

---

# C 语言程序设计

## 本讲要点

- C 语言的与算法概述
- C 语言的特点（记忆）
- C 语言的结构（理解）
- C 语言的编写规范（应用）
- 什么是算法（理解）
- 算法的特性（理解）
- 算法的表示（理解）
- 程序的三种基本结构（理解）

## 授课内容

经典教材推荐：

《C 程序设计》，谭浩强，清华大学出版社

## 第 1 节 C 语言与算法概述

### 什么是程序

由人事先编制的、要计算机完成的一系列操作或一个指令（或语句）序列

计算机语言分类：机器语言、汇编语言、高级语言

计算机语言表达方式较接近：

1. 英语和数学公式。
2. 机器语言
3. 汇编语言
4. 高级语言

常用的高级语言

---

---

BASIC

FORTRAN

PASC

高级

DELPHI

C

COBOL

### C 语言的发展

早期的 C 语言主要是用于 UNIX 系统。由于 C 语言的强大功能和各方面的优点逐渐为人们认识。

到了八十年代，C 开始进入其它操作系统，并很快在各类大、中、小和微型计算机上得到了广泛的使用，成为当代最优秀的程序设计语言之一。

### C 语言的特点

1、C 语言简洁、紧凑，使用方便、灵活。

（1）一共只有 32 个关键字

（2）9 种控制语句，程序书写自由，主要用小写字母表示，压缩了一切不必要的成分。

注意：在 C 语言中，关键字都是小写的。

2、运算符丰富。

共有 34 种。C 把括号、赋值、逗号等都作为运算符处理。从而使 C 的运算类型极为丰富，可以实现其他高级语言难以实现的运算。

3、数据结构类型丰富。

4、具有结构化的控制语句。

5、语法限制不太严格，程序设计自由度大。

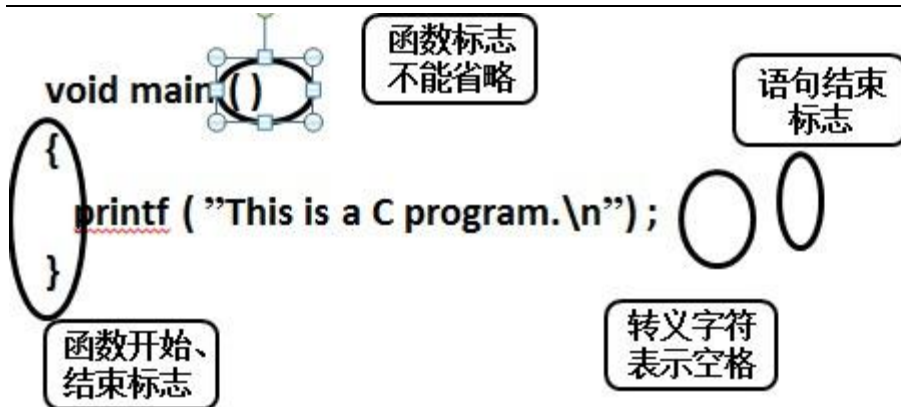
6、C 语言允许直接访问物理地址，能进行位（bit）操作，能实现汇编语言的大部分功能，可以直接对硬件进行操作。因此有人把它称为中级语言。

7、生成目标代码质量高，程序执行效率高。

8、与汇编语言相比，用 C 语言写的程序可移植性好。

### 第一个 C 语言程序

---



例

```
void main() /*求两个数的和*/  
{  
    int a,b,sum;  
    a=123;b=456;  
    sum=a+b;  
    printf("sum=%d\n",sum);  
}
```

函数体

注释

声明变量

## C 程序的结构

1) 一个 C 程序由一个主函数和 $\geq 0$  个其它的函数组成，每个函数实现某特定功能，函数间先后不限制，执行程序总是从主函数开始到主函数结束

2) 每个函数结构如下：

[函数类型]<函数名> ([形参 1][, 形参 2]……)

[形参定义; ]

{ [声明部分]

[执行部分]}

3) 可用 `/* */` 把注释内容括起来插入在程序中任何位置。应培养写注释习惯。

书写程序时应遵循的规则

1. 一个说明或一个语句占一行。

2. 用 `{ }` 括起来的部分，通常表示了程序的某一层结构。`{ }` 一般与该结构语句的第一个字母对齐，并单独占一行。

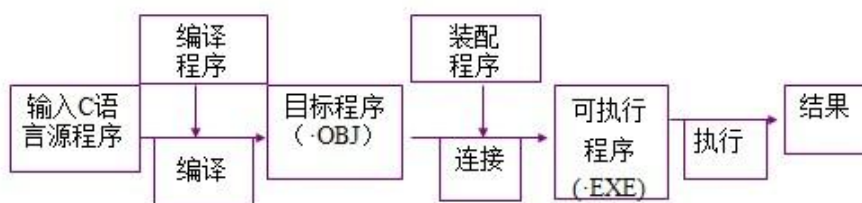
3. 低一层次的语句或说明可比高一层次的语句或说明缩进若干格后书写。以便看起来更加清晰，增加程序的可读性。

---

---

```
{  
    语句内容  
    {  
        语句内容  
        {  
            语句内容  
        }  
    }  
}
```

### 程序的编译和执行



## 算法的概述

为解决一个问题而采取的方法和步骤，就称为算法。

1. 计算机算法：计算机能够执行的算法。
2. 计算机算法可分为两大类：
3. 数值运算算法：求解数值
4. 非数值运算算法：事务管理领域。

一个程序应包括：

对数据的描述，程序中要指定数据的类型和数据的组织形式，即数据结构。

对操作的描述，即操作步骤，也就是算法。

计算机程序的构成：

程序=算法+数据结构

例：求  $1 \times 2 \times 3 \times 4 \times 5$ 。

最原始方法：

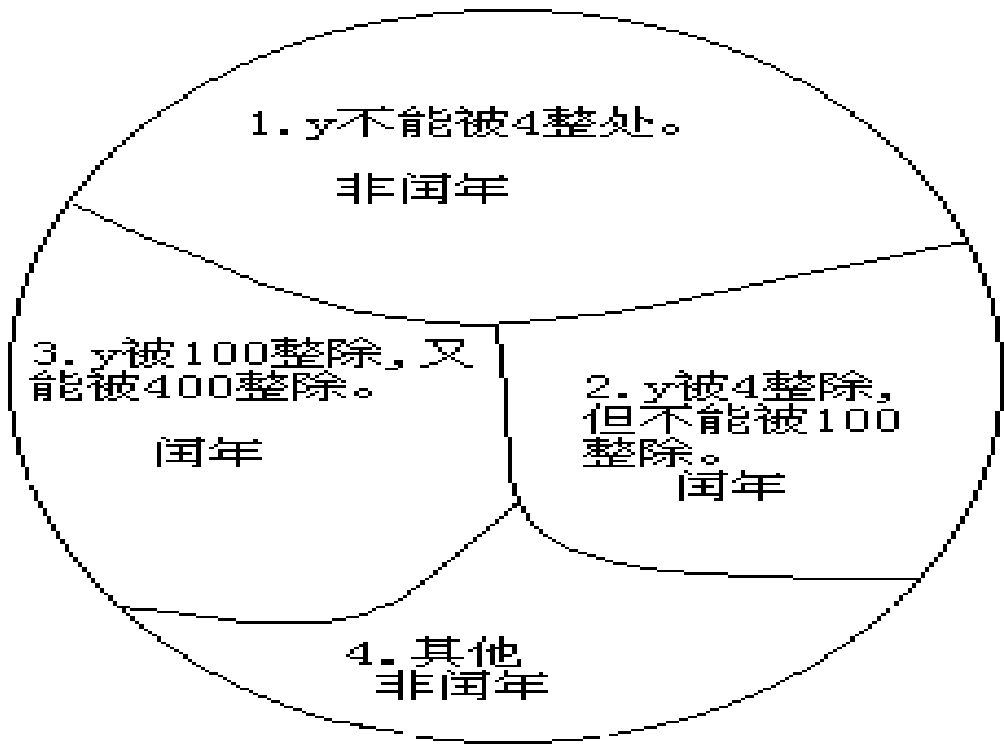
步骤 1：先求  $1 \times 2$ ，得到结果 2。

步骤 2：将步骤 1 得到的结果乘以 3，得到结果 6。

---

步骤 3：将 6 再乘以 4，得 24。  
步骤 4：将 24 再乘以 5，得 120。  
算法结束。

简单算法举例



改进的算法 (使用变量)  
S1 (即 Step 1, 下同)：使  $t=1$   
S2: 使  $i=2$   
S3: 使  $t \times i$ , 乘积仍然放在在变量  $t$  中, 可表示为  $t \times i \rightarrow t$   
S4: 使  $i$  的值+1, 即  $i+1 \rightarrow i$   
S5: 如果  $i \leq 5$ , 返回重新执行步骤 S3 以及其后的 S4 和 S5; 否则, 算法结束。

简单算法举例

例：求  $1 \times 2 \times 3 \times 4 \times 5$   
例：判定 2000 — 2500 年中的每一年是否闰年，将结果输出。  
1) 能被 4 整除，但不能被 100 整除的年份；  
2) 能被 100 整除，又能被 400 整除的年份；  
S1: 2000  $\rightarrow y$  (设  $y$  为被检测的年份)  
S2: 若  $y$  不能被 4 整除，输出  $y$  “不是闰年”，然后转到 S6  
S3: 若  $y$  能被 4 整除，不能被 100 整除，则输出  $y$  “是闰年”，然后转到 S6

---

S4:若  $y$  能被 100 整除, 又能被 400 整除, 输出  $y$  “是闰年” 否则输出  $y$  “不是闰年”, 然后转到 S6

S5:输出  $y$  “不是闰年”。

S6: $y+1 \rightarrow y$

S7:当  $y \leq 2500$  时, 返回 S2 继续执行, 否则, 结束。

返回

## 算法的特性

有穷性:

一个算法应包含有限的操作步骤而不能是无限的。

确定性:

算法中每一个步骤应当是确定的, 而不能应当是含糊的、模棱两可的。

有输入:

有零个或多个输入。

有输出:

有一个或多个输出。

有效性:

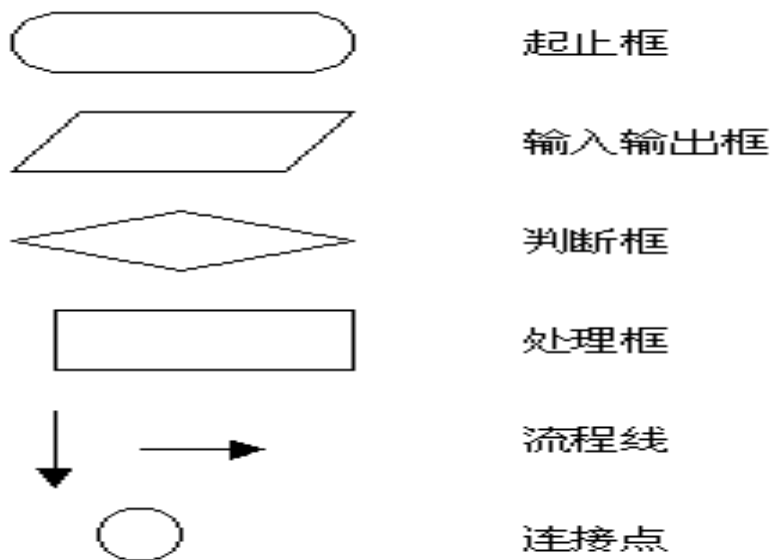
算法中每一个步骤应当能有效地执行, 并得到确定的结果。

## 算法的表示

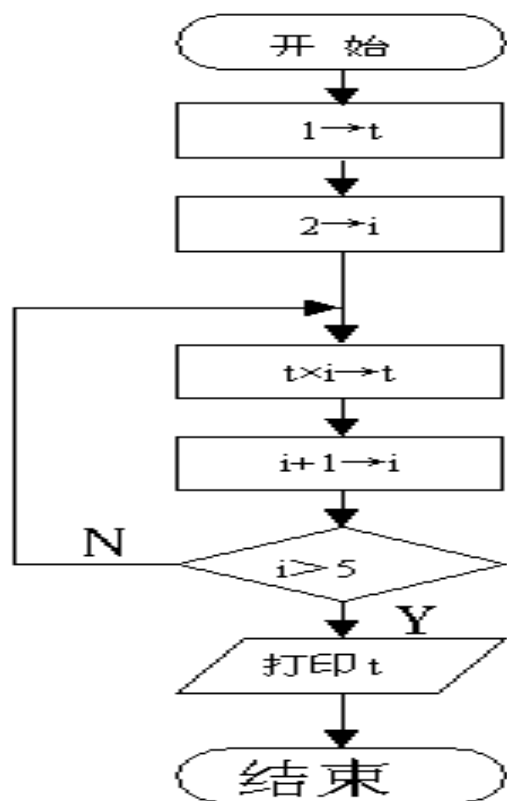
- 用自然语言表示算法
- 用流程图表示算法
- 用 N-S 流程图表示算法
- 用伪代码表示算法
- 用计算机语言表示算法

