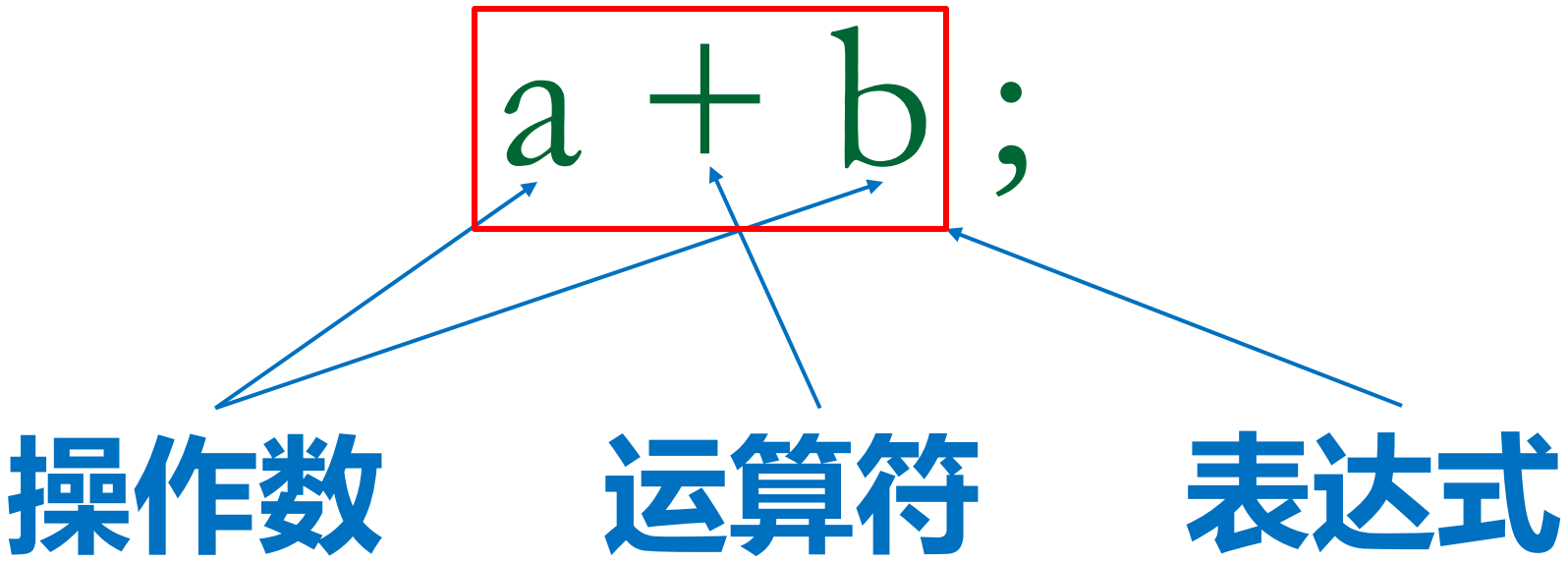
高级语言C++程序设计
Lecture 3 运算符与表达式

李雨森
南开大学 计算机学院
2021

基本概念



赋值运算符

一般赋值运算符： =

```
#include<iostream>
using namespace std;
int main(){
    int x = 3, y = 5;
    int z;
    z = (x + y);
    z = (x = y);
    return 0;
}
```

- 左操作数一般是与右操作数类型相同的变量
- 右操作数可以是常量/变量，也可以是表达式

规则：赋值运算结束后，右操作数的值赋给左操作数，该值同时也是赋值表达式的值

赋值运算符

一般赋值运算符： =

```
#include<iostream>
using namespace std;
int main(){
    int x = 3, y = 5;
    int z;
    z = (x + y);
    z = (x = y);
    return 0;
}
```

- 左操作数一般是与右操作数类型相同的变量
- 右操作数可以是常量/变量，也可以是表达式

先执行x+y，结果赋值给z，z变为8，同时8也是z=(x+y)这个赋值表达式的值

赋值运算符

一般赋值运算符： =

```
#include<iostream>
using namespace std;
int main(){
    int x = 3, y = 5;
    int z;
    z = (x + y);
    z = (x = y);
    return 0;
}
```

