

程序设计基础



马君

E_mail: majun_ufe@163.com

Tel: 13609112329

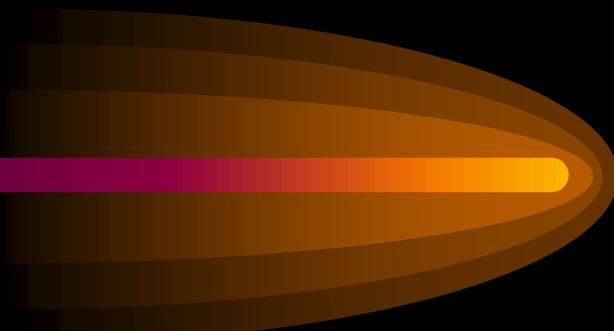
第一章 C语言概述

1. C语言的发展

与计算机对话是从低级语言开始逐步发展起来的。



2. C语言的特点

- 
- ①提供了一整套控制语句(9种)。
 - ②数据类型丰富。
 - ③可以直接访问物理地址，进行位操作。
 - ④提供了30多种丰富的运算符。
 - ⑤具有预处理功能。
 - ⑥具有很好的可移植性。
 - ⑦生成目标程序质量高，程序执行效率高。

3. 简单的C程序介绍

例

```
main( )  
{  
    printf("This is a c program. \n");  
}
```

例

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

例

```
int max(int x, int y)
{
    int z;
    if (x>y) z=x;
    else z=y;
    return(z);
}

main()
{
    int a, b, c; /*定义变量*/
    scanf ("%d, % d", &a, &b);
    c=max(a, b);
    printf ("max=%d\n", c);
}
```

C函数从main()开始执行，仅能有一个。

C 程序是由函数组成的。

变量定义和语句之后必须有一个分号（;）

一行内可写几个语句，一个语句可分写在多行。

可以用/*.....*/作注释。

C 语言没有输入输出语句。

4. 函数的组成

每一个源程序仅且仅能包含一个main()函数。

一个函数是由函数的说明部分和函数体两部分组成。

① 函数的说明部分

```
int max(int x,int y)
{
    int z;
    if (x>y) z=x;
    else z=y;
    return(z);
}
```

int max (int x, int y)

↑ ↑ ↑

函数类型 函数名 函数参数列表

② 函数体 : { } 中的部分, 功能的实现。

函数体

变量定义 (int a, b, c;)

执行部分 (由若干语句组成)

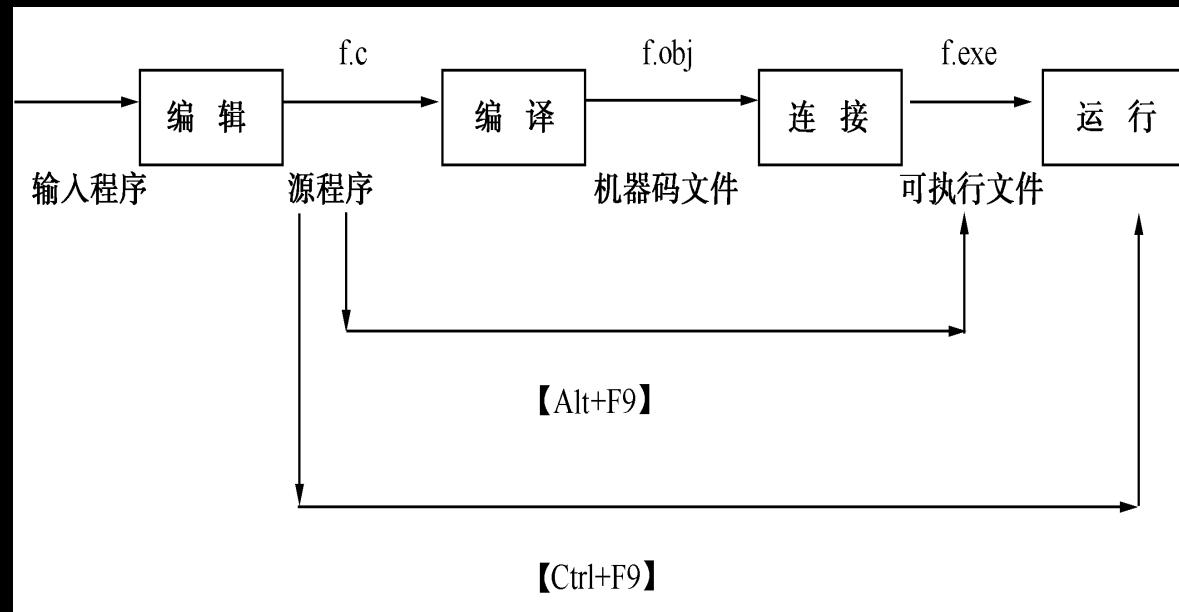
4.上机步骤

①系统的启动

鼠标双击TC.EXE的快捷图标，进入TC集成环境。

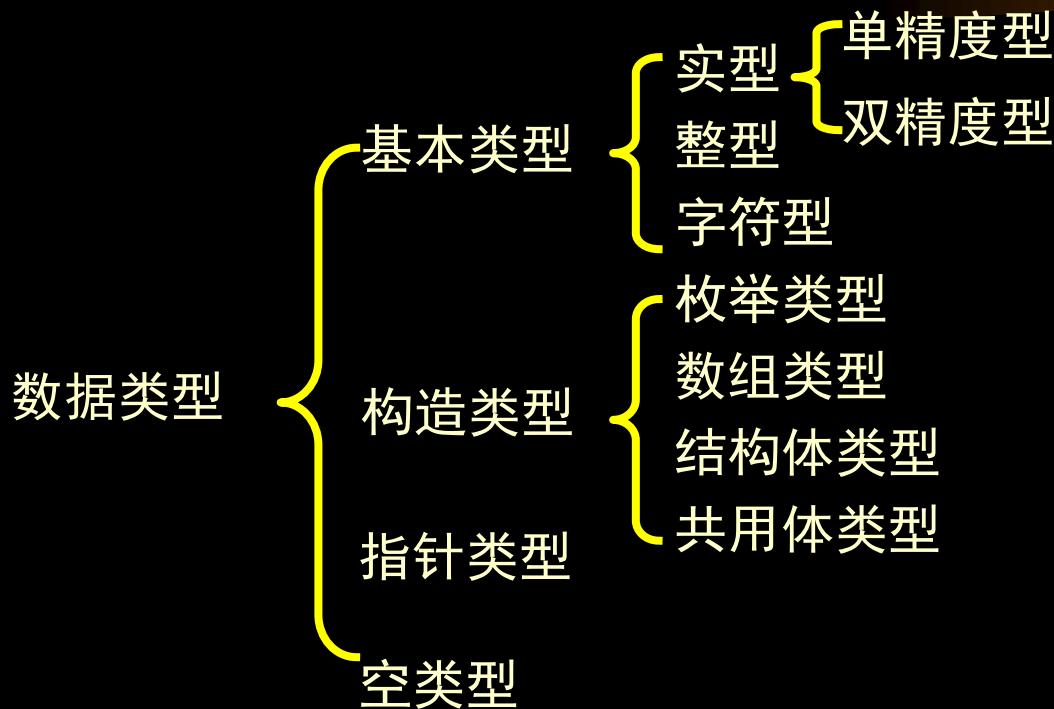
②界面介绍

③使用小结



第二章 数据类型 运算符与表达式

1. C的数据类型



2.数据类型的重要性

在编写程序时，必须做好两件事：

- ①是描述数据。
- ②是描述数据的加工方法。

前者通过数据定义语句实现，而后者通过若干执行语句来完成。

数据类型最终决定该类型数据的取值范围和基本运算。

3. 标识符

每种程序语言都规定了在程序中为对象命名的规则。

这些名字包括：变量名、常量名、数组名、函数名等。

通常被统称为标识符。

(1) 标识符命名规则：

标识符是以字母或下画线开头，由字母、数字和下画线构成的字符串。

call...name , 39test_string1

注意：

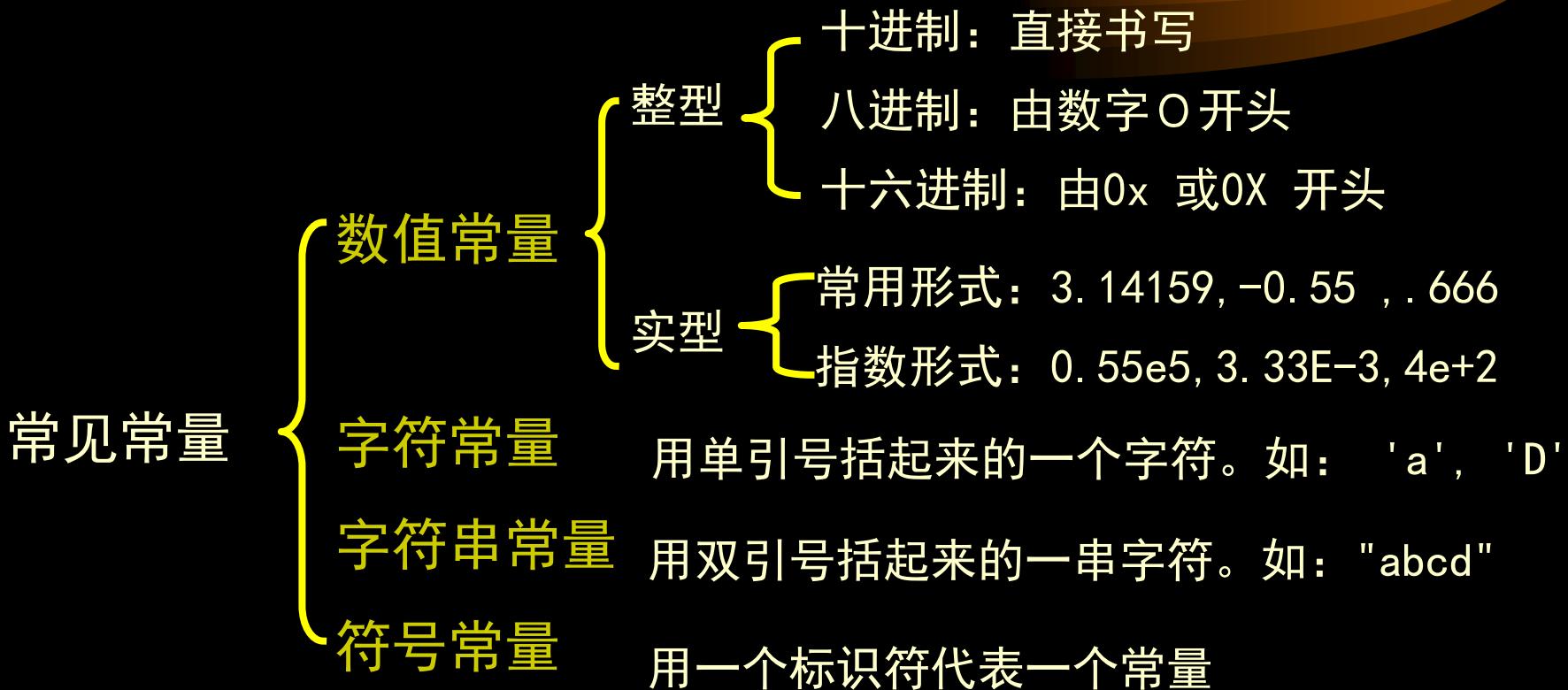
ANSI C 规定标识符的长度可达31个字符，但一般系统使用的标识符，其有效长度不超过8个字符。

标识符不能采用系统的关键字（保留字）。

4. 常量与变量

(1) 常量

常量：程序中其值不发生变化的量。



注意： 'a' 和 "a" 的区别

例 已知圆半径r，求圆周长c和圆面积s的值。

```
#define PI 3.1416

main()
{
    float r, c, s;

    scanf ("%f", &r);          /*从键盘输入一个实型数给变量r*/
    c=2*PI*r;                  /* 编译时用3.1416替换PI */
    s=PI*r*r;                  /* 编译时用3.1416替换PI */
    printf ("c=%6.2f, s=%6.2f\n", c, s);
}
```

(2)转义字符：常用“\”开头后跟一个字符，但含义改变。

字符形式	功能
\n	回车换行
\t	横向跳格（即跳到下一个输出区）
\v	横向跳格
\b	退格
\r	回车
\f	走纸换页
\\"	反斜杠字符“\”
\'	单引号字符(')
\ddd	1到3位8进制数所代表的字符
\xhh	1到2位16进制数所代表的字符

例: main()

```
{printf('ab\c\t\de\r\f\tg\n');  
printf('h\ti\b\bj\k'); }
```

例:

整型数: 125 , 0125 , 0x125 , +35 , -235 , -035

实型数: 3.14159 , -555.6 , 888.0 , 0.88 , 8.88e+18

注意:

用指数形式表示的浮点数必须有尾数, 指数部分必须是整数。

如: e4 , .e3 , 0.25e4.5 , e 等是错误的。

(3) 变量

变量：程序中其值可发生变化的量。

每一个变量都应有一个名字，称为变量名。而且在内存中占据一定的存储空间，用来存放变量的值。

C 语言规定对使用的变量必须先定义，后使用。

目的：保证程序中变量名的正确使用。

可分配相应的存储空间。

便于检查变量所进行的运算是否合法。

①整型变量：(四种类型)

类型	字节数	数的范围
基本型(int)	2	-32768~32767
短整型(short int)	2	-32768~32767
长整型(long int)	4	-2147483648~2147483647
无符号型	无符号整型(unsigned int)	2 0~65535
	无符号短型(unsigned short)	2 0~65535
	无符号长整型(unsigned long)	4 0~4294967295

整型变量的定义格式: 类型 变量名表列;

例: main()

```
{int a,b,c,d;  unsigned u;  
    a=12;  b=-24;  u=10;  
    c=a+u;  d=b+u;  
    printf("a+u=%d,b+u=%d\n",c,d);  
}
```

②实型变量：(两类)

类型	所占位数	数的范围
单精度实型(float)	32	$10^{-38} \sim 10^{38}$
双精度实型(double)	64	$10^{-308} \sim 10^{308}$

单精度实型提供7位有效数字， 双精度实型提供15~16位有效数字。

③字符变量：**char**

用来存放一个字符常量。占一个字节(8位)， 存放该字符的ASCII码值。

如：**char c1,c2;**

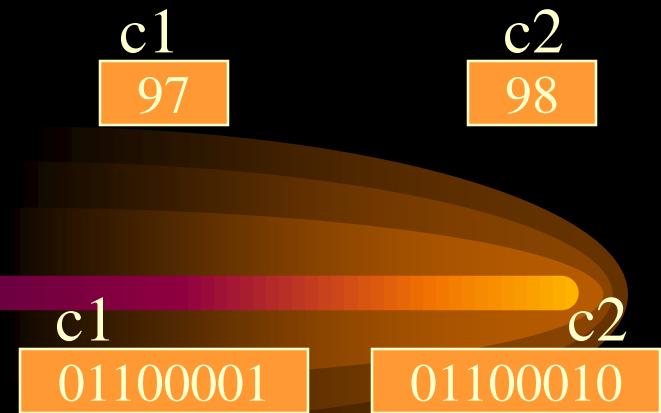
c1='a'; c2='b';

例： main()

```
{char c1,c2;  
c1=97; c2=98;  
printf("%c %c",c1,c2);
```

}

c1='a'; c2='b';



例： main()

```
{char c1,c2;  
c1='a'; c2='b';  
c1=c1-32; c2=c2-32;  
printf("%c %c",c1,c2);  
}
```

ASCII码表中大小写字母之间具有：

'a'='A'+32

C语言中允许字符型数据与整型数据互相赋值。如：

int i;	i='a';
char c;	c=97;

例： main()

```
{int i;  
char c;  
i='a'; c=97;  
printf("%c,%d\n",c,c);  
printf("%c,%d\n",i,i);  
}
```

输出结果： a,97

a,97

④变量赋初值： 格式为： 类型 变量名=常量值

在定义变量的同时给变量初始化。

如： int a=10;

float f=5.55;

char c='a';

int a=15,b=15,d=15;

相当于：
int a;
a=10;

不能写成： int a=b=d=15;

5. 运算符与表达式

C语言提供了丰富的运算符，可以对数据进行各种处理，从而保证了各种操作可以方便地实现。

表达式是由运算符和运算对象组成的式子，运算对象就是在程序中要处理的各种数据。

① 算术运算符和算术表达式

基本算术运算符： + 、 - 、 * 、 / 、 %

(注：两个整数相除，结果为整数。)

算术表达式和运算符的优先级与结合率：

用算术运算符和括号“()”将运算对象连接起来，符合C语法规则的式子称为算术表达式。

运算对象：常量、变量、函数等。

优先级：先乘除后加减。

结合率：自左至右。

例如： $a*b/c-1.5+a'$; $a-(b*c)$

自增、自减运算符：++，--（使变量的值增 1 或减 1）

结合率：自左至右。

例如： $++i$, $-i$ （先自增或自减）

$i++$, $i--$ （后自增或自减）

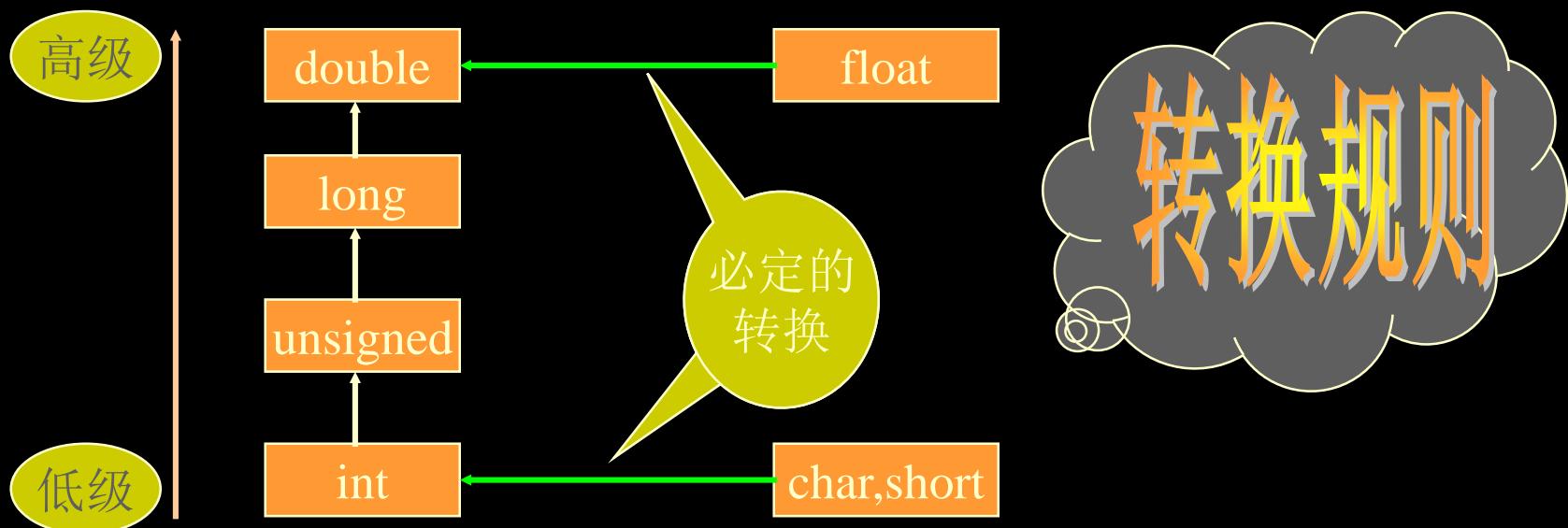
$-i++$ 相当于 $-(i++)$

$i++j$ 相当于 $(i++)+j$

只能用于变量

数据类型转换:

C语言允许不同类型的数值型数据可以混合运算，运算时系统自动将数据转换成同一类型数据。



例如：若有 int i, float f, double d, long e

则 $10 + 'a' + i * f - d / e$ 表达式运算次序为：

将'a'转换成97，然后 $10 + 'a'$ 运算。

将i和f都转换成double型，然后 $i * f$ 运算。

将 的结果转换为double型，然后与 的结果相加。

将e转换成double型，然后 d / e 运算。

用 的结果减 的结果。

强制类型转换：格式如下

(类型名)(表达式)

例如：(double) a、(int)(x+y)、(float)(5%3)、(int)x+y

②赋值运算符和赋值表达式

赋值符号： =

功能：将赋值符右边表达式的值赋给赋值符左边的一个变量。

数据类型若不一致，要进行类型转换。转换方式为：

将实型数据赋给整型变量时，舍弃实数的小数部分。

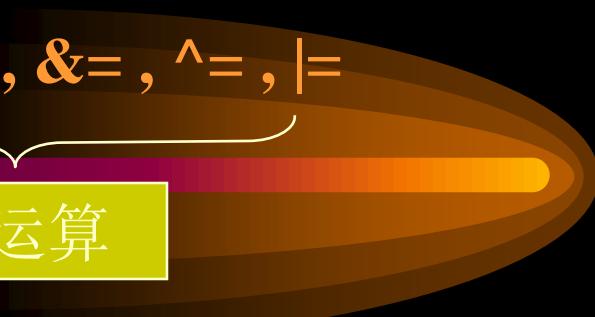
如： $i=5.65$, i 的值为 5。

将整型数据赋给实型变量时，数值不变。

将字符数据赋给整型变量时，将字符数据放到整型变量低8位中。

复合的赋值运算符：（共十个）

$+=$, $-=$, $*=$, $/=$, $\%=$, $<<=$, $>>=$, $\&=$, $\wedge=$, $\mid=$

A decorative swoosh graphic in shades of orange and yellow, positioned above the brace indicating the range of operators.

位运算

例如：
 $a+=3$ 等价于 $a=a+3$
 $x*=y+8$ 等价于 $x=x*(y+8)$
 $x\%=3$ 等价于 $x=x\%3$

注：若右边为表达式
应加圆括号“()”！

赋值表达式：

由赋值运算符将一个变量和一个表达式连接起来的式子。

格式为：<变量><赋值运算符><表达式>

赋 值 给 变 量



赋值运算符结合率为：“自右而左”。

例如： $a=b=c=5$ $a=(b=(c=5))$ a,b,c 值都是5

$a=5+(c=6)$ c值为6, a值为 $5+6=11$

$a=(b=4)+(c=6)$ b值为4,c值为6 a值为 $4+6=10$

$a=(b=10)/(c=2)$ a值为5

$a+=a-=a*a$ 若 $a=5$, 则赋值表达式的值为-40。

计算方法： $a+=(a-=a*a)$

$a=a-a*a=5-5*5=-20$

$a=a+(-20) = (-20) + (-20) = -40$

若x的值是8, 则表达式 $x*=x-=x+=x$ 的值?

③逗号运算符和逗号表达式

逗号运算符： ,

格式： 表达式1,表达式2,表达式3,……,表达式n

优先级： 最低。

从左向右计算每个表达式的值，逗号表达式的值为表达式n的值。

例如： $a=3*5,a*4$

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

$x=(a=3,6*3)$

$x=a=3,6*a$



求三个整数的最大值。

第三章 简单的C程序设计

1.C 语句概述

C语言的语句是用来向计算机系统发出操作指令。每一个为实现特定目的的程序都包含若干个C语句。

C语句的五种分类：

控制语句：完成一定的控制功能（9条）。

函数调用语句：由一次函数调用加一个分号构成。

如：`scanf("%d\n",&a);`

表达式语句：由表达式加一个分号构成。

如：`i=i+1;`

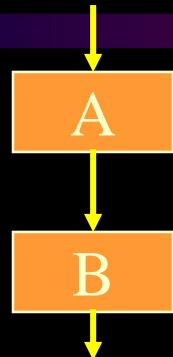
空语句：由一个分号构成。

复合语句：由一对大括号“{}”组成，相当于一个语句。₂₉

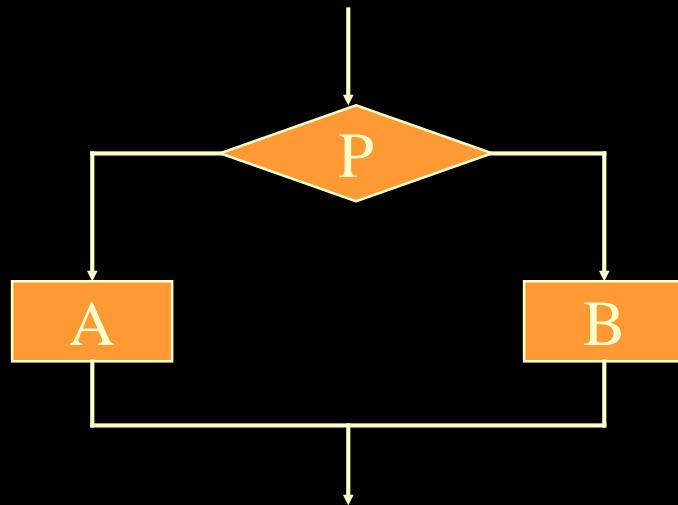
如：`{z=x+y; b=15; printf("%d",b); }`

2. 程序的三种基本结构

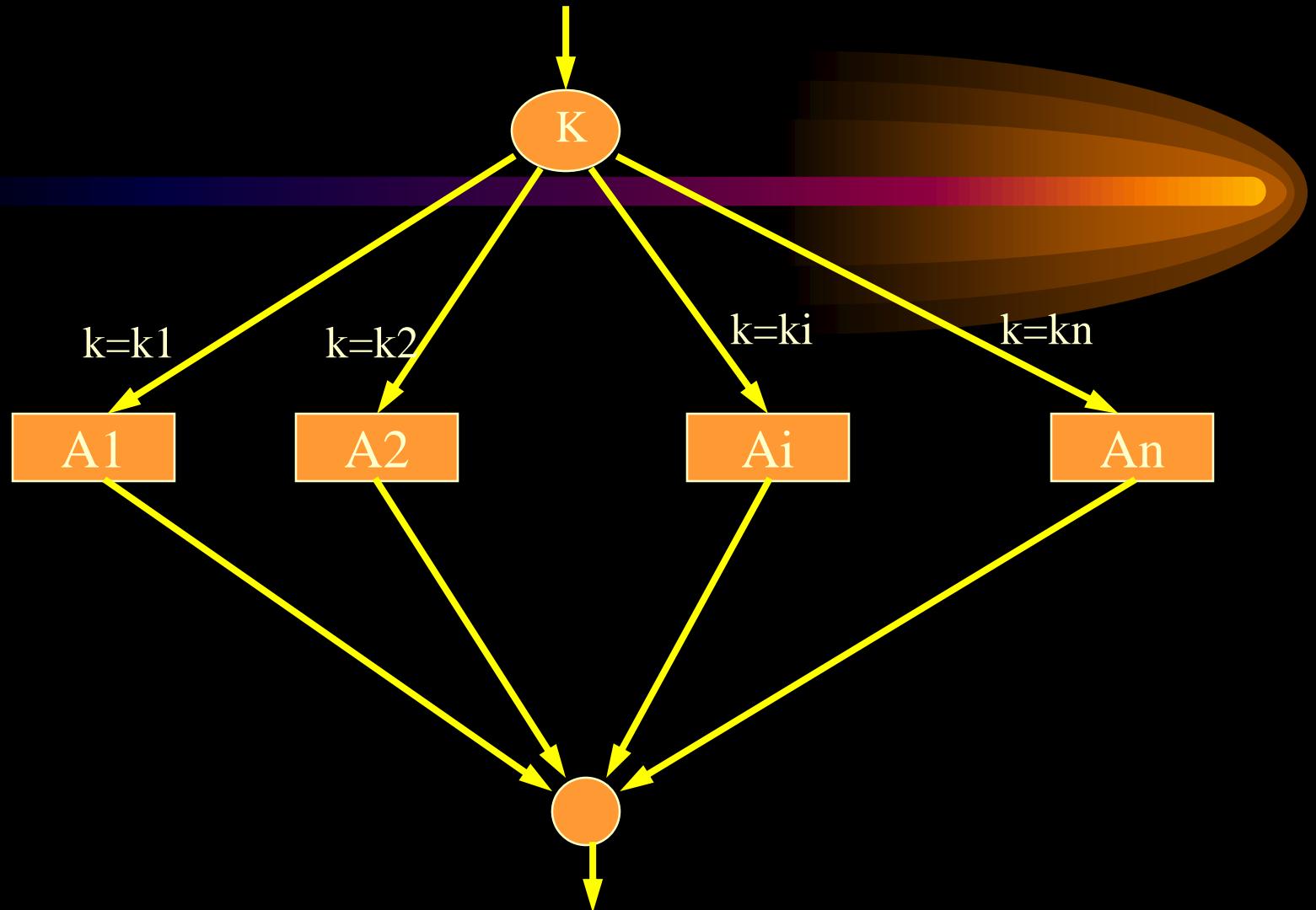
顺序结构：从前向后顺序执行程序。



选择结构：根据判断条件的结果选择执行程序。



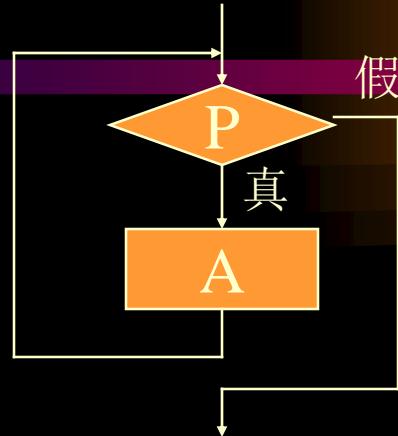
多分支选择结构:



循环结构：根据条件反复的执行某一段程序若干次。

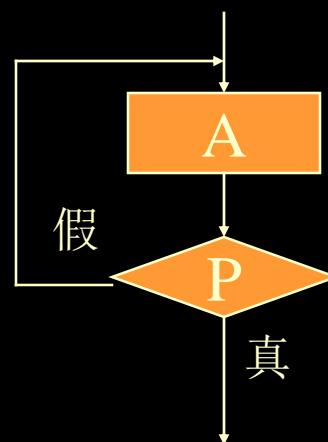
当型循环结构：

流程图



直到型循环结构：

流程图



3. 数据的输入与输出

① 格式输出函数printf()

功能：输出若干个任意类型的数据。

格式：printf("格式控制",参数1,参数2,参数3,……)

由格式说明
和普通字符构成

输出数据。由
表达式构成。

格式说明：由%后跟一个格式字符组成。中间可插入l、m、.n、0、+和-几个附加符号。

普通字符：照原样输出。

例如： `printf("a=%d b=%d",a,b);` （设 `a=12;b=15;`）

输出结果为： `a=12 b=15`

格式字符：

格式字符	作用
<code>d</code>	以带符号的十进制形式输出整数（正数不输出符号）。
<code>o</code>	以8进制无符号形式输出整数（不输出前导符 <code>0</code> ）。
<code>x</code>	以16进制无符号形式输出整数（不输出前导符 <code>0x</code> ）。
<code>u</code>	以无符号十进制形式输出整数。
<code>c</code>	以无符号形式输出,只输出一个字符。
<code>s</code>	输出字符串。
<code>f</code>	以小数形式输出单、双精度数，隐含输出6位小数。
<code>e</code>	以标准指数形式输出单、双精度数，数字部分小数位数为6位。
<code>g</code>	选用 <code>%f</code> 或 <code>%e</code> 格式中输出宽度较短的一种格式，不输出无意义的 ³⁴ <code>0</code> 。

例： main()

```
{int i=234;  
char c=a;  
printf("%d,%5d,%c,%3c",i,i,c,c);  
}
```

指定输出宽度。数据宽度不够，
前面补空格，超过原样输出。

附加格式说明字符:

字 符	作 用
字母l	用于长整型整型，可加在格式符d、o、x、u前面。
m	数据最小宽度。
.n	对实数,表示输出n位小数;对字符串,表示截取的字符个数。
-	输出的数字或字符在域内向左靠。

例 :

```
main()
```

```
{
```

```
printf("%3s,%7.2s,%.4s,%-5.3s\n", "CHINA", "CHINA", "CHINA", "CHINA");  
}
```

输出为: CHINA, CHINA, CHINA, CHINA

②格式输入函数scanf()

功能： 输入若干个任意类型的数据。

格式： scanf("格式控制",参数1,参数2,参数3,……)

由格式说明和
普通字符构成

变量的地址或字
符串的首地址。

格式说明：由%后跟一个格式字符组成。中间可插入l、h、m、
*几个附加字符。

普通字符：照原样输入。

例如：

```
main()
{int a,b,c;
scanf("%d%d%d", &a,&b,&c);
printf("%d,%d,%d\n",a,b,c);
}
```

输入格式为： 5↙6↙7

若用： scanf("%d,%d,%d", &a,&b,&c);

则输入格式为： 5,6,7

若用： scanf("%d:%d:%d", &a,&b,&c);

则输入格式为： 5:6:7

格式字符：

格式字符	作用
d	用来输入十进制整数。
o	用来输入8进制整数。
x	用来输入16进制整数。
c	用来输入单个字符。
s	用来输入字符串，在输入时以非空白字符开始，以第一个空白字符结束。字符串以串结束标志'\0'作为其最后一个字符。
f	用来输入实数，可以用小数形式或指数形式输入。
e	与f作用相同，e与f可以互相替代。

例如： main()

```
{int a; char b; float c;  
scanf("%d%c%f", &a,&b,&c);  
printf("%d,%c,%f\n",a,b,c);  
}
```

对unsigned型数据
可以用d,o,x格式输入

③字符输出函数putchar()

功能：向终端输出一个字符。

格式：putchar (ch)

函数名

参数

④字符输入函数getchar()

功能：从输入设备输入一个字符。

格式：getchar ()

函数名

参数

例 #include "stdio.h"

main()

{char ch;

ch=getchar()+32;

putchar(ch);

}

4. 顺序程序设计

例：输入三角形的三边长，求三角形的面积。

(设输入的三边长a,b,c能构成三角形)