## 流程图元素

---

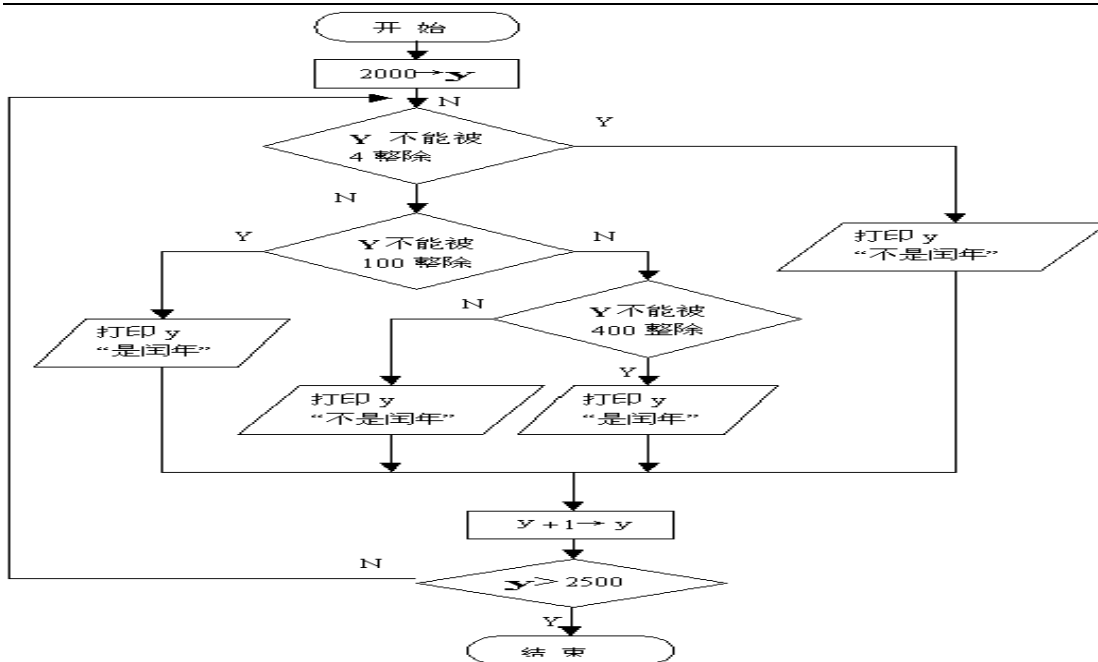


例：将  $5!$  的算法用流程图表示。



例：将判定闰年的算法用流程图表示。

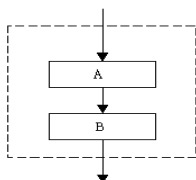
---



## 程序的三种基本结构

### 1、顺序结构:

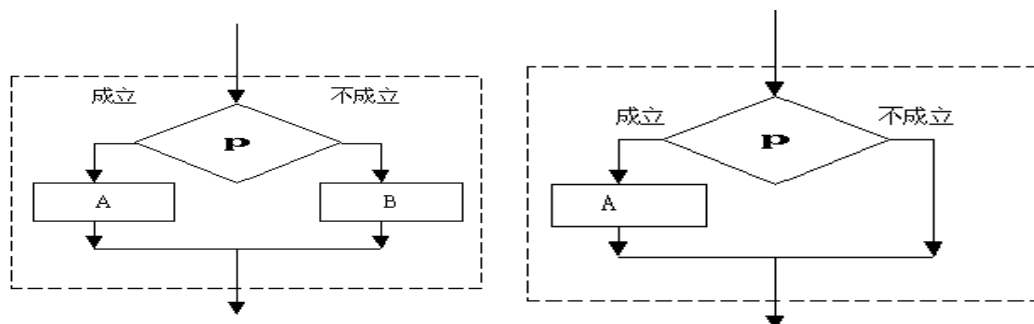
先执行 A、后执行 B、脱离本结构。



### 2、选择结构:

若条件 p 成立、执行 A，否则执行 B，脱离本结构。

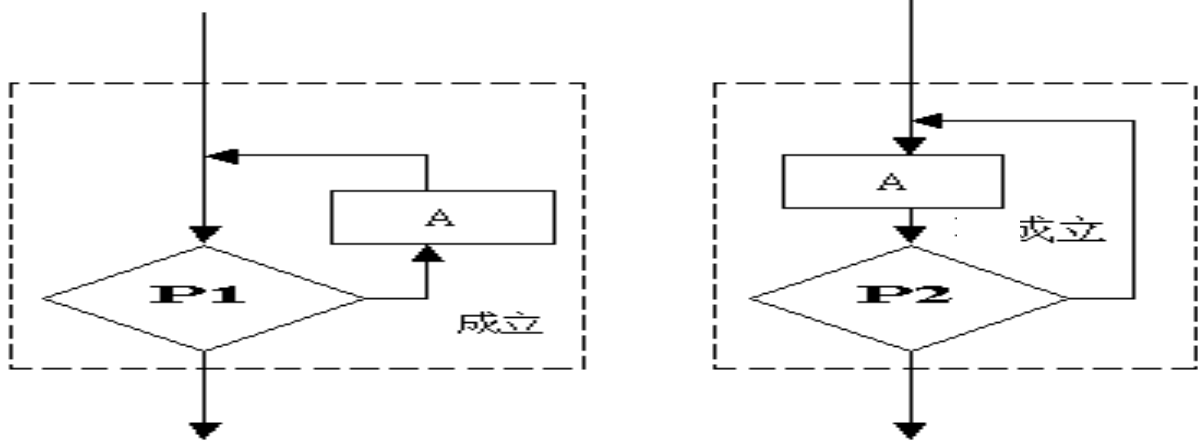
### 3、循环结构



1) 当型:重复判断条件 p 是否成立、执行 A，一旦条件 p 不成立，立即脱离本结构；至少执行 0 次

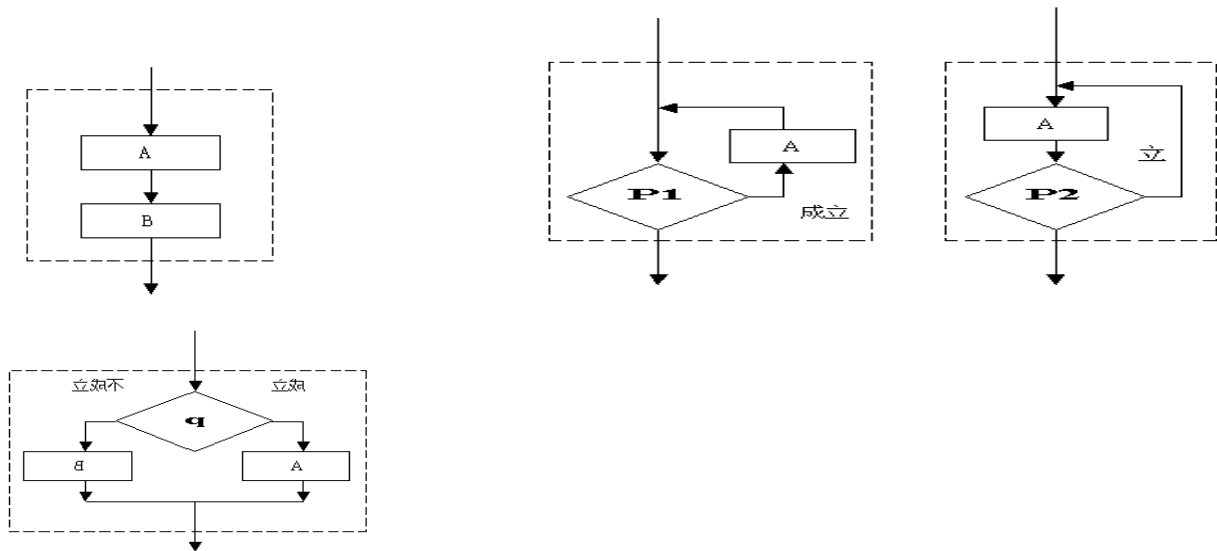
2) 直到型:重复执行 A，判断条件 p 是否成立，一旦条件 p 成立，立即脱离本结构。执行  $\geq 1$  次





三种基本结构的共同特点：

1. 只有一个入口；
2. 只有一个出口；
3. 结构内的每一部分都有机会被执行到；
4. 结构内不存在“死循环”。



### 【真题】

程序的三种基本控制结构是（B）

- A.过程，子程序和程序
- B.顺序，选择和循环
- C 递归，迭代和回溯
- D.调用.返回和转移

例：求  $1 \times 2 \times 3 \times 4 \times 5$  用 C 语言表示。

**void main()**

---

```
{ int i,t;
  t=1; i=2;
  while(i<=5)
    { t=t*i; i=i+1; }
  printf("%d",t);
}
```

用计算机语言表示算法

结构化程序设计方法

- ①自顶向下：
- ②逐步细化：
- ③模块化设计：
- ④结构化编码：

## 第 2 节 数据类型与运算符

### 本讲要点

- 字符集
- 数据类型
- 常量

### 授课内容

#### 2.1. C 语言的字符集

##### C 语言字符集:

- 1、字母：小写字母 a~z、大写字母 A~Z。
  - 2、数字：0~9 共 10 个。
  - 3、空白符：空格符、制表符、换行符等。
  - 4、标点和特殊字符。
-

---

5、字符常量：字符串常量和注释中还可以使用汉字或其它可表示的图形符号。

字符是组成语言的最基本的元素。

经典教材推荐：

《C 程序设计 》，谭浩强，清华大学出版社

## C 语言词汇（六类）

1、标识符：

变量名、函数名、标号等。

2、关键字：

关键字是由 C 语言规定的具有特定意义的字符串，通常也称为保留字。用户定义的标识符不应与关键字相同。 P365

（1）类型说明符：用于定义、说明变量、函数或其它数据结构的类型。

（2）语句定义符：用于表示一个语句的功能。

（3）预处理命令字：用于表示一个预处理命令。

标识符：由英文字母、数字和下划线这三种字符组成且第一个字符必须为字母或下划线。

作用：用来给变量、函数等命名

注意： 1.区分大小写英文字母

2.关键字（保留字）不能作标识符

例：

`book` 、 `Book` 、 `int1` 、 `Int` 、 `printf` 、 `_a12` 都可作标识符 。

`5a` 、 `$` 、 `p#`、 `int`、 `x+y` 都不可作标识符大写

3、运算符：

运算符与变量，函数一起组成表达式，表示各种运算功能。运算符由一个或多个字符组成。

4、分隔符：

在 C 语言中采用的分隔符有逗号和空格两种。

5、常量：

在后面章节中将专门给予介绍。

6、注释符：

C 语言的注释符是以 “`/*`” 开头并以 “`*/`” 结尾的串。在 “`/*`” 和 “`*/`” 之间的即为注释。程序编译时，不对注释作任何处理。注释可出现在程序中的任何位置。注释用来向用户提示或解释程序的意义。

## 2.2. C 的数据类型

C 语言提供了以下一些数据类型 。

---



## 2.3. 常量与变量

基本数据类型分为：

- 1、常量：在程序执行过程中，其值不发生改变的量。
- 2、变量：其值可变的量称为变量。

在程序中，常量是可以不经说明而直接引用的，而变量则必须先定义后使用。

直接常量(字面常量)：

整型常量：12、0、-3；

实型常量：4.6、-1.23；

字符常量：‘a’、‘b’。

字符串常量：“ABC”、“123”。

标识符：用来标识变量名、符号常量名、函数名、数组名、类型名、文件名的有效字符序列。

符号常量：用标示符代表一个常量。在C语言中，可以用一个标识符来表示一个常量，称之为符号常量。

### 2.3.1 常量

在使用之前必须先定义，

#define

---

---

**整型常量：**（程序执行过程中值不发生改变的量）

1. 十进制整数：123 、-76 、0
2. 八进制整数：076 、-0253  
076 转换成十进制： $7*8+6$
3. 十六制整数：0xa 、0x20 、0xf4  
0xf4 转换成十进制： $15*16+4$

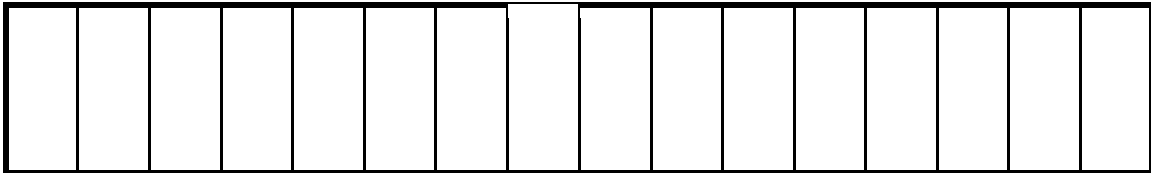
整型数据在内存中的存放形式：

如果定义了一个整型变量 i：

**int i;**

**i=10;**

十进制数 10 是以二进制形式表示的：1010，每一个整型变量在内存中占内 2 字节，存放示意图为：



事实上，数值是以补码表示的：

正数的补码和原码相同；

负数的补码：将该数的绝对值的二进制形式按位取反再加 1。

**实型常量（浮点型）：**也称为实数或者浮点数。

实数只采用十进制，有二种形式：

小数形式：由数码 0~9 和小数点组成。

小数形式必须有小数点。

指数形式

由十进制数、加阶码标志“e”或“E”、阶码（只能为整数，可以带符号）组成。其一般形式为：  
 $a \times 10^n$ （a 为十进制数，n 为十进制整数）其值为： $a*10^n$ 。

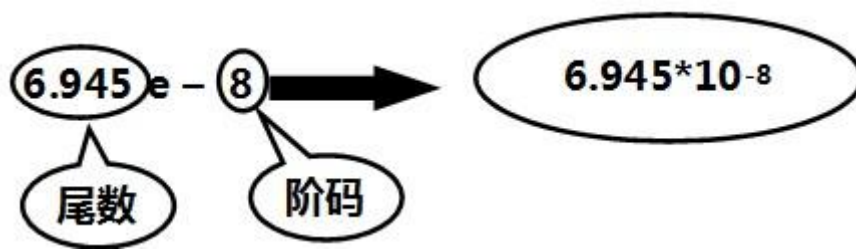
实型常量实例：

小数形式：12. 、-.71 、-0.71 、3.14（小数点不能省略）

指数形式：6.945 e -8

注意：指数形式(科学表示法)中，尾数和阶码都不能省略、阶码只能为整型常量、只有十进制形式

---



字符常量（一个字符）两种表示方式：

1. 在字符前后加单撇号：'y'、'3'、'\$'
2. 用转义字符表示（P48 表 3.3）：

'\n' → 换行	'\r' → 回车	'\\' → \
'\' ' → '	'\" ' 表示 "	'\12' → 换行
'\15' → 回车	'\134' → \	'\xa' → 换行
'\xd' → 回车	'\x5c' → \	

注：任一字符都可用 '\ddd' 或 '\xhh' 表示，其中 ddd 为该字符的 ASCII 码八进制形式、hh 为该字符的 ASCII 码十六进制形式

例：'A'、'\101'、'\x41' 均表示 A

整型常量与转义字符不能混淆

字符串常量：由两个双撇号括起来一串字符，其中字符个数  $\geq 0$  个

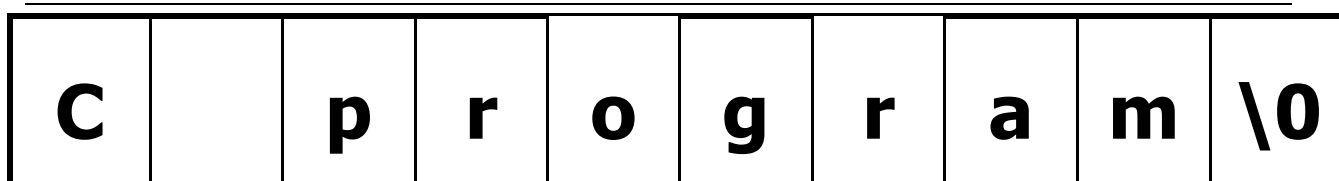
## 字符串常量和字符常量区别：

- 1) 字符常量由单引号括起来，字符串常量由双引号括起来。字符常量只能是单个字符，字符串常量则可以含一个或多个字符。
- 2) 可以把一个字符常量赋予一个字符变量，不能把一个字符串常量赋予一个字符变量。
- 3) 字符常量占一个字节的内存空间。字符串常量占的内存字节数等于字符串中字节数加 1。增加的一个字节中存放字符 "\0" (ASCII 码为 0)。这是字符串结束的标志。

例：

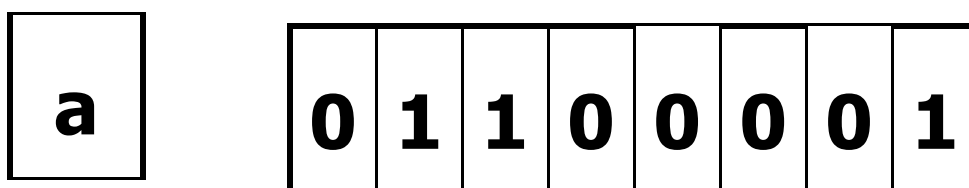
字符串 "C program" 在内存中所占的字节为：

---

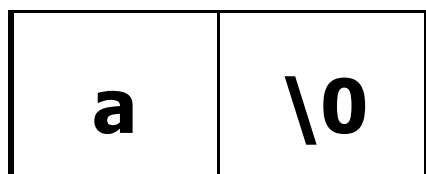


字符常量 ‘a’ 和字符串常量 “a” 在内存中的区别:

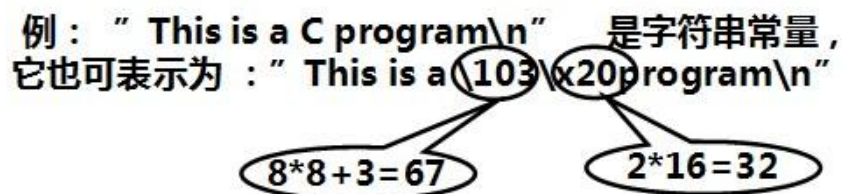
‘a’ 在内存中占一个字节，可表示为:



"a"在内存中占二个字节，可表示为:



字符串常量可用转义字符表示



1、计算字符串长度（其中字符个数）时要认真区分转义字符与一般字符

如：“\x18\17qw+\16X\1676” 中字符个数为?

