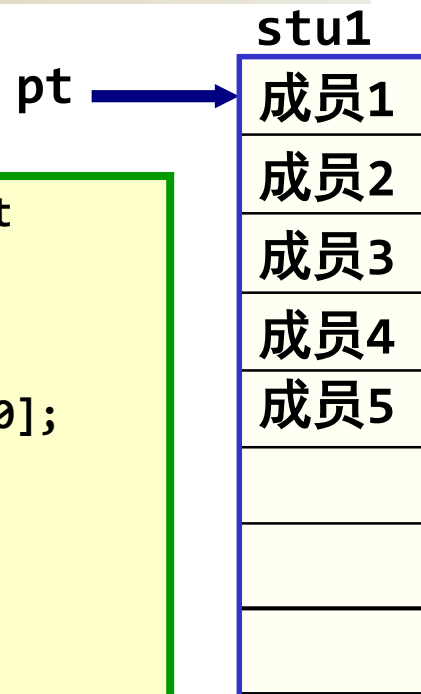
- 通过指向运算符访问

```
pt -> studentID = 1;
```

```
stu1. birthday. year = 1999;  
(*pt). birthday. year = 1999;  
pt -> birthday. year = 1999;
```

```
typedef struct student  
{  
    long studentID;  
    char studentName[10];  
    char studentSex;  
    DATE birthday;  
    int score[4];  
}STUDENT;
```

```
typedef struct date  
{  
    int year;  
    int month;  
    int day;  
}DATE;
```



指向结构体数组的指针

- 如何定义指向结构体数组的指针？

```
STUDENT stu[30];
```

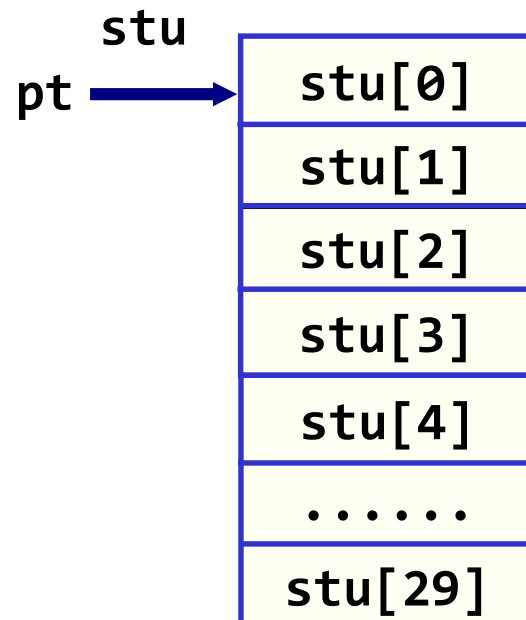
```
STUDENT *pt = stu; //&stu[0]
```

```
pt -> studentID
```

等价于 `(*pt).studentID`

`stu[0].studentID`

- `pt++`是什么意思？



```
typedef struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    DATE birthday;
    int score[4];
}STUDENT;
```

向函数传递结构体

传递结构体的 单个成员

- 复制单个成员的内容



偶是克隆的

传递结构体的 完整结构

- 复制结构体的所有成员



俺们全家都是克隆的

传递结构体的 首地址

- 仅复制一个地址值



悄悄告诉你俺家住哪里

计算时间差

- * 从键盘任意输入两个时间（例如4时55分和1时25分），计算并输出这两个时间之间的间隔。要求不输出时间差的负号。

```
typedef struct clock
{
    int hour;
    int minute;
    int second;
} CLOCK;
```



```

int main()
{
    CLOCK t1, t2, t3;
    printf("Input time one:(时, 分):");
    scanf("%d,%d", &t1.hour, &t1.minute);
    printf("Input time two:(时, 分):");
    scanf("%d,%d", &t2.hour, &t2.minute);
    t3 = CalculateTime(t1, t2);
    printf("%d小时%d分\n", t3.hour, t3.minute);
    return 0;
}

```

//函数功能：计算并返回两个时间t1和t2之间的差

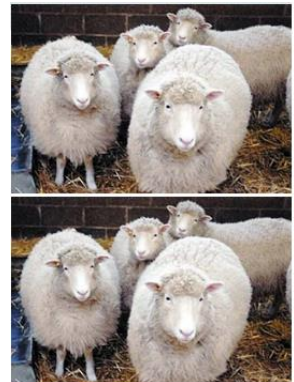
```

CLOCK CalculateTime(CLOCK t1, CLOCK t2)
{
    CLOCK t3;
    int time1, time2, time3;
    time1 = t1.hour * 60 + t1.minute; //转换成分
    time2 = t2.hour * 60 + t2.minute; //转换成分
    time3 = (int)fabs(time1 - time2);
    t3.minute = time3 % 60;
    t3.hour = time3 / 60;
    return t3;
}

