
step further

专题

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与封装（模块设计方法、函数）

表达与转换（基本操作、数据类型）

提高：

构造与访问（数组、指针、**结构**）

归纳与推广（程序设计的本质）

● 结构的基本概念

- ◆ 结构类型的构造
- ◆ 结构体的定义与初始化
- ◆ 结构体的操作
 - 含有指针成员的结构体及其操作

● 用指针操纵结构体

- ◆ 用指针操纵含有指针成员的结构体

问题的提出

● 描述一位学生的信息：

| num | name | age | score | addr |
|-----------|------|-----|-------|---------|
| 211220888 | Hans | 19 | 86.5 | Nanjing |

● 数组？

（数组类型用于表示 固定多个 **同类型** 的数据群体）

结构类型的构造与结构体的定义

- 构造一个结构类型并命名

```
struct Student
```

```
{
```

```
    int number;    //成员
```

```
    char name;    //成员
```

```
    int age;    //成员
```

```
};
```

宣布组成的
成员名称和成员类型

- 可以用构造好的结构类型名字来定义结构体

```
struct Student s1, s2;
```

- 也可以在构造结构类型的同时直接定义结构体，此时可以不给结构类型命名

```
struct Student
```

```
{
```

```
    int number;
```

```
    char name;
```

```
    int age;
```

```
} s1, s2;
```

```
struct
```

```
{
```

```
    int number;
```

```
    char name;
```

```
    int age;
```

```
} s1, s2;
```

结构类型的别名

struct Student

```
{  
    int number;    //成员  
    char name;     //成员  
    int age;       //成员  
};
```

```
struct Student s1, s2;
```

```
Student s1, s2;
```

结构体定义中，有的开发环境可以省略 struct

typedef struct Student

```
{  
    int number;  
    char name;  
    int age;  
} Student;
```

用 typedef 给构造的类型 指定的别名来定义结构体，
则不加 struct，且跟开发环境无关

```
Student s1, s2;
```

结构类型名字的（隐含）名空间与作用域

```
typedef struct
{
    int number;    //成员
    char name;     //成员
    int age;       //成员
} Student;
```

结构类型与其他变量或函数可重名；
结构类型的成员与其他变量或函数也可以重名；
结构类型与它的成员也可以重名；

不同的结构类型不能重名；
同一个结构体的各个成员不能重名。

结构类型标识符是一种标签tag，在单独的名空间里

如果在函数体里面构造结构类型，则该函数之外此结构类型名字不可用。
一般把结构类型的构造放在文件头部，也可以把结构类型的构造放在头文件中。

- 即使两个结构类型中成员的类型、名称、顺序都完全一致，它们也是不同的结构类型。

```
struct
{
    int number;
    char name;
    int age;
} s1 ;
```

```
struct
{
    int number;
    char name;
    int age;
} s2 ;
```

s1与 s2 类型不同

不同结构类型的成员可重名

结构成员的类型

- 构造结构类型时，花括号中至少要定义一个成员。
- 除void类型和本结构类型外，结构成员可以是其他任意的类型。

```
struct Employee
{
    int number;
    char name;
    struct Date
    {
        int    year;
        int    month;
        int    day;

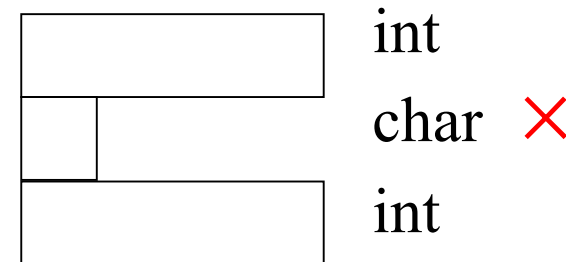
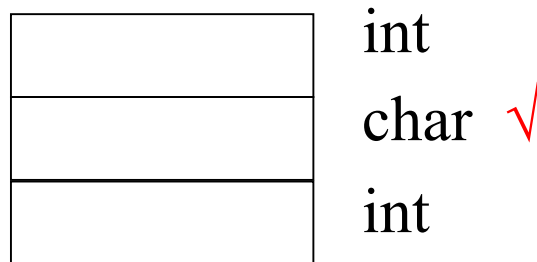
        } birthday ; // 其他类型的结构体可以作为本结构类型的成员
    } e ;
```

```
typedef struct
{
    int    year;
    int    month;
    int    day;
}Date;
struct Employee
{
    int number;
    char name;
    Date birthday;
} e ;
```