2、字符串存储到内存中所占的字节数为：字符个数+1

\* 若把字符串存储到内存中，系统会自动在其末尾加一个 ‘\0’

\*转义字符的使用例子

‘\n’ ——换行    ‘\r’ ——回车、

‘\t’ ——水平跳到下一个 tab 位置

‘\b’ ——退格    ‘\f’ ——换页

例 1.

```
void main( )
```

```
{ printf("ab\t\t\tpractice\tbook\n");}
```

## 符号常量

用#define 命令宏定义一个符号常量（用标识符命名）代表某一常量

作用：减少程序中重复书写某些常量的工作量

例

```
#define PI 3.141592
#include<math.h>
void main( )
{printf ("%f,%f \n",sin(35*PI/180), cos(35*PI/180));}
```

注意：1.符号常量名中的英文字母习惯用大写

2.不能象给变量赋值那样给符号常量赋值

3.符号常量即以后要讲的宏名的用途。

## 调用函数库中的函数

a=3.14159 合法


3.14159=a 非法


例：

```
#define S " This is a C Program. "
void main( )
{printf (S); }
```

## 常量总结

一、整型常量：    
十进制整数：10  
八进制整数：012  
十六制整数：0xa

二、实型常量：    
小数形式：0.123456  
指数形式：1.23456e-1

三、字符常量：    
ASCII码十进制：' \n' ' A'  
ASCII码八进制：' \12' ' \101'  
ASCII码十六制：' \xa' ' \x41'

四、字符串常量：以 '\0' 结束。

五、符号常量

---



---

## 第 2 节 数据类型与运算符

### 本讲要点

- 变量
- 表达式
- 赋值表达式
- 算术表达式
- 运算顺序
- 顺序程序设计
- C 语句概述
- 5 类 C 语句
- 赋值语句
- 数据输入输出

### 授课内容

#### 2.3. 常量与变量

##### 2.3.2 变量

用标识符命名，习惯用小写

##### 1. 整型变量类型：

- (1) `[signed] int`
- (2) `[signed] short int`
- (3) `[signed] long int`
- (4) `unsigned [int]`
- (5) `unsigned short [int]`
- (6) `unsigned long [int]`

区别： ①每个变量在内存中所占字节数  
②所能存放的数值范围

---

以 13 为例：

00	00	00	00	00	00	11	01
----	----	----	----	----	----	----	----

00	00	00	00	00	00	11	01
----	----	----	----	----	----	----	----

[illegible]

00	00	00	00	00	00	11	01
----	----	----	----	----	----	----	----

00	00	00	00	00	00	11	01
----	----	----	----	----	----	----	----

[illegible]

---

整型数据的溢出

```
void main()
{int a,b;
  a=32767;
  b=a+1;
  printf("%d,%d\n",a,b);
}
```

运行结果：32767， -32768

32767: 

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

-32768: 

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

## 2. 字符型变量类型

\*字符型变量是用来存放单个字符常量的 ASCII 码，不是字符本身。整型变量与字符型变量可混淆使用。

例如：

**char a,b;**

（1）char

（2）unsigned char

区别：对变量值 ASCII 码的理解

char—— 一个数的补码

unsigned char——一个不带正负号的整数

### 字符变量的类型说明符

字符数据在内存中的存储形式及使用方法

每个字符变量分配一个字节的内存空间，因此只能存放一个字符。字符值以 ASCII 码的形式存放。

如：x 的十进制 ASCII 码是 120

y 的十进制 ASCII 码是 121

x:

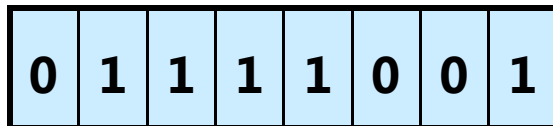
0	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---

120 的二进制代码

---

---

y:



121 的二进制代码

允许对整型变量赋以字符值，或对字符变量赋以整型值。允许把字符变量按整型量输出，或把整型量按字符量输出。

整型量为二字节量，字符量为单字节量，当整型量按字符型量处理时，只有低八位字节参与处理。

### 3. 实型变量类型

(1) float

(2) double

(3) long double

区别：

① 每个变量在内存中所占字节数

float—4、double—8

② 所能存放的数值范围

float  $10^{-38}$ — $10^{38}$

double  $10^{-308}$ — $10^{308}$

③ 有效数字位数 float<sup>-7</sup>、double<sup>-16</sup>

### 4. 变量定义与赋值

一、定义变量类型：

<类型名> <变量表> ；

若变量表中含有多个变量，它们之间  
结尾 。

用逗号隔开。最后一个变量名之后必须以 “;” 号

例： unsigned int i,j; 或 unsigned i,j;

一般所有变量都在声明部分定义类型

赋值：把 “=” 右侧的值赋给 “=” 左侧的变量

1、赋初值：定义变量类型时给变量赋值

---

---

在相应变量后写 `=<常量>`

例: `int a=123,b=456,sum;`

2、赋值表达式 : `<变量> = <表达式>`

赋值语句 : `<变量> = <表达式> ;`

例:

`a=123 sum=a+b`

`a=123 ; sum=a+b ;`

给变量赋值, 等价于

`int a,b,sum;`

`a=123;b=456;`

表达式末尾加 “;” 后即变成语句

①表达式本身有值的概念, 其值同 “=” 左边变量相同;

而语句本身无值的概念;

②表达式是用来构成语句的。

注: 表达式中变量可用给该变量赋值的赋值表

达式代替 (应加圆括号)

例: `a=123;b=456;`

`sum=a+b;`

可改为: `sum=(a=123)+(b=456);`

注意: 在定义中不允许连续赋值, 如: `int a=b=c=5` 是不合法的 ×

而 `int a, b, c; a=b=c=5;` 则是合法的 √

向字符变量赋以整数 (`//`为注释符号, 与`/* */`类似。在本讲义中两种注释并存。)

```
void main()
{ char a,b;          //定义 a, b 为字符型
  a=120;              //赋值语句中赋以整型值
  b=121;
  printf("%c,%c\n",a,b);    //格式符"c", 输出字符
  printf("%d,%d\n",a,b);    //格式符为"d", 输出整数
}
```

运行结果: x,y

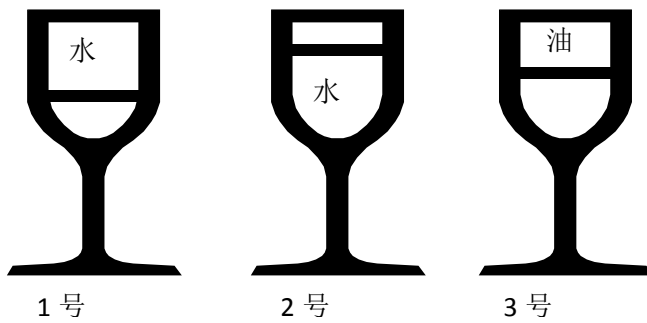
120,121

变量、变量名、变量类型、变量值、

---

---

## 给变量赋值之间的关系、



变量——杯

变量名——杯子的号码

变量类型——所装的材料类型

变量值——所装的东西多少

给变量赋值——把某材料倒进杯子

若类型不一致、以变量定义类型为准

如：

**int a;**

**a=123.5;**

则 a 为 123

变量的特性：

①一个变量同一时间只有一个值

②新的不来、旧的不去，新的一来、旧的就去

例: int a;

.....

a=3;        这段时间 a 值为 3

.....

a=7.8;      这段时间 a 值为 7

.....

两类常用赋值表达式的特别表示（用来提高可执行程序的质量）

（1） 自增、自减运算符

① <变量>=<变量> +1

可用 ++<变量> 代替

例: i=i+1、count=count+1

<变量>=<变量> - 1

---

---

可用 `--<变量>` 代替

例 `k=k-1、num=num-1`

② `<变量>++`与 `++<变量>` 的区别

`<变量>++` 的值为变量增一前的值

`++<变量>`的值为变量增一的值

同样, `<变量>--` 的值为变量减一前的值

`--<变量>`的值为变量减一的值

例:

```
void main( )
{ int i,j,k;
  i=3;
  j=++i;
  i=3;
  k=i++;
  printf("%d,%d\n",j,k);
  i=3;
  j=--i;
  i=3;
  k=i--;
  printf("%d,%d\n",j,k);}
```

对赋值过程的解释:

j 为 4

i 为 4

i 为 3

重新赋值

k 为 i 增 1 前的值

i 为 3

重新赋值

运行结果:

4, 3

2, 3

复合的赋值运算符

① `<变量>=<变量>+表达式`

`<变量>=<变量>-表达式`

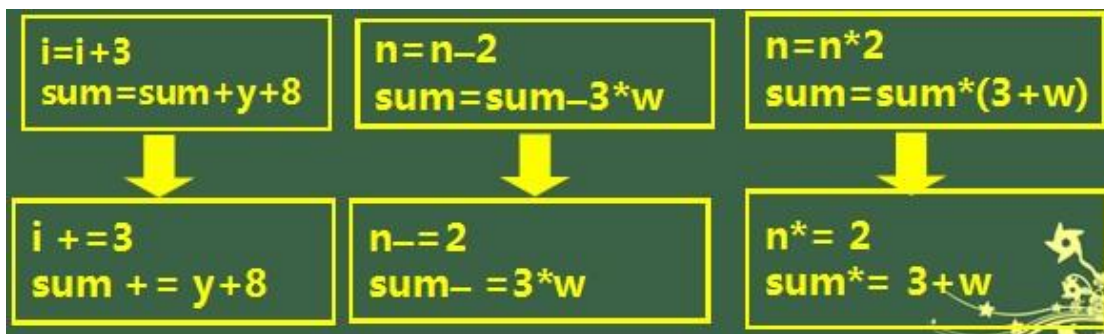
`<变量>=<变量>*表达式`

分别可用 `<变量>+=表达式`

`<变量>-=表达式`

`<变量>*=表达式` 代替 ...

---



### 10 种复合的赋值运算符:

`+=`、`-=`、`*=`、`/=`

`%=` 求余

`<<=` 左移

`>>=` 右移

`&=` 按位与

`^=` 按位异或

`|=` 按位或

## 2.4 C 语言的运算符

运算符把常量、符号常量、变量、函数等连接起来的有意义的式子:

如:

`a+=a*=(b=3)/(float)(int)(a=4.5).....`

### (1) 算术运算符:

用于各类数值运算:

加(+)、减(-)、乘(\*)、除(/)、求余(或称模运算, %)、自增(++)、自减(--)共七种。

### (2) 关系运算符:

用于比较运算:

大于(>)、小于(<)、等于(==)、大于等于(>=)、小于等于(<=)和不等不等于(!=)六种。

### (3) 逻辑运算符:

用于逻辑运算: 与(&&)、或(||)、非(!)三种。

### (4) 位操作运算符:

参与运算的量, 按二进制位进行运算:

位与(&)、位或(|)、位非(~)、位异或(^)、



---

左移(<<)、右移(>>)六种。

(5) 赋值运算符:

用于赋值运算,分为简单赋值(=)、复合算术

赋值(+, -, \*, /=, %=)和复合位运算赋值

(&=, |=, ^=, >>=, <<=)三类共十一种。

(6) 条件运算符:

这是一个三目运算符,用于条件求值(?:)。

(7) 逗号运算符:

用于把若干表达式组合成一个表达式(,)。

(8) 指针运算符:

用于取内容(\*)和取地址(&)二种运算。

(9) 求字节数运算符:

用于计算数据类型所占的字节数(sizeof)。

(10) 特殊运算符:

有括号(),下标[],成员(→,.)等几种。

【真题】

设 int a=7,b=9,t;

执行表达式 t=(a>b)?a:b 后, T 的值是多少? (B)

A.7 B.9 C.1 D.0

## C 语言程序设计

# 第 3 节 表达式与顺序程序设计

## 3.1 表达式

- 赋值表达式
- 算术表达式
- 运算顺序

### 一、赋值表达式:

---

### 1. 三种形式的赋值表达式

一般赋值表达式:  $t*=j/(2*j+1)$

自增自减表达式:  $k++$

复合赋值表达式:  $s=6*a+1$

### 2、赋值表达式举例:

**$a=b=c=5$**

表达式的值为 5, a,b,c 的值均为 5

**$a=5+(c=6)$**

表达式的值为 11, a 值为 11, c 的值为 6

**$a=(b=4)+(c=6)$**

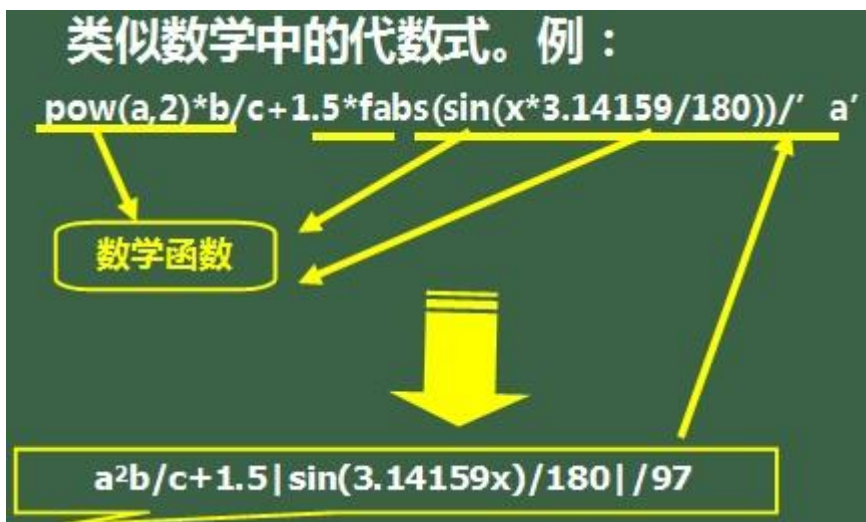
表达式的值为 10, a 为 10, b 为 4, c 为 6

**$a=(b=10)/(c=2)$**

表达式的值为 5, a 为 5, b 为 10, c 为 2

## 二、算术表达式

类似数学中的代数式。例:



abs(整型)

fabc(实型/整型)

### 1、算术运算符的有关说明:

两个整数相除, 如果有小数部分:

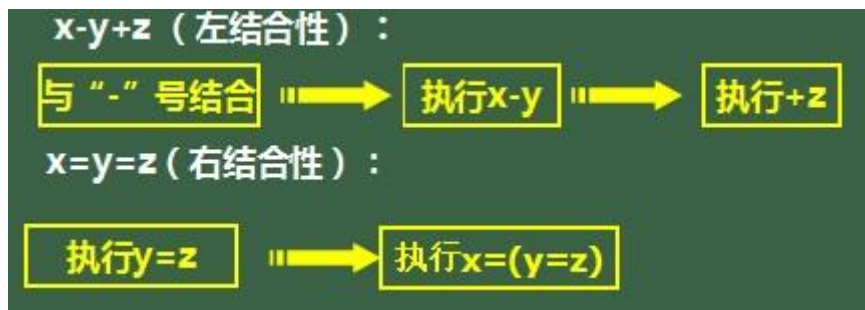
- 1、结果为整数: 舍去小数部分, 取整;
- 2、结果为负数: 舍入方向不固定, 采取

---

“向零取整”的方法，即取整后向“0”靠拢。如： $-5/3=-1$

运算符的优先级：优先级较高的先于优先级较低的。一个运算量两侧的运算符优先级相同时，则按运算符的结合性所规定的结合方向处理。

运算符的结合性：分为左结合性(自左至右)和右结合性(自右至左)。



## 2、算术表达式的有关说明：

### 1、对表达式： $(++i) + (++i) + (++i)$

Turbo C 将 3 作为表达式中所有 i 的值，因此 3 个 i 相加，得表达式的值为 9。编程时应避免这种歧义性，应写成：

```
i=3;    a=i++;    b=i++;    c=i++;  
d=a+b+c;
```

2、设 i 的初值为 3，语句 `printf(“%d,%d”,i,i++)`;有的系统具有左结合性，而另一些系统具有右结合性，结果分别为：3，3 和 4，3。编程时应避免这种歧义性，应写成：

```
j=i++;  
printf(“%d,%d”,j,i);
```