- 左操作数一般是与右操作数类型相同的变量
- 右操作数可以是常量/变量，也可以是表达式

先执行x=y，将y的值赋给x，x变为5，同时5也是x=y这个的表达式的值，再将x=y这个表达式的值赋给z，z变为5，同时5也是z=(x=y)这个赋值表达式的值

赋值运算符

复合类型赋值运算符： +=, -=, *=, /=

```
#include<iostream>
using namespace std;
int main() {
    int x = 3, y = 5;
    x += y;
    x -= y;
    cout<<x<<y;
    return 0;
}
```

x += y 等价于 x = x + y;
x -= y 等价于 x = x - y;
x *= y 等价于 x = x*y;
x /= y 等价于 x = x/y;

赋值运算符

复合类型赋值运算符： +=, -=, *=, /=

```
#include<iostream>
using namespace std;
int main() {
    int x = 3, y = 5;
    x *= y + 2;
    x /= y + 2;
    cout<<x<<y;
    return 0;
}
```

C++认为复合赋值运算符的右操作数是一个整体，可以理解为自动地为右操作数加上了括号，即 x*=y+2 等价于 x = x*(y+2)

左值与右值

左值：可以放在赋值符号的左边，通常可以修改

```
#include<iostream>
using namespace std;
int main() {
    int a = 3, b = 5;
    a = a - 1; //OK
    0 = 1; //错误!
    100 = a; //错误!
    a + b = 1; //错误!
    return 0;
}
```

变量可以作为左值，因为变量可以更改

左值与右值

左值：可以放在赋值符号的左边，通常可以修改

```
#include<iostream>
using namespace std;
int main() {
    int a = 3, b = 5;
    a = a - 1; //OK
    0 = 1; //错误!
    100 = a; //错误!
    a + b = 1; //错误!
    return 0;
}
```

常量不能作为左值，因为常量不能更改

左值与右值

左值：可以放在赋值符号的左边，通常可以修改

```
#include<iostream>
using namespace std;
int main() {
    int a = 3, b = 5;
    a = a - 1; //OK
    0 = 1; //错误!
    100 = a; //错误!
    a + b = 1; //错误!
    return 0;
}
```

表达式不能作为左值，因为表达式的结果不能更改

左值与右值

右值：不能放在赋值号左边的值，通常不能修改

```
#include<iostream>
using namespace std;
int main() {
    int a = 3, b = 5;
    a = a - 1; //OK
    0 = 1; //错误!
    100 = a; //错误!
    a + b = 1; //错误!
    return 0;
}
```

表达式不能作为左值，因此是右值

左值与右值

右值：不能放在赋值号左边的值，通常不能修改

```
#include<iostream>
using namespace std;
int main() {
    int a = 3, b = 5;
    a = a - 1; //OK
    0 = 1; //错误!
    100 = a; //错误!
    a + b = 1; //错误!
    return 0;
}
```

常数不能作为左值，因此是右值

算数运算符

双目运算符（两个操作数）： + - * / %

```
#include<iostream>
using namespace std;
int main() {
    int x = 3, y = 5;
    cout<<x+y; //加法
    cout<<x-y; //减法
    cout<<x*y; //乘法
    cout<<x/y; //除法
    cout<<x%y; //取模
    return 0;
}
```

- 取模运算%
  - 操作数都为整数
  - 右运算分量不为0
  - 结果是余数

算数运算符

单目运算符（1个操作数）

```
#include<iostream>
using namespace std;
int main() {
    int x = 3, y = 5;
    cout<<-x; //负号
    cout<<+y; //正号
    return 0;
}
```

+ (正号)

- (负号)

算数运算符

单目运算符（1个操作数）

```
int main() {
    int x = 3
    cout<<++x;
    cout<<x;
    cout<<x++;
    cout<<x;
    return 0;
}
```

增量运算符： ++

- 表达式 ++x
  - x的值加1，表达式(++x)的值是x加之后的值
- 表达式 x++
  - x的值加1，表达式(x++)的值是x加之前的值

程序输出：4 4 4 5

算数运算符

单目运算符（1个操作数）

```
int main() {
    int x = 3
    cout<<--x;
    cout<<x;
    cout<<x--;
    cout<<x;
    return 0;
}
```

