

结构体与共同体

Wang Houfeng

EECS, PKU

wanghf@pku.edu.cn

内容

➤ 结构体

- 共同体
- 枚举

问题引入

- 有的应用需要将不同类型的数据组合成一个有机的整体。如：

一个学生的属性有：学号/姓名/性别/年龄/地址

```
int num; char name[20];  char sex;  
int age; char addr[30];
```

| Num | name | sex | age | score | addr |
|--------|--------|-----|-----|-------|---------|
| 100101 | Li Fun | M | 18 | 87.5 | Beijing |

结构体——用户定义类型

- C/C++ 语言提供了基本类型（如整型、字符型）
- 基本类型无法表示用户希望的数据——怎么办？
- **结构体**——提供了一种用户定义类型的方式

- **特点**：可以把不同类型的数据组合成一个整体

- **结构体类型定义**

struct是关键字,
不能省略

```
struct [结构体名]
{
    类型  成员名;
    类型  成员名;
    .....
};
```

合法标识符

可省: 无名结构体

成员类型可以是
基本型或构造型

例子

- 关于学生student的信息

- 学号 (整型)
- 名字 (字符串型:20)
- 性别 (字符型)
- 年龄 (整型)
- 成绩 (浮点型)
- 地址 (字符串型:30)

成员的类型

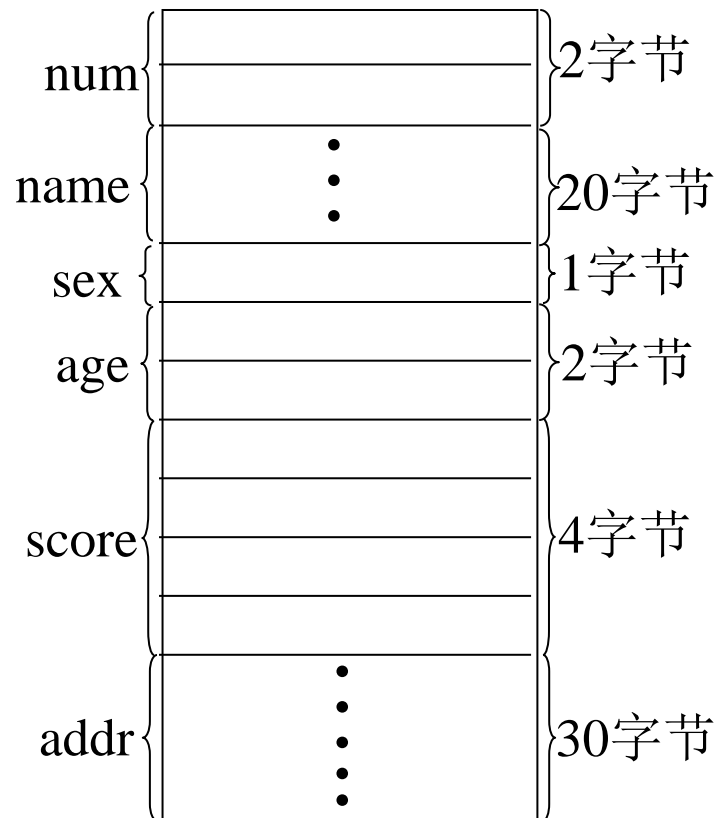
成员名

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

结构体类型定义描述结构的组织形式,不分配内存

结构对应关系

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



如何声明结构体类型的变量

- 先定义结构体类型，再定义结构体变量

```
struct    结构体名
{
    类型标识符  成员名;
    类型标识符  成员名;
    .....
};
struct 结构体名 变量名表列;
```

例

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
};
struct student stu1,stu2;
```

类型的定义和变量声明一起

```
struct      结构体名
{
    类型标识符    成员名;
    类型标识符    成员名;
    .....
} 变量名表列;
```

```
例 struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} stu1,stu2;
```


无名类型定义和变量声明一起

```
struct
{
    类型标识符  成员名;
    类型标识符  成员名;
    .....
}变量名表列;
```

用无名结构体直接定义变量只能一次

例

```
struct
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
}stu1,stu2;
```

说明

- 结构体类型与结构体变量概念不同

- 类型: 不分配内存; 变量: 分配内存
- 类型: 不能赋值、存取、运算; 变量: 可以

- 结构体可嵌套

- 结构体成员名与程序中变量名可相同, 不会混淆

- 结构体类型及变量的作用域与生存期

