CS 1502H 程序设计思想与方法 (C++)
Principles and Methodology in Programming
Chapter 2. 顺序（2）

陈东尧
上海交通大学
2025年秋季

内容回顾

- 注释
- 变量的命名规则
- 数据类型
  - 整形、浮点型、字符型、布尔型
  - 其中，整形的机器码是如何记录数据的？

补码机制中的溢出问题

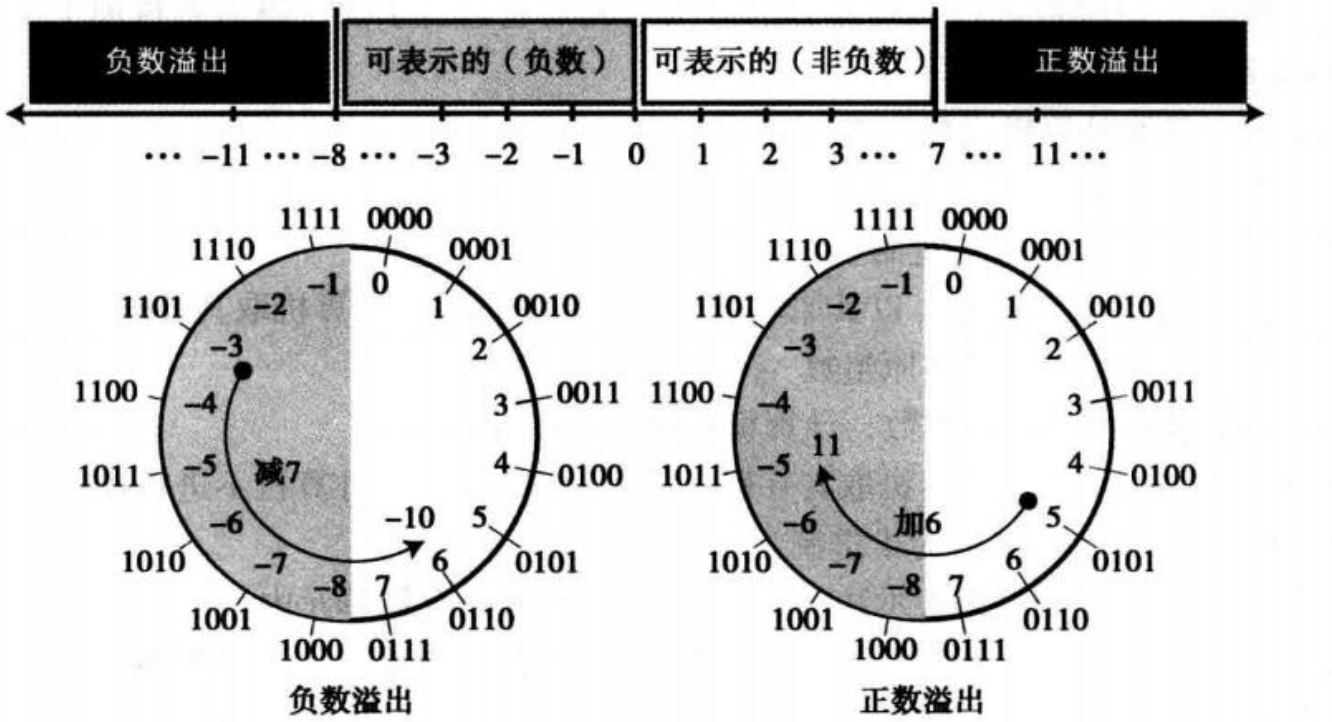


图 3-9　二进制补码表示法的溢出

数据类型-布尔（boolean）型

- 布尔型：标准C中没有布尔型数据，这是C++中新增的数据类型。占一个字节。它的值为：true，false
- 布尔型数据的内部表示：
  - true为1
  - false为0
- 布尔类型不能直接输入输出
- 布尔型数据可以进行算术、比较、逻辑运算

数据类型-枚举类型

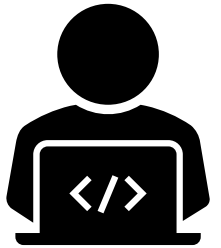
- 有时在设计程序时会用到一些特殊的对象，这些对象的取值范围是有限可数的。
  - 星期中的每一天
- 解决方法
  - 采用编码：假设0表示星期日，1表示星期一，…，6表示星期六。然后用一个整型变量如weekday表示这个对象。缺点是可读性差、健壮性弱
  - 符号常量：用#define功能将这些数字定义为符号常量，但可扩展性差
  - 定义一个新类型：枚举类型

