



中国科学技术大学  
University of Science and Technology of China

计算机程序设计

# 第三章 结构化程序设计

白雪飞

中国科学技术大学微电子学院

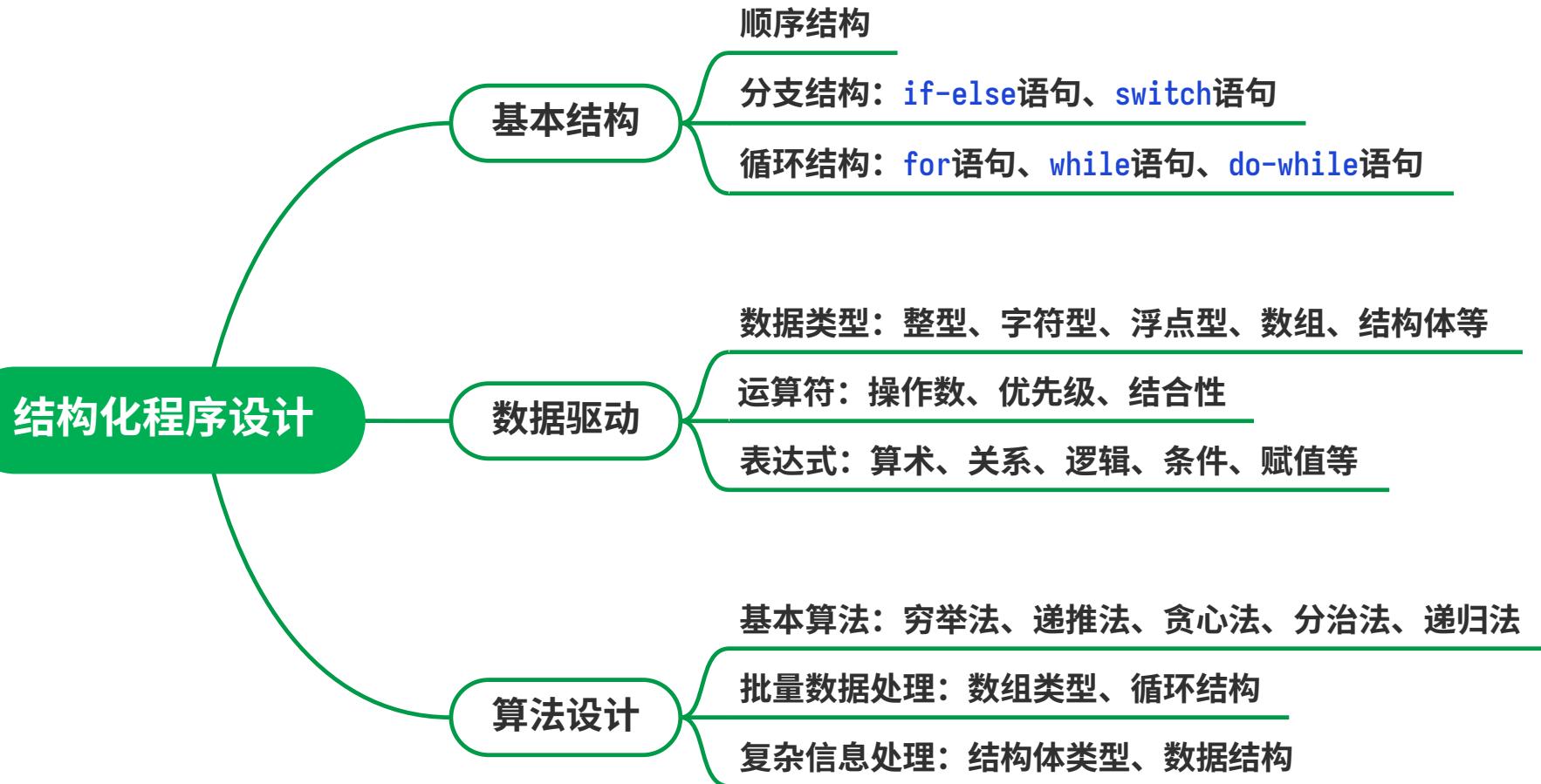
# 目录

- 引言
- 基本数据类型
- 表达式与运算规则
- 控制语句与基本算法
- 数组与批量数据处理
- 结构体与复杂信息处理
- 小结





# 引言





# 基本数据类型



- 整型数据
- 浮点型数据
- 字符型数据
- 幻数与宏定义



# 整型数据类型

## 整型数据类型及长度

数据类型			数据长度（位）				
(符号)	(长度)	(整型)	C标准	Windows 16 (LP32)	Windows 32 Unix/Linux 32 (ILP32)	Windows 64 (LLP64)	Unix/Linux 64 (LP64)
[signed]	short	[int]	≥16	16	16	16	16
	/		≥16	16	32	32	32
	long		≥32	32	32	32	64
	long long (C99)		≥64	64	64	64	64

`1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`



# 整型常量

## ■ 书写格式

- 十进制：以非0数字起始的十进制数
- 八进制：以数字0起始的八进制数
- 十六进制：以0x或0X起始的十六进制数，字母不区分大小写
- 整型常量不包含正负号，所以没有负的整型常量
- 举例：123, 0123, 0x123, 0xA71C

## ■ 数据类型

- 隐含类型：编译器自动选择无后缀整型常量的类型
- 类型后缀：
  - 无符号数 (unsigned) : u或U
  - 长整型 (long) : l或L
  - 双长整型 (long long) : ll或LL (C99)
- 举例：123u, 123L, 123ul, 123lu, 123LL, 0xFULL



# 整型数据编码和溢出

## ■ 编码格式和取值范围

- 无符号数：二进制编码，取值范围  $[0, 2^N - 1]$
- 有符号数：二进制补码，取值范围  $[-2^{N-1}, 2^{N-1} - 1]$

	高字节	低字节																	
无符号数	$2^{15} \ 2^{14}$	$2^7$	$2^0 = 32897$																
	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	0	0	0	0	0	0	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	1	0	0	0	0	0	0	1	
1	0	0	0	0	0	0	0												
1	0	0	0	0	0	0	1												
有符号数	$-2^{15} \ 2^{14}$	$2^7$	$2^0 = -32639$																

## ■ 溢出 (Overflow)

- 数值超出数据类型所能表示的取值范围
- 溢出会造成数据丢失或数值改变
- 应选择适当的数据类型，确保数据的值不超出取值范围



# 整型数据溢出举例

## ■ 例3.1-1：整型数据的溢出。

sizeof(表达式) 表达式不求值

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

```
int main()
{
```

```
    short a = 32767;
```

```
    printf("short类型的长度是%d\n", (int)sizeof(short));
```

```
    a = a + 3;
    printf("a的值是%d", a);
```

```
    return 0;
}
```

### 程序解析：

- 1 sizeof运算符的操作数可以是数据对象或数据类型，用于计算操作数的字节数；
- 2 sizeof运算符返回的字节数为无符号整型，使用强制类型转换运算符(int)获得int型数据用于输出；
- 3 16位short类型数据表达范围的最大值为 $2^{15}-1=32767$ ，a的初值已达该最大值，增加3将发生溢出。

### 结果分析：

$$\begin{array}{r} 0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111 \\ (+) 0000\ 0000\ 0000\ 0000\ 0011\ 0011\ 0011\ 0011 \\ \hline 1000\ 0000\ 0000\ 0000\ 0010\ 0010\ 0010\ 0010 \end{array} \quad \begin{array}{l} (32767) \\ (+3) \\ (-32766) \end{array}$$

### 运行结果：

short类型的长度是2  
a的值是-32766



# 整型数据应用举例

## ■ 例3.1-2：根据成绩提示是否通过考试。

```
#include <stdio.h>

int main()
{
    int i;                                // 数据个数小于int类型最大值
    int score;                            // 成绩取值为[0,100]内的整数

    for (i=0; i<8; i++) {                // 循环8次处理数据
        scanf("%d", &score);            // 输入整数形式的考试成绩
        if (score >= 60)
            printf("恭喜你，通过考试！\n");
        else
            printf("别灰心，明年再来！\n");
    }

    return 0;
}
```

### 运行结果：

90↵

恭喜你，通过考试！

85↵

恭喜你，通过考试！

35↵

别灰心，明年再来！

60↵

恭喜你，通过考试！

78↵

恭喜你，通过考试！

55↵

别灰心，明年再来！

75↵

恭喜你，通过考试！

95↵

恭喜你，通过考试！



- 整型数据
- 浮点型数据
- 字符型数据
- 幻数与宏定义



# 浮点型数据类型

## 浮点型数据类型及取值范围

数据类型	数据长度 (位)	阶码长度 (位)	尾数长度 (位)	十进制 有效数字长度	格式	正规数绝对值 取值范围
<b>float</b>	32	8	23	7.22	IEEE 754	$\sim 1.2 \times 10^{-38} - 3.4 \times 10^{38}$
<b>double</b>	64	11	52	15.95	IEEE 754	$\sim 2.2 \times 10^{-308} - 1.8 \times 10^{308}$
<b>long double</b>	80	15	64	19.27	x86 Extended	$\sim 3.4 \times 10^{-4932} - 1.2 \times 10^{4932}$
	128	15	112	34.02	IEEE 754	$\sim 3.4 \times 10^{-4932} - 1.2 \times 10^{4932}$



# 浮点型常量

## ■ 书写格式

- 一般形式：整数部分.小数部分 指数部分
- 指数部分以E或e和整数数字表示，代表乘以10的整数次幂
- 指数部分和小数点不能同时省略
- 整数部分和小数部分不能同时省略
- 浮点型常量有效数字部分不包含正负号，所以没有负的浮点型常量
- 举例：32.4, 7.0, 0.7, 15.2E8, 1e-6, .9e+4, 0., .0

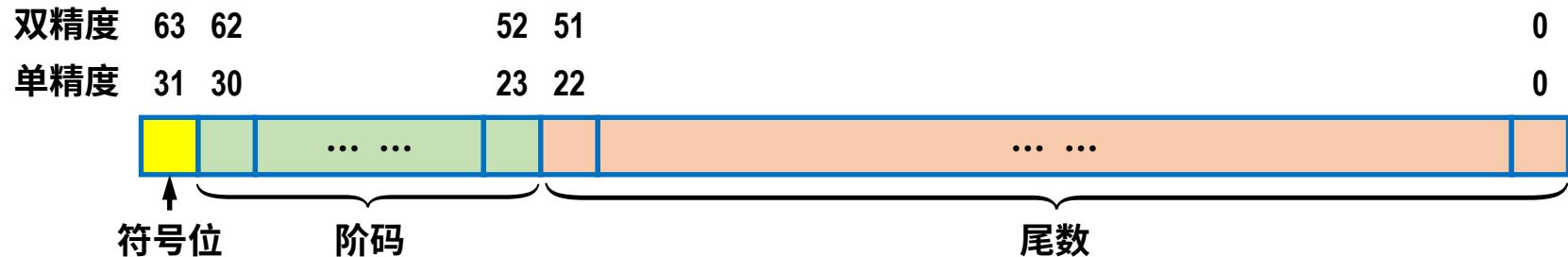
## ■ 数据类型

- 隐含类型：无后缀浮点型常量为double类型
- 类型后缀
  - 单精度浮点型 (float) : f或F
  - 长双精度浮点型 (long double) : l或L
- 举例：4.23f, 0.1F, 76.53L, 67E-12L

