再谈，我们的进度安排

感性认识作业练习
（以抄程序为主）

数据成分　　运算成分　　控制成分

数组　　字符串　　结构化的程序

函数　　指针　　结构体　　递归　　动态规划

输入输出、流与文件

前情提要

- 数据成分
  - 十进制负数：-2
  - 4比特二进制表示：
  - 反码：
  - 补码：
  - 十进制正数：+1
  - 4比特原码：
  - 反码：
  - 补码：

前情提要

- 数据成分
  - 十进制负数：-2
  - 4比特二进制表示：1010（起名原码）
  - 反码：1101（取反）
  - 补码：1110（取反加一）
  - 十进制正数：+1
  - 4比特原码：0001
  - 反码：0001
  - 补码：0001

符号位/比特

补码如何转为原码？

最高位MSB　　最低位LSB

前情提要

- 数据成分
  - 整型（short，int，long……）
  - 浮点型（float，double）
  - 字符型（char）
  - 布尔型（bool）
  - 数组和字符数组（int a[64], char b[32]……）
  - 结构体

注意各数据类型范围

前情提要

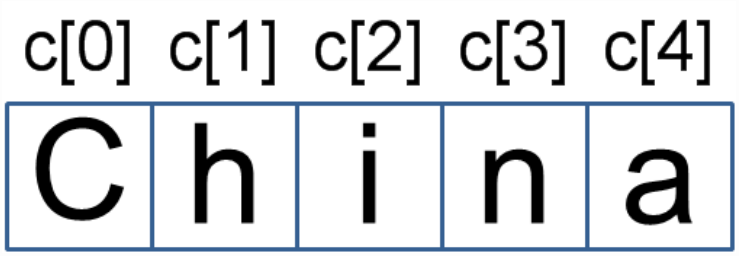
- 数据成分
  - 练习：假如整数10和-9的8位二进制原码表示在长度8的整型数组中，输出其反码和补码？

前情提要

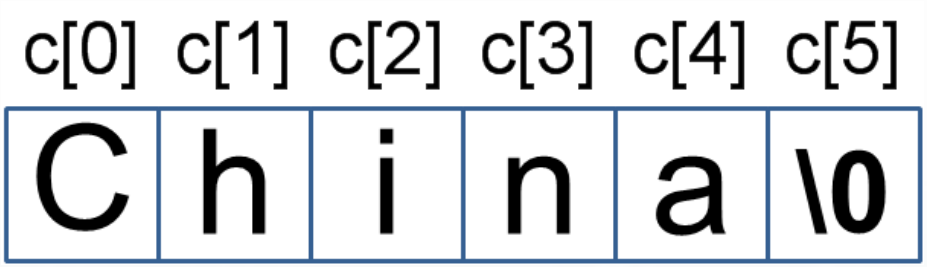
- 数据成分
  - 练习：假如整数10和-9的8位二进制原码表示在长度12 的字符数组中，输出其反码和补码？
  - 变种练习：假如上述二进制原码是按照字符数组输入的呢？

字符数组的初始化

char c[ ] = {‘C', ‘h ', ‘i', ‘n', ‘a'};



char c[ ] = “China”;



认识一下字符串

“China”;

char c[5] = “China”; ?

C 语言的构成成分——运算成分

北京大学 信息科学技术学院

Courtesy 李戈《计算概论》

C 语言中的运算符

- C 语言的运算符范围很宽
  - 求字节数运算符: sizeof
  - 下标运算符 [ ]
  - 赋值运算符 =
  - 算术运算符 + - * / %
  - 关系运算符 < > == >= <= !=
  - 逻辑运算符 ! && ||
  - 条件运算符 ? :
  - 逗号运算符 ,
  - 位运算符 >> ~ | ^ &
  - 指针运算符 * , &
  - 强制类型转换运算符: (类型)
  - 分量运算符 . →

赋值运算符

- “=” 赋值运算符
  - 给赋值号左边的变量赋予数值
- 在变量定义的同时可以为变量赋初值。如：
  - int a=3;

    相当于:

    int a; a=3;
  - int a, b, c = 5

    表示只给 c 赋初值。相当于

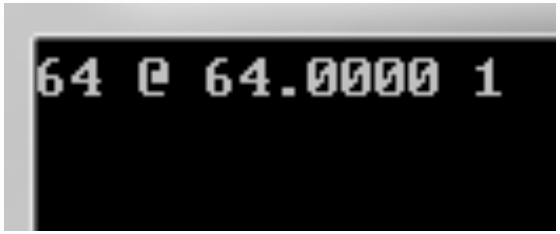
    int a, b, c;

    c = 5;
  - int a = b = c = 5; (Is this right?)