```

把参数封装在结构中，函数接口更简洁，代码更稳定，可读性更好，可扩展性更好，增加参数时无需修改接口

传结构体变量，仅传副本，函数修改结构体内容不影响原结构体，因此需要从函数返回结构体变量，效率低



```
int main()
{
    CLOCK t1, t2, t3;
    printf("Input time one:(时, 分):");
    scanf("%d,%d", &t1.hour, &t1.minute);
    printf("Input time two:(时, 分):");
    scanf("%d,%d", &t2.hour, &t2.minute);
    CalculateTime(t1, t2, &t3);
    printf("%d小时%d分\n", t3.hour, t3.minute);
    return 0;
}
```

//函数功能：计算两个时间t1和t2之间的差，通过t3返回

```
void CalculateTime(CLOCK t1, CLOCK t2, CLOCK *t3)
{
    int time1, time2, time3;
    time1 = t1.hour * 60 + t1.minute; //转换成分
    time2 = t2.hour * 60 + t2.minute; //转换成分
    time3 = (int)fabs(time1 - time2);
    t3->minute = time3 % 60;
    t3->hour = time3 / 60;
}
```

向函数传需要
修改的结构体
变量的地址，
函数修改结构
体指针指向的
内容会影响原
结构体



```

int main()
{
    CLOCK t1, t2, t3;
    printf("Input time one:(时, 分):");
    scanf("%d,%d", &t1.hour, &t1.minute);
    printf("Input time two:(时, 分):");
    scanf("%d,%d", &t2.hour, &t2.minute);
    CalculateTime(&t1, &t2, &t3);
    printf("%d小时%d分\n", t3.hour, t3.minute);
    return 0;
}

```

向函数传结构体变量的地址，通过结构体指针返回结构体变量，效率高

```

//函数功能：计算两个时间t1和t2之间的差，通过t3返回
void CalculateTime(CLOCK *t1, CLOCK *t2, CLOCK *t3)
{
    int time1, time2, time3;
    time1 = t1->hour * 60 + t1->minute; //转换成分
    time2 = t2->hour * 60 + t2->minute; //转换成分
    time3 = (int)fabs(time1 - time2);
    t3->minute = time3 % 60;
    t3->hour = time3 / 60;
}

```



小结

■ 如何向函数传递结构体这样的大数据对象

向函数传递结构体的完整结构	向函数传递结构体的首地址
用结构体变量作函数参数	用结构体数组/结构体指针作函数参数
复制整个结构体成员的内容，一组数据	仅复制结构体的首地址，一个数据
参数传递直观，但开销大，效率低	参数传递效率高
函数内对结构内容的修改不影响原结构体	可修改结构体指针所指向的结构体的内容

复数运算

```
#include <stdio.h>
typedef struct complex
{
    int real; //实部
    int im;    //虚部
}COMPLEX;
```

E:\C\demo\bin\Debug\demo.exe

```
Input x+yi:3+4i
Input a+bi:5+6i
(3+4i)×(5+6i)=(-9+38i)
```

```
COMPLEX ComplexMultiply(COMPLEX za, COMPLEX zb);
void ComplexPrint(COMPLEX za, COMPLEX zb, COMPLEX zc);
int main()
{
    COMPLEX x, y, z;
    printf("Input x+yi:");
    scanf("%d+%di", &x.real, &x.im);
    printf("Input a+bi:");
    scanf("%d+%di", &y.real, &y.im);

    z = ComplexMultiply(x, y);

    ComplexPrint(x, y, z);
    return 0;
}
```

复数运算

```
COMPLEX ComplexMultiply(COMPLEX za, COMPLEX zb)
{
    COMPLEX zc;
    zc.real = za.real*zb.real - za.im*zb.im;
    zc.im   = za.real*zb.im + za.im*zb.real;
    return zc;
}
```

```
void ComplexPrint(COMPLEX za, COMPLEX zb, COMPLEX zc)
{
    printf("(%d+%di)*(%d+%di)=", za.real, za.im, zb.real, zb.im);
    printf("(%d+%di)\n", zc.real, zc.im);
}
```

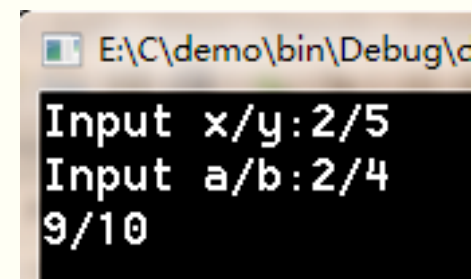
E:\C\demo\bin\Debug\demo.exe

```
Input x+yi:3+4i
Input a+bi:5+6i
(3+4i)×(5+6i)=(-9+38i)
```