结构体的存储

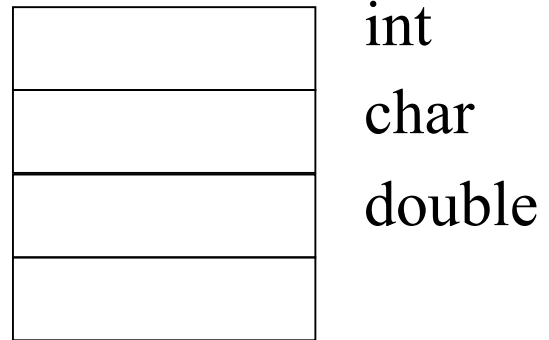
- 系统为结构体在内存栈区分配空间
 - 按构造时的顺序依次给各个成员分配空间
 - 一般以**字**为单位给结构体分配空间（对齐原理）

```
struct Student
{
    int number;
    char name;
    int age;
} s ;
```



- 定义结构体时一般不加register修饰

```
struct Student
{
    int number;
    char name;
    double score;
} s ;
```



结构体的初始化

❁ 可以在**定义结构体**的同时，给各个成员赋值，即结构体的初始化。

◆ 比如，

```
Student s1, s2 = {1220999, 'T', 19};
```

```
Employee e = {1160007, 'J', {1996, 12, 26}};
```

◆ **注意：在构造一个结构类型时，不能对其成员进行初始化。**比如，

```
typedef struct
```

```
{
```

```
    int number = 1220999;
```

```
    char name;
```

```
    int age;
```

```
}Student;
```

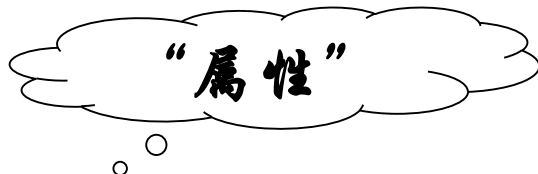
C++11新标准允许给一个默认值

// 此处的初始化是错误的

● 结构与数组 这两种类型的异同点？

◆ 结构类型：

- 固定多个
- 类型可以不同的成员所构成的数据群体（含义可以不同的相关信息）
- 成员间在逻辑上没有先后次序关系，其说明次序仅影响成员的存储安排，不影响操作，成员有成员名
- 相当于其他高级语言中的“记录”



◆ 数组类型：

- 固定多个
- 同类型（相同意义的相关信息）
- 元素间在逻辑上有先后次序关系，按序连续存储，元素有下标

结构体的操作

对结构体的操作是通过**成员操作符**操作结构体的成员完成的：

◆ `<结构体名> . <成员名>`

`e.number = 0108001;`

◆ 点号是成员操作符，它是双目操作符，具有1级优先级，结合性为自左向右。

◆ 对成员的访问不依靠次序，按名称访问，
所以，构造结构类型时成员的书写次序无关紧要。

◆ 若某成员类型是另一个结构类型，
则用若干个成员操作符访问最低一级的成员。比如，
`e.birthday.year = 1996;`

```
typedef struct
{   int    year;
    int    month;
    int    day;
}Date;
struct Employee
{   int number;
    char name;
    Date birthday;
} e ;
```

赋值操作

- 相同结构类型的不同结构体之间**可以直接相互赋值**，其实质是两个结构体的存储空间中的所有成员数据直接拷贝。比如，

```
Employee e1, e2;
```

```
e1 = e2;
```

```
typedef Employee Employ
```

```
Employee e1;
```

```
Employ e2;
```

```
e1 = e2;
```

- 不同结构类型的结构体之间不能相互赋值，

下面a、b两个结构体
不可以相互赋值：

```
struct  
{  
    int no;  
    char name;  
} a ;
```

```
struct  
{  
    int no;  
    char name;  
} b ;  
a = b; ❌
```

结构体作为函数参数/返回值

- 可作为参数传给函数：默认参数传递方式为**值传递**
（实参和形参都是结构体名，类型相同；但实参和形参代表两个不同的结构体，运行时分配不同的存储空间。）
- 函数也可以返回一个
结构体（结构类型函数）