定义枚举类型

- 格式：enum 枚举类型名 {元素表}；
- 定义一个表示一周中每天的名字的枚举类型：

  enum weekdayT { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
- 定义一个枚举类型的变量：

  weekdayT weekday；
- 枚举类型变量的操作：
  - 赋值： weekday = Friday；
  - 比较：Monday < Friday 比较这两个值的内部表示
- 枚举类型不能直接输入输出



代码演示

chapter2-enum-operation.cpp

枚举类型的内部表示

- 在内部，枚举类型采用编码表示。当定义

  enum weekdayT { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}; 时，默认用0代表Sunday，1代表Monday，…，6表示Saturday

- C++语言的编译器也允许明确指出枚举类型的元素的内部表示。例如，希望从1而不是0开始编号，可以这样定义

  enum weekdayT { Sunday=1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

- 也可以从中间某一个开始重新指定，如

  enum weekdayT { Sunday, Monday, Tuesday=5, Wednesday, Thursday, Friday, Saturday};

枚举类型

- 自定义枚举类型的好处：
  - 编译器自动编码，解放程序员工作量
  - 变量可以用有意义的类型名，提高程序的可读性
  - 为编译器提供信息（例如取值范围或允许的操作），程序的调试更容易
- 注意：虽然枚举型变量中存储的是一个整数，但不能将一个整数赋给一个自定义的枚举型变量，否则会出现编译错误。
- 布尔型就是C++事先定义好的一个枚举类型。

数据类型小结

- 理解数据类型的作用，主要用于规定：
  - 占用空间
  - 取值范围
  - 支持运算
- 可以直接用cin和cout对象输入输出的类型
  - 整型
  - 实型
  - 字符型
- 不能直接用cin和cout对象输入输出的类型
  - 布尔型：true显示1，false显示0
  - 枚举类型：显示编码值

本讲框架

- 变量与常量
  - 字符常量、别名、转义字符、符号常量
- 输入与输出
  - cin和cout
- 算术和赋值计算
  - 数学函数库（cmath）
  - 变量赋值
  - 多重赋值与赋值嵌套
  - 自增自减运算符
  - 数据类型转换

类型别名（Type alias）

- 可以给一个类型重新命名
  - 格式：typedef 原名 现名
  - 例子：

```
typedef  int  INTEGER
typedef  double  REAL

INTEGER n1 = 10;
```

- 类型或变量占用空间
  - sizeof运算符

【扩展阅读，sizeof】https://en.cppreference.com/w/cpp/language/sizeof

字符常量

- 字符常量

  ‘a’, ‘S’, ‘2’等用一对单引号括起来的数据称为字符常量。

- C++中的单引号和双引号有不同的用处。
  - 单引号括起来的是一个字符
  - 双引号括起来的是字符串
- 如何输出一些非打印字符？
  - 比如”、’、等等

转义字符（Escape character）

- 转义序列：用可打印的字符来表示非打印字符，反斜杠符号¥称为转义字符

换行符写为'＼n'，虽然它由两个字符＼和n来描述，但它表示一个ASCII字符。

- 双引号和单引号的转义

  如果在一个串中把双引号”用作一个字符，必须要对它转义，否则它会终结该字符串。

  cout << "¥"abc¥""; 输出"abc"

  双引号的单字符表示：可以写'” '，也可以写'¥"'。

- 许多特殊字符没有转义序列，可直接使用它们的内部编码（8进制）：

  例如，警告字符¥a 可以表示成:'¥007'，'¥07'，'¥7'

常用的转义字符（Escape character）

字符形式	含义
\n	换行
\t	水平制表
\b	退一格
\r	回车
\f	换页
\\	\
\’	‘
\”	“
\ddd	1到3位八进制数代表的字符
\xhh	1到2位十六进制数代表的字符