有理数运算

- 两个有理数的加法、减法（需要通分）
- 有理数比较（通分后相减）
- 有理数约简（需要求最大公约数）

```
#include <stdio.h>
#include <stdlib.h>
typedef struct rational
{
    int numerator;    //分子
    int denominator; //分母
}RATIONAL;
```



```
E:\C\demo\bin\Debug\c
Input x/y:2/5
Input a/b:2/4
9/10
```

```
RATIONAL AddRational(RATIONAL a, RATIONAL b);
RATIONAL SimplifyRational(RATIONAL a);
int Gcd(int a, int b);
int main()
{
    RATIONAL x, y, z;
    printf("Input x/y:");
    scanf("%d/%d", &x.numerator, &x.denominator); //原样输入/
    printf("Input a/b:");
    scanf("%d/%d", &y.numerator, &y.denominator);
    z = AddRational(x, y);
    printf("%d/%d\n", z.numerator, z.denominator);
    return 0;
}
```

有理数运算

```
RATIONAL AddRational(RATIONAL a, RATIONAL b)
```

```
{
```

```
    RATIONAL c;
```

```
    c.numerator = a.numerator*b.denominator + a.denominator*b.numerator;
```

```
    c.denominator = a.denominator*b.denominator; //新的分母
```

```
    c = SimplifyRational(c);
```

```
    return c;
```

```
}
```

```
RATIONAL SimplifyRational(RATIONAL a)
```

```
{
```

```
    RATIONAL c;
```

```
    int divisor;//保存有理数a的分子和分母的最大公约数
```

```
    divisor = Gcd(abs(a.numerator), abs(a.denominator));//求分子和分母的gcd
```

```
    if (divisor > 0)
```

```
    {
```

```
        c.numerator = a.numerator / divisor;
```

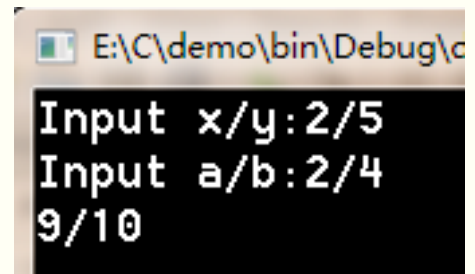
```
        c.denominator = a.denominator / divisor;
```

```
    }
```

```
    return c;
```

```
}
```

若函数返回值改为void,
如何返回结构体?



```
E:\C\demo\bin\Debug\c
Input x/y:2/5
Input a/b:2/4
9/10
```

结构体的排序和查找

- * 输入n个国家的国名及获得的金牌数， 然后输入一个国名， 查找其获得的金牌数。

```
char countryName[M][10];  
int goldMedal[M];
```



```
struct country  
{  
    char name[10];  
    int goldMedal;  
};  
struct country countries[M];
```



结构体的排序

```
struct country
{
    char name[10];
    int  goldMedal;
};
```

//按国名字典顺序排序

```
void SortString(struct country c[], int n)
{
    int  i, j, t;
    char temp[N];
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp(c[j].name, c[i].name) < 0)
            {
                strcpy(temp, c[i].name);
                strcpy(c[i].name, c[j].name);
                strcpy(c[j].name, temp);
                t = c[i].goldMedal;
                c[i].goldMedal = c[j].goldMedal;
                c[j].goldMedal = t;
            }
        }
    }
}
```

```
.....
int main()
{
    int  i, n;
    struct country countries[M];
    printf("How many countries?");
    scanf("%d",&n);
    printf("Input names and goldmedals:\n");
    for (i=0; i<n; i++)
    {
        scanf("%s%d", countries[i].name,
              &countries[i].goldMedal);
    }
    SortString(countries, n);
    printf("Sorted results:\n");
    for (i=0; i<n; i++)
    {
        printf("%s:%d\n", countries[i].name,
              countries[i].goldMedal);
    }
    return 0;
}
```

//按国名字典顺序排序

```
void SortString(struct country c[], int n)
{
    int i, j;
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp(c[j].name, c[i].name) < 0)
            {
                SwapChar(c[i].name, c[j].name);
                SwapInt(&c[i].goldMedal, &c[j].goldMedal);
            }
        }
    }
}
```

//按国名字典顺序排序

```
void SortString(struct country *p, int n)
{
    int i, j;
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp((p+j)->name, (p+i)->name) < 0)
            {
                SwapChar((p+i)->name, (p+j)->name);
                SwapInt(&(p+i)->goldMedal, &(p+j)->goldMedal);
            }
        }
    }
}
```



