

# 附录B 计算机基础知识

## 2. 计算机的信息表示

### 1) 二进制 —— 由0与1组成的数，逢2进位

十进制  $2105 = 2 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 5 \times 10^0$

$57.625 = 5 \times 10^1 + 7 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$

基数为10

基数为2

二进制  $(10110110)_2 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$(111001.101)_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

八进制  $(177)_8 = 1 \times 8^2 + 7 \times 8^1 + 7 \times 8^0$

$(1253.6)_8 = 1 \times 8^3 + 2 \times 8^2 + 5 \times 8^1 + 3 \times 8^0 + 6 \times 8^{-1}$

十六进制 (由数字0123456789ABCDEF组成的数，逢16进位)

$(7F)_{16} = 7 \times 16^1 + F \times 16^0 = 7 \times 16 + 15$

$(2AB.C)_{16} = 2 \times 16^2 + A \times 16^1 + B \times 16^0 + C \times 16^{-1}$   
 $= 2 \times 16^2 + 10 \times 16^1 + 11 \times 16^0 + 12 \times 16^{-1}$

## 2) 二进制运算

### 四则运算规则

运算分类	加法 +	乘法 ×
运算规则	$0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 1$ $1 + 1 = 10$	$0 \times 0 = 0$ $0 \times 1 = 0$ $1 \times 0 = 0$ $1 \times 1 = 1$

### 3) 进制的转换

二进制 → 十进制

$$\begin{aligned} & (111001.101)_2 \\ &= 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 32 + 16 + 8 + 1 + 0.5 + 0.125 = 57.625 \end{aligned}$$


十六进制 → 十进制

$$\begin{aligned} & (2AB.C)_{16} \\ &= 2 \times 16^2 + 10 \times 16^1 + 11 \times 16^0 + 12 \times 16^{-1} \\ &= 512 + 160 + 11 + 0.75 = 683.75 \end{aligned}$$


十进制 → 二进制

$$57 = (111001)_2$$

2		57	1
2		28	0
2		14	0
2		7	1
2		3	1
2		1	1
		0	



$$0.625 = (0.101)_2$$



	0.625	× 2
1	.25	× 2
0	.5	× 2
1	.0	

$$\begin{aligned} 57.625 &= 57 + 0.625 \\ &= (111001)_2 + (0.101)_2 = (111001.101)_2 \end{aligned}$$

十进制  $\rightarrow$  八进制

$$683 = (1253)_8$$

$$\begin{array}{r} 8 \overline{) 683} \\ 8 \overline{) 85} \\ 8 \overline{) 10} \\ 8 \overline{) 1} \\ 0 \end{array} \quad \begin{array}{l} 3 \\ 5 \\ 2 \\ 1 \end{array}$$

$$0.75 = (0.6)_8$$

$$\begin{array}{r} 0.75 \times 8 \\ \hline 6.0 \end{array}$$

$$683.75 = 683 + 0.75$$

$$= (1253)_8 + (0.6)_8 = (1253.6)_8$$



十进制  $\rightarrow$  十六进制

$$683 = (2AB)_{16}$$

$$0.75 = (0.C)_{16}$$

$$\begin{array}{r} 16 \overline{) 683} \\ \underline{16 \phantom{00} 42} \\ 16 \phantom{00} \underline{2} \\ 0 \end{array} \quad \begin{array}{c} 11 \\ 10 \\ 2 \end{array} \quad \begin{array}{c} \uparrow \end{array}$$

$$\begin{array}{r} 0.75 \times 16 \\ \hline 12 \\ \hline .0 \end{array} \quad \begin{array}{c} \downarrow \end{array}$$

$$683.75 = 683 + 0.75$$

$$= (2AB)_{16} + (0.C)_{16} = (2AB.C)_{16}$$

十六进制  $\longleftrightarrow$  二进制 (一位十六进制数相当于4位二进制数)

$$(6CF)_{16} = (0110\ 1100\ 1111)_2 = (11011001111)_2$$

$$(2AB.C)_{16} = (0010\ 1010\ 1011.1100)_2 = (1010101011.11)_2$$

$$(1101001)_2 = (0110\ 1001)_2 = (69)_{16} \quad \text{高位补足4位}$$

$$(111001.101)_2 = (0011\ 1001.1010)_2 = (39.A)_{16} \quad \text{补足4位}$$

十六进制与二进制对应关系表

16进制	二进制	16进制	二进制
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

## 4) 计算机数值表示

### 整数表示:

计算机的存储基本单位是字节，由8位二进位组成。一个整数值可以用单字节保存，也可以用2字节或4字节保存。

无符号整数: 单字节扩展为双字节: 高位字节全为0

真值	单字节	双字节
5	[ 0000 0101 ]	[0000 0000] [0000 0101]
0	[ 0000 0000 ]	[0000 0000] [0000 0000]
119	[ 0111 0111 ]	[0000 0000] [0111 0111]
255	[ 1111 1111 ]	[0000 0000] [1111 1111]
1743	无法表示	[0000 0110] [1100 1111]



有符号整数用补码表示：**正数同无符号整数**(最高位0)，  
**负数用对应正数的补数表示**(最高位1)

单字节扩展为双字节：

单字节最高位(符号位)数字作为高位字节各位数字

真值	单字节补码	双字节补码
0	[ <b>0</b> 000 0000 ]	[ <b>0</b> 000 0000] [0000 0000]
5	[ <b>0</b> 000 0101 ]	[ <b>0</b> 000 0000] [0000 0101]
-5	[ <b>1</b> 111 1011 ]	[ <b>1</b> 111 1111] [1111 1011]
6	[ <b>0</b> 000 0110 ]	[ <b>0</b> 000 0000] [0000 0110]
-6	[ <b>1</b> 111 1010 ]	[ <b>1</b> 111 1111] [1111 1010]
119	[ <b>0</b> 111 0111 ]	[ <b>0</b> 000 0000] [0111 0111]
-119	[ <b>1</b> 000 1001 ]	[ <b>1</b> 111 1111] [1000 1001]
最大正数	[ <b>0</b> 111 1111 ]	[ <b>0</b> 111 1111] [1111 1111]
最小负数	[ <b>1</b> 000 0000 ]	[ <b>1</b> 000 0000] [0000 0000]

求99的单字节补码:

(1) 转换为8位2进制数:

$$(99)_{10} = (01100011)_2$$

补码: [01100011]

求-75的单字节补码:

(1) 75转换为8位2进制数:

$$(75)_{10} = (01001011)_2$$

补码: [01001011]

(2) 75的补码求补(取反加1)

取反: [10110100]

加1: [10110101] -75的补码

99的单字节补码扩展为双字节补码:

单字节补码: [01100011]

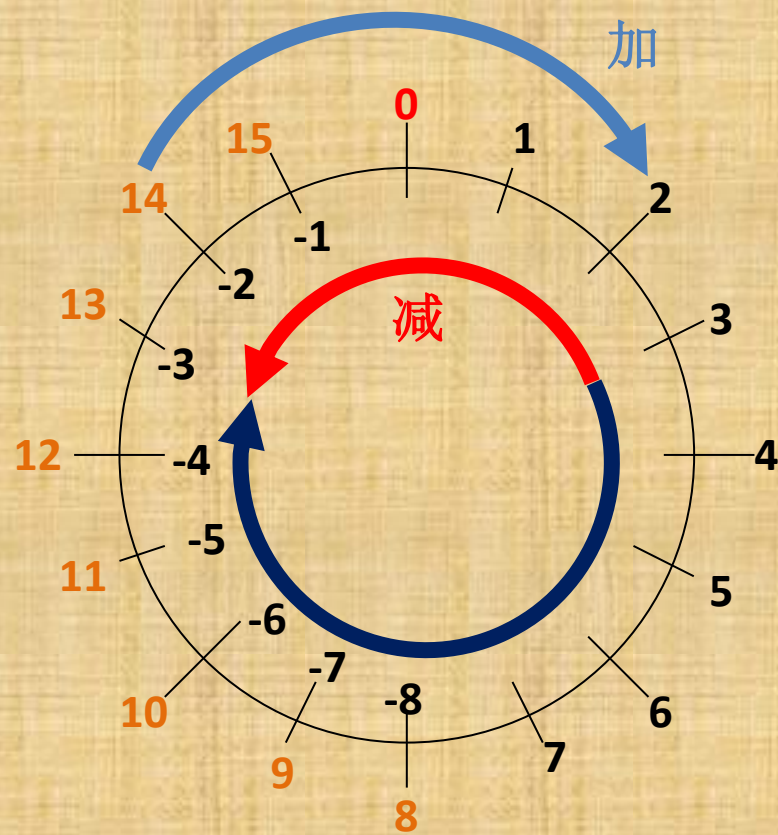
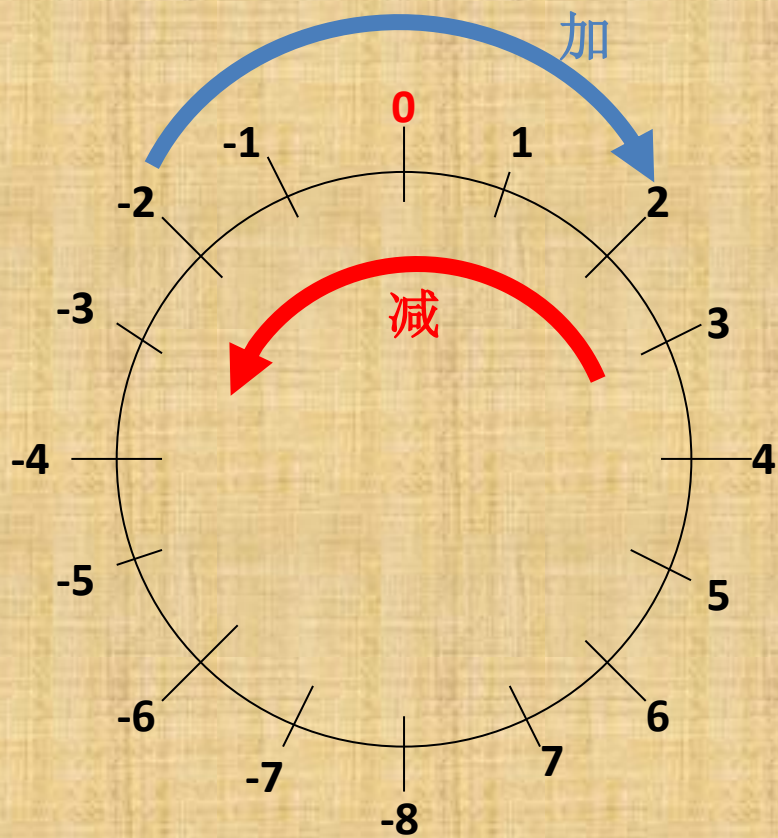
双字节补码: [00000000] [01100011]

-75的单字节补码扩展为双字节补码:

单字节补码: [10110101]

双字节补码: [11111111] [10110101]

## 补码原理:

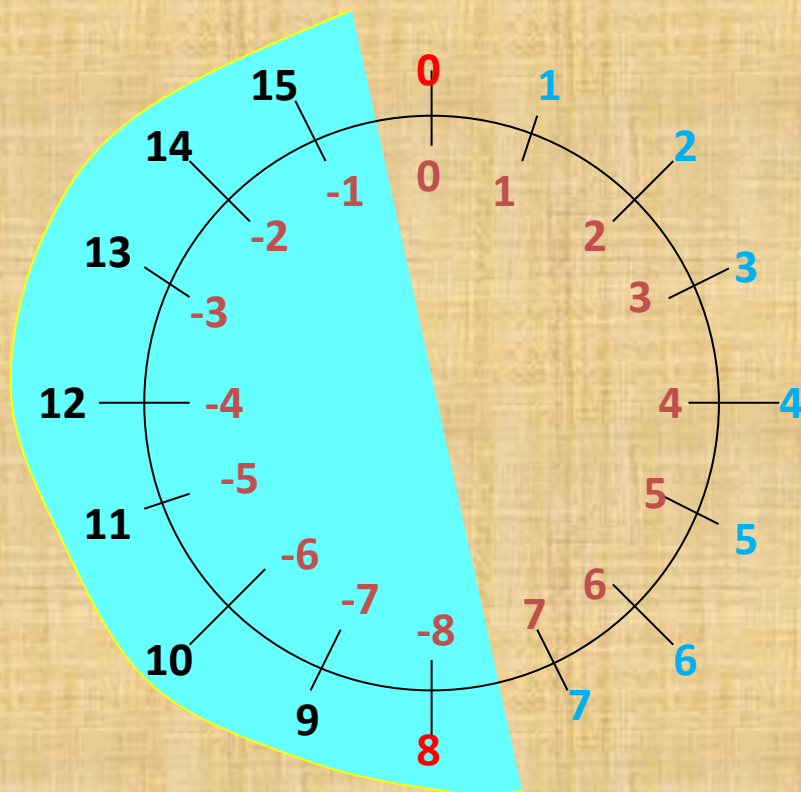


整数： 两个相反数相加，和为0。

补码： 两个相反数的补码相加，进位丢弃，剩下为0

以 4 位二进制数为例

真值	补码
0	[ 0000 ]=0
1	[ 0001 ]=1
-1	[ 1111 ]=15 =(-1) + 16
最大 7	[ 0111 ]=7
-7	[ 1001 ]= 9 =(-7) + 16
最小-8	[ 1000 ]= 8 =(-8) + 16





## 实数表示：（IEEE标准）

实数采用2进制标准化科学记数法表示

如+0.001先如右图化为二进制小数  
进一步写成科学计数法

	0.001	×2
0	.002	×2
0	.004	×2
0	.008	×2
0	.016	×2
0	.032	×2
0	.064	×2
0	.128	×2
0	.256	×2
0	.512	×2
1	.024	×2
0	.048	×2
0	.096	×2
0	.192	×2
0	.384	×2
0	.768	×2
1	.536	×2

.....

$$+0.001 = + 0.000000000 10000011 00010010 01101110 10010111 \\ 10001101 01001111 11011111 00111011 0110...$$

$$= +1.0000011 00010010 01101110 10010111 \\ 10001101 01001111 11011111 00111011 0110... \times 2^{-10}$$



$$\begin{aligned}
 -12000 &= -10111011100000.00 \quad 00000000 \quad 00000000 \\
 &\quad 00000000 \quad 00000000 \quad 00000000 \quad 00000000 \quad 0000... \\
 &= -1.0111011 \quad 10000000 \quad 00000000 \quad 00000000 \\
 &\quad 00000000 \quad 00000000 \quad 00000000 \quad 00000000 \quad 0000... \times 2^{13}
 \end{aligned}$$