### 3. 含有强制类型转换运算符的表达式： 即表达式中含 (类型名)(表达式)

例. `(int) a`

整体的值为 int 类型，a 及 a 的值为原类型

例. `w+(float)(j%k)/5`

值为 float 类型，j、k 及它们的值仍为原来的类型

例：

```
void main( )  
{float x=3.6; int k;  
  k=(int)x/5;  
  printf(“x=%f,k=%d\n”,x,k);  
}
```

结果 x=3.600000,k=0

## 三、 表达式的运算顺序：

---

---

一般按从左至右、但赋值运算、自增、自减、负号运算、类型转换等从右至左。

优先级高的先算

“左结合性”：自左至右的结合方向。算术运算符的结合性是自左至右，即先左后右。

如：表达式  $x-y+z$

(1)  $y$  应先与“-”号结合，执行  $x-y$  运算；

(2) 再执行  $+z$  的运算。

“右结合性”：自右至左的结合方向。最典型的右结合性运算符是赋值运算符。

如：表达式  $x=y=z$ ，由于“=”的右结合性，(1) 先执行  $y=z$  运算；(2) 再执行  $x=(y=z)$  运算。

C 语言运算符中有不少为右结合性，应注意区别，以避免理解错误。

## 3.2 顺序程序设计

### 1. C 语句概述

C 程序的结构：



C 语句可分为以下五类：

- 1) 表达式语句
- 2) 函数调用语句
- 3) 控制语句
- 4) 复合语句
- 5) 空语句

C 程序的执行部分是由语句组成的。程序的功能也是由执行语句实现的。

### 2. 5 类 C 语句

---

---

## 表达式语句

表达式语句由表达式加上分号“;”组成。执行表达式语句就是计算表达式的值。

其一般形式为：        表达式;

例如:

```
x=y+z;
```

```
y+z;
```

```
i++;
```

赋值语句

加法运算语句，但计算结果不能保留，无实际意义

自增 1 语句，i 值增 1

## 函数调用语句

函数调用语句由函数名、实际参数加上分号“;”组成。执行函数语句就是调用函数体 (在后面函数中再详细介绍) 。

其一般形式为：        函数名(实际参数表);

例如:

```
printf("C Program");
```

调用库函数，输出字符串。

## 控制语句

控制语句用于控制程序的流程，以实现程序的各种结构方式。可分成以下三类：

- 1) 条件判断语句：if else 语句、switch 语句;
- 2) 循环执行语句：do while 语句、while 语句、for 语句;
- 3) 转向语句：break 语句、goto 语句、continue 语句、return 语句。

## 复合语句

复合语句把多个语句用括号{}括起来组成的一个语句称复合语句。在程序中应把复合语句看成是单条语句，而不是多条语句。

例如：以下是一条复合语句。

```
{ x=y+z;
  a=b+c;
  printf( "%d%d", x, a);
}
```

复合语句内的各条语句都必须以分号“;”结尾，在括号“{”外不能加分号。

## 空语句

空语句只有分号“;”组成的语句称为空语句。空语句是什么也不执行的语句。

例如

```
while(getchar()!='\n')
```

---

---

;

本语句的功能是，只要从键盘输入的字符不是回车则重新输入。

### 3. 赋值语句

赋值语句是由赋值表达式再加上分号构成的表达式语句。其一般形式为：

变量=表达式;

赋值语句的功能和特点与赋值表达式相同。是程序中使用最多的语句之一。

在赋值语句的使用中需要注意以下几点：

#### 1、赋值表达式的嵌套：

赋值符右边的表达式又是一个赋值表达式 变量=(变量=表达式);

展开之后的一般形式为：

变量=变量=...=表达式;

例如：

**a=b=c=d=e=5;**

按照赋值运算符的右接合性，等效于：

**e=5;     d=e;     c=d;     b=c;     a=b;**

#### 2、变量说明中给变量赋初值和赋值语句的区别：

给变量赋初值是变量说明的一部分，赋初值后的变量与其后的其它同类变量之间仍必须用逗号间隔，而赋值语句则必须用分号结尾。

例如：

**int a=5,b,c;**

#### 3、在变量说明中，不允许连续给多个变量赋初值：

如下述说明是错误的：

**int a=b=c=5; ×**

而赋值语句允许连续赋值：

**a=b=c=5; ✓**

#### 4、赋值表达式和赋值语句的区别：

赋值表达式是一种表达式，它可以出现在任何允许表达式出现的地方，而赋值语句则不能。

**if((x=y+5)>0) z=x;**

语句的功能是，若表达式  $x=y+5$  大于 0 则  $z=x$ 。合法

**if((x=y+5;)>0) z=x;**

因为  $x=y+5$  是语句，不能出现在表达式中。非法。

---

---

## 4. 数据输入输出

- 1) 输入输出是从标准输入设备输入或向标准输出设备输出数据。
- 2) 在 C 语言中，所有的数据输入 / 输出都是由库函数完成的，都是函数语句。
- 3) 使用 C 语言库函数时，要用预编译命令：

`#include <stdio.h>` 或

`#include "stdio.h"`

- 4) 考虑到 `printf` 和 `scanf` 函数使用频繁，系统允许在使用这两个函数时可不加以上预编译命令

### `printf` 函数（格式输出函数）

`printf(.....)` 功能：按格式控制所指定的形式（向用户屏）输出输出项的值。

C 不提供输入输出语句，只调用 `printf(.....)` 函数

例：

```
void main( )
{int a,b,sum;
  a=123;b=456;
  sum=a+b;
  printf("a=%d,b=%d,c=%d\n",a,b,sum);
}
```

结果: a=123,b=456,c=579

格式控制符与输出项表列一一对应

格式： `printf( 格式控制, 输出项表列)`

一、输出项表列：

由一个或多个输出项组成(若多个、它们之间用逗号隔开)，常量、变量、函数、表达式、字符数组名等均可做输出项。

二、格式控制：

字符串常量或字符数组名。其中字符可由下面 3 类组成：

普通字符（原样输出）

转义字符（输出所表示的字符或实现其功能）

格式说明：控制对应输出项的输出形式

---

```
#include<math.h>
```

```
void main()
```

```
{int k=68;float t=56.3;
```

```
printf(" %d\t%d\t%f\n",k,k+123,sqrt(t));
```

```
}
```

转义字符,  
回车

结果: 68

191

7.503333

格式控制符,对应输出  
格式

转义字符,  
跳到下一  
区

格式说明:

控制对应输出项的输出形式 (每个格式说明按先后顺序与输出项一一对应), 输出结果中格式说明被对应的输出项值取代。

(1) 格式说明基本形式:

带符号  
十进制

带符号  
八进制

带符号  
十六进制

无符号  
十进制

%d (或%i)、%o、%x (或%X)、%u、  
%c 对应的输出项类型为字符型和整型。

字  
符



例:

```
void main()
```

```
{ int a=65 ; char c=' !' ;
```

```
printf (" %d,%c\n" ,a,a);
```

```
printf(" c=%c,ItsASCIIcode=%d\n" ,c,c);
```

```
}
```

结果: 65,A

c=!, Its ASCII code=33



## putchar 函数（字符输出函数）

格式: putchar ( 一个整形或字符型输出项 )

功能: 同 printf ( " %c" , 输出项 ),在显示器上  
输出单个字符。

例如: putchar('A'); (输出大写字母 A)

putchar(x); (输出字符变量 x 的值)

putchar( '\101' ); (也是输出字符 A)

putchar("\n"); (换行)

对控制字符则执行控制功能，不在屏幕上显示。使用本函数前必须要用文件包含命令：

例:

```
#include<stdio.h>
```

```
void main()
```

```
{ char c=' A' ;
```

```
putchar(66);
```

```
putchar(' \53' );
```

```
putchar(c+2);
```

```
putchar(' \n' );
```

```
putchar(' 7' );
```

```
putchar(' \n' );
```

```
}
```

结果: B+C

7



---

## scanf 函数（格式输入函数）

格式 scanf (格式控制，输入项地址表列)

地址：

- 1) 变量名前加&——变量在内存中的地址；
- 2) 数组名——数组在内存中的首地址。

例如：&a, &b

分别表示变量 a 和变量 b 的地址。这个地址就是编译系统在内存中给 a,b 变量分配的地址。

一个或多个地址组成(若多个之间用逗号隔开)

变量的地址和变量值的关系如下：

- 1、在赋值表达式中给变量赋值：

如：a=567

- 2、scanf 函数在本质上也是给变量赋值，但要求写变量的地址：

如：scanf(“%d”,&a);

- 3、这两者在形式上是不同的。&是一个取地址运算符，&a 是一个表达式，其功能是求变量的地址。

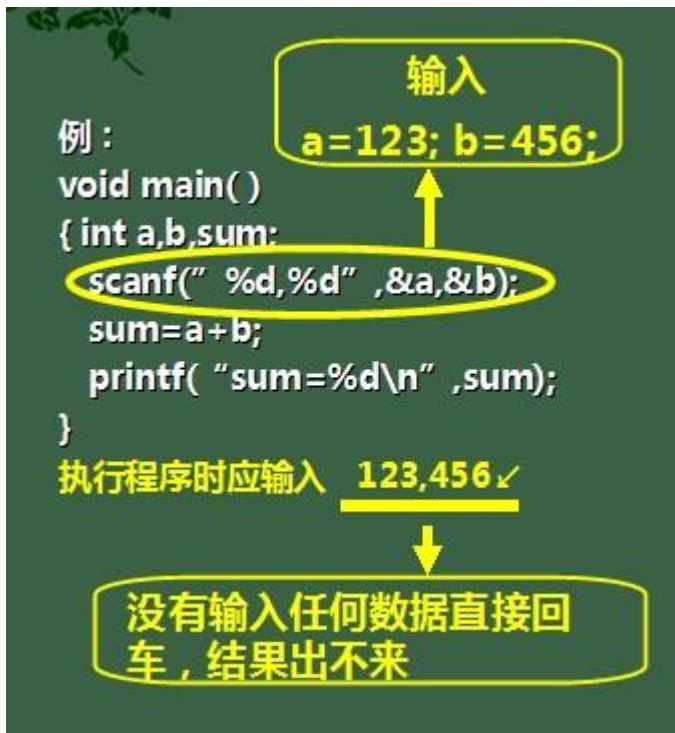
功能：系统切换到用户屏、要求并等待用户按格式控制形式和顺序输入要赋给输入项的常量，按回车键结束，系统立即把所输入的赋给相应输入项。

它是一个标准库函数，与 printf 函数相同，使用 scanf 函数之前不必包含 stdio.h 文件。

注意：格式控制字符串的作用与 printf 函数相同，但不能显示非格式字符串，也就是不能显示提示字符串。地址表列中给出各变量的地址。地址是由地址运算符“&”后跟变量名组成的。

不能是符号常量，不加单撇号或双撇号，直接输入

---



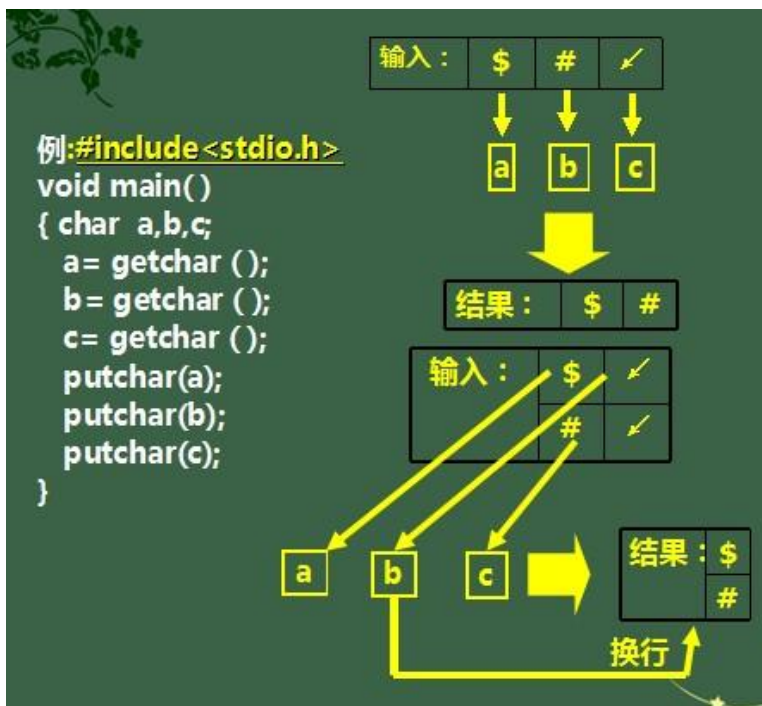
## getchar 函数（字符输入函数）

形式: getchar ( )

功能: c=getchar ( );同 scanf ( " %c" , &c);

注意: 需在文件开头加 #include " stdio.h"

输入一个字符，赋给变量 c



使用 getchar 函数还应注意几个问题:

- 
- 1) `getchar` 函数只能接受单个字符，输入数字也按字符处理。输入多于一个字符时，只接收第一个字符。
  - 2) 使用本函数前必须包含文件 “`stdio.h`”。
  - 3) 在 TC 屏幕下运行含本函数程序时，将退出 TC 屏幕进入用户屏幕等待用户输入。输入完毕再返回 TC 屏幕。

【真题】

以下程序输出的结果是（A）

```
Main()
{int a=11;
Printf("%d\n",a++);}
```

A.12    B.11    C.10    D.9

## 第 4 节 分支（选择）与循环程序设计

### 本讲要点

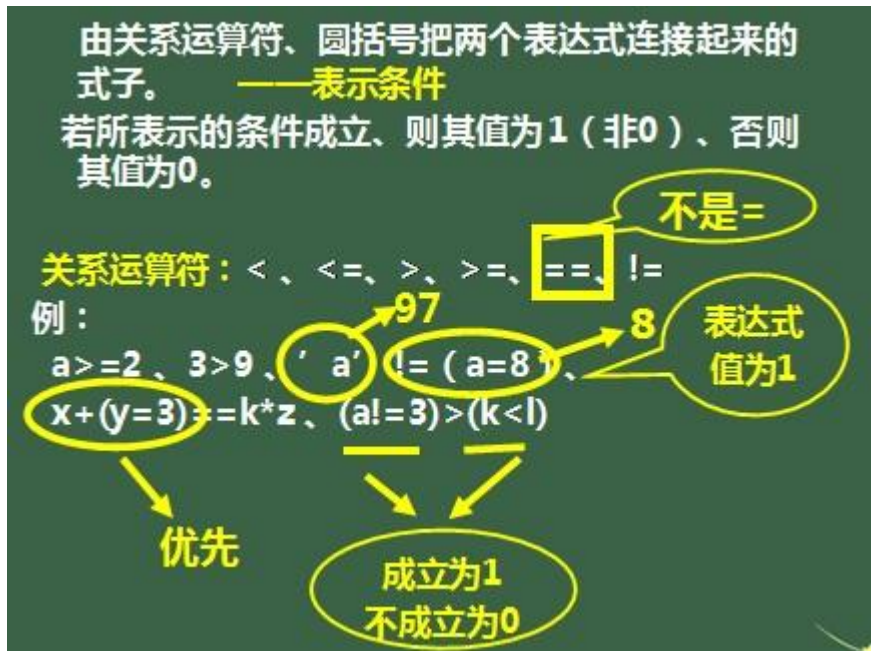
- 分支（选择）程序设计
- 关系运算符
- 逻辑运算符
- 选择结构（if）
- 循环程序设计
- Goto 语句
- While 语句和 do-while 语句
- For 语句
- 循环的嵌套
- Break 和 continue 语句