转义字符（**ESC**ape Character）

USASCII code chart

Row	b_7 b_6 b_5 = 0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1	0	1	2	3	4	5	6	7
0 0 0 0	0 NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1 SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2 STX	DC2	"	2	B	R	b	r
0 0 1 1	3 ETX	DC3	#	3	C	S	c	s
0 1 0 0	4 EOT	DC4	$	4	D	T	d	t
0 1 0 1	5 ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6 ACK	SYN	&	6	F	V	f	v
0 1 1 1	7 BEL	ETB	'	7	G	W	g	w
1 0 0 0	8 BS	CAN	(	8	H	X	h	x
1 0 0 1	9 HT	EM	)	9	I	Y	i	y
1 0 1 0	10 LF	SUB	*	:	J	Z	j	z
1 0 1 1	11 VT	ESC	+	;	K	[	k	{
1 1 0 0	12 FF	FS	,	<	L	\	l	|
1 1 0 1	13 CR	GS	-	=	M	]	m	}
1 1 1 0	14 SO	RS	.	>	N	^	n	~
1 1 1 1	15 SI	US	/	?	O	_	o	DEL

- C/C++中的¥
- URL中的%：例子：复制粘贴 https://baike.baidu.com/item/交通大学 你会得到什么？
  - https://baike.baidu.com/item/%E4%BA%A4%E9%80%9A%E5%A4%A7%E5%AD%A6/
- Latex中也通过¥来转义
  - 例如¥figure, ¥section, 和¥abstract 等等

符号常量（Symbolic constant）

- 对于一些有特殊意义或在程序出现多次的常量，可以给它们取一个名字，有名字的常量称为符号常量
- C语言风格用#define定义符号常量（也称为宏），格式：
  - #define 符号常量名 字符串，例如 #define PI 3.14159
- #define风格存在问题
  - 所定的替义的符号常量无法进行类型检查
  - #define的处理只是简单的字符串换，可能会引起一些意想不到的错误
- C++建议用const定义符号常量，且只能在定义时赋初值

  const <类型名> <常量名> = <值>;

  如：const double PI = 3.1415926;

符号常量的注意事项与好处

- 注意事项：
  - const定义的符号常量只能在定义时被赋初值。
  - 符号常量名的命名规范与变量相同。
- 好处：
  - 含义清楚，提高了程序的可读性。
  - 在需要改变一个常量时能做到“一改全改”。

本讲框架

- 变量与常量
  - 字符常量、别名、转义字符、符号常量
- 输入与输出
  - cin和cout
- 算术和赋值计算
  - 数学函数库（cmath）
  - 变量赋值
  - 多重赋值与赋值嵌套
  - 自增自减运算符
  - 数据类型转换

输入流对象cin

- 输入流（input stream）对象（object）
- 键盘流入的数据流，将键盘输入的数据存入变量
- 流提取运算符：>>
- 格式：
  - cin >> 变量
  - cin >>变量1 >> 变量2 >> … >> 变量n

用户的响应

- 当程序执行到这个语句时会停下来等待用户的输入
- 用户可以输入数据，用回车（↙）结束。
- 当有多个输入数据时，一般用空白字符（**空格、制表符和回车**）分隔。
- 如：a为整型，d为double，则对应于

  cin >> a >>d，用户的输入可以为

  12 13.2↙

  12（tab键）13.2↙

  12↙13.2↙

cin.get

- 作用：从键盘接收一个字符，可以是空白字符
- 用法：从键盘输入一个字符并存放到变量ch中

```
cin.get(ch);
ch= cin.get();
```

注意： cin.get()可以接收任意的字符，包括空白字符。

cin.get

如a, b, c为字符型变量，对应语句

```
a = cin.get();
b = cin.get();
c = cin.get();
```

如果输入a b c↙

则a的值是’a’，b的值是空格，c的值是’b’