减量运算符： --

- 表达式 --x
  - x的值减1，表达式(--x)的值是x减1之后的值
- 表达式 x--
  - x的值减1，表达式(x--)的值是x减1之前的值

程序输出：2 2 2 1

算数运算中的类型转换

- 算术运算分量的数据类型可以不同
  - 整数、浮点数、字符
- 运算分量必须转换为同一类型才能够进行运算
  - 占空间少的类型自动转换为占空间多的类型
  - 带符号数转换为无符号数（例如，int转为unsigned int）

10 + ‘a’ + 1.5 – 2.25* ‘b’

（1）先将字符b转换为double类型的98，然后计算2.25*98，得到200.5

（2）将字符a转换为int类型的97，计算10+97，结果为107

（3）将107转换为double类型，然后加上1.5，得到108.5

（4）计算108.5-200.5得到结果-92，是double型浮点数

关系运算符

关系运算符：<，>，==，>=，<=，!=

```
int main() {
    int x=2, y=1;
    bool z;
    z = (x < y);
    cout<<z;
    z = (x > y);
    cout<<z;
    return 0;
}
```

程序输出：0 1

- 关系运算也称为比较运算，对相同类型数据进行关系运算，结果为布尔类型（1或者0，分别代表true或者false）

小于号<：比较x是否小于y，是返回1，否则返回0

大于号>：比较x是否大于y，是返回1,否则返回0

关系运算符

关系运算符：<，>，==，>=，<=，!=

```
int main() {
    int x=2, y=2;
    bool z;
    z = (x <= y);
    cout<<z;
    z = (x >= y);
    cout<<z;
    return 0;
}
```

程序输出：1 1

小于等于号<=：比较x是否小于等于y，是返回1，否则返回0

大于等于号>=：比较x是否大于等于y，是返回1，否则返回0

关系运算符

关系运算符：<，>，==，>=，<=，!=

```
int main() {
    int x=2, y=3;
    bool z;
    z = (x == y);
    cout<<z;
    z = (x ! = y);
    cout<<z;
    return 0;
}
```

等于号==：比较x是否等于y，等于返回1，不等返回0

不等号!=：比较x是否等于y，不等返回1，等于返回0

程序输出：0 1

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = x && y;
    z = (x && y) && z;
    z = (x>y)&&(3>2);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑与 &&：两个操作数都是true，结果为true，否则结果为false

x为true，y为false，结果为false

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = x && y;
    z = (x && y) && z;
    z = (x>y)&&(3>2);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑与 &&：两个操作数都是true，结果为true，否则结果为false

(x && y)的结果为false, z为false, 结果为false

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = x && y;
    z = (x && y) && z;
    z = (x>y)&&(3>2);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑与 &&：两个操作数都是true，结果为true，否则结果为false

(x > y)的结果为true, (3>2)的结果为true, 结果为true

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = x || y;
    z = (x && y) || z;
    z = (x>y)||(3>2);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑或 ||：两个操作数都是false，结果为false，否则结果为true

x为true，y为false，结果为true

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = x || y;
    z = (x && y) || z;
    z = (x>y)||(3>2);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑或 ||：两个操作数都是false，结果为false，否则结果为true

(x&&y)的结果为false，z的结果为true，结果为true

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = x || y;
    z = (x && y) || z;
    z = (x>y)||(3>2);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑或 ||：两个操作数都是false，结果为false，否则结果为true

(x>y)的结果为true，(3>2)的结果为true，结果为true

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = !x;
    z = !(x && y);
    z = (!x)||(!y);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑非！：只有1个操作数，如果操作的值是false，结果为true，否则结果为false

x的值为true，结果为false

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = !x;
    z = !(x && y);
    z = (!x)||(!y);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑非！：只有1个操作数，如果操作的值是false，结果为true，否则结果为false

(x && y)的值为false，结果为true

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    bool x = true;
    bool y = false;
    bool z;
    z = !x;
    z = !(x && y);
    z = (!x)||(!y);
    return 0;
}
```

- 逻辑运算的操作数通常是bool类型的变量或表达式，运算结果为布尔类型

逻辑非！：只有1个操作数，如果操作的值是false，结果为true，否则结果为false

!x的值为false，!y的值为true，结果为true

逻辑运算符

逻辑运算符：&&，||，！