### 授课内容

---

## 4.1 分支（选择）程序设计

### 1. 关系表达式



注意：数学中多个条件，例  $a > b > c > d$

应表示为  $a > b \ \&\& \ b > c \ \&\& \ c > d$

不能顺手也写为  $a > b > c > d$

关系运算符都是双目运算符，其结合性均为左结合。关系运算符的优先级低于算术运算符，高于赋值运算符。

在六个关系运算符中，<,<=,>,>=的优先级相同，高于==和!=，==和!=的优先级相同。

### 2. 逻辑表达式

由逻辑运算符、圆括号把表达式连接起来的式子。

若所表示的条件成立其值为1(非0)，否则其值为0

逻辑运算符：&&（与）、||（或）、！（非）：

&&和||均为双目运算符，具有左结合性。!为单目运算符，具有右结合性。

逻辑运算符和其它运算符优先级的关系可表示如下：

！（非）→&&（与）→||（或）

“&&”和“||”低于关系运算符，“！”高于算术运算符。

按照运算符的优先顺序可以得出：

$a > b \ \&\& \ c > d$  等价于

$(a > b) \ \&\& \ (c > d)$

$!b == c \ || \ d < a$  等价于

$((!b) == c) \ || \ (d < a)$

---

$a+b>c \&\& x+y<b$  等价于  
 $((a+b)>c) \&\& ((x+y)<b)$

例：设  $x=0, y=2, a=8, k=-1, q=1, z=1$ ，则

1、 $x+y>3 \&\& a==8$

$\rightarrow 2>3 \&\& 1 \rightarrow 0 \&\& 1 \rightarrow 0$

2、 $x+(y=3)==k*z \mid\mid !a$

$\rightarrow 0+3==-1 \mid\mid !a \rightarrow 0 \mid\mid 0 \rightarrow 0$

3、 $!(k*q) \rightarrow !(-1) \rightarrow 0$

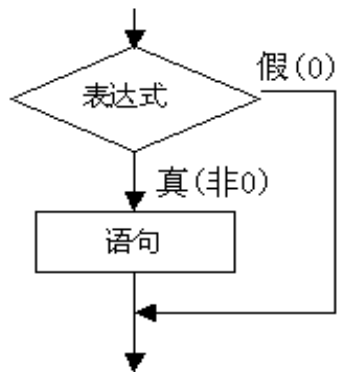
### 3.选择结构(if 语句)

if 语句的三种形式

1、第一种形式为基本形式: if

**if(表达式) 语句**

语义：如果表达式的值为真，则执行其后的语句，否则不执行该语句。其过程可表示为下图。



表达式可以是任意合法表达式，后面不能加；

例：

```
void main()
```

```
{int a,b,max;
```

```
    printf("\n input two numbers:  "); //此语句用于提示输入
```

```
    scanf("%d %d",&a,&b);
```

```
    max=a;
```

```
    if (max<b) max=b;
```

```
    printf("max=%d",max);
```

```
}
```

输入：23 46

输出：max=46

max=23

max=46

---

---

## 2、第二种形式为: if-else

**if(表达式)**

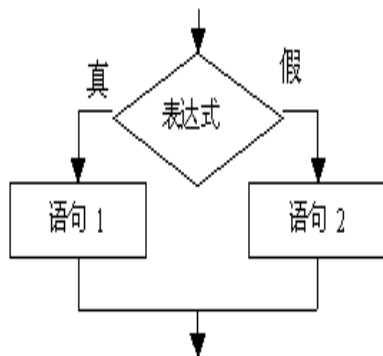
**语句 1;**

**else**

**语句 2;**

语义：如果表达式的值为真，则执行语句 1，否则执行语句 2 。

后面不能加；



### 前例的改写

输入两个整数，输出其中大数

```
void main()
{
    int a, b;
    printf("input two numbers:    ");
    scanf("%d %d",&a,&b);
    if(a>b)
        printf("max=%d\n",a);
    else
        printf("max=%d\n",b);
}
```

输入：23 46

输出：max=46

## 3、第三种形式为 if-else-if 形式:

**if(表达式 1)**

**语句 1;**

**else if(表达式 2)**

**语句 2;**

**else if(表达式 3)**

**语句 3;**

    ...

**else if(表达式 m)**

**语句 m;**

**else**

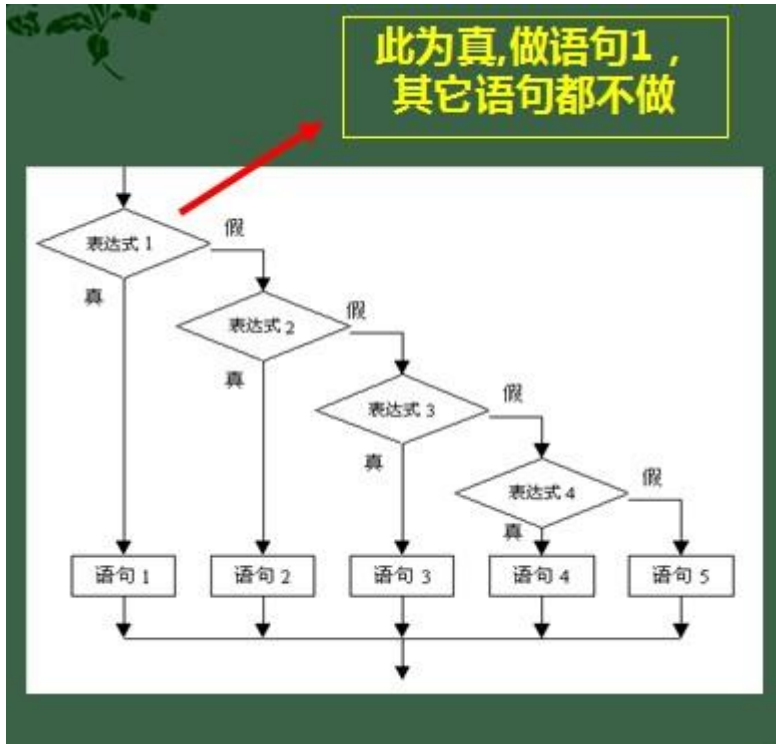
---

语句 n;

语义:

依次判断表达式的值，当出现某个值为真时，则执行其对应的语句。然后跳到整个 if 语句之外继续执行程序。如果所有的表达式均为假，则执行语句 n。然后继续执行后续程序。

后面不能加;



上面的语句为以下三种格式:

- 1、空语句
- 2、一个语句（不引起转移）
- 3、前后加大括号的语句（复合语句），也叫程序段，如:

```
{int a;  
float b;  
a=3;  
b=7.0;}
```

允许在复合语句前部定义变量，只能在复合语句中使用

不能写成

```
{int a;  
a=3;  
float b;  
b=7.0;}
```

变量要在最前面定义

例



---

```
#include "stdio.h"
void main(){
    char c;
    c=getchar();
    if(c<32)
        printf("This is a control character\n");
    else if(c>='0'&&c<='9')
        printf("This is a digit\n");
    else if(c>='A'&&c<='Z')
        printf("This is a capital letter\n");
    else if(c>='a'&&c<='z')
        printf("This is a small letter\n");
    else
        printf("This is an other character\n");
}
```

输入：A

输出：This is a capital letter

#### 4.条件运算符和条件表达式

在条件语句中，只执行单个的赋值语句时，可使用条件表达式来实现。

条件运算符为?和:，它是一个三目运算符，即有三个参与运算的量。

条件表达式的一般形式为：

表达式 1? 表达式 2：表达式 3

求值规则为：如果表达式 1 的值为真，则以表达式 2 的值作为条件表达式的值，否则以表达式 3 的值作为整个条件表达式的值。

例如条件语句：

if(a>b) max=a; else max=b;

可用条件表达式写为：

max=(a>b)?a:b;

执行该语句的语义是：

如 a>b 为真，则把 a 赋予 max，

如 a>b 为假，则把 b 赋予 max。

#### 【真题】

设 int a=7,b=9,t;

执行表达式 t=(a>b)?a:b 后，T 的值是多少？（B）

A.7    B.9    C.1    D.0

---

---

## 4.2 循环程序设计

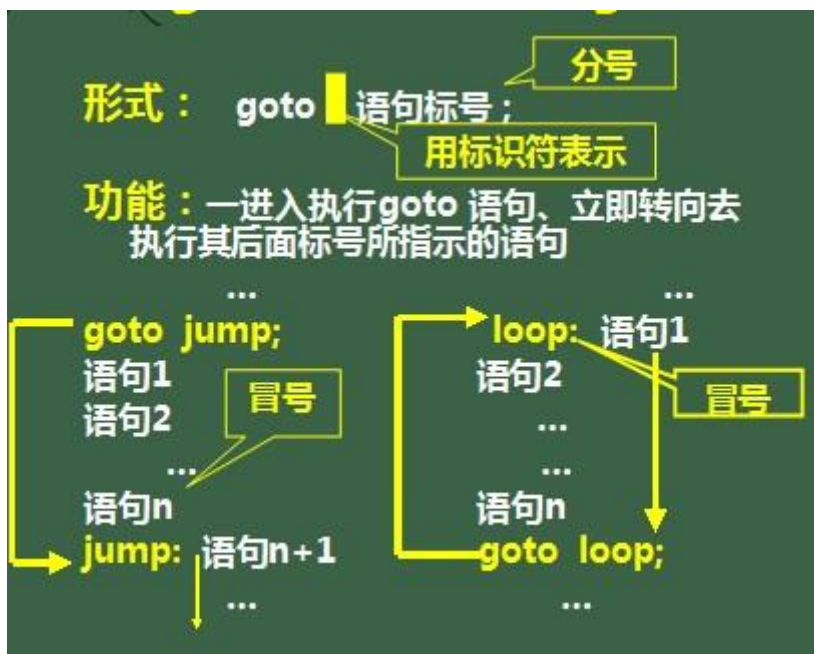
### 1.循环控制概述

循环结构的特点：在给定条件成立时，反复执行某程序段，直到条件不成立为止。

C 语言提供的循环结构类型如下：

- 1) 用 goto 语句和 if 语句构成循环；
- 2) 用 while 语句；
- 3) 用 do-while 语句；
- 4) 用 for 语句；

### 2.goto 语句以及用 goto 语句构成循环



注意：结构化程序设计方法主张限制使用 goto 语句，一般只在下面两种结构中表示循环

一、loop: if (表达式)

```
{ 语句  
    goto loop;  
}
```

表达式值为零时，立即停止循环

二、loop: 语句

if (! 表达式) goto loop;

表达式值非零时，立即停止循环

例:用 goto 语句和 if 语句构成循环。

例 用 goto 语句和 if 语句构成循环。

---

---

```

void main()
{
    int i,sum=0;
    i=1;
    loop:  if(i<=100)
            {sum=sum+i;
              i++;
              goto loop;}
    printf("%d\n",sum);
}

```

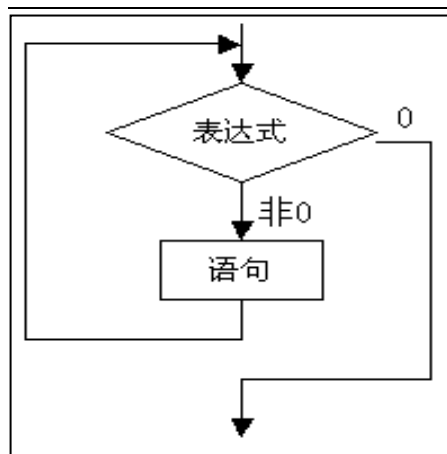
运行结果：5050

i=101

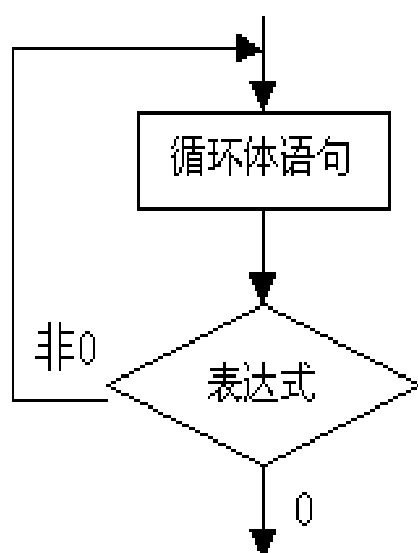


**while (表达式)**

---



**do** 语句 **while** (表达式);



## 例. 写出下面程序的运行结果

(1) void main()

{ int i=1;

while(i<=3)

{ printf(" %d\n" ,i);i++ ;}

}

复合语句  
作为循环  
体语句

结果 1  
2  
3

(2) void main()

{ int i=1;

while(i<=3);

{ printf(" %d\n" ,i);i++ ;}

}

结果 死循环

循环体

空语句

注意：若例2中while一行无分号，去掉while下面一行的{、}也是死循环。因为执行不到++，i不会变化。

复合语句作为循环体语句

(3) void main ()

{ int k=1 , n=1 , sum=0 ;

do { sum+=n ; n++ ;

if ( sum>=10 ) k=0;

} while ( k==1);

printf ( " sum=%d\n" ,sum) ;

}

改为while(k=1)  
死循环

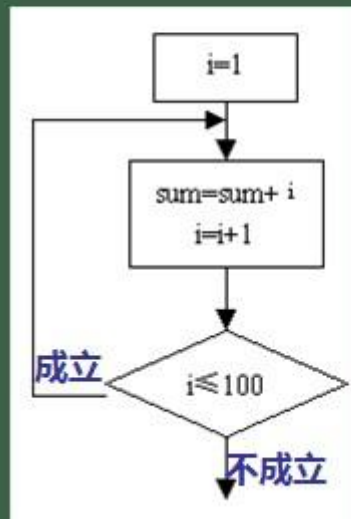
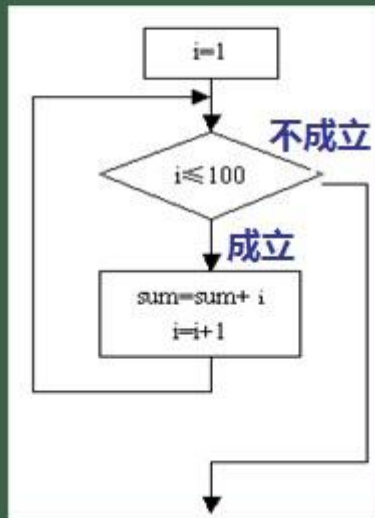
sum	n
0	1
1	2
3	3
6	4
10	

结果:  
sum=10

## 例 用while语句和do-while语句求

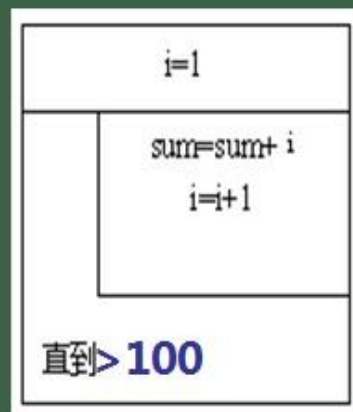
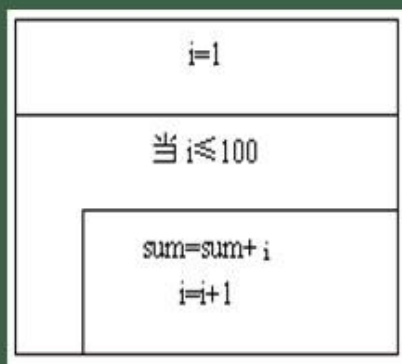
### 传统流程图表示算法

$$\sum_{n=1}^{100} n$$



## N-S图表示算法

$$\sum_{n=1}^{100} n$$



## N-S图：无线流程图，又称盒图

while 语句中的表达式一般是关系表达或逻辑表达式，也可以是其它表达式，只要表达式的值为真(非 0)即可继续循环。

例

```
void main()
```

---

```
{ int a=0,n;
  printf("\n input n:   ");
  scanf("%d",&n);
  while (n--      //执行 n 次，每执行 1 次，n 减 1。
  printf("%d  ",a++*2);} //等价于(a*2; a++)
```

输入: 8

输出: 0 2 4 6 8 10 12 14

4.for 语句——代替 while 语句

形式:

for (表达式 1;表达式 2;表达式 3) 语句要正确表达循环结构应注意三方面要求:

- 1、循环控制变量的初始化。
- 2、循环的条件。
- 3、循环控制变量的值的更新

for 语句在书写形式上集中体现了这三方面要求

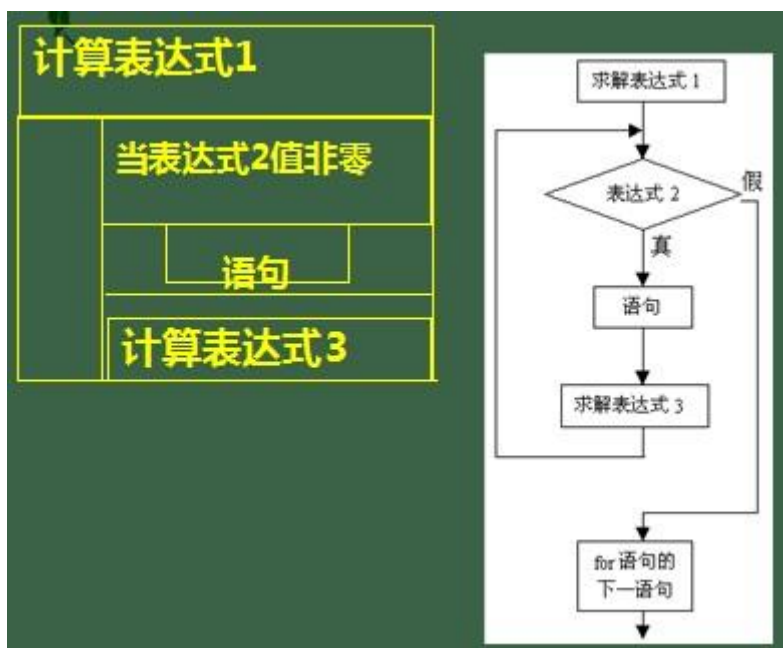
功能

for 语句完全可以取代 while 语句。

for(表达式 1; 表达式 2 ; 表达式 3) 语句

它的执行过程如下:

- 1) 先求解表达式 1。
- 2) 求解表达式 2, 若其值为真 (非 0), 则执行 for 语句中指定的内嵌语句, 然后执行第 3) 步; 若其值为假 (0), 则结束循环, 转到第 5) 步。
- 3) 求解表达式 3。
- 4) 转回第 2) 步继续执行。
- 5) 循环结束, 执行 for 语句下面的一个语句。



---

例、求  $1+2+\dots+100$

```
void main()
{ int i,sum=0;
  i=1;
  do {sum+=i; i++; }
  while(i<=100);
  printf("%d\n",sum); }
```

或

```
void main()
{ int i,sum=0;
  for (i=1;i<=100;i++) sum+=i;
  printf("%d\n",sum); }
```

对于 for 循环中语句的一般形式,

**for(表达式 1; 表达式 2 ; 表达式 3) 语句**  
就是如下的 while 循环形式:

```
表达式 1;
while (表达式 2)
{语句
  表达式 3; }
```

例.下面程序的输出结果是[     ]

```
void main ( )
{ int  x=10, y=10, i;
  for ( i=0; x>8; y= i+1 )        //表达式 1 只做一次
    printf ( " %3d%3d" , x--,y) ;
}
```

- A.  10  1  9  2
- B.   9  8  7  6
- C.  10 10  9  1
- D.  10  9  9  0

解析：输出如下表

x	y
10	10
9	1
8	2

注意:

---



- 
- 1) for 循环中的“表达式 1”、“表达式 2(循环条件)”和“表达式 3”可以缺省,但“;”不能缺省。
  - 2) 省略了“表达式 1”,表示不对循环控制变量赋初值,可在 for 循环里面或外面对其赋初值。

例如: `i=1; for (;j=20;i++)`

`{scanf ("%f",&x); sum+=x;}`

- 3) 省略“表达式 2”,则不做其它处理时成为死循环。

例如: `for(i=1;;i++)sum=sum+i;`

相当于: `i=1;`

`while(1)`

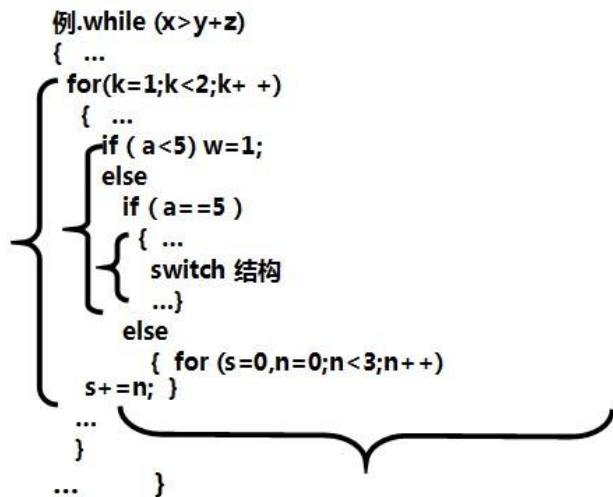
`{sum=sum+i;`

`i++;}`

### ● 循环的嵌套

- 1、各种循环结构的循环体中可以包含任一种完整的循环结构、选择结构。
- 2、选择结构的任一分支中也可以包含任一种完整的循环结构、选择结构。
- 3、且它们可多层嵌套。

例.while (x>y+z)  
{ ...  
  for(k=1;k<2;k+ +)  
  { ...  
    if ( a<5) w=1;  
    else  
      if ( a==5 )  
      { ...  
        switch 结构  
        ...}  
    else  
      { for (s=0,n=0;n<3;n+ +)  
        s+=n; }  
  ...  
}



### 6.break 语句和 continue 语句

- switch 结构中的 break: 提前结束该结构
- 循环结构中的 break: 提前结束该结构
- (while 循环、do-while 循环、for 循环)
- 循环结构中的 continue: 提前结束本次循环

若多层嵌套中使用 break 或 continue,它只影响包含它的最内层结构

例. 写出以下程序的运行结果:

`void main( )`

`{ int n;`

---

---

```

for (n=1;n<=10;n+ +)
    {if(n%3==0)break;printf("%4d",n);}
printf("!!!!\n");

```

结果: 1    2!!!!

```

void main( )
{ int  n;
  for (n=1;n<=10;n+ +)
      {if(n%3==0) cotinue;printf("%4d",n);}
  printf("!!!!\n");}

```

结果: 1    2    4    5    7    8    10!!!!

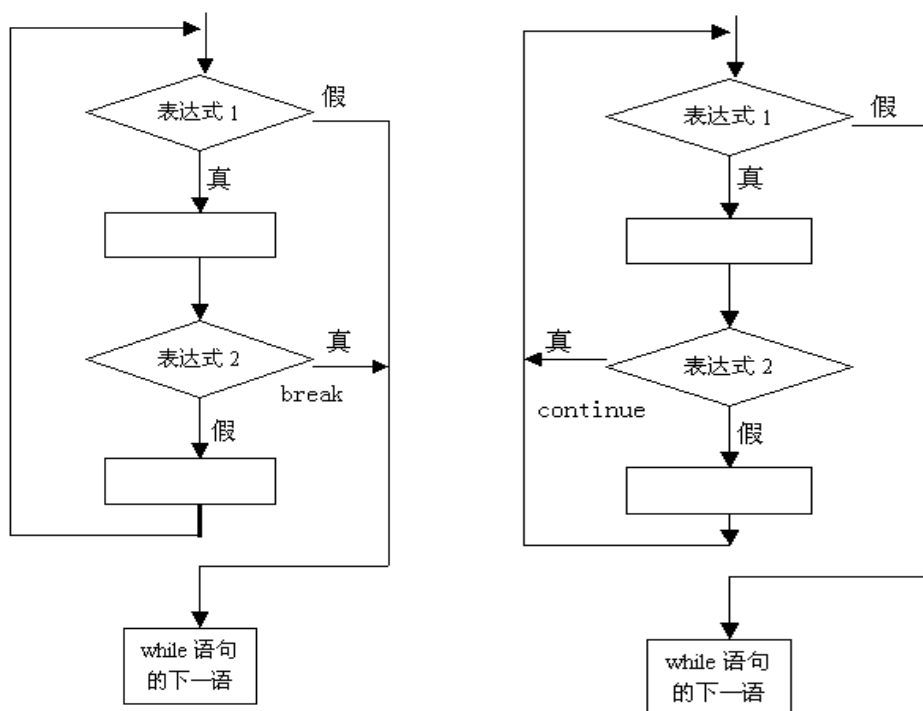


图 break 和 continue 的比较

---

---

## 第5节 函数

### 本讲要点

- 函数的分类
- 函数的定义
- 函数的调用
- 形式参数与实际参数
- 局部变量和全局变量

### 授课内容

一个C程序由一个主函数和 $\geq 0$ 个其它函数组成函数体允许是空的或无值类型,或无返回值(void)  
结构化程序设计的一个重要特点: 模块化设计, 即用一个函数表示一个模块

#### 5.1 函数分类

一、从函数定义的角度分为:

- 1、库函数:
- 2、用户自己定义的函数: 由用户按需要写的函数。对于用户自定义函数, 不仅要在程序中定义函数本身, 而且在主调函数模块中还必须对该被调函数进行类型说明, 然后才能使用。

二、从函数的形式分为:

- 1、有参函数: 在函数定义、说明时都有参数, 称为形式参数/形参。在函数调用时也必须给出参数, 称为实际参数/实参。
- 2、无参函数: 函数定义、函数说明及函数调用中均不带参数。

#### 5.2 函数的定义

形式1: [类型] 函数名()  
{ ..... } int m()

形式2: [类型] 函数名( 形参表列 )  
定义形参类型  
{ ..... } int f(m,n)

---

---

```
int m;char n;
```

形式 3:

```
[类型] 函数名(类型 1 形参 1, 类型 2 形参 2, ..... )  
{.....}int f ( int m ,char n)
```

函数名前“类型”为 int 时可省略

函数体允许是空的

可以为无值类型,无返回值(void)

例如, 定义一个函数, 用于求两个数中的大数:

```
int max(int a, int b)  
{ if (a>b) return a;  
  else return b;  
}
```

1) 函数的返回值:

函数的返回值语句的一般形式为:

```
return 表达式;    或    return (表达式);
```

第一行说明 max 函数是一个整型函数, 其返回的函数值是一个整数。形参为 a,b,均为整型量。a,b 的具体值是由主调函数在调用时传送过来的。

在 max 函数体中的 return 语句是把 a(或 b)的值作为函数的值返回给主调函数。有返回值函数中至少应有一个 return 语句。

2) 函数的类型:

因为函数有返回值, 返回值就必须指定类型。

在定义函数时指定函数值类型。

如上例的定义函数:

```
int max(int a, int b)  
{ if (a>b) return a;  
  else return b;    }
```

再如: char letter(charc1,charc2)

```
double min(int x,inty)
```

3) 如函数值为整型, 在函数定义时可以省去类型说明。

4) 在定义函数时指定的类型说明, 应与 return 语句表达式的类型一致, 如果不一致, 则以函数类型为主, 自动进行类型转换。函数类型决定返回值的类型。

5) 不返回函数值的函数, 可以明确定义为“空类型”, 如:

```
void s(int n)  
{ ..... }
```

在主调函数中不能使用空类型的被调函数的函数值。

---

### 5.3 函数的调用（出现在另一函数的执行部分）

格式： 函数名（实参列表）；

一、函数语句：

```
printstar();
```

二、函数表达式：

```
f(n)/(f(r)*f(n-r))
```

\*即出现在表达式中

三、函数参数：

```
printf("%d\n",f(n))
```

\*调用另一函数时作实参

注意：void 类型函数只能使用形式一

使用函数

返回值

例：

```
int max(int a,int b)
{ if(a>b) return a;
  else return b;
}
main()
{ int max(int a,int b);
  int x,y,z;
  printf("input two numbers:\n");
  scanf("%d%d",&x,&y);
  z=max(x,y);
  printf("maximum=%d",z);
}
```

一个函数的定义可以放在主函数main之前，也可放在main之后，**末尾无分号**

对max函数进行说明，**末尾要加分号**

调用max函数，并把x,y中的值传送给max的形参a, b。max函数执行的结果(a或b)将返回给变量z。最后由主函数输出z的值。

输入：80 61  
输出：maximum=80

### 5.4 形式参数与实际参数

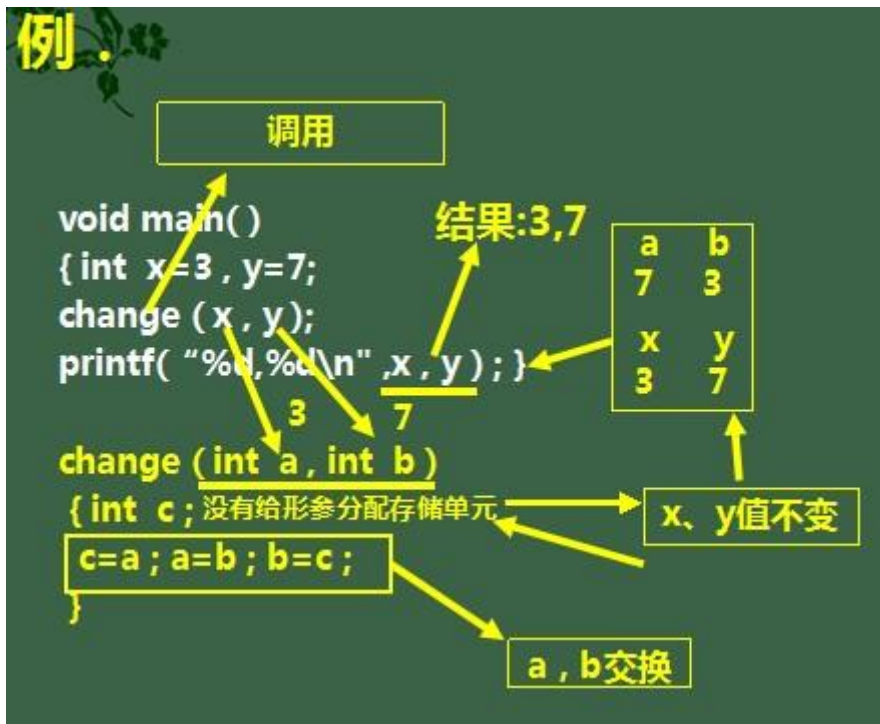
1、定义函数中、函数名后面括号中的变量名、数组名等称为形参。

2、调用函数中、函数名后面括号中的变量名、数组名等称为实参。

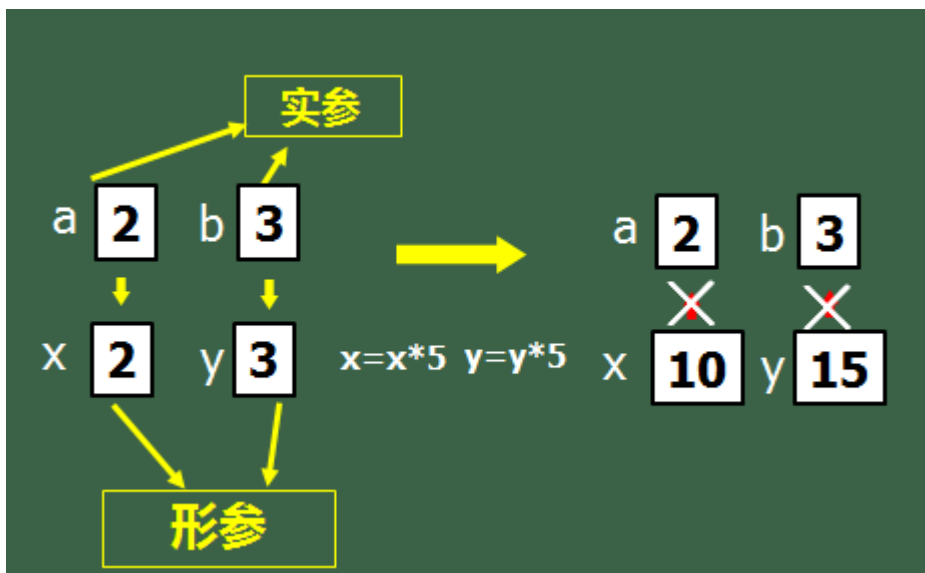
注意：

实参与形参的类型必须一致（字符与整型通用）；表示未具体给定的量，不赋值

形参为变量名时，对应实参应为表达式，调用函数时把实参值单向传递给形参；  
必须已赋值、其值用来传递、给形参、单向传递



函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。



对被调用函数的声明

三种声明形式：

- 
1. 函数类型 函数名( );
  2. 函数类型 函数名(形参 1 名, 形参 2 名, ..... );
  3. 函数类型 函数名(形参 1 类型 形参 1 名,  
形参 2 类型 形参 2 名, ..... );