```

例 struct student
{
    int num;
    char name[20];
    struct date
    {
        int month;
        int day;
        int year;
    } birthday;
} stu;

```

| num | name | birthday | | |
|-----|------|----------|-----|------|
| | | month | day | year |

```

例 struct date
{
    int month;
    int day;
    int year;
};
struct student
{
    int num;
    char name[20];
    struct date birthday;
} stu;

```

| num | name | birthday | | |
|-----|------|----------|-----|------|
| | | month | day | year |

结构体变量的初始化

```
struct    结构体名
{
    类型标识符  成员名;
    类型标识符  成员名;
    .....
};
struct 结构体名 结构体变量={初始数据};
```

```
例 struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    char addr[30];
};

struct student stu1={112,“Wang Lin”,‘M’,19, “200 Beijing Road”};
```

结构体变量的引用

– 引用规则

- 结构体变量不能整体引用, 只能引用变量成员

引用方式: 结构体变量名. 成员名

成员(分量)运算符
优先级: 1
结合性: 从左向右

例

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} stu1, stu2;
```

stu1.num=10;

stu1.score=85.5;

stu1.score+=stu2.score;
stu1.age++;

例

```

struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
}stu1,stu2;

```

不要整体引用

printf(“%d,%s,%c,%d,%f,%s\n”,**stu1**); (×)

stu1={101,“Wan Lin”,‘M’,19,87.5,“DaLian”}; (×)

例

```

struct student
{
    int num;
    char name[20];
    struct date
    {
        int month;
        int day;
        int year;
    }birthday;
}stu1,stu2;

```

| num | name | birthday | | |
|-----|------|----------|-----|------|
| | | month | day | year |

stu1.birthday.month=12;

结构体嵌套时**逐级引用**
不能直接引用 stu1.birthday

可以将结构体变量赋值给另一个结构体变量

stu2=stu1; (✓)

例

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
}stu1,stu2;
```

例

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
}stu1,stu2;
```

if(stu1==stu2)
..... (✗)

不能比较大小

结构体和数组

- 数组的元素类型可以是结构类型
- 例子：

struct student stu[3];

- stu 是数组变量，含3个元素；
- stu 的每个元素类型都是 struct student 类型

| | num | name | sex | age | score | addr |
|--------|-------|-----------|-----|-----|-------|--------------------|
| stu[0] | 10101 | Li Lin | M | 18 | 87.5 | 103 Beijing Road |
| stu[1] | 10102 | Zhang Fun | M | 19 | 99 | 130 Shanghai Road |
| stu[2] | 10104 | Wang Min | F | 20 | 78.5 | 1010Zhongshan Road |

例子

- 100名职工，包含姓名，工资号，工资，输入全部职工的信息，并按工资从高向低排序

数据类型定义：

```
struct employee {  
    char name[10];  
    long id;  
    float salary  
};  
struct employee person[100]; // 100 名员工
```

输入部分：

```
p_input(struct employee p[],int n)
{
    int k;
    for(k=0; k<n;++k)
    {
        scanf(" %s%d%f ", p[k].name,
            &p[k].id, &p[k].salary);
    }
}
```

冒泡排序部分：

```
p_sort(struct employee p[],int n)
{
    int k,j;
    struct employee temp
    for(k=0; k<n-1;++k)
        for(j=0; j<n-k-1;++j)
        {
            if(p[j].salary>p[j+1].salary)
            {
                temp=p[j];
                p[j]=p[j+1];
                p[j+1]=temp;
            }
        }
}
```