```
int main() {
    int x = 2;
    int y = 0;
    bool z;
    z = x && y;
    z = !(x && y);
    z = (!x)||(!y);
    return 0;
}
```

- 逻辑运算的操作数也可以不是bool类型，非0值看作true，0值看作false

x的值非0，看作true，y的值为0，看作false,结果为false

(x&&y)的结果为false，结果为true

(!x)的值为false，(!y)的值为true，结果为true

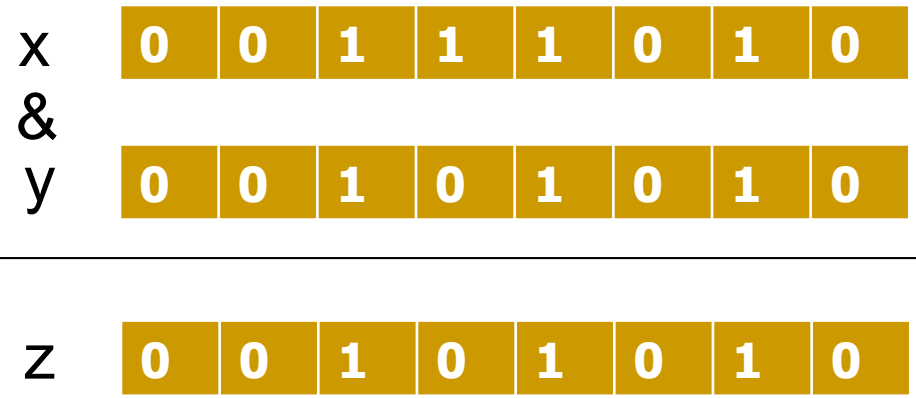
位运算

位运算是一种对操作数按二进制位进行操作的运算（作用于操作数的每一个二进制位上）。位运算的运算对象只能是整型数据(包括字符型), 且运算结果仍为整型数据

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

int main() {
   char x = 58,y = 42;
   char z = x&y;
   cout<<z<<endl;
   return 0;
}
```

按位与 &:
两个操作数的每一位进行与操作



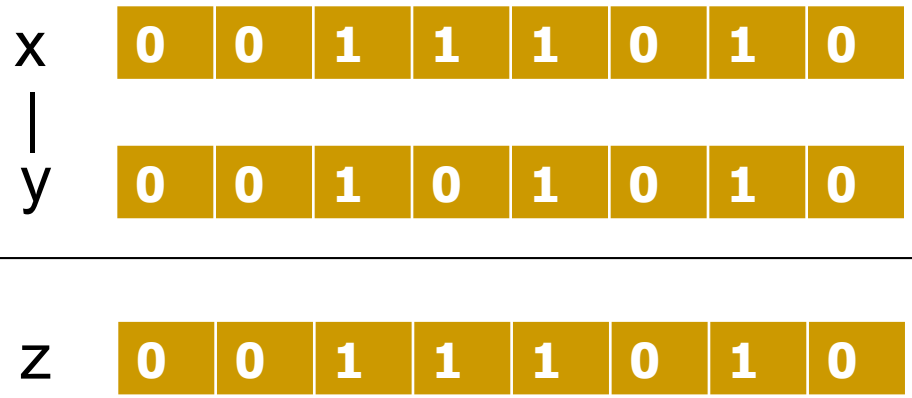
位运算

位运算是一种对操作数按二进制位进行操作的运算（作用于操作数的每一个二进制位上）。位运算的运算对象只能是整型数据(包括字符型), 且运算结果仍为整型数据

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

int main() {
   char x = 58,y = 42;
   char z = x|y;
   cout<<z<<endl;
   return 0;
}
```

按位或 |:
两个操作数的每一位进行或操作



位运算

位运算是一种对操作数按二进制位进行操作的运算（作用于操作数的每一个二进制位上）。位运算的运算对象只能是整型数据(包括字符型), 且运算结果仍为整型数据

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

int main() {
   char x = 58,y = 42;
   char z = x^y;
   cout<<z<<endl;
   return 0;
}
```