# 浮点型数据编码和误差

## ■ 编码格式

- IEEE Std 754: 二进制浮点数算术标准



## ■ 误差 (Error)

- 浮点数取值范围大，基本没有溢出问题
- 浮点数无法精确表示数据
- 浮点数的误差来源
  - 小数部分由十进制转换为二进制无限小数，从而产生的截断误差
  - 长度有限的尾数导致有效数字位数有限，进而产生的数据表示误差



# 浮点数误差举例

## ■ 例3.1-3：浮点数的误差和相等比较。

```
#include <stdio.h>

int main()
{
    float x1=0.1;
    double x2=0.1;

    if (x1 == x2)
        printf("x1与x2相等\n");      // 这是无误差的理想情况
    else
        printf("x1与x2不相等\n");  // 这是有误差的现实情况

    printf("x1=% .15f\nx2=% .15f\n", x1, x2);

    return 0;
}
```

### 程序解析：

- 1 十进制数0.1无法转换为有限长度的二进制数，float和double类型的有效数字位数不同，变量x1和x2中保存的二进制表示形式也不同，存在不同的误差；
- 2 比较x1和x2是否相等时，因为二者实际保存的二进制数并不相等，因此比较结果为“假”。

### 运行结果：

```
x1与x2不相等
x1=0.100000001490116
x2=0.100000000000000
```

### 结果分析：

```
x1 0011 1101 1100 1100 1100 1100 1100 1100 1101
x2 0011 1111 1011 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1010
```



# 浮点数误差举例

## ■ 例3.1-3：浮点数的误差和相等比较。

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

int main()
{
    float x1=0.1;
    double x2=0.1;

    if (fabs(x1-x2) < 1e-6)          // 设置合理的误差上限
        printf("x1与x2相等\n");      // 差值低于上限则相等
    else
        printf("x1与x2不相等\n");    // 差值达到上限则不相等

    printf("x1-x2=%e\n", fabs(x1-x2));

    return 0;
}
```

### 程序解析：

- 1 由于浮点型数据存在误差，因此，比较两个浮点数是否相等时的常用方法是，判断二者差值的绝对值是否小于某个合理的上限；
- 2 函数`fabs()`可以获得`double`型参数的绝对值，该函数的声明在头文件`math.h`中；
- 3 当程序中需要使用数学库函数时，应使用`#include`预处理命令包含头文件`math.h`。

### 运行结果：

```
x1与x2相等
x1-x2=1.490116e-009
```



- 整型数据
- 浮点型数据
- **字符型数据**
- 幻数与宏定义



# 字符型数据类型

## 字符型数据类型及取值范围

数据类型	符号	数据长度（位）	取值范围
signed char	有	8	[-128, 127]
unsigned char	无	8	[0, 255]
char	实现为signed char或unsigned char		

## ■ 字符型数据的表示与使用

- 字符型数据实质上是长度为一个字节的整数
- 当以字符格式(%c)进行输入输出时，字符型数据通过ASCII编码映射到字符
- 其他情况下，字符型数据可以与整型数据类似进行运算和输入输出(%d, %u)



# 字符常量和字符串常量

## ■ 字符常量的书写格式

- 一对单引号括起来的一个字符或以反斜杠起始的转义序列
- 举例：'a', 'A', '0', ' ', '#', '\n', '\x3F'

## ■ 字符常量的值与数据类型

- 字符常量的值为字符的ASCII值
- 字符常量的数据类型为int类型

## ■ 字符串常量的书写格式

- 一对双引号括起来的字符序列，可以包括零个、一个或多个字符
- 字符串常量的数据类型为字符数组
- 举例："abc", "a", "", " ", "我", "12.4", "\$#\*!", "C:\\HOME"



# 转义字符

转义字符	ASCII值	功 能
\a	0x07, 7	响铃
\b	0x08, 8	退格符，打印位置向左移一列
\f	0x0C, 12	换页符，打印位置移到下一页的开头
\n	0x0A, 10	换行符，打印位置移到下一行的开头
\r	0x0D, 13	回车符，打印位置移到当前行的开头
\t	0x09, 9	横向制表符，打印位置移到下一个横向制表位
\v	0x0B, 11	纵向制表符，打印位置移到下一个纵向制表位
\\"	0x5C, 92	反斜杠字符
\?	0x3F, 63	问号字符
\'	0x27, 39	单引号字符
\"	0x22, 34	双引号字符
\o, \oo, \ooo	0o, 0oo, 0ooo	1~3位八进制数，ASCII值等于该数值的字符
\xh, \xhh	0xh, 0xhh	1~2位十六进制数，ASCII值等于该数值的字符



# 字符型数据应用举例

## ■ 例3.1-4：输入5个字符并输出，其中的小写字母以大写字母形式输出。

```
#include <stdio.h>

int main()
{
    char c, i;
    for (i=0; i<5; i++) {
        c = getchar(); // 输入一个字符
        if (c >= 'a' && c <= 'z') // 判断c是否为小写字母
            putchar(c-'a'+'A'); // 求大写字母ASCII并输出
        else
            putchar(c);
    }
    return 0;
}
```

### 程序解析：

- 1 循环变量i虽然声明为字符型，但是实际用作小整数；
- 2 函数getchar()从键盘接收一个字符，并返回其ASCII值；
- 3 函数putchar()将参数值作为ASCII值，并输出对应字符；
- 4 表达式中的字符型变量和字符常量以其ASCII值作为整数参与关系运算和算术运算。

### 运行结果：

```
a2c@e↵  
A2C@E
```



- 整型数据
- 浮点型数据
- 字符型数据
- 幻数与宏定义



# 幻数与宏定义

## ■ 幻数 (Magic Number)

- 直接书写在程序中的字面常量
- 难以理解，难以修改

## ■ 宏定义 (Macro Definition)

```
#define 宏名 替换的文本串
```

- 可以通过宏定义赋予幻数有意义的名字
- 便于修改常量的值，仅需在宏定义中修改即可
- 宏名遵循标识符规范，习惯上使用大写字母以便与变量区别
- 在预处理中，宏名直接替换为宏定义中的文本串，称为**宏替换**
- 当文本串为常量形式时，也称为**符号常量**
- 文本串可以是任意内容，都将在宏替换时直接替换

```
#define WEIGHT 80      // 体重值，可根据需要在此修改  
#define HEIGHT 1.85    // 身高值，可根据需要在此修改
```



# 宏定义举例

## ■ 例3.1-5：使用符号常量表示身高和体重，并计算BMI。

```
#include <stdio.h>

#define WEIGHT 80 //体重值，可在此修改
#define HEIGHT 1.85 //身高值，可在此修改

int main()
{
    printf("身高%.2f, 体重%d, BMI是%.2f\n",
           HEIGHT, WEIGHT, WEIGHT/HEIGHT/HEIGHT);
    return 0;      预处理
}
```

### 运行结果：

身高1.85, 体重80, BMI是23.37

### 程序解析：

- 1 #include和#define等以#开头的行称为预处理命令，预处理过程在编译之前进行；
- 2 在预处理过程中将进行宏替换，例如，WEIGHT/HEIGHT/HEIGHT将被替换为80/1.85/1.85，在此过程中，宏定义中的文本串被视为一串字符，而不被视为数字，因此也不会进行C语言的语法检查；
- 3 若宏替换后的语句有语法错误，则将在预处理之后的编译过程中进行报告提示。



# 宏定义举例

## ■ 例3.1-6：宏展开举例。

```
#include <stdio.h>

#define N a==2) printf("a=%d\n", a);
#define b 2;

int main()
{
    int a=b
    if (N
        return 0;
}
```

预处理  
→

```
//  
// #include <stdio.h> 经过预处理后的  
// 内容较多，在此不再展示  
  
int main()  
{  
    int a=2;  
    if (a==2) printf("a=%d\n", a);  
    return 0;  
}
```

运行结果：

a=2

注意：

强烈不建议如此滥用宏定义！！！



# 表达式与运算规则



- 运算符简介
- 常用表达式
- 隐式类型转换规则
- 表达式的运算次序



# 运算符 (Operator)

类别	运算符
算术运算符	+ - * / % ++ -- + - ++ --
关系运算符	< <= > >= == !=
逻辑运算符	&&    !
位运算符	<< >> ~ & ^
赋值运算符	= += -= *= /= %= <<= >>= &= ^=  =
条件运算符	?:
逗号运算符	,
指针运算符	* &
求类型字节数运算符	sizeof
强制类型转换运算符	(类型)
成员访问运算符	. ->
下标运算符	[]
其他运算符	()



# 运算符的属性

## ■ 操作数 (Operand)

- 操作数的数量：一元（单目）、二元（双目）、三元（三目）
- 操作数的类型：部分运算符对操作数的类型有特定要求
- 操作数的存储特性：部分运算符要求操作数为**左值**

## ■ 优先级 (Precedence)

- 优先级决定运算符的运算次序
- 必要时可以使用`()`提升优先级

## ■ 结合性 (Associativity)

- 结合性决定优先级相同的运算符的运算次序
- 必要时可以使用`()`改变结合关系

## ■ 副作用 (Side Effect)

- 自增、自减、赋值等运算符，在对表达式求值的同时会修改操作数的值



# 左值 (Lvalue)

## ■ 左值表达式

- 一类特殊的表达式，是对某些具有存储空间的对象的引用
- 得名于赋值运算符的左操作数，但并非都处于运算符左边
- 用()括起来的左值表达式仍然是左值表达式
- 例如，变量、数组元素等

## ■ 左值的使用语境

- 赋值运算符=及复合赋值运算符+=等的左操作数
- 自增++、自减--运算符的操作数
- 取地址运算符&的操作数
- 成员访问运算符.的操作数



- 运算符简介
- 常用表达式
- 隐式类型转换规则
- 表达式的运算次序



# 算术运算符

优先级	运算符	描述	举例	操作数	结合性
1	<code>++</code>	后缀自增	<code>a++</code>		左结合
	<code>--</code>	后缀自减	<code>a--</code>		
2	<code>++</code>	前缀自增	<code>++a</code>		一元
	<code>--</code>	前缀自减	<code>--a</code>		
	<code>+</code>	正号	<code>+a</code>		
	<code>-</code>	负号	<code>-a</code>		
3	<code>*</code>	乘法	<code>a * b</code>		左结合
	<code>/</code>	除法	<code>a / b</code>		
	<code>%</code>	求余	<code>a % b</code>		
4	<code>+</code>	加法	<code>a + b</code>		
	<code>-</code>	减法	<code>a - b</code>		



# 算术表达式

## ■ 整数除法

- 除法运算符 / 的两个操作数都是整数时，运算结果也是整数
- 整数除法的运算结果不含小数部分，也不会进行舍入
  - 例如， $2/3$  的值为 0
- 若需要得到带有小数部分的商，应使至少一个操作数为浮点型
  - 例如， $2.0/3$ ,  $2/3.0$ ,  $(double)x/y$
- 若有负操作数，则商的截取方向取决于具体实现

## ■ 求余运算

- 求余运算符 % 的两个操作数都必须是整数
- 整数除法和求余运算结果应满足： $(a/b)*b+a \% b == a$



# 自增和自减运算

## ■ 求值规则