## 实数存储格式

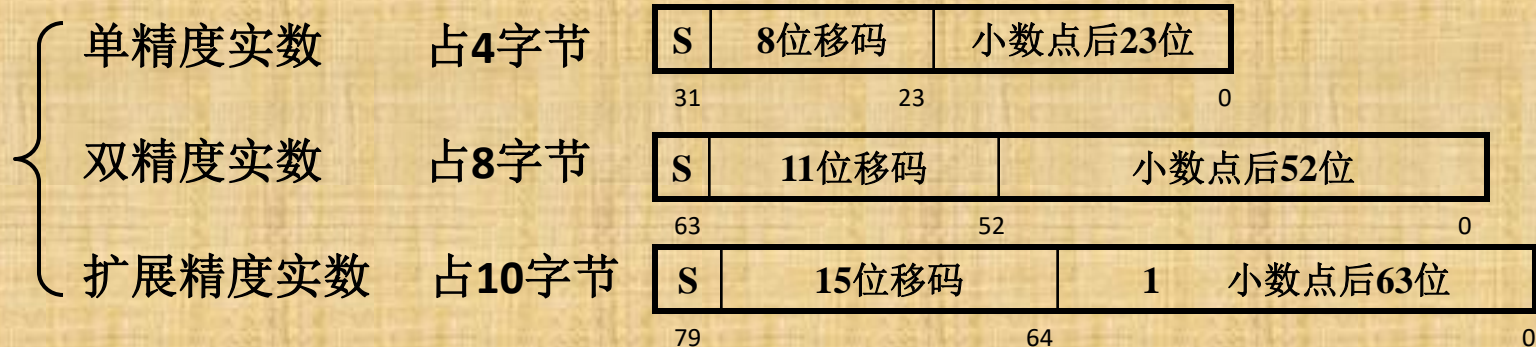
S (符号)	指数部分	尾数部分
--------	------	------

实数0的格式为各位为0，即  $(00000.....000)_2$

非零实数的格式：

- S的确定：正数S为0，负数S为1
- 指数部分的确定：指数加上数  $(0111 \dots 11)_2$ ，称为移码
- 尾数部分的确定：小数点后的小数，最后一位0舍1入进位。若扩展精度，则前面有1

# 实数分



$$+0.001 = +1.000011 \ 00010010 \ 01101110 \ 10010111 \ 10001101 \ 01001111 \\ 11011111 \ 00111011 \ 0110... \times 2^{-10}$$

$$(-10)_{10} = (-1010)_2$$

$$= \begin{array}{|c|c|c|} \hline 0 & 01110101 & 0000011 \ 00010010 \ 01101111 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|} \hline 0 & 011 \ 1111 \ 0101 & 0000 \ 01100010 \ 010 \ 01101 \ 11010010 \ 11110001 \ 10101001 \ 11111100 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|} \hline 0 & 0111111 \ 1111 \ 0101 & 10000011 \ 00010010 \ 01101110 \ 10010111 \ 10001101 \ 01001111 \\ & & 11011111 \ 00111011 \\ \hline \end{array}$$

$$-12000 = -1.0111011 \ 10000000 \ 00000000 \ 00000000 \ 00000000 \ 00000000 \\ 00000000 \ 00000000 \ 0000... \times 2^{13}$$

$$(13)_{10} = (1101)_2$$

$$= \begin{array}{|c|c|c|} \hline 1 & 10001100 & 0111011 \ 10000000 \ 00000000 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|} \hline 1 & 100 \ 0000 \ 1100 & 0111 \ 01110000 \ 00000000 \ 00000000 \ 00000000 \ 00000000 \ 00000000 \\ \hline \end{array}$$

各种值	移码	尾数
0	0	0
非标准数	0	非零
标准数	1~最大值-1	任意
$\pm\infty$	最大值	0
NaN	最大值	非零

## 5) 计算机非数值表示

在计算机中，用编号表示相应的文字。西文(指英语)通常用ASCII编码方案，用7位二进制数对128个字符进行编码。

高3位		000	001	010	011	100	101	110	111
低4位	0000	32个控制字符	空格	0	@	P	`	p	
	0001		!	1	A	Q	a	q	
	0010		"	2	B	R	b	r	
	0011		#	3	C	S	c	s	
	0100		\$	4	D	T	d	t	
	0101		%	5	E	U	e	u	
	0110		&	6	F	V	f	v	
	0111		'	7	G	W	g	w	
	1000		(	8	H	X	h	x	
	1001		)	9	I	Y	i	y	
	1010		*	:	J	Z	j	z	
	1011		+	;	K	[	k	{	
	1100		,	<	L	\	l		
	1101		-	=	M	]	m	}	
	1110		.	>	N	^	n	~	
	1111		/	?	O	_	o		DEL

94个可显字符



中文(指简体中文)有国标码和Unicode码。

国标码指国家汉字编码标准，有：

**GB2312-1980** (信息交换用汉字编码字符集基本集),双字节编码

**GBK-1995** (汉字内码扩展规范)，双字节编码

**GB18030-2000** (信息交换用汉字编码字符集基本集的扩充)

**GB18030-2005** 在GB18030-2000基础上增加了许多汉字及少数民族文字。**GB18030**由单字节、双字节、三字节、四字节混合编码

**Unicode编码**：国际上通用的包含全球文字的字符集编码，标准有：

**UTF-8** 由单字节、双字节、三字节、四字节混合编码

**UTF-16即UCS-2** 由双字节、四字节混合编码

**UTF-32即UCS-4** 四字节编码



# C++程序设计

## 第一篇 基础编程

第1章 C++概述

第2章 简单的C++程序

第3章 数据与运算

第4章 选择结构和循环结构

第5章 复合类型数据

# 第1章 C++概述

## 1.1 C++历史背景

ENIAC (1946) 美国宾夕法尼亚大学

——世界上第一台电子数字计算机

程序设计语言的发展：低级语言，高级语言

低级语言阶段：

- 机器语言
- 汇编语言

## 高级语言阶段:

- **FORTRAN语言(1956)** 通用高级程序设计语言——科学计算
- **ALGOL60语言(1960)**高级程序设计语言——离硬件远
- **CPL语言(1963)**英国剑桥大学——接近硬件，但规模大
- **BCPL语言(1966)**剑桥Martin Richards ——CPL简化版
- **B语言(1969)** 美国贝尔实验室Ken Thompson —BCPL简化版
- **C语言(1972-1973)** 贝尔实验室Ritchie, Kernighan —B增强版
- **C++语言(1979)** 贝尔实验室Bjarne Stroustrup —— C的扩充版
- **ISO/IEC 14882:1998 (C++98)** —— C++的第一个国际标准
- **ISO/IEC 14882:2011 (C++11)** —— 2011年C++的国际标准
- **ISO/IEC 14882:2017(C++17或C++1z)** —— 2017年国际标准

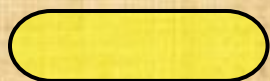
## 1.2 C++语言的特点

- 具有C语言的良好特性——语言简洁紧凑；类型运算丰富；  
有高级和低级语言特点；移植性好；  
代码效率高
- 具有面向对象的特性——由数据组织操作；数据受保护；  
代码可重用；接口抽象化
- 具有泛型程序设计的特性——类型可作为参数设计程序
- 具有开发大规模程序的相关特性——名空间名字管理；异常处理

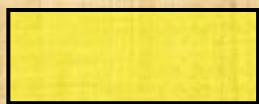


# 第2章 简单的C++程序

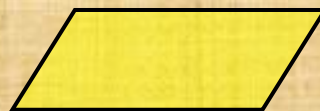
## 2.1 流程框图



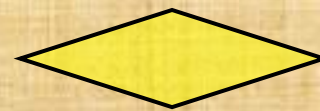
起止框



处理框



输入/输出框



判断框

例：输入一个数 $x$ ，

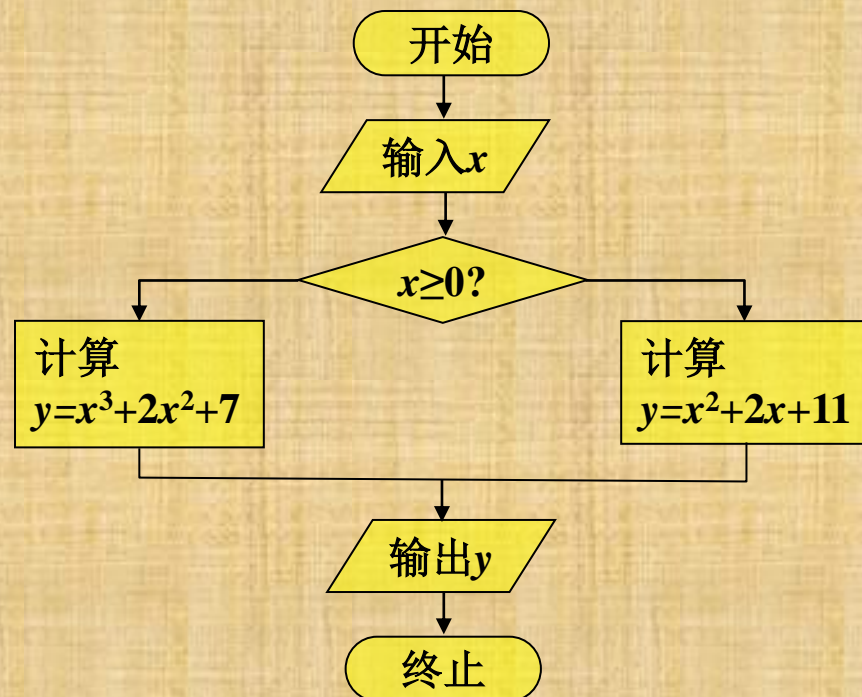
若 $x$ 非负，计算

$$y = x^3 + 2x^2 + 7$$

否则计算

$$y = x^2 + 2x + 11$$

输出结果 $y$





## 2.2 简单的顺序结构程序

C++程序框架:

```
# include <iostream>
using namespace std;
int main( )
{
    .....
    return 0;
}
```

程序要用到  
输入/输出

使用标准的  
输入/输出方式

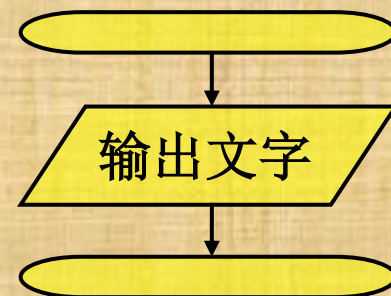
程序主要模块

程序结束  
并返回0给操作系统

## 例A2.1 编写最简单的C++程序，输出文字 “This is the first C++ program.”。

```
#include <iostream>
using namespace std;
int main( )
{
    cout<<"This is the first C++ program."<<endl;
    return 0;
}
```

流程框图



注1: “cout<<”This is the first C++ program.”<<endl;” 最后endl中的l是小写字母l，不是数字1。

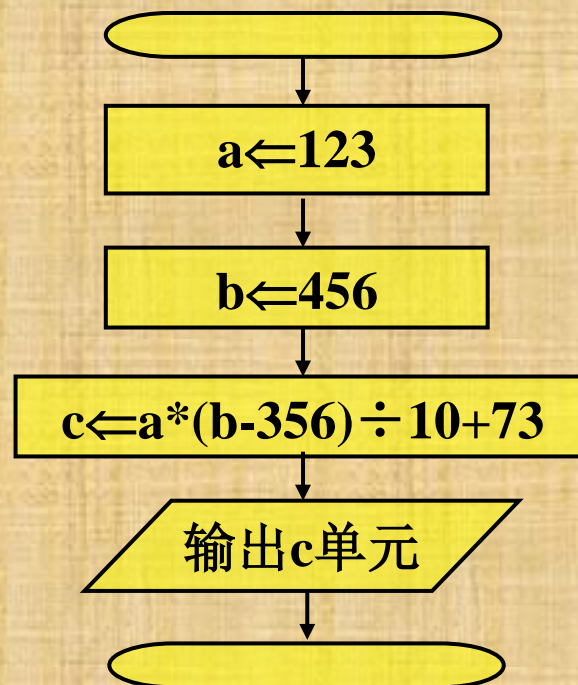
注2: “return 0;”中使用数字0。

## 例A2.2 计算算术表达式 “ $123 \times (456 - 356) + 73$ .”。

```
#include <iostream>
using namespace std;
int main( )
{
    cout<<123*(456-356)+73<<endl;
    return 0;
}
```

例A2.3 计算“ $c=a(b-356) \div 10+73$ ”的整数值，其中 $a=123$ 、 $b=456$  (使用存储单元)。

```
#include <iostream>
using namespace std;
int main( )
{
    int a , b , c ;
    a=123;
    b=456;
    c=a*(b-356)/10+73;
    cout<<c<<endl;
    return 0;
}
```





- **主函数**：主函数指 `int main() { ... }` 的所有内容，其中花括号及花括号中的内容称为主函数体，`int main()` 称为主函数首部。
- **语句**：表示程序中的一个操作执行的单位，例如程序中的“`cout<<c<<endl;`”和“`return 0;`”等，语句末尾总是要使用分号表示一个语句的结束。
- **变量**：变量是程序中用于存储数据的存储单元，通常有一个相应名字，称变量名。
- **预处理命令**：以#开头的一行内容，例如“`# include <iostream>`”
- **大小写敏感**：指书写程序时大写字母和小写字母表示不同的字母。例如 `main` 与 `Main` 表示不同的名字。  
C++ 是大小写敏感的。



**\* C++程序的书写形式：分号 “;”决定语句，便于阅读**

注3：变量要先定义，后使用。

注4：C++语句一定要用分号 “;”结束。

注5：以 “#”开头的预处理命令必须每一个命令各自占单独的一行，后面不跟分号 “;”。

**例A2.4 计算1.5的平方，并输出计算结果(使用实数)。**

```
#include <iostream>
using namespace std;
int main( )
{
    double x , y ;
    x=1.5; y=x*x;
    cout<<x<< "\t"<<y<<endl;
    return 0;
}
```


例B2.5 比较整数除和实数除。

```
# include <iostream>
using namespace std;
int main( )
{
    int a , b , c ;
    double x , y , z ;
    a=7; b=4; c=a/b;
    x=7.0; y=4.0; z=x/y;
    cout<<c<< "\\t"<<z<<endl;
    return 0;
}
```

注6: C++中的整数除以整数结果为整数，是去掉小数部分后的整数，不是下取整。

例A2.6 计算101除以5、7、11后的余数，并输出结果(使用求余数运算%)。

```
#include <iostream>
using namespace std;
int main( )
{
    int a=101, b5, b7, b11 ;
    b5=a%5; b7=a%7; b11=a%11;
    cout<< "a%5="<<b5<<endl;
    cout<< "a%7="<<b7<<endl;
    cout<< "a%11="<<b11<<endl;
    return 0;
}
```



例A2.7 输入数据 $r$ ，用公式 $S=\pi r^2$  计算圆面积并输出结果(使用输入cin)。

```
#include <iostream>
using namespace std;
int main( )
{
    double pi=3.141592653589793 , r , S ;
    cout<< "输入半径: ";
    cin>>r;
    S=pi*r*r;
    cout<<"半径="<<r<< "\t"<<"面积="<<S<<endl;
    return 0;
}
```