按位异或 ^:
两个操作数的每一位进行异或操作（相同得0，不同得1）

x	0	0	1	1	1	0	1	0
^								
y	0	0	1	0	1	0	1	0
z	0	0	0	1	0	0	0	0

位运算

位运算是一种对操作数按二进制位进行操作的运算（作用于操作数的每一个二进制位上）。位运算的运算对象只能是整型数据(包括字符型), 且运算结果仍为整型数据

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

int main() {
   char x = 58;
   char z = ~x;
   cout<<z<<endl;
   return 0;
}
```

按位取反~:
操作数的每一位都取反

x

0	0	1	1	1	0	1	0

~

z

1	1	0	0	0	1	0	1

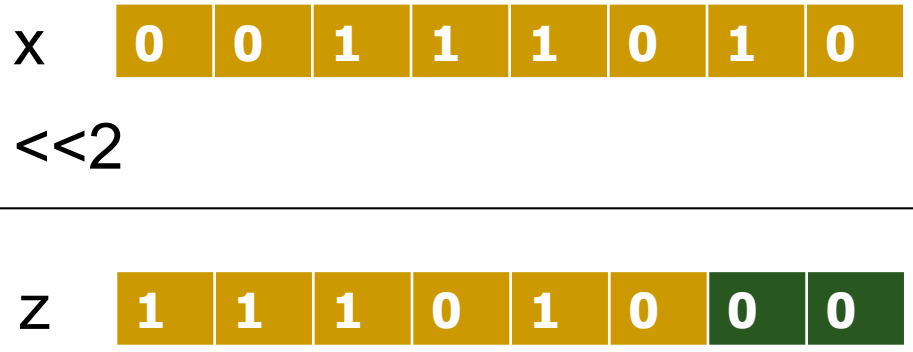
位运算

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

int main() {
   unsigned char x;
   x = 58;
   unsigned char z;
   z = x << 2;
   z = x << 4;
   return 0;
}
```

向左移位 <<:
按二进制每一位向左移动相应的位数，高位移出（舍弃），低位的空位补0

如果x是无符号数，左移N位相当于乘以2^N



注意x<<2执行后，x本身的值并没有变！

位运算

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

int main() {
   unsigned char x;
   x = 58;
   unsigned char z;
   z = x << 2;
   z = x << 4;
   return 0;
}
```

向左移位 <<:
按二进制每一位向左移动相应的位数，高位移出（舍弃），低位的空位补0

如果x是无符号数，左移N位相当于乘以2^N

x

0	0	1	1	1	0	1	0

<<4

z

1	0	1	0	0	0	0	0

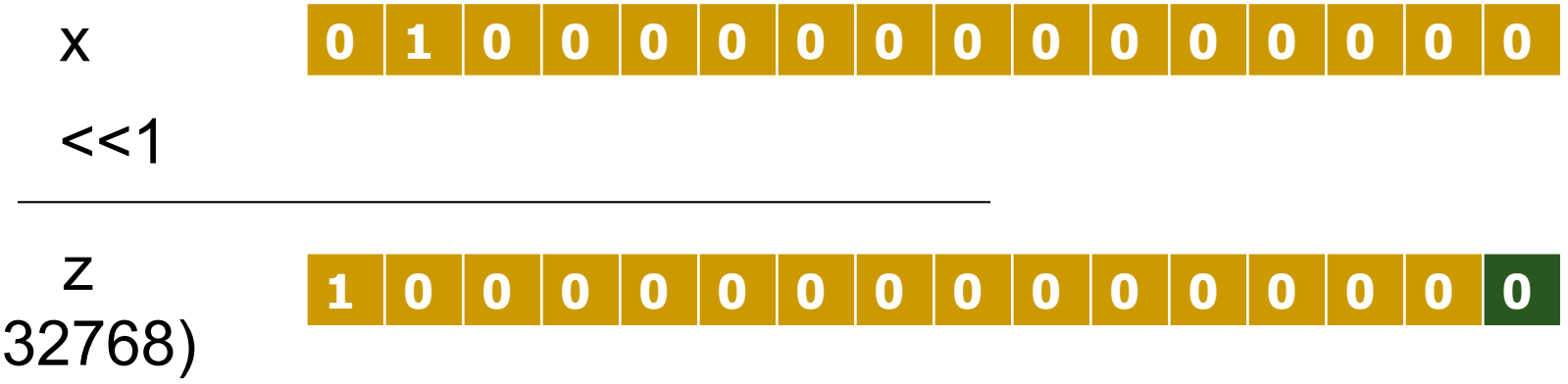
位运算

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

int main() {
   short x  = 16384;
   short z = x << 1;
   return 0;
}
```