如果将这个输入用于语句：

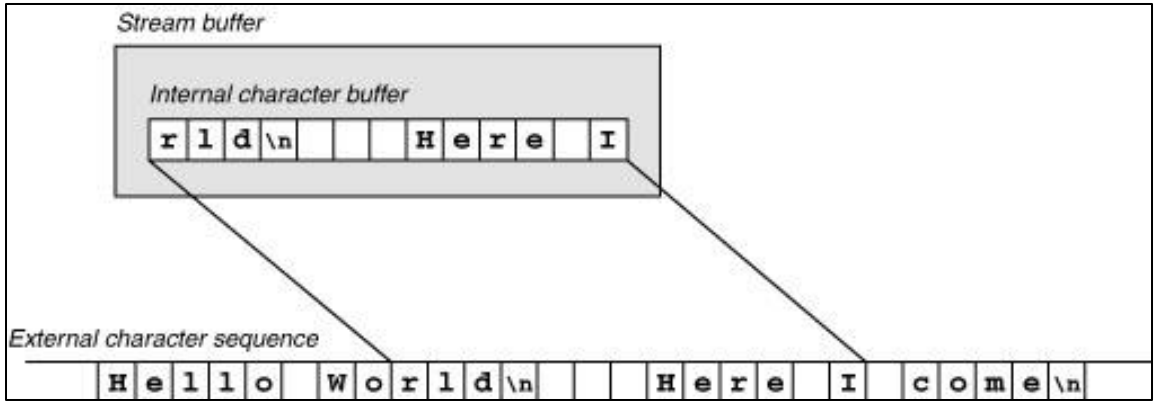
```
cin >> a >> b >> c ;
```

那么变量a、b、c的内容分别为’a’、’b’、’c’，
因为空格被作为输入值之间的分隔符。

输出流对象cout

- 将变量或表达式的内容显示在显示器上
- 流插入运算符：<<
- 格式
  - 输出一个变量的值：cout << a；
  - 输出多个变量的值：cout << a << b << c；
  - 输出表达式的结果：cout << “Hello world”;
  - 上述情况的组合：

    cout << a << ‘+’ << b << ‘=’ << a+b << endl;



本讲框架

- 变量与常量
  - 字符常量、别名、转义字符、符号常量
- 输入与输出
  - cin和cout
- 算术和赋值计算
  - 数学函数库（cmath）
  - 变量赋值
  - 多重赋值与赋值嵌套
  - 自增自减运算符
  - 数据类型转换

算术和赋值运算

- 算术表达式是完成计算的工具。
- 算术表达式由运算符和运算对象组成
- 算术运算符包括：+　-　*　/　%
- 除“-”外，所有的算术运算符都是二元运算符。“-”可为二元运算，也可为一元运算
- 优先级：高　*　/　%，低　+　-
- 结合性：左结合　//有没有右结合的运算？有=
- 运算对象可以是整型、浮点型、字符型和布尔型

除法

/	Division（除法）	Divides one value by another	x / y
%	Modulus（取模，即除法运算的余数）	Returns the division remainder	x % y

```
#include <iostream>
using namespace std;

int main() {
	int x = 3, y = 6, z = 10;
	cout<< "6 / 3 is " << y / x << endl
	<< "10 % 3 is " << z % x << endl
	<< "10 / 3 is " << (float) z / x << endl;
	return 0;
}
```

各种类型数据的混合运算

- C++只能执行同类型的数据的运算，而且运算结果的类型与运算数类型相同
- 在进行不同类型的数据运算之前，需将运算数转为同一类型
- 转换总原则：
  - 非标准转换成标准
  - 占用空间少的转换成占用空间多的
  - 数值范围小的转换成数值范围大

数学函数库

- 在C++语言中，其他的数学运算都是通过函数的形式来实现。所有的数学函数都在cmath中。
- 要使用这些数学函数，必须在程序头上写上编译预处理命令：

```
#include <cmath>
```

cmath的主要内容

绝对值函数	int abs(int x) ;　double fabs(double x)
e^x	double exp(double x)
x^y	double pow(double x, double y)
平方根	double sqrt(double x)
ln x	double log(double x)
$\log_{10}x$	double log10(double x)
三角函数	double sin(double x) double cos(double x) double tan(double x)
反三角函数	double asin(double x) double acos(double x) double atan(double x)

变量赋值

- 变量赋值是通过赋值表达式实现，赋值表达式
  - 格式：<变量> = <表达式>
  - 作用：将右边的表达式的值存入左边的变量，整个表达式的值是右边的表达式的结果。
  - 注意：x = x + 2是正确的表达式
  - 赋值运算符是右结合的
- 左值(lvalue)：在C++中，能出现在赋值运算符左边的表达式称为左值，左值必须有一块可以存储信息的内存空间

多重赋值与赋值嵌套

- 当用到多重赋值时，要保证所有的变量都是同类型的，以避免在自动类型转换时出现与预期不相符的结果的可能性。
  - 如变量d定义为double,变量i定义为int，语句：

    d = i = 1.5;