指向结构体数据的指针

- 一个结构体变量可以用一个指针变量来指向。此时该指针变量的值是结构体变量在内存中的起始地址。

结构体变量在内存的起始地址

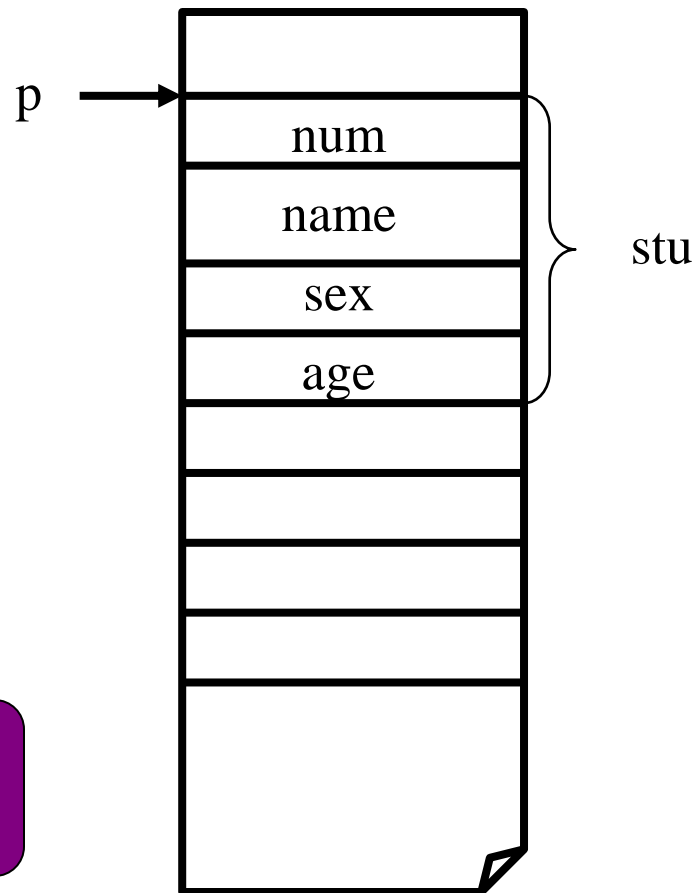
— 指向结构体变量的指针

- **定义形式：** struct 结构体名 *结构体指针名；

一个例子

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
}stu;
struct student *p=&stu;
```

p指向结构变量stu的起始地址



使用结构体指针变量引用成员形式

```
例  int  n;  
     int  *p=&n;  
     *p=10;    ⇔ n=10
```

```
struct student  stu1;  
struct student  *p=&stu1;  
stu1.num=101;  ⇔ (*p).num=101
```

(*结构体指针名).成员名 ⇔ 结构体指针名->成员名 ⇔ 结构体变量名.成员名

表示**指向**的运算符

优先级： 1

结合方向： 从左向右

如下三种等效果的引用形式：

- 结构体变量.成员名，如： **stu1.num=101**
- (*p).成员名，如： **(*p).num=101**
- p->成员名，如： **p->num=101**

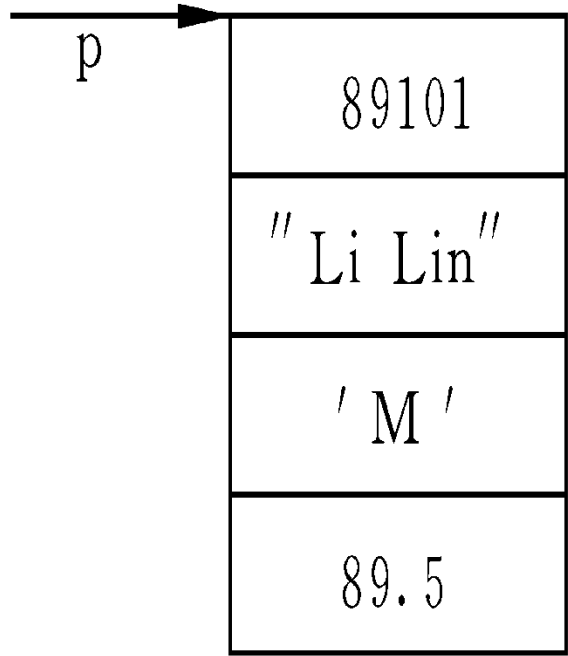
成员引用的优先级

- $p \rightarrow n$ 得到 p 指向的结构体变量中的成员 n 的值。
- $p \rightarrow n++$ 得到 p 指向的结构体变量中的成员 n 的值，用完该值后使它加 1，等价于 $(p \rightarrow n)++$
- $++p \rightarrow n$ 得到 p 指向的结构体变量中的成员 n 的值加 1，然后再使用它，等价于 $++(p \rightarrow n)$

运算符 \rightarrow 的优先级高：为 1

例子