若某函数中要调用在其后面定义的非 int 类型函数, 则需在本函数说明部分或文件开头对要调用的函数进行声明

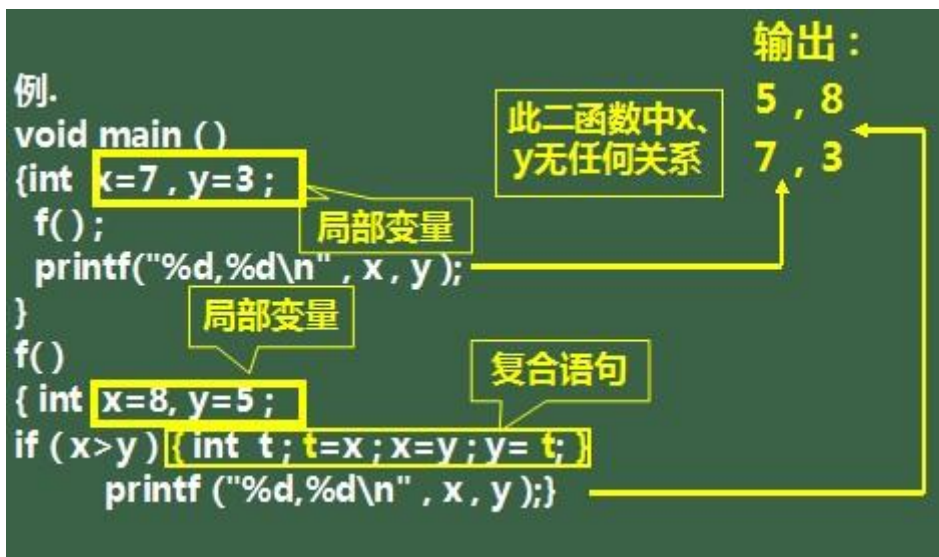
```
#include<stdio.h>
void main( )
{ float add(float x,float y);      //对被调函数 add 的申明
  float a,b,c;
  scanf("%f,%f",&a,&b);
  c=add(a,b);
  printf("sum is %f",c) ; }

float add ( float x,float y )      //被调函数 add

{float  z;
  z=x+y;
  return z;
}
```

## 6.5 局部变量和全局变量

从变量的作用域角度分类



(一)局部变量：形参、函数体中定义的变量

作用范围：某个局部

也称为内部变量，在函数内定义，作用域仅限于函数内，离开该函数后再使用这种变量是非法的。

- 1) 主函数中定义的变量也只能在主函数中使用，不能在其它函数中使用。
- 2) 形参变量是属于被调函数的局部变量，实参变量是属于主调函数的局部变量。
- 3) 允许在不同的函数中使用相同的变量名，它们代表不同的对象，互不干扰。

例如：

```

int f1(int a)      /*函数 f1 , a,b,c 有效*/
{int b,c;
  ..... }
int f2(int x)      /*函数 f2, x,y,z 有效*/
{int y,z;
  ..... }
main()            /*主函数, m, n 有效*/
{int m,n;
  ..... }
  
```

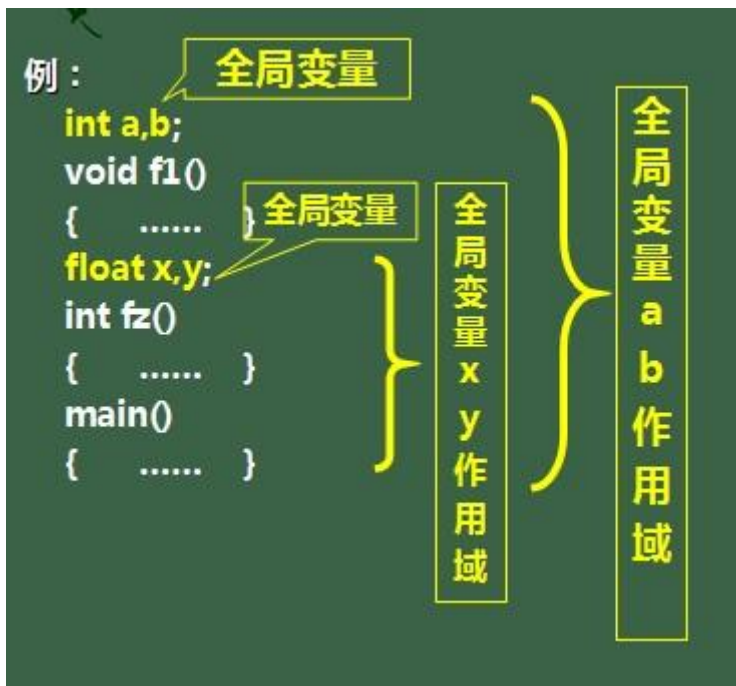
所有变量都是局部变量

(二)全局变量：在函数之外定义的变量

作用范围：从定义点到文件结束

也称为外部变量，在函数外部定义的变量，作用域是整个源程序。如果在某个函数中使用后面说明的全局变量，应用说明符 `extern` 声明后才能使用，但在一个函数之前定义的全局变量，在该函数内使用可不再加以说明。





## C 语言程序设计

# 第 6 节 数组

### 本讲要点

- 一维数组的定义
- 一维数组的引用
- 一维数组的初始化
- 二维和 multidimensional arrays

### 授课内容

把具有相同类型的若干变量按有序的形式组织起来。这些按序排列的同类数据元素的集合称为数组。一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型或是构造类型。数组有数值数组、字符数组、指针数组、结构数组等各种类别。

---

## 6.1 一维数组的定义和使用

一维数组的定义方式：任一种基本数据类型或构造数据类型

类型说明符数组名：用户定义的数组标识符

[常量表达式]：表示数据元素的个数，也称为数组的长度

例如：

```
int a[10];
float b[10],c[20];
char ch[20];
```

注意：

数组元素与变量一样地使用（整个数组不能与变量一样地使用）。

数组元素与变量主要区别：

1. 它们的名字形式不同
2. 同一数组中各元素必须同类型
3. 数组名不能与其它变量名相同。

例如：main()

```
{ int a ;
    float a[10] ;    //错误，变量重名
    .....}
```

4. 方括号中常量表达式表示数组元素的个数，但是其下标从 0 开始计算。因此 5 个元素分别为：

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

5. 定义时不能在方括号中用变量来表示元素的个数，但是可以是符号常数或常量表达式。

6. 允许在同一个类型说明中，说明多个数组和多个变量。例如：

```
int a,b,c,d,k1[10],k2[20];
```

例如：

```
#define FD 5
main()
{ int a[3+2],b[7+FD];    // ✓
    .....}

main()
{ int n=5;
    int a[n];    // × 定义时不能用,但引用时可以
```

---

---

.....}

## 6.2 一维数组元素的引用

数组元素是组成数组的基本单元，也是一种变量，其标识方法为数组名后跟一个下标，表示元素在数组中的顺序号。

数组元素的一般形式为：

数组名[下标]

其中下标只能为整型常量或整型表达式。如为小数时，C 编译将自动取整。

例如：a[5]

a[i+j] //定义时不能用,引用时可以

a[i++]

都是合法的数组元素。

数组元素通常也称为下标变量。必须先定义数组，才能使用下标变量。在 C 语言中只能逐个地使用下标变量，而不能一次引用整个数组。

例如：

输出有 10 个元素的数组必须使用循环语句逐个输出各下标变量：

for(i=0; i<10; i++) // ✓

printf("%d",a[i]);

而不能用一个语句输出整个数组。

下面的写法是错误的：

printf("%d",a); // ×

---

例

```
void main()
{ int i,a[10];
  for(i=0;i<=9;i++)
    a[i]=i;
  for(i=9;i>=0;i--)
    printf("%d",a[i]);
}
```

逐一  
赋值

逐一  
输出

a[0]	0
a[1]	1
a[2]	2
a[3]	3
a[4]	4
a[5]	5
a[6]	6
a[7]	7
a[8]	8
a[9]	9

空格

运行结果：9 8 7 6 5 4 3 2 1 0

例

例：

```
void main()
{ int i,a[10];
  for(i=0;i<10;)
    a[i++]=i;
  for(i=9;i>=0;i--)
    printf("%d",a[i]);
}
```

a[0]	0
a[1]	1
*****	
a[9]	9

a[9]	9
a[8]	8
*****	
a[0]	0

运行结果：9876543210

一维数组的初始化

---

给数组赋值的方法除了用赋值语句对数组元素逐个赋值外，还可采用初始化赋值和动态赋值的方法。

初始化赋值的一般形式为：**类型说明符 数组名[常量表达式]={值，值……值};**

其中在{}中的各数据值即为各元素的初值，各值之间用逗号间隔。

例如：

```
int a[10]={ 0,1,2,3,4,5,6,7,8,9 };
```

相当于

```
a[0]=0;a[1]=1; ...a[9]=9;
```

对数组的初始化赋值还有以下几点规定：

1) 可以只给部分元素赋初值。

当{}中值的个数少于元素个数时，只给前面部分元素赋值。

例如：

```
int a[10]={0,1,2,3,4};
```

表示只给 a[0]~a[4] 5 个元素赋值，而后 5 个元素自动赋 0 值。

2) 只能给元素逐个赋值，不能给数组整体赋值。

例如给十个元素全部赋 1 值，只能写为：

```
int a[10]={1,1,1,1,1,1,1,1,1,1};
```

而不能写为：

```
int a[10]=1; //错误
```

3) 如给全部元素赋值，则在数组说明中，可以不给出数组元素的个数。

例如：

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

可写为：

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

**例**

```
void main()
{int i,max,a[10];
 printf("input 10 numbers:\n");
 for(i=0;i<10;i++)
  scanf("%d",&a[i]);
 max=a[0];
 for(i=1;i<10;i++)
  if(a[i]>max) max=a[i];
 printf("maximum=%d\n",max);}
```

定义一个10个元素的整型数组

逐一输入十个数组元素的值

逐一比较得出最大值

输入: 1 4 5 2 3 1 4 6 7 3 0

输出: maximum=31

---

## 6.3 二维和 multidimensional 数组的定义和使用

从一维数组可以延伸到二维甚至 multidimensional 数组

数组 a: `a[0] a[1] a[2] a[3] ..... a[29] a[30]`

数组 c: `c[0][0] c[0][1] c[0][2] c[0][3] c[0][4]  
c[1][0] c[1][1] c[1][2] c[1][3] c[1][4]  
c[2][0] c[2][1] c[2][2] c[2][3] c[2][4]`

数组 x: `x[0][0][0] x[0][0][1] x[0][0][2]  
x[0][1][0] x[0][1][1] x[0][1][2]  
x[1][0][0] x[1][0][1] x[1][0][2]  
x[1][1][0] x[1][1][1] x[1][1][2]  
x[2][0][0] x[2][0][1] x[2][0][2]  
x[2][1][0] x[2][1][1] x[2][1][2]`

数组元素名一般形式

数组名[表达式 1][表达式 2][表达式 3] ... [表达式 n]

例: `a[3] b[i][j] c[2][u*4+1.5][k]`

例: 若 `i=2, j=3.7` 则

`b[i][j]`、`b[j-i+1][3.256]` 与 `b[2][3]`

表示同一个数组元素

若此某一表达式为实型, 则只把该表达式值的整数部分作为相应下标;

所有数组名部分相同、维数相同、相应下标相同的数组元素名表示同一个数组元素。

### 数组的定义

把下面形式的数组说明符写在定义部分

数组名[整型常量表达式 1][整型常量表达式 2] .....  
[整型常量表达式 n]

例

```
#define N 10
void main( )
{ int a ,b=2 , f[30],k;
  char c1 ,c2 ,a[2][2*N+1], w[k] ;
  .....
}
```

引用时注意: 只引用数组元素、不引用数组(字符型数组例外) multidimensional 数组可按行分段赋值; 也可按行连续赋值。

例如对数组 `a[5][3]`, 以下赋值结果等价:

1) 按行分段赋值可写为:

```
int a[5][3]={ {80,75,92},{61,65,71}, {59,63,70},{85,87,90},{76,77,85}};
```

2) 按行连续赋值可写为:

```
int a[5][3]={ 80,75,92,61,65,71,59,63, 70,85,87,90,76,77,85};
```

---

数组中各元素在内存中的排列顺序可以按行排列，或按列排列。以下介绍按行排列。

数组 c:

```
c[0][0] c[0][1] c[0][2] c[0][3] c[0][4]
c[1][0] c[1][1] c[1][2] c[1][3] c[1][4]
c[2][0] c[2][1] c[2][2] c[2][3] c[2][4]
```

数组 x:

```
x[0][0][0] x[0][0][1] x[0][0][2]
x[0][1][0] x[0][1][1] x[0][1][2]
x[1][0][0] x[1][0][1] x[1][0][2]
x[1][1][0] x[1][1][1] x[1][1][2]
x[2][0][0] x[2][0][1] x[2][0][2]
x[2][1][0] x[2][1][1] x[2][1][2]
```

可以只对部分元素赋初值，未赋初值的元素自动取0值。例如：

**int a[3][3]={1},{2},{3} ;**

赋值后各元素的值为：

**int a [3][3]={0,1},{0,0,2},{3}} ;**

赋值后的元素值为：

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

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

例



## 例 写出下面程序的运行结果并比较它们的区别

```
void main()  
{int a[4][4]=  
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16},j,k;  
  for(j=0;j<=3;j++)  
    for(k=0;k<=3;k++)  
      printf(" %3d",a[j][k]);  
  printf("\n");  
}
```

循环外换行,做一次

输出:

j=3 {

k=0			
a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	2	3	4
a[1][0]	a[1][1]	a[1][2]	a[1][3]
5	6	7	8
a[2][0]	a[2][1]	a[2][2]	a[2][3]
9	10	11	12
a[3][0]	a[3][1]	a[3][2]	a[3][3]
13	14	15	16

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

```
void main()  
{int a[4][4]=  
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16},j,k;  
  for(j=0;j<=3;j++)  
    { for(k=0;k<=3;k++)  
      printf("%3d",a[j][k]);  
      printf("\n"); }  
}
```

结果:

```
1   2   3   4  
5   6   7   8  
9  10  11  12  
13 14  15  16
```

解析:

第一层循环内换行,输出每一行,做一次

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



5	6	7	8
a[2][0]	a[2][1]	a[2][2]	a[2][3]
9	10	11	12
a[3][0]	a[3][1]	a[3][2]	a[3][3]
13	14	15	16

# 第 7 节 指针

## 本讲要点

- 地址指针的基本概念
- 变量的指针和指向变量的指针变量
- 一维数组与指针
- 二维数组的指针
- 字符串与指针
- 指向指针的指针

## 授课内容

### 7.1 地址指针的基本概念

内存单元的编号也叫做地址，也称为指针。

- 1、数据在内存中按单元（字节）存放，如整型量占 2 个单元等，为正确访问，每个内存单元都有编号。
  - 2、内存单元的地址即为指针，其中存放的数据才是该单元的内容。
-