结构体的排序

```
void SwapInt(int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}

void SwapChar(char *x, char *y)
{
    char t[N];
    strcpy(t, x);
    strcpy(x, y);
    strcpy(y, t);
}
```

```
struct country
{
    char name[10];
    int goldMedal;
};

struct country countries[M];
```

结构体的排序

//按国名字典顺序排序

```
void SortString(struct country c[], int n)
{
    int i, j, t;
    char temp[N];
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp(c[j].name, c[i].name) < 0)
            {
                strcpy(temp, c[i].name);
                strcpy(c[i].name, c[j].name);
                strcpy(c[j].name, temp);
                t = c[i].goldMedal;
                c[i].goldMedal = c[j].goldMedal;
                c[j].goldMedal = t;
            }
        }
    }
}
```

//按国名字典顺序排序

```
void SortString(struct country c[], int n)
{
    int i, j;
    struct country temp;
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp(c[j].name, c[i].name) < 0)
            {
                temp = c[i];
                c[i] = c[j];
                c[j] = temp;
            }
        }
    }
}
```



结构体的排序

//按国名字典顺序排序

```
void SortString(struct country c[], int n)
{
    int i, j;
    struct country temp;
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp(c[j].name, c[i].name) < 0)
            {
                temp = c[i];
                c[i] = c[j];
                c[j] = temp;
            }
        }
    }
}
```

//按国名字典顺序排序

```
void SortString(struct country c[], int n)
{
    int i, j;
    struct country temp;
    for (i=0; i<n-1; i++)
    {
        for (j=i+1; j<n; j++)
        {
            if (strcmp(c[j].name, c[i].name) < 0)
            {
                SwapStruct(&c[i], &c[j]);
            }
        }
    }
}
```

```
void SwapStruct(struct country *x, struct country *y)
{
    struct country t;
    t = *x;
    *x = *y;
    *y = t;
}
```



结构体的查找

```
int main()
{
    int    i, n, pos;
    struct country countries[M];
    char s[N];
    printf("How many countries?");
    scanf("%d",&n);
    printf("Input names and goldMedals:\n");
    for (i=0; i<n; i++)
    {
        scanf("%s%d", countries[i].name,
               &countries[i].goldMedals);
    }
    printf("Input the searching country:");
    scanf("%s", s);
    pos = SearchString(countries, n, s);
    if (pos != -1)
    {
        printf("%s:%d\n", s,
               countries[pos].goldMedals);
    }
    else
    {
        printf("Not found!\n");
    }
    return 0;
}
```

```
struct country
{
    char name[10];
    int goldMedal;
};
struct country countries[M];
```

```
int SearchString(struct country countries[],
                 int n, char name[])
{
    int    i;
    for (i=0; i<n; i++)
    {
        if (strcmp(name, countries[i].name)==0)
        {
            return i;
        }
    }
    return -1;
}
```



课后作业：手机通讯录

- * **通讯录管理系统**中，通常会提供查找联系人手机号的功能，请编程输入n个人的姓名及手机号码，然后输入一个人的姓名，查找该人对应的手机号。

```
char name[N][20];  
char telno[N][20];
```



```
struct contacts  
{  
    char name[20];  
    char telno[20];  
};  
struct contacts b[N];
```



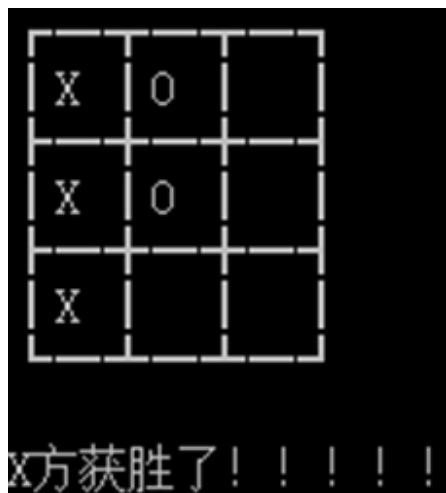
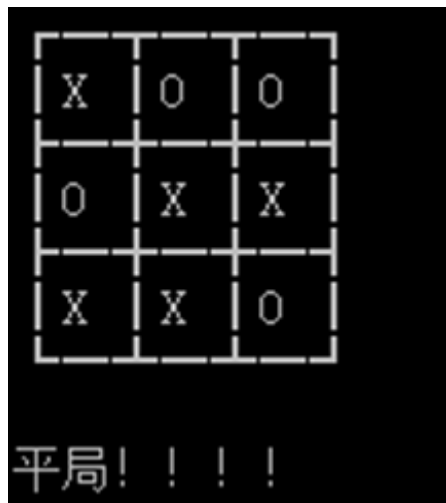
COORD类型