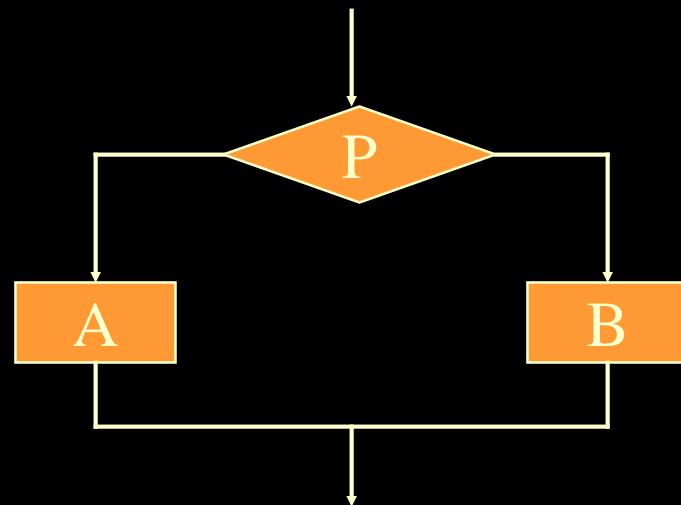
```
#include "math.h"
main()
{float a,b,c,s,area;
 s=1.0/2*(a+b+c);
 area=sqrt(s*(s-a)*(s-b)*(s-c));
 printf("a=%7.2f, b=%7.2f, c=%7.2f, s=%7.2f,\n",a,b,c,s);
 printf("area=%7.2f\n",area);
}
```

从键盘输入一个大写字母，要求改用小写字母输出。

求 $ax^2+bx+c=0$ 方程的根。a,b,c由键盘输入，设 $b^2-4ac>0$

第五章 选择结构程序设计

选择结构语句是指程序在运行中，能依据运行时某些变量或表达式的值，确定哪些程序段被执行以及哪些程序段不被执行。



5.1 关系运算符和关系表达式

关系运算符用于两个数值之间的比较运算。C语言提供6种关系运算符，它们是：

\langle 、 $\langle=$ 、 \rangle 、 $\rangle=$ 、 \equiv 、 $!\equiv$

优先级相同

优先级相同

关系运算符、算术运算符和赋值运算符的优先级为：



例如：	$c > a + b$	等效于	$c > (a + b)$
	$a > b != c$	等效于	$(a > b) != c$
	$a == b < c$	等效于	$a == (b < c)$
	$a = b > c$	等效于	$a = (b > c)$

关系表达式：用关系运算符将两个表达式连接起来的式子。

关系表达式运算结果为：“真”或“假”值。

C语言用1代表“真”值，用0代表“假”值。

例如：若 $a=3, b=2, c=1$

$f = a > b > c$ 则 f 的值为 0。

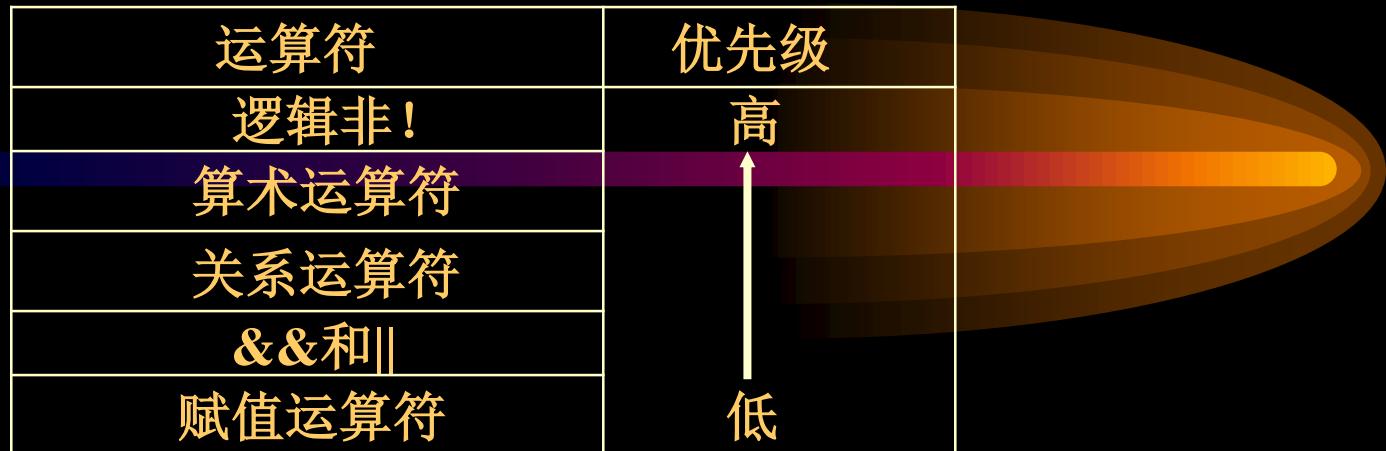
5.2 逻辑运算符和逻辑表达式

C 语言提供3种逻辑运算符：

逻辑运算符	结合率	优先级
<code>&&</code> 与	自左至右	中
<code> </code> 或	自左至右	低
<code>!</code> 非	自右至左	高

逻辑、关系、算术和赋值运算符的优先级为：

运算符	优先级
逻辑非!	高
算术运算符	
关系运算符	
&&和	
赋值运算符	低



例如： $(a>b)\&\&(x>y)$	可以写成	$a>b\&\&x>y$
$(a==b) (x==y)$	可以写成	$a==b x==y$
$(!a) (a>b)$	可以写成	$!a a>b$
$5>3\&\&2 8<4-!0$	的值为	1
‘c’&&‘d’	的值为	1

逻辑表达式：用逻辑运算符将关系表达式或逻辑量连接起来的式子。运算结果为：“真”或“假”值。系统在

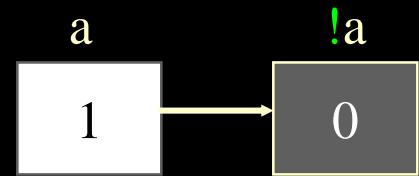
逻辑运算： 运算时以非0为“真”值，以0为“假”值。

a  b 与运算

	假	真
假	0	0
真	0	1

a  b 或运算

	假	真
假	0	1
真	1	1



例如：
 $4 \&\& 0 \mid\mid 2$ 的值为1
 $5 \&\& !0$ 的值为1

注意：

在逻辑表达式求解时，有可能出现某些逻辑运算符不被执行，但整个表达式的结果已经得到。

a&&b&&c

若a为**0**，则**b**和**c**不再判断。表达式结果为**0**，即“假”值。

a||b||c

若a为**1**，则**b**和**c**不再判断。表达式结果为**1**，即“真”值。

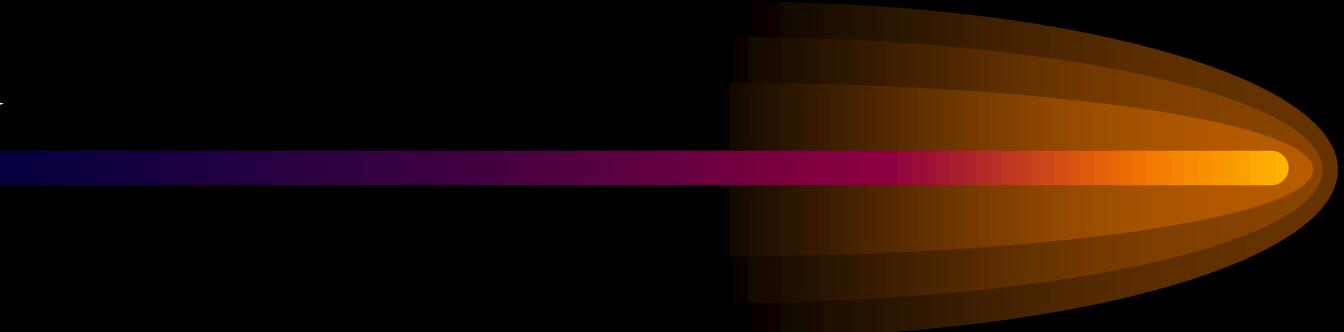
设 **a=1,b=2,c=3,m=1,n=1**

若 **m=(a>b)&&(n=c>4)**

则 **m**为**0**，**n**为**1**。



注意：



$x > 0 \& \& x < 10$

$0 < x < 10$

5.2 条件运算符和条件表达式

条件运算符： ? :

三目运算符，条件表达式的一般格式为：

表达式1?表达式2:表达式3

运算过程：表达式 1 的结果为真（非 0 ）时， 表达式 2 的计算结果作为条件表达式的值；否则， 取表达式 3 的计算结果为条件表达式的值。如： $a>b?a:b$

条件运算符的优先级低于逻辑、关系、算术运算符高于赋值运算符。

如： $a>b?a:b+1$ 相当于 $(a>b)?a:(b+1)$

条件运算符的结合率为：“自右至左”。

如： $a>b?a:c>d?c:d$ 相当于 $a>b?a:(c>d?c:d)$

注意： 条件表达式中的表达式1、表达式2、表达式3可以是不同的类型。

如： main()

```
{float p;  
char x,y;  
scanf("%c%c",&x,&y);  
p=x>y?1:1.5;  
printf("\n%f",p);  
}
```

5.3 if语句

选择结构语句是指程序在运行中，能依据运行时某些变量或表达式的值，确定哪些程序段被执行以及哪些程序段不被执行。

① 单分支语句：if语句

作用：当条件为真时执行语句。

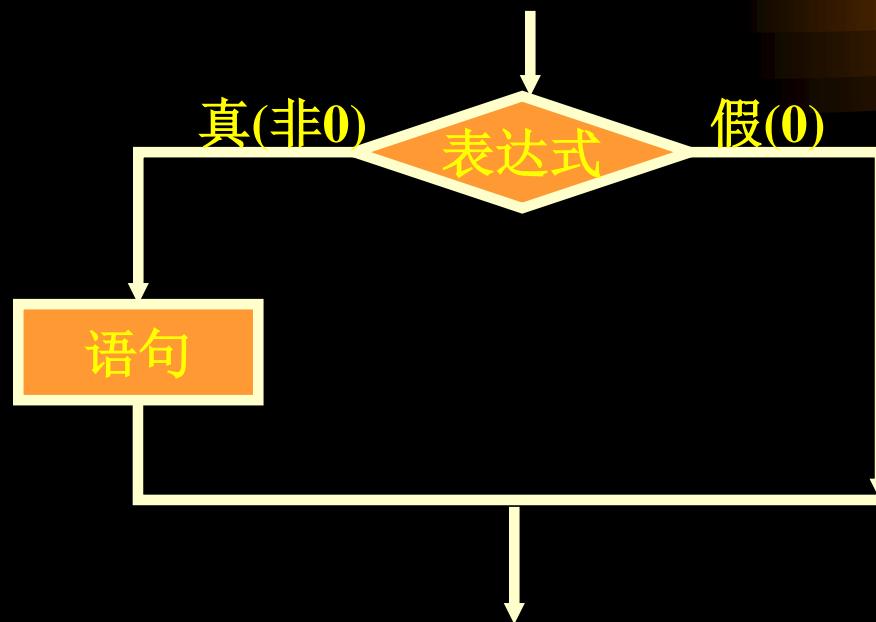
语句格式：

if (表达式) 语句

括号“()”不能缺省

若有多个语句须用
复合语句{}。

执行过程:



例：任意输入一个整数，求其绝对值并输出。

分析： 输入一个整数，此数可以是正数、零或负数，若为负数则将其转化为正数，若为正数或零什么也不做，最后输出结果。

```
main()
{
    int x;
    printf("please input a integer: ");
    scanf("%d",&x);
    if(x<0)  x= -x;
    printf("%d\n",x);
}
```

②双分支语句：if...else 语句

作用：根据给定的条件，选择两段程序之一执行。

语句格式：

`if (表达式)`

语句1

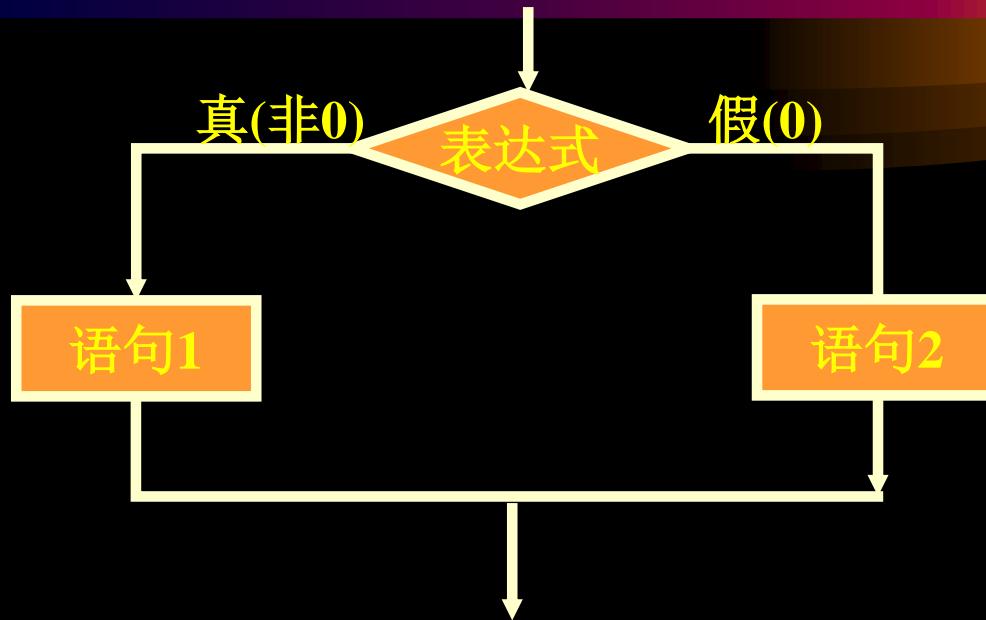
括号“()”不能缺省

`else`

语句2

若有多个语句须用
复合语句 {} 。

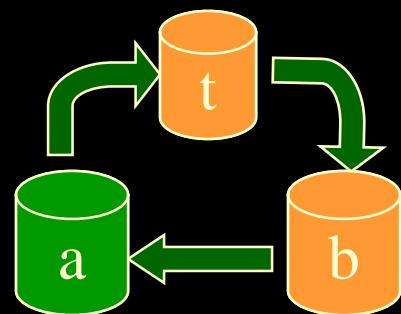
执行过程:



例：输入两个实数，按代数值由小到大次序输出这两个数。

```
main()
```

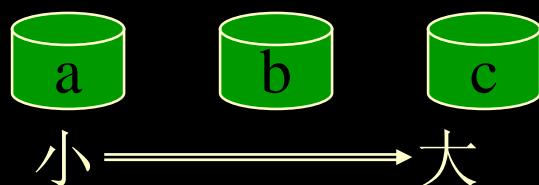
```
{float a,b,t;  
scanf("%f,%f",&a,&b);  
if (a>b) {t=a; a=b; b=t;}  
printf("%5.2f,%5.2f",a,b);  
}
```



例： 输入三个实数，按代数值由小到大次序输出这三个数。

main()

```
{float a,b,c,t;  
scanf("%f,%f,%f",&a,&b,&c);  
if (a>b) {t=a;a=b;b=t;}  
if (a>c) {t=a;a=c;b=t;}  
if (b>c) {t=b;b=c;c=t;}  
printf("%5.2f,%5.2f,%5.2f",a,b,c);  
}
```



③if语句的嵌套：

在if语句的语句1或语句2部分，若使用if语句，称为if语句的嵌套。

如： if (表达式1)

 if (表达式2)

 语句1

 else

 语句2

else

 if (表达式3)

 语句3

 else

 语句4



例： 有一函数如下， 编一程序， 输入一个x值， 输出y值。

$$y = \begin{cases} -1 & (x < 0) \\ 0 & (x = 0) \\ 1 & (x > 0) \end{cases}$$

```
main()
{int x,y;
scanf("%d",&x);
if (x<0) y=-1;
else if (x==0) y=0;
    else y=1;
printf("x=%d,y=%d\n",x,y);
}
```

也可将if语句改为：

```
if (x>=0)
    if (x>0) y=1;
    else y=0;
else y=-1;
```

 注意： if语句嵌套使用时， else与最近的if匹配。

思

考：

下列程序运行后,x的值是多少:

a=b=c=0;

if(!a) x--;else if (b) ; if (c) x=3;else x=4;

结果:



x=4

一般为整型或
字符型表达式

⑤多分支语句： switch 语句

格式：

switch(表达式)

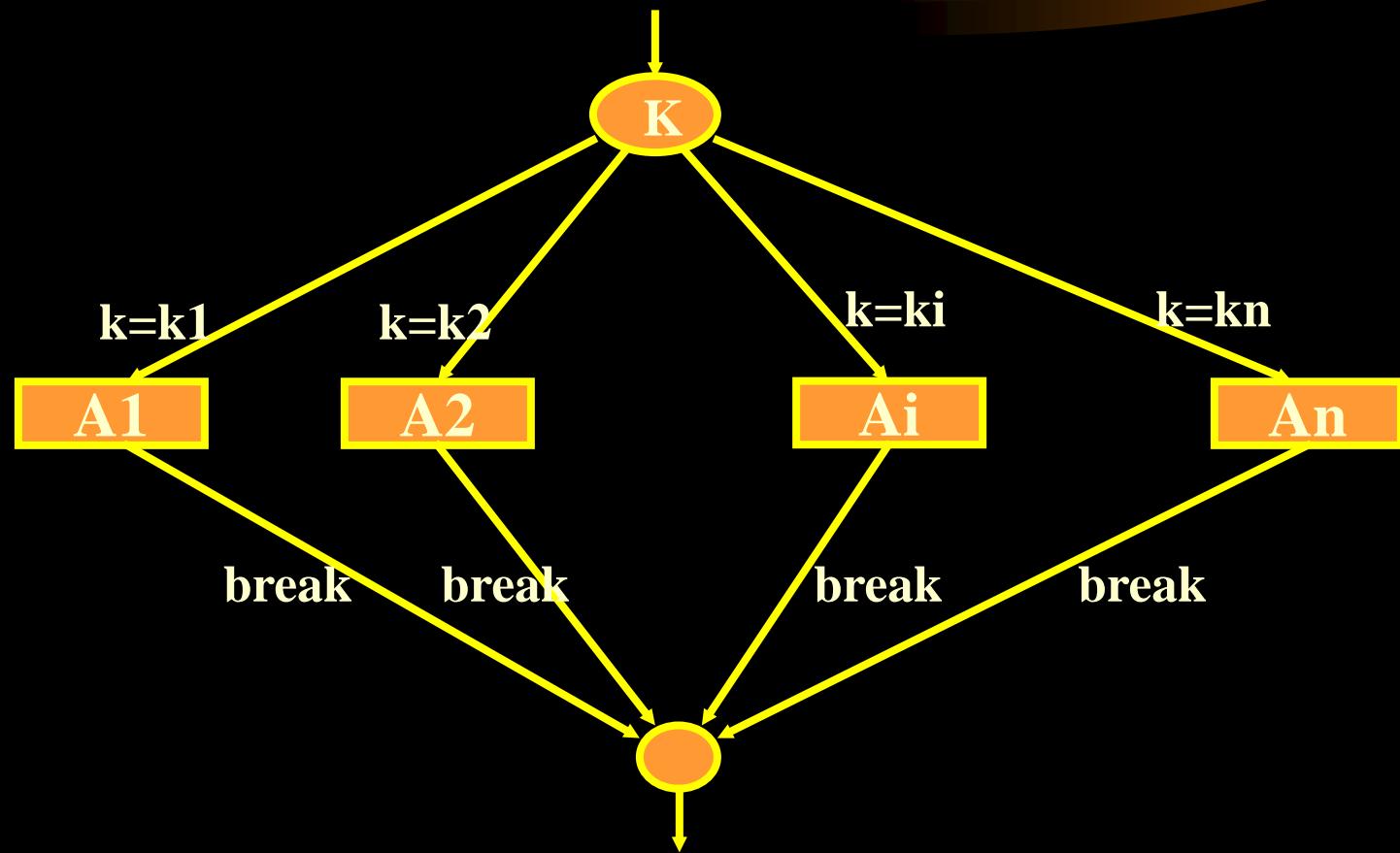
```
{case 常量表达式1:语句序列1  
    case 常量表达式2:语句序列2  
    case 常量表达式3:语句序列3  
        :  
    case 常量表达式n:语句序列n  
default: 语句序列n+1  
}
```

执行过程：

当表达式的值等于常量表达式i的值时，
从语句序列i开始执行到语句序列n+1为
止。

若表达式的值不等于任何一个常量表达
式的值，
则只执行default后面的语句。

一般在每个语句序列之后加一个break语句，这样在执行语句序列*i*之后，使流程跳出switch结构，实现多分支选择。



例：编写一个能进行两个操作数加减乘除四则运算的计算器模拟程序。

输入： 两个操作数和运算符
计算： 根据运算符确定运算
输出： 运算结果



‘+’ 进行加运算。
‘-’ 进行减运算。
‘*’ 进行乘运算。
‘/’ 进行除运算。

```
main()
{char op; float x,y;
printf("input a arithmetic expression:");
scanf("%f%c%f",&x,&op,&y);
switch (op)
{ case '+': printf("=%f\n",x+y); break;
  case '-': printf("=%f\n",x-y); break;
  case '*': printf("=%f\n",x*y); break;
  case '/': if (y!=0.0)
               printf("=%f\n",x/y);
            else
               printf('Divisor is zero\n');
            break;
  default: printf('Illegal operator\n');
}
}
```

例：分析下列程序的输出结果。

```
main()
{int i=1,j=0,m=1,n=2;
 switch(i++)
 {case 1: m++;n++;
 case 2: switch(++j)
 {case 1: m++;
 case 2: n++;
 }
 case 3: m++;n++;break;
 case 4: m++;n++;
 }
 printf("m=%d,n=%d\n",m,n);
}
```

switch语句
的嵌套

执行结果为： m=4,n=5

例：求 $ax^2+bx+c=0$ 方程的解。

输入： a,b,c

计算：求解方程

输出：两个实根或两个复根



$a=0$,不是二次方程。

$b^2-4ac=0$,有两个相等实根。

$b^2-4ac>0$,有两个不等实根。

$b^2-4ac<0$,有两个共轭复根。

```
#include "math.h"
main()
{float a,b,c,disc,x1,x2,realpart,imagpart;
 scanf("%f,%f,%f",&a,&b,&c);
if (fabs(a)<=1e-6)
    printf("is not quadratic");
else disc=b*b-4*a*c;
if (fabs(disc)<=1e-6)
    printf("has two equal roots:%8.4f\n",-b/(2*a));
else
    if (disc>1e-6)
        {x1=(-b+sqrt(disc))/(2*a);
         x2=(-b-sqrt(disc))/(2*a);
         printf(" has distinct real roots:%8.4f and %8.4f\n",x1,x2);
        }
    else
        {realpart=-b/(2*a);
         imagpart=sqrt(-disc)/(2*a);
         printf("has complex roots:\n");
         printf("%8.4f+%8.4fi\n",realpart,imagpart);
         printf("%8.4f-%8.4fi\n",realpart,imagpart); }
}
```

例：给出一百分制成绩，要求输出成绩等级。

其中：

90分以上为‘A’，80~89分为‘B’，

70~79分为‘C’，60~69分为‘D’，

60分以下为‘E’。

```
main()
{
    int x;
    scanf("%d",&x);
    x=x/10;
    if (x>10||x<0)
        printf("成绩输入有错，请重新输入\n");
    else
        switch (x)
        {
            case 10:
            case 9: printf("A\n");break;
            case 8: printf("B\n");break;
            case 7: printf("C\n");break;
            case 6: printf("D\n");break;
            default: printf("E\n");
        }
}
```

作业:

1.阅读程序,写出运行结果.

```
main()
{int x=100,a=10,b=20,ok1=5,ok2=0;
 if(a<b)  if (b!=15) if (!ok1) x=1;
 else if (ok2) x=10;
 else  x=-1;
 printf(“%d\n”,x);
}
```

P111 5.3 5.4 5.5 5.7

第六章 循环控制

循环控制结构在程序中是指对某段程序或某条语句根据条件重复执行。

C 语言提供了**while**、**do-while**和**for**三种循环控制结构。

① while语句

while语句是支持“当型”循环控制结构的语句。

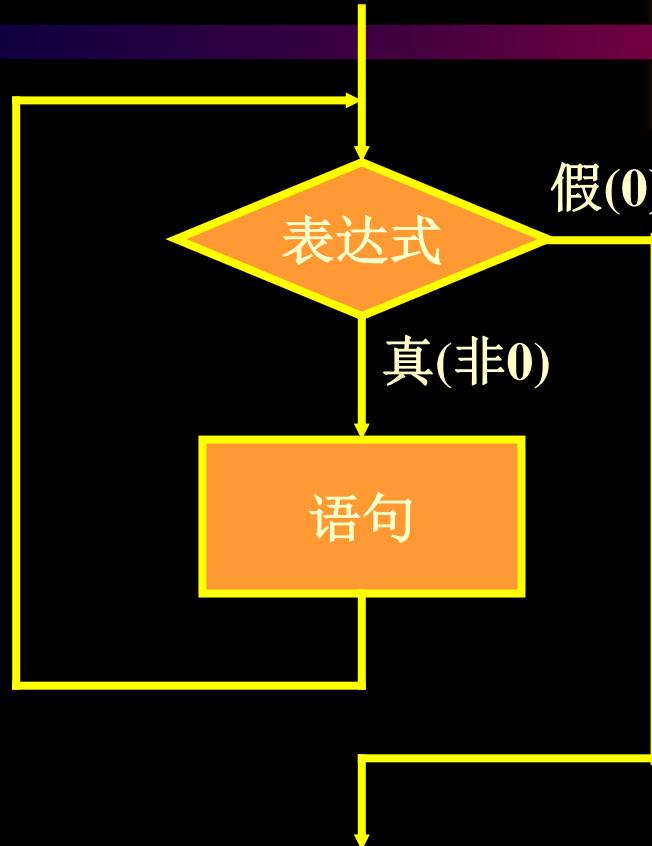
格式：

while (表达式)

语句

若有多个语句须
用复合语句 {}

执行过程：



例 求 $\sum_{n=1}^{100} n$

输入：无

计算： $1+2+3+\dots+100$

输出：计算的和

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

练习：计算

$1^2+2^2+3^2+\dots+10^2$

② do_while语句

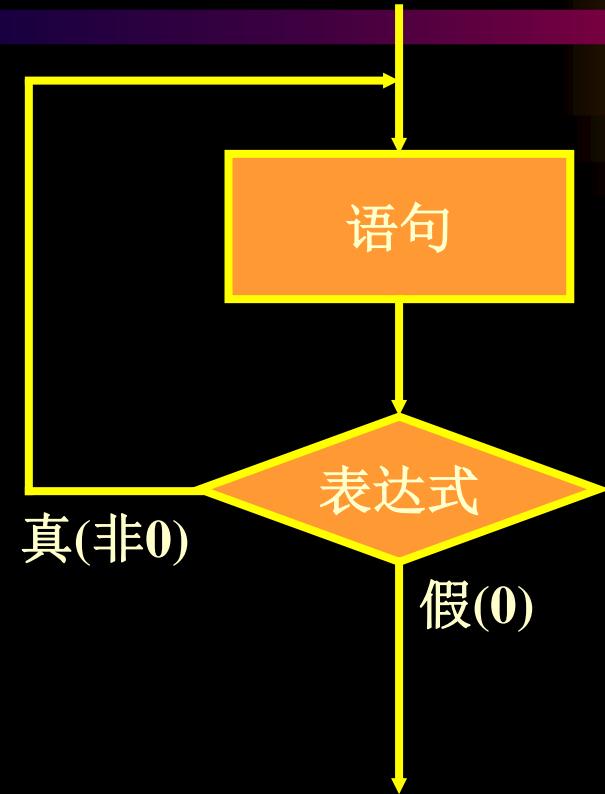
do_while语句是支持“直到型”循环控制结构的语句。

一般格式为：

```
do  
    语句  
    while (表达式);
```

若有多个语句须
用复合语句 {}

执行过程:



例 求 $\sum_{n=1}^{100} n$ 。

输入: 无

计算: $1+2+3+\dots+100$

输出: 计算的和

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

while和do_while语句的比较:

```
main()
{int sum=0;i;
scanf("%d",&i);
while (i<=10)
{sum=sum+i;
 i++;
}
printf("sum=%d\n",sum);
}
```

```
main()
{int sum=0;i;
scanf("%d",&i);
do {sum=sum+i;
 i++;
}
while (i<=10)
printf("sum=%d\n",sum);
}
```

当*i*<=10时， 两程序的结果相同。

当*i*>10时， 两程序的结果不同。

首次条件为真， 两者等价； 首次条件为假， do~while执行一次。

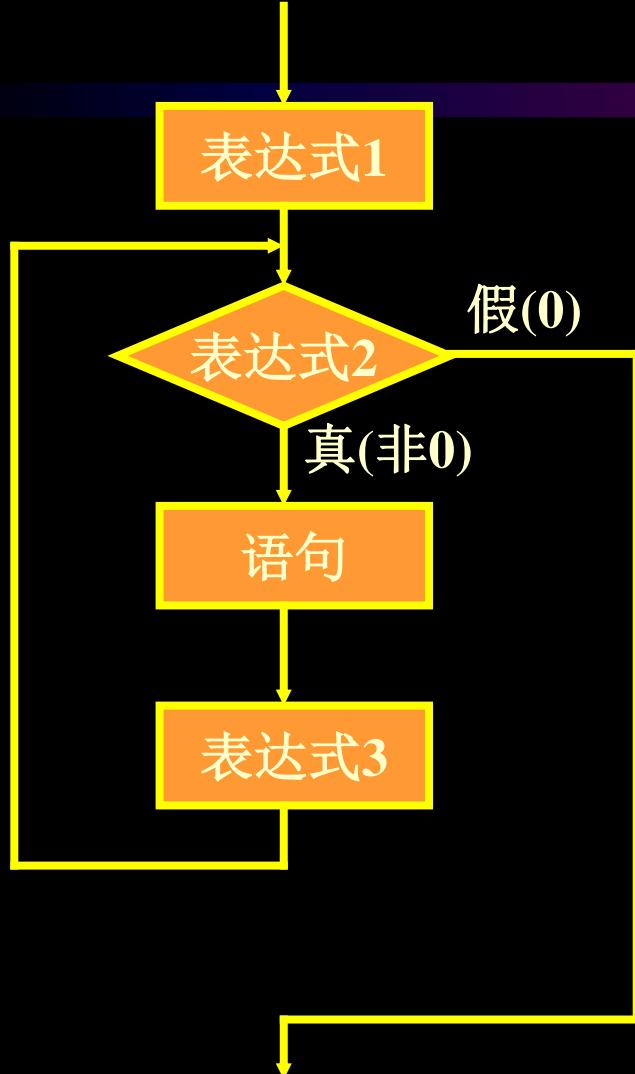
③ for 语句

for语句是一种使用比while语句更加灵活的循环控制语句。

一般格式为：

for (表达式1;表达式2;表达式3) 语句

执行过程:



- (1)求解表达式 1；
- (2)求解表达式2，
若为真(非0)值，则执行语句，
再转到求解表达式2。
- 若为假(0)值，则结束执行。

例如: **main()**

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

和下面程序比较:

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

练习: 计算

$$1^2+2^2+3^2+\dots+10^2$$

for语句三个表达式的使用说明：

- ①三个表达式都可缺省,但分号(;)不能省略。
- ②若表达式1缺省, 则从表达式2开始执行。
- ③若表达式2缺省, 则认为表达式2始终为真, 循环无终止进行。

省略表达式1

例如：

```
for ( ;i<=100;i++) sum=sum+i;
```

省略表达式2

```
for (i=1; ;i++) sum=sum+i;
```

```
for (i=1;i<=100;) {sum=sum+i;i++;}
```

省略表达式3

省略表达式1,3

```
for ( ;i<=100;) {sum=sum+i;i++;}
```

for (; ;) 语句

循环无终止进行

④三个表达式可以是C语言的任意型表达式。

例： `for (sum=0,i=1;i<=100;i++) sum=sum+i;`

↑
逗号表达式

`for (i=0,j=100 ;i<=j;i++,j--) k =i+j;`

↑
逗号表达式

`for (i=0 ;(c=getchar())!='\n';i+=c) ;`

空语句

例：用for语句编写一个计算 $1+3+5+\dots+(2*i-1)$ 的程序，其中
 $i=1,2,3,\dots,100$ 。

输入：无

计算： $1+3+5+\dots+(2*i-1)$

输出：计算的和

```
main()
{int sum=0,i;
 for (i=1;i<=100;i++)
     sum+=2*i-1;
 printf('sum=%d\n',sum);
}
```

练习：任给n,计算

$1!+2!+3!+\dots+n!$

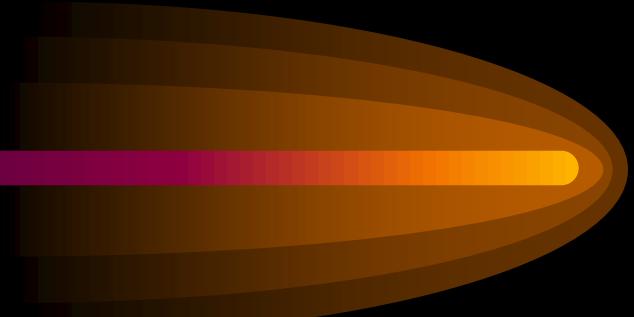
三种循环结构

用于反复的进行某个操作，直到条件不成立时为止。

(1) **while** (表达式) {语句系列}

(2) **do** {语句系列} **while** (表达式);

(3) **for** (表达式1;表达式2;表达式3) {语句系列}



三种循环功能完全等价：

(1),(2)两种主要用于循环次数不定。

而(3)主要用于循环次数确定的情况。

使用循环语句要注意和数组的结合：

关键在于找出数组下标和循环变量之间
的关系。

例：任给n，计算 $1!+2!+3!+\cdots+n!$ 。

输入：无

计算： $1!+2!+3!+\cdots+n!$

输出：计算的和

```
main()
{
    long sum,t;
    int i,n;
    printf("input n=");
    scanf("%d",&n);
    t=1;sum=0;
    for (i=2;i<=n;i++)
    {
        t=t*i;
        sum=sum+t;
    }
    printf("sum=%ld\n",sum);
}
```

7. 循环语句的嵌套

一个循环语句内又包含另一个完整的循环语句，称为循环语句的嵌套。内嵌的循环语句一般称为内循环，包含内循环的循环语句称为外循环。内循环再嵌套内层循环，就够成了多重循环。

例：

求2和32766之间的素数。

```
main()
{int i,k;
for (k=2;k<=32766;k++)
{for (i=2;i<k;i++)
    if (k%i==0)
        break;
    if (i==k)
        printf("%d,",k);
}
}
```

如何对其进行改进?



