

# UNIT 18 课程内容总结

---

主讲人：谭成予

副教授

武汉大学计算机学院

E-MAIL : nadinetan@163.com

<http://jpkc.whu.edu.cn/jpkc2005/alprogram>



# 本课程主要知识点

---

1. 标识符、数据和数据类型
2. 运算符和表达式
3. 基本结构和控制流
4. 函数
5. 编译预处理
6. 数组和串
7. 指针
8. 结构、联合和枚举类型
9. 流和文件



# 1、标识符、数据和数据类型



# 1.1、标识符和关键字

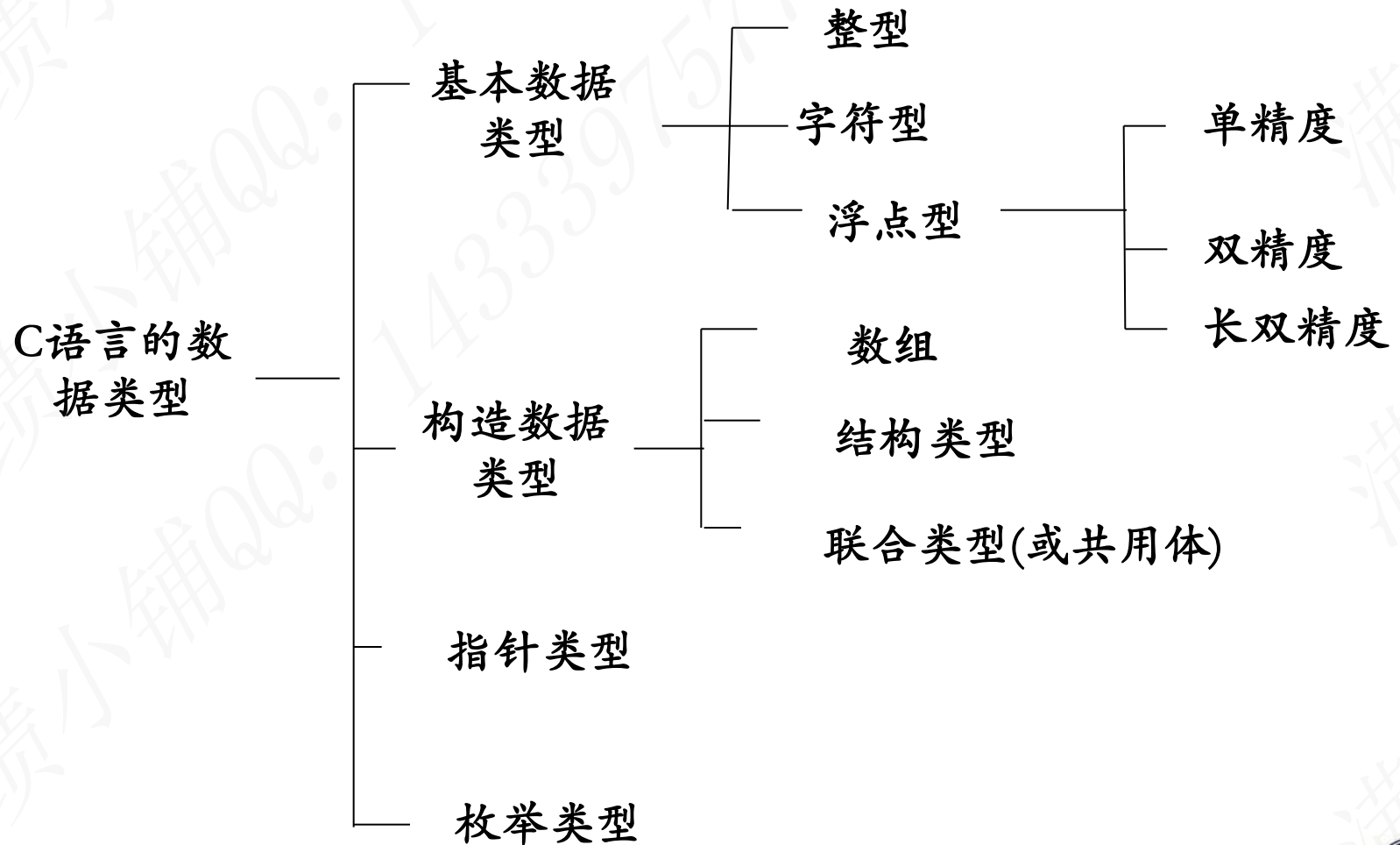
---

## 标识符命名规则

- 标识符由三类字符构成：英文大小写字母；数字0.....9；下划线。
- 必须由字母或下划线开头；后面可以跟随字母、数字或下划线。
- C语言区分大小写。
- 不能与关键字同名。



# 1.2、C的数据类型



## 1.3、C的基本数据类型

### 基本数据类型名

整型：int，1个字长

浮点型：float

双精度浮点型：double

字符型：char，1个字节

无值型：void



### 基本数据类型修饰符

有符号：signed

无符号：unsigned

长型：long

短型：short



负数采用补码表示

## 1.4、C的基本数据类型

有  
符  
号  
整  
型

**短整型：** short int

2个字节长度，数据范围-32768~32767

**整型：** int

1个字长，2个字节或者4个字节

**长整型：** long int

4个字节，数据范围-2 147 483 648~2 147 483 647

整  
型

无  
符  
号  
整  
型

**无符号短整型：** unsigned short int

2个字节长度，数据范围0~65535

**无符号整型：** unsigned int

1个字长，2个字节或者4个字节

**无符号长整型：** unsigned long int

4个字节，数据范围0~4 284 967 295

满绩小铺：1433397577，搜集整理不易，自用就好，谢谢！



## 1.4、C的基本数据类型

有  
符  
号

`char`

1个字节长度，数据范围-128~127

C语言将字符型看作是1个字节的整数

字  
符  
型

无  
符  
号

`unsigned char`

1个字长，数据范围0~255





## 1.6、C的基本数据类型

浮点型： float

6位精度

双精度浮点型： double

10位精度

长双精度浮点型： long double

10位精度

浮  
点  
型



## 1.7、常 量

整型常量:

- 十进制、八进制、十六进制表示: 123 056 0xa23f
- 后缀: L 长整型; U 无符号整型。

字符型常量:

- 单引号括起来的单个字符: '2'
- 转义字符: '\n' '\47' '\x7f'

字符串常量:

- 双引号括起来的字符序列: "wuhan"

实型常量:

- 小数形式和指数形式: 1.23 .56 2.3E-9
- 后缀: L 长型; F 浮点型



## 1.8、数据类型的选择

---


C语言中那么多种数据类型，如何选择？

- 数据范围：足够表示所有可能出现的数据取值；
- 精度：满足精度要求；
- 操作便捷；
- 所需内存空间：满足上述条件的前提下，尽可能少占据内存空间。



## 1.9、数据溢出和计算误差

整型数据：数据范围  数据溢出问题

实型数据：可表示误差，  
即无法精确表示无限位  
数的实数。  计算误差问题



实数如何判断是否相等？  
循环变量尽量不要用实数类型！



# 2、运算符和 表达式



## 2.1、C语言中运算符、结合性及优先级

运算符类型	优先级	运算符	结合性
基本	1	( ) [ ] - > .	从左至右
单目	2	! ~ ++ -- + - (type) * & sizeof()	从右至左
算术	3	* / %	从左至右
	4	+ -	
移位	5	>> <<	从左至右
关系	6	<<= >>=	从左至右
	7	= = ! =	
位逻辑	8	&	从左至右
	9	^	
	10		
逻辑	11	&&	从左至右
	12		
条件	13	?:	从右至左
赋值	14	= += -= *= /= %=  = ^= &= >>= <<=	从右至左
逗号	15	,	从左至右



## 2.2、算术运算符

运算符	作用
-	减法、负号
+	加法
*	乘法
/	除法
%	模除
--	减量
++	增量

/ 除法

○ 两个整数除法的结果是整数

5/2                      结果    2

-5/2                     结果   -2

% 模除，整数除法的余数。

○ %运算的符号只取决于第一个运算数的符号。

7%4 结果 3                      -7%4 结果 -3

-7%-4 结果 -3                      7%-4 结果 3

++ --

○ 前缀形式      后缀形式

○ 注意增量、减量运算符的副作用



## 2.3、赋值运算符

对象名称=表达式

```
int a, b, c;
```

/\*说明a, b, c为整型变量\*/

```
a=12;
```

```
b=c=a;
```

/\*多重赋值\*/

```
a+=a-=a*a;
```

/\*表达式的结果是多少? \*/

对象名称 运算符=表达式

复合赋值运算符，等价于

对象名称=对象名称 运算符(表达式)





## 2.4、关系和逻辑运算符

关系运算符		逻辑运算符	
运算符	作用	运算符	作用
<	小于	&&	与
>	大于		或
<=	小于等于	!	非
>=	大于等于	<p>优先级</p> <p>! &gt; &gt;= &lt; &lt;=</p> <p>== !=</p> <p>&amp;&amp;   </p>	
==	等于		
!=	不等		