赋值运算符

- 要点一：两边类型不同
  - 若 = 两边的类型不一致，赋值时要进行类型转换。
  - 不管 = 右边的操作数是什么类型，都要转换为 = 左边的类型

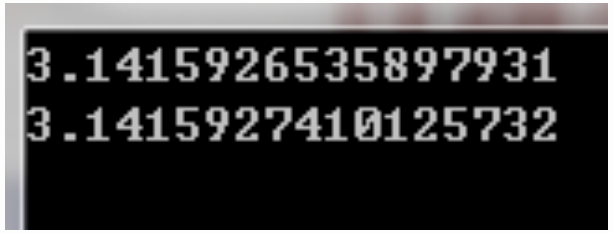
```
int main()
{
        int int_i = 64.12345;
        char char_i = int_i;
        float float_i = char_i;
        bool bool_i = float_i;
        cout << showpoint << int_i << " " << char_i <<
                                " " << float_i << " " << bool_i << endl;
        return 0;
}
```



自动类型转换（1）

- 如果=右边的操作数具有**较高级别**的类型，则在类型转换时，进行**截断取舍**，可能会损失精度!

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        double double_i = 3.14159265358979323846;
        cout << setprecision(30) << double_i << endl;
        float float_i = double_i;
        cout << float_i << endl;
        return 0;
}
```



自动类型转换（2）

- 如果=右边的操作数类型级别较低，则在类型转换时，采用补齐方式，不会损失精度。

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        float float_i = 3;
        cout << showpoint << float_i << endl;
        double double_i = float_i;
        cout << setprecision(20) << double_i << endl;
        return 0;
}
```

赋值运算符

- 要点二：长数赋给短数
  - 截取长数的低n位送给短数


```
int main()
{
        char char_a  = ' ';
        int int_i = 0x361;
        cout << hex << int_i << endl;
        char_a = int_i;
        cout << char_a << endl;
        return 0;
}
```


```
361
a
```

赋值运算符

- 举例：short = long
  - 截取长整型数的低16位送给short字符。
  - 如果最高位为1，则得到负数，否则得到正数;


```
int main()
{
        long int long_i = 0x2AAAAAAA;
        cout << long_i << endl;
        short short_j = long_i;
        cout << hex << short_j <<endl;
        cout << dec << short_j << endl;
        return 0;
}
```


```
715827882
aaaa
-21846
```

赋值运算符

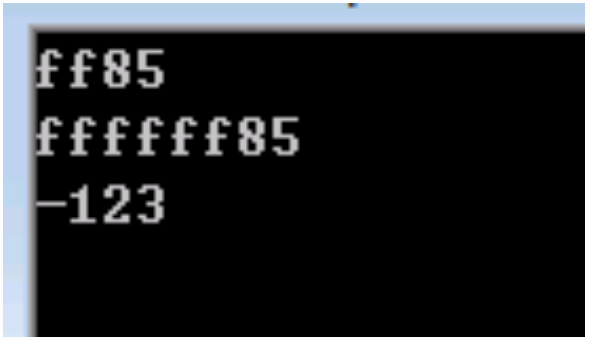
- 要点三：短数赋给长数
  - 最好处理的情况！原来是什么数，现在还是什么数！
  - short int a = -1;　　int b = a;

计算机的处理过程：

- 若short 型数为**无符号数**：
  - short 型16位到long型低16位，long型高16位补0；
- 若 short型数为**有符号数**：
  - short型16位到 long型低16位；
  - 若short型最高位为0，则long型高16位补0；
  - 若short型最高位为1，则long型高16位补1；

示例

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        short short_i = -123;
        cout << hex << short_i << endl;
        int  int_j = short_i;
        cout << hex << int_j <<endl;
        cout << dec << int_j << endl;
        return 0;
}
```



赋值运算符

- 要点四：符号位的赋值处理
  - 也很好处理的情况！直接搬运！
  - 直接赋值，不管符号位还是数字位！

例如：

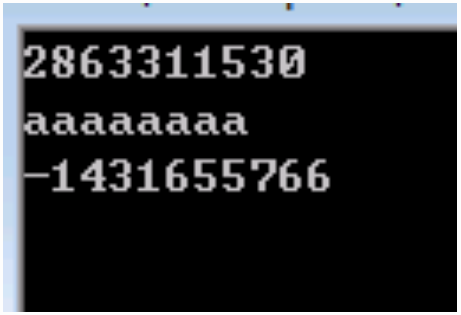
- unsigned = int 或long
  - 直接赋值，符号位当作数字。
- int = unsigned int
  - 直接赋值，数字当作符号位。

signed

unsigned

示例

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        unsigned int unsigned_int_i = 0xAAAAAAAA;
        cout << unsigned_int_i << endl;
        signed int  signed_int_j = unsigned_int_i;
        cout << hex << signed_int_j << endl;
        cout << dec << signed_int_j << endl;
        return 0;
}
```



signed

unsigned

赋值运算总结

- 当：两边类型不同
  - 自动完成类型转换！
- 当：长数赋给短数
  - 截取长数的低位送给短数！
- 当：短数赋给长数
  - 原来是什么数，现在还是什么数！
- 当：符号位的赋值处理
  - 直接赋值，不管符号位还是数字位！