- COORD是Windows API中定义的一种结构类型
 - * 表示一个字符在控制台屏幕上的坐标

```
#include <stdio.h>
#include <windows.h>
void Gotoxy(int x, int y)
{
    COORD pos = {x, y};
    HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE); //获取标准输出设备的句柄
    SetConsoleCursorPosition(hOutput, pos); //定位光标位置为pos
}
int main()
{
    Gotoxy(2, 2);
    printf("Hello World!\n");
    system("pause");
    return 0;
}
```

```
typedef struct _COORD
{
    short X; //horizontal coordinate
    short Y; //vertical coordinate
}COORD; //在windows.h中定义
```

双人对弈井字棋

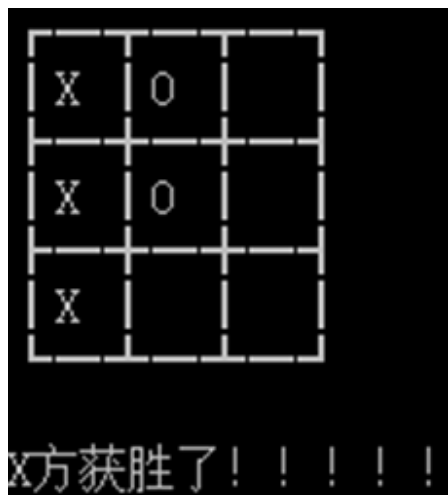
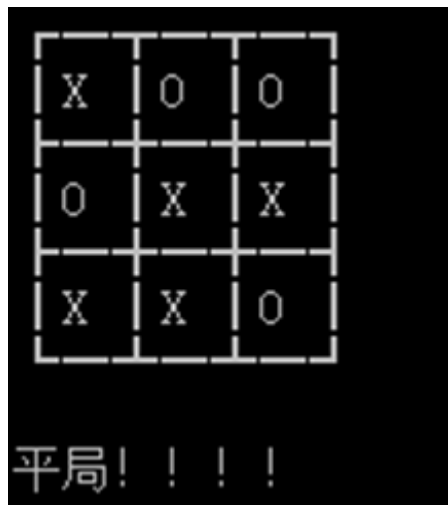


- 游戏规则：X方先行，有一方棋子三个连成一条直线即为胜出，棋盘上9个棋盘格子落子满，且没有一方三子连线，则为平局

```
int main()
{
    char board[ROWS][COLS] = { 0 };
    init_board(board); //初始化棋盘
    print_board(board); //打印棋盘
    play_game(board); //玩家循环落子，重绘棋盘，判断胜负
    return 0;
}
```

双人对弈井字棋

- 棋盘边界符号为搜狗输入法中的制表符



双人对弈五子棋

0=黑色 1=蓝色 2=绿色 3=湖蓝色 4=红色 5=紫色 6=黄色 7=浅灰色
8=灰色 9=淡蓝色 A=淡绿色 B=淡浅绿色 C=淡红色 D=淡紫色 E=淡黄色 F=亮白色

```
int main()
{
    system("title 翻转五子棋 一张佳乐");//设置标题
    system("mode con cols=73 lines=36");//设置窗口大小
    system("color 70");//设置颜色
    while(1) //循环执行游戏
    {
        RunGame();
    }
}
```

用户自定义的数据类型

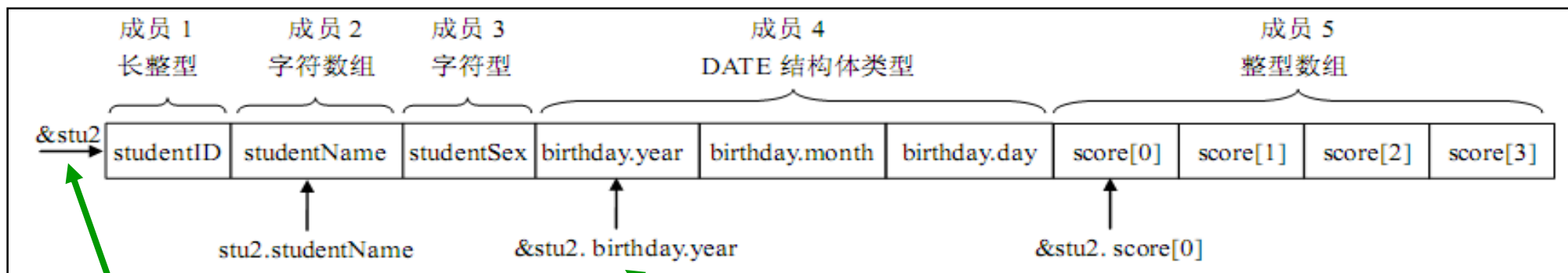
结构体，也称结构 (struct)