分析下列程序的输出结果：

```
main()
{int i=1,a=0;
for ( ;a<=5;i++)
{do
    {i++;a++;}
    while (i<3);
    i++;
}
printf("a=%d,i=%d\n",a,i);
}
```

结果： a=6,i=17

多重循环程序设计时，应注意以下几点：

三种循环不仅可以自身嵌套，而切可以互相嵌套。

嵌套时，要在一个循环体内包含另一个完整的循环结构。



运行时，应注意内嵌的语句执行过程。

```
for (i=1;i<=n;i++)  
{ j=1;  
    while (j<=m )  
    { printf("a");j++; }  
}
```

该语句执行多少次？

8. Break语句和continue语句

①break语句

break语句的功能是：

在switch语句中使流程跳出switch结构。

在循环语句中使流程跳出当前循环。

例：

编程将从键盘上输入的若干个正整数求和，遇到负数则终止程序，并且输入的数不超过10个。

```
#define M 10  
  
main()  
{int i,x,sum;  
    sum=0;  
    for (i=1;i<=M;i++)  
    {printf("\ninput x=");  
     scanf("%d",&x);  
     if (x<0) break;  
     sum+=x;  
    }  
    printf("%d",sum);  
}
```

②continue语句

continue语句的功能

是在循环语句中使本次循环结束，即跳过循环体中下面尚未执行的语句，接着进行下次是否执行循环的判断。

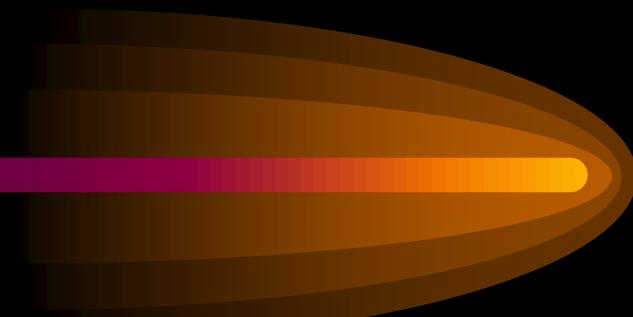
例：

编程把100~200之间的不能被3整除的数输出。

```
main()
{int i;
 for (i=100;i<=200;i++)
 {  if (i%3=0)
        continue;
        printf("%d,",i);
    }
}
```

```
main()
{int a,b;
for (a=1;a<=9;a++)
{for (b=1;b<=a;b++)
printf("%d ",a*b);
printf("\n");
}
}
```

结果?



1									
2	4								
3	6	9							
4	8	12	16						
5	10	15	20	25					
6	12	18	24	30	36				
7	14	21	28	35	42	49			
8	16	24	32	40	48	56	64		
9	18	27	36	45	54	64	72	81	

作业:

1. 编程分别输出下列图形:

*

**

**

*

2.P129 6.3 6.5 6.6 6.8

第七章 数组

数组是C语言提供的一种构造类型数据，由基本数据类型按一定规则组成。

数组是有序数据的集合。数组中的每一个元素都属于同一个数据类型。用一个统一的数组名和下标来唯一地确定数组中的元素。

1. 一维数组的定义和引用

① 一维数组的定义：

只能包含常量和符号常量。
表示数组包含的元素个数。
既数组的长度。

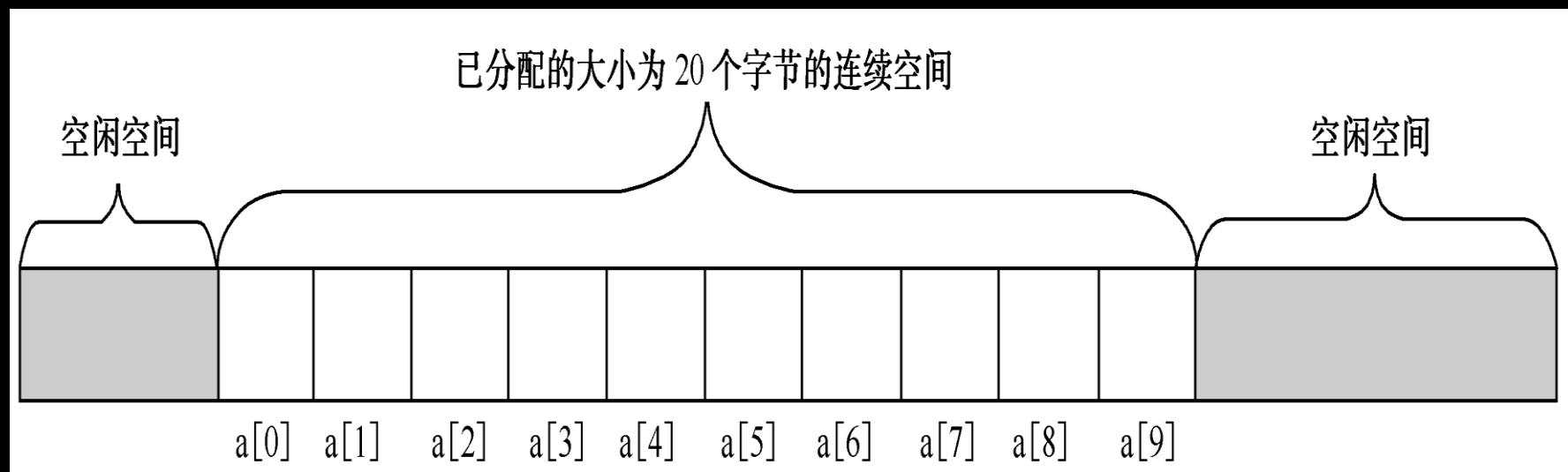
类型说明符 数组名[常量表达式];

如： int char float等。

遵循标识符
命名规则

如： int a[10];

定义一个一维数组a，它包含了10个具有整型数据类型的元素： a[0],a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9]



②一维数组的引用

C语言规定只能逐个引用数组元素，不能一次引用整个数组。而且必须先定义数组，然后才能使用数组元素。

数组元素的表示形式：

下标可以是表达式

数组名[下标]

例 main()

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

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

③一维数组的初始化

初始化就是在定义数组的同时，给数组元素赋初始值。

C语言规定：只有静态存储和外部存储数组才能初始化。

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

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

```
int b[10]={0,1,2,3,4};      (后5个元素赋值0。)
```

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

```
int c[10]={0,0,0,0,0};
```

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

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

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

可以不指定数组长度， 系统根据后面花括号里的数据决定数组的长度。

d[0] d[1] d[2] d[3] d[4]

1	2	3	4	5
---	---	---	---	---

3. 一维数组程序举例

例1：

在数组a中存入10个整数，将数组中的最大数存入a[0]位置。

例2：

用起泡法对6个数排序（由小到大）。

起泡法：即是相临两个数比较，将小的调到前头。

第一趟						最大数以 得到。
第1次	第2次	第3次	第4次	第5次	结果	
9 ↘	8	8	8	8	8	
8 ↘	9 ↘	5	5	5	5	
5	5 ↘	9 ↘	4	4	4	
4	4	4 ↘	9 ↘	2	2	
2	2	2	2 ↘	9 ↘	0	
0	0	0	0	0 ↘	9	

第二趟						5 4 2 0 8 9
第1次	第2次	第3次	第4次	结果		
8 ↘	5	5	5	5		
5 ↘	8 ↘	4	4	4		
4	4 ↘	8 ↘	2	2		
2	2	2 ↘	8 ↘	0		
0	0	0	0 ↘	8		
				结果		

第三趟	5 ↘	4	4	4	
	4 ↘	5 ↘	2	2	
	2	2 ↘	5 ↘	0	
	0	0	0 ↘	5	结果
	第1次	第2次	第3次		结果

第四趟	4 ↘	4	4	
	2 ↘	2 ↘	0	
	0	0 ↘	2	
	第1次	第2次	结果	

第五趟	2 ↘	0	
	0 ↘	2	
	第1次	结果	

最后
结果

0
2
4
5
8
9

即：如果有n个数，则要进行n-1趟比较。在第j趟比较中
要进行n-j次两两比较。

```
main()
{int a[11],i,j,t;
 for (i=1;i<11;i++)
    scanf("%d",&a[i]);
 printf("\n");
 for (j=1;j<=9;j++)
    for (i=1;i<=10-j;i++)
        if (a[i]>a[i+1])
            {t=a[i];a[i]=a[i+1];a[i+1]=t;}
    for (i=1;i<11;i++)
        printf("%d ",a[i]);
}
```

编写程序完成如下功能：

- (1) 从键盘输入字符串，统计各个字母出现的次数。
- (2) 从键盘输入20个整数，求其最大值、最小值、平均值。
- (3) 用程序实现以下图像的显示。

*

* *
* *
* *

- (4) 从键盘输入20个整数，转置并输出值。
- (5) 求 $2+22+222+2222+22222$ 。

4. 二维数组

① 二维数组的定义

类型说明符 数组名[常量表达式][常量表达式];

例：

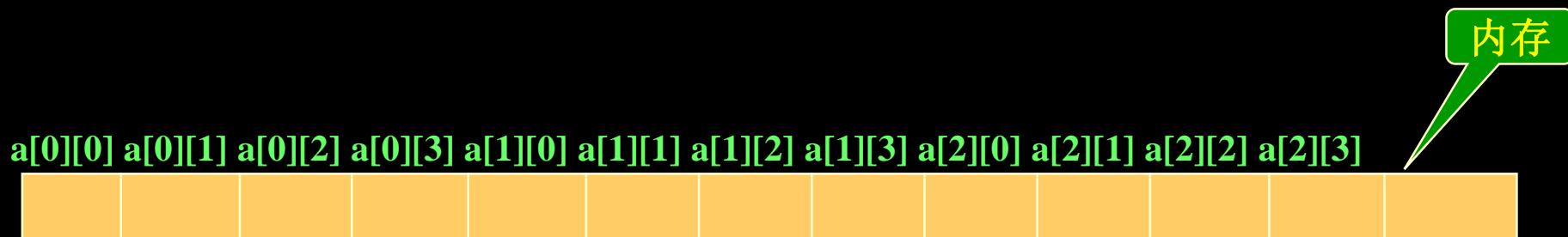
float d[3][4]; /*定义d为3行4列的实型数组*/

int b[5][2]; /*定义b为5行2列的整型数组*/

char c[2][3]; /*定义c为2行3列的字符数组*/

例如，“float d[3][4];”相当于一次定义了12个float型变量。
依次表示为d[0][0]、d[0][1]、d[0][2]、d[0][3]、d[1][0]、
d[1][1]、d[1][2]、d[1][3]、d[2][0]、d[2][1]、d[2][2]和d[2][3]

二维数组元素在内存中的排放顺序是：按行存放。
如数组a[3][4]的存放为



②二维数组元素的引用

与一维数组一样，只能逐个引用数组元素，不能一次引用整个数组。而且必须先定义二维数组，然后才能使用数组元素。

引用方式：

下标可以是表达式

数组名[行标][列标]

如： $a[2][1]=a[0][0]+a[0][1];$

注意：在引用数组元素时，下标不要越界。

③二维数组的初始化

对应元素初始化

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

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

	1	2	3	4	5	6	7	8	9	10	11	12	
--	---	---	---	---	---	---	---	---	---	----	----	----	--

内存

分行赋初值：

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

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

	1	2	3	4	5	6	7	8	9	10	11	12	
--	---	---	---	---	---	---	---	---	---	----	----	----	--

注意：

C 语言规定一维长度可省略，但二维长度不能省略。
系统会根据顺序赋初值或分行赋初值自动分配存储空间。

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

系统根据顺序赋初值自动分配3行4列存储空间。

```
int b[][]={{0,3},{},{}},{{0,12}};
```

系统根据分行赋初值自动分配3行2列存储空间。

5. 二维数组程序举例

例：

将一个二维数组行和列元素互换，存到另一个二维数组中。

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \longrightarrow b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
main()
{int a[2][3]={{1,2,3},{4,5,6}},b[3][2],i,j;
 printf("array a:\n");
for (i=0;i<=1;i++)
    for (j=0;j<=2;j++)    b[j][i]=a[i][j];
for (i=0;i<=2;i++)
{
    for (j=0;j<=1;j++)
        printf("%5d",b[i][j]);
    printf("\n");
}
```



例：

从键盘输入一个5行5列的矩阵，求其最大元素、最小元素、矩阵主对角线元素的平均值，并显示该矩阵。

```
main()
{
    int a[5][5],i,j,b[4]={0},sum=0;
    for(i=0;i<5;i++)
        for(j=0;j<5;j++) scanf("%d",&a[i][j]);
    for(i=0;i<5;i++)
        for(j=0;j<5;j++)
    {
        if(a[i][j]>a[b[0]][b[1]]) {b[0]=i;b[1]=j;}
        if(a[i][j]<a[b[2]][b[3]]) {b[2]=i;b[3]=j;}
        if(i==j) sum+=a[i][j];
    }
    printf("\n the max is %d",a[b[0]][b[1]]);
    printf("\n the min is %d",a[b[2]][b[3]]);
    printf("\n the average is %.2f\n",sum/5.0);
    for(i=0;i<5;i++)
    {
        printf("\n");
        for(j=0;j<5;j++) printf("%6d",a[i][j]);
    }
}
```

6. 字符串与字符数组

字符串：

是指由若干个字符（可以为0个）所构成的字符序列。

字符数组：

是指数组元素是char型的一种数组。

- 通过字符串表示文字；通过字符数组来存放字符串。
- 一维字符数组用于存放一个字符串；
- 多维字符数组用于存放多个字符串。

几点说明：

- ① 系统在存放字符串时，会自动在字符串最后一个字符的后边加上字符串结束标志 ‘\0’（要占一个字节的存储空间）。
- ② 和其他数组一样，字符数组的越界判断要由用户自己来完成。

①字符数组初始化

通过字符初始化

例：

```
char c[4]={ 'a', 'b', 'c', 'd'};
```

c[0]	c[1]	c[2]	c[3]
a	b	c	d

通过字符串初始化

例：

char s [10] = "abcd"; 或者 char s [10] = { "abcd" };



注意：

- (1) 在定义数组时，可以用省略第一维大小的方法来初始化一个数组的各个元素。
- (2) 二维字符数组的初始化可以用初始值表的方法，也可以用字符串常量的方法。
- (3) 字符数组在使用时，不能将一个字符串直接赋给一个字符数组名，只能对字符数组的元素逐个赋值。

②字符串处理函数介绍

(1)strcpy函数。

其格式为： strcpy(字符数组名,"字符串");

作用是将字符串复制到字符数组中且自动加 '\0'。

(2)strcmp函数。

其格式为： strcmp (字符串1,字符串2)；

作用是比较两个字符串。

若两个字符串相等，返回值为0；

若字符串1大于字符串2，则返回值为一正整数；

若字符串1小于字符串2，则返回值为一负整数。

(3)strcat函数。

其格式为： **strcat(字符数组名1, 字符数组名2);**

作用是将第二个字符串的内容连接到第一个字符串的后边。

(4)strlen函数。

其格式为： **strlen (字符串)；**

作用是计算以 ‘\0’ 结尾的字符串长度，并且返回字符串的长度，结尾字符 ‘\0’ 不计算在内。

注意：

如果在程序中要使用这四个函数，需要包含头文件 “**string.h**”。

③字符数组的输入输出：

(1) 通过scanf()函数和printf()函数

逐个字符输入输出：用格式“%c”。

例： main()

```
{char c[10];
int i;
for (i=0;i<10;i++)
    scanf("%c",&c[i]);
for (i=0;i<10;i++)
    printf("%c",c[i]);
}
```

数组元素地址

数组元素

将整个字符串一次输入输出：用格式“%s”。

例： main()

```
{char c[10];
scanf("%s",c);
printf("%s",c);
}
```

数组名

输出时遇到第一个
“\0”字符结束输出。
输入字符串时,不要
超过数组长度。

(2)通过gets()函数和puts()函数

gets(字符数组)

从终端输入一个字符串到字符数组，并且得到一个函数值，该函数值是字符数组的起始地址。

如： gets(str);

puts(字符数组)

将一个字符串输出到终端。

如： char str[]='China\nXi'an';

puts(str);

7. 字符数组应用举例

例：

编写程序，实现字符串的复制。

```
#include "stdio.h"
main()
{
    char str1[100],str2[100];
    int i=0;
    gets(str1);
    while(str1[i]!='\0')
        {str2[i]=str1[i];  i++;  }
    str2[i]='\0';
    puts(str2);
}
```

```
#include "stdio.h"
main()
{
    char str1[100],str2[100];
    int i=0, j=0;
    gets(str1);
    while(str2[i]==str1[j]);
        puts(str2);
    }
```



- (1) 输入一行字符，统计其中有多少个单词，单词之间用空格分割开。
- (2) 输入10个字符串，要求找出其中最大者。
- (3) 输入两个字符串，连接并输出。



- 1、编写程序完成如下功能：分别将字符串a和字符串b中的字符倒置。然后按交叉的顺序将两个字符数组合并到字符数组c中，过长的部分直接连接在c的尾部。（例如，若字符串a的内容为“abcdefg”，字符串b的内容为“1990”，则结果为：“h0g9f9e1dcba”）
- 2、键入一串字符（换行符结束），用循环语句将其中的小写英文字母互换后输出。
- 3、设有语句“int a[3][4];”，先为数组输满数据，再将该数组周边的元素输出（元素输出次序不限）。

第八章 函数

问题在处理过程中，往往需要进行大量的重复性工作。如果每次这些工作均要编写一段程序，不但麻烦，而且冗余，更不会体现到程序设计的乐趣。
对于这样的问题，该如何解决？

1. 概述

- 一个C程序可由一个主函数和若干个非主函数构成。
- 主函数调用其它函数，其它函数也可以互相调用。
- 同一个函数可以被一个或多个函数调用任意多次。
- 程序的执行从**main()**函数开始。
- 在**main()**函数中结束整个程序的运行。

①引入函数的目的

简化程序设计的复杂程度。

可以将一个任务划分为若干个功能模块，每一个功能模块用一个函数来完成。功能相同的模块只编一个函数。常用的功能模块编写成函数，放在函数库中供公共选用，如：**printf()**、**gets()**等函数。

例

printstar()

```
{ printf("*****\n");  
}
```

printmessage()

```
{ printf("      How do you do!\n");  
}
```

main()

```
{printstar();  
printmessage();  
printstar();  
}
```

运行结果为：

```
*****
```

How do you do!

```
*****
```

②函数的分类

(1)从用户的角度看:

函数分为 { 标准函数(库函数)。由系统提供
 用户自定义函数。 用户自己编制

(2)从函数的形式看:

函数分为 { 无参函数
 有参函数

```
#include "stdio.h"  
main()  
{char c;  
c=getchar();  
printf("%c",c);  
}
```

无参函数
标准函数
自定义函数
有参函数

③函数的定义

(1) 无参函数的定义形式

```
类型标识符 函数名( )  
{ 说明部分  
语句  
}
```

```
long s()  
{int n,i=1;long m=1;  
scanf("%d",&n);  
while(i<=n){m*=i;i++;}  
return (m);  
}  
main()  
{  
printf("\n%d",s());  
}
```

(2)有参函数的定义形式

类型标识符 函数名(形式参数表列)

{ 说明部分

语句 }

省略类型标识符
的函数，一律按
整型处理。

形式参数说明

```
max(int x,int y)
{
    int z;
    z=x>y?x:y;
    return(z);
}

main()
{
    int a,b,c;
    scanf("%d,%d",&a,&b);
    c=max(a,b);
    printf("the max is %d",c);
}
```

④自定义函数的说明

C语言规定：

除主函数外，自定义函数都应遵循“先说明，后使用”的规则。

(1) 函数的两种说明形式

- 经典方式。
- 函数类型 函数名();
- ANSI规定方式。

函数类型 函数名(数据类型 形参1, 数据类型 形参2,.....,数据类型 形参n);

(2) 函数说明的位置

C语言规定：函数类型为int型和被调函数的定义出现在主调函数之前时，可省略函数说明外，而其他情况均要求对被调函数进行说明。

函数说明的位置可以视情况而定。

既可以放在所有函数之外，也可以放在主调函数体内的说明部分。

如果某函数说明放在所有函数之外，则在该函数说明位置后面的所有函数均可调用该函数。

2. 函数参数和函数的值

① 形式参数和实际参数

形式参数：定义函数时所用的参数，用于接收从调用函数传递来的数据。
在未被调用时，不占内存中的存储单元，只有调用时才被分配内存单元。
实际参数：在函数调用时，所传递的参数。

形式参数

```
max(int x,int y)
{retnrn (x>y?x:y);}
```

```
main()
```

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

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

实际参数

```
c=max(a,b);
```

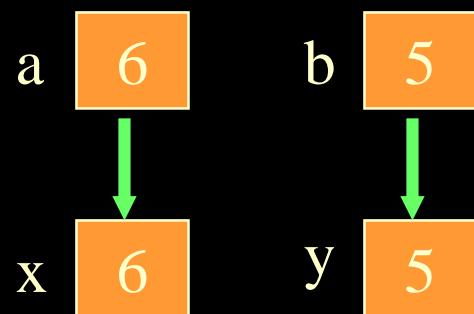
```
printf("max is %d",c);
```

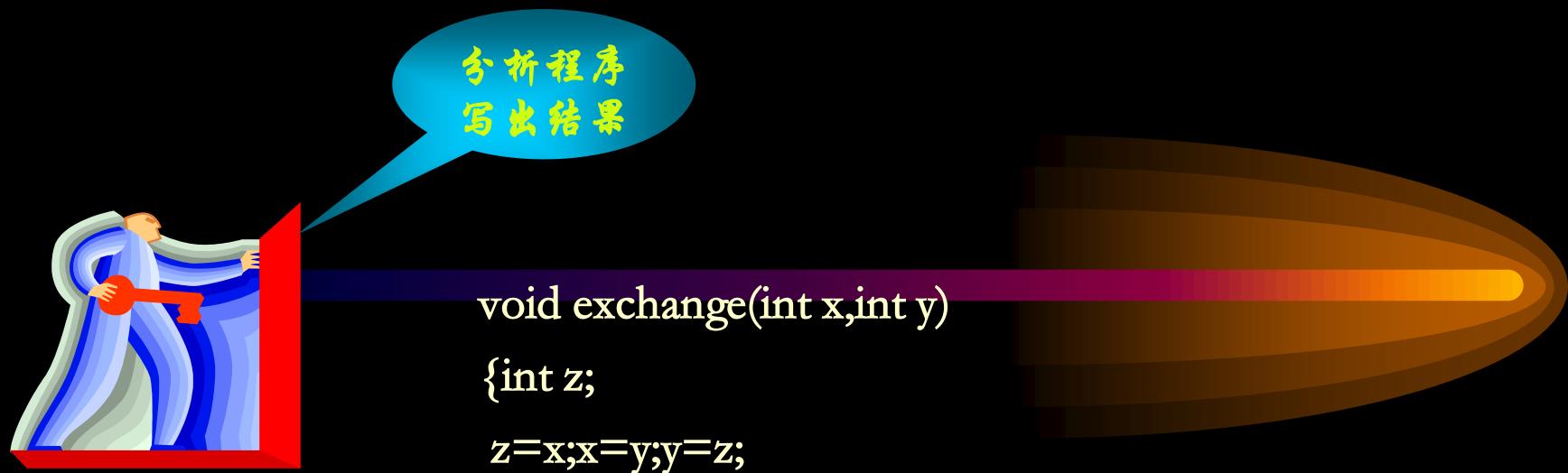
```
}
```

说明： 实参可以是常量、变量或表达式。

必须指定形参的类型，且实参与形参的类型应一致。

C 语言规定，实参变量对形参变量的数据传递是“值传递”，由实参传递给形参，而不能由形参传回来给实参。





```
void exchange(int x,int y)
{
    int z;
    z=x;x=y;y=z;
    printf( “\nx=%d,y=%d” ,x,y);
}
main()
{
    int a,b;
    scanf( “%d,%d” ,&a,&b);
    printf( “\na=%d,b=%d” ,a,b);
    echange(a,b);
    printf( “\na=%d,b=%d” ,a,b);
}
```

②函数的返回值

通过函数调用使主函数能得到一个确定的值。这个值就是函数的返回值。

- 函数的返回值是通过函数中的**return**语句获得的。
- 如果主调函数不要返回值，则可以不要**return**语句。
- 函数值的类型应该和**return**语句中表达式的类型一致。
- 如果不一致，以函数类型为准。

```
main()
{float a,b;
 int c;
 max();
 scanf("%f,%f",&a,&b);
 c=max(a,b);
 printf("max is
%d\n",c);
}
```

```
max(float x,float y)
{float z;
 z=x>y?z:y;
 return(z);
}
```

运行时若输入 1.5,2.5
则输出结果为：

max is 2



编写函数判断某一数字是否为素数。

要求：

- (1) 被判断数字作为函数参数传入。
- (2) 函数的返回值为0或1。0表示不是，1 表示是。



几点说明

- 如果被调函数中没有return语句，此时函数带回一个不确定的值。
- 可以用“**void**”定义“无类型”（或称“空类型”）。函数不带回任何值。
如：有 void printstar()

```
{...}
```

则： a=printstar();

在编译时系统会给出出错信息。

- 一个函数中可以有一个以上的**return**语句，首先被执行的起作用。
- **return**后面的值可以是一个表达式。

```
max(int x, int y)
{
    if (x>y) return(x);
    else return(y);
}
```

```
max(int x, int y)
{
    return(x>y?x:y);
}
```

3. 函数的调用

①一般形式：函数名(实参表列)；

注意：实参与形参按顺序对应，一一传递数据。

```
main()
{int i=2,p;
 int f();
 p=f(i,++i);
 printf("%d",p);
}

int f(int a,int b)
{int c;
 if (a>b) c=1;
 else if (a==b) c=0;
 else c=-1;
 return(c);
}
```

Turbo C实参求值按：
自右至左顺序。

如果要传递自左至右
顺序可采用如下格式：

```
j=i;
k=++i;
p=f(j,k);
```

②常见函数的调用方式

- 函数语句。如：

```
printstar();
```

- 函数表达式。（即函数出现在表达式中。）

```
c=2*max(a, b);
```

- 函数参数。函数调用作为一个函数的实参。如：

```
m=max(a, max(b, c));
```

```
printf("%d", max(a, b));
```



几点说明

- 被调用函数必须存在。
- 如果使用库函数，在文件必须使用include包含对应的头文件。

3. 函数参数传递

例：分析程序。

```
void swap1(int x,int y)
{int temp;
temp=x;x=y;y=temp;
}

void swap2(int *p,int *q)
{int *temp;
temp=p;p=q;q=temp;
}

void swap3(int *p,int *q)
{int temp;
temp=*p;*p=*&q;*&q=temp;
}
```

```
main()
{int a,b;
scanf("%d,%d",&a,&b);
printf("\n a=%d,b=%d",a,b);
swap1(a,b);
printf("\n a=%d,b=%d",a,b);
swap2(&a,&b);
printf("\n a=%d,b=%d",a,b);
swap3(&a,&b);
printf("\n a=%d,b=%d",a,b);
}
```

如果在输入过程中，给a输入100，给b输入200

则显示结果如下：

a=100,b=200 /*原始的输入数据*/

a=100,b=200 /* swap1(a,b);执行的结果*/

a=100,b=200 /* swap2(&a,&b);执行的结果*/

a=200,b=100 /* swap3(&a,&b);执行的结果*/

①数值作为函数参数

把数值作为实参传递给形参，这是最为常见的一种参数传递方式。该方式最大的特点：对形参的改变不会影响到实参。

```
void fun(int x,int y,int z)
{
    printf("2: x=%d  y=%d
           z=%d\n",x,y,z);
    x=x+3;
    y=x-z+2;
    z=x*y;
    printf("3: x=%d  y=%d
           z=%d\n",x,y,z);
}
main()
{
    int x,y,z;
    x=1;y=2;z=3;
    printf("1: x=%d  y=%d
           z=%d\n",x,y,z);
    fun(x,y,z);
    printf("4: x=%d  y=%d
           z=%d\n",x,y,z);
}
```

②地址作为函数参数

调用函数时，将地址作为实参传递给形参，形参和实参均为地址。

在这种方式中，被调函数改变该存储单元中的数值，即使没有return语句，也能将其值“带回”到主调函数中。

采用地址传递方式的实参一般为变量地址、数组名或指针变量。同样，接受地址值的形参变量也只能是数组名和指针变量。

① 地址作为参数时，参数的传递仍是单向的。
即对形参内容的改变不会影响实参。如果仅是在函数中对形参的值进行了修改，而没有通过引用对象运算对形参所指对象进行修改，则实参所指对象的内容不会发生变化。

```
void swap2(int *p,int *q)
{int *temp;
temp=p;p=q;q=temp;
}
```

```
void swap3(int *p,int *q)
{int temp;
temp=*p;*p=*&q;*&q=temp;
}
```

如在函数中通过引用对象运算对形参所指对象进行了修改，则可以改变实参所指对象的值。这是在使用地址作为参数时应该注意的一个问题。

②当数组作为形式参数向被调函数传递时，只传递数组的起始地址，而不是将整个数组元素都复制到函数中去。

即用数组名作为实参调用子函数，调用时将指向该数组第一个元素的指针传递给子函数。数组变量的类型在两个函数中必须相同。

例：输入n个学生成绩，通过函数求平均成绩。

```
float average(float b[],int n)
{
    int i;
    float sum;
    sum=b[0];
    for(i=1;i<n;sum+=b[i++]);
    return(sum/n);
}
```

```
main()
{
    float a[80],aver;
    int i,n;
    scanf("%d",&n);
    printf("input n scores:\n");
    for(i=0;i<n;i++)
        scanf("%f",&a[i]);
    aver=average(a,n);
    printf('average score is %7.2f',aver);
}
```



编程实现

编写函数实现： $1+2+3+4+\dots+n$

例 有一个3*4的矩阵，求其中的最大元素。

```
max_value(int array[ ][4])
```

```
{int i,j,k,max;
```

```
    max=array[0][0];
```

```
    for (i=0;i<3;i++)
```

```
        for (j=0;j<4;j++)
```

```
            if (array[i][j]>max) max=array[i][j];
```

```
    return(max);
```

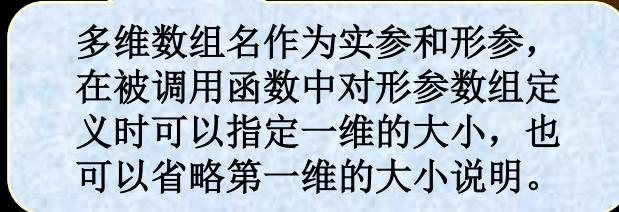
```
}
```

```
main()
```

```
{int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}};
```

```
printf("max value is %d\n",max_value(a));
```

```
}
```



多维数组名作为实参和形参，在被调用函数中对形参数组定义时可以指定一维的大小，也可以省略第一维的大小说明。



定义一个函数，功能是将含有n个元素的整型数组中的数据前后顺序颠倒。

有一个字符串。今输入一个字符，
将字符串中该字符删除。

4 .返回运算结果

①返回一个运算结果

一般被调函数使用**return**语句向主调函数返回结果。

return语句有两个主要用途：一是它能立即将程序流程从所在的函数中退出，返回到调用它的程序中去；另一个作用是为调用它的函数返回一个值。

例：编写函数完成多个数的累加求和。

```
long int sum(int b[],int n)
{int i;
 long s;
for(i=0,s=0;i<n;s+=b[i++]);
return (s);
}
```

```
main()
{
    int a[20],i,n,*p;
    scanf("%d",&n);
    for(i=0,p=a;i<n;i++) scanf("%d",p++);
    printf("\n the sum is %ld",sum(a,n));
}
```

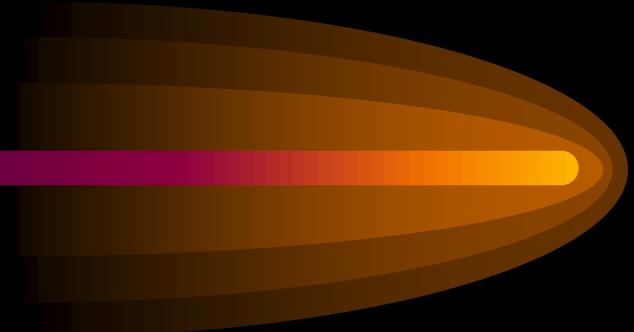
②返回多个运算结果

如果一次需要返回多个运算结果，通过return语句则无法解决这一问题。为此，C语言提供了另一种方法，就是主调函数向被调函数传递地址，当被调函数向相应的地址写入不同的数值之后，也就改变了主调函数中相应变量的值，从而达到了返回多个运算结果的目的。

例：编写函数，完成以下功能：求出一个长整数的所有位的值，并将其按照从高位到低位的顺序存放在数组之中。

```
void reverse(int b[],int n)
{int i,j,t;
for(i=0,j=n-1;i<j;i++,j--)
{t=b[i];b[i]=b[j];b[j]=t;}
}

int get_number(int a[],long x)
{int i=0;
while(x>0)
{
    a[i++]=x%10;
    x/=10;
}
reverse(a,i);
return (i);
}
```