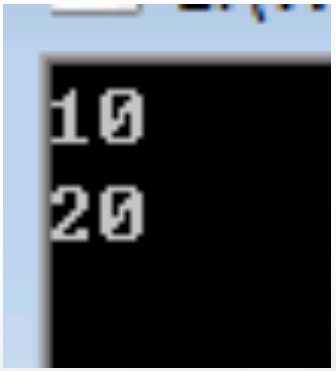
表达式

- 什么是表达式
  - 程序中由运算符、操作数和括号组成等所组成的计算式，是计算求值的基本单位。

a * b + c;　　123 < 10;　　'a'*3.14f == 1;　　a = b;

  - 表达式是有“值”的，赋值语句也不例外；

```
int main()
{
        int i = 0;
        cout << (i = 10) << endl;
        cout << (i = i + i) << endl;
        return 0;
}
```

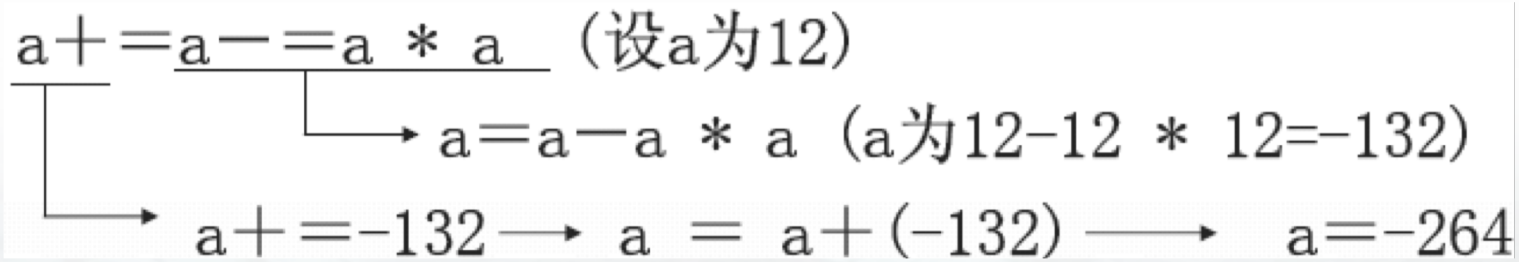


赋值表达式

- 复合的赋值运算
  - 在赋值符号前加上其它运算符号则构成复合赋值运算；
  - 例如：
    - a + = 3;　　等价于a = a + 3;
    - x * = y + 8;　　等价于x = x * (y + 8);
    - x % = 3;　　等价于x = x % 3;

赋值表达式

- 连续的赋值运算
  - 自右而左的结合顺序
    - a = b = 5; //a, b均为5
    - a = b = c = 5; //a, b, c均为5
      - int a=b=c=5; //编译错！
  - a = (b = 4) + (c = 6); //a为10, b为4, c为6
- 举例：



算术运算符和算术表达式

- 算术运算符和算术表达式
  - 基本的算术运算 + 、 - 、*、 / 、%
    - % 是模运算，即，求余运算，必须是整数
      - 如 7 % 4 = 3
- 注意：
  - 整数运算，结果仍为整数
    - 5/3 的结果仍是整数，小数部分被忽略
  - 实数运算，结果为double型
    - 5.3/3 或 5/3.0 的结果为double型
  - 舍入的方向随编译器的不同而不同

算术表达式

- 算术运算符的优先级

  ( )

  * / %

  + -

  - 在同一级别中，采取由左至右的结合方向
    - 如：a – b + c 相当于(a – b) + c
    - 如：a % b * c / d 相当于(((a % b)*c))/d
  - 当数据类型非常复杂时
    - 如：123%s + (i+‘@’) + i*u – f/d

    其中 short s; int i; float f; double d; unsigned u;