例8.1 验证结构体的值传递方式。

```
void MyFoo(Student s);
```

```
int main( )
```

```
{
```

```
    Student s;
```

```
    s.number = 1220999;
```

```
    s.name = 'T';
```

```
    s.age = 18;
```

```
    printf("%d %c %d\n", s.number, s.name, s.age);
```

```
    MyFoo(s);
```

```
    printf("%d %c %d\n", s.number, s.name, s.age);
```

```
    return 0;
```

```
}
```

```
void MyFoo(Student s)
```

```
{    s.age += 1;
```

```
    printf("%d %c %d\n", s.number, s.name, s.age);
```

```
}
```

程序结果会显示:

```
1220999 T 18
```

```
1220999 T 19
```

```
1220999 T 18
```


例8.2 验证结构体可以作为返回值。

```
#include <stdio.h>
typedef struct
{
    int x; //描述横坐标
    int y; //描述纵坐标
}Point;    //描述点

typedef struct
{
    Point pt1; //描述左下角顶点
    Point pt2; //描述右上角顶点
}Rect;      //描述矩形

bool PtInRect(Point, Rect);
    //判断一个点是否在一个矩形内

Rect CanonRect(Rect);
    //规范化矩形坐标
```

```
int main( )
{
    Rect r1;
    printf("输入矩形左下角顶点的坐标: \n");
    scanf("%d%d", &r1.pt1.x, &r1.pt1.y);
    printf("输入矩形右上角顶点的坐标: \n");
    scanf("%d%d", &r1.pt2.x, &r1.pt2.y);
    Rect r2 = CanonRect(r1);
        //结构体r2获得函数返回值

    Point pt;
    printf("输入一个点的坐标: \n");
    scanf("%d%d", &pt.x, &pt.y);
    if( PtInRect(pt, r2) )
        printf("该点在矩形内 \n");
    else
        printf("该点不在矩形内 \n");
    return 0;
}
```

```
#define min(x, y) ((x) < (y) ? (x) : (y))
#define max(x, y) ((x) > (y) ? (x) : (y))
Rect CanonRect(Rect r)
{
    Rect temp;
    temp.pt1.x = min(r.pt1.x, r.pt2.x);
    temp.pt1.y = min(r.pt1.y, r.pt2.y);
    temp.pt2.x = max(r.pt1.x, r.pt2.x);
    temp.pt2.y = max(r.pt1.y, r.pt2.y);
    return temp;
} //规范化矩形坐标, 即保证左下角顶点pt1的坐标小于右上角顶点pt2的坐标
```

```
bool PtInRect(Point p, Rect r)
{
    if(p.x >= r.pt1.x && p.x <= r.pt2.x
        && p.y >= r.pt1.y && p.y <= r.pt2.y)
        return true;
    else
        return false;
} //判断一个点是否在一个矩形内
```