```
main()
{
    long int x;
    int i,k,number[20];
    scanf("%ld",&x);
    k=get_number(number,x);
    for(i=0;i<k;i++)
        printf("%d",number[i]);
}
```

编程实现



编写一个函数change，要求它能将一个整数字符串转换成一个整数。如将“2345”转换为2345。

编写函数计算aaa...aaa(共n个a)的值。例如， $a=3$, $n=4$ ，则为3333。

编写一个能测出一个字符串长度的函数，函数返回值就是字符串的长度。

第九章 预处理命令

C语言提供三种预处理功能：

宏定义；

文件包含；

条件编译。

分别用宏定义命令、文件包含命令和条件编译命令来实现。这些命令以符号“#”开头，结尾没有分号(;)。

该命令的作用域是从定义的地方开始到源文件结束。

在 C 编译系统对程序进行通常的编译之前，先对程序中这些命令进行“预处理”。然后将结果和源程序一起再进行编译处理，最后得到目标代码。

1. 宏定义

① 不带参数的宏定义

功能：用一个指定的标识符(宏名)来代表一个字符串。

一般形式： #define 标识符 任意字符串序列

```
#define PI 3.1415926
main()
{float l,s,r,v;
 printf("input radius:");
 scanf(%f,&r);
 l=2.0*PI*r;
 s=PI*r*r;
 v=4.0/3*PI*r*r*r;
 printf("l=%f\ns=%f\nv=%f\n",l,s,v);
}
```

说明：

宏定义是用宏名代替一个字符串，也就是作简单的置换，不作语法检查。

#define PI 3.1415926p;

area=PI*r;

预处理后为

area=3.1415926p;*r⁵⁹

②带参数的宏定义

进行字符串替换，还要进行参数替换。

定义形式为：#define 宏名(参数表) 字符串

包含括弧中指定的参数

如： #define S(a,b) a*b

:

area=S(3,2);

area=3*2

注意：不能写成 #define S (a,b) a*b

不能有空格

```
#define PI 3.1415926
#define S(r) PI*r*r
main()
{float a,area;
 a=3.6;
 area=S(a);
 printf("r=%f\narea=%\n",a,area);
}
```

预处理

```
main()
{float a,area;
 a=3.6;
 area=3.1415926*a*a;
 printf("r=%f\narea=%\n",a,area);
}
```

如果有以下语句,

```
area=S(x+y);
```

预处理后
结果为

```
area=3.1415926*x+y*x+y;
```

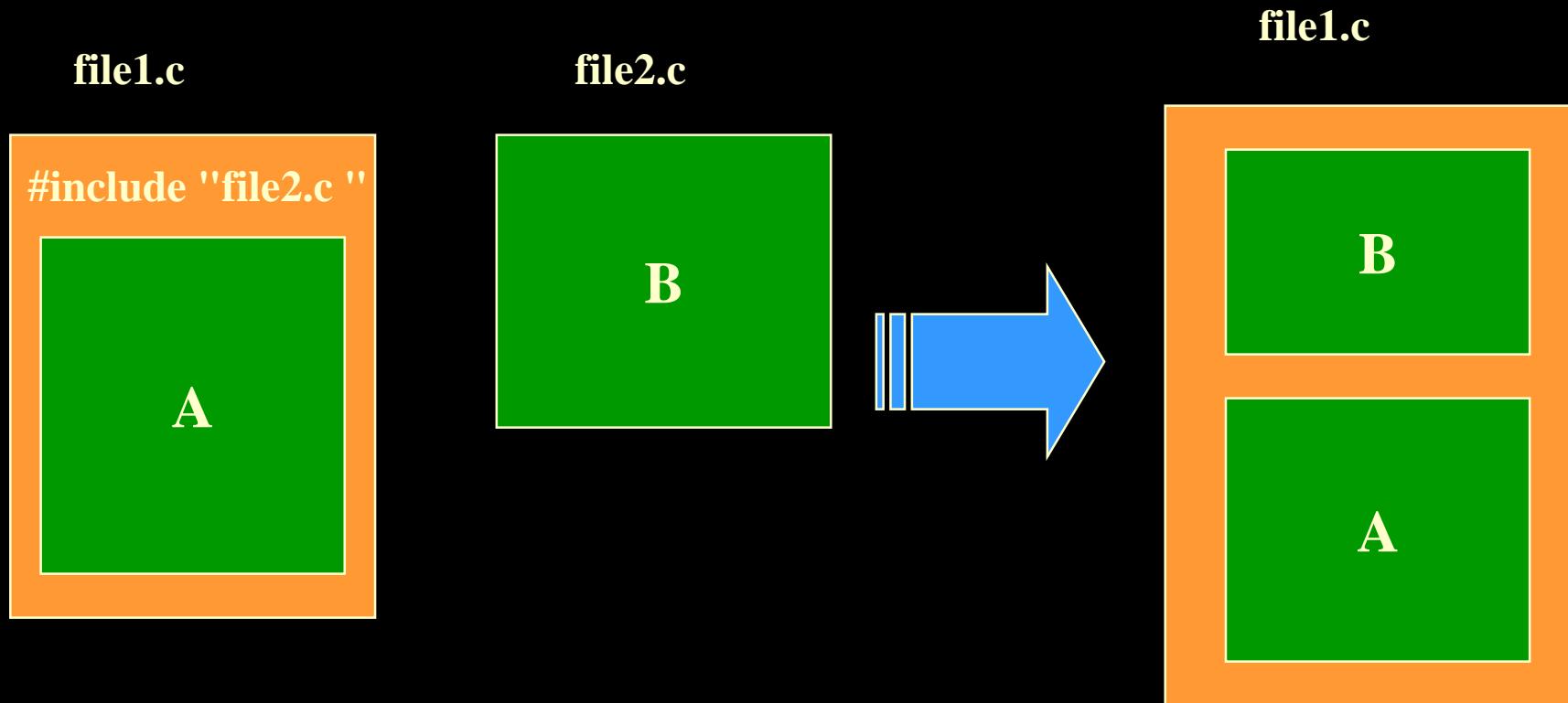
```
#define S(r) PI*(r)*(r)
```

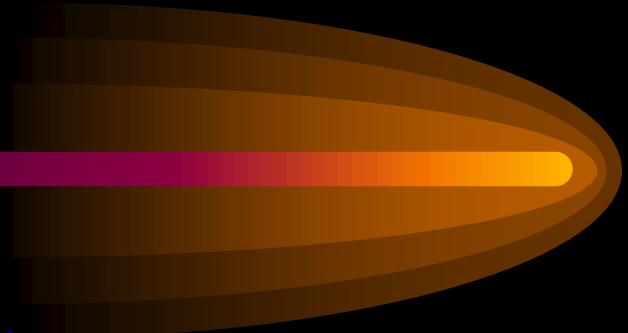
```
area=3.1415926*(x+y)*(x+y);
```

2. “文件包含” 处理

文件包含处理是指一个源文件可以将另一个源文件的全部内容包含进来。

一般形式为：#include "文件名"





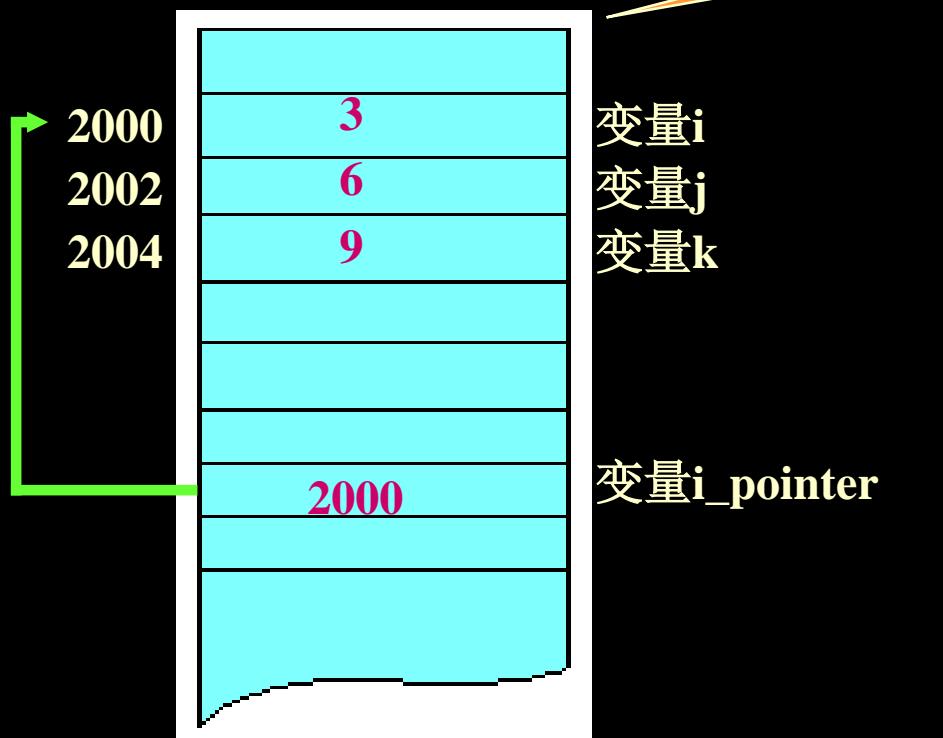
第十章 指針

1. 指针

①指针与指针变量

指针：变量的地址。

指针变量：存放另一个变量地址的变量。



注意：

(1) 对具有多个内存单元的数据，其地址为连续单元的首地址
(第一个单元的地址编码)。

(2) 注意区分指针和指针变量的概念。

指针是一个对象的地址，而指针变量是一个变量，在指针变量中存放另一个对象的地址(即指针)。

(3) 指针就是变量的地址。

同其他类型的数据一样，指针类型的数据也有指针常量、指针变量，以及各种各样的运算。

②指针变量的定义：

所指对象的类型

类型说明符

*标识符

*是指针变量标志

指针变量的名字

例如： int *p1,*p2;

float *p3,*p4;

char *c1,*c2;

③指针变量的初始化：

在定义指针变量的同时，还可以对其进行初始化，以保证指针变量中的指针有明确的指向。

例如：

```
int a,*p=&a;
```

将变量a的地址放到指针变量p中。a现在就是p所指向的对象。

错误的初始化：

(1) 情形一。

```
float *p=&b;
```

```
float b;
```

(2)情形二。

```
int a;
```

```
float *p=&a;
```

(3)情形三。

```
float *p=100;
```

(4)情形四。

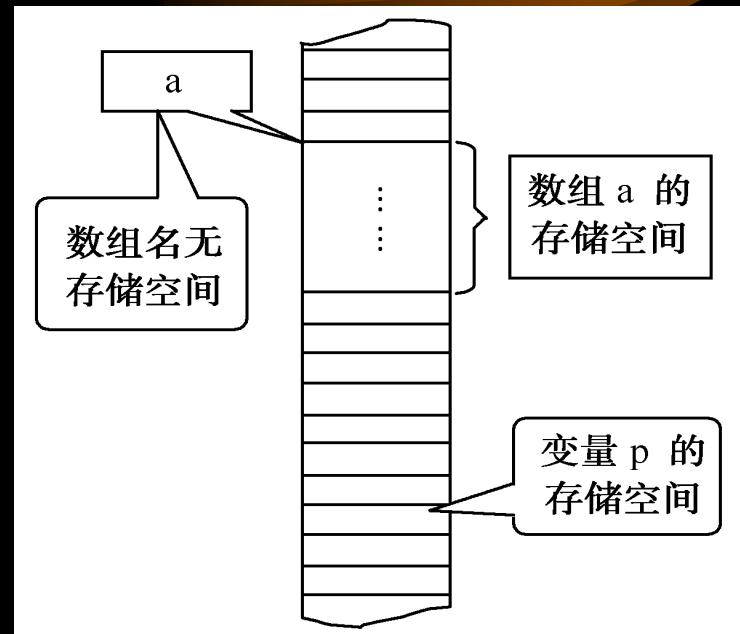
```
int *p=0;.
```

④指针常量

同其他基本数据类型一样，指针类型的数据也有常量。在C语言中，常用的指针类型常量有数组名、函数名、字符串常量、等。而在这些指针常量中，用得最多的是数组名。

例如：

```
int a[100], *p;
```



- `a`和`p`都是指针类型的数据，但是`a`是指针常量，`p`是指针变量。
- `a`指向一个能够存放100个int型数据的内存区域，而`p`在没有赋值之前没有明确的指向。

⑤指针的基本运算

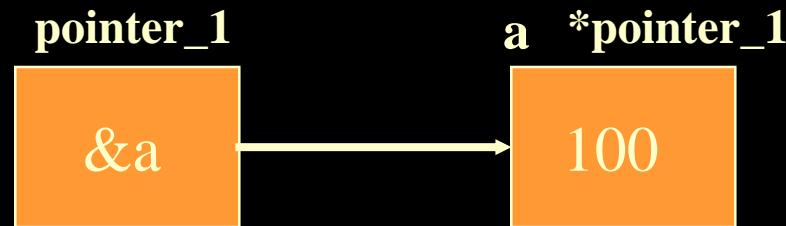
两个运算符：
 & 取地址运算符
 * 指针运算符

指针变量中只能存放地址，不能将其它类型的数值赋给一个指针变量。

例如：若有 int *pointer_1,a;

则可 pointer_1=&a; *pointer_1=100;

不能有 pointer_1=100;



注意：

(1)为了运算正确，一个指针变量应指向一个同类型的变量。

例如

```
int a, *p;
```

```
float f;
```

```
p=&a; /* 指针变量p指向整型变量a */
```

```
p=&f; /* 逻辑错误， p不能指向浮点型变量 */
```

(2)表达式*p表示引用p所指向的对象，该对象必须是确定的。

因此引用一个指针变量之前，该指针必须已经指向一个变量。

(3) 运算符&、*、++、--的优先级相同，按自右向左的方向结合。

例如

```
int a[10],*p=a;
```

则有如下的等价关系。

- p等价于a，等价于&a[0]，等价于&*p；
- *p等价于a[0]，等价于*a，等价于*&a[0]；
- (*p)++等价于a[0]++；
- *p++等价于*(p++)。

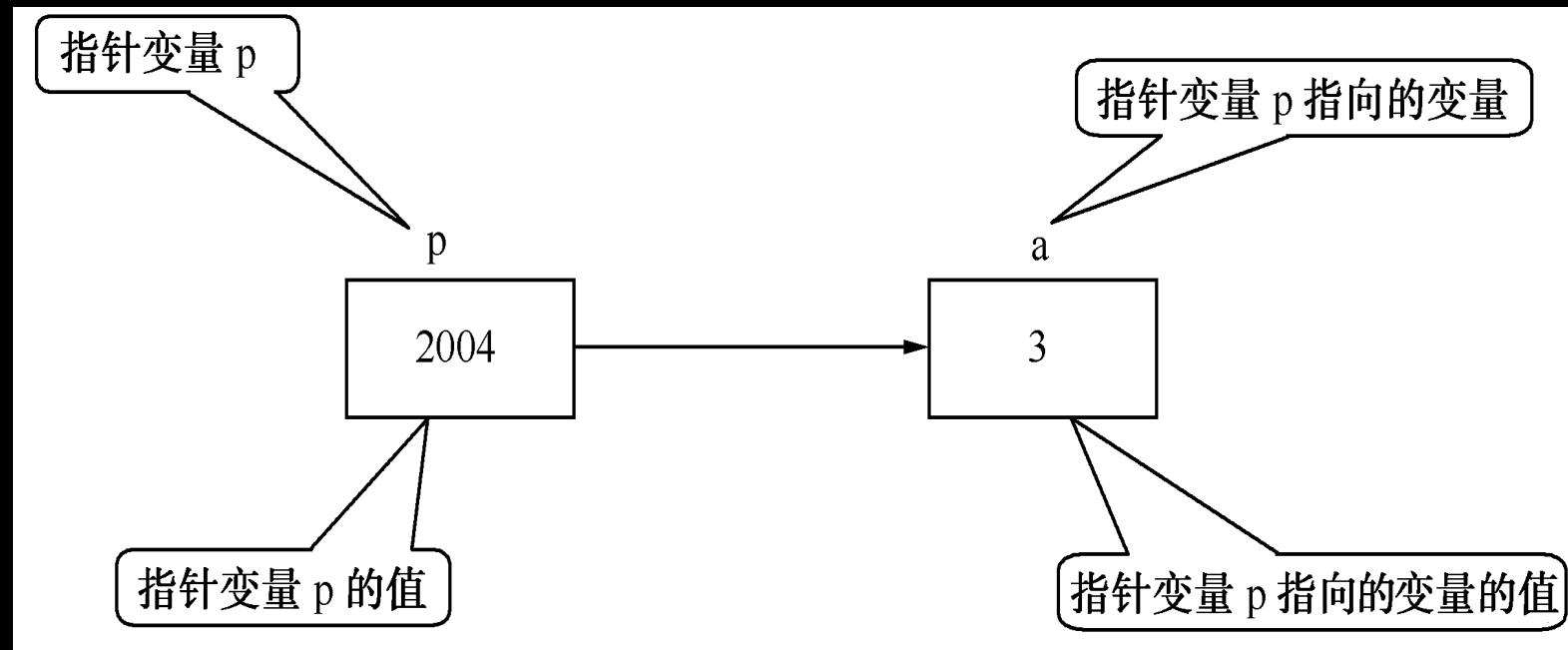
(4) 区别几个关键的概念，即指针变量和指针变量指向的变量，指针变量的值和指针变量所指向的变量的值。

例如

```
int a,*p;
```

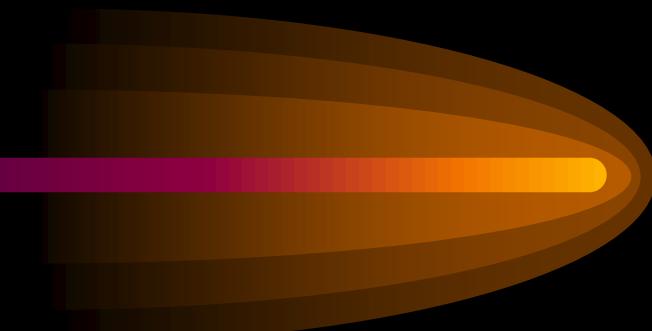
```
p=&a;
```

```
*p=3;
```



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);
}
```



若： pointer_1=&a;

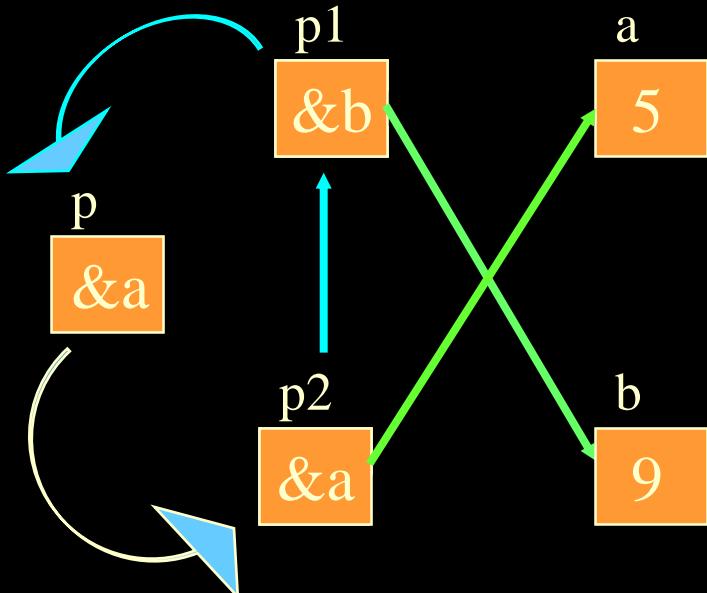
下面的表达式含义是什么？

- &*pointer_1
- *&a
- (*pointer_1)++
- *pointer_1++

例

main()

```
{int *p1,*p2,*p,a,b;  
scanf("%d,%d",&a,&b);  
p1=&a;p2=&b;  
if (a<b)  
{p=p1;p1=p2;p2=p3;}  
printf("a=%d,b=%d\n",a,b);  
printf("max=%d,min=%d\n",*p1,*p2);  
}
```



⑥指针的加减运算与关系运算

(1)指针的关系运算。

当两个指针指向同一个数组时，两个指针可以比较大小。当这两个指针相等时，说明这两个指针指向同一个数组元素。

运算符包括：<、>、<=、>=、==、!=等关系运算符。

注意：

对指针的关系运算同其他关系表达式一样，关系成立时，表达式的计算结果为1；否则为0。

(2)指针的加减运算。

一个指针可以加上或减去一个整数值，包括加1或减1运算。
如 $p++$ 、 $p--$ 、 $++p$ 、 $--p$ 等。

注意：

$p \pm n$ 为一指针表达式，C编译程序在处理一个指针型数据加（减）一个整数n这种指针表达式时，并不是简单地将指针加（减）n，而是要将指针加上（减去）n个目标数据元素所占据的内存单元数目。

例：分析程序，理解指针运算。

```
main()
```

```
{
```

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

```
    int *p;
```

```
    p=&a[0];
```

```
    printf("%3d",*p);
```

```
    p=p+2;
```

```
    printf("%3d",*p);
```

```
    p--;
```

```
    printf("%3d",*p);
```

```
}
```



运行结果为：

1 3 2

9. 指针与一维数组

数组名与指针是完全相同的数据类型，它们之间的运算是通用的。它们之间的区别仅在于数组名是指针常量而已。

在使用数组时，通过下标引用来使用数组元素。

实际上，[]运算在编译时会自动转换为指针目标引用运算。

以下等价关系成立：

- (指针表达式)[n] 等价于 *(指针表达式+n)
- 数组名[n] 等价于 *(数组名+n)

例：

main()

{

```
int a[10],*p=a,i;  
for(i=0;i<10;i++) scanf("%d",p++);  
for(i=0;i<10;i++)  
    printf("a[%d]=%d\n",i,a[i]);  
for(i=0;i<10;i++)  
    printf("*(%d)=%d\n",i,*(a+i));  
}
```

$*(a+i)$ 与数组 $a[i]$ 可实现相同的功能。实际上，表达式 $a[i]$ 是访问以 a 为起点的第 i 个数据，它先进行地址计算，求 $a+i$ ，然后访问该地址的对象。运算符“*”是访问地址所指向的对象。

例：比较程序

① 下标法

```
main ()  
{  
    int a[10], i;  
    for(i=0;i<10;i++)  
        scanf("%d", &a[i]);  
    printf("\n");  
    for(i=0;i<10;i++)  
        printf("%d ",a[i]);  
}
```

② 用数组名计算地址

```
main ()  
{  
    int a[10],i;  
    for(i=0;i<10;i++)  
        scanf("%d", a+i);  
    printf("\n");  
    for(i=0;i<10;i++)  
        printf("%d ",*(a+i));  
}
```

③ 用指针访问各元素

```
main ()  
{  
    int a[10],*p, i;  
    for(p=a,i=0;i<10;i++)  
        scanf("%d", p++);  
    printf("\n");  
    for(p=a,i=0;i<10;i++)  
        printf("%d ",*p++);  
}
```



- 1、编写程序完成如下功能：分别将字符串a和字符串b中的字符倒置。然后按交叉的顺序将两个字符数组合并到字符数组c中，过长的部分直接连接在c的尾部。（例如，若字符串a的内容为“abcdefg”，字符串b的内容为“1990”，则结果为：“h0g9f9e1dcba”）
- 2、键入一串字符（换行符结束），用循环语句将其中的小写英文字母互换后输出。
- 3、设有语句“int a[3][4];”，先为数组输满数据，再将该数组周边的元素输出（元素输出次序不限）。

指针与多维数组

指针极大地丰富了C语言的功能。学习指针是学习C语言最重要的一环，正确地理解指针与多维数组的关系、指针数组的概念、函数指针以及多级指针的概念，将有助于编写高效的程序。

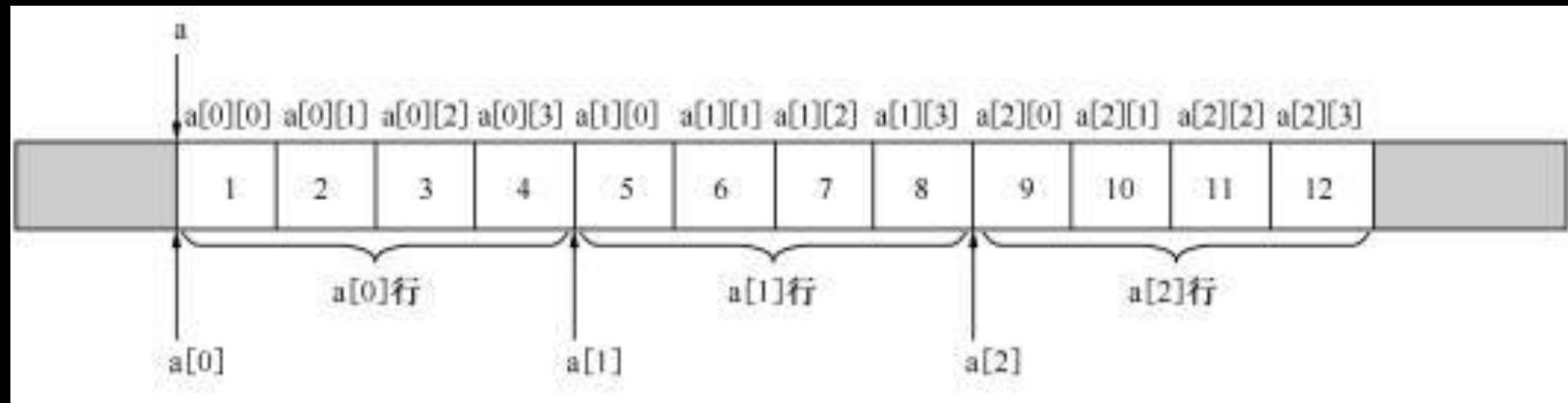
1.指针与多维数组

①二维数组的线性存储

设有

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

定义了一个二维数组a，共有12个元素。从逻辑上可以把该二维数组a看成一张由3行4列元素组成的二维表格。



可以看出，它同一维数组的存放形式一样，但二者引用数组元素地址的计算方法有所区别。

②理解二维数组

(1)C语言将二维数组理解为其构成元素是一维数组的一维数组。

二维数组a中有3个元素： a[0]、 a[1]、 a[2]。

而每个元素a[i]（行数组）又由4个数据元素a[i][0]、 a[i][1]、 a[i][2]、 a[i][3]组成。

(2)表达式a+i（行指针）。

对于二维数组，数组名a是一个“指向行的指针”，它指向数组第0行。而且它仍然是数组的首地址，即第0个元素的地址。由于a是指向行的指针，表达式a+i中的偏移量i是以“行”为单位的，所以a+i就是第i行的地址。

(3)二维数组元素的地址。

表达式 $a[i]$ 和 $*(a+i)$ 表示第*i*行的首地址，因而表达式 $*(a+i)+j$ 就是第*i*行第*j*个元素的地址，即数组a中 $a[i][j]$ 元素的地址。所以有如下等价关系： $*(a+i)+j$ 等价于 $a[i]+j$ 等价于 $\&a[i][j]$

(4)二维数组中元素偏移量计算。

C语言对二维数组元素地址的处理方法是，先计算行地址，再计算列地址。对于二维数组 $a[n][m]$ 中的任意一个元素 $a[i][j]$ ，相对于 $a[0][0]$ 的偏移量计算公式为： $i*m+j$ 。其中m为该二维数组的列数。

例：用数组元素的偏移量访问数组。

main()

{

int a[3][4]={{1,2,3,4},{2,3,4,5},{3,4,5,6}};

int i,j;

for(i=0; i<3; i++)

{ for(j=0; j<4; j++)

 printf("a[%d][%d]=%-5d",i,j,*(&a[0]+i*4+j));

 printf("\n");

}

}

运行结果为：

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

a[1][0]=2 a[1][1]=3 a[1][2]=4 a[1][3]=5

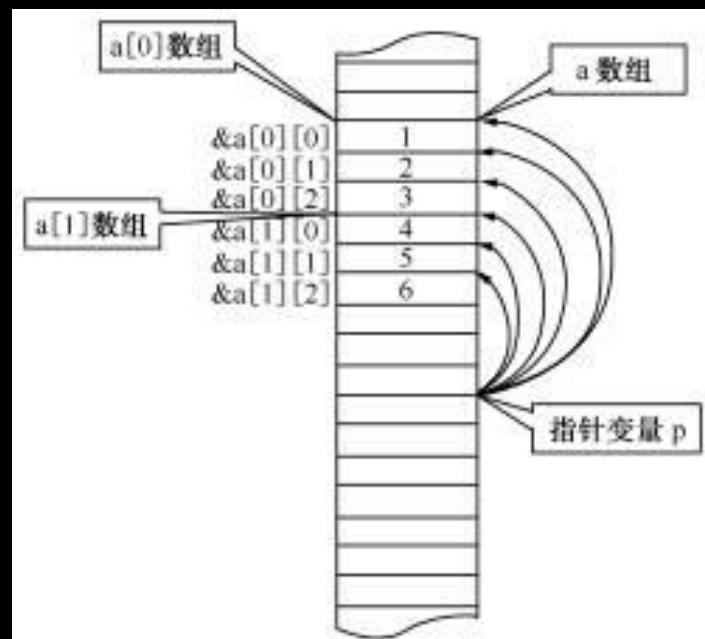
a[2][0]=3 a[2][1]=4 a[2][2]=5 a[2][3]=6

③通过指针访问二维数组

由于二维数组在计算机中的存放形式是顺序存放的，所以只要定义一个与数组元素类型相一致的指针变量，再将数组中某个元素的地址赋给这个指针变量，通过对该指针的移动和引用，就可以访问到数组中的每一个元素。

例：用指针变量输出二维数组的元素。

```
main()
{
int i,a[2][3]={1,2,3,4,5,6};
int *p;
for(p=&a[0][0],i=0;i<6; i++)
{
    if(i%3==0) printf("\n");
    printf("%3d", *p++);
}
```



```
main()
{
int a[2][3]={1,2,3,4,5,6};
int *p,i,j;
for(p=&a[0][0],i=0;i<2; i++)
{
    printf("\n");
    for(j=0;j<3;j++)
        printf("%3d", *(p+i*3+j));
}
```

```
main()
{
int a[2][3]={1,2,3,4,5,6};
int i,j;
for(i=0;i<2;i++)
{
printf("\n");
for(j=0;j<3;j++)
printf("%3d", *(*(a+i)+j));
}
}
```

三个程序的运行结果相同，
都为：

1 2 3
4 5 6

可以发现，方法二和方法三都是通过计算元素地址的偏移量实现对数组元素的访问，可是两者的使用方法却不同：
一个是*(p+i*3+j)，
另一个是*(*(a+i)+j)。无法实现指针使用格式的统一，原因主要在于指针p所指向的对象是一个整型元素，而指针a所指向的对象却是一个一维数组。

④指向一维数组的指针变量

这是一种指向对象是一维数组的指针变量，可以用它指向一个二维数组的某一行，然后进一步通过它再访问数组中的元素。

类型标识符 (* 标识符)[所指数组元素个数];

注意：圆括号不能省

例如：

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

表示变量p是指向有4个元素的一维整型数组的指针变量

```
char (*q)[20];
```

表示变量q是指向有20个元素的一维字符数组的指针变量

例：输出二维数组任一元素的值。

```
main()
{
int a[2][6]={0,1,2,3,4,5,6,7,8,9,10,11};
int (*p)[6],i,j;
p=a;
scanf('%d,%d",&i,&j);
printf('\na[%d][%d]=%d\n',i,j,*(*(p+i
)+j));
}
```

若输入： 1,2

则输出为： a[1][2]=8

假设有如下定义：

```
int a[3][4], *p, (*pa)[4]; p = a[0]; pa = a;
```

注意区分下列表达式的含义。

- ① a、 *a、 **a、 a[2]、 a+2、 *a+2；
- ② p、 p++、 p+2、 *(p+2)、 *p+2、 p+1*4+2、 *(p+2*4)；
- ③ pa、 pa++、 pa+2、 *pa、 *pa+2、 *(pa+2)、 *(*pa+2)、
*(*pa+1)+2。

通过指针变量存取数组元素速度快且程序简明。用指针变量作形参，可以允许数组的行数不同。因此数组与指针联系紧密。

2.指针数组与指针的指针

①指针数组

定义形式如下： 类型标识符 *数组名[整型常量表达式];

例如： char *strings[10]; int *a[20];

例：用指针数组输出n个字符串。

```
#include "stdio.h"
```

```
main()
```

```
{ char *ps[4]={"Unix","Linux","Windows","Dos"};
```

```
int i;
```

```
for(i=0;i<4;i++) puts(ps[i]);
```

```
}
```

运行结果为

Unix

Linux

Windows

Dos

②指向指针的指针

指针数组的数组名是一个指针常量，它所指向的目标是指针型数据。也就是说，其目标又是指向其他基本类型数据的指针，所以指针数组名是指向指针类型数据的指针，故称它为指针的指针。

定义形式如下： 类型符 **变量名；

例如：

float **pp;

表示定义指向指针类型的指针变量pp，它所指向的对象是“float *”类型（即指向实型数的指针变量）。

例如，有如下程序段：

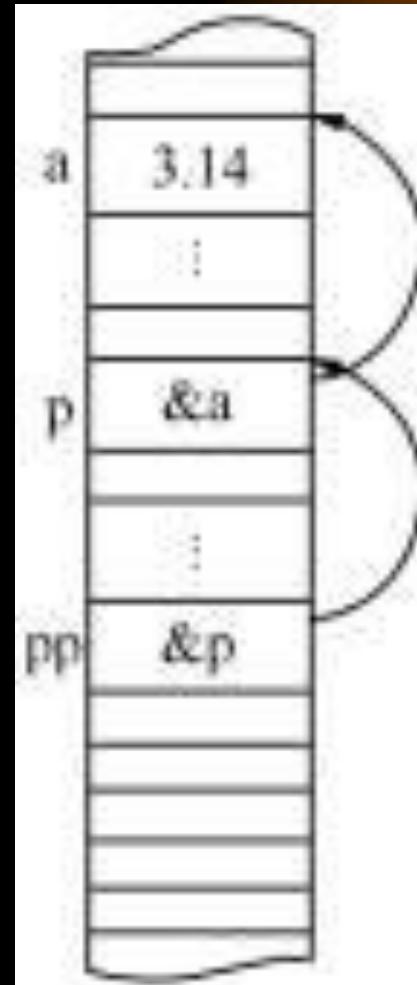
```
float a=3.14;
```

```
float *p;
```

```
float **pp; /* pp是指向float *类型数据的指针 */
```

```
p=&a;
```

```
pp=&p; /* 将p的地址赋给pp */
```