```
expr++, expr-- // 后缀形式  
++expr, --expr // 前缀形式
```

- 操作数必须是整数类型、浮点型等数据类型的**左值**
- 副作用**: 操作数的值增加或减少1
- 后缀形式**: 表达式expr++和expr--的值是操作数加1或减1前的**原值**
- 前缀形式**: 表达式++expr和--expr的值是操作数加1或减1后的**新值**

```
int i=3, j, a, b=1, c=2;  
j = ++i;           // i==4, (++i)==4, j==4  
j = i++;          // i==5, (i++)==4, j==4  
j = -i++;         // -(i++), i==6, (i++)==5, j==-5  
j = i++*2;         // (i++)*2, i==7, (i++)==6, j==12  
a = (b+c)++;      // 错误! 操作数(b+c)不是左值  
a = 34++;          // 错误! 操作数34不是左值  
j = ++i++;         // 错误! ++(i++), 操作数(i++)不是左值
```



# 赋值运算符

优先级	运算符	描述	举例	等价于	操作数	结合性
14	=	基本赋值	$a = b$			二元 右结合
	+=	加法赋值	$a += b$	$a = a + b$		
	-=	减法赋值	$a -= b$	$a = a - b$		
	*=	乘法赋值	$a *= b$	$a = a * b$		
	/=	除法赋值	$a /= b$	$a = a / b$		
	%=	求余赋值	$a \%= b$	$a = a \% b$		
	&=	按位与赋值	$a \&= b$	$a = a \& b$		
	=	按位或赋值	$a  = b$	$a = a   b$		
	^=	按位异或赋值	$a ^= b$	$a = a ^ b$		
	<<=	按位左移赋值	$a <<= b$	$a = a << b$		
	>>=	按位右移赋值	$a >>= b$	$a = a >> b$		



# 赋值表达式

## ■ 基本赋值运算

- 将右操作数的值赋给左操作数
- 左操作数必须是**左值**
- 左右操作数类型不同时，先将右操作数的值**隐式转换**为左操作数类型再赋值
- 赋值表达式的值是赋值后左操作数的值，即所赋之值
- **副作用：**左操作数的值发生改变

## ■ 复合赋值运算

- 基本赋值运算与二元算术运算、二元位运算的结合
- 复合赋值运算符的优先级与基本赋值运算符相同

$i += 2;$	$\Leftrightarrow$	$i = i + 2;$
$a[i*j+b] += c[j];$	$\Leftrightarrow$	$a[i*j+b] = a[i*j+b] + c[j];$
$x *= y + 1;$	$\Leftrightarrow$	$x = x * (y + 1);$



# 关系与逻辑运算符

优先级	运算符	描述	举例	操作数	结合性
2	!	逻辑非	$!a$	一元	右结合
6	<	小于关系	$a < b$	二元	左结合
	$\leq$	小于等于关系	$a \leq b$		
	>	大于关系	$a > b$		
	$\geq$	大于等于关系	$a \geq b$		
	$=$	相等关系	$a == b$		
7	$\neq$	不等关系	$a \neq b$		
11	$\&\&$	逻辑与	$a \&\& b$		
12	$\  \ $	逻辑或	$a \  \  b$		



# 关系与逻辑表达式

## ■ 关系表达式

- 关系表达式的值是以1和0表示的逻辑值，1—“真”，0—“假”
- 注意区分相等关系运算符==和赋值运算符=

## ■ 逻辑表达式

- 逻辑表达式的值是以1和0表示的逻辑值，1—“真”，0—“假”
- 若将操作数的值用作逻辑值，非零⇒“真”，零⇒“假”
- 在a&&b中，若a为“假”，则不再判断b      short cut
- 在a||b中，若a为“真”，则不再判断b

```
leap = (y%4==0&&y%100!=0) || (y%400==0); // 判断是否闰年
f = a>b>c; // f = (a>b)>c;
z = (m=a>b)&&(n=c>d); // n是否被赋值，取决于a>b是否为“真”
```

if (!a) ...	↔	if (a==0) ...	// !a与a==0的逻辑值等价
if (a) ...	↔	if (a!=0) ...	// a与a!=0的逻辑值等价



# 条件运算符和条件表达式

## ■ 条件运算符

优先级	运算符	描述	举例	操作数	结合性
13	?:	条件运算	a ? b : c	三元	右结合

## ■ 条件表达式

expr1 ? expr2 : expr3

- 先计算expr1的值
- 若expr1的值**非零**（“真”），则计算expr2的值，并作为条件表达式的值
- 若expr1的值**为零**（“假”），则计算expr3的值，并作为条件表达式的值
- 用于简单的选择结构，使代码更加紧凑、简洁

```
z = (a > b) ? a : b; // 求a和b的最大值，并存入z
ch = (ch>='A' && ch<='Z') ? (ch+32) : ch; // 将大写字母转为小写字母
for (i=0; i<n; i++) // 输出数组元素，每10个元素一行
    printf("%6d%c", a[i], (i%10==9 || i==n-1) ? '\n' : ' ');
```



# 逗号运算符和逗号表达式

## ■ 逗号运算符

优先级	运算符	描述	举例	操作数	结合性
15	,	逗号运算	a, b	二元	左结合

## ■ 逗号表达式

expr1, expr2

- 先计算expr1的值，再计算expr2的值，并将expr2的值作为逗号表达式的值
- 用于将多个表达式合并为一个表达式

```
temp = a, a = b, b = temp;           // 一条表达式语句，交换变量a和b的值
for (fact=1.0, i=1; i<=n; ++i)      // 一个逗号表达式，初始化两个变量fact和i
    fact *= i;                         // 求n的阶乘
```



# 求类型字节数运算符

## ■ 求类型字节数运算符

优先级	运算符	描述	举例	操作数	结合性
2	<code>sizeof</code>	求类型字节数	<code>sizeof (int)</code> <code>sizeof expr</code>	一元	右结合

## ■ 求值规则

- 操作数可以是数据类型或数据对象
- 若操作数是数据类型，则返回该数据类型的字节数
- 若操作数是数据对象，则判断数据对象的类型，并返回该数据类型的字节数
- 若数据对象是表达式，则**不对表达式求值，仅判断表达式类型**
- 运算结果是无符号整型数据



# 强制类型转换运算符

## ■ 强制类型转换运算符

优先级	运算符	描述	举例	操作数	结合性
2	(类型)	强制类型转换	(int) b	一元	右结合

## ■ 求值规则

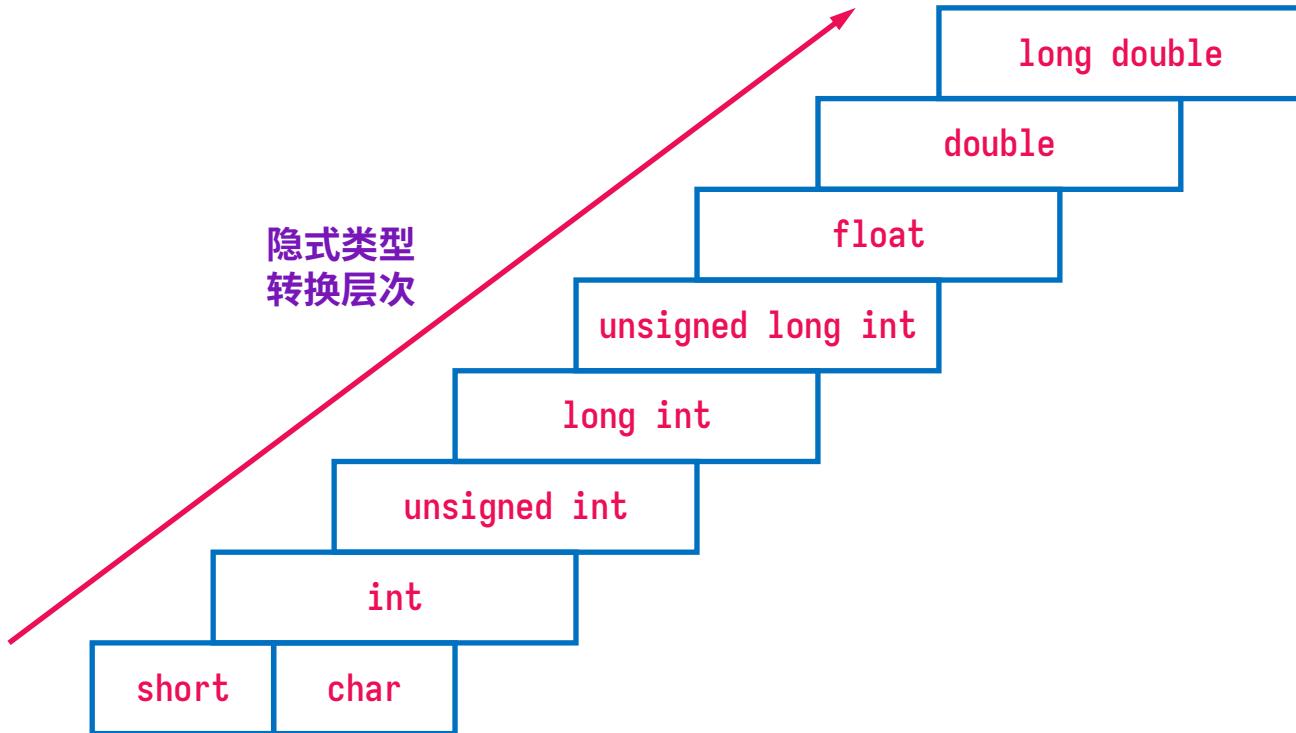
- 将操作数的值转换成指定数据类型的运算结果
- 操作数自身的数据类型和值不会发生变化
- 多用于隐式类型转换不能满足要求的情况

```
float x=2.8, y=3.7, z;
int a, b, c, d=5, e=2;
a = (int)x+y;                      // a=5, 2+3.7 => 5
b = (int)(x+y);                    // b=6, (int)(6.5) => 6
c = (int)x;
printf("x=%f, c=%d\n", x, c);    // x=2.800000, c=2
z = (float)d/e;                   // z=2.5, 5.0/2 => 2.5
```



- 运算符简介
- 常用表达式
- 隐式类型转换规则
- 表达式的运算次序

# 隐式类型转换规则



算术类型的隐式类型转换规则



# 隐式类型转换举例

## ■ 例3.2-a：隐式类型转换问题举例。

```
#include <stdio.h>

int main()
{
    float a = 0.1;
    printf("%.15f", a-0.1);
    return 0;
}
```

运行结果：

0.00000001490116

### 程序解析：

- 1 当float类型的变量a和double类型的常量0.1进行算术运算时，按照隐式类型转换规则，应先将a的值转换为double类型数据，然而float类型与double类型相比，其有效数字更少，误差更大，无法转换为以double类型精度表达的0.1；
- 2 由于float和double类型存储误差的不同，表达式a-0.1的运算结果不是0.0，而是体现出两种数据类型在表达0.1时的差异。



# 隐式类型转换举例

## ■ 例3.2-1：隐式类型转换问题举例。