## 2.4、逻辑运算符（短路原则）

**$a \& \& b$**

当a为0时，可提前计算表达式结果为0，因此不再处理b。

例如，设变量int m,n,a,b的值均为0，则执行表达式  $(m=a>b) \& \& (n=a>=b)$  后，m、n 的值分别为（ 0 ）和（ 0 ）。

**$a \parallel b$**

当a为1时，可提前计算表达式结果为1，因此不再处理b。

例如，设变量int m,n,a,b的值均为0，则执行表达式  $(m=a>=b) \parallel (n=a>=b)$  后，m、n 的值分别为（ 1 ）和（ 0 ）。



## 2.5、逗号运算符

**Exp1 , Exp2**

**Exp1 , Exp2, .....,Expn**

1. 计算Exp1的值；
2. 计算Exp2的值；
3. 以此类推，以最后一项Expn的值作为表达式的结果。

例： 1)  $a=3*5, a*4$

结果为60

2)  $(a=3*5, a*4), a+5$

结果为20



## 2.6、位运算符

运算符	作用
&	按位与
	按位或
^	异或（没有进位的二进制加法运算）
~	求1的补
>>	右移
<<	左移

位运算运算规则

		&		^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

位逻辑运算符与  
逻辑运算符区别：

4&6                      结果为 4

4&&6                     结果为 1



## 2.7、其他运算符

---

- 条件运算符
- sizeof
- **&和\***
- .
- ->
- **[ ]**
- 强制类型转换



# 3、基本结构和 控制流



# 3.1、三种基本结构

---

顺序、选择、循环；

- 单入口单出口；
- 优点：便于编写结构良好、易于调试的程序；增加程序可读性。



## 3.2 C语言中的语句类别

### 1. 表达式语句

- 表达式语句
- 函数调用语句
- 空语句

### 2. 选择语句

- if语句
- switch语句

### 3. 循环语句

- for
- while
- do-while

### 4. 跳转语句

- return
- goto
- break
- exit()函数
- continue

### 5. 复合语句（块语句）





## 3.3、 if语句

```
if (expression)
    statement1;
else
    statement2;
```

```
if (expression)
    statement1;
```

1. statement1和statement2只能是一条语句，或者块语句，或者是空语句；
2. expression取值非零（真），执行statement1；
3. expression取值零（假），执行statement2



## 3.4、switch语句

```
switch(expression){  
    case constant1:  
        statement sequence  
        break;  
    case constant2:  
        statement sequence  
        break;  
    .....  
    default:  
        statement sequence  
}
```

1. **break**: 跳出case分支的跳转语句，必不可少。
2. **expression**: 字符型或整型表达式；
3. **case constant**: case后面只能为常量表达式；



## 3.5、循环语句

```
for(initialization; condition; increment)
    statement;
```

```
initialization;
while(condition)
    statement;
```

```
initialization;
do{
    statement sequence
}while(condition);
```

1. **initialization**: 初始化, 一般为赋值语句;
2. **condition**: 循环条件, 循环一直执行直到条件为假为止;
3. **statement**: 循环体, 单个语句、块语句、空语句;
4. **increment**: 修改控制变量。



## 3.5、循环语句使用要点

---

### 1. 循环条件的控制：

- 循环变量通常为整型类别；
- 如果必须用实数控制循环条件，请注意实数的可表示误差、比较方法等问题。

### 2. 注意初始化、循环条件、循环体、修改循环控制条件等部分的先后次序。

### 3. 循环条件的真假判断。



## 3.6、其他控制语句

---

### 1. **break:**

- 用于循环语句中结束整个循环；
- 用于switch语句。

### 2. **continue:** 用于循环语句中，提前结束本次循环。

### 3. **goto:** 不能在函数间跳转。



# 4、函数



# 结构化程序设计的思想

---

- **代码的可读性**：三种基本结构，代码结构清晰，可读性强；
- **自顶向下，逐步求精**；
- **模块抽取的原则**：相对独立、功能单一、结构清晰、接口简单；



## 4.1、函数的定义

```
return_value_type function_name(parameter list)
{
    body of function
}
```

每个函数都是一个独立的代码块

- **函数的块作用域**：每个函数的代码块专用于该函数，其他函数无法访问；
- **局部变量**：函数内部定义的变量成为局部变量，专用于该函数；
- **全局变量**：函数外部定义的变量成为全局变量，定义点后的各个函数共用；
- **形式参数**：用来接收调用者传入信息的特殊局部变量。





## 4.2、函数的调用

### 函数名(实参表)

- 实参与形参个数相等，类型一致，按顺序一一**对应**；
- 实参表**求值顺序**，因系统而定（Turbo C 自右向左）。

**函数语句**：不要求函数返回确定的值。

例      `printstar();`  
          `printf("Hello,World!\n");`

**函数表达式**：要求函数必须返回一个确定的值；

例      `m=max(a,b)*2;`



## 4.3、函数原型（使用时机）

### 函数原型

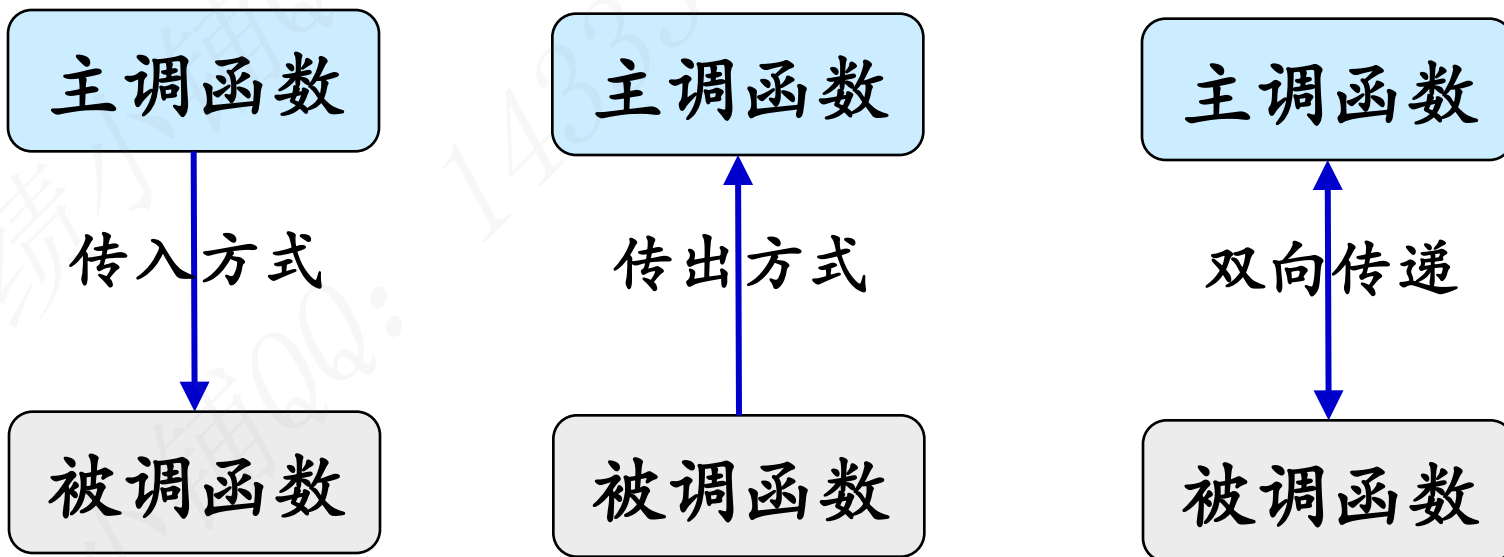
```
return_value_type function_name(parameter list);
```

1. 把被调函数定义的位置放在主调函数之前，用这种方法也可以省去被调函数的原型说明；
2. 被调函数定义的位置放在主调函数之后，则必须在函数调用之前使用被调函数的原型说明；

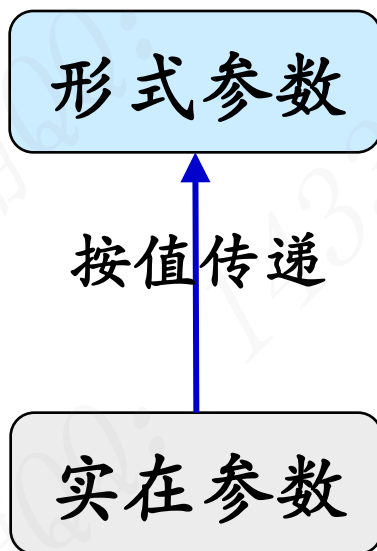


## 4.4、模块间的数据通信方式

主调函数与被调函数之间的三种数据通信方式：



## 4.5、C函数的传入方式： 形参和实参的值传递



C语言中实参和  
形参之间的传递  
的唯一方式

○ 传入方式



## 4.5、C函数的传入方式： 形参和实参的值传递

说明：

- 实参必须有确定的值；
- 形参必须指定类型；
- 形参与实参**类型一致，个数相同**；
- 若形参与实参类型不一致，自动按形参类型转换；
- 形参在函数被调用前不占内存；函数调用时为形参分配内存；调用结束，内存释放。