向左移位 <<:
按二进制每一位向左移动相应的位数，高位移出（舍弃），低位的空位补0

如果x是有符号数，左移可能导致符号变化

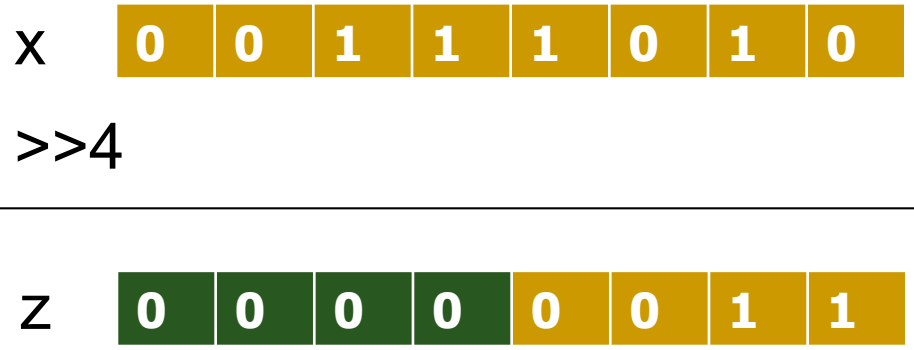


位运算

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

int main() {
   unsigned char x;
   x = 58;
   unsigned char z;
   z = x >> 4;
   cout<<z;
   return 0;
}
```

向右移位 >>:
对于无符号数，按二进制每一位向右移动相应的位数，低位移出（舍弃），高位的空位补0，相当于除以2^N

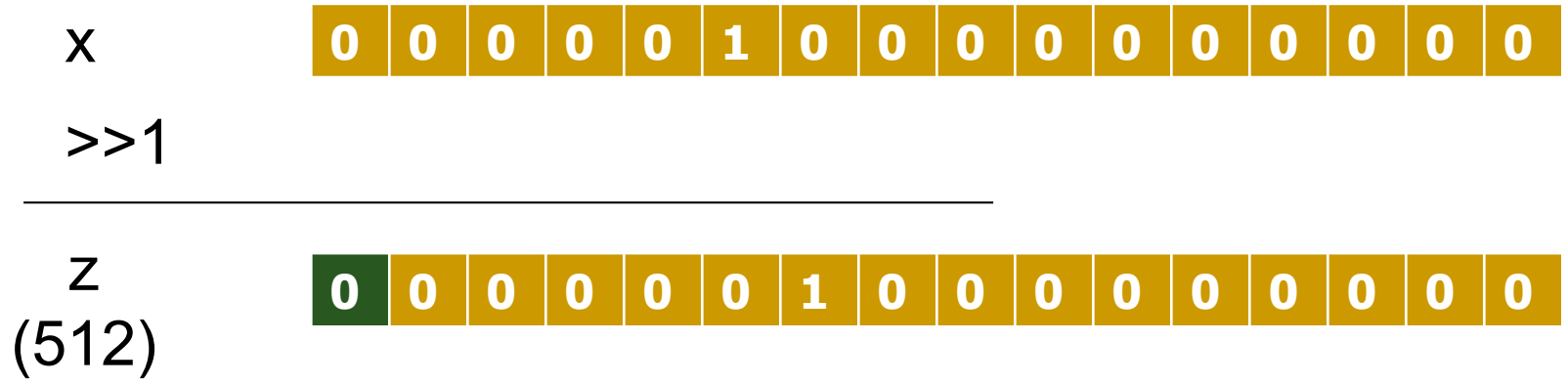


位运算

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

int main() {
   short x  = 1024;
   short z = x >> 1;
   return 0;
}
```