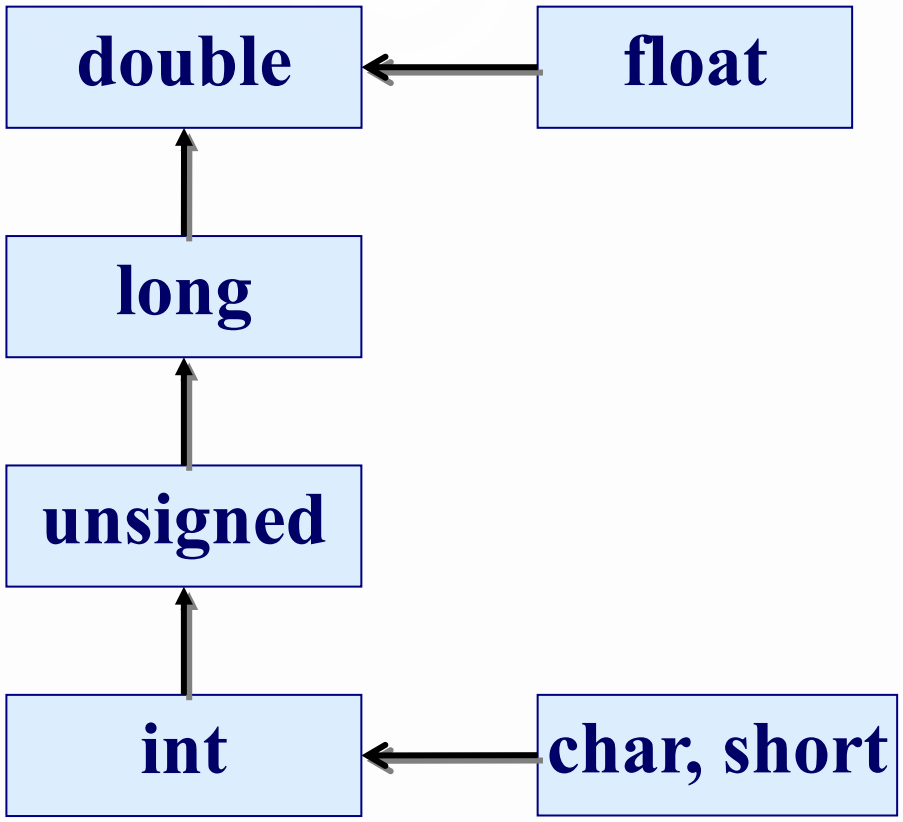
算术表达式

- **剪刀法求表达式的值**

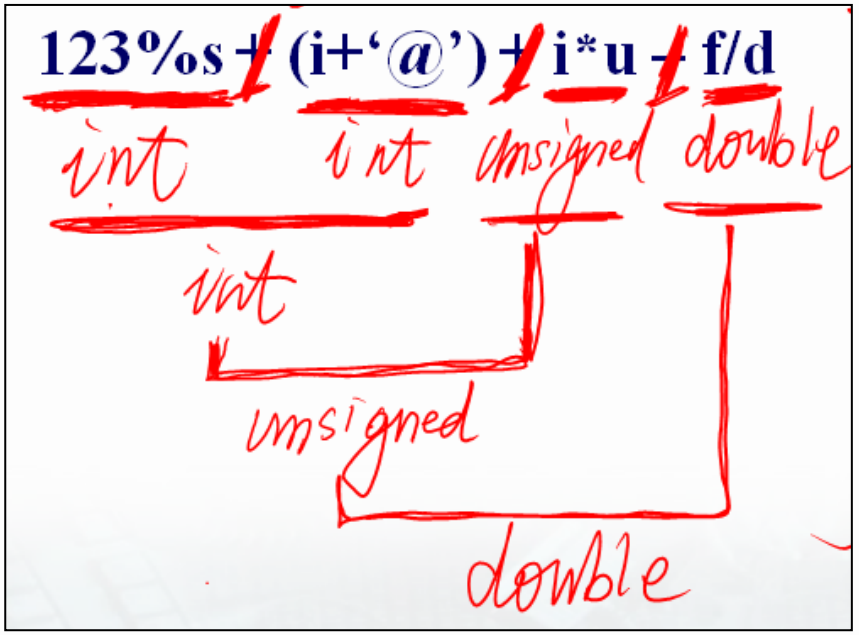
  short s; int i; float f; double d; unsigned u;

  123%s + (i+‘@’) + i*u – f/d

计算过程中的类型转换



short s; int i; float f; double d; unsigned u;



算术表达式

- 自增、自减运算符：使变量的值加1或减1
  - ++i,　--i
    - 在使用i之前，先将i的值加（减）1
  - i++,　i--
    - 在使用i之后，再将i的值加（减）1
- 例如：i 的值为3，则
  - j = ++i;
  - j = i++;
  - cout<<++i;
  - cout<<i++;

自增、自减运算符

- 举例：
  - 设i的值为3；则

```
cout<<-i++<<endl;
cout<<-++i<<endl;
cout<<(-i)++<<endl;
cout<<++i++<<endl;
```

- 注意：
  - ++和--只能用于变量

自增、自减运算符

```
int main()
{
        int a = 0, b = 0, c = 2, d = 0, e = 2, f = 2;
        cout << a << " " << a++ << " " << endl;
        cout << ++b << " " << b++ << " " << endl;
        cout << c << " " << (c++) + (++c) << " " << endl;
        cout << (d = f++) + (e = f) << endl;
        cout << f << " " << d << " " << e << endl;
        return 0;
}
```

自增、自减运算符

```
1 0
2 0
4 6
4
3 2 2
```

```
int main()
{
    int a = 0, b = 0, c = 2, d = 0, e = 2, f = 2;
    cout << a << " " << a++ << " " << endl;
    cout << ++b << " " << b++ << " " << endl;
    cout << c << " " << (c++) + (++c) << " " << endl;
    cout << (d = f++) + (e = f) << endl;
    cout << f << " " << d << " " << e << endl;
    return 0;
}
```

关系运算符

- C语言提供6种关系运算符：

① <　　(小于)
② <=　(小于或等于)
③ >　　(大于)
④ >=　(大于或等于)