## 4.6、C函数的传出方式：返回值

### return语句用于函数中：

- 使函数立刻退出，程序的运行返回给调用者；
- 可以向调用者传出一个返回值。

### return语句的一般形式：

- **return(表达式);**
- **return 表达式;**
- **return;**



## 4.7、C函数的双向传递： 全局变量、模拟引用传递

### 全局变量：

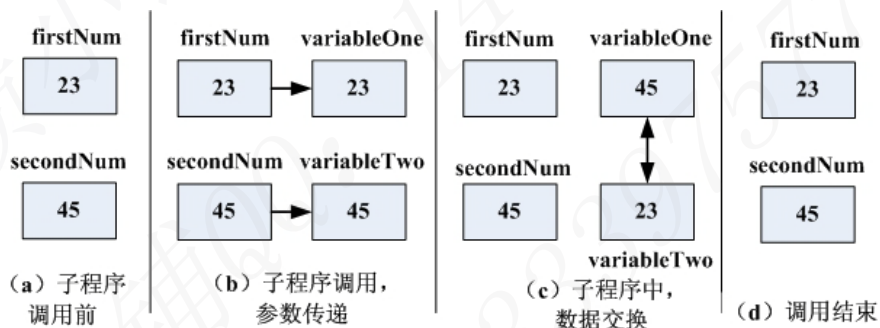
- 所有函数共享；
- 容易被误用引发错误；占据内存时间长。

### 地址类形参模拟引用传递，实现双向通信：

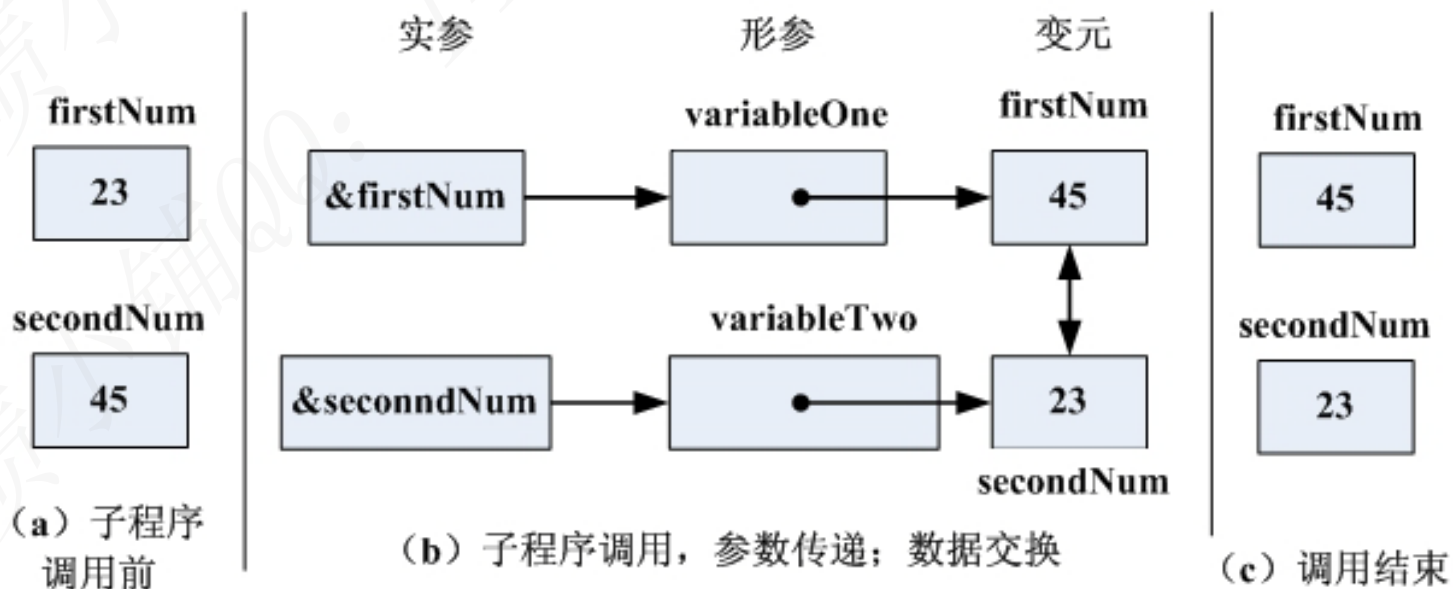
- 指针或数组名做函数形参；
- 例如：函数形参 $p$ 为一级指针类型，用于传入信息
- 则 $*p$ 为被调函数和主调函数共享；传入、传出信息单元 $*p$ 。