```
#include <stdio.h>

int main()
{
    unsigned uint = 0;
    int      i = -1;

    if (i < uint) ←
        printf("-1当然小于0");
    else
        printf("-1并不小于0");

    return 0;
}
```

运行结果：

-1并不小于0

结果分析：

uint	0000 0000 0000 0000 0000 0000 0000 0000
i	1111 1111 1111 1111 1111 1111 1111 1111

### 程序解析：

- 1 变量 `uint` 的数据类型 `unsigned` 是 `unsigned int` 的简写，即无符号整型；
- 2 当 `unsigned` 类型的变量 `unit` 和 `int` 类型的变量 `i` 进行关系运算时，按照隐式类型转换规则，先将 `i` 的值转换为一个 `unsigned` 类型数据，而 `-1` 的补码是所有位全为 `1` 的二进制数，将转换得到一个很大的无符号数，因此 `i < uint` 的关系运算结果为“假”；
- 3 类型转换将操作数的值转换得到一个目标类型的数据，并用于运算求值，而不会改变操作数本身的数据类型。



- 运算符简介
- 常用表达式
- 隐式类型转换规则
- 表达式的运算次序



# 运算符的优先级和结合性

优先级	运算符	类别	操作数	结合性
1	<code>++ -- () [] . -&gt;</code>			左结合
2	<code>++ -- + - ! ~ (类型) * &amp; sizeof</code>		一元	右结合
3	<code>* / %</code>	算术运算符		
4	<code>+ -</code>			
5	<code>&lt;&lt; &gt;&gt;</code>	位运算符		
6	<code>&lt; &lt;= &gt; &gt;=</code>			
7	<code>== !=</code>	关系运算符		
8	<code>&amp;</code>			
9	<code>^</code>	位运算符		
10	<code> </code>			
11	<code>&amp;&amp;</code>	逻辑运算符		
12	<code>  </code>			
13	<code>?:</code>	条件运算符	三元	
14	<code>= += -= *= /= %= &lt;=&gt;= &amp;= ^=  =</code>		二元	右结合
15	<code>,</code>	逗号运算符	二元	左结合



# 优先级与结合性举例

## ■ 例3.2-2：验证运算符的优先级与结合性。

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

```
int main()
```

```
{
```

```
    int a;
```

```
    printf("%d\n", 3>2);
```

```
    printf("%d\n", 3>2>1);
```

```
    printf("%d\n", a=3>2>1);
```

```
    return 0;
```

```
}
```

### 程序解析：

- 1 关系表达式的值以1表示“真”；
- 2 按照关系运算符的结合性和求值规则分步计算， $3>2$ 的值为1， $(3>2)>1$ 的值为0；
- 3 赋值表达式的值即为所赋之值。

### 运行结果：

```
1  
0  
0
```

## ■ 例3.2-b：连续输入字符并存入数组中，直至遇到回车或数组已满。

```
char s[20], c;
```

```
for (i=0; (i<20) && ((c=getchar()) != '\n'); ++i)
```

```
    s[i] = c;
```

### 程序解析：

- 1 适当使用()明确运算次序，有助于使代码条理清晰，减少运算次序、数据类型等方面错误。



# 控制语句与基本算法



## ■ C语言程序语句

### ■ 分支语句

### ■ 循环语句

### ■ 中断与跳转语句

### ■ 基本算法



# 语句 (Statement)

## ■ 表达式语句

- 表达式后跟随一个分号；
- 表达式缺省时，称为空语句
- 函数调用也是表达式的一种，函数调用后跟随分号也是表达式语句

## ■ 复合语句

- 以{}括起来的声明与语句序列
- 复合语句可用于任何需要单个语句的场合
- 表示复合语句结束的}后不需要加分号，若有分号，则是另一条空语句

## ■ 控制语句

- 分支语句（选择语句）：if-else语句、switch语句
- 循环语句：for语句、while语句、do-while语句
- 中断和跳转语句：break语句、continue语句、goto语句、return语句



# 语句类型举例

## ■ 例3.3-a：语句类型举例。

```
#include <stdio.h>

int main()
{
    char c, i;                                // 函数体复合语句开始
                                                // 变量的声明，不属于语句

    for (i=0; i<5; i++);
    {
        c = getchar();                         // 循环控制语句：for语句，其循环体为空语句
                                                // 复合语句开始，不是for语句的循环体
        putchar(c-'a'+'A');                   // 赋值语句，表达式语句的一种
                                                // 函数调用语句，表达式语句的一种
    }                                         // 复合语句结束，结尾不需要加分号

    return 0;                                  // 跳转控制语句：return语句，用于函数调用结束返回
}                                         // 函数体复合语句结束
```



- C语言程序语句
- 分支语句
- 循环语句
- 中断与跳转语句
- 基本算法

# if-else语句

## ■ 一般形式

`if (表达式)`

语句1

`else`

语句2

## ■ 省略else部分的形式

`if (表达式)`

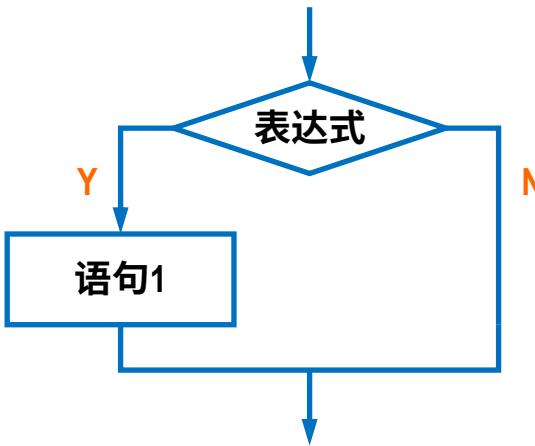
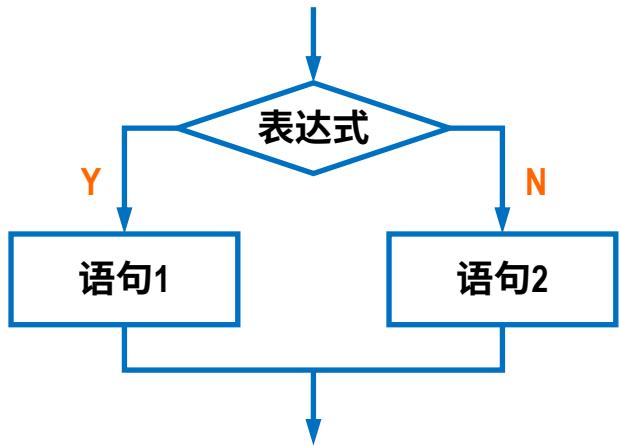
语句1

## ■ 说明

- 二分支语句
- 可通过**嵌套**实现复杂的选择逻辑

## ■ 说明

- 表达式为“假”时无任何操作
- 等价于语句2为空语句





# if-else语句举例

## ■ 例3.3-1：输入三个整数并求其最大值。

```
#include <stdio.h>

int main()
{
    int num1, num2, num3, max;

    printf("请输入三个整数: ");
    scanf("%d%d%d", &num1, &num2, &num3);

    if (num1 > num2) // 求num1和num2的最大值，并存入max
        max = num1;
    else
        max = num2;

    if (num3 > max) // 求num3和max的最大值，并存入max
        max = num3;

    printf("最大值为: %d\n", max);

    return 0;
}
```

### 程序解析:

- 1 格式化输入多个除字符型(%c)以外的数据时，如果没有指定分隔字符，则使用空白字符分隔，包括空格、换行符、制表符；
- 2 分支比较简单的if-else语句也可以使用条件表达式来实现，例如， $max=(num1>num2)?num1:num2;$ 。
- 3 若if语句条件为“假”时不需要任何操作，可以省略else部分。

### 运行结果一:

```
请输入三个整数: 3 9 6↵
最大值为: 9
```

### 运行结果二:

```
请输入三个整数: -3 0 8↵
最大值为: 8
```



# if-else语句举例

## ■ 例3.3-2：输入三个整数并按升序排列。

```
#include <stdio.h>

int main()
{
    int num1, num2, num3, temp;

    printf("请输入三个整数: ");
    scanf("%d%d%d", &num1, &num2, &num3);

    if (num1 > num2) {
        temp = num1;
        num1 = num2;
        num2 = temp;
    }
    if (num2 > num3) {
        temp = num2; num2 = num3; num3 = temp;
    }
    if (num1 > num2) {temp = num1; num1 = num2; num2 = temp;}

    printf("按升序排列为: %d,%d,%d\n", num1, num2, num3);

    return 0;
}
```

### 程序解析:

- 1 借助临时变量进行三次赋值，是交换两个变量值的常用方法；
- 2 C语言对换行等书写格式要求非常宽松，只要不会引发语法上的混淆就能够成功编译，但是仍然建议遵守良好的编程风格；
- 3 格式化输出时，格式字符串中普通字符将原样输出，如逗号。

### 运行结果一:

```
请输入三个整数: 3 9 6
按升序排列为: 3,6,9
```

### 运行结果二:

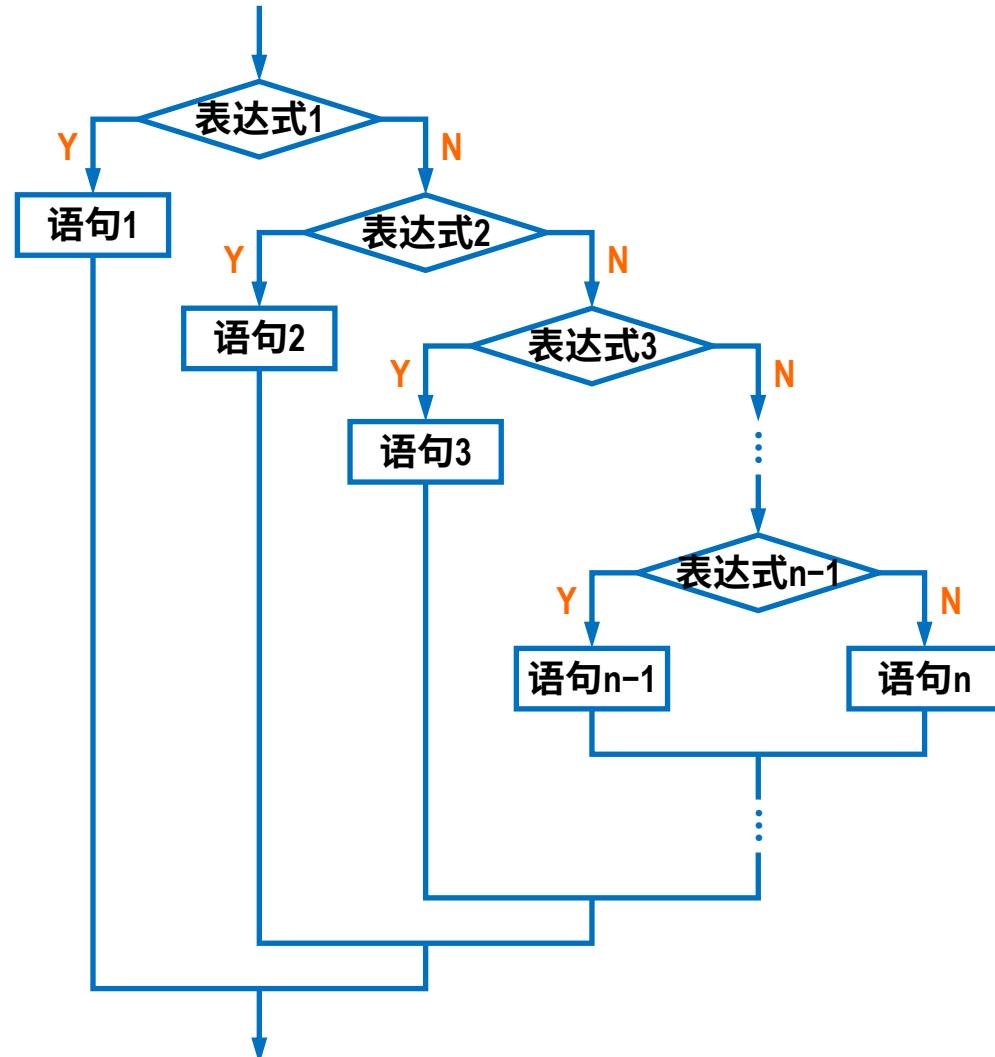
```
请输入三个整数: 8 0 -3
按升序排列为: -3,0,8
```

# 多重if-else语句

## 一般形式