优先级相同　高

⑤ ==　(等于)
⑥ !=　(不等于)

优先级相同　低

关系运算表达式

- **关系运算表达式的值：**

  “真”/“假”

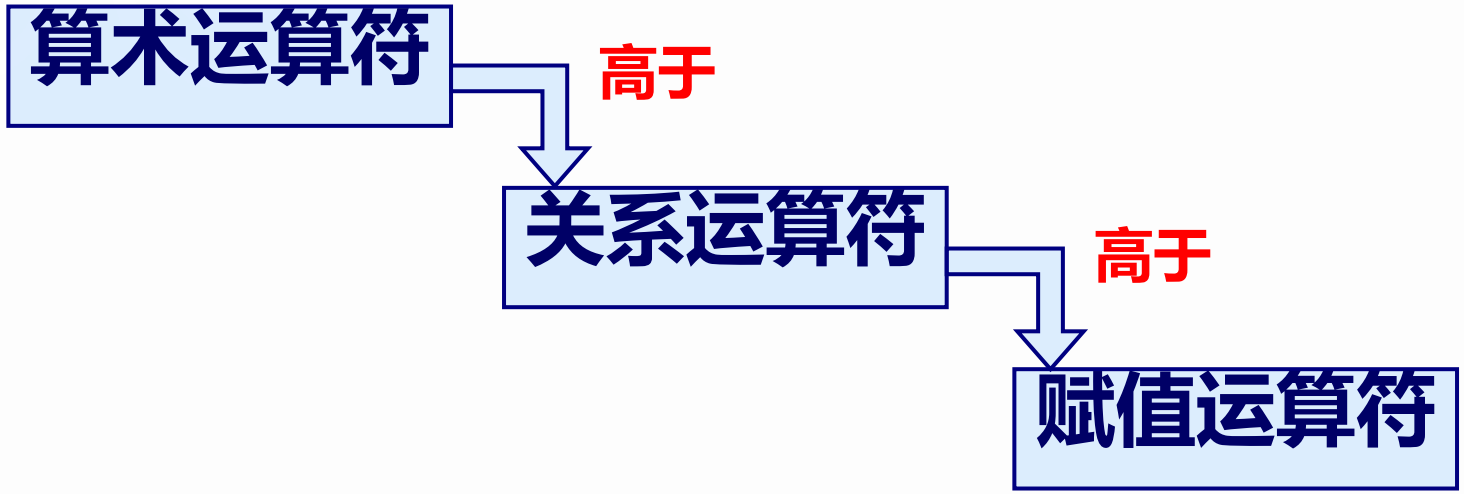
  - **如 a = 3，b = 4**

    a > b 的值为 0

    a != b 的值为 1

    a == b 的值为 0

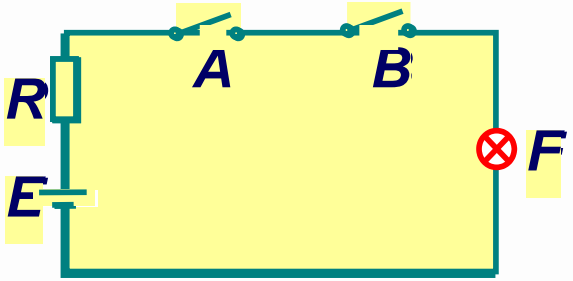
运算符的优先级



- 例如：
  - 1 + 2 % 3 * 4 > 5 / 6 – 7
  - 1 + 2 % (3 * 4) > (5 / 6 – 7) = = 8
  - X = 1 + 2 % 3 * 4 > 5 / 6 – 7 = = 8
  - 1>2= =3>4

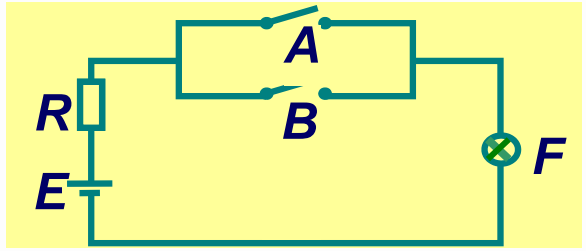
逻辑运算符

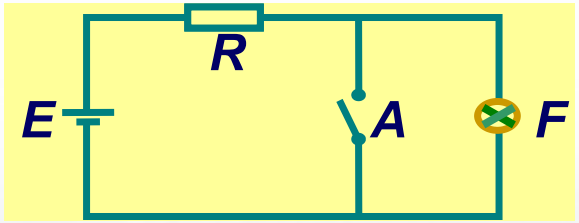
- 三种逻辑运算符：
  - 逻辑与 &&
  - 逻辑或 ||
  - 逻辑非 !