## 4.7、C函数的双向传递： 地址参数模拟引用传递

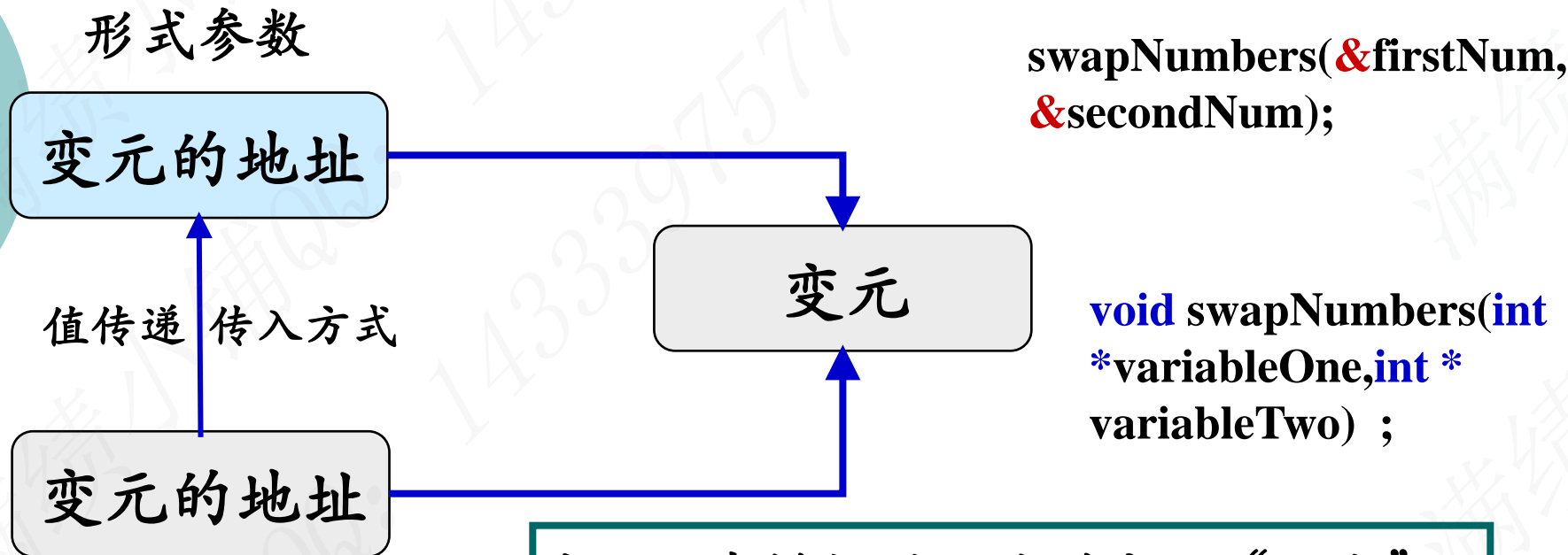


将形参变量的类型从int  
改为 int \*





## 4.7、C函数的双向传递： 地址参数模拟引用传递



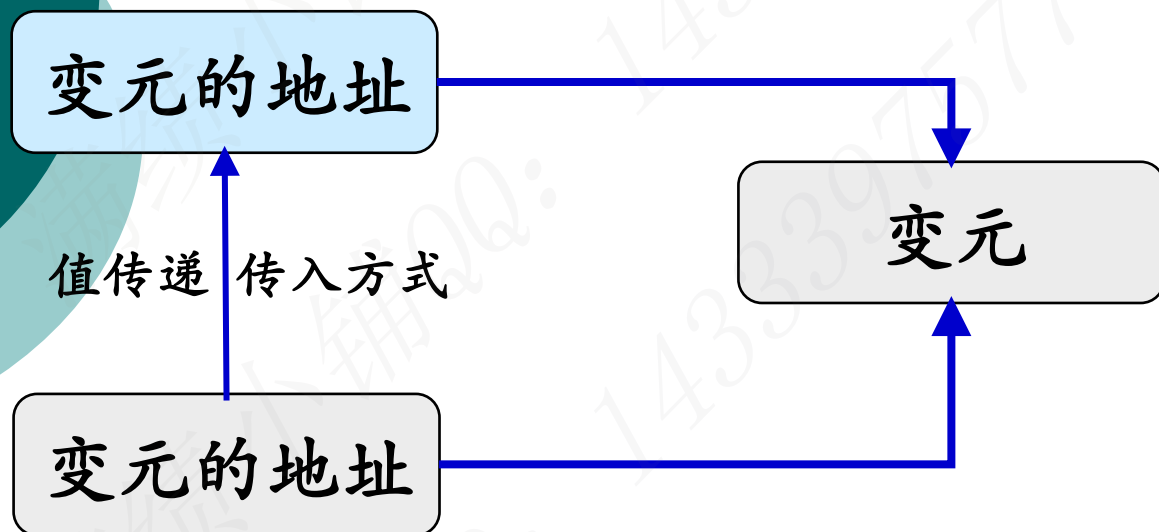
地址形参模拟引用传递实现“双向”传递：

- 本质上仍是值传递；
- 形参和实参“共享内存”：形参和实参指向同一个对象。



## 4.7、C函数的双向传递： 地址参数模拟引用传递

形式参数



实在参数

地址形参模拟引用传递实现“双向”传递，函数带出多个结果：

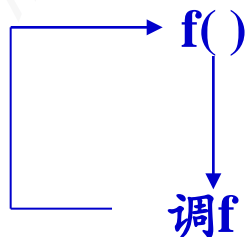
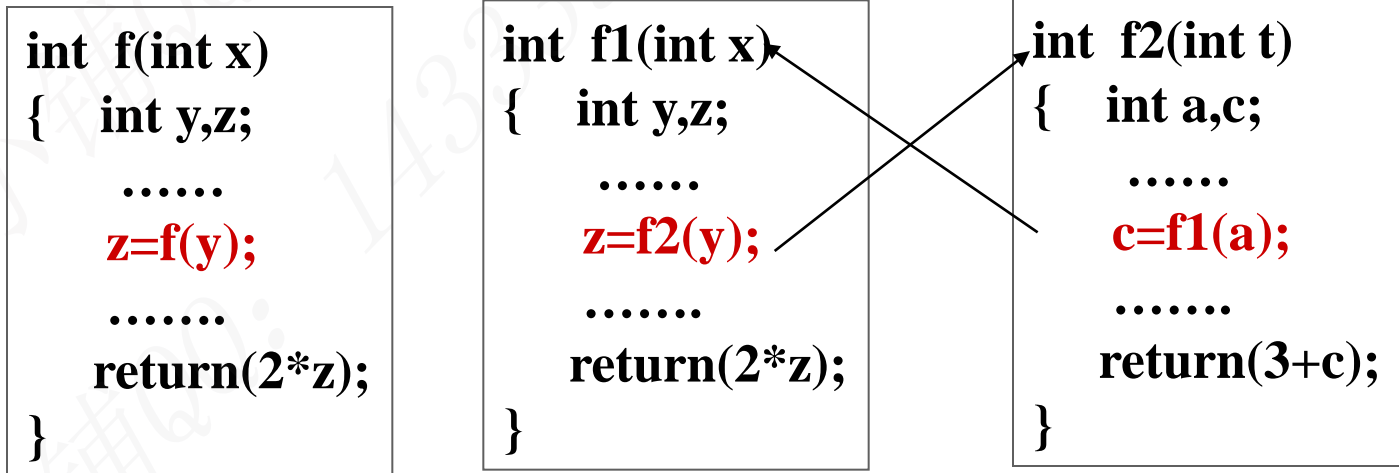
- 变元（结果）为简单类型，则形参为一级指针；
- 变元（结果）为一级指针，则形参为二级指针；



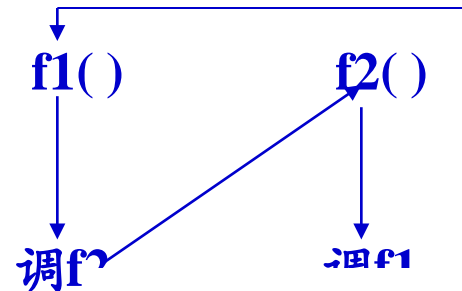
# 4.8、递归函数

## 递归调用

定义：函数直接或间接的调用自身叫函数的递归调用



直接递归



间接递归



## 4.9、数据模块化

### 作用域：可视性

- 一个标识符的作用域指程序中可以访问（可见到）该标识符所代表的变量的区域。
- 作用域：局部可视、全局可视；

### 存储周期：生存周期，或指存在时间

- 标识符所代表的变量存在于内存中的时间；
- 存储周期：程序的整个执行时间存在；需要时创建，存在时间很短。



## 4.9、数据模块化

	局部变量			全局变量	
存储类别	auto	register	static局部	static全局	全局
存储周期	自动存储期		静态存储期		
存储区	动态区	寄存器	静态存储区		
生存期	函数调用开始至结束		程序整个运行期间		
赋初值	每次函数调用时		编译时赋初值，只赋一次		
未赋初值	不确定		自动赋初值0或空字符		

- ◆ 局部变量默认为auto型
- ◆ register型变量个数受限,且不能为long, double, float型
- ◆ 局部static变量具有全局寿命和局部可见性
- ◆ 局部static变量具有可继承性
- ◆ extern不是变量定义,可扩展外部变量作用域

满绩小铺: 1499397577, 搜集整理不易, 自用就好, 谢谢!



# 5、编译预处理



## 5.1、什么是编译预处理

预处理命令：在对源程序编译之前的处理命令。

种类

- 宏定义 `#define`
- 文件嵌入 `#include`
- 条件编译 `#if--#else--#endif`等
- 其他

格式：

- “#”开头
- 占单独书写行
- 语句尾不加分号



## 5.2、宏(一般形式, 不带参数的宏)

宏定义的一般形式

**#define 宏名 【宏体】**

- 功能：用指定标识符(宏名)代替字符序列(宏体)；
- 定义位置：任意(一般在函数外面)；
- 良好习惯：习惯上，用大写字母和下划线来为宏命名。

宏替换：在源程序中发现与**宏名相同的标识符**后，用宏体替换之。

宏定义取消的一般形式

**#undef 宏名**





## 5.2、宏（不带参数的宏，注意事项）

例 #define WIDTH 80  
#define LENGTH WIDTH+40  
var=LENGTH\*2;  
宏展开：var= 80+40 \*2;

例 #define WIDTH 80  
#define LENGTH (WIDTH+40)  
var=LENGTH\*2;  
宏展开：var= (80+40) \*2;

常见错误：宏体没有用圆括号括起来。

例 #define PI 3.14159  
printf("2\*PI=%f\n",PI\*2);  
宏展开：printf("2\*PI=%f\n",3.14159\*2);

注意点：源程序中只有相同的标识符才被替换。



## 5.3、宏(带参数的宏，类函数宏)

---

类函数宏定义的一般形式

**#define 宏名(参数表) 【宏体】**

宏替换：在源程序中发现与宏名相同的标识符后，用宏体替换之。形参用实参换，其它字符保留。



## 5.3、宏（类函数宏，注意事项）

例 #define POWER(x) x\*x

x=4; y=6;

z=POWER(x+y);

宏展开：z=x+y\*x+y;

例 #define POWER(x) ((x)\*(x))

宏展开：z=((x+y)\*(x+y));

常见错误：宏体和参数没有用圆括号括起来。

例 #define S (r) PI\*r\*r

相当于定义了不带参宏S,代表字符串“(r) PI\*r\*r”

常见错误：宏名和参数之间不能有空格。



# 6、数组和串



# 6.1、一维数组

**type array\_name[size]**

- **type**: 数组的基类型，各数组元素的数据类型；
- **array\_name**: 数组名，数组内存的起始地址，常量；
- **array\_name[下标表达式]**: 下标从0开始；

```
int score[30];
```

85
67
.....
98

数组：一组同类型数据的有序集合。

数组：一段连续被等分的内存空间。

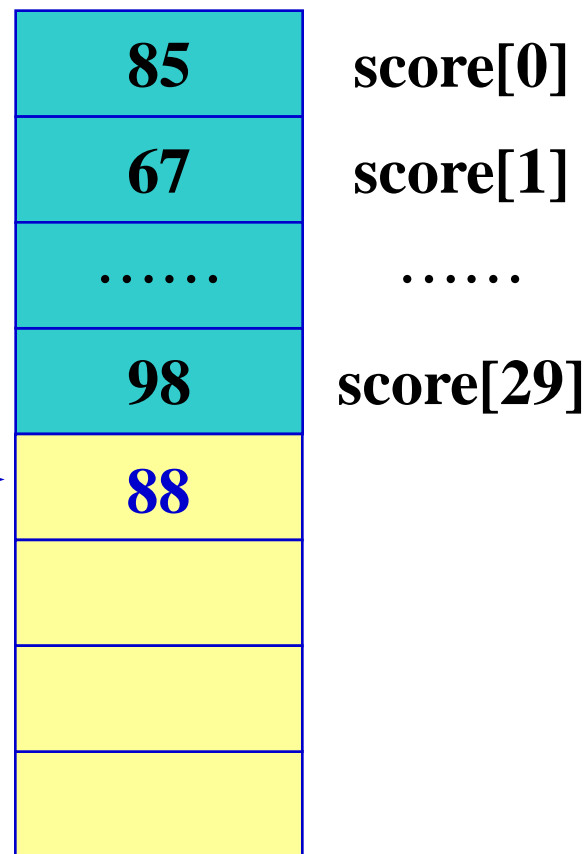


## 6.2、访问一维数组 (越界使用存储单元)

越界访问存储单元：C编译系统不检查。

```
int score[30];
```

```
score[30]=88;
```



开发程序时预防这种错误是最好的！



## 6.2、访问一维数组 (越界使用存储单元)

越界访问存储单元：C编译系统不检查。

**错误，后果严重！**

- 程序运行看起来正确！
- 程序崩溃；
- 硬件错误：内存故障；  
总线错误；  
分段错误。

**开发程序时预防这种错误是最好的！**



## 6.2、C语言中的字符串

**C语言中的字符串：**一个以空字符（'\0'）作为结束符的字符数组。

→ **字符串结束标志：** '\0'