```
main()
{
    struct student
    {
        long int num;
        char name[20];
        char sex;
        float score;
    } stu_1, *p;
    p=&stu_1;
    stu_1.num=89101;
    strcpy(stu_1.name,"Li Lin");
    p->sex='M';
    p->score=89.5;
    printf("\nNo:%ld\nname:%s\nsex:%c\nscore:%f\n",
        p->num, (*p).name, stu_1.sex, p->score);
}
```



| |
|----------|
| 89101 |
| "Li Lin" |
| 'M' |
| 89.5 |

例子：数组、指针、结构

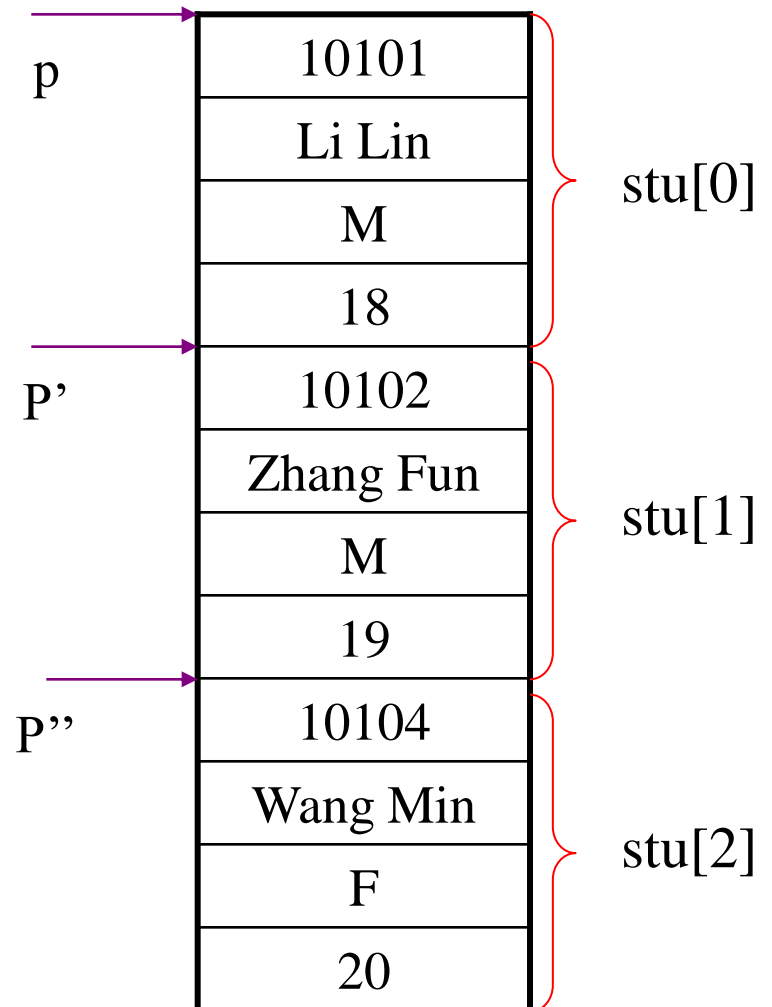
```
#include <stdio.h>
struct student
{
    int num; char name[20]; char sex; int age;};
    struct student stu[3]={ {10101,"Li Lin",'M',18},{10102, " Zhang
    Fun",'M',19},{10104,"Wang Min",'F", 20}};
void main()
{
    struct student *p;
    printf("No. Name sex age \");
    for (p=str; p<str+3; p++)
        printf("%5d %-20s %2c %4d\n", p->num,p->name, p->sex, p->age);
}
```

运行结果：

| No. | Name | Sex | age |
|-------|-----------|-----|-----|
| 10101 | Li Lin | M | 18 |
| 10102 | Zhang Fun | M | 19 |
| 10104 | WangMing | F | 20 |

程序分析

- 循环过程使 $p++$ ，即：
 p 自加 1；
- 加 1 意味着 p 的值等于 $stu+1$ ， p 指向 $stu[1]$ ；
- 再执行 $p++$ 后， p 的值等于 $stu+2$ ，即 $stu[2]$ 。



两个运算的区别

如果 p 的初值为 stu ，即指向第一个元素，则 p 加 1 后 p 就指向下一个元素。

- $(++p) \rightarrow num$ 先使 p 自加 1，然后得到它指向的元素中的 num 成员值（即 10102）。
- $(p++) \rightarrow num$ 先得到 $p \rightarrow num$ 的值（即 10101），然后使 p 自加 1，指向 $stu[1]$ 。

请注意以上二者的不同。

结构体变量作为函数参数

- 用结构体变量的成员作实在参数——值传递
- 用指向结构体变量或数组的指针作参数——地址传递
- 用结构体变量作参数——多值传递，效率低

例子

- 有一个结构体变量`stu`，内含学生学号、姓名和3门课程的成绩。要求在`main`函数中赋予值，在另一函数`print`中将它们输出。用结构体变量作函数参数。