3、如果用一个变量来存放指针，这种变量称为指针变量，指针变量的值就是某个内存单元的地址即指针。



## 7.2 变量的指针、指向变量的指针变量

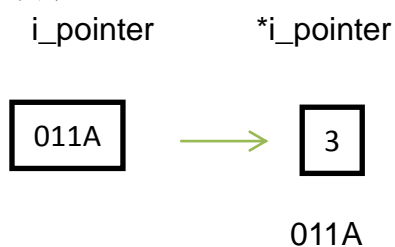
存放地址（指针）的变量称为指针变量。一个指针变量的值就是某个变量的地址或称为某变量的指针。

例如，**i\_pointer** 代表指针变量，而**\*i\_pointer** 是 **i\_pointer** 所指向的变量。  
下面语句作用相同，均为将 3 赋给指针变量 **i\_pointer** 所指向的变量。

**i=3;**

**\*i\_pointer=3;**

图示：



定义一个指针变量

格式：类型说明符 \*变量名；

例如：



`int *p2;` 指向整型变量的指针变量  
`float *p3;` 指向浮点变量的指针变量  
`char *p4;` 指向字符变量的指针变量

至于 p2、p3、p4 究竟指向哪一个变量，  
应由向 p2、p3、p4 赋予的地址来决定。

一个指针变量只能指向同类型的变量，如 p3 只能指向浮点变量，不能时而指向一个浮点变量，时而又指向一个字符变量。

指针变量的引用

指针变量使用之前也要定义说明，而且必须赋予具体的值。指针变量的赋值只能赋予地址。

两个有关的运算符：

- 1) &：取地址运算符。
- 2) \*：指针运算符或称“间接访问”运算符。

如：&j 表示变量 j 的地址——2000

\*p 表示指针变量 p 所指向的变量 ——j

指针变量初始化和赋值

`int j=153,k=-42,m=12561,*p=&j,*q; q=&k;`

表示定义 p、q 为指针变量 (\*p、\*q 为 int 型变量)

---

(1) 指针变量初始化的方法:

```
int a;  
int *p=&a;    //定义时带*号
```

(2) 赋值语句的方法:

```
int a;  
int *p;  
p=&a;    //赋值时不带*号
```

1) 不能把一个数赋予指针变量:

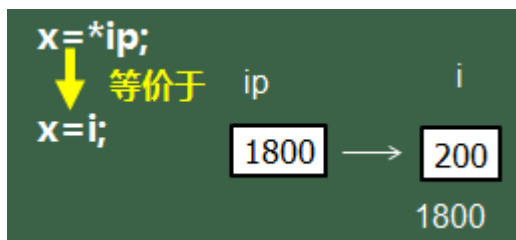
```
int *p;  
p=1000;
```

2) 被赋值的指针变量前不能再加“\*”说明符:

```
*p=&a
```

3) 可以通过指针变量 ip 间接访问变量 i,

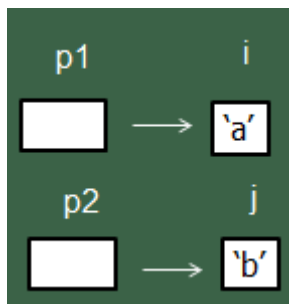
例如



再例如:

```
int i,j,*p1,*p2;  
i='a';  
j='b';  
p1=&i;  
p2=&j;
```

如图:



---

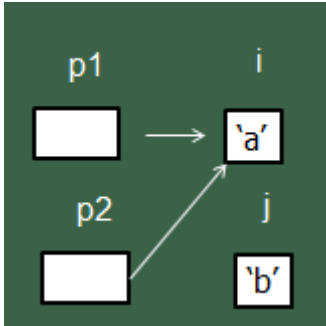
这时赋值表达式:

**p2=p1**

就使 p2 与 p1 指向同一对象 i,

此时 \*p2 就等价于 i,而不是 j,

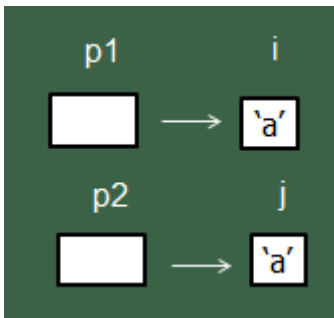
如图:



如果执行如下表达式:

**\*p2=\*p1;**

则表示把 p1 指向的内容赋给 p2 所指的区域, 此时就变成图所示



若指针变量 p 指向变量 j (即 `p=&j`)

则有:      1、 `*&j` ——— `*(&j)` ——— `*p` ——— `j`

             2、 `&*p` ——— `&(*p)` ——— `&j` ——— `p`

指针变量可出现在表达式中。

设: `int x,y, *px=&x;`

指针变量 `px` 指向整数 `x`,则 `*px` 可出现在 `x` 能出现的任何地方。例如:

`y=*px+5;`      //把 `x` 的内容加 5 并赋给 `y`

`y=++*px;`      // `px` 的内容加上 1 之后赋给 `y` , 等价于 `++(*px)`

`y=*px++;`      //等价于 `y=*px; px++`

---

例

main()

{int a,b;

int \*pointer\_1, \*pointer\_2;

a=100;b=10;

pointer\_1=&a;

pointer\_2=&b;

printf("%d,%d\n",a,b);

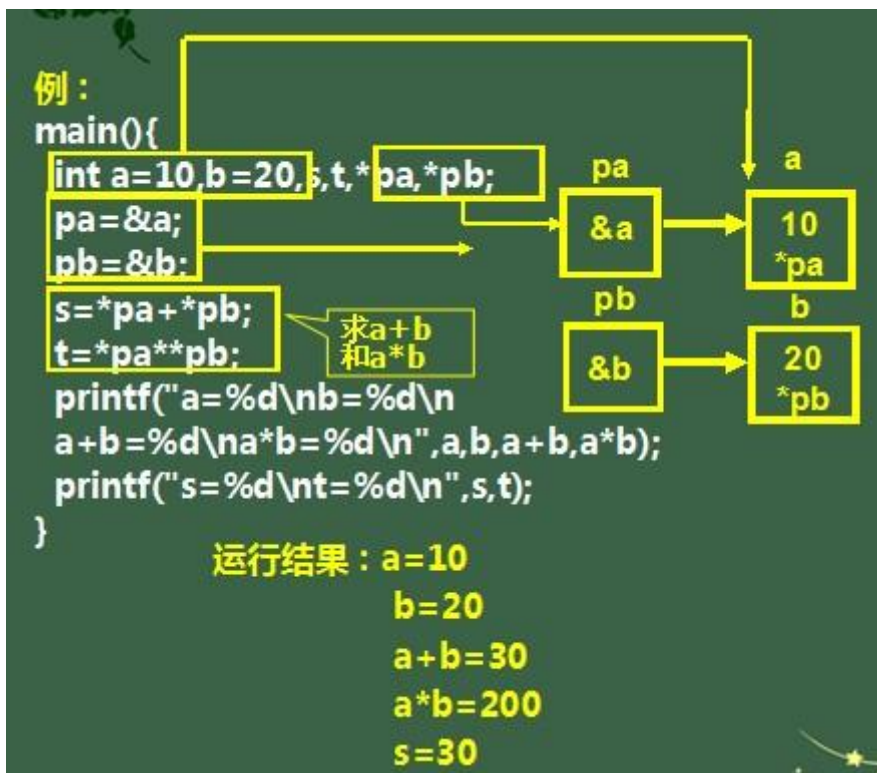
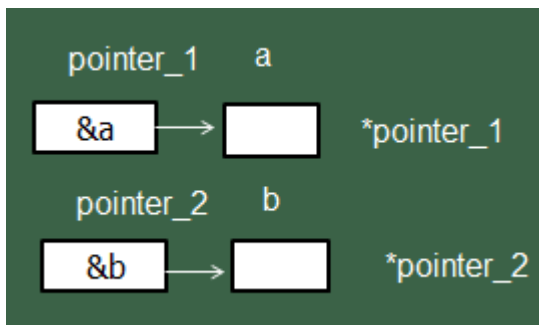
printf("%d,%d\n",\*pointer\_1, \*pointer\_2);

}

运行结果: 100, 10

100, 10

图示:



---

### 7.3 一维数组与指针

- 数组的指针是指数组的起始地址。
- 数组元素的指针是数组元素的地址。
- 一个数组是由连续的一块内存单元组成的。
- 数组名就是这块连续内存单元的首地址。
- 定义一个指向数组元素的指针变量的方法，与以前介绍的指针变量相同。

如果指针变量  $p$  的初值为  $\&a[0]$ , 则:

- 1)  $p+i$ 、 $a+i$ 、 $\&a[i]$  等价，  
指向  $a$  数组的第  $i$  个元素。
- 2)  $*(p+i)$ 、 $*(a+i)$ 、 $a[i]$   
等价，第  $i$  个元素的值。
- 3) 规定:  $p[i]$  与  $*(p+i)$  和  $a[i]$  等价。

例如:

`int a[10];`          定义  $a$  为包含 10 个整型数据的数组  
`int *p;`            定义  $p$  为指向整型变量的指针  
`p=&a[0];`          对指针变量赋值

把  $a[0]$  元素的地址赋给指针变量  $p$ 。也就是说， $p$  指向  $a$  数组的第 0 号元素。

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

### 7.4 二维数组的指针（元素的指针、行指针）

定义一个数组:

`int a[3][4]={{2,5,1,7},{3,9,10,6},{4,8,12,11}}`

---

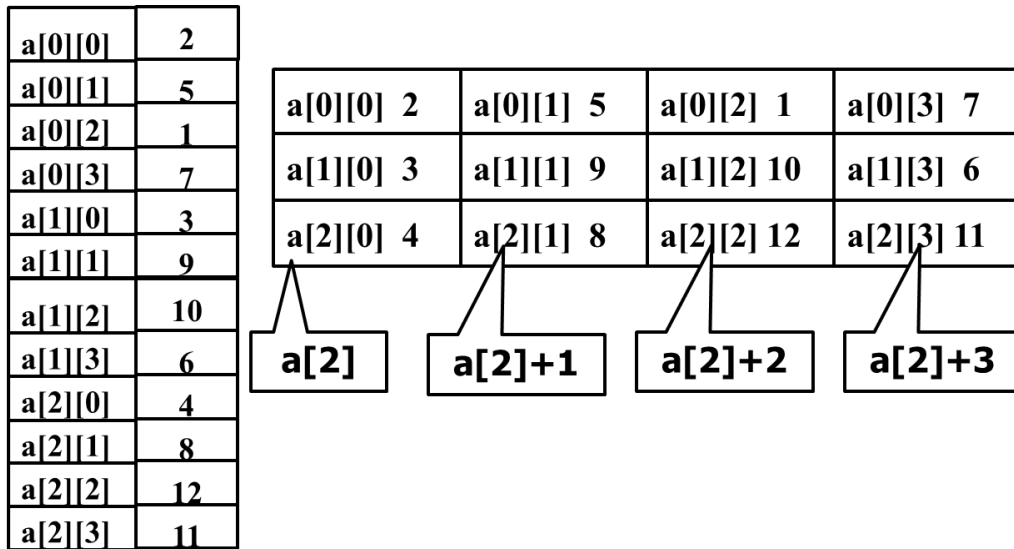
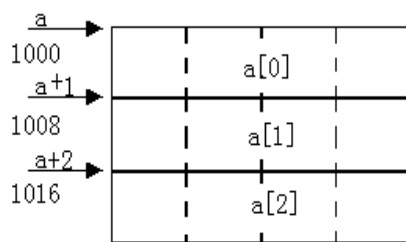


图 数组 a 物理结构:

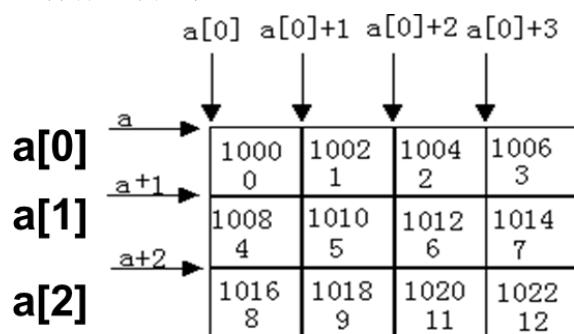
数组 a 逻辑结构:

int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12}

**a** 是二维数组名，是数组的首地址，也是 0 行的首地址，等于 1000



允许把一个二维数组分解为多个一维数组来处理，**a** 分解为一维数组：**a[0]**，**a[1]**，**a[2]**。每一个一维数组又含有四个元素。





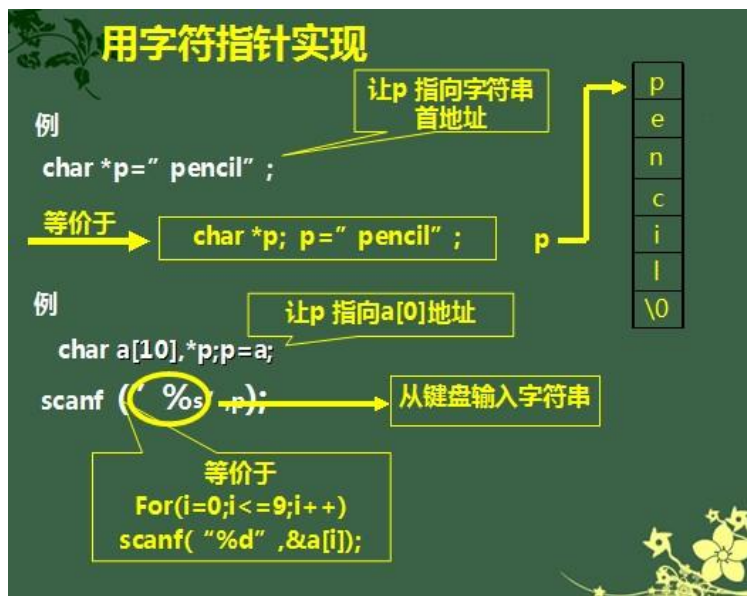
## 7.5 字符串与指针

C语言中字符串的表示形式有两种：

1. 用一个一维数组实现：

```
char c[]={'p','e','n','c','i','l','\0'}
```

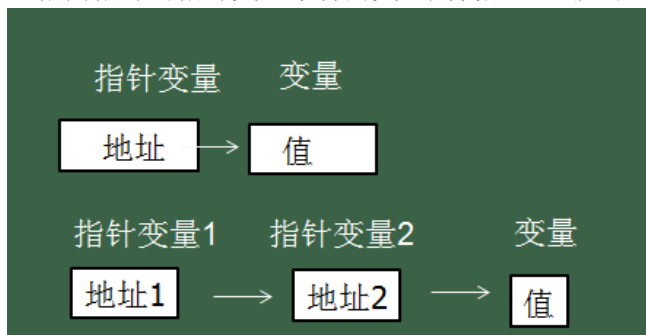
2. 用字符指针实现，如下图：



## 7.6 指向指针的指针

如果一个指针变量存放的又是另一个指针变量的地址，则称这个指针变量为指向指针的指针变量。

通过指针访问变量称为间接访问。由于指针变量直接指向变量，所以称为“单级间址”。而如果通过指向指针的指针变量来访问变量则构成“二级间址”。

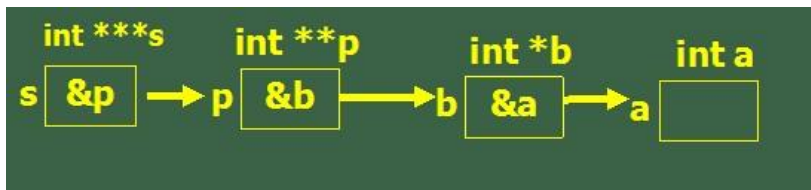


指向指针的指针变量：存放指针变量的地址

例：

```
int ***s,**p,**q,a,*b,*w[10];  
b=&a;p=&b;s=&p;q=w;
```

图示:



例. 使用指向指针的指针。

`main()`

```
{char *name[]={ "CHINA","AMERICA",  
    "AUSTRALIA", "FRANCE","GERMAN"};  
char **p;  
int i;  
for(i=0;i<5;i++)  
    {p=name+i; printf("%s\n",*p); }  
}
```