```

if (表达式1)
    语句1
else if (表达式2)
    语句2
else if (表达式3)
    语句3
...
...
else if (表达式n-1)
    语句n-1
else
    语句n
    
```



## 说明

- if-else语句嵌套的特殊形式
- 最后的else部分是可选的
- 各表达式依次求值，有优先级



# 多重if-else语句举例

## ■ 例3.3-3：输入身高和体重，计算BMI并判断体型是否健康。

```
#include <stdio.h>

int main()
{
    float height, bmi;
    int weight;

    printf("输入身高 (米) : ");
    scanf("%f", &height);
    printf("输入体重 (千克) : ");
    scanf("%d", &weight);
    bmi = weight / (height * height);
    printf("BMI指数为: %f\n", bmi);

    if (bmi < 18.5)
        printf("体重过轻");
    else if (bmi < 24.9) // 18.5 ≤ bmi < 24.9
        printf("正常范围, 注意保持");
    else if (bmi < 29.9) // 24.9 ≤ bmi < 29.9
        printf("体重偏重");
    else // bmi ≥ 29.9
        printf("太重了! ");

    return 0;
}
```

### 程序解析:

1 利用嵌套if-else语句中条件判断的优先顺序，可以简化条件判断表达式的书写，但不利于程序的阅读和理解。

### 运行结果一:

```
输入身高 (米) : 1.85
输入体重 (千克) : 80
BMI指数为: 23.374725
正常范围, 注意保持
```

### 运行结果二:

```
输入身高 (米) : 1.75
输入体重 (千克) : 85
BMI指数为: 27.755102
体重偏重
```



# if-else语句的嵌套

## ■ 一般形式

```
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
```

## ■ if-else配对

- else总是与前面最接近的未配对if进行匹配
- 缩进不影响if-else配对关系

```
if (n > 0)
    if (a > b)
        z = a;
else
    z = b;
```

- 可使用{}明确if-else配对关系

```
if (n > 0) {
    if (a > b)
        z = a;
} else
    z = b;
```

## ■ 说明

- if-else语句中包含if-else语句
- else部分都是可选的
- 注意if-else配对关系是否正确



# if-else语句的嵌套举例

## ■ 例3.3-4：求一元二次方程 $ax^2+bx+c=0$ ( $a \neq 0$ ) 的解。

```
#include <stdio.h>
#include <math.h> // 数学库函数头文件

int main()
{
    float a, b, c, disc, x1, x2, p, q;
    scanf("%f%f%f", &a, &b, &c);
    disc = b*b - 4*a*c;
    if (fabs(disc) <= 1e-6)
        printf("x1=x2=% .2f\n", -b/(2*a));
    else {
        if (disc > 1e-6) {
            x1 = (-b+sqrt(disc)) / (2*a);
            x2 = (-b-sqrt(disc)) / (2*a);
            printf("x1=% .2f\nx2=% .2f\n", x1, x2);
        } else {
            p= -b / (2*a);
            q= sqrt(fabs(disc)) / (2*a);
            printf("x1=% .2f% .2fi\n", p, q);
            printf("x2=% .2f% .2fi\n", p, q);
        }
    }
    return 0;
}
```

### 运行结果一：

4 6 3↓  
x1=-0.75+0.43i  
x2=-0.75-0.43i

### 运行结果二：

1 4 4↓  
x1=x2=-2.00

### 运行结果三：

2 5 1↓  
x1=-0.22  
x2=-2.28

### 运行结果四：

4 3 0↓  
x1=0.00  
x2=-0.75



# switch语句

## ■ 一般形式

```
switch (表达式) {  
    case 常量表达式1: 语句组 [break;]  
    case 常量表达式2: 语句组 [break;]  
    ... ...  
    case 常量表达式n: 语句组 [break;]  
    default:          语句组 [break;]  
}
```

## ■ 表达式要求

- switch后表达式必须为**整数类型**  
(整型、字符型、枚举类型)
- case后常量表达式必须为**整数类  
型常量或常量表达式**

## ■ 分支入口

- 若switch后表达式与某一case后常量表达式的值相等，则跳至该case后开始执行
- 若switch后表达式与所有case后常量表达式的值都不相等，则跳至default后开始执行
- 可以没有default分支

## ■ 语句出口

- 情况一：遇到break语句后跳出
- 情况二：依次执行后续**所有语句**直至switch语句结束



# switch语句举例

## ■ 例3.3-5：考试成绩分级。

```
#include <stdio.h>

int main()
{
    int score;

    scanf("%d", &score);
    if (score >= 0 && score <= 100) {
        switch (score / 10) {
            case 10:
            case 9: printf("优秀"); break;
            case 8: printf("良好"); break;
            case 7: printf("中等"); break;
            case 6: printf("及格"); break;
            default: printf("不及格"); break;
        }
    } else
        printf("输入错误");

    return 0;
}
```

### 程序解析:

- 利用整数除法的运算规则，以求得的商表示各等级对应的分数段；
- 通过适当缺省break语句，可实现多个case入口共用全部或部分语句；
- switch语句是多分支语句，只需计算一次表达式的值，就可以引导程序进入不同的分支，但是需要把分支条件转换为整型数据表示，不如if-else语句灵活方便。

### 运行结果一:

92↙

优秀

### 运行结果二:

55↙

不及格

### 运行结果三:

83↙

良好

### 运行结果四:

-100↙

输入错误



# switch语句举例

## ■ 例3.3-6：菜单显示和选择。

```
for (;;) { // 不限循环次数，由菜单项结束循环
    printf("成绩管理系统\n");
    printf("1.录入成绩\n");
    printf("2.修改成绩\n");
    printf("3.查询成绩\n");
    printf("0.退出系统\n");
    printf("请选择0-3: ");

    scanf("%d", &num); // 输入菜单选项编号，以整型表示
    switch (num) { // 菜单选项编号也可以使用字符型
        case 1:
            Input(); break; // 录入成绩的语句组
        case 2:
            Edit(); break; // 修改成绩的语句组
        case 3:
            Query(); break; // 查询成绩的语句组
        case 0:
            return 0; // 退出程序
    }
}
```

### 运行结果：

```
成绩管理系统
1.录入成绩
2.修改成绩
3.查询成绩
0.退出系统
请选择0-3: 1<
(录入成绩过程)
成绩管理系统
1.录入成绩
2.修改成绩
3.查询成绩
0.退出系统
请选择0-3: 2<
(修改成绩过程)
成绩管理系统
1.录入成绩
2.修改成绩
3.查询成绩
0.退出系统
请选择0-3: 0<
```



- C语言程序语句
- 分支语句
- **循环语句**
- 中断与跳转语句
- 基本算法

# for循环

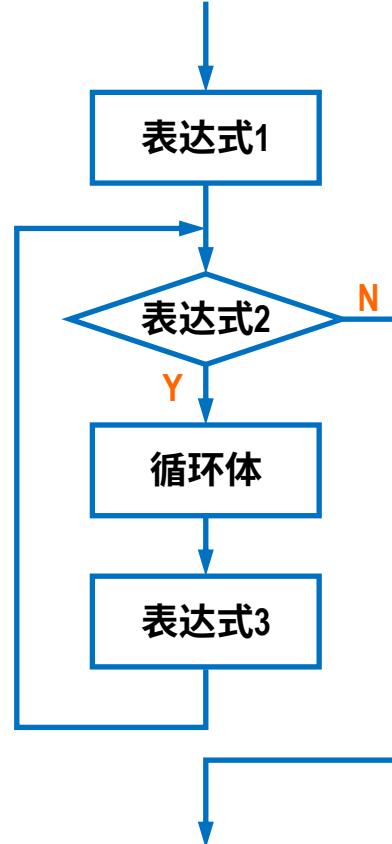
## ■ 一般形式

```
for (表达式1; 表达式2; 表达式3)  
    循环体语句
```

## ■ 说明

- 表达式1：循环控制变量初始化
- 表达式2：循环条件判断
- 表达式3：循环控制变量值的修改
- 三个表达式都可以省略，两个分号不可省略
- 若表达式2省略，则循环条件缺省为“真”

```
// 对1~100之间的自然数累加求和  
for (i=1; i<=100; i++)  
    sum += i;
```



# while循环

## ■ 一般形式

while (表达式)

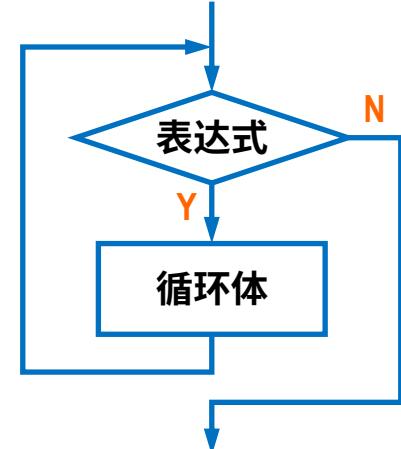
    循环体语句

## ■ 说明

- 首先求表达式的值
- 若表达式的值**非零**（“真”），则继续循环
- 若表达式的值**为零**（“假”），则循环结束
- 若表达式首次求值为零，则循环体一次都不执行

// 对1~100之间的自然数累加求和

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





# while循环举例

## ■ 例3.3-7：辗转相除法求最大公约数。

```
#include <stdio.h>

int main()
{
    int m, n, temp;

    printf("请输入两个整数:\n");
    scanf("%d,%d", &m, &n); // 输入数据以逗号分隔

    while (temp = m % n) { // 赋值表达式的值作为判断条件
        m = n;
        n = temp;
    }

    printf("最大公约数为%d\n", n);

    return 0;
}
```

### 程序解析:

1 格式化输入时，格式字符串中普通字符应原样输入，如逗号；

2 赋值表达式的值即所赋之值，可以直接作为判断条件，等价于判断其值是否不等于0。

### 运行结果:

请输入两个整数：

115,25

最大公约数为5

### 变量监测:

	m	n	temp
初始状态	??	??	??
输入数据	115	25	??
开始循环			
求余判断	115	25	15
循环体	25	15	15
求余判断	25	15	10
循环体	15	10	10
求余判断	15	10	5
循环体	10	5	5
求余判断	10	5	0
循环结束			

# do-while循环

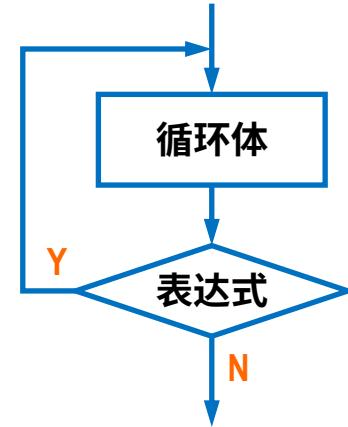
## ■ 一般形式