- 把关系紧密且逻辑相关的多种不同类型的变量，组织到一个统一的名字之下

共用体，也称联合 (union)

- 把情形互斥但逻辑相关的多种不同类型的变量，组织到一个统一的名字之下

结构体变量的取地址值操作

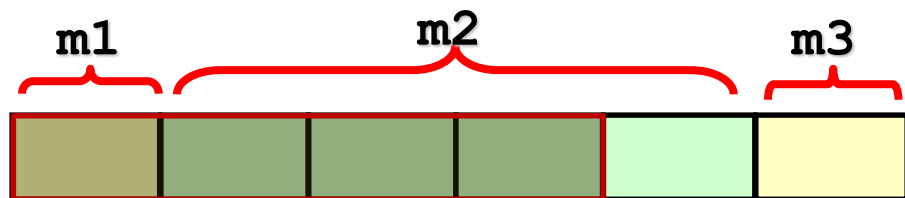


结构体变量的地址
&stu2是该变量所占
内存空间的首地址

结构体**成员**的地址与该成
员在结构体中所处的位置
及其所占内存字节数相关

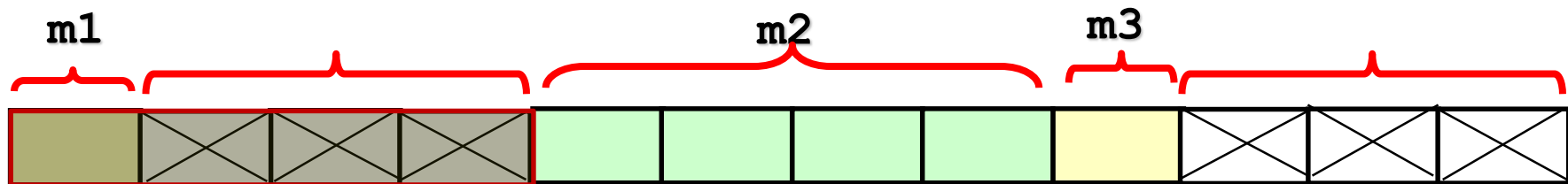
注意：这两种指针的**基类型**不同哦！

如何计算结构体占内存的字节数？



是所有成员占内存字节数的总和吗？
这样存储有什么问题吗？

```
typedef struct
{
    char m1;
    int  m2;
    char m3;
} SAMPLE;
```

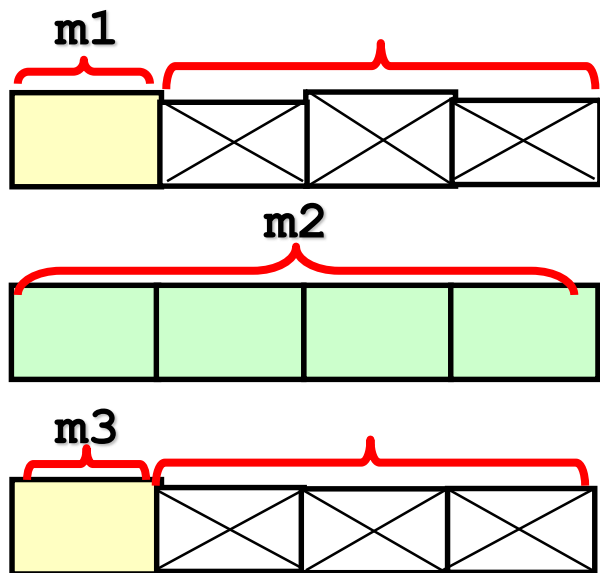


- 内存对齐（Memory-alignment）
 - 为提高内存寻址效率，多数计算机要求从某个数量字节的倍数开始存放数据
 - 需在较小的成员后加入补位
- 会不会在结构体的开始处出现补位？



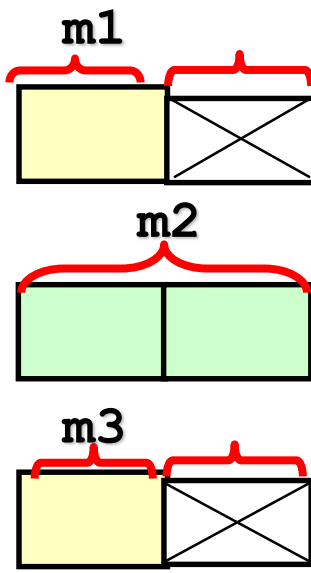
如何计算结构体占内存的字节数？

```
typedef struct sample
{
    char m1;
    int m2;
    char m3;
}SAMPLE;
SAMPLE s;
```