```
#include <stdio.h>

struct student
{
    int num;
    char name[20];
    float score[3];
};
```

程序

```
void print ( struct student);  
void main()  
{  
    struct student stu;  
    stu.num=12345; strcpy(stu.name,“LiLi”);  
    stu.score[0]=67.5; stu.score[1]=89;  
    stu.score[2] =78.6); print(stu);  
}  
void print ( struct  student  stu)  
{  
    printf(FORMAT, stu.num,stu.name, stu.score[0],  
    stu.score[1],stu.score[2]);  
    printf(“\n”); //FORMAT 是表示格式的宏， 自己补上。  
}
```

运行结果:

12345

LiLi

67.500000

89.000000

78.599998

用指针实现

```
#include <stdio.h>
Struct student{
    int num;
    char name[20];
    float score[3]; } stu={12345, "LiLi",67.5,89,78.6};
void print (struct student *p) /*形参类型修改了*/
{ printf(FORMAT, p->num, p->name, p->score[0], p->
    >score[1], p->score[2]); printf("\n");}
void main()
{print(&stu); /*实参改为stu的起始地址*/
}
```

运行结果:
12345
LiLi
67.500000
89.000000
78.599998

另一个例子

```
void func(struct data);
```

```
struct data
```

```
{ int a, b, c; };
```

```
main()
```

```
{
```

```
    struct data arg;
```

```
    arg.a=27; arg.b=3; arg.c=arg.a+arg.b;
```

```
    printf("arg.a=%d arg.b=%d arg.c=%d\n",arg.a,arg.b,arg.c);
```

```
    printf("Call Func()....\n");
```

```
    func(arg);
```

```
    printf("arg.a=%d arg.b=%d arg.c=%d\n",arg.a,arg.b,arg.c);
```

```
}
```

```
void func(struct data parm)
```

```
{ printf("parm.a=%d parm.b=%d parm.c=%d\n",parm.a,parm.b,parm.c);
```

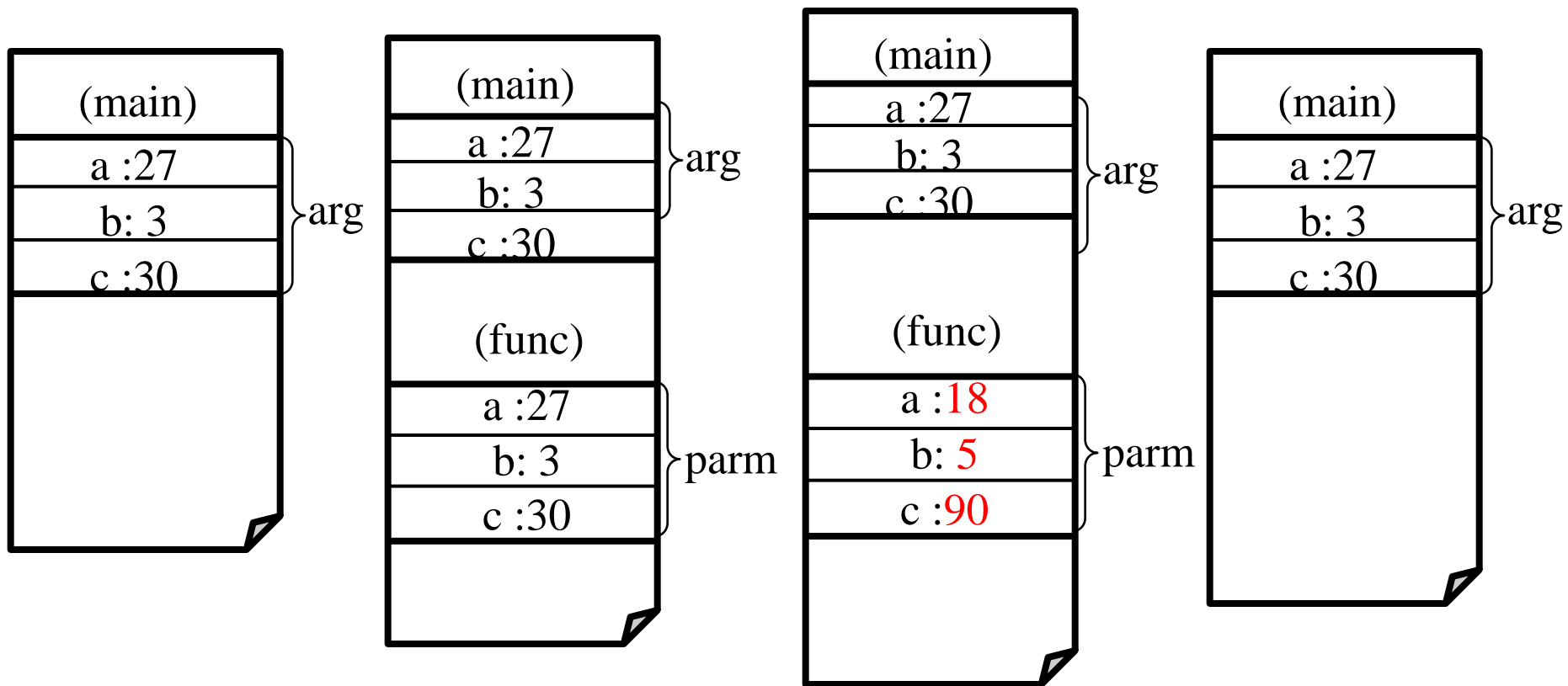
```
    printf("Process...\n");
```