【注】：

- 若A、B为真，则F=A&&B为真。
- 若A、B之一为真，则F=A||B为真。
- 若A为真，则!A为假。





逻辑表达式的值

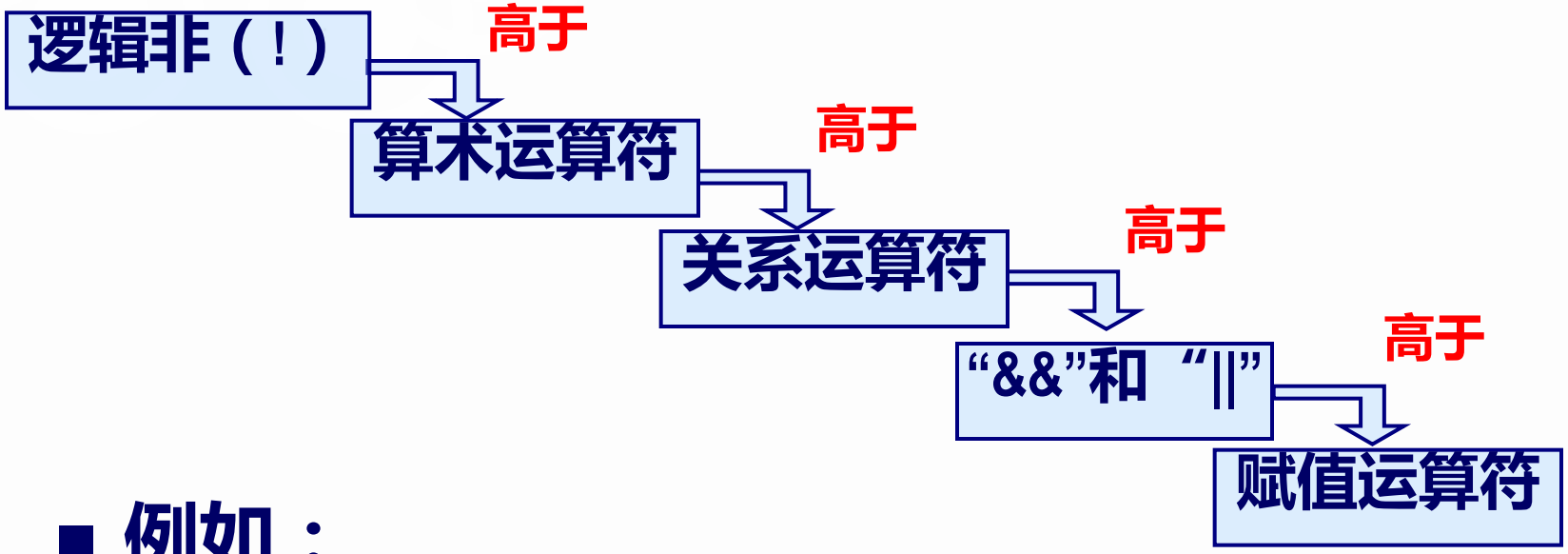
- 以0代表“假”；以非0代表“真”。
  - 若 a = 4 ，则 ! a 的值为 0。
  - 若 a = 4，b = 5，则 a && b 的值为1。
  - 若 a = 4，b = 5，则 a || b 的值为1。
  - 若 a = 4，b = 5，则 !a || b 的值为1。
  - 表达式 4 && 0 || 2 的值为1。

逻辑运算符优先级

- 逻辑表达式
  - 一个逻辑表达式中若包含多个逻辑运算符，则按以下的优先次序：
    - ! (非)→&& (与)→|| (或)，即“!”优先级最高。
  - 例如：

```
!a&&b||c;
! a && b || x > y && c;
! a && b || x > y && c + 1;
```

混合运算的优先级



- 例如：
  - a > b && x > y可写成(a > b) && (x > y)
  - a == b || x == y可写成(a ==b) || (x == y)
  - ! a || a > b可写成 (! a) || (a > b)

思考题

```
#include<iostream>
using namespace std;
int main()
{
    int a = 0, b = 0;
    a = 5 > 3 && 2 || 8 < 4 - (b = !0);
    cout<<a<<" "<<b;
    return 0;
}
```

逻辑运算的取舍

- 逻辑表达式求解中，并不总是执行所有的运算
  - 只有在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符！
    - 对于表达式 a && b && c
      - 只有a为真(非0)时，才需要判别b的值，
      - 只有a和b都为真的情况下才需要判别c的值。
    - 对于表达式 a || b || c
      - 只要a为真(非0)，就不必判断b和c；只有a为假，才判别b；a和b都为假才判别c。

逻辑运算的取舍举例

```
#include<iostream>
using namespace std;
int main( )
{
   int i = 0, a = 1, b = 2, c = 3;
    i = ++a || ++b || c++;
   cout<<i <<" "<< a<<" "<<b<<" "<<c<<endl;
   return 0;
}
```

运算对象的扩展

- 逻辑运算符两侧可以是任何类型
  - 如：字符型、实型或指针型等；
  - 系统最终以 0 和 非0 来判定它们，例如

    'c' && 'd'

    - ‘c’和 ‘d’的ASCII值都不为0，按“真”处理。
- 思考：
  - ‘a’ = = ‘b’ &&! ‘c’