注7：在本书的运行结果中，阴影部分表示要用户输入的数据。



例A2.8 输入两个点的坐标 $(x_1, y_1)$ 和 $(x_2, y_2)$ ，计算两点间的距离并输出(使用数学函数)。

```
#include <iostream>
#include <cmath>
using namespace std;
int main( )
{   double x1,y1,x2,y2,dist;
    cout<< "输入两个点的x,y坐标:x1 y1 x2 y2"<<endl;
    cin>>x1>>y1>>x2>>y2;
    dist=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
    cout<<"("<<x1<<","<<y1<<")和("<<x2<<","<<y2<<") 间的距离为"<<dist<<endl;
    return 0;
}
```

注8: 在用键盘输入多个数据时规定数据之间只能用一个或多个空白间隔符(空格、Tab、回车)进行间隔，如果用逗号间隔输入数据会出错。

## 表2- 1常用的数学库函数

<b>sqrt(x)</b> 实数开平方	<b>log(x)</b> 实数自然对数 $\ln x$	<b>tan(x)</b> 实数正切
<b>fabs(x)</b> 实数绝对值	<b>pow(x,y)</b> 实数幂指函数 $x^y$	<b>asin(x)</b> 实数反正弦
<b>abs(x)</b> 整数绝对值	<b>sin(x)</b> 实数正弦	<b>acos(x)</b> 实数反余弦
<b>exp(x)</b> 实数指数函数 $e^x$	<b>cos(x)</b> 实数余弦	<b>atan(x)</b> 实数反正切

注9：输入数据给变量x1、y1、x2、y2要用“`cin>>x1>>y1>>x2>>y2;`”，若用“`cin>>x1,y1,x2,y2;`”将出错，实际只输入数据给>>操作后面的一项x1，而y1、x2、y2得不到数据。

注10：如果要使用上述数学库函数，需要在源程序前面加上“`#include <cmath>`”，可以加在“`#include <iostream>`”的下一行，也可以加在上一行。

注11：三角、反三角库函数的角度使用弧度制。

注12：函数abs(x)可以用于实数，此时与fabs(x)一致。函数pow(x,y)中x必须是实数，y可以是实数，也可以是整数。

**例A2.9** 输入三角形的3个边长，计算并输出三角形的面积  
(使用注解)。

比较下面的两个程序

程序1

```
# include <iostream>
# include <cmath>
using namespace std;
int main( )
{   double a,b,c,s,area;
    cout<< "输入三角形的3个边长: ";
    cin>>a>>b>>c;
    s=(a+b+c)/2;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    cout<<"area="<<area<<endl;
    return 0;
}
```



上例程序加上注解后如下：程序2

```
/******
```

```
* MA2_9.cpp
```

```
* 输入三角形的3个边长，计算并输出三角形的面积(使用注解)
```

```
*****/
```

```
# include <iostream>
```

```
# include <cmath>
```

```
using namespace std;
```

```
int main( )
```

```
{ double a,b,c,s,area;
```

```
cout<< "输入三角形的3个边长: ";
```

```
cin>>a>>b>>c; /*输入数据*/
```

```
s=(a+b+c)/2;
```

```
area=sqrt(s*(s-a)*(s-b)*(s-c)); //海伦公式计算面积,s半周长
```

```
cout<<"area="<<area<<endl;
```

```
return 0;
```

```
}
```

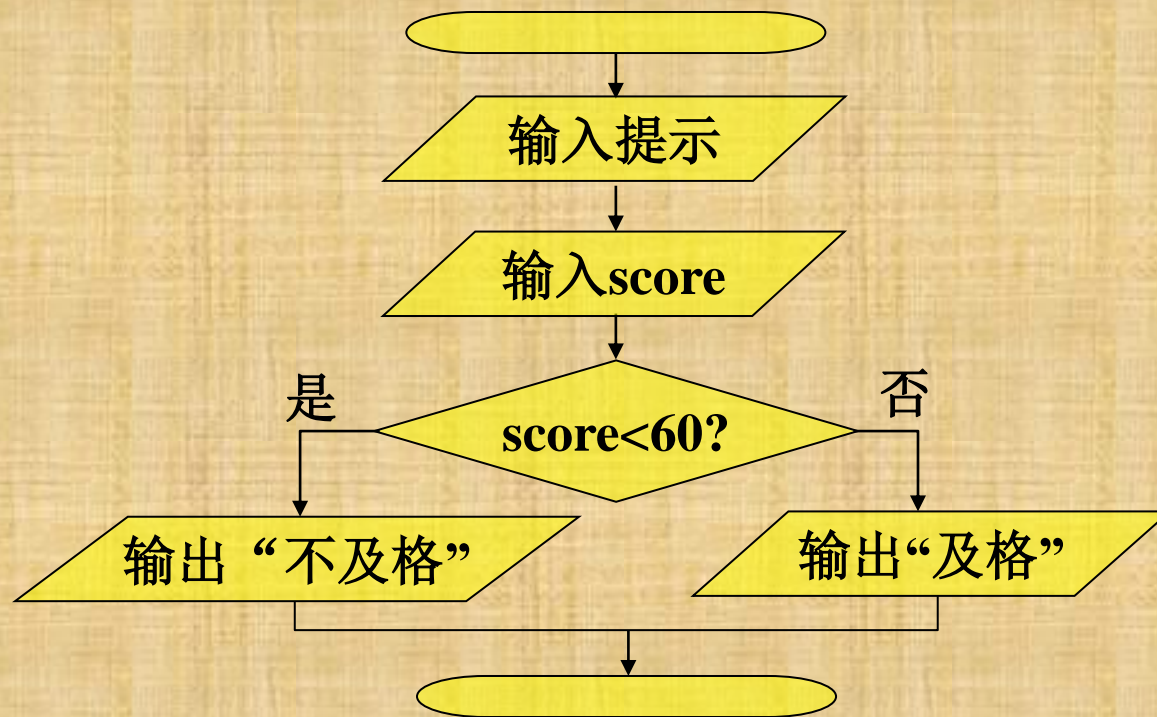
注13：若语句“ $s=(a+b+c)/2;$ ”改写成“ $s=1/2*(a+b+c);$ ”，计算结果将出错。因为在计算1/2时是整数除，结果为0，最后s得到的值是0。



## 2.3 有分支和循环结构的程序

### 2.3.1 有分支结构的程序

例A2.10 输入一个考试分数(百分制)，判断是否及格(使用if-else)。



```
/******
```

```
* MA2_10.cpp
```

```
* 输入一个考试分数(百分制), 输出及格或不及格(使用if-else)
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{ int score;
```

```
  cout<< "输入一个考试分数(百分制): ";
```

```
  cin>>score;
```

```
  if(score<60)
```

```
    //判断score<60是否成立
```

```
  { cout<< "不及格" <<endl; }    //条件成立要执行的语句
```

```
  else
```

```
  { cout<< "及格" <<endl; }    //条件不成立要执行的语句
```

```
  return 0;
```

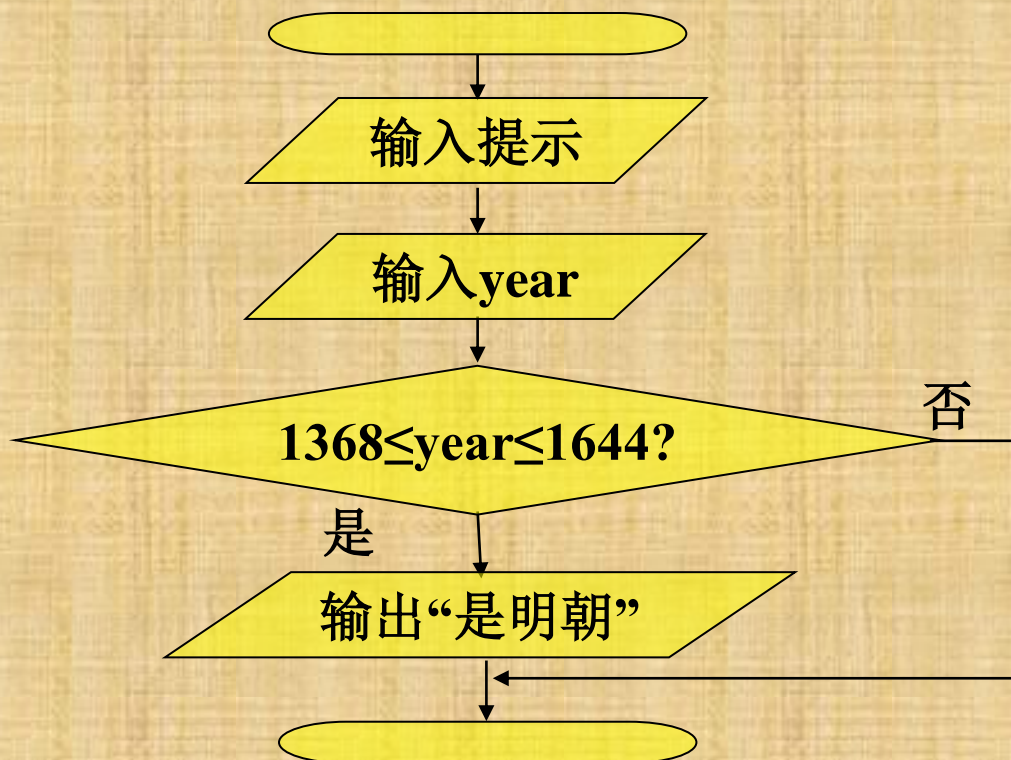
```
}
```

比较运算有: < 小于      <=小于等于 > 大于      >=大于等于  
              == 等于      != 不等于

注1: 比较相等用 “==” 而不是 “=”。

注2: 在双字符的比较中不能出现空格如 “> =”。

**例A2.11** 输入一个年份，判断年份是否在明朝  
(明朝为1368-1644，使用复合条件)。



```

/*****
*  MA2_11.cpp
*  输入一个年份，判断是否在明朝期间(使用复合条件)
*****/
#include <iostream>
using namespace std;
int main( )
{
    int year;
    cout<< "输入一个年份: ";
    cin>>year;
    if(1368<=year && year<=1644) //复合条件, 无else
    { cout<<year<< "是明朝" <<endl; }
    return 0;
}

```



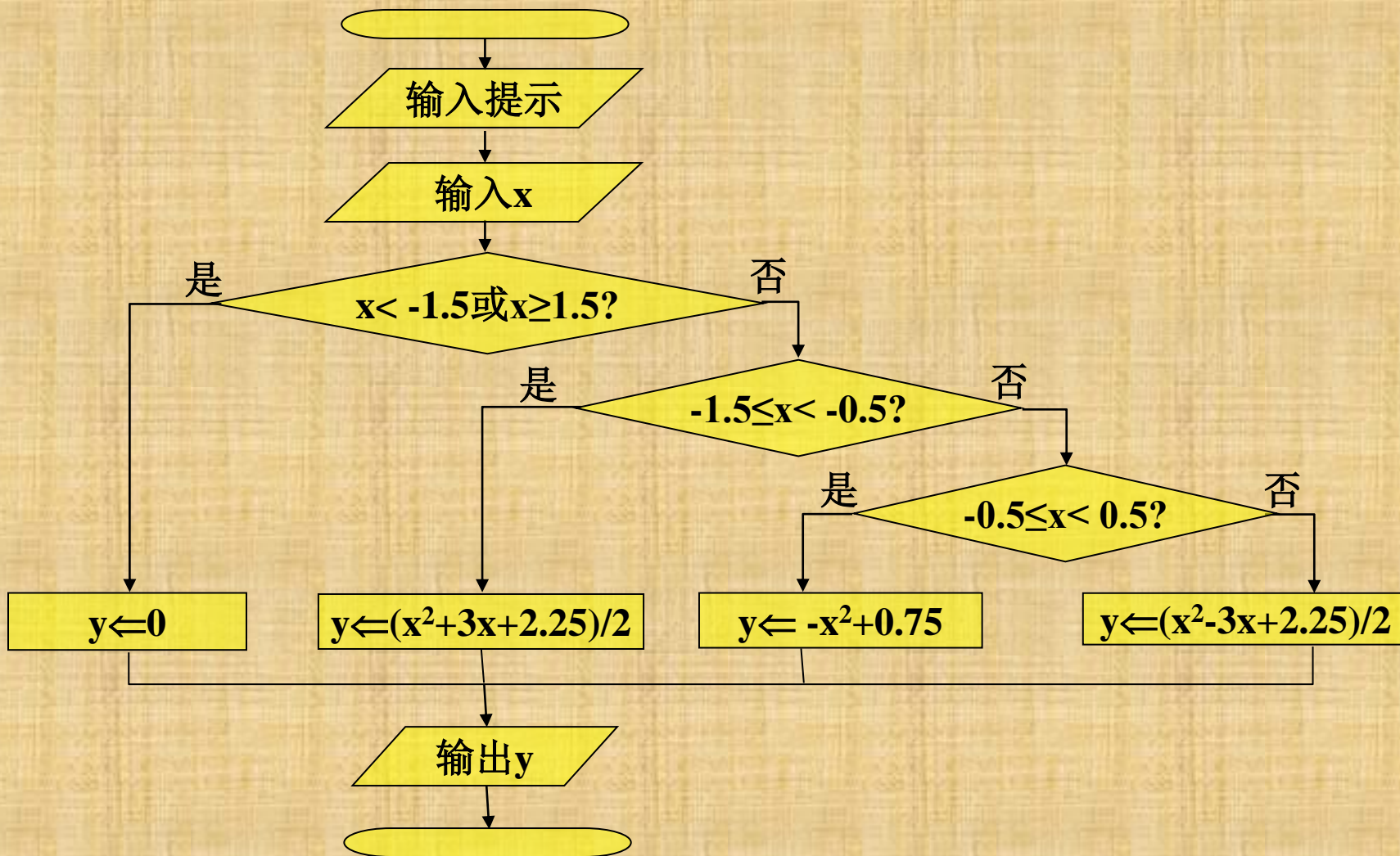
## 用逻辑运算&&, ||, ! 可以构成复杂的条件

表2- 2 C++逻辑运算

运算	C++形式	运算规则	说明	含义
与	&&	真&&真得真 真&&假得假 假&&真得假 假&&假得假	两个值均为真时，结果才为真； 两个值只要有一个值为假，结果就为假	条件1&&条件2 表示条件1和条件2都成立，整个条件才成立。
或		真  真得真 真  假得真 假  真得真 假  假得假	两个值只要有一个为真，结果就为真； 两个值均为假时结果才为假	条件1    条件2 表示条件1或者条件2成立，整个条件就成立。
非	!	!真得假 !假得真	真变为假，假变为真	! 条件1 表示条件1成立，整个条件不成立；若条件1不成立，则整个条件成立