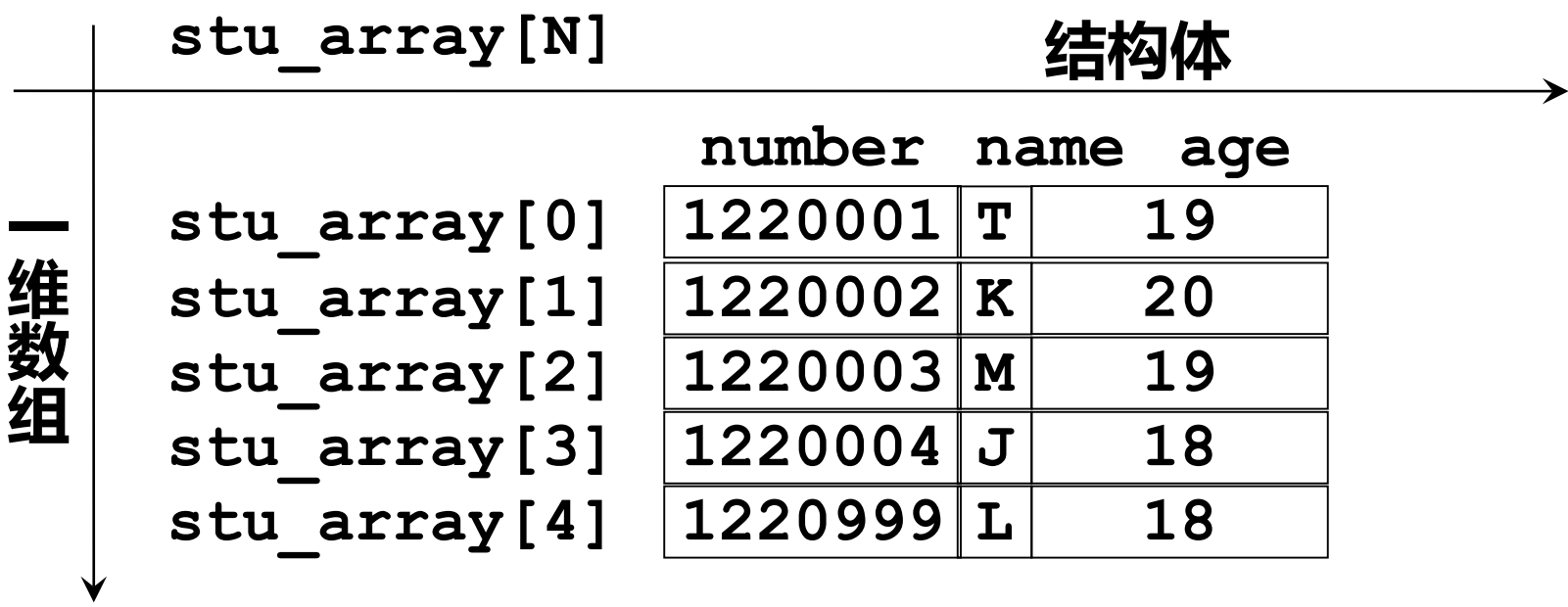
结构数组

● 结构数组可用于表示二维表格。比如，名表：

```
typedef struct
{
    int number;
    char name;
    int age;
} Student;
```

```
const int N = 5;
Student stu_array[N] = { {1220001, 'T', 19}, ...,
                          {1220999, 'L', 18} }; //定义了一个一维结构数组，并初始化

for(int i=0; i<N; ++i)
    scanf("%d%c%d", &stu_array[i].number, &stu_array[i].name, & stu_array[i].age);
```



例8.3 基于结构数组的顺序查找。

`int x = Search(a, N, id);` //在 N 个学生中查找学号为 id 的学生年龄

```
int Search(Student stu_array[], int count, int id)
{
    for(int i = 0; i < count; ++i)
        if(id == stu_array[i].number)
            return stu_array[i].age;
    return -1;
}
```

● 例8.3' 基于结构数组的折半（二分法）查找。先按某一列排序

```
void BubbleSort(Student stu_array[ ], int count)
{
    for(int i = 0; i < count-1; ++i)
        for(int j = 0; j < count-1-i; ++j)
            if(stu_array[j].number > stu_array[j+1].number)
            {
                Student temp = stu_array[j];
                stu_array[j] = stu_array[j+1];
                stu_array[j+1] = temp;
            }
}
```

`int x = BiSearchR(a, 0, N, no);` //在 N 个学生中查找学号为 no 的学生年龄

```
int BiSearchR(Student stu_array[], int first, int last, int id)
{
    if(first > last)
        return -1;
    int mid = (first + last) / 2;
    if(id == stu_array[mid].number)
        return stu_array[mid].age;
    else if(id > stu_array[mid].number)
        return BiSearchR(stu_array, mid + 1, last, id);
    else
        return BiSearchR(stu_array, first, mid - 1, id);
}
```

用指针操纵结构体

- 将某结构体的地址赋给指针变量，用指针变量操纵结构体的成员，这时，成员操作符写成箭头形式（->），而不是点形式（.）。比如，

```
struct
```

```
{
```

```
    int no;
```

```
    float score;
```

```
} s, *ps ;
```

```
ps = &s;
```

```
ps -> no = 1220999;
```

```
//相当于(*ps).no或s.no
```

```
ps -> score = 90.0;
```

```
//相当于(*ps).score或s.score
```


-
- 为了提高程序的效率，函数间传递结构体时，实参可以用结构体的地址，形参用相同结构类型的指针。

传值方式-效率不高

例8.4-1 未用指针变量操纵结构体。

```
int main( )
{
    YDate d1;
    scanf("%d%d%d", &d1.year, &d1.month, &d1.day);
    Days(d1);

    return 0;
}
```

```
typedef struct
{
    int year;
    int month;
    int day;
    int yearday;
} YDate;
```

| | | |
|----|---------|--|
| d1 | year | |
| | month | |
| | day | |
| | yearday | |

```
void Days(YDate d2)
```

| | | |
|----|---------|--|
| d2 | year | |
| | month | |
| | day | |
| | yearday | |

```
void Days(YDate d2)
{
    int monthtable[ ][13]= {
        {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
        {1, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

    int leap = 0;
    d2.yearday = d2.day;

    if( (d2.year %4 == 0) && (d2.year %100 != 0)
        || (d2.year % 400 == 0) )
        leap = 1;

    for(int i = 1; i < d2.month; ++i)
        d2.yearday += monthtable[leap][i];
    printf("所输入的日期是该年的第几天: %d", d2.yearday);
}
```