→ **字符串的值：**字符串中第一个字符的地址；

→ **C语言中的表示方法：**字符数组；字符指针。





## 6.3、 字符数组

基类型      字符数组名      串  
   长度  
初始化部分（可选）

**char president[18]="Abraham Lincoln";**

'A'	'b'	'r'	'a'	'h'	'a'	'm'	' '	'L'	'i'	'n'	'c'	'o'	'l'	'n'	'\0'		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17



## 6.4、 字符数组的使用

---

- 字符串的输入输出；
- 字符串处理常用库函数；
- 注意内存访问越界问题；
- 字符串的结束标记。



## 6.5、多维数组

---

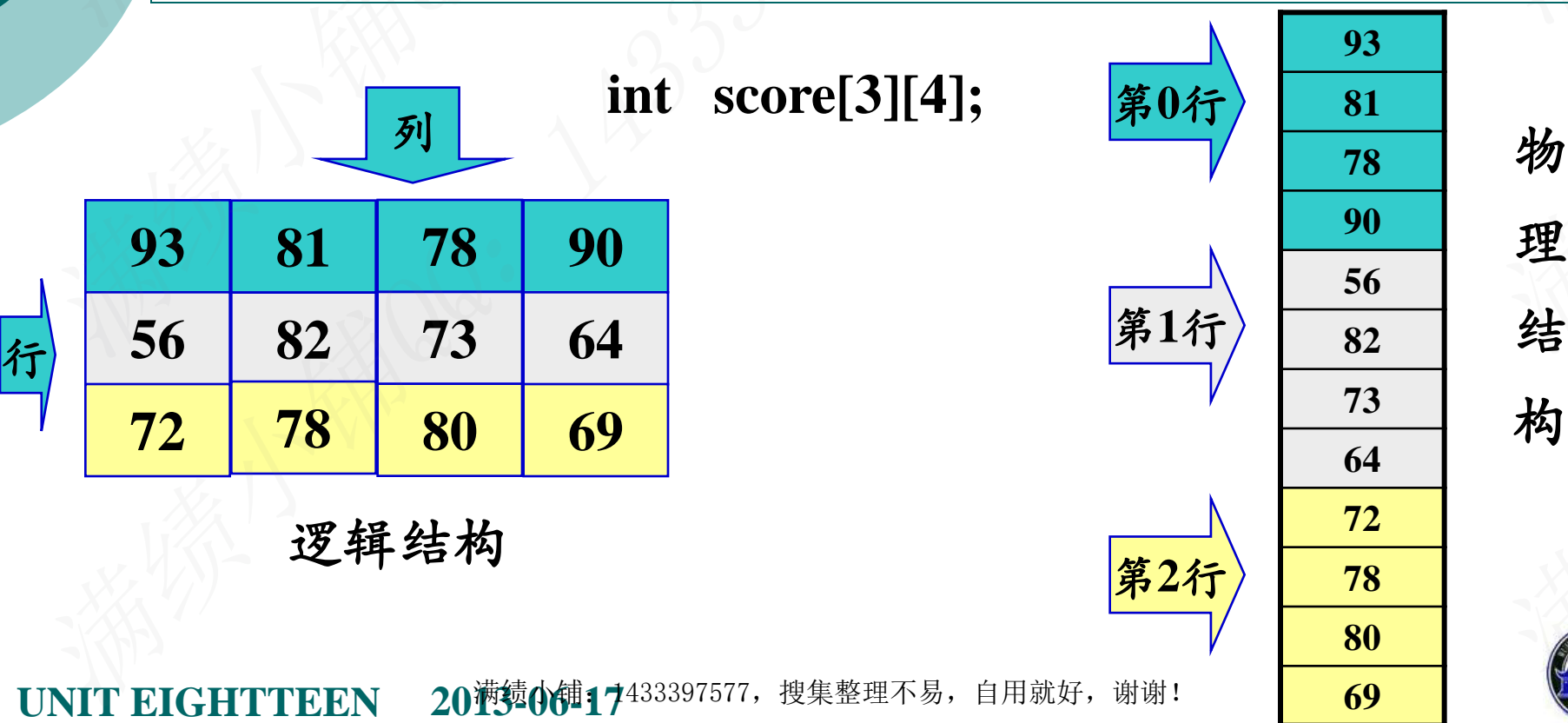
- 二维数组；
- 深入理解多维数组；
- 字符串数组。



## 6.6、二维数组的逻辑结构和物理结构

**type array\_name[size1][size2]**

- 物理结构（存储结构）：按行优先的顺序存储结构。



## 6.7、 进一步理解二维数组 (从逻辑结构来看)

```
type array_name[下标1][下标2];
```

从逻辑结构上来理解：

- 矩阵；
- 数组的数组。



## 6.8、 进一步理解二维数组 (从物理结构来看)

```
type array_name[下标1][下标2];
```

从物理结构上来理解：

- 数组的数组：二维数组是一维数组的一维数组；
- C语言的观点：本质上只有一维数组，多维数组是一维数组的特例。

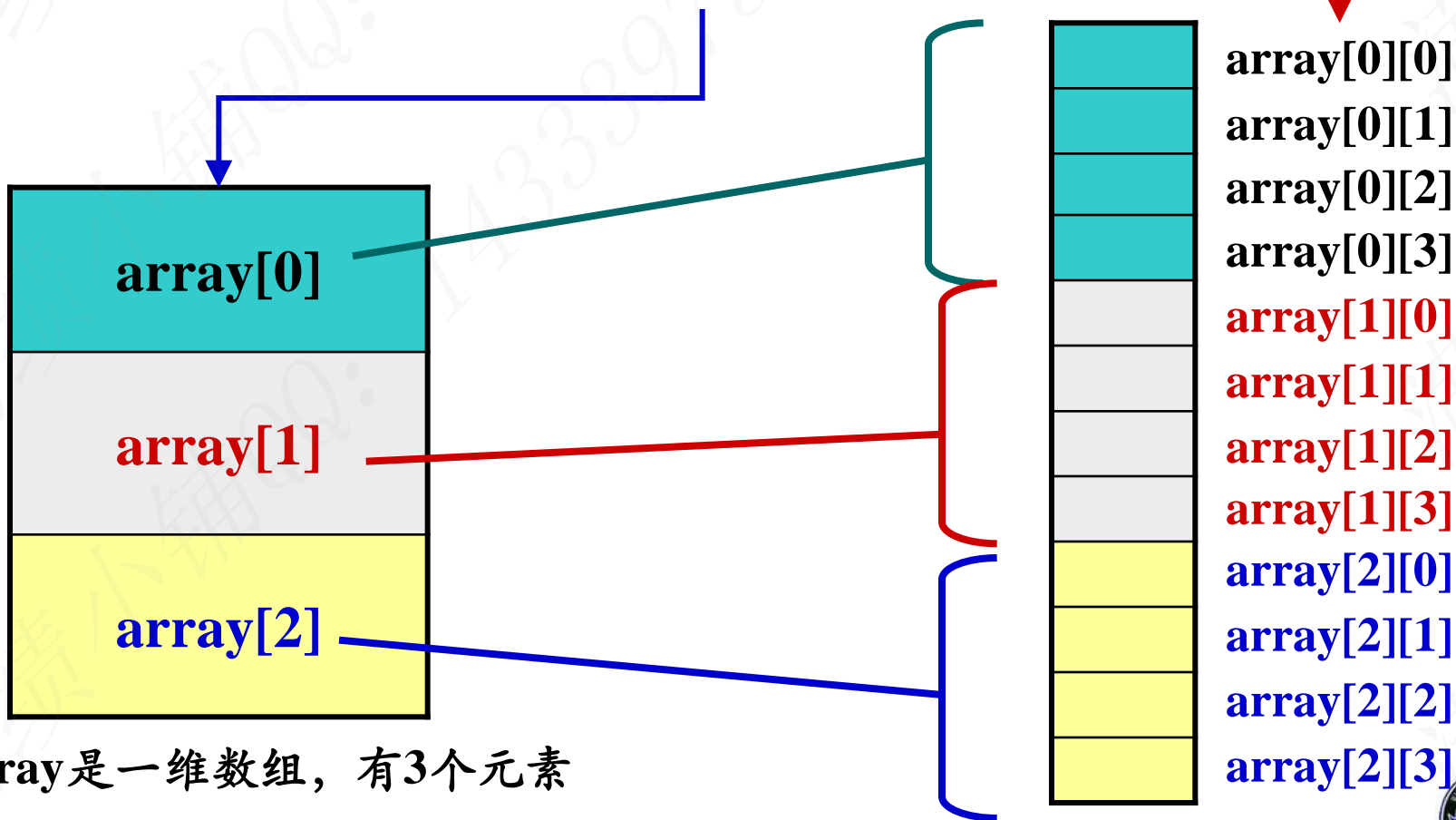


## 6.8、 进一步理解二维数组 (从物理结构来看)

array[i][j]