示例

```
#include<iostream>
using namespace std;
int main()
{
    int a = 0, b = 1;
    a =  8>4 - (b = !'c') && 5>3 + 'a' % 6 == 'b';
    cout<<a<<" "<<b;
    return 0;
}
```

应用示例

- 问题：要判别某一年year是否闰年。闰年的条件是符合下面二者之一：
  - ①能被4整除，但不能被100整除。
  - ②能被100整除，又能被400整除。

```cpp
#include <iostream>
using namespace std;
int main()
{
	int year = 0;
	cin >> year;
	if (year % 4 == 0)
	{
		if (year % 100 == 0)
		{
			if (year % 400 == 0)
				cout << "Y";
			else
				cout << "N";
		}
		else
			cout << "Y";
	}
	else
		cout << "N";
	return 0;
}
```

应用示例

- 问题：要判别某一年year是否闰年。闰年的条件是符合下面二者之一：
  - ①能被4整除，但不能被100整除。
  - ②能被100整除，又能被400整除。
- 求解：
  - 可以用如下逻辑表达式来判别year是否为闰年：

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

  - 也可以用如下逻辑表达式判别year是否非闰年：

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

条件运算符

```
#include <iostream>
using namespace std;
void main()
{
    int i;
    int n = -2147483648;
    unsigned int b = 0x80000000;
    for (i = 0; i < 32; i ++) {
        cout<<n & b ? 1 : 0;
        b >>= 1;
    }
}
```

条件运算符

表达式1？表达式2： 表达式3

求值规则：如果表达式1的值为真，则以表达式2 的值作为条件表达式的值；否则以表达式3的值作为整个条件表达式的值。

例如：

```
max=(a>b)?a:b;
```

相当于：

```
if(a>b)  max=a;
else max=b;
```

逗号运算和逗号表达式

- 用逗号将两个表达式连起来
  - 表达式1，表达式2，表达式3，…… 表达式n
  - 先求表达式1，再求表达式2，……，再求表达式n ，整个表达式的值为表达式n的值

```
a = 3 * 5, a * 4;
```

- 下式中是否相同？
  - x = (a = 3, 6 * 3) ;
  - x = a = 3 , 6 * 3 ;

强制类型转换

- 形式:
  - (类型名)（表达式）
- 举例:
  - (double)a　　将a的值转换成double类型
  - (int)(x+y)　　将x+y的值转换成int类型
  - (float)(5/3)　　将5/3的值转换成float类型
- 注意:
  - 强制类型转换后，被转换的量的类型并没有发生变化

破案

- 题目
  - 试写一个程序，帮助某地刑侦大队利用以下已经掌握的确切线索，从6个嫌疑人中找出作案人：
    - A，B至少有一人作案；
    - A，E，F 3人中至少有2人参与作案；
    - A，D不可能是同案犯；
    - B，C或同时作案，或与本案无关；
    - C，D中有且仅有1人作案；
    - 如果D没有参与作案，则E也不可能参与作案

解题步骤

- 基本方法
  - 测试所有可能的组合；
  - 对于每种组合，测试是否满足已知条件
  - 若满足，该组合是目标；
- 前提条件
  - 将所有已知条件表达出来
    - 如何表示A是作案者？
  - 找到控制循环的方法

条件表达

- A，B至少有一人作案；
  - cc1 = A||B;
- A，E，F 3人中至少有2人参与作案；
  - cc2 = (A&&E)||(A&&F)||(E&&F)
- A，D不可能是同案犯；
  - cc3=!(A&&D)
- B，C或同时作案，或与本案无关；
  - cc4=(B&&C)||(!B&&!C)
- C，D中有且仅有1人作案；
  - cc5=(C&!D)||(!C&&D)
- 如果D没有参与作案，则E也不可能参与作案
  - cc6=D||!E

程序框架

```
main( )
{    … …
     for(A = 0; A <= 1; A++)
               for(B = 0; B <= 1; B++)
                         for(C = 0; C <= 1; C++)
                                   for(D = 0; D <= 1; D++)
                                             for(E = 0; E <= 1; E++)
                                                       for(F = 0; F <= 1; F++)
                                                       {
                                                                 if(cc1+cc2+…+cc6)
                                                                 {
                                                                           //打印结果；
                                                                 }
                                                       }
}
```

选讲内容 —— 位运算

位运算

- 位运算
  - 所谓位运算是指进行二进制位的运算。
- C++语言中的位运算符
  - 按位与（&）　　双目运算符
  - 按位或（|）　　双目运算符
  - 按位异或（^）　　双目运算符
  - 取反（~）　　单目运算符
  - 左移（<<）　　单目运算符
  - 右移（>>）　　单目运算符