  结果是：i等于1，d等于1.0
- 赋值嵌套：将赋值表达式作为更大的表达式的一部分 。如：a = (x = 6) + (y = 7) 等价于?
  - 分别将x 和 y 的值设为6 和 7，并将6和7相加，结果存于变量a。
- 赋值运算符=的优先级比算术运算符低

复合赋值运算

- 其他运算符与赋值运算符结合的运算符称为复合赋值运算符
- 常用的复合赋值运算符有：+=， -=， *=， /=， %=
- 变量 op = 表达式;

  等价于：变量 = 变量 op 表达式;
- 如： balance += deposit; // balance = balance + deposit;

```
balance -= surcharge;
x /= 10;
salary *=2;
```

自增、自减运算符

- 自增、自减运算符：++，- -

  相当于+=1和-=1, 它有前缀和后缀两种用法

  前缀：先运算再赋值，后缀：先完成赋值再运算

  ++k，k++，- -k，k- -，

  但含义有所不同。如：i=3

  j=i++ ⟶ i=4 j=3

  j=++i ⟶ i=4 j=4

  j=i-- ⟶ i=2 j=3

  j=--i ⟶ i=2 j=2

自增自减运算符示例

```
#include <iostream>
using namespace std;

int main(){
int a = 20;
int b = 20;

cout<<"a="<<a<<"; b="<<b<<'\n';
cout<<"a++="<<a++<<"; ++b="<<++b<<'\n';
cout<<"a="<<a<<"; b="<<b<<'\n';

return 0;
}
```

```
a=20; b=20
a++=20; ++b=21
a=21; b=21
```

赋值时的自动类型转换

- 当表达式的结果类型和变量类型不一致时，系统会将右边的表达式的结果转换成左边的变量的类型，再赋给左边的变量。

数据类型转换：规则

- 类型转换规则
  - bool、char和short这些非标准的整数在运算前都必须转换为int。
  - int和float运算时，将int转换成float。
  - int和long运算时，将int转换成long。
  - int和double运算时，将int转换成double。
  - float和double运算时，将float转换成double。
- 数据转换规则
  - 实型转换成整型时，舍弃小数部分
  - 整型转换成实型时，数值不变，但表示成实数形式
  - 字符型转换成整型时，不同编译器有不同处理方法（无符号或有符号数）
  - 布尔型转换成整型时，true为1，false为0
  - 整型转换成字符型时，直接取整型数据的最低8位

【扩展阅读：C++中的类型转换】https://www.geeksforgeeks.org/type-conversion-in-c/

具体转换规则

- All the data types of the variables are upgraded to the data type of the variable with largest data type.

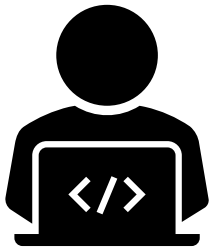
```
bool -> char -> short int -> int ->

unsigned int -> long -> unsigned ->

long long -> float -> double -> long double
```

【扩展阅读：C++中的类型转换】https://www.geeksforgeeks.org/type-conversion-in-c/

自动类型转换示例



代码演示

chapter2-conversion.cpp

```
#include <iostream>
using namespace std;

int main()
{
int x = 10; // integer x
char y = 'a'; // character c

// Note the ASCII value of 'a' is 97
x = x + y;

float z = x + 1.0;

cout << "x = " << x << endl
<< "y = " << y << endl
<< "z = " << z << endl;

return 0;
}
```

```
x = 107
y = a
z = 108
```

注意此处z为float类型。但在屏幕中展示时不会有".0"

强制类型转换

- 赋值和算术运算时会执行自动类型（automatic type conversion）转换
- 如要想使4/5的结果是0.8，而不是0，该怎么办？
  - 可以将其中一个写成浮点数。例如：4.0 / 5或4 / 5.0
- 如果不修改变量，该怎么办？
  - 答案是：用强制类型转换
- 强制类型转换格式：

  （类型名）（表达式）或 类型名 （表达式）
- 例如，要想使两个整型变量x和y出的结果为double型，可以用下列语句

double z; z = (double)x / y;

转换类型

- 强制类型转换很有可能是导致错误的原因
  - 除非你真的知道你在做什么
- 为了方便查找这些错误，C++提供了在强制类型转换时指明转换的性质
- 转换的性质有四种：
  - 静态转换(static_cast)：用于编译器隐式执行类型转换
  - 重解释转换(reinterpret_cast)
  - 常量转换(const_cast)
  - 动态转换(dynamic_cast)
- C++风格的类型转换格式：