是short  
int类型的。

short int array[3][4];



array是一维数组，有3个元素



# 7、指 针





# 7.1、定义指针变量

**type \*name**

**name**: 指针变量;

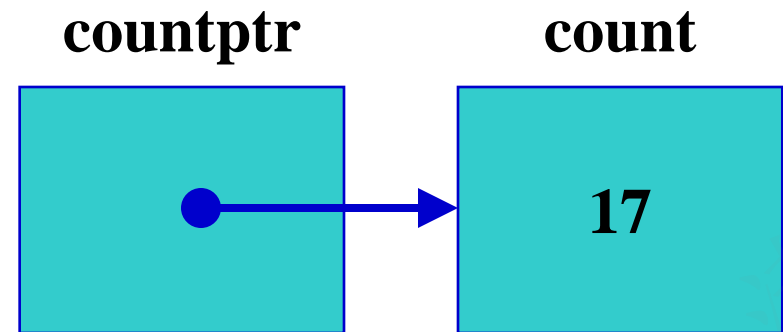
**\*name**: 指针变量**name**指向的对象;

- 定义的是指针变量**name**;
- 没有定义指针指向的对象, **\*name**.

```
int count , *countptr;
```

```
countptr = &count;
```

```
*countptr = 17 ;
```



**\*countptr**



## 7.2、 定义指针变量 (使用要点)

**type \*name**

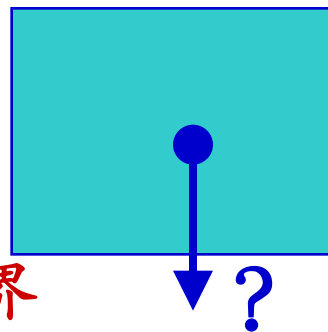
- **name**: 指针变量, 编译系统自动分配;
- **\*name**: 指针变量指向的对象, 编译系统不会自动分配。
- 指针变量在使用前必须初始化: 确保指针变量指向明确的、合法的對象。

```
int count , *countptr;
```

```
*countptr = 17 ;
```

✗ 访问非法内存: 内存越界

countptr



count



## 7.3、 指针运算符

### &variable

地址运算符&，变量variable的地址：

- **variable**： 变量，有左值的表达式。

### \*address

指针运算符\*，内存地址为address的数据单元：

- **address**： 指针（地址）表达式。

```
int count , *countptr;  
countptr = &count;  
*countptr = 17 ;
```

满绩小铺 17433397577, 搜集整理不易, 自用就好, 谢谢!



## 7.4、 指针变量初始化

```
type *name = initialization;
```

### ○ 不要使用未初始化的指针变量:

- 全局/静态指针变量未初始化，系统自动初始化为NULL，此时指向对象不存在；
- 自动指针变量未初始化，原来存储在内存空间中数据被保留；此时指针指向的对象没有意义。

### ○ initialization:

- NULL：不指向任何对象
- 已经定义的地址或指针；
- 动态分配内存。



## 7.5、指针算术运算

### (指针加、减一个整数)

---

**$p1+d$**

- 该表达式的值为 $p1$ 当前执行对象的下面第 $d$ 个对象。
- $+d$ 表示向后移动 $d$ 个 $*p1$ 的内存空间，因此， $p1+d$ 的值等于 $p1+sizeof(*p1)*d$ ;
- $-d$ 不是先前移动 $d$ 个 $*p1$ 的内存空间大小。



## 7.5、指针算术运算

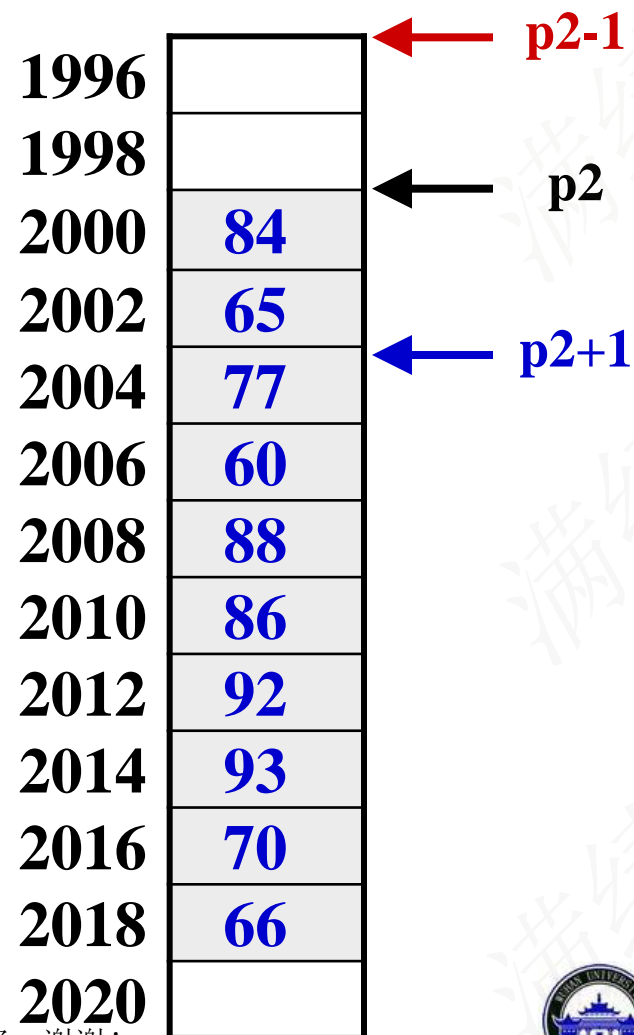
### (指针加、减一个整数)

例如，有如下定义：

```
long int *p2;  
short score[10]  
    = {84, 65, 77, 60, 88, 86, 92, 93, 70, 66};  
p2 = (long int *) &score[0];
```

```
p2++;    p2的值为?    2004  
p2--;    p2的值为?    1996
```

因为：\*p2是long类型的，占4字节。



## 7.6、 指针算术运算

### (两个指针的减法)

$p1 - p2$

- $p1$ 、  $p2$ 必须是同类型的指针变量;
- 结果为整数:  $p1$ 和 $p2$ 指向的对象之间间隔的数据个数;
- $p1$ 、  $p2$ 通常指向同一个数组空间。



## 7.7、 指针运算 (赋值)

---

**ptr1 = ptr2**

- **ptr1**: 指针变量;
- **ptr2**: 指针变量或者地址表达式;
- **限制条件**: ptr1和ptr2两者数据类型必须相同。





## 7.8、 指针运算 (转换)

### void \*指针转换

- void \*指针代表原始内存，因此又称一般指针。
- 转换为void \*指针，或者将void \*指针转换为其他类型指针，无需使用强制类型转换。

### 其他类型指针转换

- void \*指针之外的其他类型指针相互转换时，必须使用强制类型转换。



## 7.9、指针比较

若p1和p2指向同一数组，则：

**p1<p2** 表示p1指的元素在前；

**p1>p2** 表示p1指的元素在后；

**p1==p2** 表示p1与p2指向同一元素。

若p1与p2不指向同一数组，比较无意义；

**p==NULL或p!=NULL**



## 7.10、一维数组与一级指针

**int \*p 与 int q[10]**

- 数组名是指针（地址）常量；
- $p+i$  是  $q[i]$  的地址；
- 数组元素的表示方法：下标法和指针法， 即若  $p=q$ ，  
则  $p[i] \Leftrightarrow q[i] \Leftrightarrow *(p+i) \Leftrightarrow *(q+i)$
- 形参数组实质上是指针变量，即  $\text{int } q[ ] \Leftrightarrow \text{int } *q$
- 系统只给  $p$  分配能保存一个指针值的内存区（一般2字节）；而给  $q$  分配  $2*10$  字节的内存区；
- 指针和数组存储结构不同。