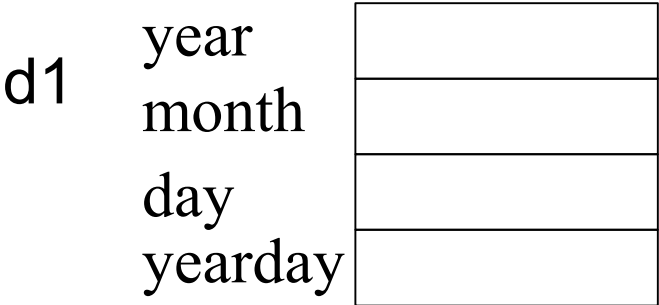
传址方式-提高效率

例8.4-2 用指针变量操纵结构体。

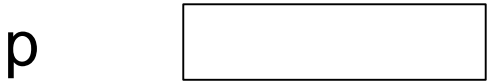
```
int main( )
{
    struct Date d1;
    scanf("%d%d%d", &d1.year, &d1.month, &d1.day);
    Days(&d1);

    return 0;
}
```

```
typedef struct
{
    int year;
    int month;
    int day;
    int yearday;
} YDate;
```



```
void Days(YDate *p)
```



```
void Days(struct Date *p)    Days(&d1) ;
{
    int monthtable[ ][13]= {
        {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
        {1, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};

    int leap = 0;
    p -> yearday = p -> day;

    if( ((p -> year %4 == 0) && (p -> year %100 != 0))
        || (p -> year % 400 == 0) )
        leap = 1;

    for(int i = 1; i < p -> month; ++i)
        p -> yearday += monthtable[leap][i];
    printf("所输入的日期是该年的第几天: %d", p -> yearday);
}
```

传址方式-提高效率，并“返回”结果

例8.4-2’ 用指针变量操纵结构体。

```
typedef struct
{
    int year;
    int month;
    int day;
    int yearday;
} YDate;
```

```
int main( )
{
    YDate d1;
    scanf("%d%d%d", &d1.year, &d1.month, &d1.day);
    Days(&d1);
    printf("所输入的日期是该年的第几天: %d", d1.yearday);
    return 0;
}
```

| | | |
|----|---------|----------------------|
| d1 | year | <input type="text"/> |
| | month | <input type="text"/> |
| | day | <input type="text"/> |
| | yearday | <input type="text"/> |

```
void Days(YDate *p)
```



- ❁ 如果不需要通过参数返回数据，则可以用`const`避免函数的副作用。比如

,

```
void G(const YDate *p)
```

```
{
```

```
...
```

```
p -> day = 20;    //会出错，因为不能通过p改变p所指向的数据
```

```
...
```

```
}
```

- ❁ 函数也可以返回一个结构体的地址。

- 结构类型可以含有**指针类型成员**，操作方法与其他类型成员的类似。

比如，

```
struct  
{  
    int no;  
    int *p;  
} s ;
```

```
s.no = 1001;
```

```
s.p = &i;    // int i = 3;
```

```
s.p = &s.no;
```


- 如果有成员是另一结构类型的指针变量，则可以用若干个箭头形式的成员操作符访问最低一级的成员。比如，

```
struct
{
    Student *p1;
    float score;
} s, *pps ;
pps = &s;
pps -> p1 = &s1;
pps -> score = 85;
pps -> p1 -> number = 1220001;
pps -> p1 -> name = 'Q';
pps -> p1 -> age = 19;
```

```
struct Student
{
    int number;      //成员
    char name;       //成员
    int age;         //成员
} s1;
```

```
struct
{ char name[20];
  struct Date
  { int year;
    int month;
    int day;
  } birthday;
} s = {"Joe", {1996,12,14}};
```

```
s.name = "Joe"; x
```

```
scanf("%s", s.name);
```

```
struct
{ const char *name;
  struct Date
  { int year;
    int month;
    int day;
  } birthday;
} s = {"Joe", {1996,12,14}};
```

```
s.name = "Joe";
```

```
scanf("%s", s.name); x
```