```
    parm.a=18; parm.b=5; parm.c=parm.a*parm.b;
```

```
    printf("parm.a=%d parm.b=%d parm.c=%d\n",parm.a,parm.b,parm.c);
```

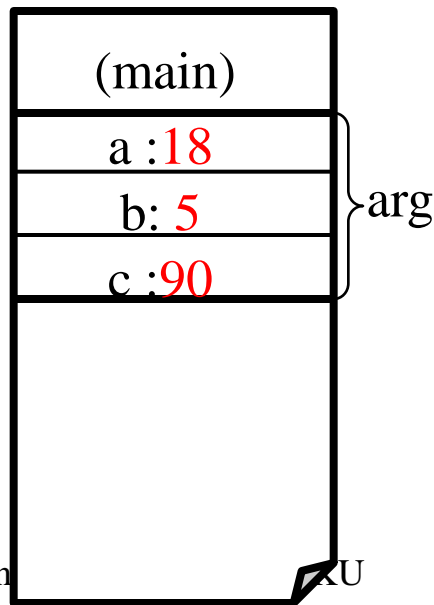
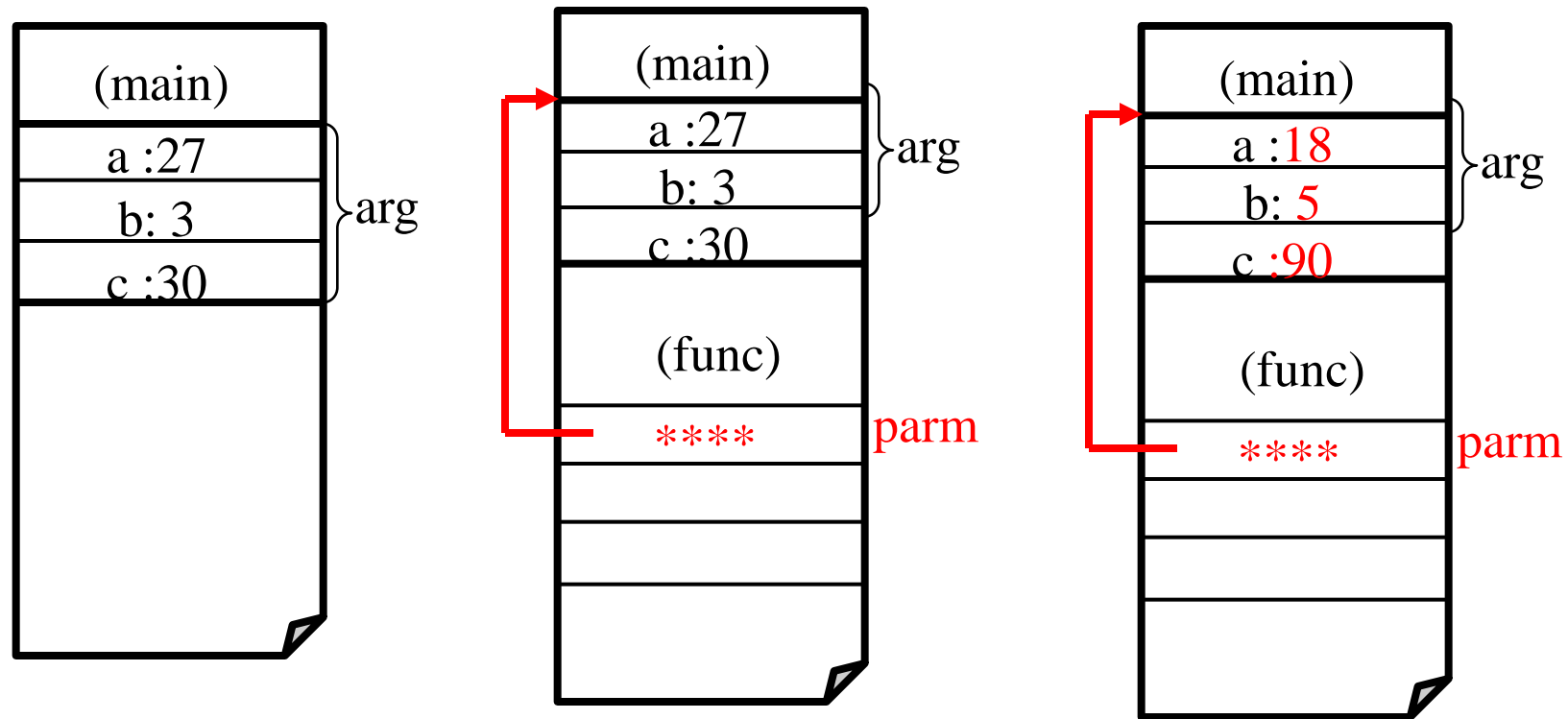
```
    printf("Return...\n");
```

```
}
```



用结构体指针变量作函数参数

```
struct data
{  int a, b, c; };
void func(struct data *parm);
main()
{  struct data arg;
   arg.a=27; arg.b=3;  arg.c=arg.a+arg.b;
   printf("arg.a=%d arg.b=%d arg.c=%d\n",arg.a,arg.b,arg.c);
   printf("Call Func()....\n");
   func(&arg);
   printf("arg.a=%d arg.b=%d arg.c=%d\n",arg.a,arg.b,arg.c);
}
void func(struct data *parm)
{  printf("parm->a=%d parm->b=%d parm->c=%d\n",parm->a,parm->b,parm->c);
   printf("Process...\n");
   parm->a=18;  parm->b=5;  parm->c=parm->a*parm->b;
   printf("parm->a=%d parm->b=%d parm->c=%d\n",parm->a,parm->b,parm->c);
   printf("Return...\n");
}
```



内容

- 结构体
- 共同体
- 枚举

共同体简介

- **定义**

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。

- **一般形式为：**

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

举例

union data

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

union data

```
{ int i;  
  char ch;  
  float f;  
};
```

union data a,b,c;

1000地址

| | | | |
|------------|----|---|---|
| 整型 i | 变量 | | |
| 字符变 量ch | | | |
| 实 | 型变 | 量 | f |

共用体和结构体的比较

- 结构体变量所占内存长度是各成员占的内存长度之和。每个成员分别占有其自己的内存单元。
- 共用体变量所占的内存长度等于最长的成员的长度。
- 例如：上面定义的“共用体”变量 a、b、c 共占 4 个字节（因为一个实型变量占 4 个字节），而**不是**各占 $2 + 1 + 4 = 7$ 个字节。

共同体引用

只有先定义了共用体变量才能引用它，而且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:前面定义了a、b、c为共用体变量

- a.i (引用共用体变量中的整型变量 i)
- a.ch (引用共用体变量中的字符变量 c h)
- a.f (引用共用体变量中的实型变量 f)

引用de 几种形式

共用体变量名.成员名 \longleftrightarrow 共用体指针名->成员名 \longleftrightarrow (*共用体指针名).成员名