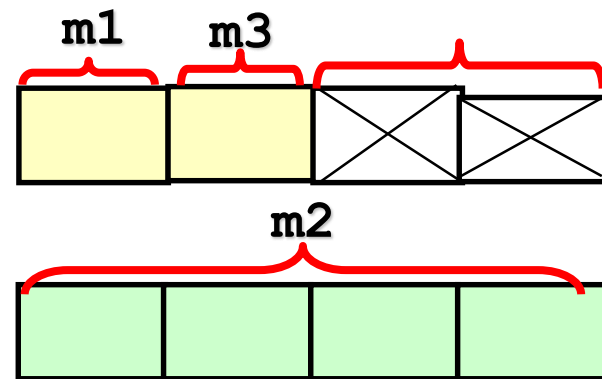


```
printf("%d\n", sizeof(SAMPLE));
printf("%d\n", sizeof(s));
```

```
typedef struct sample
{
    char m1;
    short m2;
    char m3;
}SAMPLE;
SAMPLE s;
```



```
typedef struct sample
{
    char m1;
    char m3;
    int m2;
}SAMPLE;
SAMPLE s;
```



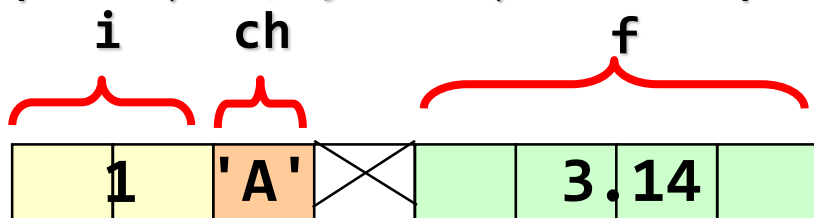
- 结构体在内存中所占的字节数不仅与系统有关，还与结构体类型的定义有关

共用体 vs 结构体

```
struct sample
{
    short    i;
    char     ch;
    float    f;
};
```

```
struct sample s;
s.i = 1;
s.ch = 'A';
s.f = 3.14;
```

```
printf("%d\n", sizeof(struct sample));
```

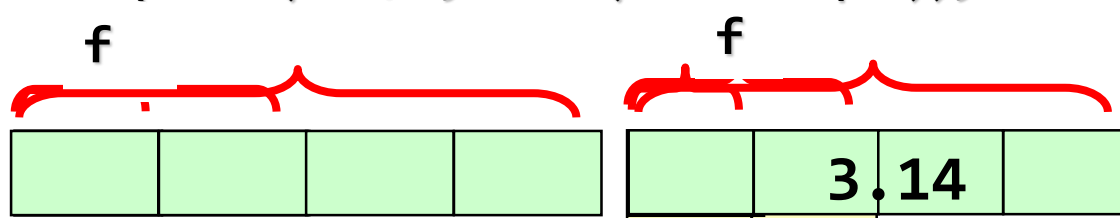


```
struct sample s = {1, 'A', 3.14};
```

```
union sample
{
    short    i;
    char     ch;
    float    f;
};
```

```
union sample u;
u.i = 1;
u.ch = 'A';
u.f = 3.14;
```

```
printf("%d\n", sizeof(union sample));
```



字段共享内存，占空间大小取决于**占空间最多**的那个成员，共用体不能比较

每一瞬时只保存一个成员(**最后一次赋值的**)

C89规定只能对第一个成员初始化

C99允许按名设置成员初值

指定初始化器，能提高可读性

```
union sample u = {1};
```

```
union sample u = {.ch='a'};
```

```
struct sample u = {.i=1, .ch='a'};
```

同一类事物，状态互斥

姓名	性别	年龄	婚姻状况					婚姻状况 标记	
			未婚	已婚			离婚		
				结婚日期	配偶姓名	子女数量	离婚日期		子女数量

```
struct person
{
    char name[20];
    char sex;
    int age;
    union maritalState marital;
    int marryFlag; //婚姻状态标记
};
```

```
struct marriedState
{
    struct date marryDay;
    char spouseName[20];
    int child;
};
```

```
struct divorceState
{
    struct date divorceDay;
    int child;
};
```

```
union maritalState
{
    int single; /*未婚*/
    struct marriedState married; /*已婚*/
    struct divorceState divorce; /*离婚*/
};
```

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

同一类事物，状态互斥

姓名	性别	年龄	婚姻状况						婚姻状况 标记
			未婚	已婚			离婚		
				结婚日期	配偶姓名	子女数量	离婚日期	子女数量	

```
struct person
{
    char name[20];
    char sex;
    int age;
    union maritalState marital;
    int marryFlag; //婚姻状态标记
};
struct person p1;
```