- ❁ 结构类型不可以含有本结构类型成员。
- ❁ 结构类型可以含有基类型是本结构类型的指针类型成员。比如，

```
typedef struct Stup Stup;
```

```
struct Stup
```

```
{
```

```
    int no;
```

```
    struct Stup *p; // Stup *p;
```

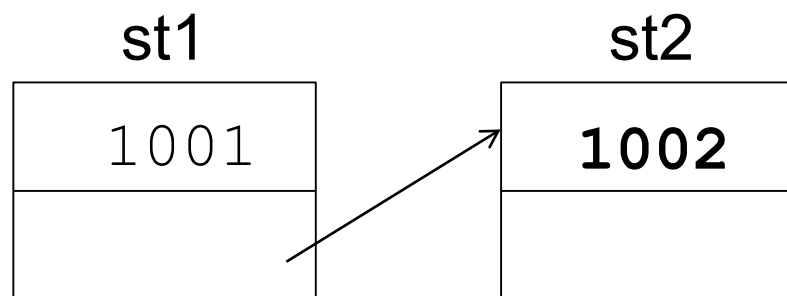
```
} st1, st2;
```

```
st1.no = 1001;
```

```
st1.p = &st2;
```

```
st2.no = 1002; // st1.p -> no = 1002;
```

```
st2.p = .....; // st1.p -> p = .....;
```

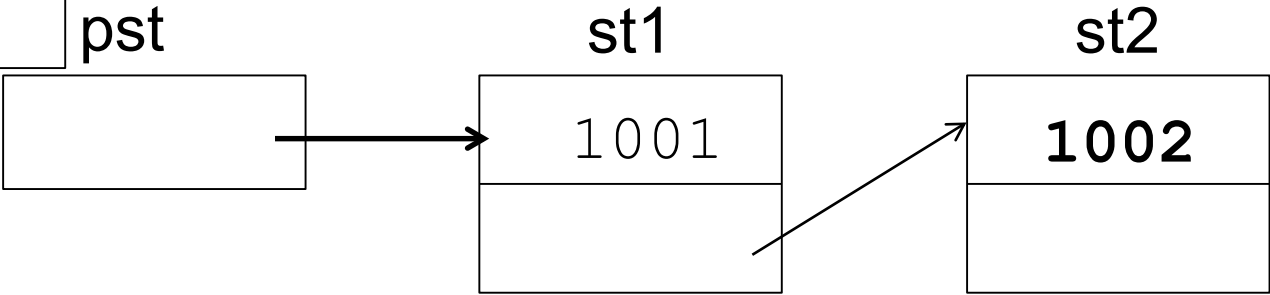


● 用指针变量操纵**指针类型成员**。比如，

```
typedef struct Stup Stup;
```

```
struct Stup  
{  
    int no;  
    struct Stup *p; // Stup *p;  
} st1, st2, *pst;
```

```
pst = &st1;  
st1.no = 1001; // pst -> no = 1001;  
st1.p = &st2; // pst -> p = &st2;  
st2.no = 1002; // st1.p -> no = 1002; // pst -> p -> no = 1002;  
st2.p = .....; // st1.p -> p = .....; // pst -> p -> p = .....;
```



● 用指针变量操纵动态结构体的基类型是本结构类型的指针类型成员。

```
typedef struct Stup Stup;
```

```
struct Stup  
{  
    int no;  
    struct Stup *p; // Stup *p;  
} *pst;
```

```
pst = (Stup *)malloc(sizeof(Stup)); // 创建第一个动态结构体  
pst -> no = 1001;  
pst -> p = (Stup *)malloc(sizeof(Stup)); // 创建第二个动态结构体  
pst -> p -> no = 1002;  
pst -> p -> p = .....;
```

小结

● 结构是一种派生数据类型，用来描述多个不同类型的数据群体

● 要求：

- ◆ 掌握结构类型的构造方法
- ◆ 掌握结构体的定义、初始化和操作方法
- ◆ 掌握结构数组的特点
 - 结构数组可存储名表，在实际生产生活中发挥作用
- ◆ 掌握用指针操纵结构体的方法
 - 用指针操纵含有指针成员的结构体
- ◆ 继续保持良好的编程习惯

Thanks!