例A2.12 输入一个实数值x，计算函数 $\Omega(x)$ 的值(使用else if结构)。

$$y=\Omega(x)=\begin{cases} 0, & x \in (-\infty, -1.5) \cup [1.5, +\infty) \\ \frac{1}{2}x^2 + \frac{3}{2}x + \frac{9}{8}, & x \in [-1.5, -0.5) \\ -x^2 + \frac{3}{4}, & x \in [-0.5, 0.5) \\ \frac{1}{2}x^2 - \frac{3}{2}x + \frac{9}{8}, & x \in [0.5, 1.5) \end{cases}$$



```
/**/
```

```
* MA2_12.cpp
```

```
* 输入一个实数x，计算和输出函数 $\Omega(x)$ 的值(使用else if结构)
```

```
****/
```

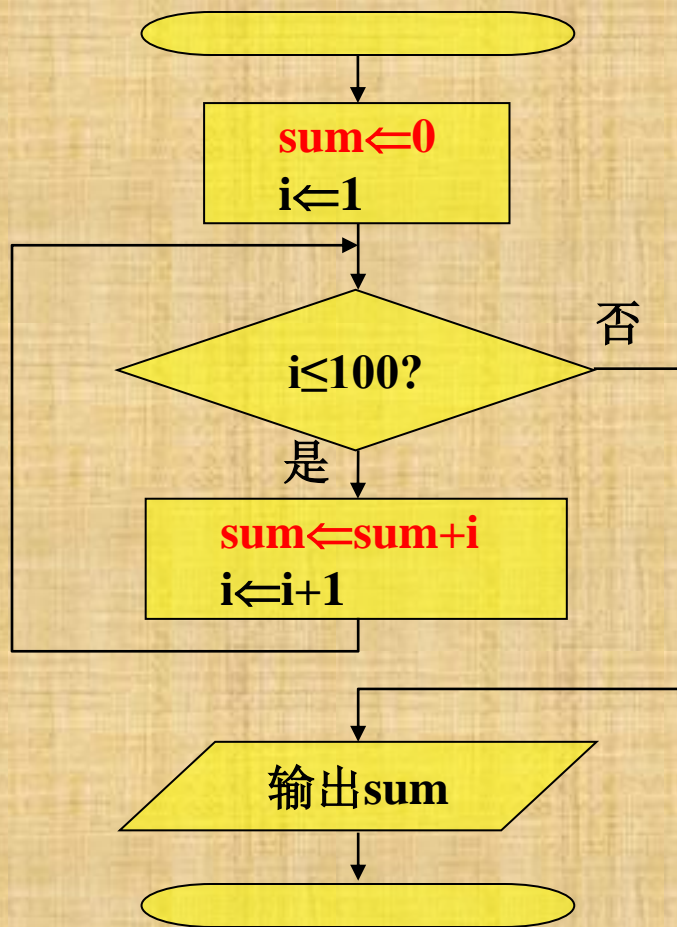
```
#include <iostream>
using namespace std;
int main( )
{
    double x, y;
    cout<< "输入一个实数: ";
    cin>>x;
    if(x<-1.5 || x>=1.5)          { y=0; }           //条件1, 分支1
    else if(-1.5<=x&& x<-0.5) { y=(x*x+3*x+2.25)/2; } //分支2
    else if(-0.5<=x&& x<0.5) { y= -x*x+0.75; }       //分支3
    else                          { y=(x*x-3*x+2.25)/2; } //分支4
    cout<< "所求函数值= " <<y<<endl;
    return 0;
}
```

注3: else if中间一定要有空格，不能连写成elseif。



## 2.3.2 有循环结构的程序

例A2.13 计算  $1+2+\cdots+100$  并输出(使用循环)。



```

/*****
*  MA2_13A.cpp
*  输出 1+2+ ... +100 的值(使用while循环)
*****/
#include <iostream>
using namespace std;
int main( )
{   int i , sum;
    sum=0 ;      //sum清零
    i=1 ;
    while( i<=100 ) //当 i<=100 时反复执行下面的循环体
    {   sum=sum+i ;    //将i的值加到sum中
        i=i+1;        //i的值增加1
    }
    cout<< " 1+2+ ... +100= " <<sum<<endl;
    return 0;
}

```

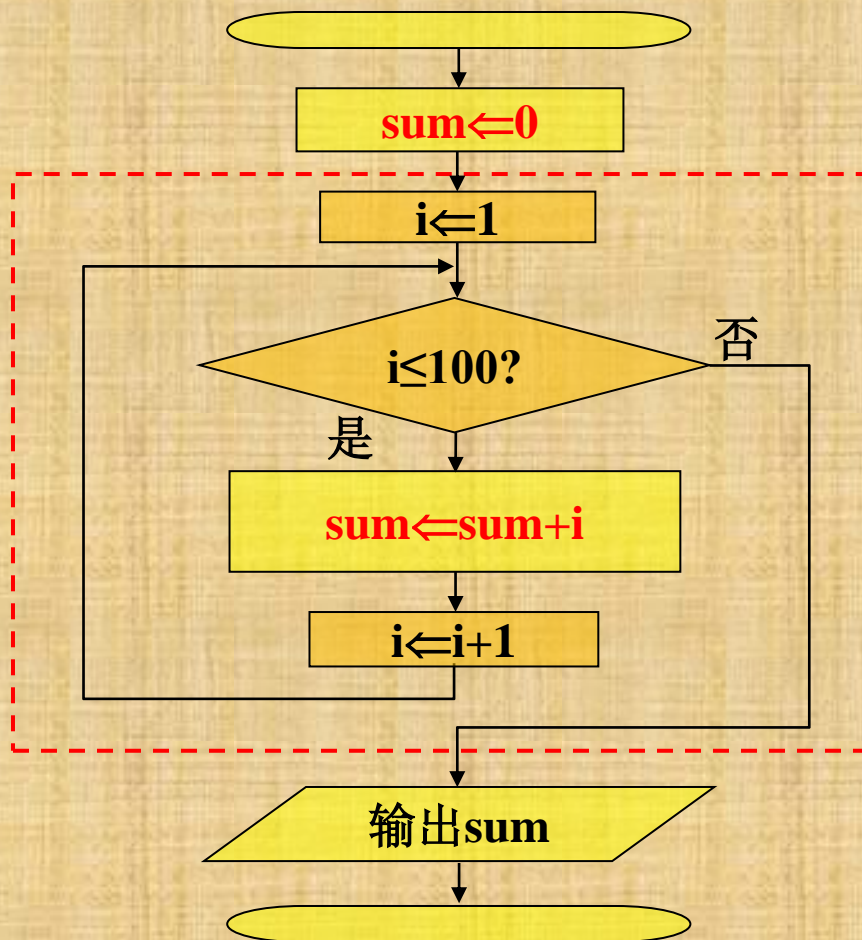
```
/******  
*  MA2_13B.cpp  
*  输出 1+2+ ... +100 的值 (使用for循环)  
*****/
```

```
#include <iostream>  
using namespace std;  
int main( )  
{   int i , sum;  
    sum=0 ;  
    for(i=1;i<=100;i++ ) // 对i取1~100, 反复执行下面函数体  
    {   sum=sum+i ;   }  
    cout<< "1+2+ ... +100= " <<sum<<endl;  
    return 0;  
}
```

注4: for的圆括号中的第一个式子“i=1”只执行一次, 而每次执行完循环体后一定要执行第三个式子“i++”, 即 i=i+1。

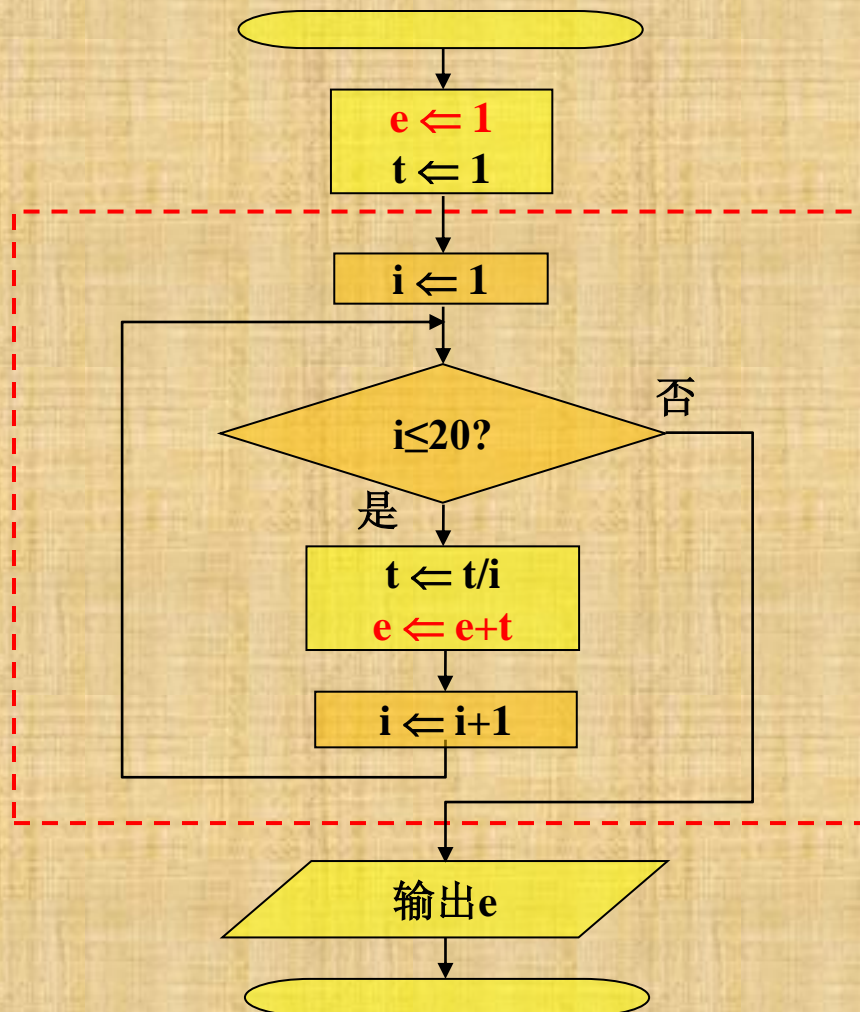
注5: 用循环进行数据累加, 累加器一定要先清0。

## 程序MA2\_13B.cpp的流程框图





例C2.14 计算 式子  $1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{20!}$  。



```
/******
```

```
* MC2_14.cpp
```

```
* 输出  $1+1/1!+1/2! \cdots +1/20!$  的值
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{ int i;
```

```
    double e=1 , t=1 ; //定义变量e、t并赋初值e、t都为1
```

```
    for(i=1;i<=20;i++) //对i=1~20反复执行循环体
```

```
    { t=t/i; e=e+t; } //t为每一项的值1/i!,为前一项除以i
```

```
    cout<< "  $1+1/1!+1/2! \cdots +1/20!=$  " <<e<<endl;
```

```
    return 0;
```

```
}
```

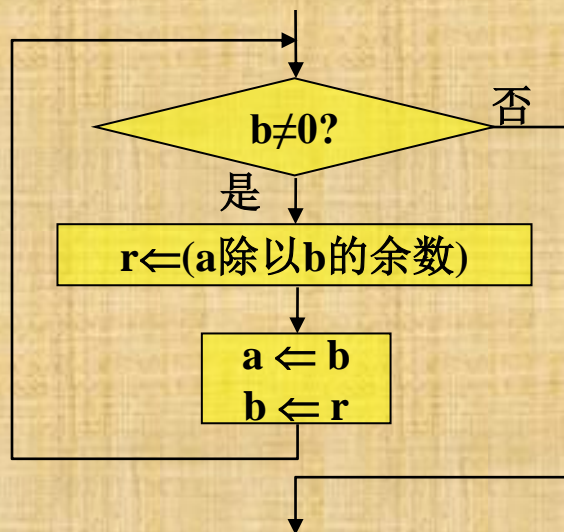
例**C2.15** 输入两个正整数，用辗转相除法求它们的最大公约数。

对于正整数**a**和**b**，反复进行带余数除法如下：

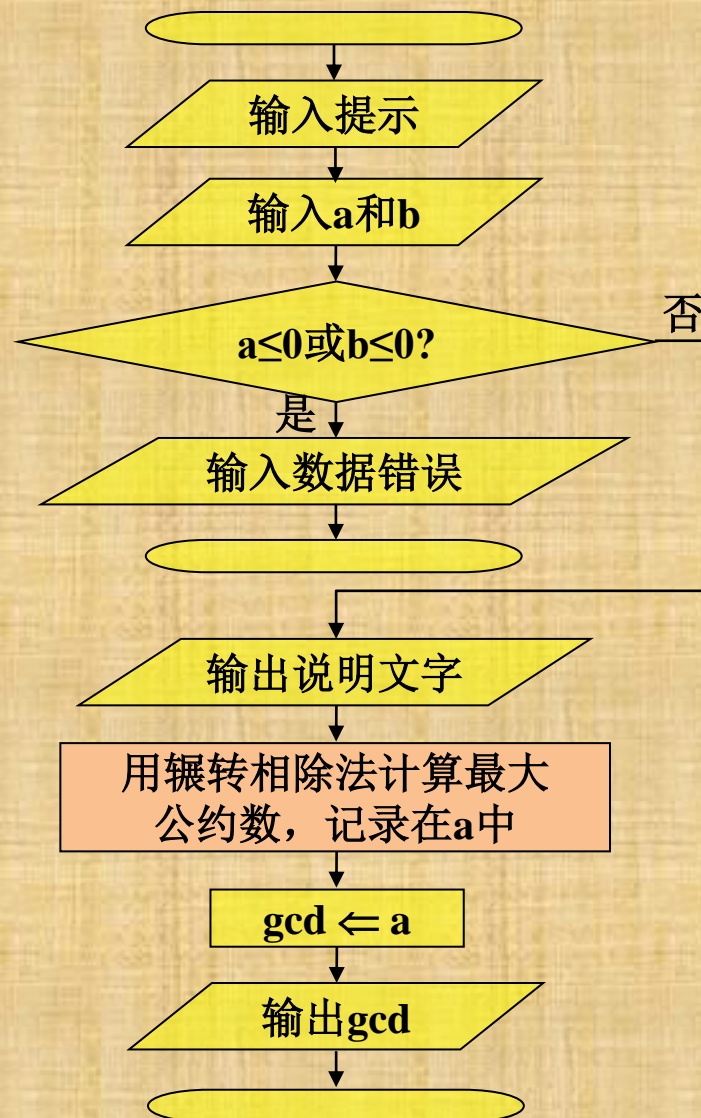
$$\begin{aligned} \mathbf{a} &= \mathbf{b}q_0 + r_1, & 0 < r_1 < b \\ b &= r_1q_1 + r_2, & 0 < r_2 < r_1 \\ &\dots \dots \dots \\ r_{n-2} &= r_{n-1}q_{n-1} + r_n, & 0 < r_n < r_{n-1} \\ r_{n-1} &= \mathbf{r_n}q_n + \mathbf{0} \end{aligned}$$