```
union data
```

```
{  int i;  
    char ch;  
    float f;  
};
```

```
union data a,b,c,*p,d[3];
```

```
a.i  a.ch  a.f
```

```
p->i  p->ch  p->f
```

```
(*p).i  (*p).ch  (*p).f
```

```
d[0].i  d[0].ch  d[0].f
```

共用体变量中起作用的成员是最后一次存放的成员

例 a.i=1;

a.ch='a';

a.f=1.5;

printf(“%d”,a.i);

(✗ 编译通过，运行结果不对)

引用问题

引用规则: 不能引用共用体变量, 只能引用其成员

不能在定义共用体变量时初始化

```
例 union
{   int i;
    char ch;
    float f;
}a;
a=1;      (✗)
```

可以用一个共用体变量为另一个变量赋值

```
例 union
{   int i;
    char ch;
    float f;
}a={1,'a',1.5};  (✗)
```

```
例 float x;
union
{   int i;  char ch;  float f;
}a,b;
a.i=1;  a.ch='a';  a.f=1.5;
b=a;    (✓)
x=a.f;  (✓)
```

union 总结

- (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，还不能在定义共用体变量时对它初始化。
- (5) 不能把共用体变量作为函数参数，也不能使函数带回共用体变量，可以使用指向共用体变量的指针
- (6) 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。

内容

- 结构体
- 共同体
- 枚举

枚举类型

- 枚举

- 如果一个变量只有几种可能的取值，则可以将该变量定义为“枚举类型”。

- 枚举类型的定义

- 定义一个枚举数据类型

一个例子： `enum weekday {sun, mon, tue, wed, thu, fri, sat};`

花括号中 sun,mon,...,sat等称为枚举元素

- 声明枚举变量：

`enum weekday` weekday, weekend;

或 `weekday` weekday, weekend;

- 枚举变量赋值：

`weekday = sun; weekend = mon;`

枚举类型

- 枚举元素有值

- 定义时枚举元素如未指定值，编译系统按定义顺序取默认值依次为0，1，2，3...
- 也可以给枚举值指定对应值

```
enum weekday { Sun=7, Mon=1, Tue, Wed, Thu, Fri, Sat };
```

这时，Sun=7，Mon=1，Tue=2，Wed=3

- 整数不能直接赋给枚举变量

如：workday = 2; 错误！

- 应先进行强制类型转换如：

```
workday = ( enum weekday ) 2;
```

枚举类型使用注意事项

1. 枚举元素按常量处理，不能对它们赋值

`sun = mon;` （错误）

2. 枚举型变量不能直接输出元素的名字.

[例如] `enum color {red, green, white, black};`

`color cloth = red;`

`cout<<cloth;` //结果为0

3. 枚举型可以比较（按内在的值比较）

`if (cloth > white) count++;`

4. 一个整型不能直接赋给一个枚举变量

`workday = 2;` //错误！

枚举类型

```
#include<iostream>
using namespace std;
enum color {red,yellow,green=3,blue };
enum color c1;
int main()
{
    c1=blue;
    cout<<"red="<<red<<" yellow="<<yellow<<" green="<<green<<endl;
    cout<<"blue="<<blue<<" c1="<<c1<<endl;
    system("pause");
}
```

```
red=0 yellow=1 green=3
blue=4 c1=4
请按任意键继续. . .
```

枚举类型的使用

- 如何输出枚举型变量的内容

```
enum color {red,green,blue,brown,white,black};  
enum color choice;  
switch(choice)  
{ case red:      cout<<"red\n";      break;  
  case green:    cout<<"green\n";    break;  
  case blue:     cout<<"blue\n";     break;  
  case brown:    cout<<"brown\n";    break;  
  case white:    cout<<"white\n";    break;  
  case black:    cout<<"black\n";    break;  
}
```