向右移位 >>:
对于有符号数，按二进制每一位向右移动相应的位数，低位移出（舍弃），正数高位的空位补0，负数高位的空位补1

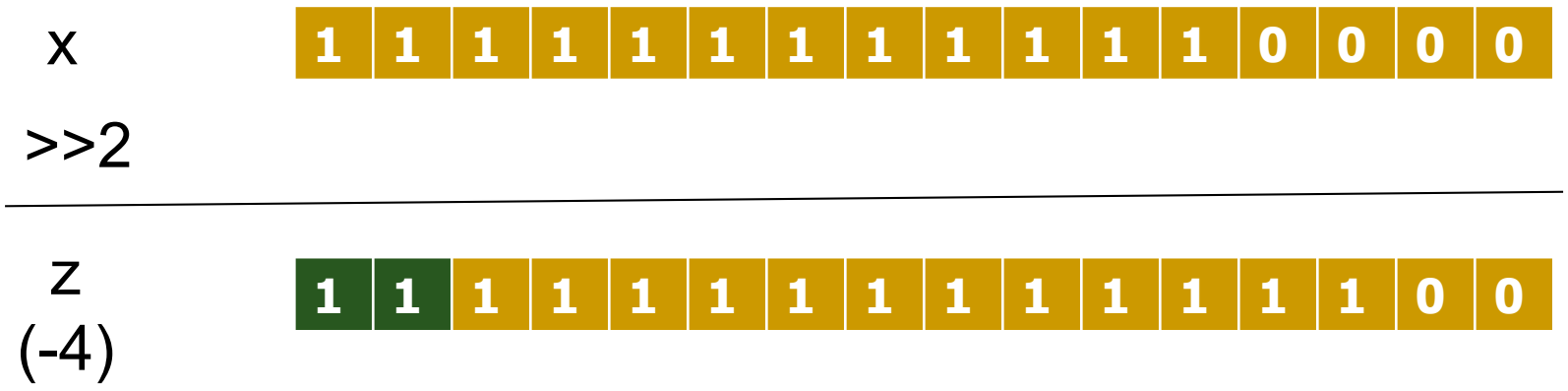


位运算

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

int main() {
   short x  = -16;
   short z = x >> 2;
   return 0;
}
```

向右移位 >>:
对于有符号数，按二进制每一位向右移动相应的位数，低位移出（舍弃），正数高位的空位补0，负数高位的空位补1



条件运算符

- 条件运算符
  - <表达式1>？<表达式2>：<表达式3>
  - 执行过程：计算表达式1，如果结果为true，计算表达式2，否则计算表达式3

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

int main() {
   int x = 1, y = 2;
   int z = (x>y)?x:y;
   return 0;
}
```

(x>y)的值是false，因此执行表达式3，即y，所以，z的值为y

逗号运算符

- 逗号运算符
  - 用逗号隔开的多个表达式
  - 执行过程：从左到右依次执行每个表达式，将最后一个表达式的作为整个逗号表达式的值

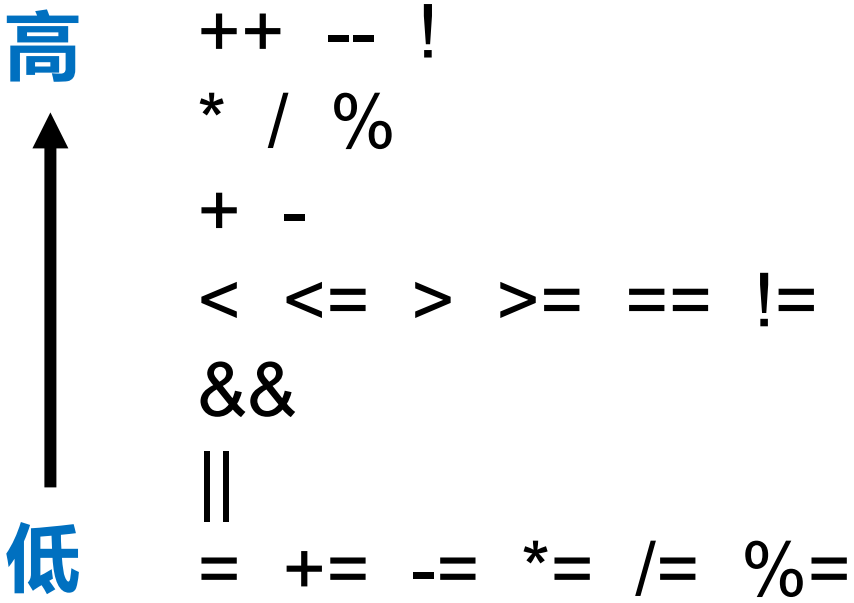
```
int main() {
   int x = 1, y = 2;
   int z = (x++, x-y, x*y);
   cout<<x<<y<<z<<endl;
   return 0;
}
```