指向字符串的指针

- 将指针变量指向字符串常量的方法:
- 1.在数据定义语句中用赋值
 - *指针变量=字符串常量
 - `char *p=“abcdef”;`
- 2.直接赋值
 - `char *p;`
 -
 - `p=“abcdef”`

例：用指向指针的指针变量将一批顺序给定的字符串按反序输出。

```
main()
{
int i;
char *name[ ]={"Unix","Linux","Windows","C
language","Internet"};
for(i=4; i>=0; i--)
{ p=name+i; printf("%s\n",*p);
}
}
```

运行结果为

Internet

C language

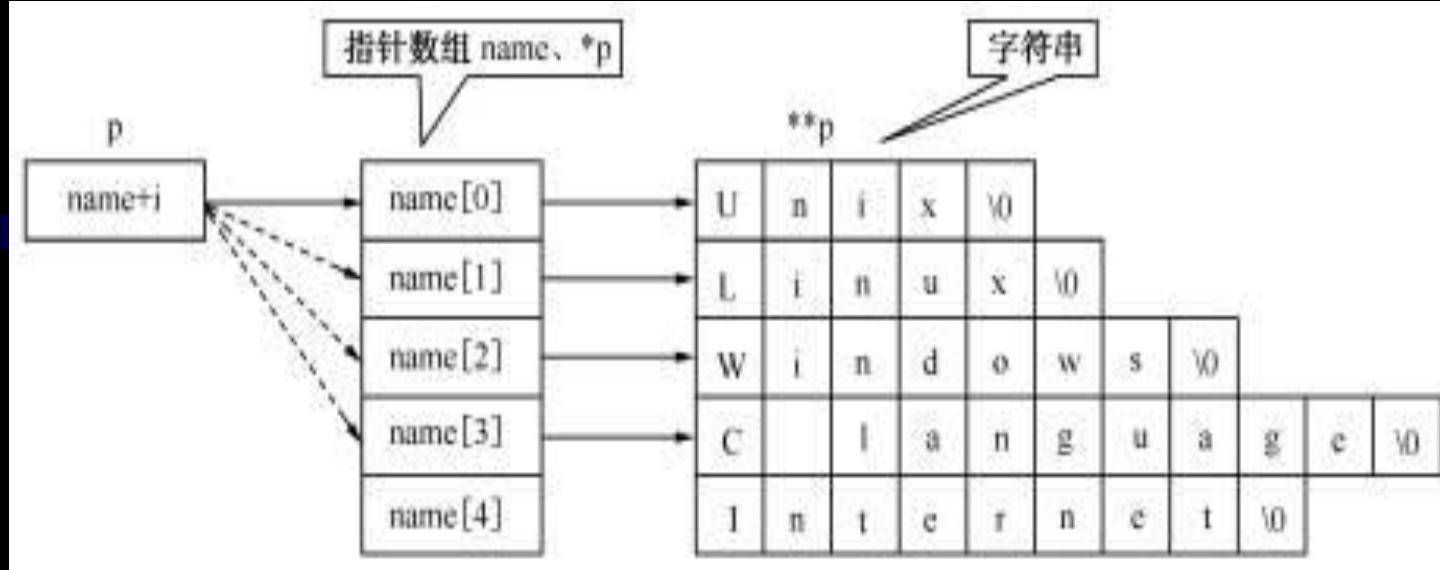
Windows

Linux

Unix

注意：

该程序是用指向指针的指针变量来访问字符串，所以在“`printf("%s\n", *p);`”语句中使用了`*p`形式。请注意其与`**p`的区别。`**p`表示一个具体的字符对象，`p`存放的是`name`数组元素的地址，而`*p`是目标对象的地址。



指向函数的指针

指向函数的指针的定义：

类型标识符 (*标识符)();

通过函数指针调用函数的形式：

(*函数指针变量) (实参表) ;

例： 输入一个两个数的四则运算式，通过函数指针求该运算式的值。

```
float add(float a,float b)
{return a+b;}

float minus(float a,float b)
{return a-b;}

float mult(float a,float b)
{return a*b;}

float div(float a,float b)
{return a/b;}
```

```
main()
{ float m,n,r;
char op;
float (*p)();
scanf("%f%c%f",&m,&op,&n);
switch(op)
{case '+':p=add;break;
 case '-':p=minus;break;
 case '*':p=mult;break;
 case '/':p=div;break;
 default:printf("error operation");
}
r=(*p)(m,n);printf("%f",r);}
```

3. 对指针的几点说明

(1) 有关指针的说明很多是由指针、数组、函数说明组合而成的。但它们并不是可以任意组合的，如数组就不能由函数组成，即数组元素不能是一个函数；函数也不能返回一个数组或返回另一个函数。例如，“int a[5]();”就是错误的。

(2) 与指针有关的常见说明和意义

表 14.2 与指针有关的常见说明和意义表

指 针	意 义
<code>int *p;</code>	p 为指向整型量的指针变量
<code>int *p[n];</code>	p 为指针数组，由 n 个指向整型量的指针元素组成
<code>int (*p)[n];</code>	p 为指向整型一维数组的指针变量，一维数组的大小为 n
<code>int *p();</code>	p 为返回指针值的函数，该指针指向整型量
<code>int (*p)();</code>	p 为指向函数的指针，该函数返回整型量
<code>int **p;</code>	p 为一个指向另一指针的指针变量，该指针指向一个整型量

练习题

- 1、输入一个有数字组成的字符串，输出对应的数值。
- 2、输入2个字符串分别存入两个一维数组，将其连接后存入第3个一维数组，不使用系统函数`strcat`，用指针完成。



第十一章 结构体与共用体

结构体与共用体

①结构体

结构体是将不同的数据类型组织在一起而形成的一个有机的整体。

num	name	sex	age	score	addr
10010	Li Fun	F	18	87.5	Xi'an

定义结构体类型的一般形式:

struct 结构体名

{成员1;

成员n;

};

每个成员的定义格式:

类型标识符 成员名

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

struct 是关键字, **student** 是定义的类型名。**struct student** 是定义一个结构体类型, 它包括了 **num, name, sex, age, score, addr** 不同类型的的数据项。

定义一个结构体类型
struct student

②定义结构体类型变量的方法

有三种定义方法:



- 先定义结构体类型再定义变量。
- 在定义类型的同时定义变量。
- 直接定义结构类型变量。

(1)先定义结构体类型再定义变量

例如上面定了一个结构体类型**struct student**, 可以用它来定义变量。如:

```
struct student s1,s2;
```

s1和**s2**为**struct student**类型变量, 即它们具有如下的结构:

	num	name	sex	age	score	addr
s1						
s2						

(2)在定义类型的同时定义变量

```
struct student  
{int num;  
 char name[20];  
 char sex;  
 int age;  
 float score;  
 char addr[30];  
 }s1,s2;
```

定义了两个具有如下结构的变量s1和s2。

	num	name	sex	age	score	addr
s1						
s2						

(3) 直接定义结构类型变量

定义一般形式：

```
struct  
{成员1;  
 ...  
 成员n;  
 }变量名表列;
```

例如

```
struct  
{int num;  
 char name[20];  
 char sex;  
 int age;  
 float score;  
 char addr[30];  
 }s1,s2;
```

注意： 先定义类型，后定义变量。只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。
成员名可以与程序中的变量名相同，二者代表不同的对象。

成员也可以是一个结构体变量，即结构可以嵌套。

例如：

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

struct staff
{
    unsigned int number;
    char name[20];
    char sex;
    struct date birthday;
    char address[30];
}s1;
```

number	name	sex	birthday			address
			year	mouth	day	

③结构体类型变量的引用

当两个结构变量的类型相同时，可以互相赋值。

对成员引用的一般格式：

结构体变量名.成员名

成员(分量)运算符

互相赋值：

`s2=s1;`

例如： `s1.number`

`s1.name`

`s1.birthday.year`

`s2.number=s1.number+1`

`scanf("%d",&s1.number)`

`s1.birthday.month`

`s1.birthday.day`

`s1.addrees`

④结构体类型变量的初始化

在定义结构变量时，给其置初值。

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

```
struct student s1={10010,"Li ming", 'F',18,87.5, "Xi'an"};
```

⑤结构体数组

一个数组的各个元素都是一个结构类型数据，这样的数组就是结构数组。

(1) 结构体数组的定义

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

struct student s1[5];
```

(2)结构体数组的初始化

```
struct student
{int num;
char name[20];
char sex;
int age;
float score;
char addr[30];
}stu[3]={{{10101,"LI Lin", 'M',18},
          {10102, "Zhang Fun", M',19},
          {10104, "Wang Min", 'F',20}}};
```

例：计算学生的平均成绩和不及格的人数。

```
struct stu
{
    int num;
    char *name;
    char sex;
    float score;
}boy[5]={
{101,"Li ping",'M',45},
{102,"Zhang ping",'M',62.5},
{103,"He fang",'F',92.5},
{104,"Cheng ling",'F',87},
{105,"Wang ming",'M',58},
};
```

```
main()
{
    int i,c=0;
    float ave,s=0;
    for(i=0;i<5;i++)
    {
        s+=boy[i].score;
        if(boy[i].score<60) c+=1;
    }
    printf("s=%f\n",s);
    ave=s/5;
    printf("average=%f\n",ave);
    printf("count=%d\n",c);
}
```

某小区选举，共有50人参加，候选人为
“zhang”， “ li”， “ wang”，统计选举结果

```
#include "stdio.h"
#include "string.h"
struct f
{char name[6];
int s;
}a[3]={{"zhang",0}, {"li",0}, {"wang",0}}
main()
{char f[6];int i,j;
for(i=0;i<50;i++)
{gets(f);
for(j=0;j<3;j++)
if (strcmp(f,a[j].name)==0)a[j].s++;
}
for(i=0;i<3;i++)
Printf("\n%s:%d",a[i].name,a[i].s);
}
```

⑥指向结构体类型数据的指针

一个结构体变量的指针就是该变量所占据的内存段的起始地址。可以设一个指针变量，用来指向一个结构体变量。

(1)指向结构体变量的指针

```
#include "string.h"
main()
{struct student
    {long int num;
     char name[20];
     char sex;
     float score;
    };
struct student stu_1,*p=&stu_1;
stu_1.num=89101;
strcpy(stu_1.name,"Li Lin");
stu_1.sex='M';}
```

```
stu_1.score=89.5;
```

```
printf('No.:%ld\nname:%s\nsex:%c\nscore:%f\n'"
```

```
    stu_1.num,stu_1.name,stu_1.sex,stu_1.score);
```

```
printf('\nNo.:%ld\nname:%s\nsex:%c\nscore:%f\n'"
```

```
(*p).num, (*p).name, (*p).sex, (*p).score);
```



在用指针引用结构体成员时， *p两侧的括号不能省略， 应为成员运算符". "优先于" * "运算符。

C 语言中还可以通过 “->” 引用成员。

如： **p->name**

p->sex

p->score

等价于

(*p).name

(*p).sex

(*p).score

注意：“->”的优先级与“()”、“[]”和句点“.”相同。

++p->num

等价于 **++(p->num)**

(++p)->num

先对p加1， 再引用num

p++->num

等价于 **(p++) ->num**

再看一种结构成员为指针的情况：

struct

```
{int x;  
    int *y;  
}*q;
```

则：	$*q \rightarrow y$	表示y所指的内容；
	$*q \rightarrow y++$	访问y所指的内容后y自增；
	$(*q \rightarrow y)++$	自增y所指的内容；
	$*q++ \rightarrow y$	访问y所指的内容后q自增。

例：有一个结构体变量stu，内含学生学号、姓名和三门课的成绩。要求在main函数中赋以值，在另一函数print中将它们打印输出。

```

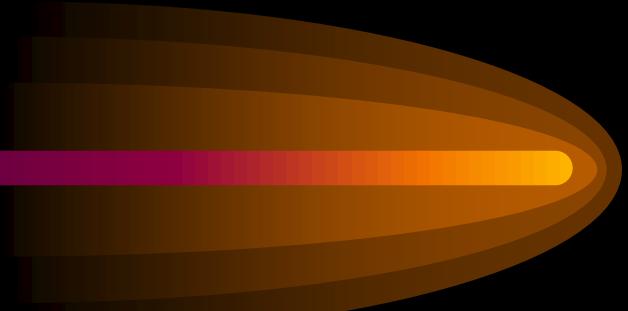
#include "string.h"
#define format "%d\n%s\n%f\n%f\n%f\n"
struct student
{
    int num;
    char name[20];
    float score[3];
};

main()
{
    void print();
    struct student stu;
    stu.num=12345;
    strcpy(stu.name, "Li Li");
    stu.score[0]=67.5;
    stu.score[1]=89;
    stu.score[2]=78.6;
    print(&stu);
}

void print(p)
{
    struct student *p;
    {printf(format,p->num,p->name,p->score[0],p->score[1],p->score[2]);
     printf("\n");
}
}

```

The diagram shows the memory layout of a `student` structure. A pointer `p` points to the beginning of the structure. The structure itself is represented as a vertical stack of five boxes. From top to bottom, the boxes are labeled: `Num`, `name`, `score[0]`, `score[1]`, and `score[2]`. The `name` box is orange and contains the string "Li Li". The `score` box is orange and contains three floating-point numbers: 67.5, 89, and 78.6. A callout box labeled "stu" points to the pointer `p`.



11.7 用指针处理链表

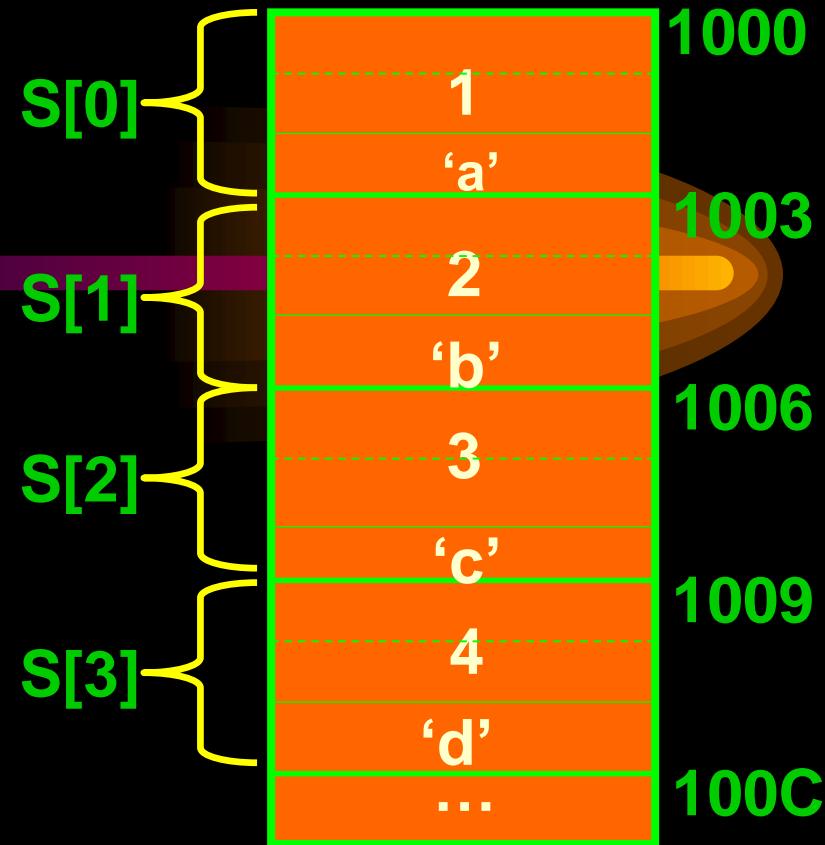
```
struct data
```

```
{ int x;
```

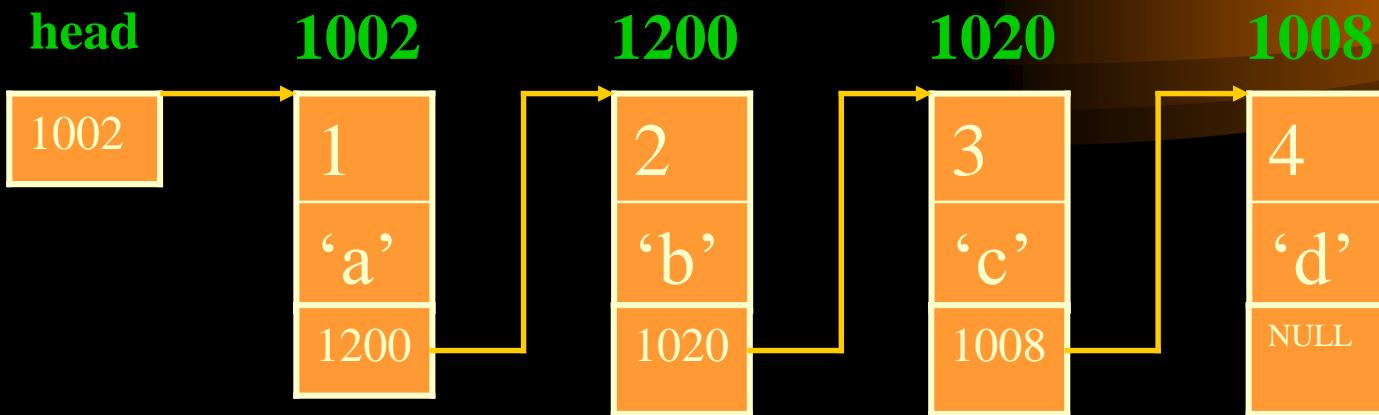
```
char c;
```

```
}s[4]={1,'a',2,'b',3,'c',4,'d'};
```

```
struct data s[100];
```



静态分配存储单元,容易造成内存浪费



链表：是一种常见的重要的数据结构，它可根据需要，动态分配内存单元。

链表中每个元素称为结点，其内容：

数据部分：可有若干项（整、实、字符、结构体类型等）

**指针变量：下一结点的地址，最后一个
结点的地址部分为NULL**

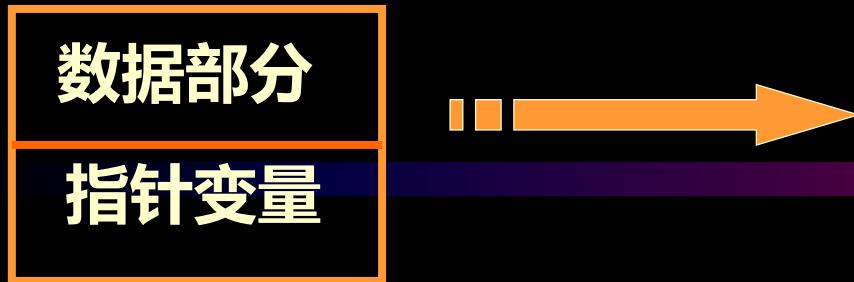
结点

数据部分
指针变量

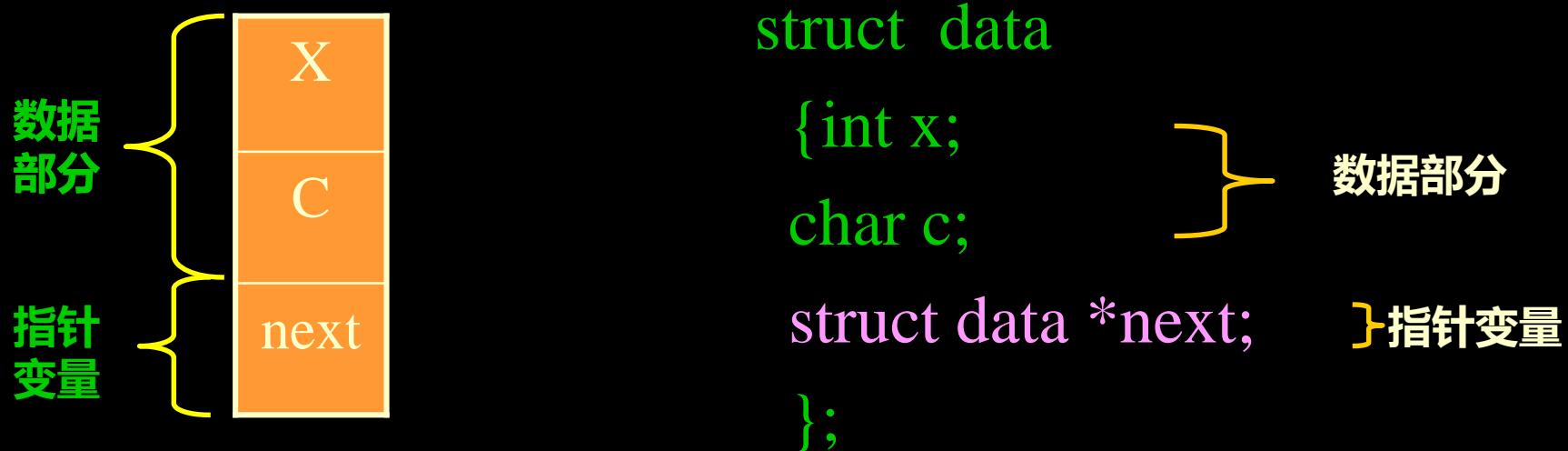


由结构体类型变量实现****

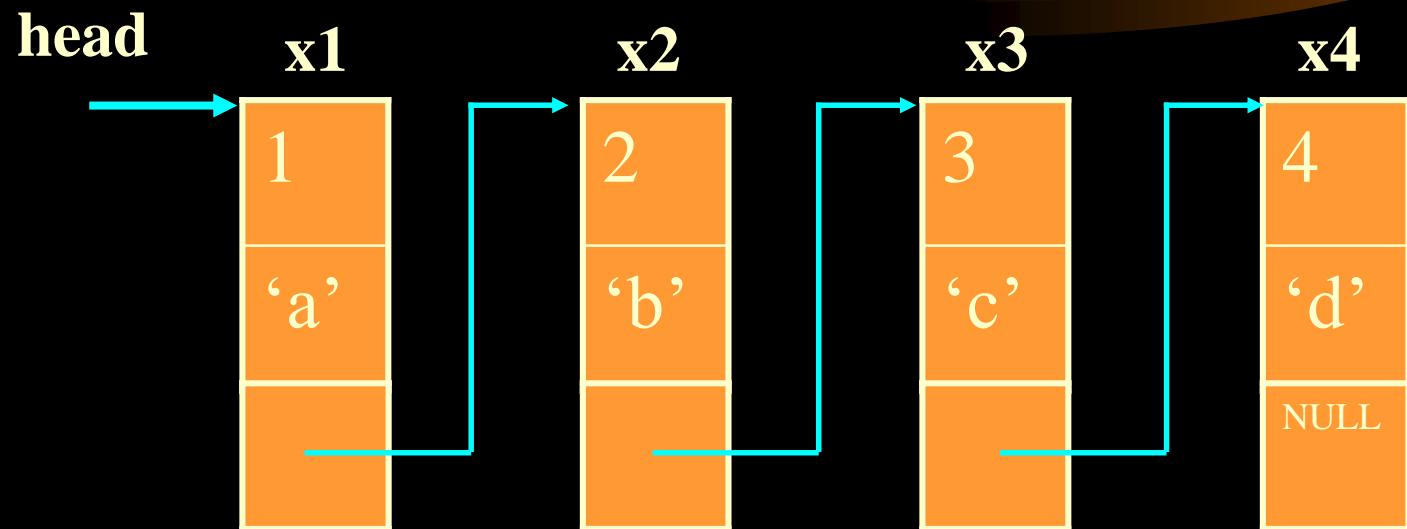
结点



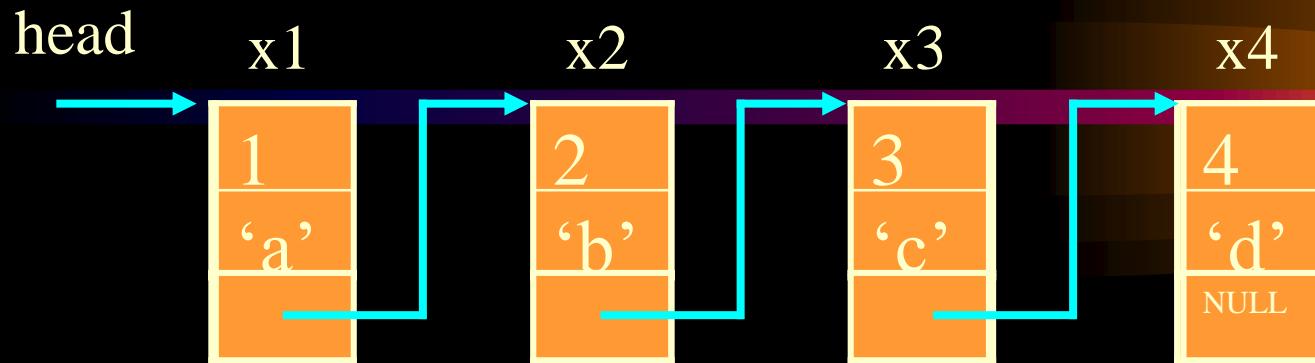
```
struct 结构体名  
{ 数据部分  
    指针变量  
};
```



建立简单链表



建立简单链表

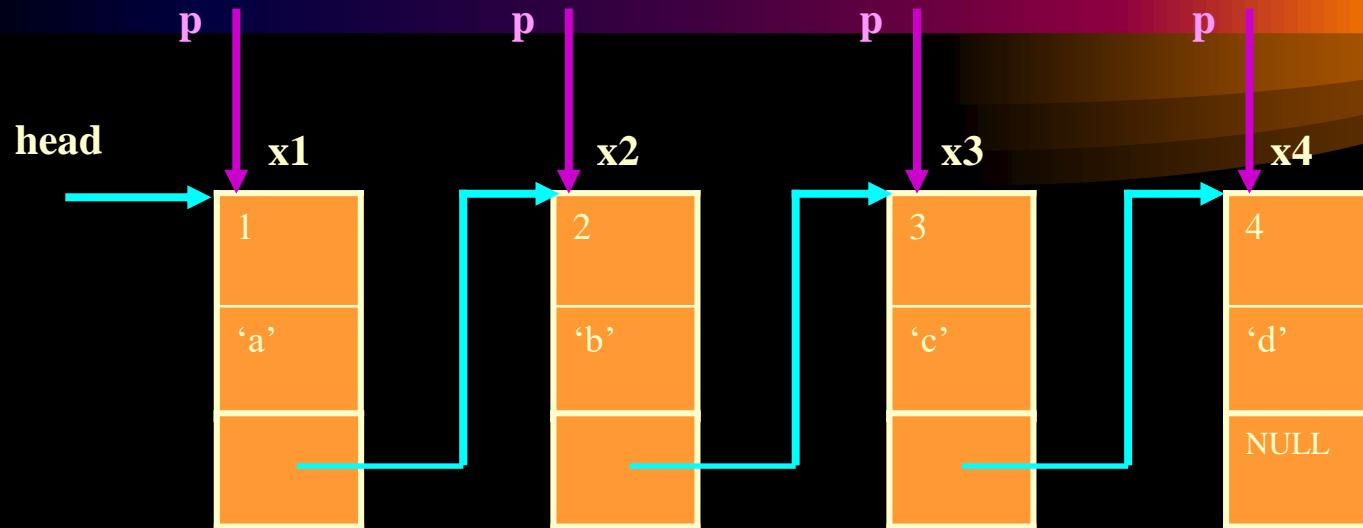


```
#define NULL 0
struct data
{int x;
char c;
struct data *next; };

main()
{ struct data x1,x2,x3,x4,*head;
```

```
x1. x=1;    x1. c='a';
x2. x=2;    x2. c='b';
x3. x=3;    x3. c='c';
x4. x=4;    x4. c='d';
x1. next=&x2;
x2. next=&x3;
x3. next=&x4;
x4.next=NULL;
head=&x1; }
```

输出链表



p=head;

```
[<-->] while(p!=NULL, p->x, p->c); } while(p!=NULL)
[<-->] { printf("%d,%c\n", p->x, p->c); p=p->next; }
```

```
void output(struct data *head)
{struct data *p;
p=head;
while(p!=NULL)
{printf("\n%d,%c",p->x,p->c);
p=p->next;}
return;}
```

完整程序：

```
#define NULL 0
struct data
{int x;
int c;
struct data *next;};
void f(struct data *head)
{struct data *p;
p=head;
while(p!=NULL)
{printf("\n%d,%c",p->x,p->c);
p=p->next;}
return;}
main()
{struct data x1,x2,x3,x4,*head;
x1.x=1;x1.c='a';
x2.x=2;x2.c='b';
x3.x=3;x3.c='c';
x4.x=4;x4.c='d';
x1.next=&x2;x2.next=&x3;x3.next=&x4;
x4.next=NULL;
head=&x1;
f(head); }
```

输出链表

建立链表

File Edit Run Compile Project Options Debug Break/watch

Edit

Line 17 Col 37 Insert Indent Tab Fill Unindent * C:\MAJUN1.C

```
#define NULL 0
struct data
{
    int x;
    int c;
    struct data *next;
};

void f(struct data *head)
{
    struct data *p; p=head;
    while(p!=NULL)
    {
        printf("\n%d,%c",p->x,p->c);
        p=p->next;
    }
    return;
}

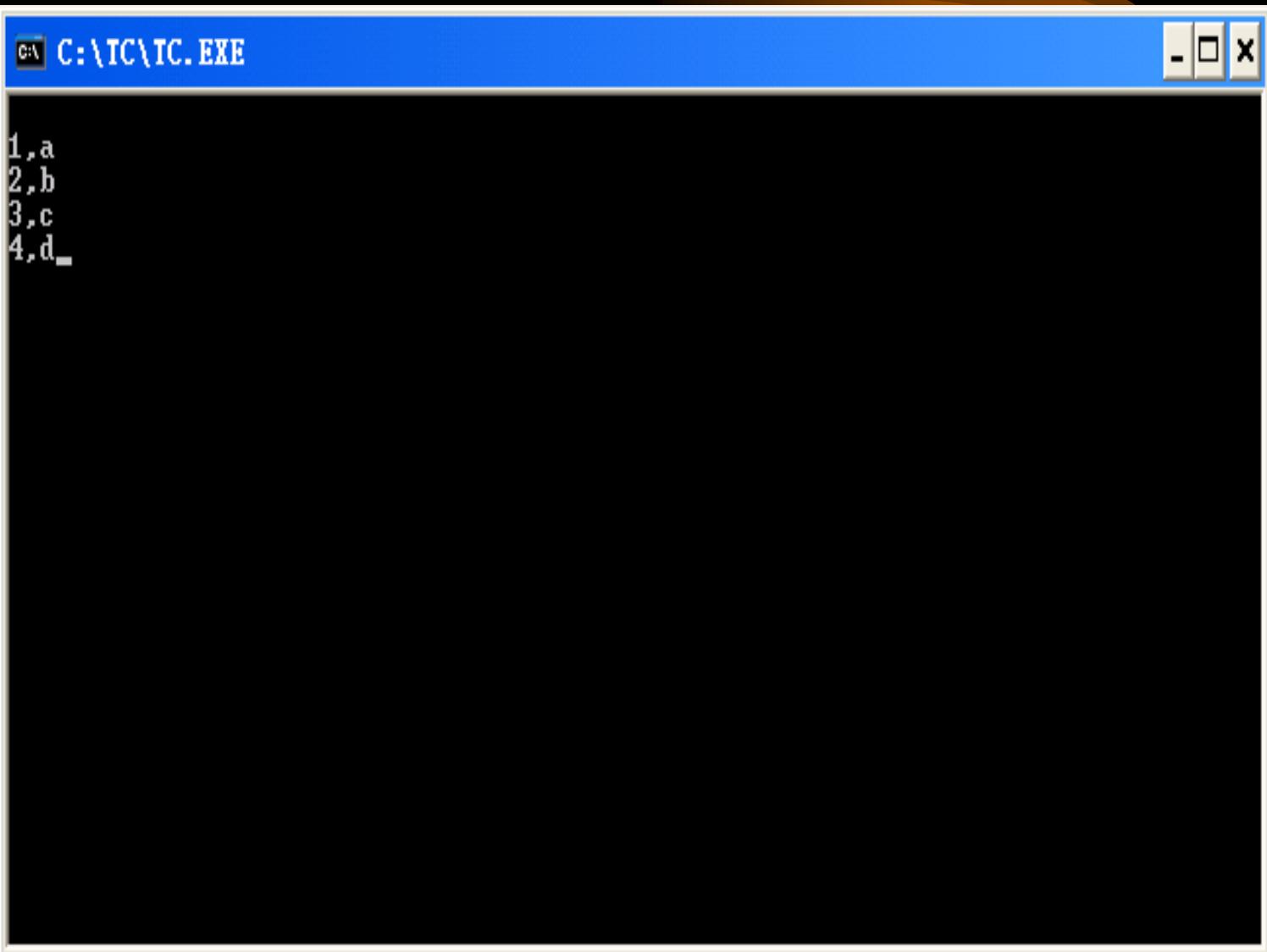
main()
{
    struct data x1,x2,x3,x4,*head;
    x1.x=1;x1.c='a';x2.x=2;x2.c='b';x3.x=3;x3.c='c';x4.x=4;x4.c='d';
    x1.next=&x2;x2.next=&x3;x3.next=&x4;x4.next=NULL;
    head=&x1;
    f(head);
}
```