资源来源：14309557 搜集整理不易，自用就好，谢谢！



## 7.11、 数组与指针 “等价”,不是 “相同”

不表示它们相同，甚至也不能互换。

可以用指针方便的访问数组或者模拟数组。

**等价的基础：**一旦数组出现在表达式中，编译器会隐式地生成一个指向数组第一个成员的指针，就像程序员写出了`&a[0]`一样。

例外的情况是，数组为`sizeof`或`&`操作符的操作数，或者为字符数组的字符串初始值。



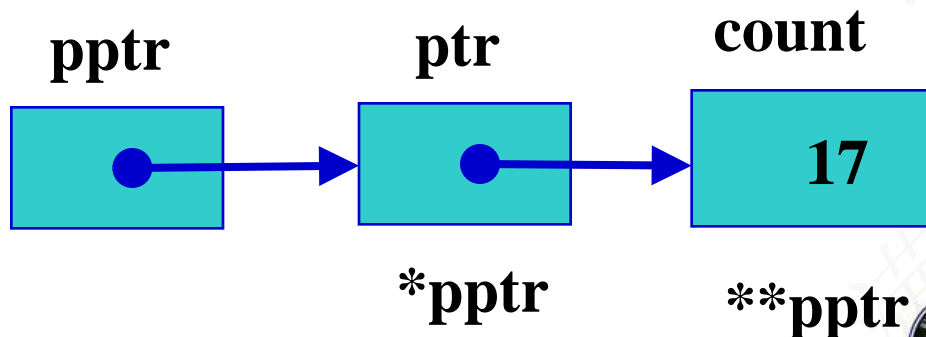
## 7.12、二级指针（定义）

**type \*\*name**

**多级间址：**多级间接寻址、指向指针的指针；

- **name**：二级指针变量名称；
- **\*\***：二级指针变量的标志；
- **type \***：指针变量**name**直接指向的对象的数据类型；
- **type**：**\*name**指向的数据对象数据类型。

```
int count, *ptr, **pptr;  
ptr = &count;  
pptr = &ptr;  
**pptr = 17;
```



## 7.13、多级指针

数组  
指针

```
int (*q)[4], a[2][4];  
q = a;
```

指针  
数组

```
int *p[4];
```

二维数组形参实际上是一维数组指针变量，

即 `int x[][10] ⇔ int (*x)[10]`



# 8、 结构、 联合 和枚举

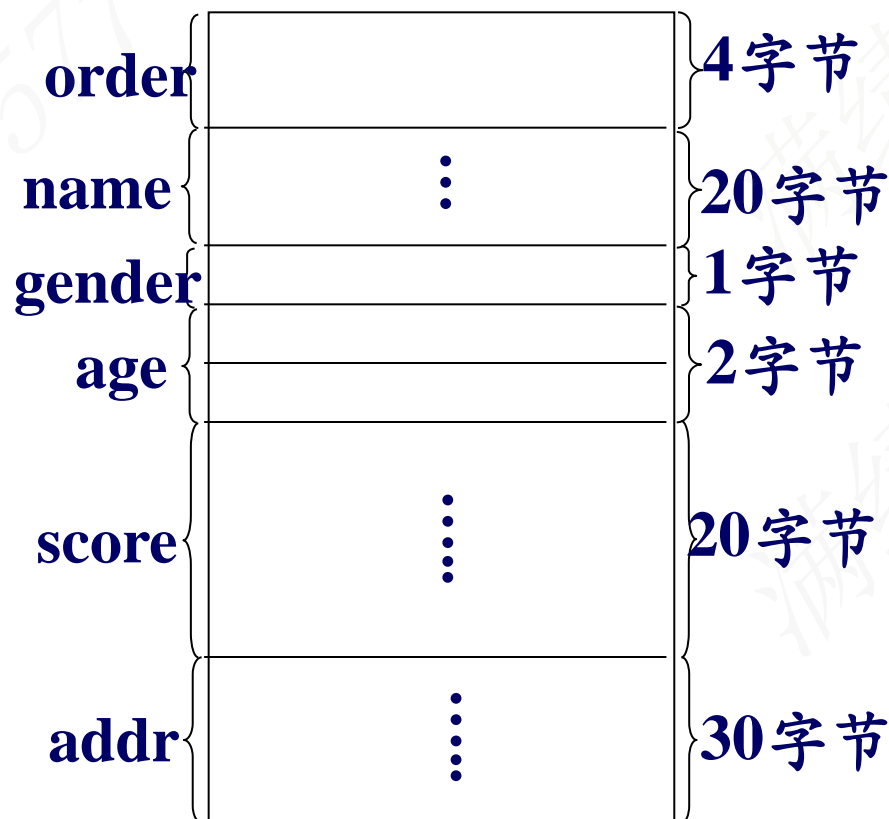


# 8.1、定义结构类型

例 struct **student**  
{ long int order;  
char name[20];  
char gender;  
short int age;  
int score[10];  
char addr[30];  
};

结构类型定义的**作用域**

结构类型定义描述结构的组织形式，**不分配内存**





## 8.2、 结构类型变量的引用

### 结构类型变量.成员名

结构类型变量引用的基本原则：

- 不能整体引用；
- 只能引用结构类型变量的成员；
- `·`：成员引用运算符；
- 赋值运算：两个同类型的结构类型变量可以相互赋值；
- 必须逐级引用结构类型变量的成员。



## 8.2、结构类型变量的引用

### 存取结构类型变量的成员名

例

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

stu1.num = 10;

stu1.score = 85.5;

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



## 8.3、 指向结构的指针

**struct 结构类型名 \*结构类型指针变量名;**

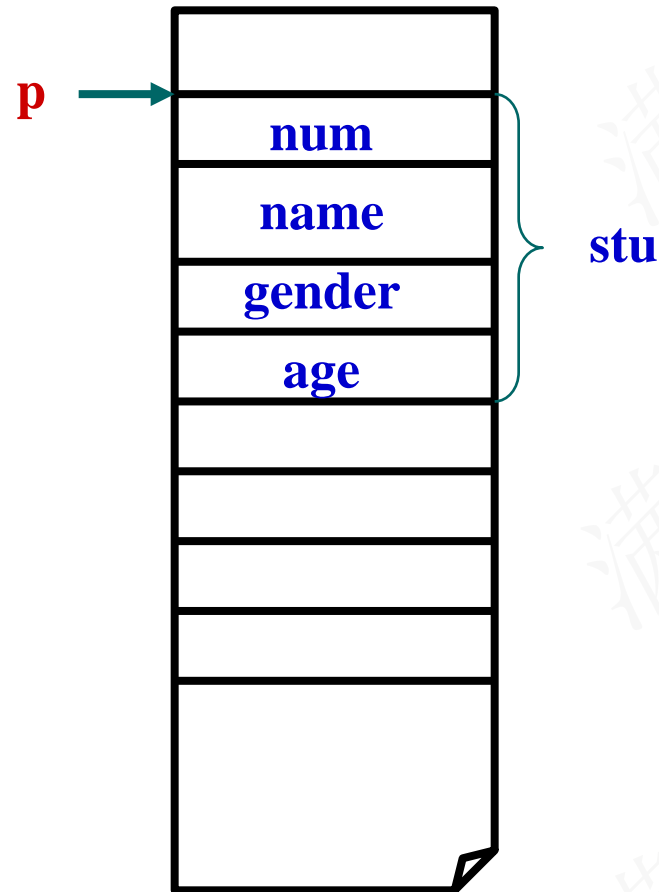
->: 指向运算符;