则 $\mathbf{r_n}$ 是**a**和**b**的最大公约数。

## 计算a,b最大公约数的流程框图



## 整个程序的流程框图





```
/**
*****

```

```
* MC2_15.cpp

```

```
* 输入两个正整数，输出两数的最大公约数

```

```
*****

```

```
#include <iostream>

```

```
using namespace std;

```

```
int main( )

```

```
{   int a , b , r , gcd ;

```

```
    cout<< "输入两个正整数" <<endl;

```

```
    cin>>a>>b;

```

```
    if(a<=0 || b<=0)

```

```
    {   cout<< "输入的数据有误，程序终止" <<endl; return 1; }

```

```
    cout<<a<< "和" <<b<< "的最大公约数为: ";

```

```
    while( b!=0 )    //辗转相除法求最大公约数.若b=0，则a为最大公约数

```

```
    {   r=a%b; a=b; b=r; }    //将b和a%b重新保存在a、b中

```

```
    gcd=a;                //将最大公约数存放在变量gcd中

```

```
    cout<<gcd<<endl;    return 0; }

```

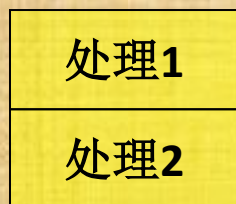
注6：上述程序的if语句中的“return 1;”执行后将终止程序的运行。程序非正常终止时,习惯用“return 1;”或“return 2;”等表示因为1号错误或2号错误而提前终止程序运行。

## 2.3.3 结构化的程序设计及结构化流程图

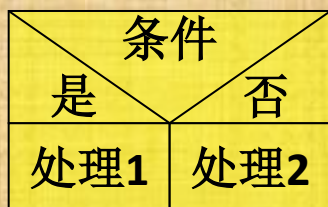
结构化程序设计基本结构：

顺序结构，分支结构；循环结构

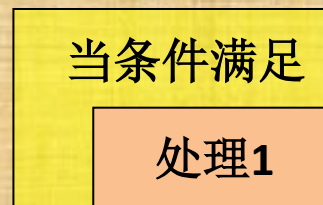
结构化流程图：Nassi-Shneiderman图（N-S图）



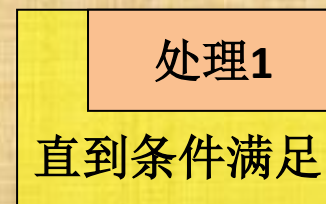
顺序结构



分支结构



当型循环

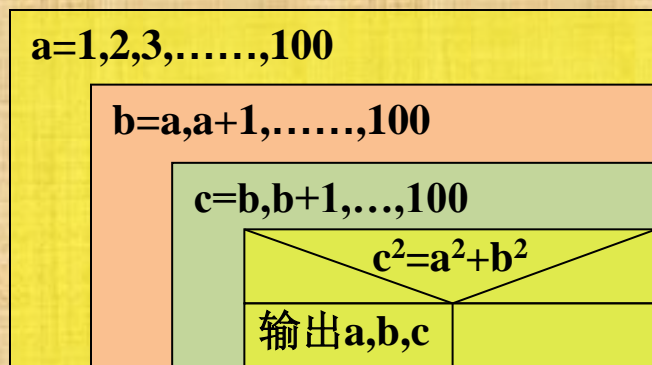


直到型循环

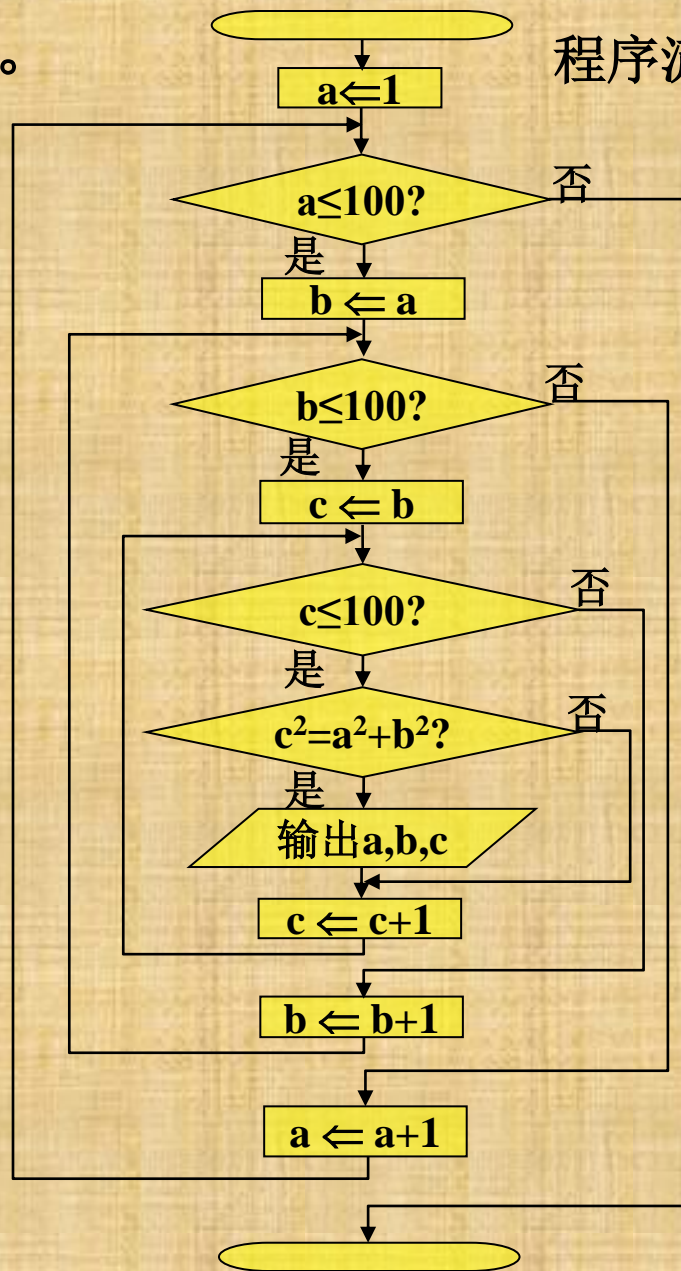
例C2.16 找出100以内的勾股数。

## 例C2.16 找出100以内的勾股数。

N-S图



程序流程图



```
/******  
*   MC2_16.cpp  
*   找出100以内的勾股数 (满足 $a^2+b^2=c^2$ 的数)  
*****/
```

```
#include <iostream>  
using namespace std;  
int main( )  
{   int a , b , c ;  
    for( a=1;a<=100;a++ )      //a值尝试1、 2、 ...、 100  
    {   for( b=a;b<=100;b++ )  //b值尝试a、 a+1、 ...、 100  
        {   for( c=b; c<=100; c++ ) //c值尝试 b、 b+1、 ...、 100  
            {   if( c*c==a*a+b*b ) //判断 $c^2=a^2+b^2$ ,等式用==  
                {   cout<<a<< "\\t"<<b<<"\\t"<<c<<endl;   }  
            }  
        }  
    }  
    }   return 0;  
}
```

注7：进一步分析知 $b=a$ 时一定不是勾股数，但对于这些工作大家不要费力去做，让计算机多计算一点也不会严重影响运行时间，却能够让编程者节省很多的时间。



# 第3章 数据与运算

## 3.1 引言

第2章的知识对于编写程序来说仅仅是入门，进一步编写程序还会遇到一些问题，看后面几个例子。

例**B3.1** 输入5个正整数，求它们的最大公约数。

```
# include <iostream>           //程序MB3_1.cpp
using namespace std;
int main( )
{   int a,b,c,d,e,r;
    cin>>a>>b>>c>>d>>e;
    while(b!=0)    //求a、b的最大公约数，记录在a中
    {   r=a%b; a=b; b=r; }
    while(d!=0)    //求c、d的最大公约数，记录在 c中
    {   r=c%d; c=d; d=r; }
    while(c!=0) //求a、c的最大公约数,即原来a,b,c,d的最大公约数,记录在a中
    {   r=a%c; a=c; c=r; }
    while(e!=0)    //求a、e的最大公约数,即输入数的最大公约数,记录在a中
    {   r=a%e; a=e; e=r; }
    cout<<"5个数的最大公约数为: "<<a<<endl;
    return 0;
}
```

程序 MB3\_1.cpp 重复太多 —— 可用第6章介绍的函数改写

## 程序MB3\_1.cpp的改写

```
#include <iostream>
using namespace std;
int gcd(int x,int y) //函数gcd用于求x和y最大公约数,函数值为int型
{   int r;
    while(y!=0) { r=x%y; x=y; y=r; }
    return x;
}
int main( )
{   int a,b,c,d,e,y;
    cin>>a>>b>>c>>d>>e;
    y=gcd(gcd(a,b),gcd(c,d)); y=gcd(y,e); //调用函数gcd计算最大公约数
    cout<<"5个数的最大公约数为: "<<y<<endl;
    return 0;
}
```

例B3.2 计算算术表达式“ $123*(456-356)+73$ ”。

```
#include <iostream>           //程序MB3_2.cpp
using namespace std;
int main( )
{   cout<< 123*(456-72)+1137;
    return 0; //分号要用英文字符
}
```

汉字分号“;”

程序 MB3\_2.cpp 误用了汉字字符——改用英文字符

例B3.3 输入一个学生的数学、英语和C++成绩并输出平均成绩。

```
#include <iostream>           //程序MB3_3.cpp
using namespace std;
int main( )
{   int math,english,C++; double average;
    cin>>math>>english>>C++;
    average=(math+english+C+)/3.0;
    cout<<"平均成绩为: "<<average<<endl;
    return 0;
}
```

程序 MB3\_3.cpp C++不能做变量名——可改成C++



例B3.4 输入非零数n，计算阶乘n!。

```
#include <iostream> //程序MB3_4.cpp
using namespace std;
int main( )
{   int f=1,n,i;
    cin>>n; cout<<n<<"!=";
    for(i=1;i<=n;i++) { f=f*i; }
    cout<<f<<endl;
    return 0;
}
```

运行结果为：

12 ✓

12!=479001600

或

17 ✓

17!= -288522240

程序 MB3\_4.cpp 当输入n大于12时，结果超出存储单元中数据的大小范围——改用double类型

## 3.2 字符集与标识符

表3-1 C++字符集

26个小写字母	abcdefghijklmnopqrstuvwxyz
26个大写字母	ABCDEFGHIJKLMNOPQRSTUVWXYZ
10个数字	0123456789
其它字符	+ - * / = , . _ : ; ? \ ' " ~   ! # % & ( ) [ ] { } ^ < > 空格

**C++程序只能用英文字符，不能用汉字**

首字符：\_ 和字母

后续字符：\_ 和字母、数字

构成规则

**table2**

C++标识符：C++的名字

标识符设计

**myCar**、**sizeOfChar** 骆驼法

**nScore**、**dSalary** 匈牙利法

**C++关键字：C++预定义的名字，不能另作它用。见表3-2**

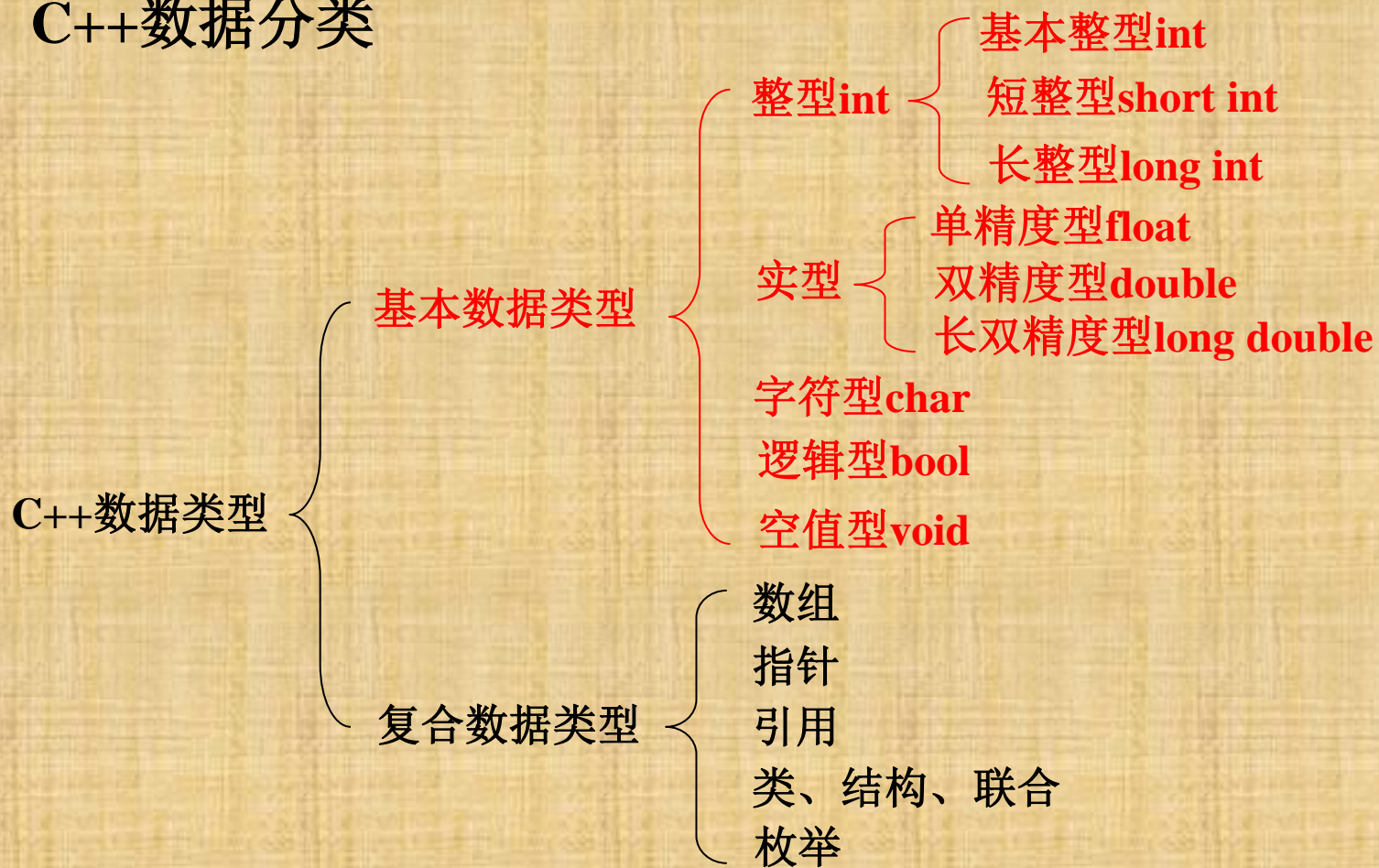
**表3-2 C++关键字**

<b>asm</b>	<b>auto</b>	<b>bool</b>	<b>break</b>	<b>case</b>	<b>catch</b>	<b>char</b>
<b>class</b>	<b>const</b>	<b>const_cast</b>	<b>continue</b>	<b>default</b>	<b>delete</b>	<b>do</b>
<b>double</b>	<b>dynamic_cast</b>	<b>else</b>	<b>enum</b>	<b>explicit</b>	<b>export</b>	<b>extern</b>
<b>false</b>	<b>float</b>	<b>for</b>	<b>friend</b>	<b>goto</b>	<b>if</b>	<b>inline</b>
<b>int</b>	<b>long</b>	<b>mutable</b>	<b>namespace</b>	<b>new</b>	<b>operator</b>	<b>private</b>
<b>protected</b>	<b>public</b>	<b>register</b>	<b>reinterpret_cast</b>	<b>return</b>	<b>short</b>	<b>signed</b>
<b>sizeof</b>	<b>static</b>	<b>static_cast</b>	<b>struct</b>	<b>switch</b>	<b>template</b>	<b>this</b>
<b>throw</b>	<b>true</b>	<b>try</b>	<b>typedef</b>	<b>typeid</b>	<b>typename</b>	<b>union</b>
<b>unsigned</b>	<b>using</b>	<b>virtual</b>	<b>void</b>	<b>volatile</b>	<b>wchar_t</b>	<b>while</b>