位运算符（1）

- “按位与”运算符(&)
  - 参加运算的两个数据，各位均独立进行“与”运算。
- 例如：对于表达式：a＝3 & 5，有：

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

因此，3&5的值为：1

位运算符（2）

- “按位或”运算符（|）
  - 参加运算的两个数据，各位均独立进行“或”运算。
- 例如：对于表达式：a = 3 | 5，有：

$$
\begin{array}{r}
3 = 00000011 \\
(|)\ 5 = 00000101 \\
\hline
00000111
\end{array}
$$

因此，3 | 5的值为：7

位运算符（3）

- “异或”运算符（∧）
  - 异或运算符∧也称XOR运算符。参加运算的两个数，各位均独立进行 异或运算：
  - 即0∧0 = 0；0∧1 = 1；1∧0 = 1；1∧1 = 0；

  例如：

```
        00111001   (十进制数57，八进制数071)
(∧)     00101010   (十进制数42，八进制数052)
--------------------------------------------
        00010011   (十进制数19，八进制数023)
```

  即071∧052，结果为023(八进制数)。

位运算符（4）

- 取反运算符“~”是一个单目(元)运算符，用来对一个二进制数按位取反，
  - 即将0变1，1变0。
  - 例如：

000000000010101

(~) ↓

111111111101010

因此，~(025)$_8$的值为(177752)$_8$

位运算符（5）

- 左移运算符(<<)
  - 用来将一个数的各二进位全部左移若干位。
  - 高位左移后溢出，舍弃不起作用。
  - 若a=15，即二进制数00001111，左移2位得00111100，即十进制数60

$$a = a<<2$$

  - 左移1位相当于该数乘以2，左移2位相当于该数乘以$2^2=4$。
    - 只适用于该数左移时被溢出舍弃的高位中不包含1的情况。

位运算符（6）

- 右移运算符(>>)
  - 用来将一个数的各二进位全部右移若干位。
  - 移到右端的低位被舍弃，对无符号数，高位补0。
  - 若a=15，即二进制数00001111，右移2位得00000011，即十进制数3；

$$a = a >> 2$$

  - 当没有非零数位被抛弃时，右移一位相当于除以2，右移n位相当于除以2^n。

关于位运算的几个问题(1)

- 右移运算符号位的处理
  - 对无符号数，右移时左边高位移入0。
  - 对于有符号的值,
    - 若原来符号位为0(该数为正)，则左边移入0；
    - 若符号位原来为1(即负数)，则左边移入0还是1,要取决于所用的计算机系统。
      - 若移入0，称为“逻辑右移”或“简单右移”
      - 若移入1，称为“算术右移”（VC）

关于位运算的几个问题(2)

- 不同长度的数据进行位运算
  - 如果两个数据长度不同进行位运算时，系统会将二者按右端对齐。
    - 如a & b，而a为int型，b为short型
  - 如何补位
    - 如果b为无符号整数型，则左侧添满0。
    - 如果b为有符号整数型：
      - 如果b为正数，则左侧16位补满0。
      - 如果b为负数，则左端16位补满1。

关于位运算的几个问题(3)

- 位运算赋值运算符
  - 位运算符与赋值运算符可以组成复合赋值运算符如: &= ，|= ， >>= ， <<= ， ∧=
    - a & = b　相当于　a = a & b
    - a | = b　相当于　a = a | b
    - a >> =2　相当于　a = a >> 2
    - a << =2　相当于　a = a << 2
    - a∧= b　相当于　a = a ∧ b

关于位运算的几个问题(4)

- 算符优先级
  - 取反运算符“~”
  - 算数运算符
  - 左移<< 右移>>
  - 关系运算符
  - 按位与&
  - 按位异或^
  - 按位或 |
  - 逻辑运算符

高

低

位运算的使用（1）

- “按位与”运算的用途
  - 任意存储单元与0进行“按位与”运算可清零；
  - 取一个数中某些指定位
    - 如有一个整数a，想要其中的低字节。只需将a与 $(377)_8$ 按位与即可。

a	00 10 11 00	10 10 11 00
b	00 00 00 00	11 11 11 11
c	00 00 00 00	10 10 11 00

- “按位或”运算
  - 对一个数据的某些位取定值为1；

位运算的使用（2）

- “异或”运算符的使用

（1）使特定位翻转

- 使01111010低4位翻转（即1变为0，0变为1）可将其与00001111进行∧运算，即：

```
    01111010
(∧) 00001111
------------
    01110101
```

（2）使特定位保持不变

- 与0相∧，保留原值如012∧00=012

```
    00001010
(∧) 00000000
------------
    00001010
```

“异或”运算符示例

- 交换两个值，而不必使用临时变量
  - 假如a = 3，b = 4。想将a和b的值互换，可以用以下赋值语句实现：

a = a ∧ b ;

b = b ∧ a ;

a = a ∧ b ;

设：a = 3, b = 4

a=011

(∧) b=100

a=111 (a∧b的结果，a已变成7)

(∧) b=100

b=011 (b∧a的结果，b已变成3)

(∧) a=111

a=100 (a∧b的结果，a变成4)

好好想想，有没有问题?

谢　谢！