```
do
    循环体语句
    while (表达式);
```

## ■ 说明

- 首先执行循环体，然后求表达式的值
- 若表达式的值**非零**（“真”），则继续循环
- 若表达式的值**为零**（“假”），则循环结束
- 循环体至少会执行一次

```
// 对1~100之间的自然数累加求和
i = 1;
do
    sum += i++;
    while (i <= 100);
```





# do-while循环举例

## ■ 例3.3-8：整数按位倒序输出。

```
#include <stdio.h>

int main()
{
    int n;

    scanf("%d", &n);

    do {
        printf("%d", n%10);
        n = n / 10;
    } while (n != 0);

    return 0;
}
```

### 程序解析：

- 1 灵活利用整数除法，在处理整数问题时有其便利之处；
- 2 条件判断 `while(n!=0)` 也可以简化为 `while(n);`  
// 即使 `n==0` 也要输出一位  
// 输出当前最低位  
// 移除当前最低位  
// 末尾的分号必不可少
- 3 `do-while` 语句条件判断后的分号作为语句结束标志不可缺少。

### 运行结果一：

1234↙  
4321

### 运行结果二：

0↙  
0

无法正确处理  
负整数输入

### 运行结果三：

-1234↙  
-4-3-2-1

### 思考拓展：

修改程序支持负整数。



# 多层循环举例

## ■ 例3.3-9：比较数组中是否有相同元素。

```
#include <stdio.h>

int main()
{
    int a[5]={3,5,7,9,11}, i, j;
    int b[6]={1,2,3,4,5,6}, found=0;

    for (i=0; i<5; i++)           // 数组a的下标范围: 0~4
        for (j=0; j<6; j++)       // 数组b的下标范围: 0~5
            if (a[i] == b[j])
                found = 1;          // 是否存在相同元素的标志

    if (found)
        printf("a与b有相同的元素");
    else
        printf("a与b没有相同的元素");

    return 0;
}
```

### 程序解析：

- 1 数组下标范围为 $0 \sim \text{长度}-1$ ，利用两层循环可以遍历两个数组，完成两个数组的元素所有可能的两两比较；
- 2 变量`found`表示是否找到相同元素的标志，初始值为`0`，表示尚未找到，当找到时，将`found`赋值为`1`，用于后续代码的分支语句；
- 3 当找到相同元素后，循环仍然继续进行，直至完成全部`30`次循环，若希望提前中止循环，需要使用`break`语句。

### 运行结果：

a与b有相同的元素



# 提纲

- C语言程序语句
- 分支语句
- 循环语句
- 中断与跳转语句
- 基本算法



# 中断与跳转语句

## ■ break语句

`break;`

- 在`switch`语句中，提前跳出`switch`语句
- 在循环语句中，提前结束**本层**循环

## ■ continue语句

`continue;`

- 在循环语句中，提前结束**本次**循环

## ■ goto语句

`goto 语句标号;`

- 无条件跳转到语句标号所标识的语句处执行
- **语句标号：**标识符后加一个冒号，用于标识一条语句



# break语句举例

## ■ 例3.3-10：对1~100之间的自然数累加，求加到哪个数时累加和首次达到1000。

```
#include <stdio.h>

int main()
{
    int i, sum=0;

    for (i=1; i<=100; i++){
        sum += i;
        if (sum >= 1000) // 提前结束循环的条件
            break; // 提前结束本层循环
    }

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

    return 0;
}
```

### 程序解析：

1 循环中的break语句一般放在分支语句中，指定提前结束循环的条件，分支语句条件判断为“真”时，提前结束本层循环；

2 执行break语句将提前结束本层循环语句，与其所在分支语句的层次无关，当存在多层循环和分支语句嵌套时，应当注意程序将跳转到何处并继续执行。

### 运行结果：

i=45, sum=1035



# break语句举例

## ■ 例3.3-11：比较数组中是否有相同元素。

```
#include <stdio.h>
int main()
{
    int a[5]={3,5,7,9,11}, i, j;
    int b[6]={1,2,3,4,5,6}, found=0;
    for (i=0; i<5; i++) {
        for (j=0; j<6; j++) {
            if (a[i] == b[j]) {
                found = 1;
                break;
            }
        }
        if (found)
            break;
    }
    if (found)
        printf("a与b有相同的元素");
    else
        printf("a与b没有相同的元素");
    return 0;
}
```

程序解析：

- 当找到相同元素时，可以使用break语句提前结束循环；
- 当找到相同元素时，首先将变量found赋值为1，表示已找到相同元素，然后中止内层循环，继续执行外层循环体；
- 在外层循环中，根据变量found的值判断是否已找到相同元素，若为“真”，则中止外层循环；
- 每条break语句只能中止本层循环，中止多层循环需要使用多条break语句接力完成。

## 运行结果：

a与b有相同的元素



# continue语句举例

■ 例3.3-12：对1~100之间的自然数累加求和，但跳过所有3的倍数。

```
#include <stdio.h>

int main()
{
    int i, sum=0;

    for (i=1; i<=100; i++) {
        if (i%3 == 0) // 若i是3的倍数
            continue; // 提前结束本次循环
        sum += i;
    }

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

    return 0;
}
```

## 程序解析：

- 1 `continue`语句一般放在分支语句中，条件判断为“真”时，提前结束本次循环；
- 2 当*i*是3的倍数时，跳过循环体中后续累加语句，执行for语句中的表示式3；
- 3 `continue`语句的作用一般可以使用`if-else`语句替代，但是合理使用`continue`语句可以使程序简洁易读。

## 运行结果：

sum=3367



# goto语句举例

## ■ 例3.3-13：对1~100之间的自然数累加求和。

```
#include <stdio.h>

int main()
{
    int n=1, sum=0;

loop:   sum += n;           // loop为语句标号
    n++;
    if (n <= 100)            // 跳转条件控制
        goto loop;          // 跳转到loop标号语句处执行
    printf("sum=%d\n", sum);

    return 0;
}
```

### 程序解析：

- 1 goto语句一般放在分支语句中，当条件判断为“真”时，直接跳转至目标语句；
- 2 goto语句和分支语句配合，可以实现循环结构；
- 3 goto语句可以跳转至本函数内部的任意语句，合理使用goto语句可以简化代码，提高执行效率，同时滥用goto语句也会破坏程序的结构化设计。

### 运行结果：

sum=5050



# goto语句举例

## ■ 例3.3-14：比较数组中是否有相同元素，若有，则分别输出相同元素的下标。

```
#include <stdio.h>

int main()
{
    int a[5]={3,5,7,9,11}, i, j;
    int b[6]={1,2,3,4,5,6};

    for (i=0; i<5; i++)
        for (j=0; j<6; j++)
            if (a[i] == b[j])
                goto found; // 直接跳出多层循环

    printf("没找到...");
    return 1; // 函数返回，返回值为1

found: printf("找到了，在a[%d], b[%d]", i, j);
    return 0;
}
```

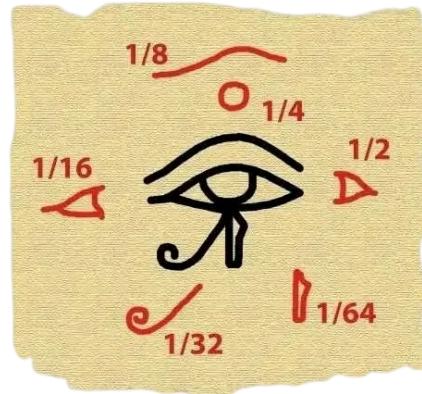
### 程序解析：

- 1 goto语句可用于直接跳出多层循环语句；
- 2 return语句表示函数结束返回，不再执行函数体中后续语句，main函数返回表示程序执行结束；
- 3 goto语句和分支语句配合，可以实现选择结构；
- 4 变量声明和复合语句结尾处的}不是语句，如需在此添加语句标号，需要增加空语句。

### 运行结果：

找到了，在a[0], b[2]

- C语言程序语句
- 分支语句
- 循环语句
- 中断与跳转语句
- 基本算法



*The Eye of Horus and Egyptian Fractions*



# 基本算法

## ■ 算法 (Algorithm)

- 解决问题的思路
- 确定性、有限性、有效性、零个或多个输入、至少一个输出

## ■ 穷举法 (枚举法)

- 穷尽解空间以寻找问题的解，充分利用计算机的强大运算能力

## ■ 递推法 (顺推法、逆推法)

- 从已知初始条件入手，根据递推关系推出中间结果，进而得到最终结果

## ■ 贪心法 (贪婪法)

- 针对能分成多阶段求解的问题，每个阶段总是做出当前看来最好的选择
- 不保证对所有问题都能取得整体最优解

## ■ 分治法

- 将原问题划分成多个规模较小而结构与原问题类似的子问题
- 逐层划分直至能直接求解的程度，然后将子问题结果合并成原问题的解



# 穷举法举例：百钱买百鸡

## ■ 例3.3-16：百钱买百鸡问题。

今有鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。凡百钱买鸡百只，问鸡翁、母、雏各几何？（张丘建《算经》）

■ 解：设公鸡、母鸡、小鸡数量分别为 $x, y, z$ ，按题意有：

$$\begin{cases} x + y + z = 100 \\ 5x + 3y + z/3 = 100 \end{cases}$$

由于 $x, y, z$ 都是非负整数，可知：  
 $0 \leq x \leq 20$ ,  $0 \leq y \leq 33$ ,  $z$ 是3的倍数。

运行结果：

0	25	75
4	18	78
8	11	81
12	4	84

```
#include <stdio.h>

int main()
{
    int x, y, z;

    for (x=0; x<=20; x++)
        for (y=0; y<=33; y++) {
            z = 100 - x - y;
            if ((5*x+3*y+z/3==100) && (z%3==0))
                printf("%6d%6d%6d\n", x, y, z);
        }
    return 0;
}
```



# 递推法举例：斐波那契数列

## ■ 例3.3-18：计算并输出斐波那契数列前20项。

$$f(1) = f(2) = 1; f(n) = f(n-1) + f(n-2), n \geq 3$$

```
#include <stdio.h>

int main()
{
    int f1=1, f2=1; // 定义并初始化数列前2项
    int i; // 定义循环控制变量i

    for (i=1; i<=10; i++) { // 每组2项，10组共20项
        printf("%6d%6d", f1, f2); // 输出当前2项
        if(i%2 == 0)
            printf("\n"); // 每输出2次（4个数）换行
        f1 += f2; // 计算下一个奇数项
        f2 += f1; // 计算下一个偶数项
    }

    return 0;
}
```

### 运行结果：

1	1	2	3
5	8	13	21
34	55	89	144
233	377	610	987
1597	2584	4181	6765

### 变量监测：

	i	f1	f2
初始状态	??	1	1
循环01	1	1	1
输出	1	1	1
计算	1	2	3
循环02	2	2	3
输出	2	2	3
计算	2	5	8
...	...		



# 递推法举例：自然常数

## ■ 例3.3-19：求自然常数e的近似值。

$$e = \sum_{i=0}^{\infty} \frac{1}{i!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots = 2.7182818284590452\dots$$

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