**注1：**在C++程序中，只有出现在双引号中的字符可以是汉字字符，程序其它部分不能出现汉字字符，尤其是标识符不能是汉字，分号、逗号等不能是汉字，特别要注意不能出现汉字空格。

## 3.3 基本数据类型

### C++数据分类





## 整数类型

	基本类型	有符号类型	无符号类型	所占字节
基本整型	<b>int</b>	<b>signed int</b>	<b>unsigned int</b>	<b>2 或 4</b>
短整型	<b>short int</b>	<b>signed short int</b>	<b>unsigned short int</b>	<b>2</b>
长整型	<b>long int</b>	<b>signed long int</b>	<b>unsigned long int</b>	<b>4</b>

表3-3 整数类型的属性

	类型	占用的字节数	数的表示范围
基本类型 (默认有符号)	<b>int</b>	<b>2或4</b>	<b><math>-2^{15} \sim 2^{15}-1</math> 或 <math>-2^{31} \sim 2^{31}-1</math></b>
	<b>short int</b>	<b>2</b>	<b><math>-2^{15} \sim 2^{15}-1</math> (<math>-32768 \sim 32767</math>)</b>
	<b>long int</b>	<b>4</b>	<b><math>-2^{31} \sim 2^{31}-1</math> (<math>-2147483648 \sim 2147483647</math>)</b>
有符号类型	<b>signed int</b>	<b>2或4</b>	<b><math>-2^{15} \sim 2^{15}-1</math> 或 <math>-2^{31} \sim 2^{31}-1</math></b>
	<b>signed short int</b>	<b>2</b>	<b><math>-2^{15} \sim 2^{15}-1</math></b>
	<b>signed long int</b>	<b>4</b>	<b><math>-2^{31} \sim 2^{31}-1</math></b>
无符号类型	<b>unsigned int</b>	<b>2或4</b>	<b><math>0 \sim 2^{16}-1</math> 或 <math>0 \sim 2^{32}-1</math></b>
	<b>unsigned short int</b>	<b>2</b>	<b><math>0 \sim 2^{16}-1</math> (<math>0 \sim 65535</math>)</b>
	<b>unsigned long int</b>	<b>4</b>	<b><math>0 \sim 2^{32}-1</math> (<math>0 \sim 4294967295</math>)</b>

注1: 现在流行的C++编译软件(包括VS2008), **int**、**signed int**和**unsigned int**都占4个字节。

注2: 对于各种不同的**int**型, 可以省略**int**, 如**short int**可以简单地用**short**表示、**long int**可以用**long**表示、**unsigned int**可以用**unsigned**表示。

### 表3-4 实数类型的属性

类型	占用的字节数	绝对值大小的范围	有效数字个数
float	4	$10^{-38} \sim 10^{38}$	7
double	8	$10^{-308} \sim 10^{308}$	15
long double	8或10	$10^{-308} \sim 10^{308}$ 或 $10^{-4932} \sim 10^{4932}$	15或19

## 字符类型

### 表3-5 字符类型的属性

类型	占用的字节数	数的表示范围
char (默认有符号)	1	$-2^7 \sim 2^7 - 1$ (-128~127)
signed char	1	$-2^7 \sim 2^7 - 1$
unsigned char	1	$0 \sim 2^8 - 1$ (0~255)

逻辑类型: **bool**

占1个字节, 值为 **true**或 **1**, **false**或 **0**

空值类型: **void**

表示无值(函数返回)或不分类型(存储单元地址)

注3: 字符型(char、signed char、unsigned char)在计算机内部就是一个单字节的整数类型, 可以进行整数的各种运算, 但在输入和输出时则是以数值(即ASCII码)所对应的字符的形式出现的。

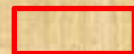
注4: 逻辑型(bool)在计算机内部就是1位二进制数的整数类型, 只含0(表示false)或者1(表示true), 但是占用一个字节, 同样可以进行整数的各种运算。

## 基本类型变量所占字节比较图示

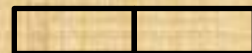
bool



char



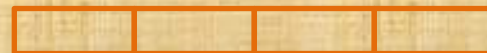
short int



int (VS2008)和 long int



float



double



## 变量及变量定义——变量使用前要先定义

```
int  a , b ;  
float c,d;  
double x , y , z ;
```

### 变量初始化

```
int  a=3 , b=4 ;  
float c= -0.75,d=8;  
double x , y=3.14 , z=355.0/113 ;
```

注5：变量在使用前必须先定义。而且变量只能定义一次，不能重复定义。

### 变量说明

```
extern int  u,v;
```



## 3.4 常量

### 3.4.1 字面常量

表3-6 各种形式的常量

整型常量  
实型常量  
字符常量

	基本整型	长整型	无符号型
十进制	11, -135, +17, 0	-135L	11u, 0U
八进制(前缀0)	012, 057613, -07311	057613L	012u
16进制(前缀0x或0X)	0x12ff, -0X12aa, 0xF	0x12ffL, 0xFI	0x12ffu
单精度(后缀f或F)	12.75f, 2.f, -3.14e2f, 3E-9f, -3.14e2F		
双精度	12.75, -314., .178, 1e-6, 7.18E5 小数点形式或e形式		
可显字符	'a', 'Y', '\$', '@', '=', '8', '?', ''		
转义序列(预定义)	'\\n', '\\t', '\\r', '\\a', '\\b', '\\\\', '\\'		
转义序列(ASCII)	'\\12', '\\103', '\\xA', '\\x43', '\\x63', '\\x38', '\\xFF', '\\0'		
字符串常量	"Hello", "This is C++\\n", "\\\"Books\\\"", "a", ""		
逻辑常量	true, false		

16进制

8进制

- 注1: 双精度常量后面跟后缀f或F即为单精度常量, 双精度常量的小数点前面或后面可以没有数字, 但不能都没有数字。常量1e-6表示实数 $1 \times 10^{-6}$ , 称为指数形式常量。
- 注2: 在字符常量中只能表示一个字符, 若字符常量写成 'ab'、'>='则是错误的。
- 注3: 在字符常量的预定义转义序列中: '\n'表示回车换行, '\t'表示跳格, '\r'表示回车, '\a'表示警报声, '\b'表示回退一个字符, '\\'表示反斜杠字符, '\''表示单引号字符。
- 注4: 字符常量中的ASCII转义序列是用最多3位八进制或最多两位十六进制ASCII码值表示字符, 如'\103'表示ASCII码为 $(103)_8$ 的字符, 即 'C', '\x63'表示ASCII码为 $(63)_{16}$ 的字符, 即 'c'。在八进制表示的字符常量中不能出现非八进制的数字, 如'\83'就是错误的常量。

## 使用常量的程序例子

### 例A3.5 使用十六进制表示最大整数和最小整数。

```
/******
```

```
* MA3_5.cpp
```

```
* 输出最大和最小的整数(使用十六进制整型常量)
```

```
*****/
```

```
# include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{ int maxInt=0x7fffffff,minInt=0x80000000;
```

```
  cout<<"最大整数="<<maxInt<<endl;
```

```
  cout<<"最小整数="<<minInt<<endl;
```

```
  //输出八进制数、十六进制数
```

```
  cout<<oct<<"八进制输出: maxInt="<<maxInt
```

八进制输出

```
    <<"\tminInt="<<minInt<<endl;
```

```
  cout<<hex<<"十六进制输出: maxInt="<<maxInt
```

十六进制输出

```
    <<"\tminInt="<<minInt<<endl;
```

```
  cout<<dec<<"恢复十进制输出: maxInt="<<maxInt
```

十进制输出


```
    <<"\tminInt="<<minInt<<endl;
```

```
  return 0; }
```



## 例A3.6 求最小n使得 $n! > 10^{14}$ (使用指数形式常量)。

```
/******  
*   MA3_6.cpp  
*   求最小n使得 $n! > 1e14$  (使用指数形式常量)  
*****/  
  
#include <iostream>  
using namespace std;  
int main( )  
{   double t=1;  
    int n=1;  
    while(t<=1e14)  
    {   n++; t=t*n; }  
    cout<<"n="<<n<<endl;  
    return 0;  
}
```





### 例A3.7 输入一个字符，判断是否为字母(使用可显字符常量)。

```
/******
```

```
* MA3_7.cpp
```

```
* 输入一个字符，判断是否为字母(使用可显字符常量)
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{   char ch;   cout<<"输入一个字符: ";
```

```
    cin>>ch; ch是大写字母 //用“cin>>ch;”无法输入空格、跳格字符
```

```
    if( (ch>='A'&&ch<='Z') || (ch>='a'&&ch<='z') ) //字符ASCII码比较
```

```
    {   cout<<"输入字符为字母字符"<<endl; }
```

**ch是小写字母**

```
    else
```

```
    {   cout<<"输入字符为非字母字符"<<endl; }
```

```
    return 0;
```

```
}
```

注5：判断ch中的字符是否为大写字母用“**ch>='A'&&ch<='Z'**”，而不用“**ch>=65&&ch<=90**”，因为在计算机内部'A'表示大写字母A的ASCII编码65。

### 3.4.2 符号常量

符号常量定义形式:

```
const int  maxInt = 0x7fffffff;
```

```
const double pi = 3.141592653589793;
```

例A3.8 输入一个角度度数(角度制), 输出相应的弧度数(使用符号常量)。

```
/******
```

```
*  MA3_8.cpp
```

```
*  输入一个角度度数(角度制), 输出相应的弧度数(使用符号常量)
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
const double pi=3.141592653589793; //定义符号常量pi, 通常在main前定义
```

```
int main( )
```

```
{    double degree,arc;
```

```
    cin>>degree;
```

```
    arc=dereec*pi/180;    //使用符号常量
```

```
    cout<<degree<<"度的角度等于"<<arc<<"弧度"<<endl;
```

```
    return 0;
```

```
}
```

符号常量pi  
表示3.141592653589793

## 3.5 基本运算及表达式

### C++运算符分成

二元运算符(最常见):

$456-72$  ,  $a\%5$  ,  $x/y$  ,  $\text{score}<60$  ,  $a=123$  ,  $a==123$

一元运算符:

$-x$  ,  $i++$

三元运算符:

$x\geq 0?x*x*x+2*x*x+7:x*x+2*x+11$  表示有条件计算

## 表3-7 C++运算符的优先级和结合方向

优先级	运 算 符	结合方向
1	( )、.、->、[]、::、.*、->*、& (引用)	→
2	* (指向)、&(取地址)、new、delete、!、~、++、--、-(取负)、sizeof( )、显式转换	←
3	*/、/、%	→
4	+、-	→
5	<<、>>	→
6	<、<=、>、>=	→
7	==、!=	→
8	&(按位与)	→
9	^	→
10		→
11	&&	→
12		→
13	? : (三元运算符)	←
14	=、+=、-=、*=、/=、%=、<<=、>>=、&=、^=、 =	←
15	,	→



### 3.5.1 算术运算、比较运算和逻辑运算

算术运算符	$+, -, *, /$ (分整数除和实数除), $\%$	结果为整数或实数
比较运算符	$<, <=, >, >=, ==, !=$	结果为true或false
逻辑运算符	$\&\&,   , !$	结果为true或false

除法:                       $1/2$ 为0                       $1.0/2$ 为0.5  
                                  $-15/4$ 为-3                       $-15/4.0$ 为-3.75

取模:                       $27\%8$ 为3                       $27\%(-8)$ 为3  
                                  $(-27)\%8$ 为-3                       $(-27)\%(-8)$ 为-3

符号与被  
除数相同

逻辑运算可构成复杂的条件: 如判断闰年

$(year\%4==0\&\&year\%100!=0)||year\%400==0$

注1: 逻辑表达式的操作数也要求是逻辑型, 若是数值类型, 则非零数会转换为逻辑量true, 零会转换为逻辑量false。

## 3.5.2 增量/减量运算、赋值运算和逗号运算

增量减量运算符	$++$ , $--$ (分前置 $++n$ 和后置 $n++$ )	前置结果为变量 后置结果为数值
赋值运算符	$=$ , $+=$ , $-=$ , $*=$ , $/=$ , $\%=$ $<<=$ , $>>=$ , $\&=$ , $\^=$ , $ =$	结果为 赋值后的变量
逗号运算符	,	结果为 最后计算的值

若有 `int n=10; double x=3.14;`

增量运算符: `n++`    `++n`    均表示 `n=n+1`

`x++`    `++x`    均表示 `x=x+1`

减量运算符: `n--`    `--n`    均表示 `n=n-1`

`x--`    `--x`    均表示 `x=x-1`

若有 `int n=10; double x=3.14;`

则执行 `n++; --x;` 后的值 `n` 为 11, `x` 为 2.14

注2: 只有表示存储单元的表达式可以作为赋值运算的左操作数, 这样的操作数称为左值, 所有不是左值的表达式称为右值。常量及计算值显然是右值。



赋值运算后的结果仍然是左值, 如:  $(y=3+5)=7$

注3: 用户可以对变量连续赋值, 如 “ $a=b=c=1$ ”表示 $a$ 、 $b$ 、 $c$ 都赋上值1,其本质是用赋值表达式给变量赋值,即 “ $a=(b=(c=1))$ ”。

复合赋值运算符:  $x+=5$  表示  $x=x+5$ ;  $y*=3-4$  表示  $y=y*(3-4)$   
 $+=, -=, *=, /=, \%=$   
 $<<=, >>=, \&=, ^=, |=$

注4: 在C++中, 前置增量/减量运算、赋值运算和复合赋值运算的结果都是左值。



## 增量运算及复合赋值运算的程序例子

例C3.9 输入一批数据,输出数据个数及平均值(增量及复合赋值)。

```
/******
```

```
* MC3_9.cpp
```

```
* 输入一批数据,输出数据的个数及平均值(增量及复合赋值)
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{ double x,sum; int n;
```

```
sum=n=0; //连续赋值, 0连续赋值给n和sum, 即 sum=(n=0)
```

```
while(cin>>x) //反复输入x, 直到单独一行输入^Z终止输入
```

```
{
```

```
    n++; sum+=x;
```

```
}
```

```
cout<<"共"<<n<<"个数据, 平均值为"<<sum/n<<endl; return 0; }
```