-

Message



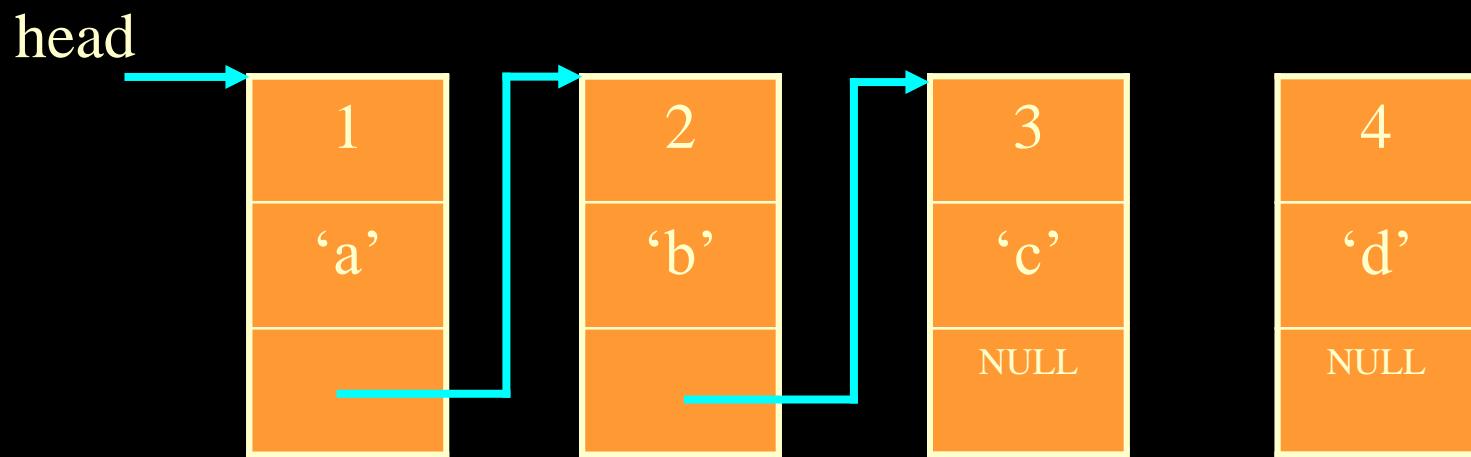
结果



C:\IC\IC.EXE

```
1,a  
2,b  
3,c  
4,d
```

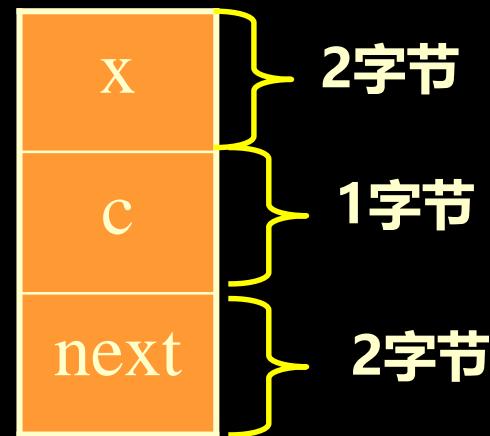
建立动态链表



基本步骤：

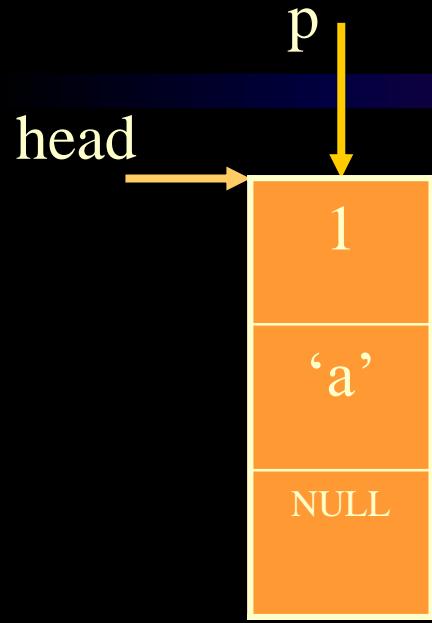
1、**定义结点数据类型，测试该类型所占内存的字节数**

```
struct data  
{ int x;  
char c;  
struct data *next;  
};
```



`len=sizeof (struct data); /*len=2+1+2=5字节*/`

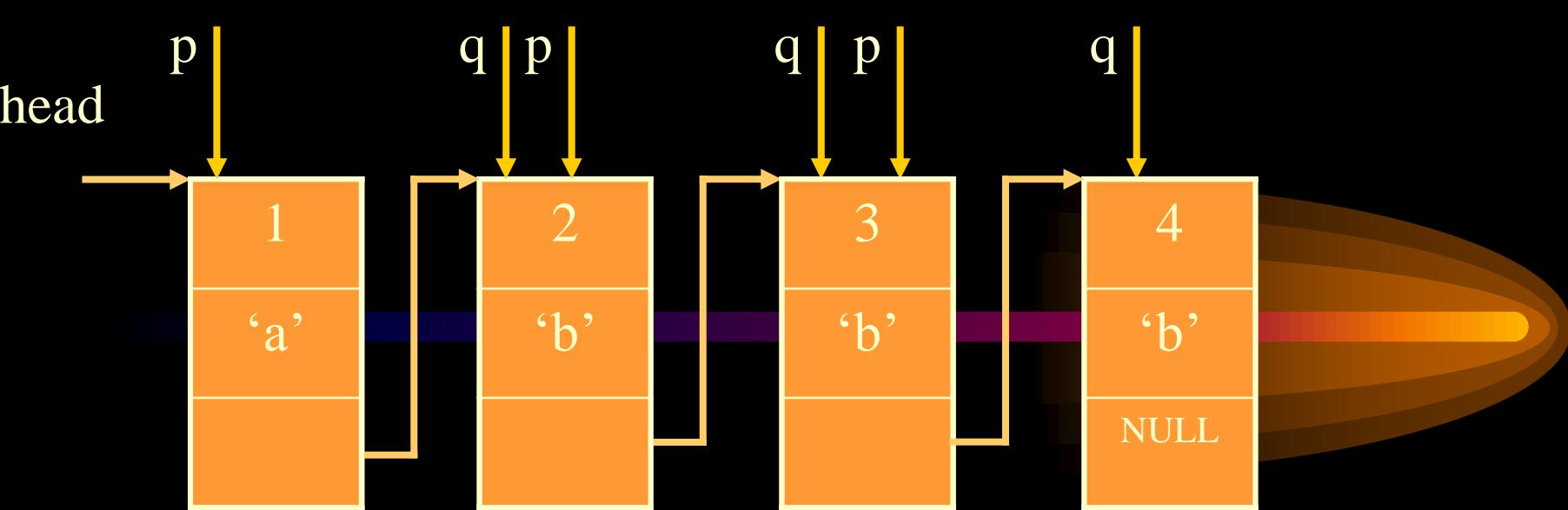
2、申请一个新单元，并赋值



```
p=(struct data *)malloc (len);  
scanf("%d",&p->x);  
scanf("%c",&p->c);  
p->next=NULL;  
head=p;
```

3、保留前一个结点地址

4、重复2-3步



`q=(struct data *)malloc (len);`

`scanf("%d",&q->x);`

`scanf("%c%c",&q->c,&q->c);`

`q->next=NULL;`

`p->next=q;`

`p=q;`

循环条件:

结点个数

输入结束符

.....

/*消化分隔符号*/

完整程序： #include "stdlib.h"

```
#define NULL 0
```

```
struct data
```

```
{int x; char c;struct data *next;};
```

```
main()
```

```
{struct data *head,*p,*q;int i,len;
```

```
len=sizeof(struct data);
```

```
q=(struct data*)malloc(len);
```

```
scanf("%d",&q->x); scanf("%c%c",&q->c,&q->c);
```

```
q->next=NULL; head=p=q;
```

```
for(i=0;i<3;i++)
```

```
{q=(struct data*)malloc(len);scanf("%d",&q->x);
```

```
scanf("%c%c",&q->c,&q->c);q->next=NULL;p->next=q; p=q;}
```

```
p=head;
```

```
while(p!=NULL)
```

```
{printf("\n%d,%c",p->x,p->c);p=p->next;} }
```

File Edit Run Compile Project Options Debug Break/watch

Edit

Line 10 Col 24 Insert Indent Tab Fill Unindent * C:MAJUN2.C

```
#include "stdlib.h"
#define NULL 0
struct data
{int x;char c;struct data *next;};
main()
{struct data *head,*p,*q;int i,len;
 len=sizeof(struct data);
 q=(struct data*)malloc(len);
 scanf("%d",&q->x); scanf("%c%c",&q->c,&q->c);
 q->next=NULL; head=p=q;
 for(i=0;i<3;i++)
 {q=(struct data*)malloc(len);scanf("%d",&q->x);
  scanf("%c%c",&q->c,&q->c);
  q->next=NULL;p->next=q; p=q;}
 p=head;
 while(p!=NULL)
 {printf("\n%d,%c",p->x,p->c);
  p=p->next;} }
```

Watch

C:\IC\IC.EXE



12 a
90 b
21 x
88 y

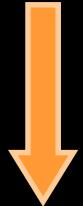
12,a
90,b
21,x
88,y



```
scanf("%d",&q->x);
```

```
scanf("%c%c",&q->c,&q->c);
```

```
q->next=NULL;
```



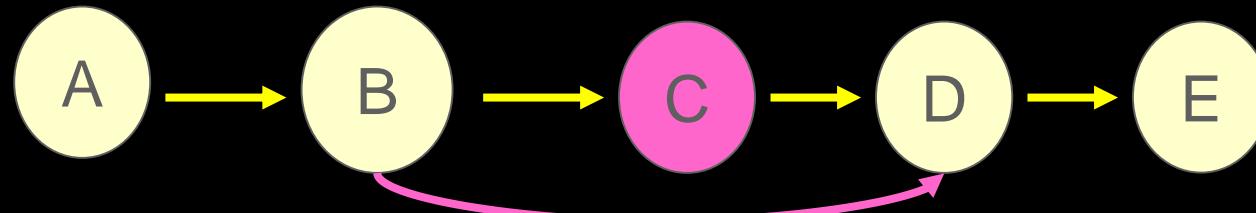
```
scanf("%d",&q->x);
```

```
scanf("%c",&q->c);
```

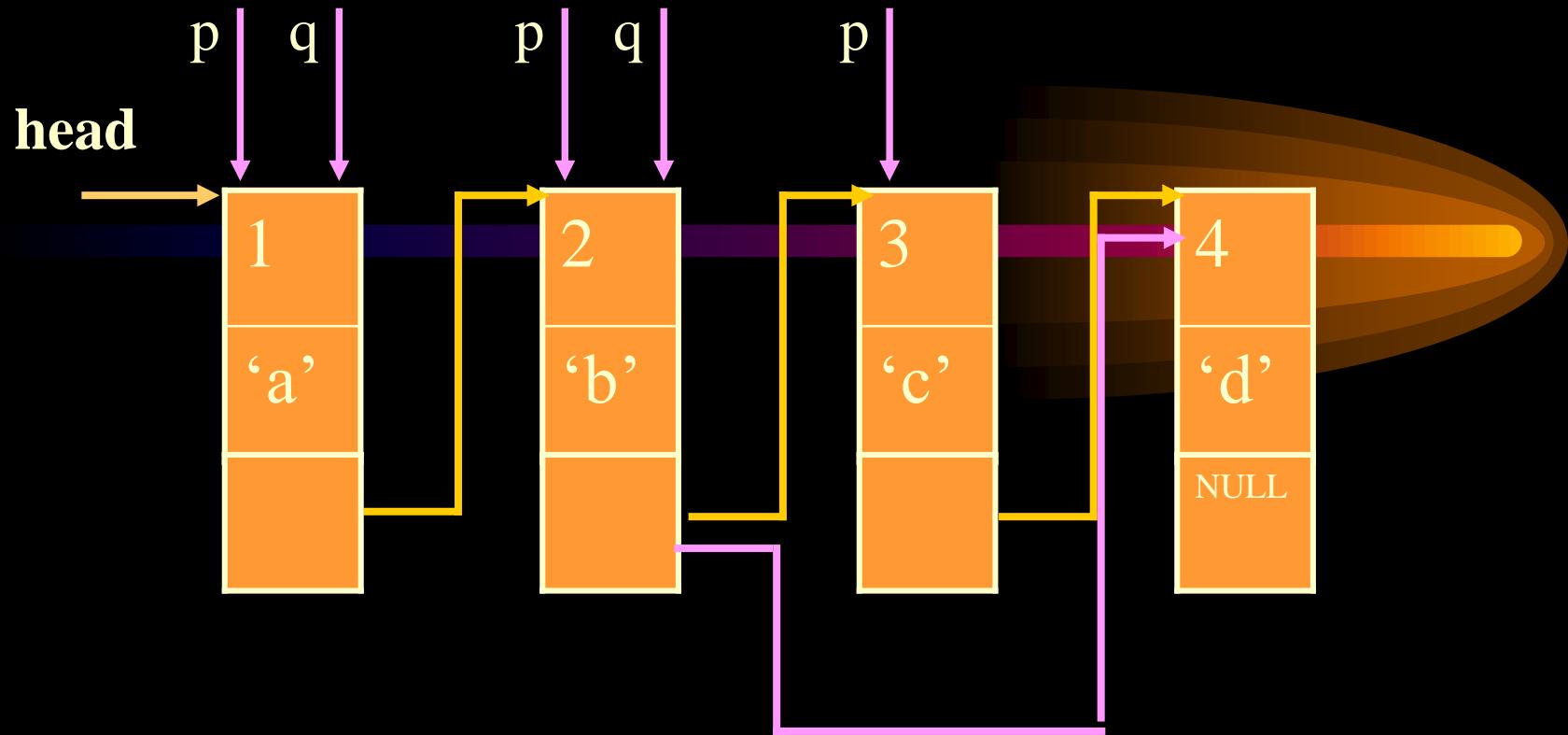
```
q->next=NULL;
```



一队小孩 (A、B、C、D、E) 手拉手，如果某一小孩 (C) 想离队有事，而队形保持不变。只要将C的手从两边脱开，B改为与D拉手即可。



- 1、如何找到要删除的结点？
- 2、如何实现结点的删除？



1. 查找要删除结点(以指定数据值m(3)作为删除结点的标志)

```
while(p->x!=m&&p!=NULL){q=p;p=p->next;}
```

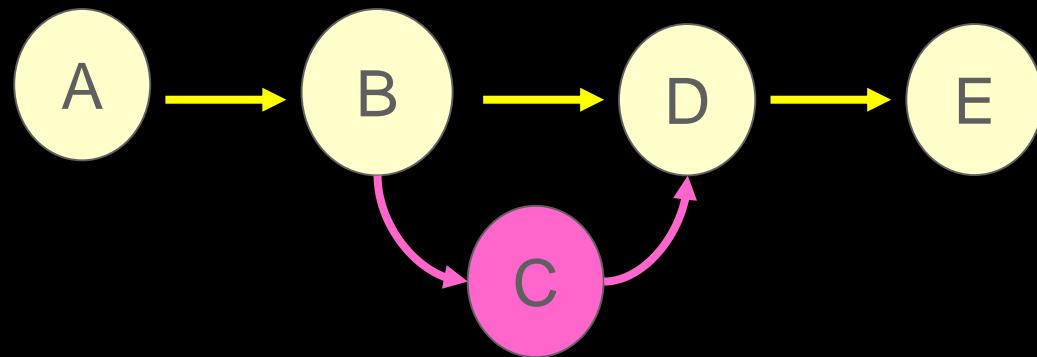
2. 删除结点

```
q->next=p->next; free(p);
```

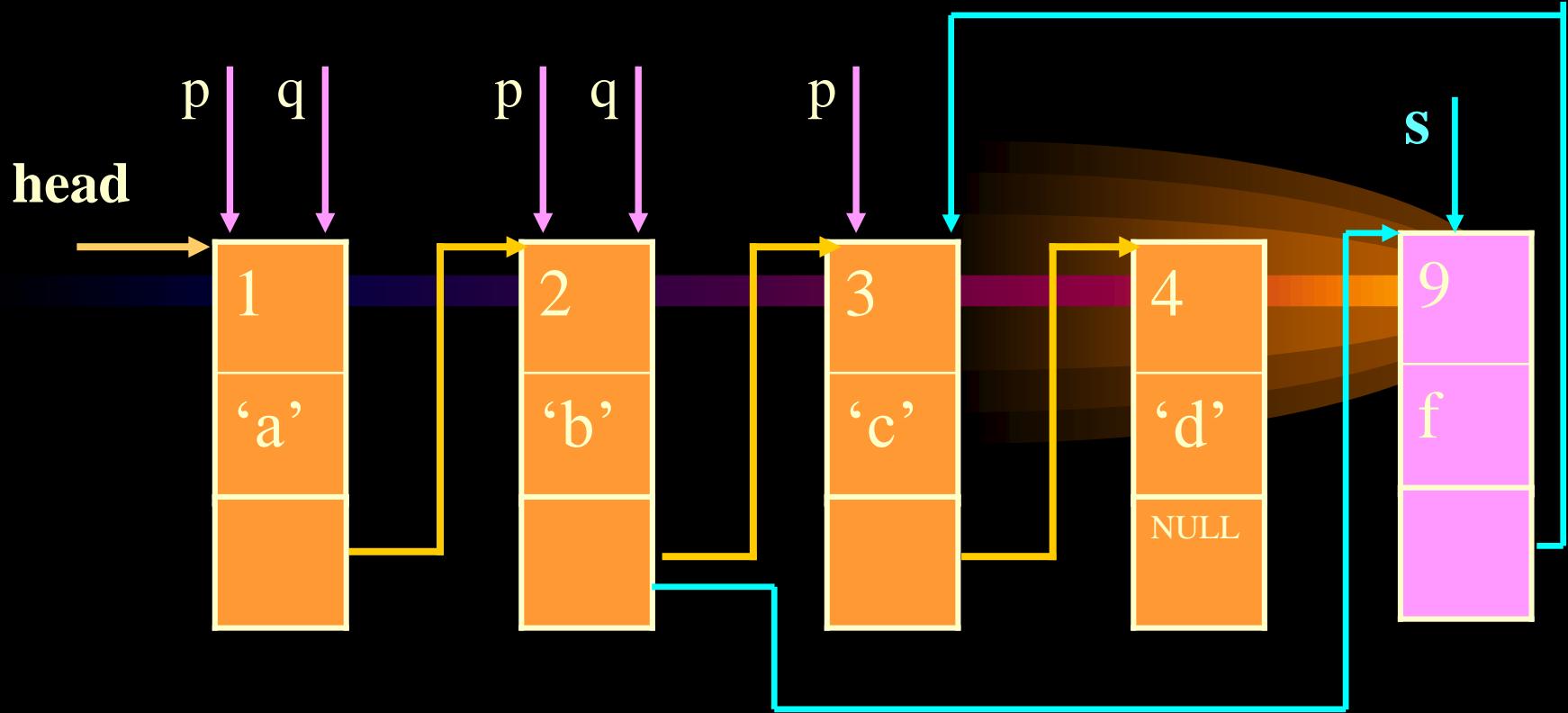
完整程序：

```
void delete(struct data *head,int m)
{struct data *p,*q;
p=head;
while(p->x!=m&&p!=NULL){q=p;p=p->next;}
if (p->x==m){q->next=p->next;    free(p); }
else    printf("error!");      /*没有要删除的结点*/
return;}
```

一队小孩（标号分别为A、B、D、E）手拉手按字母标号顺序排列，如果某一小孩（标号为C）想插入队伍，而标号保持不变。只要将B的手与D脱开，拉C的手，C与D直接拉手即可。



- 1、如何找到插入位置？
- 2、如何实现结点的插入？



13. 确定插入的位置(在第2个结点之后插入值为9和' f'的结点)

```
for(i=0;i<3;i++){q=p;p=p->next;}
```

2. 申请一个新结点并赋值结点

```
s=(struct data *)malloc (len);s->x=9;s->c='f';
```

完整程序：

```
void insert(struct data *head,int m)  
{struct data *p,*q;  
p=head;  
  
for(i=0;i<m;i++){q=p;p=p->next;}  
  
s=(struct data *)malloc (len);  
  
s->x=9;s->c='f';  
  
q->next=s;s->next=p; /*插入结点*/  
  
return;}
```

例：以下程序运行后的输出结果是 D。

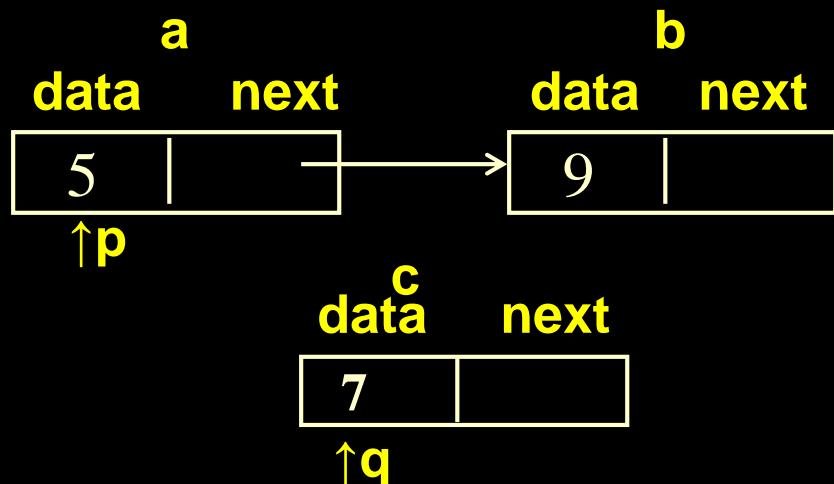
```
#include <stdlib.h>
struct NODE
{
    int num;
    struct NODE *next;
}
main()
{
    struct NODE *p,*q,*r;
    p=(struct NODE *)malloc(sizeof(struct NODE));
    q=(struct NODE *)malloc(sizeof(struct NODE));
    r=(struct NODE *)malloc(sizeof(struct NODE));
    p->num=10;q->num=20;r->num=30;
    p->next=q;q->next=r;
    printf("%d\n",p->num+q->next->num);
}
```

A) 10 B) 20 C) 30 D) 40

例：若以下定义：

```
struct link  
{ int data;  
    struct link *next;  
}a,b,c,*p,*q;
```

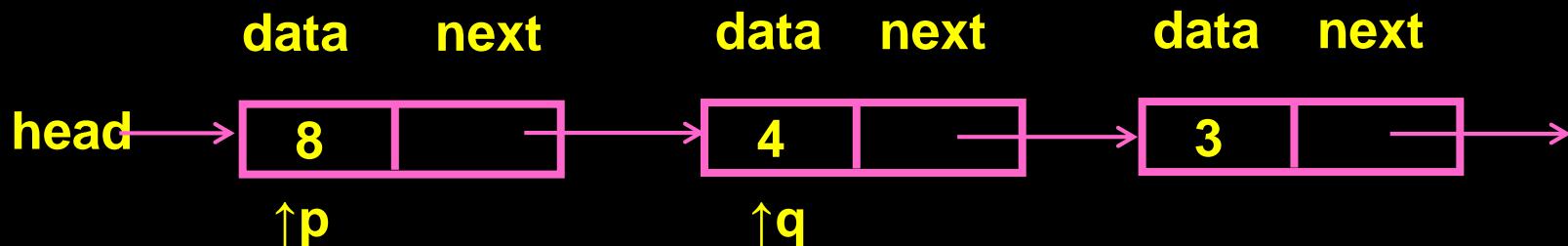
且变量a和b之间已有如下图所示的链表结构：指针p指向变量a，
q指向变量c。则能够把c插入到a和b 之间并形成新的链表的
语句组是 D

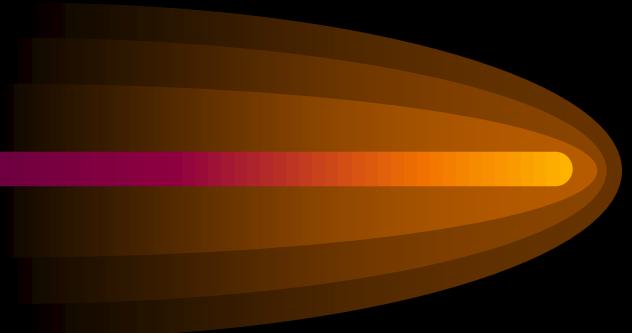


- A) a.next=c; c.next=b;
- B) p.next=q; q.next=p.next;
- C) p->next=&c; q->next=p->next;
- D) (*p).next=q; (*q).next=&b;

例：假定建立了以下链表结构，指针p、q分别指向如图所示的结点，则以下可以将q所指结点从链表中删除并释放该结点的语句组是 B

- A) free(q); p->next=q->next;
- B) (*p).next=(*q).next; free(q);
- C) q=(*q).next; (*p).next=q; free(q);
- D) q=q->next; p->next=q; p=p->next; free(p);





作业:

P318 11.1 11.3 11.8

11.8 共用体

几个不同的变量共占同一段内存的结构，称为**共用体**

定义形式： union 共用体名

{成员1；

⋮

成员n；

}变量名表列；

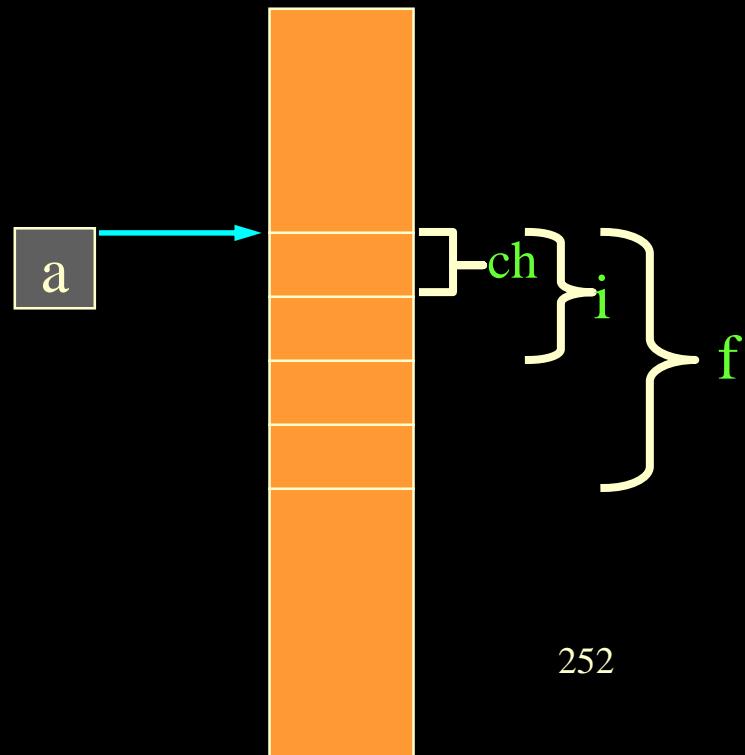
union data

{int i;

char ch;

float f;

}a;



共用体成员的引用格式与结构体成员的引用一样。

同一个内存段可以用来存放几种不同类型的成员，但在每一瞬间时只能存放其中一种，而不是同时存放几种。

共用体变量中起作用的成员是最后一次存入的成员，在存放一个新的成员后原来的成员就失去作用。

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。

例1:写出下列程序的运行结果(字符0的ASCII码为16进制3

```
main()
```

```
{union
```

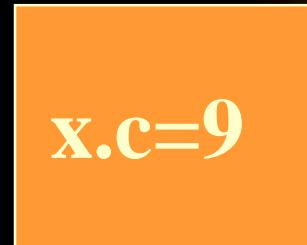
```
    {char c;  
     char i[4];  
     }x;
```

```
x.i[0]=0x39;
```

```
x.i[1]=0x36;
```

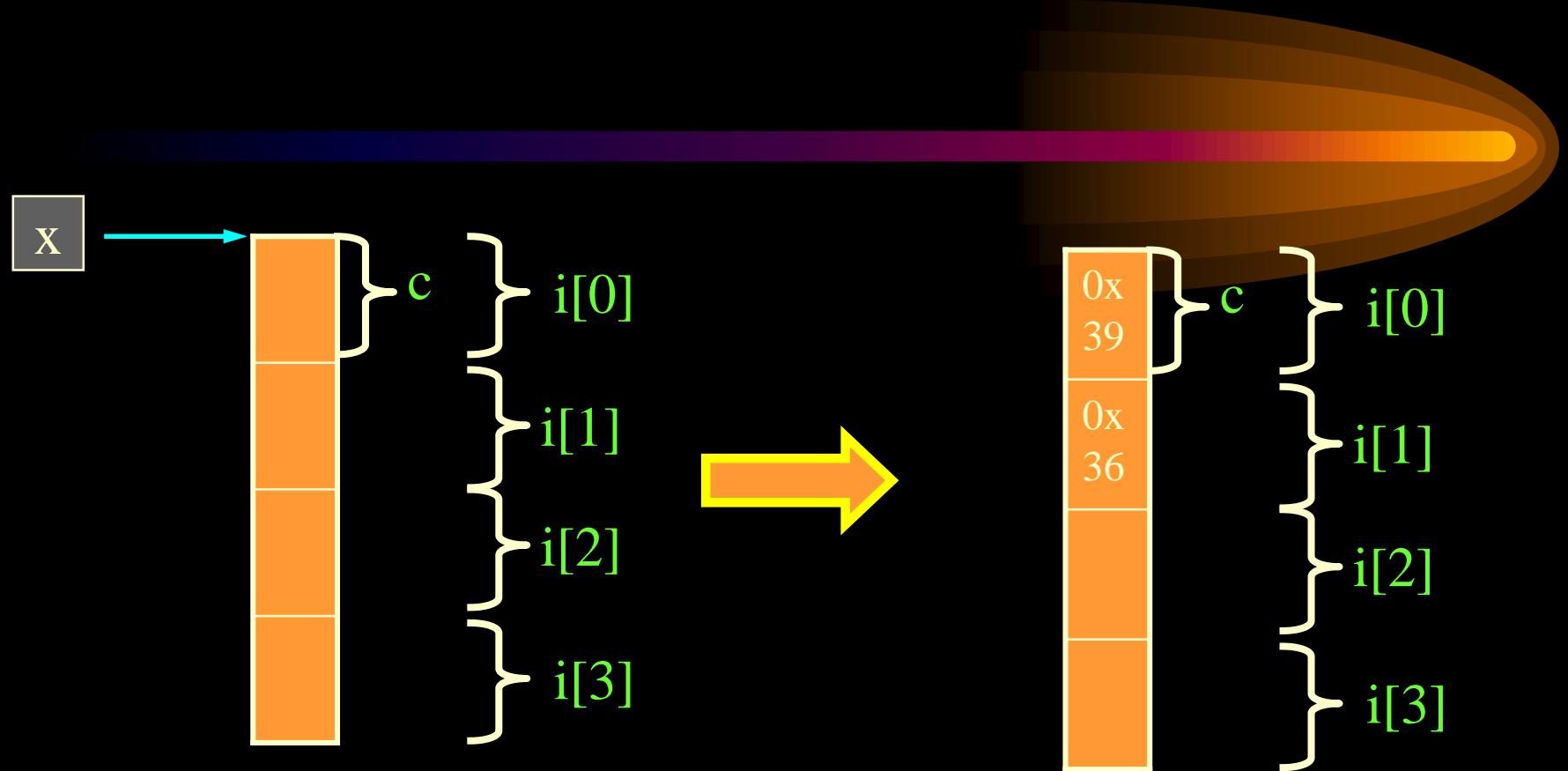
```
printf("%c\n",x.c);
```

```
}
```



```
x.c=9
```



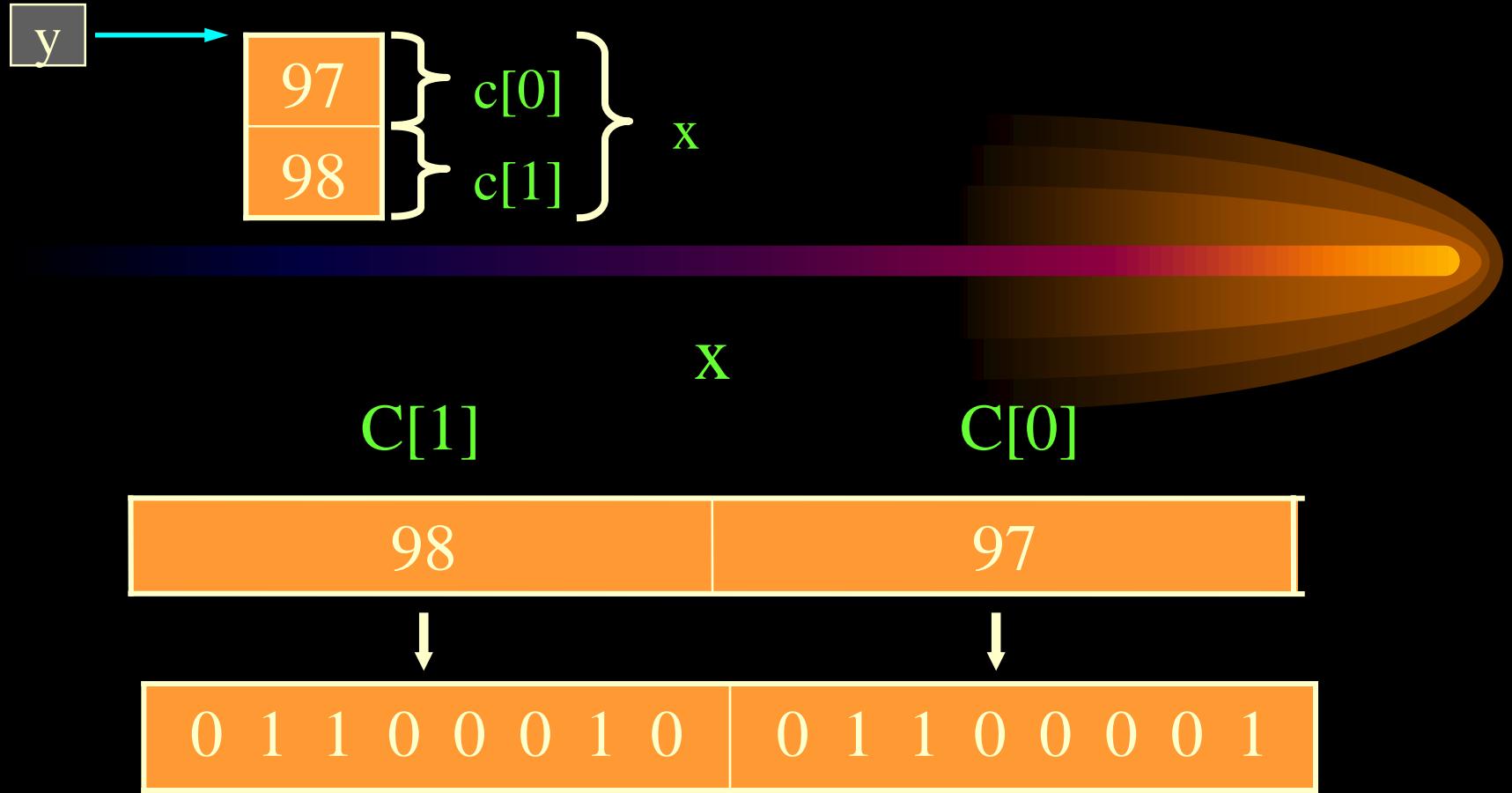


例2：写出下列程序的运行结果

```
main()
{union
    {int x;
     char c[2];
    }y;
y.c[0]='a';
y.c[1]='b';
printf("\ny.x=%d",y.x);
}
```

y.x=25185





$$2^0 + 2^5 + 2^6 + 2^9 + 2^{13} + 2^{14} = \mathbf{25185}$$

例3：某单位对职工进行计算机能力考核,规定35岁(含35岁)以下进行笔试,成绩为百分制;35岁以上进行上机考核,成绩为A、B、C,C为不及格.编写程序,输入10名职工的考核结果,输出及格者的编号、姓名和成绩.

```
struct f
{int num;
char name[10];
int age;
int score;
};
```

```
union score
{int x;
char c;
};
```

```
struct f
{int num;
char name[10];
int age;
union score xy;
};
```

main()

```
{struct f s[10],*p;  
int i;  
for(i=0;i<10;i++)  
{printf("\n\tname:");  
scanf("%d",&s[i].num);  
printf("\tname:");  
scanf("%s",s[i].name);  
printf("\tage:");  
scanf("%d",&s[i].age);  
printf("\tscore:");
```

```
if (s[i].age<=35)
    scanf("%d",&s[i].xy.x);
else
    scanf("%c%c",&s[i].xy.c,&s[i].xy.c);}
```

```
for(p=s;p<s+10;p++)
{ if ((p->age<=35&&p->xy.x>=60)
    ||(p->age>35&&p->xy.c!='C') )
    printf("\n%d:",p->num);
    printf("\t%s:",p->name);
    if (p->age<=35) printf("\t%d",p->xy.x);
    else printf("\t%c",p->xy.c);}}
```