为共用体添加标记字段

每次对共用体的成员赋值时，程序负责
改变标记的内容

关键问题：如何知道共用体中当前
起作用的成员是哪一个？

```
if (p1.marryFlag == 1)
{
    //未婚
}
else if (p1.marryFlag == 2)
{
    //已婚
}
else
{
    //离婚
}
```

同一类事物，状态互斥

```
struct person
{
    char name[20];
    char sex;
    int age;
    union maritalState marital;
    int marryFlag; //婚姻状态标记
};

struct person p1;
```

用**枚举类型**声明结构体中的标记字段
枚举类型 (Enumeration) : 当某些量仅由有限个整型数据值组成时

```
struct person
{
    char name[20];
    char sex;
    int age;
    union maritalState marital;
    enum {SINGLE, MARRIED, DIVORCE} marryFlag;
};

struct person p1;
```

值为0 值为1 值为2

同一类事物，不同类型



3个苹果：个数

int



1斤葡萄：重量

float



0.5升苹果：容积

float

数据类型不同，但都表示量

```
typedef struct
{
    int count;        //个数
    float weight;     //重量
    float volume;     //容积
}QUANTITY;
```



```
typedef union
{
    int count;        //个数
    float weight;     //重量
    float volume;     //容积
}QUANTITY;
```

允许以多种
类型来引用
一个对象

同一类事物，不同类型

```
typedef union
{
    int count;
    float weight;
    float volume;
}QUANTITY;
```



3个苹果：个数

int



1斤葡萄：重量

float



0.5升苹果：容积

float

```
typedef struct
{
    const char *name;        //名字
    const char *country;     //产地
    QUANTITY amount;         //量
}FRUIT;
```



```
typedef struct
{
    const char *name;        //名字
    const char *country;     //产地
    QUANTITY amount;         //量
    UNIT_OF_MEASURE units;
}FRUIT;
```

```
typedef enum {COUNT, POUNDS, LITRE}UNIT_OF_MEASURE;
```

共用体的主要应用

- 应用
 - * 有效使用存储空间
 - * 构造混合的数据结构
- 假设需要的数组元素是int型和float型数据的混合

```
typedef union  
{  
    int    i;  
    float  f;  
}NUMBER;
```

```
NUMBER array[100];
```

```
array[0].i = 10;
```

```
array[1].f = 3.14;
```

每个NUMBER类型的数组array的数组元素都有两个成员，既可以存储int型数据，也可以存储float型数据

测试你的机器是大端还是小端？

■ 如何存储一个多字节整数？



■ **小端次序：**便于计算机从低位字节向高位字节运算

■ **大端次序：**与人的书写顺序相同，便于处理字符串



小端次序 (Little-endian)

```
Memory
Address: &c Bytes: 64 Go
(e.g. 0x401060, or &variable, or $eax)
0x28ff04: 03 00 00 00 02 00 00 00|01 00 00 00 1e 00 00 00 .....
0x28ff14: 02 00 00 00 94 ff 28 00|fd 10 40 00 01 00 00 00 ....ÿ(.ý.@.....
0x28ff24: 58 16 9f 00 f0 18 9f 00|00 50 40 00 50 ff 28 00 X..8...P@.Pÿ(.
0x28ff34: ff ff ff ff 54 ff 28 00|20 72 8b 75 d0 de b4 21 ŸÿÿÿTÿ(. ruÐP'!
```



大端次序 (Big-endian)

第1种测试方法

```
#include <stdio.h>
int CheckEndian(int *a)
{
    //用指针p强制让它指向a的低地址，才能知道p拿到的是a的低位还是高位
    char *p = (char*)a;
    if (*p == 1)
    {
        printf("你的机器是 Little-endian");
    }
    else
    {
        printf("你的机器是 Big-endian");
    }
    return 0;
}
int main()
{
    int a = 1;

    CheckEndian(&a);

    return 0;
}
```

第2种测试方法

```
#include <stdio.h>

union A
{
    char c;
    int a;
}sample;

int main()
{
    sample.a = 1;    //共用体占用的内存大小是最大元素所占内存大小
    if (sample.c == 1)
    {
        printf("你的机器是 Little-endian");
    }
    else
    {
        printf("你的机器是 Big-endian");
    }
    return 0;
}
```

小结

■ 两种新的数据类型

结构体（struct）	共用体（union）
关系紧密且逻辑相关的多种不同类型的数据的集合	情形互斥但逻辑相关的多种不同类型的数据的集合
可以做函数参数和返回值	不能做函数参数，不能进行比较操作
各成员占相邻的存储单元，用 <code>sizeof</code> 来计算占用内存的总字节数	每一瞬时只能存其中一种类型的成员 最后一次赋值的成员起作用
对所有成员初始化	只能对第一个成员初始化