```
int main() {
    int i, j, n, fact=1;
    double sum=1.0;

    printf("n=");
    scanf("%d", &n);
    for (i=1; i<=n; i++) {
        fact *= i;
        sum += 1.0 / fact;
    }
    printf("e=%.16f\n", sum);

    return 0;
}
```

### 程序解析：

1 阶乘使用int类型变量表示，需要注意溢出问题；

2 根据前后两项之间的数学关系简化运算，充分利用运算结果，提高算法效率；

// 阶乘：利用(i-1)!求i!  
// 避免整数除法

// 小数点后保留16位数字

3 左操作数为浮点型常量1.0，以避免整数除法。

### 运行结果一：

n=12↙  
e=2.7182818282861687

### 运行结果二：

n=20↙  
e=2.7182818346494479

### 运行结果三：

n=34↙  
e=1.#INF00000000000000

### 结果分析：

32位int类型可表达的最大阶乘为12！，当i>12时，fact开始溢出，至i==34时溢出为0。



# 贪心法举例：最少纸币支付

## ■ 例3.3-20：最少纸币支付问题。

假设有7种面值的纸币1元、2元、5元、10元、20元、50元、100元各5、5、0、0、2、1、3张。现需用这些纸币支付特定金额，问最少需要几张纸币？

```
#include <stdio.h>
#define N 7          // 符号常量便于多次引用
int main()
{
    int value[N]={1,2,5,10,20,50,100};
    int a[N]={5,5,0,0,2,1,3};
    int money, i, num=0, c=0;

    printf("请输入需要支付的总额: ");
    scanf("%d", &money);

    for (i=N-1; i>=0; i--) {
        if (money/value[i] <= a[i])
            c = money / value[i];
        else
            c = a[i];
        money = money - c * value[i];
        num += c;
    }
}
```

```
if (money > 0)
    printf("无解! ");
else
    printf("需要纸币张数: %d\n", num);
return 0;
}
```

### 运行结果一：

```
请输入需要支付的总额: 391
需要纸币张数: 7
```

### 运行结果二：

```
请输入需要支付的总额: 405
需要纸币张数: 16
```

### 运行结果三：

```
请输入需要支付的总额: 698
无解!
```



# 数组与批量数据处理



# 提纲

- 一维数组

- 多维数组

- 字符数组



# 一维数组

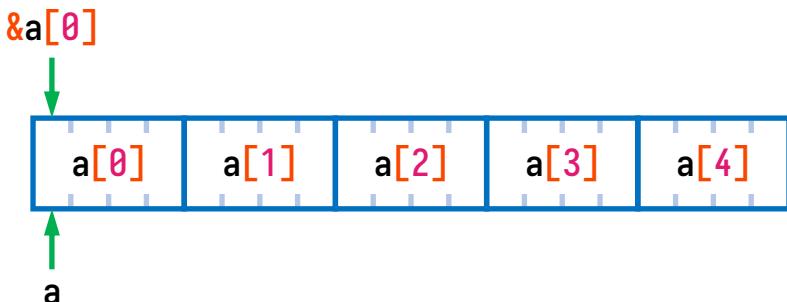
## ■ 一维数组的定义与初始化

- 数组只能在定义时进行整体初始化，其后只能对元素分别进行赋值

```
int a[5]={1,2,3,4,5};           // 正确，数组初始化  
int a[5]={1,2,3};             // 正确，对前3个元素初始化，其余元素初值为0  
int a[5]; a[0]=1, a[1]=2, a[2]=3, a[3]=4, a[4]=5; // 正确，逐个元素赋值  
int a[5]; a[5]={1,2,3,4,5};   // 错误，a[5]表示下标为5的元素，已越界  
int a[5]; a={1,2,3,4,5};     // 错误，只能在定义数组时进行初始化  
int a[5]; a=1;               // 错误，数组名不能作为左值
```

## ■ 数组与地址

- 数组名表示数组存储空间的起始地址，不能被修改，不能作为左值





# 一维数组举例

## ■ 例3.4-1：数组初始化、运算与输出举例。

```
#include <stdio.h>

int main()
{
    int a[5]={1,2,3}, b[5]={0}, i; // 同类型变量、数组同时声明

    // b = a;
    b[1] = a[1]; // 数组不能进行整体赋值
    printf("b[1]=%d a[1]=%d\n", b[1], a[1]); // 数组元素与同类型变量用法相同 // 输出两个数组元素的值

    for (i=0; i<5; i++) {
        b[i] = a[i] + 1; // 循环与数组经常配合使用
        printf("%d %d\n", a[i], b[i]); // 每组输出结果以换行结束
    }

    return 0;
}
```

### 程序解析：

1 对数组b初始化后，b[0]和其余元素的初值都为0；若不进行初始化，则b的元素初值不确定。

### 运行结果：

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



# 一维数组应用：实验数据处理

## ■ 例3.4-2：实验数据处理，求数据的均值ave、样本方差var、均方差SD。

$$\text{ave} = \frac{1}{n} \sum_{i=1}^n \text{data}_i, \quad \text{var} = \frac{1}{n-1} \sum_{i=1}^n (\text{data}_i - \text{ave})^2, \quad \text{SD} = \sqrt{\text{var}}$$

```
#include <stdio.h>
#include <math.h>          // 数学库函数头文件

#define N 8                // 符号常量便于修改

int main()
{
    float data[N] = {3.2, 2.3, 2.2, 2.4,
                      2.1, 3.0, 2.9, 2.8};
    float sum=0;           // 求和, 初值为0
    float ave, var, sd;   // 均值、方差、均方差
    int i;                 // 循环控制变量

    for (i=0; i<N; i++)
        sum = sum + data[i];
```

```
ave = sum / N;           // 计算均值
sum = 0;                  // 清零, 用于下次求和
for (i=0; i<N; i++)
    sum += (data[i]-ave) * (data[i]-ave);
var = sum / (N-1);        // 计算样本方差
sd = sqrt(var);          // 计算均方差

printf("均值%.3f, 方差%.3f, 均方差%.3f",
       ave, var, sd);

return 0;
```

运行结果：

均值2.612, 方差0.170, 均方差0.412



## ■ 一维数组

## ■ 多维数组

## ■ 字符数组

# 多维数组

## ■ 多维数组的定义与初始化

- 数组元素也是数组类型，即构成多维数组
- 每增加一个维度，定义时增加一对[]
- 初始化可以使用多维度和单维度两种方式

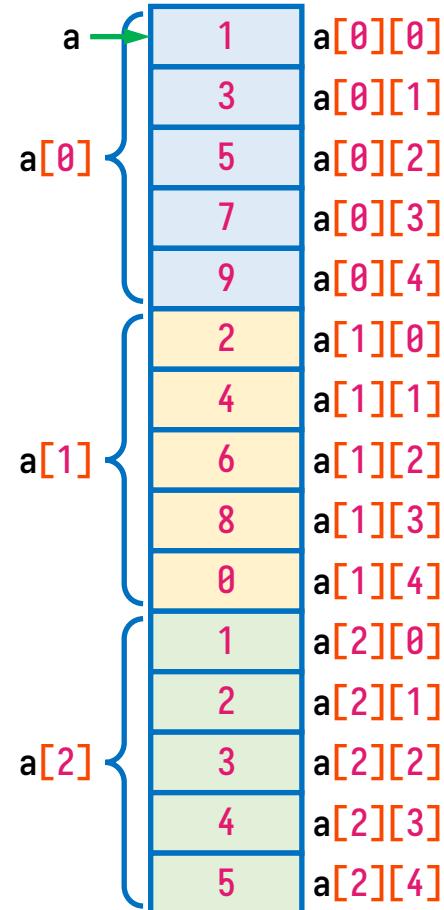
```

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

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 \\ 2 & 4 & 6 & 8 & 0 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}, B = \begin{bmatrix} 1 & 3 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \end{bmatrix}, C = \begin{bmatrix} 1 & 3 & 5 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, D = \begin{bmatrix} 1 & 3 & 5 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

## ■ 多维数组的存储

- 多维数组元素仍按照一维线性方式顺序存储
- 按照数据的逻辑结构划分为多个维度以便于处理





# 多维数组应用：矩阵乘法

## ■ 例3.4-5：矩阵乘法。

当矩阵 $A$ 的列数与矩阵 $B$ 的行数相等时，二者可以相乘，乘积矩阵 $C$ 的行数与 $A$ 相同，列数与 $B$ 相同。

矩阵 $A=(a_{ik})_{m \times p}$ 和 $B=(b_{kj})_{p \times n}$ 的乘积 $C=(c_{ij})_{m \times n}$ 的计算规则为：

$$c_{ij} = \sum_{k=0}^{p-1} a_{ik} b_{kj}$$

例如：( $m=2$ ,  $p=3$ ,  $n=2$ )

$$\begin{bmatrix} 5 & 8 & 3 \\ 11 & 0 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 18 \\ 2 & 11 \\ 10 & 3 \end{bmatrix} = \begin{bmatrix} 51 & 187 \\ 61 & 213 \end{bmatrix}$$

运行结果：

51	187
61	213

```
#include <stdio.h>

int main()
{
    int a[2][3]={{5,8,3},{11,0,5}};
    int b[3][2]={{1,18},{2,11},{10,3}};
    int c[2][2]={0}, i, j, k;

    for (i=0; i<2; i++)
        for (j=0; j<2; j++)
            for (k=0; k<3; k++)
                c[i][j] += a[i][k] * b[k][j];

    for (i=0; i<2; i++) {
        for (j=0; j<2; j++)
            printf("%6d\t", c[i][j]);
        printf("\n");
    }

    return 0;
}
```



- 一维数组
- 多维数组
- 字符数组



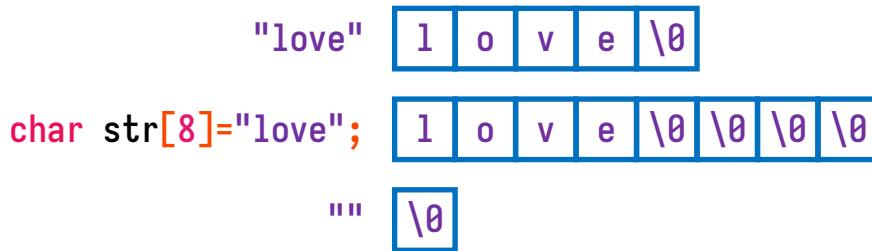
# 字符数组与字符串

## ■ 字符串 (String)

- 字符串的数据类型为字符数组
- 编译系统自动在字符串常量末尾添加结束标志字符 '\0'

  - 字符 '\0' 称为**空字符**, ASCII值为0, 即 '\0' == 0
  - 注意区分: 字符 '\0' 与字符 '0', '\0' 的ASCII值为0, '0' 的ASCII值为48

- 字符串常量为刚好能容纳其内容及结束标志字符 '\0' 的无名字符数组
- 注意区分: 字符串 "a" 与字符 'a', "a" 是字符数组, 包括两个字符元素 'a' 和 '\0'



## ■ 字符数组的初始化

```
char str[8]={ 'l', 'o', 'v', 'e' }; // 逐个字符进行初始化, 其余元素初值为0  
char str[8]={ "love" }; // 使用字符串常量初始化, 其余元素初值为0  
char str[8]= "love"; // 同上
```