結果

```
C:\> C:\TC\TC.EXE
num:1
name:a1
age:23
score:98

num:2
name:a2
age:45
score:a

num:3
name:a3
age:35
score:45

num:4
name:a4
age:50
score:c

1:      a1:      98
2:      a2:      a
```

11.9 枚举类型

定义：

enum 枚举类型名{枚举常量表}；

例如：

enum f {x=5,y=7,z=9};

enum f a,b;

a=x;b=z;

a=0 **×** a=(enum f)0;

11.10 自定义类型

```
typedef int f;  
int x,y;  
f x,y;  
type struct student pp;
```



第十二章 位运算

• 主要内容



12.1位运算符和位运算

12.2位运算举例

12.3位段

概念

- 位运算是指按二进制位进行的运算。因为在系统软件中，常要处理二进制位的问题。
- 例如：将一个存储单元中的各二进制位左移或右移一位，两个数按位相加等。
- C 语言提供位运算的功能，与其他高级语言（如PASCAL）相比，具有很大的优越性。

12.1 位运算符和位运算

C语言提供的位运算符有：

运算符	含义	运算符	含义
&	按位与	~	取反
	按位或	<<	左移
^	按位异或	>>	右移

说明：

- (1)位运算符中除 ~ 以外，均为二目（元）运算符，即要求两侧各有一个运算量。
- (2)运算量只能是整型或字符型的数据，不能为实型数据。

12.1.1“按位与”运算符 (&)

按位与是指：参加运算的两个数据，按二进制位进行“与”运算。如果两个相应的二进制位都为1，则该位的结果值为1；否则为0。即：

$$0 \& 0 = 0, 0 \& 1 = 0, 1 \& 0 = 0, 1 \& 1 = 1$$

例： 3 & 5 并不等于 8，应该是按位与运算：

$$\begin{array}{r} 00000011(3) \\ \& 00000101(5) \\ \hline 00000001(1) \end{array}$$

3&5的值得 1

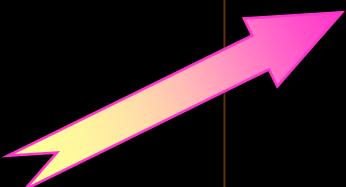
注意：如果参加&运算是负数（如-3&-5），则要以补码形式表示为二进制数，然后再按位进行“与”运算。

按位与的用途： (1) 清零。

若想对一个存储单元清零，即使其全部二进制位为 0，只要找一个二进制数，其中各个位符合以下条件：原来的数中为 1 的位，新数中相应位为 0。然后使二者进行 & 运算，即可达到清零目的。

例：原有数为 0 0 1 0 1 0 1 1，另找一个数，设它为 1 0 0 1 0 1 0 0，这样在原数为 1 的位置上，该数的相应位值均为 0。将这两个数进行 & 运算：

$$\begin{array}{r} 00101011 \\ \& 10010100 \\ \hline 00000000 \end{array}$$



(2) 取一个数中某些指定位。

如有一个整数 a (2个字节)，想要取其中的低字节，只需将 a 与8个1按位与即可。

a	00101100	10101100
b	00000000	11111111
c	00000000	10101100

(3) 保留一位的方法：与一个数进行&运算，此数在该位取1。

例：有一数01010100，想把其中左面第3、4、5、7、8位保留下，运算如下：

$$\begin{array}{r} 01010100(84) \\ \& 00111011(59) \\ \hline 00010000(16) \end{array}$$

即： $a=84, b=59$
 $c=a \& b=16$

12.1.2 “按位或” 运算符 (|)

两个相应的二进制位中只要有一个为 1，该位的结果值为 1。

即 $0|0 = 0$, $0|1 = 1$, $1|0 = 1$, $1|1 = 1$

例： 060|017, 将八进制数60与八进制数17进行按位或运算。

$$\begin{array}{r} 00110000 \\ | 00001111 \\ \hline 00111111 \end{array}$$

应用：按位或运算常用来对一个数据的某些位定值为 1。例如：如果想使一个数 a 的低 4 位改为 1，只需将 a 与 0 1 7 进行按位或运算即可。

例： a 是一个整数（16位），
有表达式： $a \mid 0377$
则低 8 位全置为 1，高 8 位保留原样。

12.1.3“异或”运算符 (\wedge)

异或运算符 \wedge 也称XOR运算符。它的规则是：

若参加运算的两个二进制位同号则结果为 0 (假)

异号则结果为 1 (真)

即: $0 \wedge 0 = 0$, $0 \wedge 1 = 1$, $1 \wedge 0 = 1$, $1 \wedge 1 = 0$

例:

$$\begin{array}{r} 00111001 \\ \wedge 00101010 \\ \hline 00010011 \end{array}$$

即: $071 \wedge 052 = 023$ (八进制数)

^运算符应用：

(1) 使特定位翻转

设有 01111010，想使其低 4 位翻转，即 1 变为 0，0 变为 1。可以将它与 00001111 进行 ^ 运算，即：

运算结果的低 4 位正好是原数低 4 位的翻转。可见，要使哪几位翻转就将与其进行 ^ 运算的该几位置为 1 即可。


$$\begin{array}{r} 01111010 \\ \wedge 00001111 \\ \hline 01110101 \end{array}$$

(2) 与 0 相 \wedge ，保留原值

例如： $012 \wedge 00 = 012$

$$\begin{array}{r} 00001010 \\ \wedge 00000000 \\ \hline 00001010 \end{array}$$

因为原数中的 1 与 0 进行 \wedge 运算得 1， $0 \wedge 0$ 得 0，故保留原数。

(3) 交换两个值，不用临时变量

例如： $a = 3$, $b = 4$ 。

想将 a 和 b 的值互换，可以用以下赋值语句实现：

$a = a \wedge b;$

$b = b \wedge a;$

$a = a \wedge b;$

$a = 0 \ 1 \ 1$

$(\wedge) \ b = 1 \ 0 \ 0$

$a = 1 \ 1 \ 1$ ($a \wedge b$ 的结果， a 已变成 7)

$(\wedge) \ b = 1 \ 0 \ 0$

$b = 0 \ 1 \ 1$ ($b \wedge a$ 的结果， b 已变成 3)

$(\wedge) \ a = 1 \ 1 \ 1$

$a = 1 \ 0 \ 0$ ($a \wedge b$ 的结果， a 已变成 4)

即等效于以下两步：

- ① 执行前两个赋值语句：“ $a = a \wedge b;$ ”和“ $b = b \wedge a;$ ”相当于 $b=b \wedge (a \wedge b)$ 。
- ② 再执行第三个赋值语句： $a = a \wedge b$ 。由于 a 的值等于 $(a \wedge b)$ ， b 的值等于 $(b \wedge a \wedge b)$ ，因此，相当于 $a=a \wedge b \wedge b \wedge a \wedge b$ ，即 a 的值等于 $a \wedge a \wedge b \wedge b \wedge b$ ，等于 b 。

a 得到 b 原来的值。

12.1.4 “取反” 运算符 (~)

~是一个单目（元）运算符，用来对一个二进制数按位取反，即将0变1，将1变0。例如， ~ 025 是对八进制数25（即二进制数00010101）按位求反。

000000000010101

(~)

111111111101010 (八进制数177752)

12.1.5 左移运算符 ($<<$)

左移运算符是用来将一个数的各二进制位全部左移若干位。

例如： $a = <<2$ 将 a 的二进制数左移 2 位，右补 0。

若 $a = 15$ ，即二进制数 00001111，

左移 2 位得 00111000，(十进制数 60)

高位左移后溢出，舍弃。

12.1.5 左移运算符 ($<<$)

左移 1 位相当于该数乘以 2，左移 2 位相当于该数乘以 $2^2 = 4$, $15 << 2 = 60$ ，即乘了 4。但此结论只适用于该数左移时被溢出舍弃的高位中不包含 1 的情况。

假设以一个字节（8 位）存一个整数，若 a 为无符号整型变量，则 $a = 64$ 时，左移一位时溢出的是 0，而左移 2 位时，溢出的高位中包含 1。

12.1.6 右移运算符(>>)

右移运算符是 $a >> 2$ 表示将a的各二进制位右移2位，移到右端的低位被舍弃,对无符号数,高位补0。

例如：a=017时：

a的值用二进制形式表示为00001111， 舍弃低2位11：
 $a >> 2 = 00000011$

右移一位相当于除以2

右移n位相当于除以 2^n 。

在右移时,需要注意符号位问题:

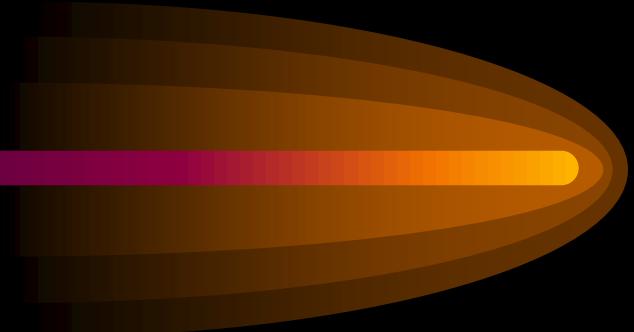
对无符号数,右移时左边高位移入0; 对于有符号的值,如果原来符号位为0(该数为正),则左边也是移入0。如果符号位原来为1(即负数),则左边移入0还是1,要取决于所用的计算机系统。有的系统移入0,有的系统移入1。移入0的称为“逻辑右移”,即简单右移; 移入1的称为“算术右移”。

例： a的值是八进制数113755：

a:1001011111101101 (用二进制形式表示)

a>>1: 0100101111110110 (逻辑右移时)

a>>1: 1100101111110110 (算术右移时)



在有些系统中,a>>1得八进制数045766,而在另一些系统上可能得到的是145766。Turbo C和其他一些C编译采用的是算术右移,即对有符号数右移时,如果符号位原来为1,左面移入高位的是1。

12.1.7 位运算赋值运算符

位运算符与赋值运算符可以组成复合赋值运算符。

例如: $\&=$, $|=$, $>>=$, $<<=$, $\wedge=$

例: $a \&= b$ 相当于 $a = a \& b$

$a <<= 2$ 相当于 $a = a << 2$

12.1.8 不同长度的数据进行位运算

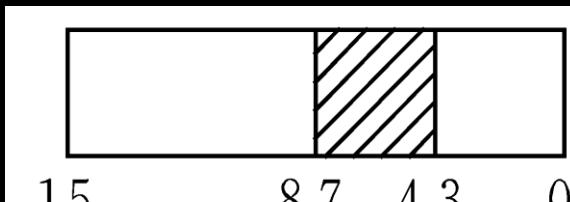
如果两个数据长度不同(例如long型和int型), 进行位运算时(如`a & b`, 而`a`为long型,`b`为int型), 系统会将二者按右端对齐。如果`b`为正数, 则左侧16位补满0; 若`b`为负数, 左端应补满1; 如果`b`为无符号整数型, 则左侧添满0。

12.2 位运算举例

例12.1 取一个整数a从右端开始的4~7位

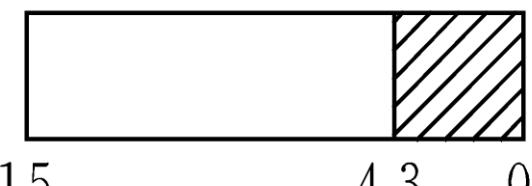
① 先使a右移4位： $a >> 4$

目的是使要取出的那几位移到最右端



(a)

未右移时的情况



(b)

右移4位后的情况

② 设置一个低4位全为1,其余全为0的数。

$\sim(\sim 0 << 4)$

③ 将上面①、②进行&运算。

$(a >> 4) \& \sim(\sim 0 << 4)$

程序如下：

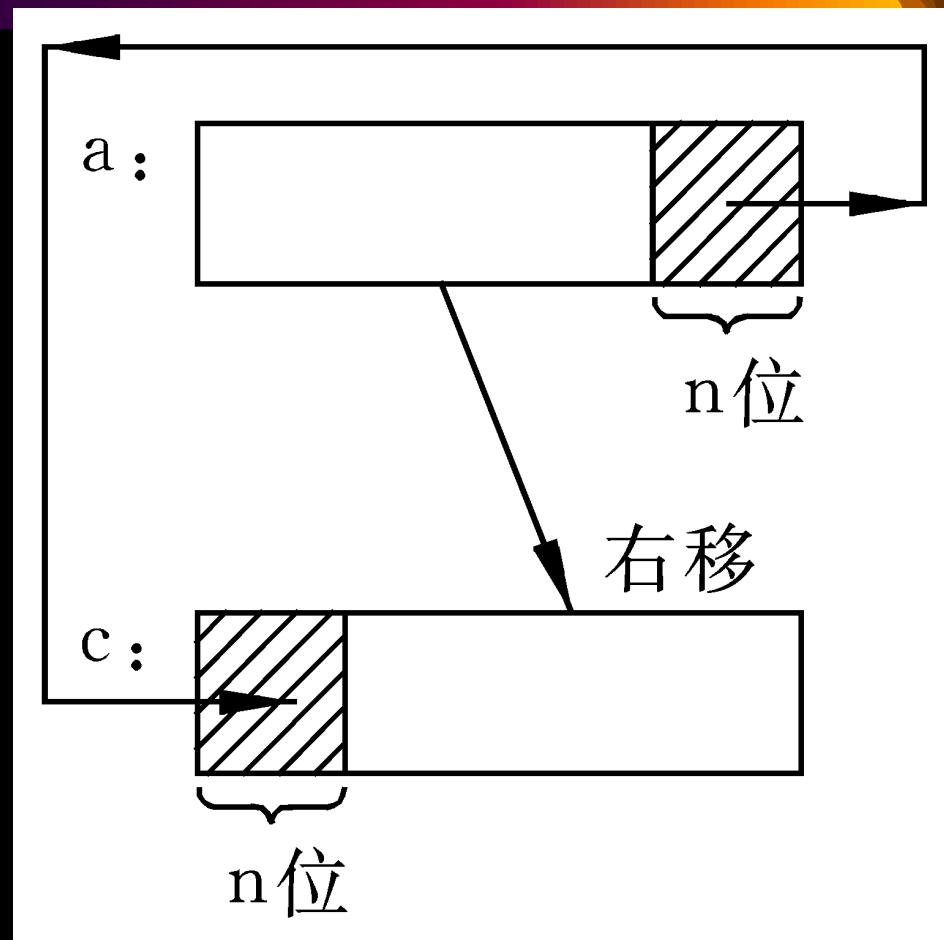
```
#include <stdio.h>
void main()
{ unsigned a,b,c,d;
scanf("%o",&a);
b=a>>4;
c=~(~0<<4);
d=b&c;
printf("%o,%d\n",
```

运行情况如下： 3 3 1 (输入)
3 3 1 , 217 (a 的值)
1 5 , 13 (d 的值)
输入 a 的值为八进制数331,
其二进制形式为11011001
经运算最后得到的d为00001101
即八进制数 1 5 , 十进制数13。

例12.2 循环移位。

要求将 a 进行右循环移位

将 a 右循环移 n 位，
即将 a 中原来左面
 $(16 - n)$ 位右
移 n 位，原来右端
n 位移到最左面 n
位。



步骤：

① 将 a 的右端 n 位先放到 b 中的高 n 位中，

实现语句： $b = a \ll (16 - n);$

② 将 a 右移 n 位， 其左面高位 n 位补 0，

实现语句： $c = a \gg n;$

③ 将 c 与 b 进行按位或运算， 即 $c = c | b;$

程序如下：

```
#include <stdio.h>
void main()
{ unsigned a,b,c;
int n;
scanf("a=%o,n=%d",&a,&n);
b=a<<(16-n);
c=a>>n;
c=c|b;
printf("%o\n%o",a,c);
}
```

运行情况如下：

```
a = 1 5 7 6 5 3, n = 3
1 5 7 6 5 3
7 5 7 6 5
```

运行开始时输入八进制数157653，即二进制数

110111111010101

1

循环右移3位后得二进制数

011110111111010

1

12.3 位段

信息的存取一般以字节为单位。实际上，有时存储一个信息不必用一个或多个字节，例如，“真”或“假”用 0 或 1 表示，只需 1 位即可。在计算机用于过程控制、参数检测或数据通信领域时，控制信息往往只占一个字节中的一个或几个二进制位，常常在一个字节中放几个信息。

怎样向一个字节中的一个或几个二进制位赋值和改变它的值呢？可以用以下两种方法：

(1) 可以人为地将一个整型变量data分为几部分。

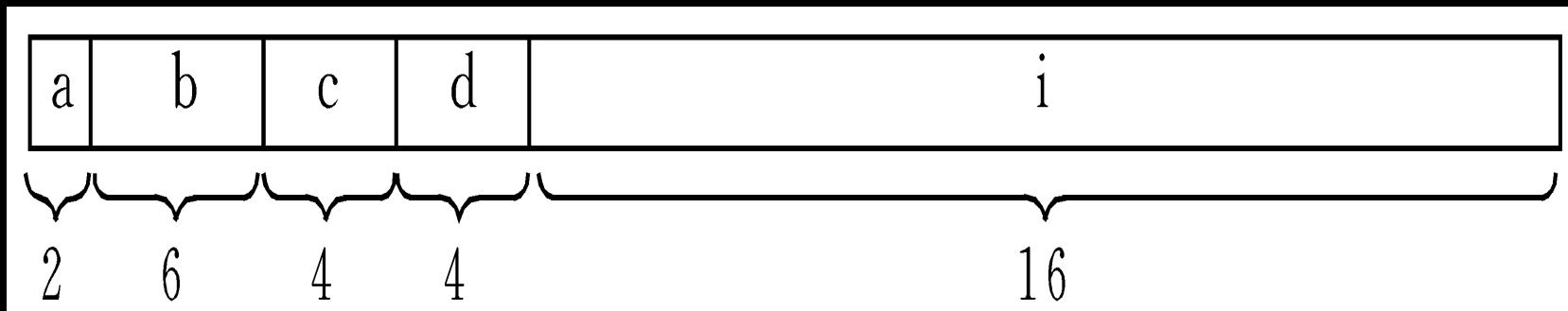
但是用这种方法给一个字节中某几位赋值太麻烦。可以位段结构体的方法。

(2) 位段

C语言允许在一个结构体中以位为单位来指定其成员所占内存长度，这种以位为单位的成员称为“位段”或称“位域”(bit field)。利用位段能够用较少的位数存储数据。

程序如下：

```
struct packed-data
{ unsigned a : 2;
  unsigned b : 6;
  unsigned c : 4;
  unsigned d : 4;
  int i ;
} data;
```



关于位段的定义和引用的说明：

- (1) 位段成员的类型必须指定为unsigned或int类型。
- (2) 若某一位段要从另一个字开始存放，可用以下形式定义：

 unsigned a : 1;

 unsigned b: 2 ; 一个存储单元

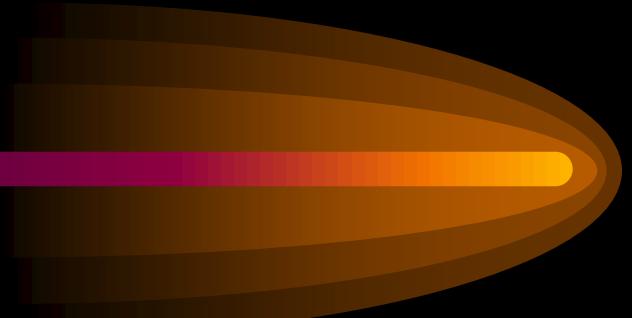
 unsigned: 0 ;

 unsigned c : 3 ; 另一存储单元

a、b、c应连续存放在一个存储单元中，由于用了长度为0的位段，其作用是使下一个位段从下一个存储单元开始存放。因此，只将a、b存储在一个存储单元中，c另存在下一个单元(“存储单元”可能是一个字节，也可能是2个字节，视不同的编译系统而异)。

关于位段的定义和引用的说明：

- (3) 一个位段必须存储在同一存储单元中，不能跨两个单元。如果第一个单元空间不能容纳下一个位段，则该空间不用，而从下一个单元起存放该位段。
- (4) 可以定义无名位段。
- (5) 位段的长度不能大于存储单元的长度，也不能定义位段数组。
- (6) 位段可以用整型格式符输出。
- (7) 位段可以在数值表达式中引用，它会被系统自动地转换成整型数。



第十三章 文件

1.文件概述

文件：一般指存储在外部介质上的有组织的数据集合。

C语言支持的是流式文件，它把文件看做由一个一个的字符（字节）数据组成的序列。

根据数据的组织和操作形式，可分为ASCII文件和二进制文件。

①二进制文件和文本文件

二进制文件是将内存中的数据按其在内存中的存储形式原样保存在文件中。

因此整型数127在二进制文件中用2个字节保存为：

00000000	01111111
----------	----------

(整型数127)

文本文件是将数据按字符形式存放在文件中。

因此整型数127在文本文件中用3个字节存放：

00110001	00110010	00110111
----------	----------	----------

(字符 ‘1’) (字符 ‘2’) (字符 ‘7’)

②文件操作的基本思路

(1)首先使用库函数fopen()打开该文件。

fopen()的作用是建立文件和操作系统之间的连接，进行必要的通信（实现所谓对文件的按名操作），并返回一个在以后可以用于文件读/写操作的指针。

(2)文件被打开后，需要进行文件的读/写。

以库函数fgetc()和fputc()最为简单。fgetc()的作用是从文件读取一个字符，然后将其存放到用户自己定义的变量中；fputc()的作用是将一个字符写到文件中。

(3)操作完成后，关闭文件。

使用库函数fclose()关闭文件。该操作将释放不再需要的文件指针，并能保证信息被完整地存储。

2.文件基本操作

①文件指针

文件指针是一个指向FILE类型结构体的指针变量。可以使指向某一个文件的结构体变量，从而通过该结构变量中的文件信息能够访问该文件。

②文件指针的定义

FILE *标识符1,*标识符2,.....,*标识符n;

例如： FILE *fp;

③文件的打开与关闭

对文件作任何读写操作之前必须“打开”该文件。在程序读写操作结束后必须“关闭”该文件。

(1)文件的打开(fopen函数)

C语言用fopen()函数来实现打开文件。

用fopen函数打开一个指定文件后，返回一个指向该文件的指针，在程序中通过这个指针来实现对文件的读写操作

fopen函数的使用形式为：

`fp=fopen(文件名, 文件打开方式);`

如果函数调用成功，fopen函数的返回值是指向该文件的指针，程序可以使用这个指针对所打开的文件进行读写操作。否则返回一个空指针—NULL；

文件使用方式：

文件使用方式	含义
“r”	只读 为输入打开一个文本文件进行读操作
“w” 只写	为输出打开一个文本文件进行写操作
“a”	追加 向文本文件尾追加数据
“rb” 只读	为输入打开一个二进制文件进行读操作
“wb” 只写	为输出打开一个二进制文件进行写操作
“ab” 追加	向二进制文件尾追加数据
“r+”	读写 为读/写打开一个文本文件
“w+”	读写 为读/写建立一个新的文本文件
“a+” 读写	同” r+”
“rb+” 读写	为读/写打开打开一个二进制文件
“wb+” 读写	为读/写建立一个新的二进制文件，若文件不存在则创建
“ab+” 读写	同” r+”

(2)文件的关闭(fclose函数)

fclose函数用来关闭fp所指向的文件。该文件必须是用fopen函数打开的。如果关闭成功则返回1，否则返回0。

fclose函数的一般使用方式：

```
fclose(文件指针);
```

如果打开文件失败，
则退出程序

进行读写操作

```
main()
{
FILE * fp;
fp = fopen("d:\\text.c", "w");
if (fp == NULL)
{
printf("cannot open this file!\n");
exit(0);
}
:
fclose(fp);
}
```

关闭文件

④文件的基本读写 fputc和fgetc函数:

fputc函数的功能是将一个字符写入文件的当前位置。
一般使用形式为：

```
fputc(ch, fp);
```

其中ch是要输出的字符，它可以是一个字符常量，也可以是一个字符变量。fp是文件指针变量，它从函数得到返回值。如果函数调用成功则返回ch的值，否则返回EOF。

fgetc函数从文件中读取当前位置的一个字符返回。
其一般形式为：

```
ch=fgetc(fp);
```

字符变量

文件型指针变量

例： 读入字符并存入文件，直到用户输入一个“#”符为止。

```
#include "stdio.h"  
main()  
{  
    char filename[20];  
    FILE *fp;  
    char ch;  
    scanf("%s",filename);  
    if((fp=fopen(filename, "w"))==NULL)  
    {  
        printf("can't create the file\n");  
        exit(0);  
    }  
    while((ch=getchar())!='#')  
        fputc(ch,fp);  
    fclose(fp);  
}
```

例： 从一个磁盘文件顺序读入字符并在屏幕上显示出来。

```
#include "stdio.h"
main()
{
    FILE *fp;
    char ch;
    if((fp=fopen("d:\\my.dat","r"))==NULL)
    { printf("\n this file does not exist\n");
        exit(1);
    }
    while((ch=fgetc(fp))!=EOF)    putchar(ch);
    fclose(fp);
}
```

⑤文件结束判断函数

C 语言中使用feof函数来判断文件是否结束。如果是文件结束，函数feof(fp)的值为1(真)，否则为0(假)。

例： 打开一个ASCII文件，将文件内容显示到显示器上。然后输入一行字符串，将其保存到该文件中。

```
#include "stdio.h"
main()
{FILE *fp;
 char c,str[100], filename[30],i=0;
 scanf("%s",filename);
 if( (fp=fopen(filename, "r+")) == NULL)
 {printf("file can't open!\n");
 exit(0); }
 while((c=fgetc(fp))!=EOF)
 putchar(c);
 gets(s);
 while(str[i]!='\0')
 {fputc(str[i], fp); i++; }
 fclose(fp);
}
```

例：有两个磁盘文件x和y，各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），结果输出到一个新文件z中。

```
#include 'stdio.h'

void get_inf(FILE,*fp,char str[],int *n)      /*读文件*/
{
    while(!feof(fp)) str[(*n)++] = fgetc(fp);

}

void sort_inf(char str[],int n)                  /*对内容进行排序*/
{
    int i,j;
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
            if(str[i]>str[j])
                {t=str[i];str[i]=str[j];str[j]=t;}
}
```

文件高级操作

1.字符串级的文件读/写

所谓字符串级的文件读/写，指文件的读/写单位是字符串。
字符串级的文件读/写函数主要有： fgets() 函数和 fputs() 函数。

①fgets()函数

fgets()函数的一般形式为：

```
char *fgets ( char *str, int n, FILE *fp);
```

基本功能：从指定文件fp读字符并将其存储到字符串str中。其中，str为指向存放字符串的存储空间的地址，n为读取字符串的总长，fp为所要操作的文件。

②fputs()函数

fputs()函数的一般形式为：

```
int fputs ( char *str, FILE *fp) ;
```

基本功能：把字符串str输出到fp所指向的文件。其中，第一个参数可以是字符串常量，也可以是字符数组名或字符型指针。若输出成功，函数返回最后写入的字符；若失败，返回EOF。

例：从某个已经存在的文件中读取一个含有10个字符的字符串。

```
#include "stdio.h"
```

```
main()
```

```
{
```

```
FILE *fp;
```

```
char str[11];
```

```
if((fp=fopen("d:\\inf.c",'r'))==NULL)
```

```
{
```

```
printf("Cannot open file");
```

```
exit(1);
```

```
}
```

```
fgets(str,11,fp);
```

```
printf("%s",str);
```

```
fclose(fp);
```

```
}
```

例：编写一程序完成文本文件的连接功能。

```
#include "stdio.h"
#define BUFSIZE 256
main(int argc,char *argv[])
{
    int i;
    char buf[BUFSIZE];
    FILE *fp1,*fp2;
    if(argc==1)
    {
        printf("\n The usage is : d_cat filename1 filename2...\\n");
        exit(0);
    }
    if((fp1=fopen(argv[1], "ab"))==NULL)
    {
        printf("\n file %s does not open \\n",argv[i]);
        exit(1);
    }
```

```
for(i=2;i<argc;i++)
{
    if((fp2=fopen(argv[i], "rb"))==NULL)
    {
        printf("\n file %s does not open \n",argv[i]);
        exit(2);
    }
    while(fgets(buf,BUFSIZE,fp2)!=NULL)
    {
        fputs(buf,fp1);
    }
    fclose(fp2);
}
fclose(fp1);
}
```

2.信息的格式化读/写

fprintf()函数、**fscanf()**函数与**printf()**函数、**scanf()**函数作用相似，都是信息的格式化输入/输出函数。

只有一点不同：**fprintf()**函数和**fscanf()**函数的读/写对象是磁盘文件；而**printf()**函数和**scanf()**函数的读/写对象是标准输出流（**stdout**）和标准输入流（**stdin**）。

①**fprintf()**函数

fprintf()函数的一般形式为

```
int fprintf(FILE *fp, char *format [,argument,...]);
```

基本功能：按照指定的格式将输出列表中的内容写入指定的文件中。其中，**fp**用于指明所要操作的文件，**format**（格式字符串）用于指明写入信息的写入格式，**argument**用于指明所要写入的信息。

例如：

```
fprintf(fp, " % d ,%7.3f " ,i,f);
```

②fscanf()函数

fscanf()函数的一般形式为：

```
int fscanf(FILE *fp, char *format [,argument,...]);
```

基本功能：从文件中读取数据，并将其按照格式字符串所指定的格式写入到地址参数&arg1、...、&argn所指定的地址中。

返回：成功扫描、转换、存储的输入字段数。如果该函数试图在文件末尾进行读操作，则返回EOF；如果没有字段被存储，则返回0。

例如

```
fscanf(fp,"%d,%f",&i,&i);
```

例：从键盘输入5个学生数据，写入一个文件中，再读出数据显示。

```
#include "stdio.h"
struct stu
{
    char name[10];
    int num;
    int age;
    char addr[15];
}s[5],*p;
main()
{
    FILE *fp;
    char ch;
    int i;
    p=s;
    if((fp=fopen("stu_list","wb+"))==NULL)
    {
        printf("Cannot create file!");
        exit(0);
    }
```

```
printf("\ninput data\n");
for(i=0;i<5;i++,p++)
    scanf("%s%d%d%s",p->name,&p->num,&p->age,p->addr);
p=s;
for(i=0;i<5;i++,p++)
    fprintf(fp,"%s %d %d %s\n",p->name,p->num,pp->age,p->addr);
rewind(fp);          /*使文件位置指针重新返回文件的开头*/
for(p=s,i=0;i<5;i++,p++)
    fscanf(fp,"%s %d %d %s\n",p->name,&p->num,&p->age,p->addr);
printf("\n\nname\tnumber age addr\n");
for(p=s,i=0;i<5;i++,p++)
    printf("%s\t%5d %7d %s\n",p->name,p->num,p->age,p->addr);
fclose(fp);
}
```

3.记录级的文件读/写

谓“记录”，从本质上讲是一个没有格式的数据块。

ANSI C提供两个函数来读/写一个数据块： `fread()` 函数和 `fwrite()` 函数。

记录有两种：一种是定长的，另一种是不定长的。值得注意的是，从函数的角度来讲，ANSI C只提供了对定长记录的支持，不定长记录也是通过这两个函数读/写的，但它需要更多的技巧、更精心构思的算法。

①fread函数

其一般调用形式为：

```
fread(buffer, size, count, fp);
```

表示从fp所指向的文件的当前位置读入count个大小为size字节的数据块放入buffer所指向的内存区域。

buffer: 是一个指针。

size: 要读写的字节数。

count: 要进行读写多少个size字节的数据项。

fp: 文件型指针。

②fwrite函数

其一般调用形式为：

```
fwrite(buffer, size, count, fp);
```

表示将从buffer开始的count个大小为size字节的数据块写入fp所指向的文件的当前位置。

例：改写文件myfile.dat中的第1个记录。

```
#include "stdio.h"
typedef struct my_struct
{
    int part_code; int quantity; float price;
}part;
main()
{
    FILE *fp; int n; part d_part={2000,30,38.75};
    if((fp=fopen('d:\\myfile.dat', "wb+"))!=NULL)
    {
        n=fwrite(&d_part,sizeof(part),1,fp);
        printf('the number of records written is %d',n);
        fclose(fp);
    }
    else
    {
        printf("fail to open the file ....\\n");
    }
}
```