先执行x++，x变为2，再执行x-y，再执行x*y，将x*y的值作为整个表达式的赋给z（注意此时x为2）

程序输出：2 2 4

运算符的优先级

不同的运算符的优先级不同



- 运算的优先顺序
  - 括号优先
  - 优先级高的运算符优先
  - 优先级相同的运算按照运算符结合性依次进行

运算符的优先级

不同的运算符的优先级不同

高

低

```
++  --  !
*  /  %
+  -
<  <=  >  >=  ==  !=
&&
||
=  +=  -=  *=  /=  %=
```

```
a + b*c;
```

乘法的优先级高于加法，所以应该是 a + (b*c)，而不是 (a+b)*c

无需记忆所有的优先级，通过加括号来避免错误!

运算符的结合性

优先级相同的运算符的执行顺序

- 左结合规则
  - 从左向右依次计算 (a + b + c)
    - 双目的算术运算符、关系运算符、逻辑运算符、位运算符、逗号运算符
- 右结合规则
  - 从右向左依次计算 (a = b = c)
    - 可以连续运算的单目运算符、赋值运算符、条件运算符

表达式的执行顺序

由优先级和结合性共同决定

6+3-5*4+2; 的计算顺序是?

括号标记法：优先级高的先加括号，优先级相同的，按结合性加括号

步骤1: 6+3-5*4+2中，*运算符优先级最高，因此，将5*4加括号，变为6+3-(5*4)+2

步骤2: 6+3-(5*4)+2中，剩余的运算符优先级相同，这些运算符是左结合，所以将6+3加括号，得到(6+3)-(5*4)+2

表达式的执行顺序

由优先级和结合性共同决定

6+3-5*4+2; 的计算顺序是?

括号标记法：优先级高的先加括号，优先级相同的，按结合性加括号

步骤3: (6+3)-(5*4)+2中剩余的运算符优先级相同，并且是左结合，所以将(6+3)-(5*4)加括号，得到((6+3)-(5*4))+2，至此，只剩下一个+运算符，分析完毕

表达式的执行顺序

由优先级和结合性共同决定

6+3-5*4+2; 的计算顺序是?

括号标记法：优先级高的先加括号，优先级相同的，按结合性加括号

((6+3)-(5*4))+2的计算过程: 先计算+号左边，即((6+3)-(5*4))，但这个括号内还有括号，先计算-号左边，即(6+3)，再计算(5*4)，然后计算(9+20)，最后计算29+2

虽然*优先级最高，但其实6+3先算!

逻辑短路

```
int main() {
   int a = 1, b = 2, c = 3;
   '0' || a++ && b++ || (c=2);
   cout<<a<<b<<c;
   return 0;
}
```

程序输出:
1
2
3

括号标记法得到:

```
'0' || ((a++) && (b++)) || (c = 2)
```

‘0’ 为字符，值为true，因此，整个表达式的值必为true,后续运算被省略，称为 “逻辑短路”

逻辑短路

```
int main() {
    int i = 3, j = 5;
    int c = i > j && j++ || i++;
    cout<<c<<j<<i<<endl;
    return 0;
}
```

程序输出： 1 5 4

括号标记法得到：

```
( (i > j)  &&  (j++) )  ||  (i++) ;
```

(i>j)的值为false，因此(i>y)&&(j++)肯定为false,所以(j++)被省略

END