连续赋值

增量运算

复合赋值运算

注5: ^Z表示文件结束符, 可在键盘上按住Ctrl键再按Z键输入该结束符。



逗号运算符：， 表示从左向右依次计算表达式

如      a=3, b=4      i=0, j=n-1

```
for(F1=F2=Fn=1,i=3;i<=20;F1=F2,F2=Fn,i++)  
{ Fn=F1+F2; }
```

### 3.5.3 字符数据的处理

字符变量中存放的是字符的**ASCII码**

如：char c1='A', c2='6';

则变量的内部数据如下：

变量c1    **0x41**

变量c2    **0x36**

字符变量处理时：

- 输入输出时是**字符**
- 比较和算术运算时是ASCII码，即**整数值**

# 字符数据处理的程序例子

例**C3.10** 输入一行字符，将其中的小写字母转换成大写字母  
(字符算术运算)。

```
/******
```

```
* MC3_10.cpp
```

```
* 输入一行字符，将其中的小写字母转换成大写字母(字符算术运算)
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{ char ch;
```

结果为ch

```
while( (ch=cin.get( ))!='\n' ) //反复输入字符直到回车'\n'
```

```
{ if( ch>='a'&&ch<='z' ) //判断ch为小写字母
```

字符比较

```
{ ch=ch+('A'-'a'); } //字符加上大写到小写的差值即转为大写
```

```
cout<<ch;
```

字符算术运算

```
}
```

```
cout<<endl;
```

```
return 0;
```

```
}
```

注6: 从键盘输入时是成批地输入数据直到回车, 输入的数据暂时保存在内存的某个地方(即输入缓冲区), 回车后才利用cin或cin.get( )反复读取暂存的数据。

注7: 输入字符也可用cin>>ch的形式, 但是这样输入字符会忽略输入的空白间隔符(空格、Tab键(跳格)和回车)。

## 3.6 对表达式的进一步说明

### 3.6.1 整数除和算术溢出

整数除以整数，结果为**整数**

除法两边有一个实数，结果为**实数**

运行的整数结果**超出4字节，高位丢失**(算术溢出)



## 例B3.11 演示整数除和算术溢出。

```
/******
```

```
* MB3_11.cpp
```

```
* 演示整数除和算术溢出
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{ //计算 1+1/2+1/3+...+1/10
```

```
int i; double s1=0,s2=0;
```

```
for(i=1;i<=10;i++) { s1+=1/i; } //1/i为整数除, i>1时结果为0
```

```
for(i=1;i<=10;i++) { s2+=1.0/i; } //1.0/i为实数除, i>1时结果为小数
```

```
cout<<"s1="<<s1<<",s2="<<s2<<endl;
```

```
//两个变量i、j的乘积超出int型数据范围,则超出4字节的高位部分丢失
```

```
int j,m,k1,k2; i=0x30000; j=0x20000; m=0x10000;
```

```
cout<<hex<<"i="<<i<<",j="<<j<<",m="<<m<<endl; //十六进制输出
```

```
k1=i*j/m; //等价于 k1=(i*j)/m;计算i*j时超出int数据范围(溢出),残留0
```

```
k2=i/m*j; //i/m值为3, 再计算3*j得0x60000, 此时不会产生算术溢出
```

```
cout<<"k1="<<k1<<",k2="<<k2<<endl; //十六进制输出
```

```
return 0; }
```

### 3.6.2 比较运算的特殊用法

实数相等比较要考虑**误差影响**  
连续不等式要用**逻辑运算**

#### 实数相等比较

当  $x=3.0*0.1$  ,  $y=0.3$  时:

条件  $x==y$  结果为 **false**, 因  $x$  和  $y$  有**误差**

条件  $\text{fabs}(x-y)<\text{eps}$  结果为 **true** ( $\text{eps}=1\text{e-}7$ )

正确

#### 连续不等式

当  $m=20$  时:

条件  $0\leq m\leq 10$  结果为 **true**, 实际比较  $\text{true}\leq 10$

条件  $0\leq m\&\&m\leq 10$  结果为 **false**

正确

演示  
程序  
见后面

## 例B3.12 演示实数相等比较和连续不等式的表示。

```
# include <iostream>           //程序 MB3_12.cpp
# include <cmath>
using namespace std;
const double eps=1e-7;         //定义误差常量
int main( )
{   //误差影响实数的相等比较
    double x,y; x=3.0*0.1; y=0.3; cout<<"x="<<x<<"\ty="<<y<<endl;
    if(x==y) cout<<"(x==y)结果为true"<<endl; //if后单个语句可省略花括号
    else cout<<"(x==y)结果为false"<<endl; //else后单个语句也可省花括号
    if(fabs(x-y)<eps) cout<<"|x-y|<1e-7结果为true"<<endl;
    else cout<<"|x-y|<1e-7结果为false"<<endl;
    //连续不等式的数学式与C++表示式不同
    int m=20; cout<<"m="<<m<<endl;
    if(0<=m<=10) cout<<"(0<=m<=10)结果为true"<<endl;
    else cout<<"(0<=m<=10)结果为false"<<endl;
    if(0<=m&&m<=10) cout<<"(0<=m&&m<=10)结果为true"<<endl;
    else cout<<"(0<=m&&m<=10)结果为false"<<endl;
    return 0; }
```

等价于  $(0 \leq m) \leq 10$   
即  $0$  或  $1 \leq 10$  的比较



注1：一般来讲，编程时对于 $>$ 、 $>=$ 、 $<$ 、 $<=$ 等大小比较不去考虑误差，对于经过一系列计算的实数值要进行相等比较时才会考虑误差，并使用`fabs(x-y)<eps`之类的式子进行比较。

注2：两个double型实数进行相等比较时误差界限通常使用 $10^{-7}$ 。若两个比较的实数都相当大，需要比较相对误差界限，如`fabs((x-y)/x)<1e-7`。在使用double型的字面常量时可以取存储精度，即取16位有效数字，比double型有效位数多一位。

注3：在C++中，数学关系 $0 \leq m \leq 10$ 要用“`0<=m&&m<=10`”表示，不能用“`0<=m<=10`”表示。



### 3.6.3 前置与后置的增量和减量

增量运算为变量增加1，减量运算为变量减去1

作为表达式的一部分时：以增量为例

- 前置增量为**先变量增1**，**变量再参与整个表达式计算**，增量表达式的值为**左值**。
- 后置增量为**暂存原值**，**变量增1**，暂存的**原值再参与整个表达式计算**，增量表达式的值为**右值**。

例如：

当a值为10时：  $b=++a$ ；      结果为 a值11，b值也为**11**

当a值为10时：  $b=a++$ ；      结果为a值11，b值**10**

注4：前置增量和后置增量，前置减量和后置减量，其操作数必须是左值。例如 $a++$ ，则a必须是左值。

注5：前置增量/减量表达式为左值，后置增量/减量表达式为右值。如 $++a$ 为左值， $a++$ 为右值。

## 例B3.13 演示前置与后置的增量和减量运算。

```
/**
```

```
* MB3_13.cpp
```

```
* 演示前置与后置的增量和减量运算
```

```
*/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{   int a,b; double x;
```

```
    // 独立的增量和减量运算
```

```
    a=10; x=3.14; cout<<"a="<<a<<"\tx="<<x<<endl;
```

```
    --a; ++x;      //--使变量值减1, ++使变量值增1, 不管对整数还是实数
```

```
    cout<<"after - -a; a="<<a<<"\t after ++x; x="<<x<<endl;
```

```
    a=10; x=3.14;      cout<<"a="<<a<<"\t x="<<x<<endl;
```

```
    a-- ; x++;      //单独使用时, --、++的前置运算与后置运算效果一样
```

```
    cout<<"after a- -; a="<<a<<"\t after x++; x="<<x<<endl;
```

```
    // 参与表达式的增量与减量前置与后置有不同效果
```

```
    a=10; cout<<"begin a="<<a<<endl ;
```

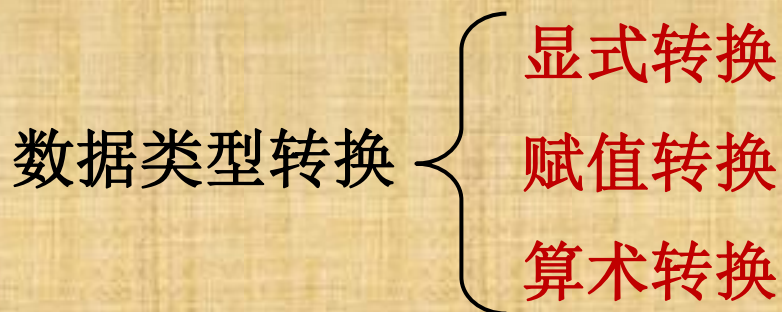
```
    b=++a;      cout<<"after b=++a; a="<<a<<" , b="<<b<<endl;
```

```
    a=10; cout<<"begin a="<<a<<endl ;
```

```
    b=a++;      cout<<"after b=a++; a="<<a<<" , b="<<b<<endl;
```

```
    return 0; }
```

### 3.6.4 类型的转换



显式转换：

若 `int a=3; double x=-3.84; char ch='0';`

表3-8 不同类型转换时的形式与结果值

<code>int(x)</code>	<code>(int)x</code>	<code>static_cast&lt;int&gt;(x)</code>	取整数部分，值为 -3
<code>double(a)</code>	<code>(double)a</code>	<code>static_cast&lt;double&gt;(a)</code>	变成double形式，值为3.0
<code>int(ch)</code>	<code>(int)ch</code>	<code>static_cast&lt;int&gt;(x)</code>	变成int形式，值为48

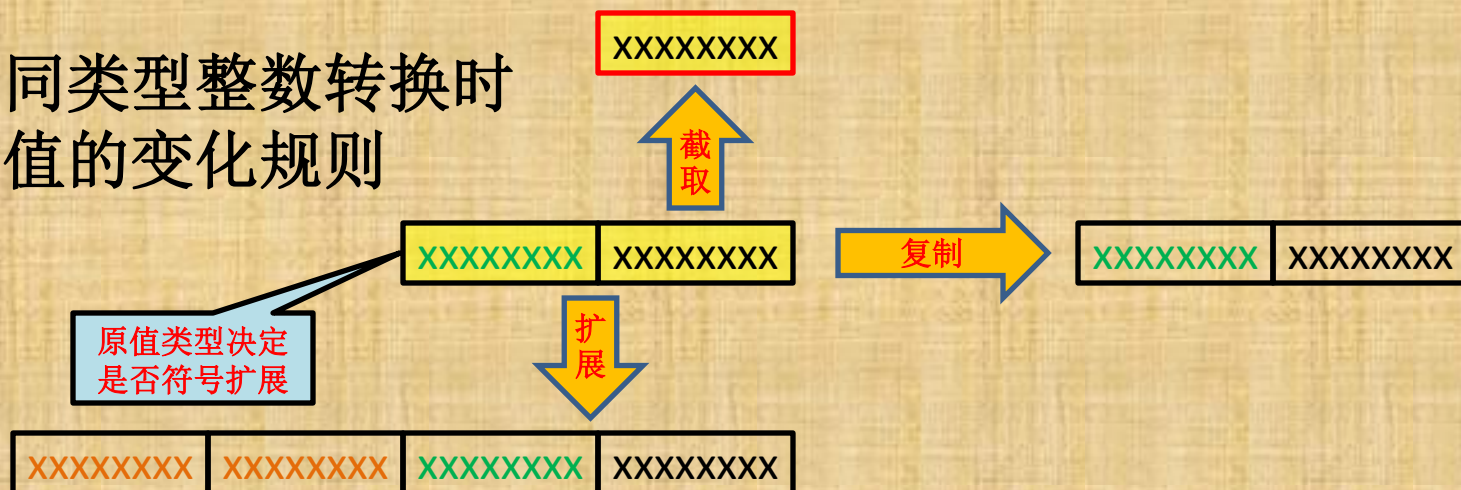
注6：在表3-8中，`int(x)`、`(int)x`、`static_cast<int>(x)`都表示将x的值显式转换为int类型的值，是3种不同但是等价的显式转换的写法，效果一样。



## 表3-9 不同类型转换时数值的变化

表达式	转换成的类型	数值变化
整型	等字节的整型	字节内二进制数不变
较长字节整型	较短字节整型	截取低位字节的二进制数
较短字节无符号整型	较长字节整型	高位字节进行0扩展的二进制数
较短字节有符号整型	较长字节整型	高位字节进行符号扩展的二进制数
double	float	舍入到更短的有效位，降低精确度
float	double	取原数值，保留原数的精确度
整型或实型	逻辑型	非零取true，零取false
逻辑型	整型或实型	true为1，false为0
实型	整型	截断小数，取整数部分
整型	实型	整数转成实数，数值不变

### 不同类型整数转换时 数值的变化规则





注7：符号扩展是指如果原来字节的最高位是0，则高位字节进行0扩展(高位全部补0)；如果原来字节的最高位是1，则高位字节进行1扩展(高位全部补1)。符号扩展保证正数扩展后仍为正数，负数扩展后仍是负数。

注8：整数类型转换成整数类型时，内部的二进制(分析时常用十六进制)数值的变化规律比较简单。整数转换成等长字节类型整数时，内部二进制数值不变；转换成较短字节类型整数时，取低位字节二进制数值；转换成较长字节类型整数时，内部二进制数值进行扩展。若原来是无符号类型整数，则进行无符号扩展(高位字节0扩展)，若原来是有符号类型整数，则进行符号扩展。

## 表3-10不同类型整型转换时数值变化的例子

	原类型	原值	结果类型	结果值	表示值
(int)b	unsigned long	(FFFFFFFF) <sub>16</sub>	int	(FFFFFFFF) <sub>16</sub>	-1
(char)a	int	(0A00CD39) <sub>16</sub>	char	(39) <sub>16</sub>	'9'
(long int)ch	unsigned char	(F0) <sub>16</sub>	long int	(000000F0) <sub>16</sub>	240=+(F0) <sub>16</sub>
(unsigned int)c	short int	(FF00) <sub>16</sub>	unsigned int	(FFFFFFF00) <sub>16</sub>	4294967040= +(FFFFFFF00) <sub>16</sub>