- 结构类型的指针->成员名: 等价于  
(\*结构类型的指针).成员名



## 8.3、指向结构的指针

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



以下三种形式等价：

**p->num**

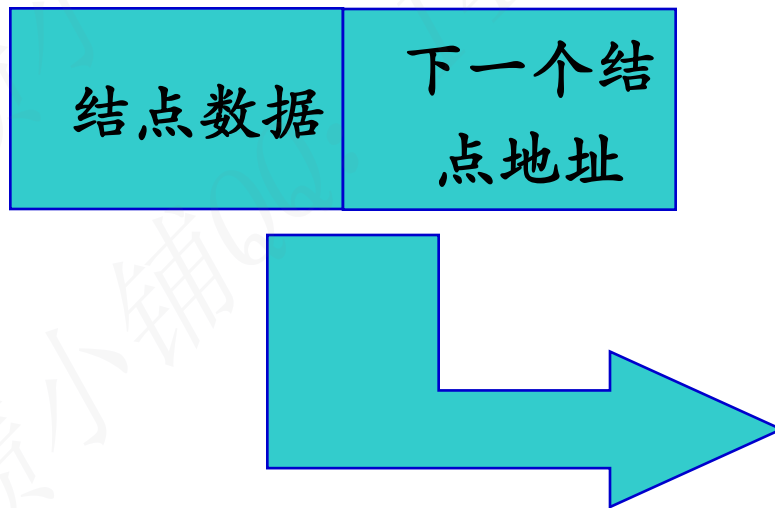
**(\*p).num**

**stu.num**



## 8.4、动态数据结构 (链表)

**链表**：链接方式存储的线性表简称为链表 (Linked List) 。



链表结点类型：

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next;
```

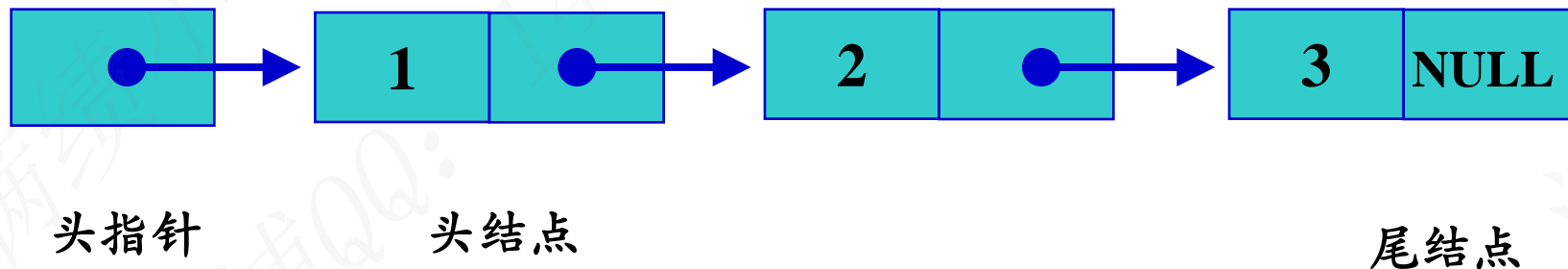
```
};
```



## 8.5、动态数据结构

### (链表：链表的类型)

单链表：每个结点只有一个链域的链表。



○ 单链表的基本操作：创建、遍历、删除和插入

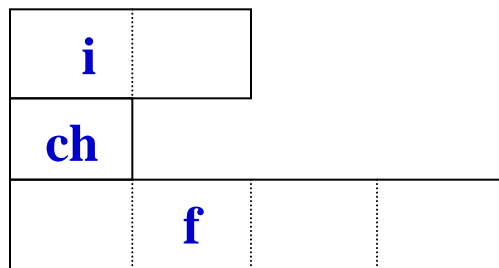


## 8.6、联合类型

### 联合类型：

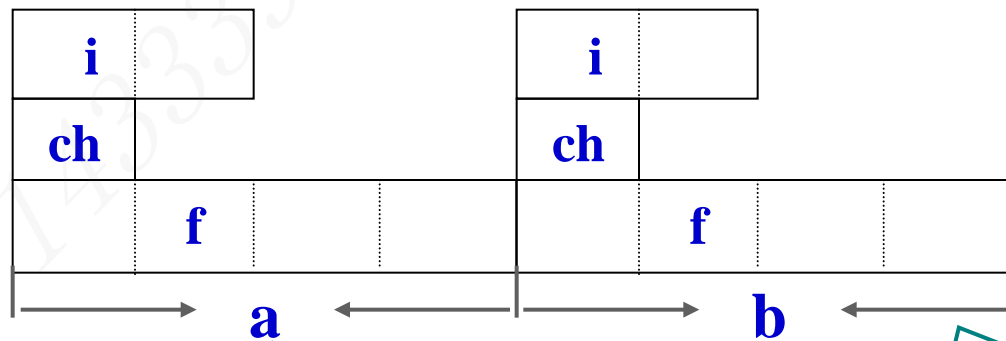
- 共用体；
- 多个成员（变量）共用同一段内存；
- C语言对同一段内存的多模式解读。

例 **union data**  
**{ int i;**  
**char ch;**  
**float f;**  
**};**



## 8.6、联合类型变量

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



联合变量任何时刻  
只有一个成员存在

联合变量定义分配内存，  
长度=最长成员所占字节数





## 8.7、联合类型变量的引用

以下三种形式均可访问联合类型变量的成员：

- 联合类型指针->成员名
- (\*联合类型指针).成员名
- 联合类型变量名.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



## 8.8、枚举类型的定义

**枚举常量：**命名的整数常量。

枚举类型定义

```
enum 枚举名{ 枚举常量值表 };
```

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

- 该枚举名为weekday，枚举值共有7个，即一周中的七天。
- 凡被说明为weekday类型变量的取值只能是七天中的某一天。
- 类型名enum weekday



## 8.8、枚举类型变量的使用

编译程序把第一个枚举常量赋值为0，以后依次赋值为1、2.....

例

```
enum color{red,green,blue,yellow,white}select;
```

red的值为0，green、blue、yellow和white取值分别为1、2、3、4。

```
select=red;
```

select被赋值为0。



## 8.9、typedef定义类型别名

**typedef type name;**

定义类型别名：

- **type**: 原有数据类型名;
- **name**: 类型别名;
- 仅仅定义类型别名, 没有定义新的数据类型。

例 INTEGER a,b,c;  
REAL f1,f2;

typedef int INTEGER;  
typedef float REAL;



int a,b,c;  
float f1,f2;

说明:

1. typedef **没有创造**新数据类型
2. typedef 是定义类型, **不能定义变量**
3. typedef 与 **define** 不同

## 8.9、typedef的使用 (定义步骤)

---

### typedef定义类型步骤

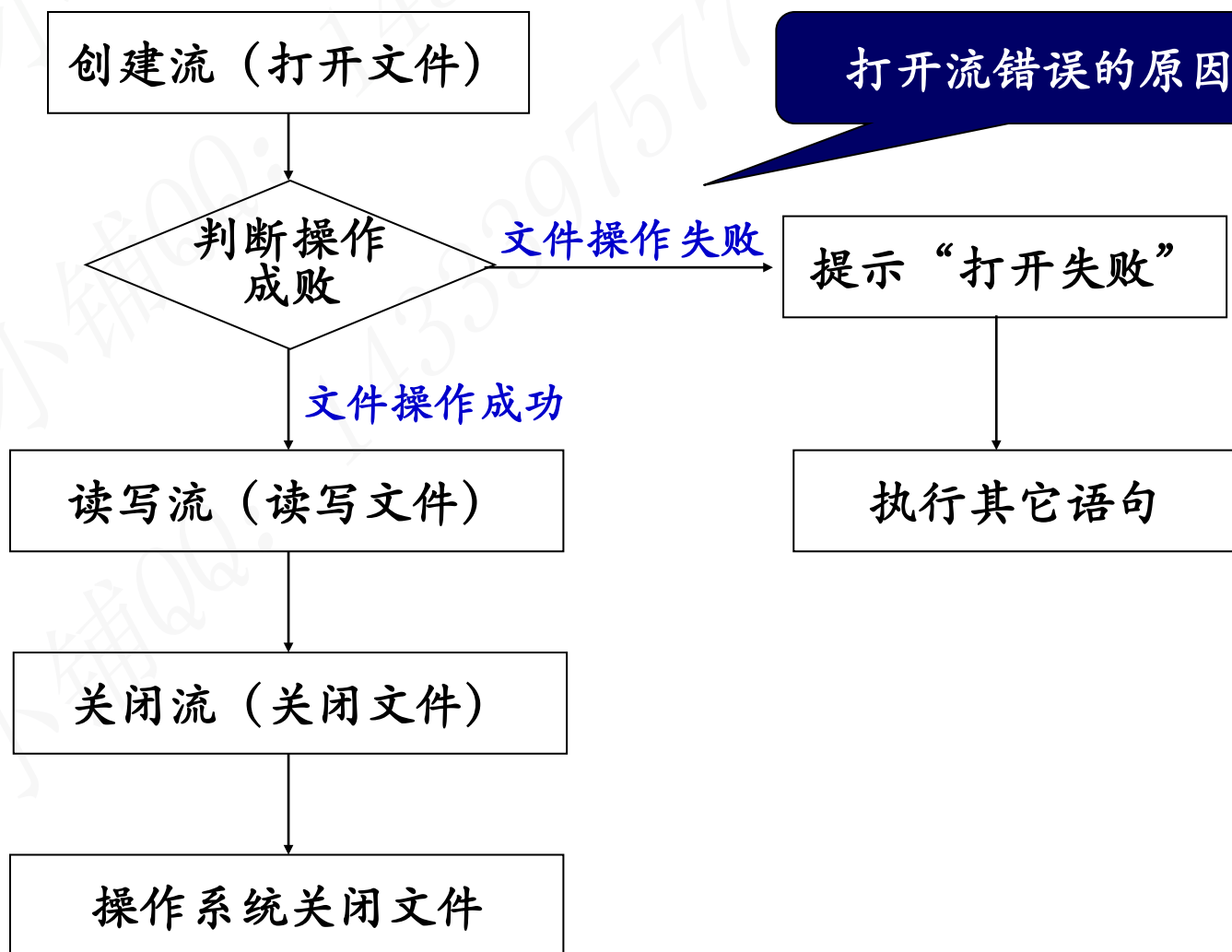
- 按定义变量方法先写出定义体 如 `int i;`
- 将变量名换成新类型名 如 `int INTEGER;`
- 最前面加typedef 如 `typedef int INTEGER;`
- 用新类型名定义变量 如 `INTEGER i,j;`



# 9、流和文件



## 9.1、C语言文件操作基本流程



## 9.2、定义和打开流

```
FILE *filePtr;  
filePtr = fopen( filename , mode);
```

文件使用方式	含义
"r/rb" (只读)	为输入打开一个文本/二进制文件
"w/wb" (只写)	为输出打开或建立一个文本/二进制文件
"a/ab" (追加)	向文本/二进制文件尾追加数据
"r+/rb+" (读写)	为读/写打开一个文本/二进制文件
"w+/wb+" (读写)	为读/写建立一个文本/二进制文件
"a+/ab+" (读写)	为读/写打开或建立一个文本/二进制文件





## 9.3、输入输出流

### 输入输出文本流:

- `fputc()` `fgetc()`
- `fputs()` `fgets()`
- `fprintf()` `fscanf()`

### 输入输出二进制流:

- `fwrite()` `fread()`

### 其他文件处理库函数:

- `feof()` `fseek()` .....