例：从键盘读取学生信息，并将其保存在外存上。

```
#include "stdio.h"  
#define NUM 6  
typedef struct stu_type  
{  
    char name[16];  
    int code;  
    char birthday[11];  
    char addr[30];  
}stu;  
main()  
{  
    int i;  
    stu d_stu[NUM];  
    FILE *d_fp;  
    void d_save();
```

```
if((d_fp=fopen("d:\stu.dat", "wb"))==NULL)
{
    printf("\n open file error \n");
    exit(0);
}

printf("\n please input the information of all students \n");
for(i=0;i<NUM;i++)
{
    printf("\n name : ");        scanf("%s",d_stu[i].name);
    printf("\n code : ");       scanf("%d",&d_stu[i].code);
    printf("\n birthday : ");   scanf("%s",d_stu[i].birthday);
    printf("\n address : ");    scanf("%s",d_stu[i].addr);
}
```

```
d_save(d_stu,NUM,d_fp);
fclose(d_fp);
}

void d_save(stu d_stu[],int n,FILE *fp)
{
    int i;
    for(i=0;i<n;i++)
    {
        if(fwrite(&d_stu[i],sizeof(stu),1,d_fp)!=1)
        {
            printf("\n file write error \n");
        }
    }
}
```

4.文件位置指针的移动

在文件的读/写过程中，为了能够正确地完成输入与输出，系统需要有一个指示标志来指明当前正在读/写的位置，将这一指示标志称为文件位置指针。

在顺序读/写一个文件时，每次读/写后，文件位置指针自动后移。

ANSI C中提供了移动文件位置指针的函数，用于解决该问题。最常用的有3个： `rewind()` 函数、 `ftell()` 函数、 `fseek()` 函数。

①**rewind()**函数

rewind()函数的一般形式为

```
int rewind(FILE *stream);
```

基本功能： 该函数使文件位置指针重新返回文件的开头。如果指针移动成功， 返回值为0； 如果移动失败， 返回一非0值。

②**ftell()**函数

ftell()函数的一般格式为

```
long ftell(FILE*stream);
```

基本功能： 该函数用于读取文件位置指针当前位置相对于文件起点的偏移量。

③fseek()函数

fseek()函数可以实现改变文件位置指针。

该函数的一般形式为

```
int fseek(FILE *stream,long offset,int origin);
```

stream 为所要操作的文件的文件指针。

offset 为从指定位置移动指针时的偏移量（所需移动的大小）,
它必须为长整型。

origin为指针移动的开始位置： 必须是0、1、2中的一个。

0代表“文件开始”， 1为“当前位置”， 2为“文件末尾”。

③fseek()函数

fseek()函数可以实现改变文件位置指针。

该函数的一般形式为

```
int fseek(FILE *stream,long offset,int origin);
```

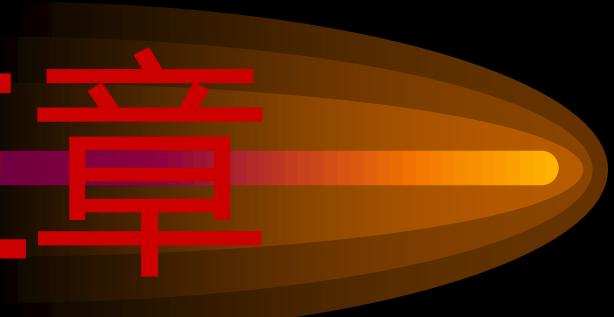
stream 为所要操作的文件的文件指针。

offset 为从指定位置移动指针时的偏移量（所需移动的大小），
它必须为长整型。

origin为指针移动的开始位置：必须是0、1、2中的一个。

0代表“文件开始”，1为“当前位置”，2为“文件末尾”。

第十二章



文件

● 本章要点

- 文件的基本概念
- 文件的基本函数
- 文件的顺序读写
- 文件的随机读写
- 文件简单应用

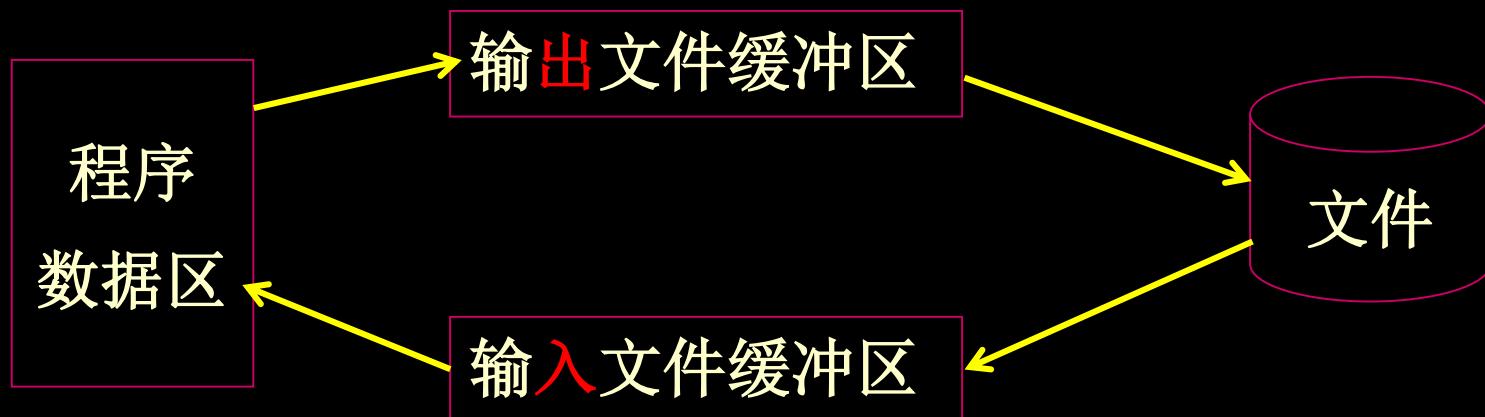
• 主要内容

- 13.1 C文件概述
- 13.2 文件类型指针
- 13.3 文件的打开与关闭
- 13.4 文件的读写
- 13.5 文件的定位
- 13.6 出错的检测
- 13.7 文件输入输出小结

13.1 C文件概述

文件：文件指存储在外部介质(如磁盘磁带)上数据的集合。

操作系统是以文件为单位对数据进行管理的。



13.1 C文件概述(续)

文件的分类

- 从用户观点：

特殊文件(标准输入输出文件或标准设备文件)。

普通文件(磁盘文件)。

- 从操作系统的角度看，每一个与主机相连的输入输出设备看作是一个文件。

例： 输入文件：终端键盘

输出文件：显示屏和打印机

13.1 C文件概述(续)

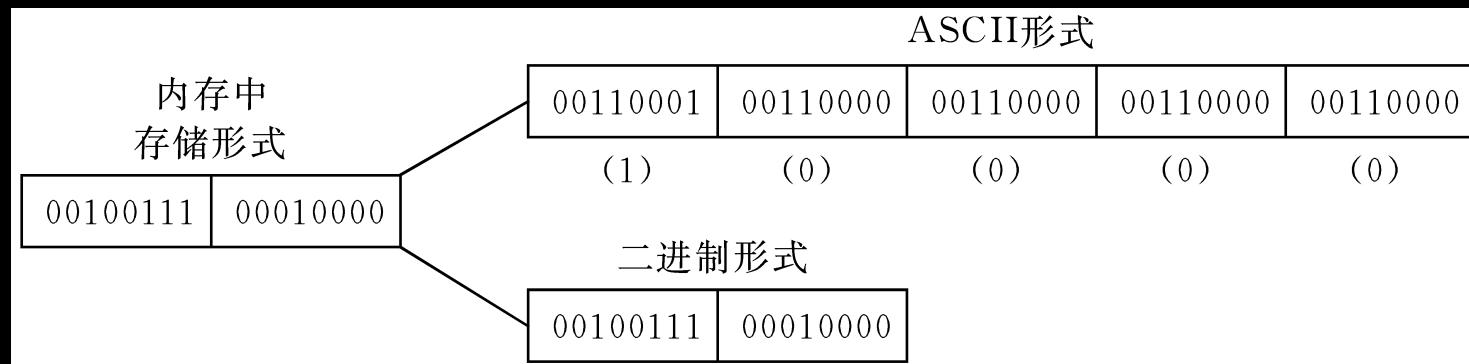
文件的分类

- 按数据的组织形式:

ASCII文件(文本文件):每一个字节放一个ASCII代码

二进制文件:把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放。

例: 整数10000在内存中的存储形式以及分别按ASCII码形式和二进制形式输出如下图所示:



13.1 C文件概述(续)

文件的分类

ASCII文件和二进制文件的比较：

ASCII文件便于对字符进行逐个处理，也便于输出字符。但一般占存储空间较多，而且要花费转换时间。

二进制文件可以节省外存空间和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。

一般中间结果数据需要暂时保存在外存上，以后又需要输入内存的，常用二进制文件保存。

13.1 C文件概述(续)

文件的分类

- C语言对文件的处理方法：

缓冲文件系统：系统自动地在内存区为每一个正在使用的文件开辟一个缓冲区。用缓冲文件系统进行的输入输出又称为**高级磁盘输入输出**。

非缓冲文件系统：系统不自动开辟确定大小的缓冲区，而由程序为每个文件设定缓冲区。用非缓冲文件系统进行的输入输出又称为**低级输入输出系统**。

13.1 C文件概述(续)

说明:

在UNIX系统下,用缓冲文件系统来处理文本文件,
用非缓冲文件系统来处理二进制文件。

ANSI C 标准只采用缓冲文件系统来处理文本文件
和二进制文件。

C语言中对文件的读写都是用库函数来实现。

13.2 文件类型指针

Turbo C在stdio.h文件中有以下的文件类型声明：

```
typedef struct
```

```
{ shortlevel; /*缓冲区“满”或“空”的程度*/  
unsignedflags; /*文件状态标志*/  
charfd; /*文件描述符*/  
unsignedcharhold; /*如无缓冲区不读取字符*/  
shortbsize; /*缓冲区的大小*/  
unsignedchar*buffer; /*数据缓冲区的位置*/  
unsignedchar*curp; /*指针，当前的指向*/  
unsignedchar*temp; /*临时文件，指示器*/  
shorttoken; /*用于有效性检查*/ } FILE;
```

在缓冲文件系统中,每个被使用的文件都要在内存中开辟一个FILE类型的区,存放文件的有关信息。

13.2 文件类型指针(续)

FILE类型的数组：

FILE f [5]; 定义了一个结构体数组f，它有5个元素，可以用来存放5个文件的信息。

文件型指针变量：

FILE *fp; fp是一个指向FILE类型结构体的指针变量。可以使fp指向某一个文件的结构体变量，从而通过该结构体变量中的文件信息能够访问该文件。如果有n个文件，一般应设n个指针变量，使它们分别指向n个文件，以实现对文件的访问。

13.3 文件的打开与关闭

一.文件的打开(fopen函数)

函数调用:

```
FILE *fp;
```

```
fp = fopen (文件名, 使用文件方式) ;
```

- ①需要打开的文件名，也就是准备访问的文件的名字；
- ②使用文件的方式（“读”还是“写”等）；
- ③让哪一个指针变量指向被打开的文件。

13.3 文件的打开与关闭(续)

文件使用方式	含 义
"r"	(只读)为 输入 打开一个 文本文件
"w"	(只写)为 输出 打开一个 文本文件
"a"	(追加)向 文本文件 尾增加数据
"rb"	(只读)为 输入 打开一个 二进制文件
"wb"	(只写)为 输出 打开一个 二进制文件
"ab"	(追加)向 二进制文件 尾增加数据
"r+"	(读写)为 读/写 打开一个 文本文件
"w+"	(读写)为 读/写建立 一个 新的文本文件
"a+"	(读写)为 读/写 打开一个 文本文件
"rb+"	(读写)为 读/写 打开一个 二进制文件
"wb+"	(读写)为 读/写建立 一个 新的二进制文件
"ab+"	(读写)为 读/写 打开一个 二进制文件

13.3 文件的打开与关闭(续)

二.文件的关闭(**fclose**函数)

函数调用:

fclose (文件指针) ;

函数功能:

使文件指针变量不指向该文件，也就是文件指针变量与文件“脱钩”，此后不能再通过该指针对原来与其相联系的文件进行读写操作。

返回值:

关闭成功返回值为 0；否则返回EOF(-1)。

13.4 文件的读写

一、字符输入输出函数(**fputs()**和**fgets()**)

fputs函数

函数调用:

```
fputs ( ch, fp );
```

函数功能:

将字符 (ch的值) 输出到fp所指向的文件中去。

返回值:

如果输出成功，则返回值就是输出的字符；

如果输出失败，则返回一个EOF。

13.4 文件的读写(续)

fgets函数

函数调用:

```
ch = fgets (fp) ;
```

函数功能:

从指定的文件读入一个字符,该文件必须是以读或读写方式打开的。

返回值:

读取成功一个字符, 赋给 c h。

如果遇到文件结束符, 返回一个文件结束标志 EOF。

13.4 文件的读写(续)

常见的读取字符操作

从一个**文本文件**顺序读入字符并在屏幕上显示出来：

```
ch = fgetc (fp) ;  
while (ch! =EOF)  
{  
    putchar (ch) ;  
    ch = fgetc (fp) ;  
}
```

注意： EOF不是可输出字符，因此不能在屏幕上显示。由于字符的ASCII码不可能出现 - 1，因此EOF定义为 - 1是合适的。当读入的字符值等于 - 1时，表示读入的已不是正常的字符而是文件结束符。

13.4 文件的读写(续)

常见的读取字符操作

从一个**二进制文件**顺序读入字符：

```
while (!feof (fp) )  
{  
    ch = fgetc (fp) ;  
}
```

注意：ANSI C提供一个feof () 函数来判断文件是否真的结束。如果是文件结束，函数feof (fp) 的值为 1 (真)；否则为 0 (假)。以上也适用于文本文件的读取。

```
#include <stdlib.h>
#include <stdio.h>
void main(void)
{ FILE *fp;
    char ch,filename[10];
    scanf("%s",filename);
    if((fp=fopen(filename,"w"))==NULL) {
        printf("cannot open file\n");
        exit(0); /*终止程序*/}
    ch=getchar(); /*接收执行scanf语句时最后输入的回车符 */
    ch=getchar(); /* 接收输入的第一个字符 */
    /*...*/
}
```

运行情况如下：

file1.c (输入磁盘文件名)

computer and c# (输入一个字符串)

computer and c (输出一个字符串)

```
#include <stdlib.h>
#include <stdio.h>
main()
{FILE *in,*out;
char ch,infile[10],outfile[10];
printf("Enter the infile name:\n");
scanf("%s",infile);
printf("Enter the outfile name:\n");
scanf("%s",outfile);
if((in=fopen(infile,"r"))==NULL)
```

运行情况如下：

Enter the infile name

file1.c (输入原有磁盘文件名)

Enter the outfile name:

file2.c (输入新复制的磁盘文件名)

程序运行结果是将 file1.c 文件中的内容复制到
file2.c 中去。

```
#include <stdlib.h>
#include <stdio.h>
main(int argc,char *argv[ ])
{FILE *in,*out;
char ch;
if (argc!=3)
{
    printf("You forgot to enter a filename\n");
    exit(0);
}
if((in=fopen(argv[1],"rb"))==NULL)
```

运行方法：

设经编译连接后得到的可执行文件名为a.exe，则在DOS命令工作方式下，可以输入以下的命令行：

C>a file1.c file2.c
file1.c和file2.c，分别输入到 argv[1] 和 argv[2] 中， argv[0] 的内容为a， argc 的值等于3。

13.4 文件的读写(续)

二、数据块读写函数(fread()和fwrite())

函数调用：

```
fread (buffer,size,count, fp);  
fwrite(buffer,size,count,fp);
```

参数说明：

buffer: 是一个指针。

对**fread** 来说，它是读入数据的存放地址。

对**fwrite**来说，是要输出数据的地址（均指起始地址）。

size: 要读写的字节数。

count: 要进行读写多少个**size**字节的数据项。

fp: 文件型指针。

13.4 文件的读写(续)

使用举例：

若文件以二进制形式打开：

```
fread(f,4,2,fp);
```

此函数从fp所指向的文件中读入2个4个字节的数据，存储到数组f中。

13.4 文件的读写(续)

使用举例：

若有如下结构类型：

```
struct student_type  
{char name[10];  
int num;  
int age;  
char addr[30];}stud[40];
```

可以用fread和fwrite来进行数据的操作：

```
for ( i = 0 ; i < 40 ; i ++ )  
    fread(&stud [i] , sizeof(struct student-type) , 1 , fp);  
for ( i = 0 ; i < 40 , i ++ )  
    fwrite(&stud [i] , sizeof(struct student-type) , 1 , fp);
```

13.4 文件的读写(续)

使用举例：

例 13.3 从键盘输入 4 个学生的有关数据，然后把它们转存到磁盘文件上去。

```
#include <stdio.h>
#define SIZE 4
struct student_type
{
    char name[10];
    int num;
    int age;
    char addr[15];
}stud[SIZE]; /*定义结构*/
```

```
void save( )
{FILE *fp;
int i;
if((fp=fopen("stu-list","wb"))==NULL)
{ printf("cannot open file\n");
return;}
for(i=0;i<SIZE;i++)/*二进制写*/
if(fwrite(&stud[i],sizeof(struct student_type),1,fp)!=1)
```

运行情况如下：

输入 4 个学生的姓名、学号、年龄和地址：

Zhang 1001 19 room-101

Fan 1002 20 room-102

Tan 1003 21 room-103

Ling 1004 21 room-104

```
#include <stdio.h>
#define SIZE 4
struct student_type
{
    char name[10];
    int num;
    int age;
    char addr[15];
}stud[SIZE];
main()
```

屏幕上显示出以下信息：

Zhang	1001	19	room-101
Fun	1002	20	room-102
Tan	1003	21	room-103
Ling	1004	21	room-104

13.4 文件的读写(续)

如果已有的数据已经以二进制形式存储在一个磁盘文件

“stu.dat”中，要求从其中读入数据并输出到
“student.list”文件中，可以编写一个load函数，
从磁盘文件中读二进制数据。

```
void load()
{FILE *fp;int i;
if((fp=fopen("stu-dat","rb"))==NULL)
{ printf("cannot open infile\n");
return;}
for(i=0;i<SIZE;i++)
if(fread(&stud[i],sizeof(struct student_type),1,fp)!=1)
{if(feof(fp)) {fclose(fp); return;}
printf("file read error\n");}
fclose (fp); }
```

13.4 文件的读写(续)

三、格式化读写函数(fprintf()和fscanf())

函数调用:

fprintf (文件指针, 格式字符串, 输出表列) ;
fscanf (文件指针, 格式字符串, 输入表列) ;

函数功能:

注意:

用fprintf和fscanf函数对磁盘文件读写, 使用方便, 容易理解, 但由于在输入时要将ASCII码转换为二进制形式, 在输出时又要将二进制形式转换成字符, 花费时间比较多。因此, 在内存与磁盘频繁交换数据的情况下, 最好不用fprintf和fscanf函数, 而用fread和fwrite函数。

13.4 文件的读写(续)

三、其他读写函数

putw()和**getw()**

函数调用:

```
putw(int i,FILE * fp);
```

```
int i = getw(FILE * fp);
```

函数功能:

对磁盘文件中读写一个字 (整数)

例:

```
putw(10,fp);
```

```
i = getw(fp);
```

getw函数定义如下:

```
getw(FILE *fp)
{
    char s;
    s=char *&i;
    s[0] = getc(fp);
    s[1] = getc(fp);
    return i;
}
```

13.4 文件的读写(续)

用户自定义读取其他类型数据的函数。

向磁盘文件写一个实数 (用二进制方式) 的函数putfloat：

```
putfloat(float num,FILE *fp)
```

```
{
```

```
    char s;
```

```
    int count;
```

```
    s = (char*)&num;
```

```
    for(count = 0;count < 4;count++)
```

```
        putc(s[count],fp);
```

```
}
```

13.4 文件的读写(续)

fgets函数

函数作用：

从指定文件读入一个字符串。

函数调用：

`fgets(str,n,fp);`

从`fp`指向的文件输入`n-1`个字符，在最后加一个' \0'。

返回值：

`str`的首地址。

13.4 文件的读写(续)

fputs函数

函数作用：

向指定的文件输出一个字符串。

函数调用：

```
fgets("china",fp);
```

第一个参数可以是字符串常量、字符数组名或字符型指针。字符串末尾的' \ 0 '不输出。

返回值：

输入成功，返回值为0；

输入失败，返回EOF。

13.5 文件的定位

```
#include<stdio.h>
main()
{
    FILE *fp1,*fp2;
    fp1=fopen("file1.c","r");
    fp2=fopen("file2.c","w");
    while(!feof(fp1)) putchar(getc(fp1));
    rewind(fp1);
    while(!feof(fp1))
        putc(getc(fp1),fp2);
    fclose(fp1);fclose(fp2);
}
```

13.5 文件的定位

顺序读写和随机读写

顺序读写：

位置指针按字节位置顺序移动。

随机读写：

读写完上一个字符（字节）后，并不一定要读写其后续的字符（字节），而可以读些文件中任意位置上所需要的字符（字节）。

13.5 文件的定位

fseek函数 (一般用于二进制文件)

函数功能：

改变文件的位置指针。

函数调用形式：

`fseek(文件类型指针, 位移量, 起始点)`

起始点：文件开头 SEEK_SET 0

 文件当前位置 SEEK_CUR 1

 文件末尾 SEEK_END 2

位移量：以起始点为基点，向前移动的字节数。一般要求为long型。

13.5 文件的定位

fseek函数应用举例

`fseek(fp, 100L, 0) ;`

将位置指针移到离文件头100个字节处。

`fseek(fp, 50L, 1) ;`

将位置指针移到离当前位置50个字节处。

`fseek(fp, 50L, 2) ;`

将位置指针从文件末尾处向后退10个字节。

```
#include <stdlib.h>
#include<stdio.h>
struct student_type
{
    char name[10];
    int num;
    int age;
    char sex;
}stud[10];    main()
{ int i;
FILE *fp;
if((fp=fopen("stud-dat","rb"))==NULL)
{printf("can not open file\n");
exit(0);}
for(i=0;i<10;i+=2)
{fseek(fp,i*sizeof(struct student_type),0);
 fread(&stud[i], sizeof(struct student_type),1,fp);
printf( "%s %d %d %c\n" ,stud[i].name,
stud[i].num,stud[i].age,stud[i].sex);}
fclose(fp)}
```

13.5 文件的定位

ftell函数

函数作用：

得到流式文件中的当前位置，用相对于文件开头的位移量来表示。

返回值：

返回当前位置，出错时返回-1L。

应用举例：

```
i = ftell(fp);  
if(i== -1L) printf("error\n");
```

13.6 出错的检测

ferror函数

调用形式：

`ferror(fp);`

返回值：

返回0，表示未出错；返回非0，表示出错。

在调用一个输入输出函数后立即检查ferror函数的值，否则信息会丢失。在执行fopen函数时，ferror函数的初始值自动置为0。

13.6 出错的检测

clearerr函数

调用形式：

clearerr(fp);

函数作用：

使文件错误标志和文件结束标志置为0。

只要出现错误标志，就一直保留，直到对同一文件调用clearerr函数或rewind函数，或任何其他一个输入输出函数。

13.7 文件输入输出小结

分类	函数名	功能
打开文件	fopen()	打开文件
关闭文件	fclose()	关闭文件
文件定位	fseek() Rewind()	改变文件位置指针的位置 使文件位置指针重新至于文件开头
	Ftell()	返回文件位置指针的当前值
文件状态	feof() Ferror() Clearerr()	若到文件末尾，函数值为真 若对文件操作出错，函数值为真 使ferror和feof()函数值置零

13.7 文件输入输出小结

分类

文件读写

分类	函数名	功能
文件读写	fgetc(),getc() fputc(),putc() fgets() fputs() getw() putw() fread() fwrite() fscanf() fprintf()	从指定文件取得一个字符 把字符输出到指定文件 从指定文件读取字符串 把字符串输出到指定文件 从指定文件读取一个字（int型） 把一个字输出到指定文件 从指定文件中读取数据项 把数据项写到指定文件中 从指定文件按格式输入数据 按指定格式将数据写到指定文件中

第十五章 预处理与复杂函数应用

C语言提供三种预处理功能：

宏定义；

文件包含；

条件编译。

分别用宏定义命令、文件包含命令和条件编译命令来实现。这些命令以符号“#”开头，结尾没有分号(;)。

该命令的作用域是从定义的地方开始到源文件结束。

在 C 编译系统对程序进行通常的编译之前，先对程序中这些命令进行“预处理”。然后将结果和源程序一起再进行编译处理，最后得到目标代码。

1. 宏定义

① 不带参数的宏定义

功能：用一个指定的标识符(宏名)来代表一个字符串。

一般形式：`#define 标识符 任意字符串序列`

```
#define PI 3.1415926
main()
{float l,s,r,v;
 printf("input radius:");
 scanf(%f,&r);
 l=2.0*PI*r;
 s=PI*r*r;
 v=4.0/3*PI*r*r*r;
 printf("l=%f\ns=%f\nv=%f\n",l,s,v);
}
```

说明：

宏定义是用宏名代替一个字符串，也就是作简单的置换，不作语法检查。

#define PI 3.1415926p;

area=PI*r;

预处理后为

area=3.1415926p;*r⁷⁵

②带参数的宏定义

进行字符串替换，还要进行参数替换。

定义形式为：#define 宏名(参数表) 字符串

包含括弧中指定的参数

如： #define S(a,b) a*b

:

area=S(3,2);

area=3*2

注意：不能写成 #define S (a,b) a*b

不能有空格

```
#define PI 3.1415926
#define S(r) PI*r*r
main()
{float a,area;
 a=3.6;
 area=S(a);
 printf("r=%f\narea=%\n",a,area);
}
```

预处理

```
main()
{float a,area;
 a=3.6;
 area=3.1415926*a*a;
printf("r=%f\narea=%\n",a,area);
}
```

如果有以下语句,

```
area=S(x+y);
```

预处理后
结果为

```
area=3.1415926*x+y*x+y;
```

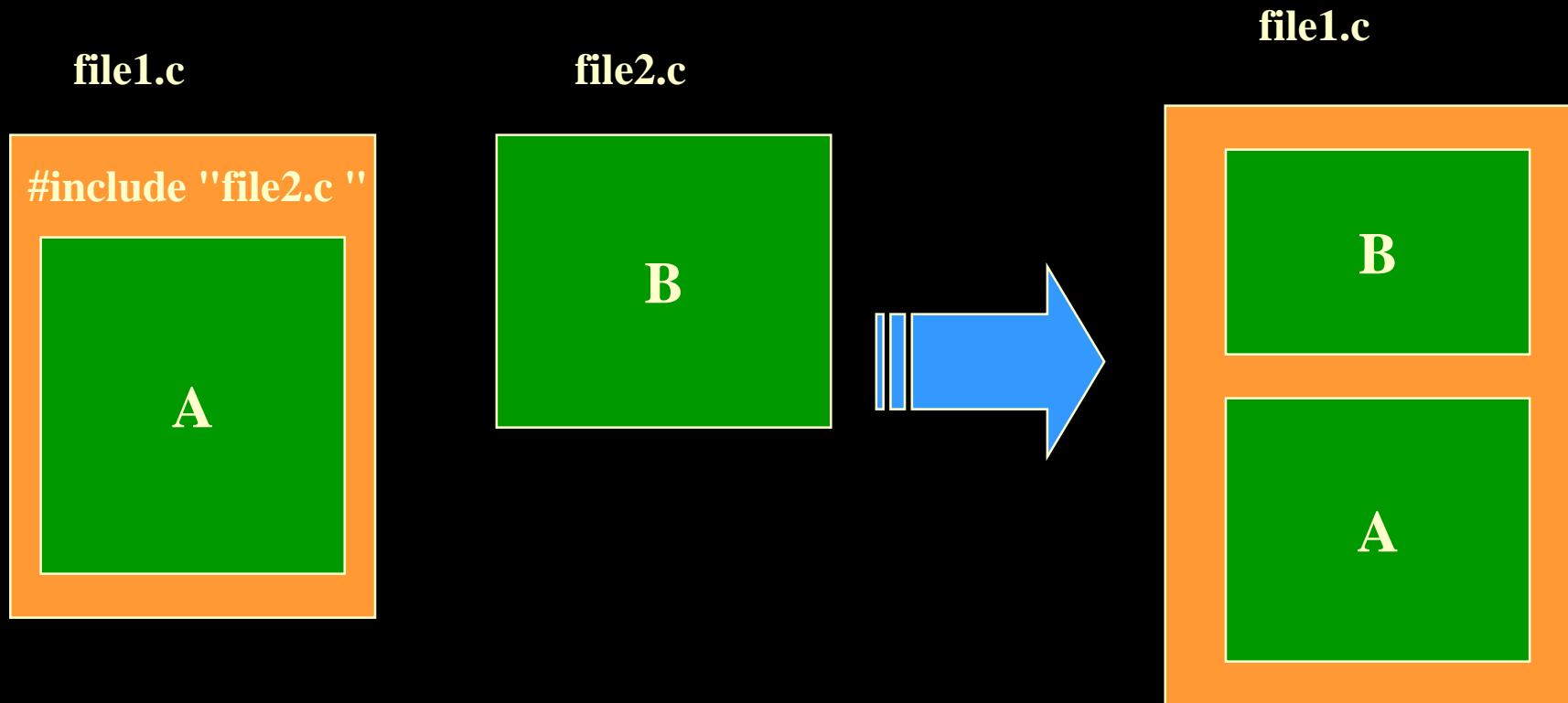
```
#define S(r) PI*(r)*(r)
```

```
area=3.1415926*(x+y)*(x+y);
```

2. “文件包含” 处理

文件包含处理是指一个源文件可以将另一个源文件的全部内容包含进来。

一般形式为：#include "文件名"



3.条件编译

条件编译是指让编译程序根据条件有选择地对源程序的一部分进行编译。

C语言提供三组条件编译命令，其形式如下：

#if 标识符	程序段1
#else	程序段2
#endif	

#ifndef 标识符	程序段1
#else	程序段2
#endif	

#if 表达式	程序段1
#else	程序段2
#endif	

4. 命令行参数

C语言规定，主函数可以带参数，也可以不带参数。主函数的形参只有两个。

第一个形参记录了实际参数的个数，其名称规定为“`argc`”；第二个形参依次记录了在调用该主函数（即执行命令）时给出的实参内容，形参名称规定为“`argv[]`”。

`argc`表示命令行参数个数（包括运行程序文件名），`argv`是指向命令行参数的指针数组。指针`argv[0]`指向的字符串是运行程序文件名，`argv[1]`指向的字符串是命令行参数1，`argv[2]`指向的字符串是命令行参数2……

例： 编一个带参数的主函数。运行时在程序名后带2个整数，程序的功能是输出这个整数的积。源程序文件名为mult.c， 程序中用到的atoi()函数的功能是将ASCII数字串转换为整型数。

```
#include "stdlib.h"
main(argc,argv)
int argc;
char *argv[];
{
int x,y,mult;
x=atoi(argv[1]);/*atoi()函数是将ASCII数字串转换为整型*/
y=atoi(argv[2]);
mult=x*y;
printf("%d*%d=%d\n",x,y,mult);
}
```

本程序编译后的目标程序名为mult.exe， 执行该程序时输入的命令行如下：
mult 7 8

指针练习

- 1、 int a=3,b,*p=&a;
- b=*&a;b=*p;b=a;b=*a
- 2、 int a[]={1,2,3,4,5,6,7,8,9,10},*p=a;
- P+=2,*(p++); p+=2,*++p;
- p+=3,*p++; p+=2,++*p
- 3、 char a[`0]={“abcd”},*p=a;
- *(p+4)值？
- 4、 int **pp,*p,a=10,b=20;
- pp=&p;p=&a;p=&b;printf(“%d,%d”,*p,**pp);
- 5、 char *a=“abcd”,printf(“%s”,a);printf(“%c”,*a);

- 6、

```
int a[3][2]={1,2,3,4,5,6},(*p)[2];
```
- ```
*(p+2)=
```
- 7、

```
char a[]="abcd",*p;
```
- ```
for(p=a;p<=a+4;p++)printf("%s\n",p);
```
- 8、

```
int a[10],*p=a+9;
```
- ```
for (i=0;i<10;i++)scanf("%d",&a[i]);
```
- ```
for(;p>=a;p--)printf("%d ",*p);
```
- 9、键盘输入若干字符（回车结束）组成一字符串存入字符数组，输出字符数组中的字符串，填空：

```
char a[81],*p;int i;
```



```
for(i=0;i<80;i++){a[i]=getchar();if(a[i]=='\n')break;}
```



```
a[i]=      ;
```



```
p=a;
```



```
While(*p)putchar(*p      );
```
- 10、在W数组中插入x,n所指向的存储单元中存放W数组中字符个数，数组W中的字符已按从小到大顺序排列，插入后数组W仍有序，填空

```
Int i,p=0;
```



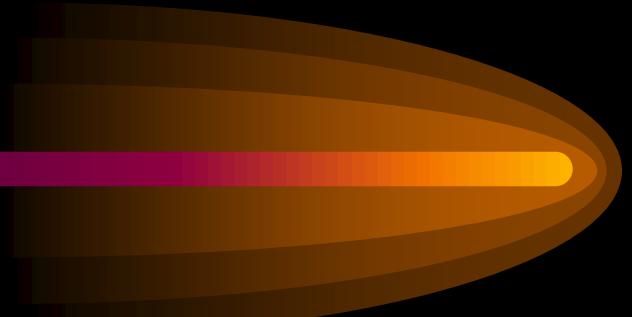
```
W[*n]=x;while(x>w[p])p++;
```



```
For(i=*n,i>p;i--)w[i]=      ;
```



```
W[p]=x;++*n;
```



The end