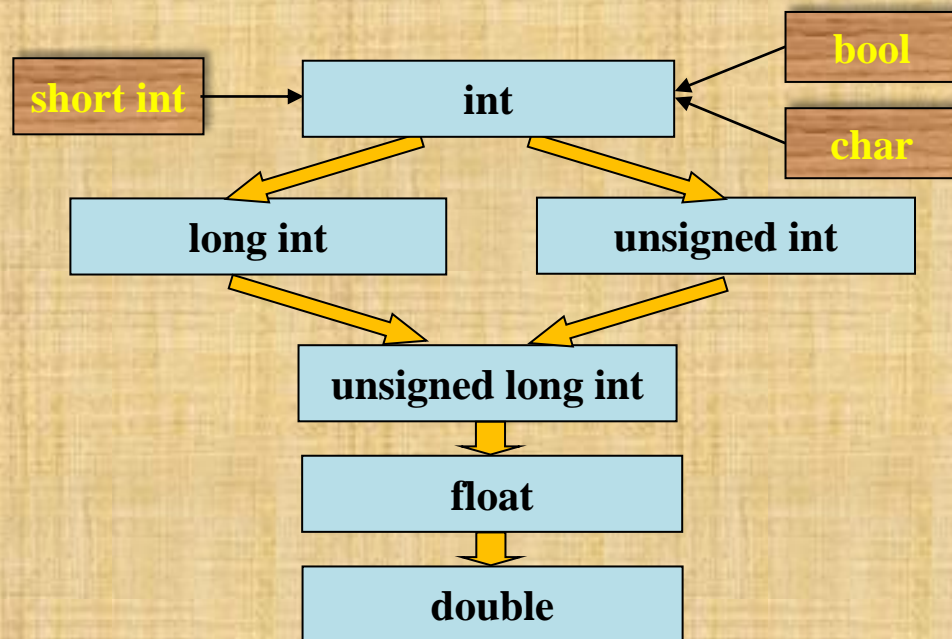
**注9：**若数据是无符号类型，则表示值就是内部二进制数(或十六进制数)转换成十进制数，如unsigned int型的值 (FFFFFFF00)<sub>16</sub>=4294967040；若数据是有符号类型，则表示值就是以内部十六进制数作为补码的数，如int型的值 (FFFFFFFF)<sub>16</sub>，最高位为1，表示负数，相反数的值为 (00000001)<sub>16</sub>=1，故表示值为-1。进制转换与补码见附录B。

### 赋值转换：

左值=表达式      等价于      左值=(左值类型)(表达式)

## 算术转换：

算术运算中两个不同类型数据要转换成统一类型的数据，然后才进行运算。算术转换规则如下图



注10： `short int`、`bool`、`char`类型数据参与算术运算时会首先自动转换成`int`，然后再参与运算；不同类型数据进行算术运算时按向下箭头方向首先自动转换成相同类型；`long int`和`unsigned int`混合运算时都自动转换成`unsigned long int`类型。



# 表3-11 几个混合类型算术表达式的自动类型转换

若定义 `int a=3; long b=4; float x=4.0f; double y=5.2; char ch='a'; short int c=10;`

表达式	数据类型	转换后的统一类型	表达式类型
<code>a/x</code>	<code>int / float</code>	<code>float</code>	<code>float</code>
<code>b+y</code>	<code>long + double</code>	<code>double</code>	<code>double</code>
<code>c*c</code>	<code>short * short</code>	<code>int</code>	<code>int</code>
<code>ch-32</code>	<code>char - int</code>	<code>int</code>	<code>int</code>
<code>ch- 'A'</code>	<code>char - char</code>	<code>int</code>	<code>int</code>

赋值转换: `int a=10,b=20; float x=1.0f; double y=1.0;`

`a=3; x=a;` 结果值: `3.0f` 类型: `float`

`b=x;` 结果值: `3` 类型: `int`

`y=a=x= -3.1416f;`

等价于 `y=( a= ( x= -3.1416f) ) );`

结果: `x=-3.1416f,a=-3,y=-3.0`

演示程序  
见后面

算术转换:

`'a'+0` 结果: `int`

`'a'- 'A'` 结果: `int`

`true+true` 结果: `int`

`3/7` 结果: `int`

`0x80000000u+1234` 结果: `unsigned int`

`1.0f/3` 结果: `float`

`1.2f+3*4.0` 结果: `double`



## 例B3.14 演示转换规则。

```
/******
```

```
* MB3_14.cpp
```

```
* 演示转换规则：显式转换、赋值转换、算术转换
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
void type(bool) { cout<<"bool"<<endl; }
```

```
void type(char) { cout<<"char"<<endl; }
```

```
void type(int) { cout<<"int"<<endl; }
```

```
void type(unsigned int) { cout<<"unsigned int"<<endl; }
```

```
void type(float) { cout<<"float"<<endl; }
```

```
void type(double) { cout<<"double"<<endl; }
```

```
int main( )
```

```
{
```

```
    // 显式转换
```

```
    int a=10,b=20; float x=1.0f; double y=1.0;
```

```
    type((int)x);    //float转换为int
```

```
    type(char(x+y));    //double转换为char
```

```
    type(double(a<b));    //bool转换为double
```

// 赋值转换

**a=3;**

**x=a;** //等价于 **x=(float)a;** int转换为x类型float, x得到转换后的值3.0f

**b=x;** //等价于 **b=(int)x;** float 转换为b类型int, b得到转换后的值3

**cout<<"a="<<a<<" , x="<<x<<" , b="<<b<<endl;**

**y=a=x= -3.1416f;** //等价于 **y= (a= (x=-3.1416f));**第一次赋值x得到-3.1416f;

//第二次赋值x的float型值转换为a类型int值-3, 赋值给a;

//第三次赋值a的int型值转换为y类型double值-3.0, 赋值给y

**cout<<"x="<<x<<" , a="<<a<<" , y="<<y<<endl;**

// 算术转换

**type( 'a'+0);** //char+int 转换后 int+int, 结果int

**type( 'a'-'A');** //char-char 转换后 int-int, 结果int

**type(true+true);** //bool+bool 转换后 int+int, 结果int

**type(3/7);** //int/int , 类型一致, 不转换, 结果int

**type(0x80000000u+1234);** //unsigned+int,转换unsigned+unsigned,结果unsigned

**type(1.0f/3);** //float/int 转换后 float/float, 结果float

**type(1.2f+3\*4.0);** //float+(int\*double): 乘法 int\*double转换成double\*double,

//中间结果double; 加法 float+double转换成double+double,结果double

**return 0;**

### 3.6.5 短路表达式

逻辑表达式(短路表达式): 一旦有结果就结束计算  
如

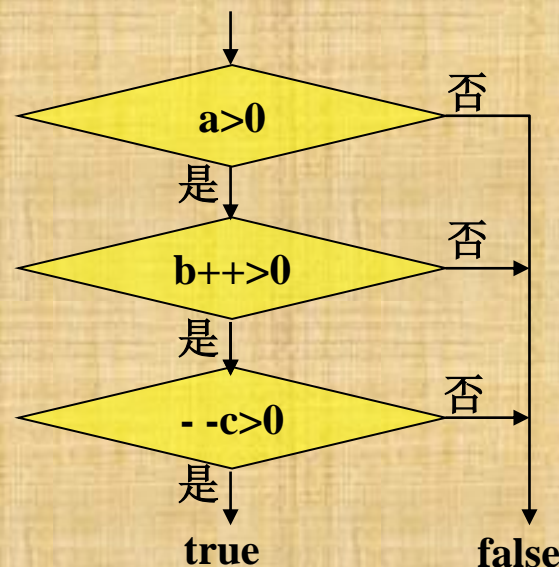
**$(a > 0) \ \&\& \ (b++ > 0) \ \&\& \ (--c > 0)$**

若 $a$ 值 $\leq 0$ 则结果一定为false, 提前结束计算, 从而 $b++$ 、 $--c$ 的计算并没有进行

若 $a$ 值 $> 0$ , 则还没有最终结果, 继续计算 $b++ > 0$ 。

若 $b$ 原值 $\leq 0$ , 则结果为false, 结束,  
 $--c$ 没有计算;

若 $b$ 原值 $> 0$ , 则继续计算 $--c > 0$ , 最后结果由 $--c > 0$ 确定。





## 例B3.15 演示逻辑运算的短路表达式。

```
/******
```

```
* MB3_15.cpp
```

```
* 演示逻辑运算的短路表达式
```

```
*****/
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{    int a , b , c ;  bool k;  //bool类型变量k用于记录条件表达式的值
    a=10 ; b=20 ; c=30 ;
    cout<<"a="<<a<<" ,b="<<b<<" ,c="<<c<<endl;
    k=(a>0)&&(b++>0)&&(--c>0) ; //即 k= ((a>0)&&(b++>0))&&(--c>0) ;
    cout<<"a="<<a<<" ,b="<<b<<" ,c="<<c<<" ,k="<<k<<endl;
    k=(a<0)&&(b++>0)&&(--c>0) ; //即k= ((a<0)&&(b++>0)) &&(--c>0) ;
    cout<<"a="<<a<<" ,b="<<b<<" ,c="<<c<<" ,k="<<k<<endl;
    return 0;
}
```

a=10,b=20,c=30

k=(a>0)&&(b++>0)&&(--c>0);

结果全算: a=10,b=21,c=29

a=10,b=21,c=29

k=(a<0)&&(b++>0)&&(--c>0);

提前结束: a=10,b=21,c=29



**注11：**当数值型量参与逻辑运算时，非0为true，0为false，故数值型量可以构成逻辑表达式，并且同样会提前终止。如果“`double a=0.01; int b=0,c=10;`”，则表达式“`a&&b++&&--c`”结果为false，a、c值不变，b值为1，因为b++值为0，表示false，计算终止，--c没有运行。类似分析，表达式“`!a&&b++&&--c`”的结果为false，a、b、c值都不变。

## 3.7 位运算

位运算就是二进制位上的逻辑运算和移位运算，  
逻辑运算的规则见 表3-12

表3-12逻辑的运算规则

逻辑运算	运算规则		特点
与×	$1 \times 1 = 1$ $0 \times 1 = 0$	$1 \times 0 = 0$ $0 \times 0 = 0$	操作数都为1，结果才为1，否则为0
或+	$1 + 1 = 1$ $0 + 1 = 1$	$1 + 0 = 1$ $0 + 0 = 0$	操作数都为0，结果才为0，否则为1
非'	$1' = 0$	$0' = 1$	非1即0，非0即1
异或⊕	$1 \oplus 1 = 0$ $0 \oplus 1 = 1$	$1 \oplus 0 = 1$ $0 \oplus 0 = 0$	操作数不同，结果为1，否则为0，即不进位相加

位运算符的操作数必须为整型：**char , short int , int**等。

若 signed char a=0xCF, b=0x6C ;  
位运算结果见表3-13

表3-13 位运算举例

$a = (11001111)_2$ $b = (01101100)_2$ $a \& b = (01001100)_2$	$a = (11001111)_2$ $b = (01101100)_2$ $a   b = (11101111)_2$	$a = (11001111)_2$ $b = (01101100)_2$ $a \wedge b = (10100011)_2$
$a = (11001111)_2$ $\sim a = (00110000)_2$	$a = (11001111)_2$ $a \ll 3 = (01111000)_2$	$a = (11001111)_2$ $a \gg 3 = (1111001)_2$

有符号类型  
左边补最高位数字

注1: 对于二进制数来说, 左移一位相当于乘以2, 右移一位相当于除以2(结果下取整)。变量在计算机内是以二进制数保存的, 所以左移和右移相当于乘以或除以2或2的乘幂。于是不管是否有符号, 左移运算时右边都应该补0。而右移运算时, 无符号数表示正数, 故左边移进0。如果是有符号数右移运算, 则右移后正数应该仍是正数, 负数仍是负数。故左边应该补最高位数字。



## 例B3.16 演示位运算的结果。

```
/******
```

```
*   MB3_16.cpp
```

```
*   演示位运算的结果
```

```
*****/
```

```
# include <iostream>
```

```
# include <iomanip>    //输出时用了格式控制setfill('0')和setw(2)，需加上此行
```

```
using namespace std;
```

```
int main( )
```

```
{    short int  a=0xFF, b=0xAAAA , cAnd, cOr, cXor , cNotb;
```

```
    //按位运算
```

```
    cAnd=a&b; cOr=a|b; cXor=a^b; cNotb= ~b;
```

```
    cout<<setfill( '0')<<hex ; //输出时填充字符设为'0'，并用十六进制输出整数
```

```
    cout<<"a="<<setw(4)<<a    //setw(4)表示输出至少4位,右对齐,左补填充字符'0'
```

```
        <<"\t b="<<setw(4)<<b<<endl;
```

```
    cout<<"a&b="<<setw(4)<<cAnd<<"\t a|b="<<setw(4)<<cOr<<endl;
```

```
    cout<<"a^b="<<setw(4)<<cXor<<"\t ~b="<<setw(4)<<cNotb<<endl;
```

```
    //移位运算
```

```
    short int bL,bR;    unsigned short int u=0xAAAA,uR;
```

```
    bL=b<<3; bR=b>>4; uR=u>>4;
```

```
    cout<<"b<<3="<<setw(4)<<bL<<"\t b>>4="<<setw(4)<<bR<<endl;
```

```
    cout<<"unsigned u="<<setw(4)<<u<<"\t u>>4="<<setw(4)<<uR<<endl;
```

```
    return 0; }
```



## 位运算的用处:

- & 用于将某些位设置成0或取某些位

$$\begin{array}{rcl} a & = & 00001111 \\ t & = & 11111101 \\ \hline a \& t & = & 00001101 \end{array}$$

$$\begin{array}{rcl} a & = & 00001111 \\ t & = & 00000010 \\ \hline a \& t & = & 00000010 \end{array}$$

- | 用于将某些位设置成1

$$\begin{array}{rcl} a & = & 00001111 \\ t & = & 01000000 \\ \hline a | t & = & 01001111 \end{array}$$

- ^ 用于将某些位取反

$$\begin{array}{rcl} a & = & 00001111 \\ t & = & 00111100 \\ \hline a \wedge t & = & 00110011 \end{array}$$

例C3.17 输入一个 int 型的数，输出该数的32位二进制数。

```
/******
```

```
* MC3_17.cpp
```

```
* 输入一个int型的数，输出该数的32位二进制数
```

```
*****/
```

```
# include <iostream>
```

```
using namespace std;
```

```
int main( )
```

```
{   int a;   unsigned int t; //t右移运算移进的数字是0
```

```
    cout<<"输入一个整数: ";
```

```
    cin>>a;
```

```
    cout<<hex<<"a的十六进制数是: "<<a<<", 32位二进制数是: ";
```

```
    for(t=0x80000000 ; t != 0 ; t=t>>1 ) //不断右移测试位，从 $(100...0)_2$ 到 $(00...01)_2$ 
```

```
    {   if( (a & t) != 0 ) cout<< 1; //取测试位后非零，表示该位数字为1，否则为0
```

```
        else      cout<< 0 ;
```

```
    }
```

```
    cout<<endl;
```

```
    return 0;
```

```
}
```

注2:  $(100...0)_2$ 右移后要成为 $(010...0)_2$ ，则数据必须是无符号的，否则会成为 $(110...0)_2$ 。