# 字符数组与字符串举例

## ■ 例3.4-6：字符、字符数组与字符串的关系示例。

```
#include <stdio.h>

int main()
{
    char str1[8] = "";      // 初值: {'\0','\0',...}
    char str2[8] = "a";     // 初值: {'a','\0','\0',...}
    char str3[8] = {'a'};   // 初值: {'a','\0',...}

    printf("str1内容: %s, 长度: %d\n", str1, sizeof(str1)); // 数组str1字节数, 与字符串无关
    printf("str2内容: %s, 长度: %d\n", str2, sizeof(str2)); // 数组str2字节数, 与字符串无关
    printf("str3内容: %s, 长度: %d\n", str3, sizeof(str3)); // 数组str3字节数, 与字符串无关
    printf("\\"a\\"内容: %s, 长度: %d\n", "a", sizeof("a")); // 字符串常量字节数, 包括结束字符
    printf("\\'a\'\内容: %c, 长度: %d\n", 'a', sizeof('a')); // 字符常量为int类型

    return 0;
}
```

### 运行结果:

str1内容: , 长度: 8  
str2内容: a, 长度: 8  
str3内容: a, 长度: 8  
"a"内容: a, 长度: 2  
'a'内容: a, 长度: 4

### 程序解析:

1 %s是输入输出字符串的格式字符。



# 字符数组的输入

## ■ 逐个字符输入

```
char str1[8];
for (i=0; i<8; i++)
    scanf("%c", &str1[i]);
```

- 格式字符`%c`可以输入一个字符，包括空格、回车、制表符
- 若字符数组将用作字符串，则须在末尾至少保留一个元素，并赋值为`'\0'`

## ■ 字符串整体输入

```
char str2[8];
scanf("%s", str2);
```

- 格式字符`%s`可以输入一个字符串，但不能包含空格、回车、制表符
- 当遇到以上空白字符时，表示字符串输入结束
- 数组名表示其起始地址，作为`scanf()`的参数时不需要加`&`运算符
- 输入的字符串末尾自动添加结束标志字符`'\0'`并存入字符数组
- 字符数组应能容纳输入的字符串及结束标志字符`'\0'`，以免数组越界



# 字符数组应用：字符串比较

## ■ 例3.4-7：比较输入密码是否正确。

```
#include <stdio.h>

#define N 15

int main()
{
    char pwd[N] = "zhimakaimen";
    char str[N] = {0};
    int i, j;

    for (j=0; j<5; j++) {
        printf("Input password:\n");
        scanf("%s", str); // 输入密码字符串
        for (i=0; i<N && str[i]==pwd[i]; i++); // 逐个字符比较
        if (i == N) { // 若全部字符一致
            printf("Welcome to our world.");
            return 0;
        }
        else // 若密码错误
            printf("Warning! Incorrect!\n");
    }

    printf("Fail to enter.\n"); // 超过最大输入次数
    return 1;
}
```

### 程序解析：

- 1 输入字符数不得超过  $N-1$ ，以免数组str越界；
- 2 逐个比较str和pwd的元素，当数组结束或发现不相等元素时，循环结束；
- 3 根据i的值判断循环结束原因和数组比较结果。

### 运行结果一：

```
Input password:  
zhimakaimen  
Welcome to our world.
```

### 运行结果二：

```
Input password:  
putaokaimen  
Warning! Incorrect!  
Input password:  
zhimakaimen  
Welcome to our world.
```



# 结构体与复杂信息处理



## ■ 结构体的基本用法

## ■ 结构体数组

## ■ 结构体嵌套



# 结构体类型及变量定义

## ■ 结构体类型的定义

- 将有意义且相互关联的不同类型数据组织在一起

```
struct student {           // 学生结构体类型
    int      no;          // 学号
    char    name[20];     // 姓名
    float   gpa;         // 绩点
};                         // 以分号结束
```

## ■ 结构体变量的定义

```
struct student st1, st2; // 结构体类型名为 struct student
```

- 可以在定义结构体类型的同时定义结构体变量

```
struct student {           // 定义结构体类型
    int      no;
    char    name[20];
    float   gpa;
} st1, st2;                // 同时定义结构体变量
```



# 结构体变量初始化及运算

## ■ 结构体变量的初始化

- 以{}括起来并以,隔开的初值，依次赋给结构体变量中的成员
- 若结构体变量初始化时只给出部分初值，则未指定初值的成员初值为零

```
struct student {  
    int          no;  
    char         name[20];  
    float        gpa;  
} st1 = {220099, "Zhang San", 4.3};  
struct student st2 = {221101, "Li Si", 3.2};  
struct student st3 = {220055};
```

	no	name	gpa
st1	220099	"Zhang San"	4.3
st2	221101	"Li Si"	3.2
st3	220055	""	0.0

## ■ 结构体变量的运算

- 赋值运算、取地址运算、访问成员运算

```
st3 = st2;           // 同类型结构体变量之间可以整体赋值  
st3.no = st2.no;    // 访问结构体成员，结构体成员的用法与同类型变量相同  
scanf("%f", &st3.gpa); // 取结构体成员的地址，运算符.的优先级高于&  
scanf("%s", st3.name); // 数组名即代表地址，不需要使用取地址运算符
```



# 结构体应用：求斜率

## ■ 例3.5-1：求两点所在直线的斜率。

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

int main()
{
    struct point {           // 定义结构体类型
        float x;             // X轴坐标成员
        float y;             // Y轴坐标成员
    };
    struct point p1={1.1, 2.5}; // 定义结构体变量并初始化
    struct point p2={2.2};     // 初始化只提供部分初值时，未提供初值的成员初值为零

    printf("1.直线斜率为%f\n", (p2.y-p1.y)/(p2.x-p1.x)); // 结构体成员用法与同类型变量相同

    printf("2.请输入点p2的y坐标: ");
    scanf("%f", &p2.y);          // 取结构体成员的地址， 运算符.的优先级高于&
    printf(" 直线斜率为%f\n", (p2.y-p1.y)/(p2.x-p1.x));

    p2 = p1;                   // 同类型结构体变量互相赋值
    if ((fabs(p2.x-p1.x)) < 1e-6) // 判断两个浮点数是否相等
        printf("3.两点X轴坐标相同\n");

    return 0;
}
```

### 运行结果：

- 1.直线斜率为-2.272727
- 2.请输入点p2的y坐标: 2.0
  - 直线斜率为-0.454545
- 3.两点X轴坐标相同



- 结构体的基本用法
- 结构体数组
- 结构体嵌套



# 结构体数组

## ■ 结构体数组的定义及初始化

```
struct student st1[5]; // 结构体数组定义  
struct student st2[3] = {{220099, "Zhang San", 4.3}, // 结构体数组定义及初始化  
                         {221101, "Li Si", 3.2}, // 数组逐个元素、逐个成员  
                         {220055, "Wang Wu", 3.8}}; // 依次进行初始化
```

## ■ 结构体数组可以用于描述二维表格

数组下标	学生 (struct student)		
	学号 (int no)	姓名 (char name[20])	绩点 (float gpa)
0	220099	"Zhang San"	4.3
1	221101	"Li Si"	3.2
2	220055	"Wang Wu"	3.8

## ■ 结构体数组元素及其成员的引用

```
scanf("%d%s%f", &st1[0].no, st1[0].name, &st1[0].gpa); // 使用[]访问数组元素  
printf("%6d%s%.2f", st2[i].no, st2[i].name, st2[i].gpa); // 使用.访问结构体成员
```



# 结构体数组应用：三点共线

## ■ 例3.5-2：判断三个点是否在一条直线上。

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

int main() {
    struct point {
        float x;
        float y;
    };
    struct point pnt[3]={{1.1,2.2},{3.3,4.4},{5.5,6.6}}; // 存放三个点的坐标并初始化
    float slp[2]; // 存放两条直线的斜率

    slp[0] = (pnt[1].y-pnt[0].y) / (pnt[1].x-pnt[0].x); // 计算点1,2所在直线的斜率
    slp[1] = (pnt[2].y-pnt[0].y) / (pnt[2].x-pnt[0].x); // 计算点1,3所在直线的斜率

    if (fabs(slp[0]-slp[1]) < 1e-6) // 若两个斜率相等，则三点共线
        printf("三个点在一条直线上");

    return 0;
}
```

### 程序解析：

1 下标运算符[]和成员运算符.都具有最高优先级，按照运算次序，依次得到数组元素和结构体成员。

### 运行结果：

三个点在一条直线上

思考拓展：  
修改程序，由键盘输入三点的坐标，并考虑横坐标相等、多点重合等特殊情况。



- 结构体的基本用法
- 结构体数组
- 结构体嵌套

# 结构体嵌套



## ■ 结构体嵌套

### ■ 结构体类型中包含其他结构体类型的成员

```
struct date { // 日期结构体类型
    short year, month, day; // 年、月、日
};

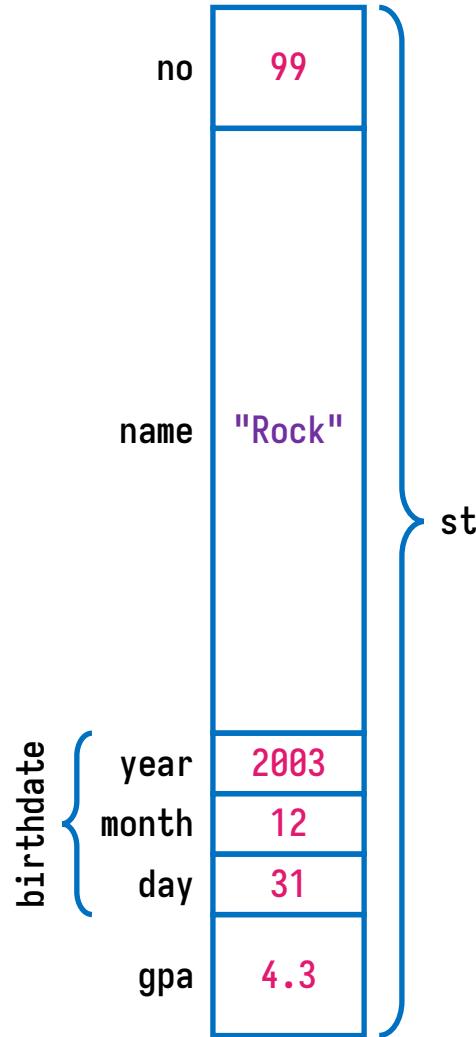
struct student { // 学生结构体类型
    int no; // 学号
    char name[20]; // 姓名
    struct date birthdate; // 出生日期，结构体类型
    float gpa; // 绩点
};
```

## ■ 嵌套结构体的初始化

```
struct student st = {99, "Rock", 2003, 12, 31, 4.3};
struct student st = {99, "Rock", {2003, 12, 31}, 4.3};
```

## ■ 嵌套结构体的成员引用

```
printf("%d", st.birthdate.day);
```





# 小结



# 主要知识点与能力要求

## ■ 主要知识点

- 基本数据类型：整型、浮点型、字符型
- 运算符的特性：操作数、优先级、结合性
- 表达式求值规则：算术、赋值、关系、逻辑、条件、逗号表达式
- 控制语句的语法和使用：分支语句、循环语句、中断和跳转语句
- 数组与批量数据处理：一维数组、多维数组、字符数组与字符串
- 结构体与复杂信息处理：结构体类型、结构体数组

## ■ 能力要求

- 根据问题需求，定义合适的变量、数组或结构体
- 分析问题并设计处理问题的算法逻辑，绘制流程图或书写伪代码
- 使用顺序、分支、循环结构编程实现算法逻辑
- 测试并完善后得到能够解决问题的程序



# 本章结束