  转换类型<类型名>(表达式)

  z = static_cast<double>(x) / y;

【扩展阅读：C++中的类型转换】https://www.jianshu.com/p/5cb9800b6697

类型转换示例

- 自动类型转换

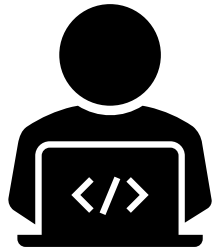
```
double num;
num = 9 / 4.0;
num = 9.0 / 4;
num = 9.0 / 4.0;
```

- 强制类型转换

```
double num = 9 / 4;
double num = (double)9 / 4;     //或者 9 / double(4)
```

- 静态转换

```
char ch;       int d = 65;
ch = static_cast<char> (d);
```



代码演示

chapter2-static-cast.cpp

C++类型的转换支持编译器检查。C风格的强制类型转换不在编译过程中查错，因此如果有错误则会在运行过程中出错。
C++ style casts are checked by the compiler. C style casts aren't and can fail at runtime.

总结

- 本章介绍了C++程序的总体结构及工作方式 。具体包括：
  - 用变量记录数据
  - 读取用户提供的数值数据；
  - 在屏幕上显示文本和数据；
  - 对现有数据进行算术运算得到新的结果

附：保留n位小数的设置

1. 利用头文件<iomanip>和setprecision()函数
    1. 可以把数字都保留n位小数，如：cout<<fixed<<setprecision(n)
2. 利用C语言中的printf()函数，功能如C++中的cout函数，如 double num = 1.203; printf(“%.2lf”, num); 该语句可以保留2位小数

参考链接：https://blog.csdn.net/qq_36667170/article/details/79265224

附：运算符优先级

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	后置++	后置自增运算符	变量名++	左到右	
	后置--	后置自减运算符	变量名--		
	[]	数组下标	数组名[整型表达式]		
	( )	圆括号	（表达式）/函数名(形参表)		
	.	成员选择（对象）	对象.成员名		
	->	成员选择（指针）	对象指针->成员名		
2	-	负号运算符	-表达式	右到左	单目运算符
	(类型)	强制类型转换	(数据类型)表达式		
	前置++	前置自增运算符	++变量名		单目运算符
	前置--	前置自减运算符	--变量名		单目运算符
	*	取值运算符	*指针表达式		单目运算符
	&	取地址运算符	&左值表达式		单目运算符
	!	逻辑非运算符	!表达式		单目运算符
	~	按位取反运算符	~表达式		单目运算符
	sizeof	长度运算符	sizeof 表达式/sizeof(类型)		

附：运算符优先级

3	/	除	表达式/表达式	左到右	双目运算符
	*	乘	表达式*表达式		双目运算符
	%	余数（取模）	整型表达式%整型表达式		双目运算符
4	+	加	表达式+表达式	左到右	双目运算符
	-	减	表达式-表达式		双目运算符
5	<<	左移	表达式<<表达式	左到右	双目运算符
	>>	右移	表达式>>表达式		双目运算符
6	>	大于	表达式>表达式	左到右	双目运算符
	>=	大于等于	表达式>=表达式		双目运算符
	<	小于	表达式<表达式		双目运算符
	<=	小于等于	表达式<=表达式		双目运算符
7	==	等于	表达式==表达式	左到右	双目运算符
	!=	不等于	表达式!= 表达式		双目运算符
8	&	按位与	整型表达式&整型表达式	左到右	双目运算符
9	^	按位异或	整型表达式^整型表达式	左到右	双目运算符
10	|	按位或	整型表达式|整型表达式	左到右	双目运算符

附：运算符优先级

11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12	||	逻辑或	表达式||表达式	左到右	双目运算符
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左	三目运算符
14	=	赋值运算符	变量=表达式	右到左	
	/=	除后赋值	变量/=表达式		
	*=	乘后赋值	变量*=表达式		
	%=	取模后赋值	变量%=表达式		
	+=	加后赋值	变量+=表达式		
	-=	减后赋值	变量-=表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&=	按位与后赋值	变量&=表达式		
	^=	按位异或后赋值	变量^=表达式		
	|=	按位或后赋值	变量|=表达式		
15	,	逗号运算符	表达式,表达式,...	左到右	从左向右顺序运算