

规格严格 功夫到家

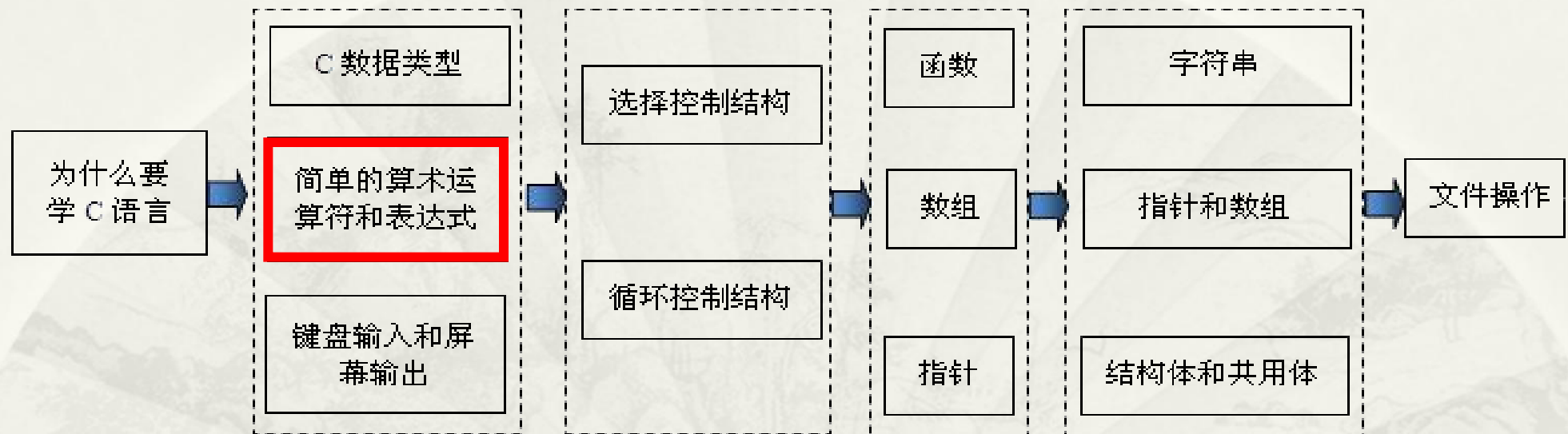


第3章 简单的算术运算和表达式

哈尔滨工业大学（深圳）
计算机科学与技术学院
刘洋

liu.yang@hit.edu.cn





第3章 学习内容

- 算术运算符
- 增1和减1运算符
- 宏常量与**const**常量
- 自动类型转换
- 强制类型转换运算符
- 常用的标准数学函数



为什么要学运算符呢？

- 计算机归根结底所做的事情只有一件——**计算**
- 最基本的运算——**算术运算**
- C语言中的**运算符**有哪些呢？



运算符 (Operator)

- 详见附录C
- 常见的运算符
 - * 赋值运算符
 - * 算术运算符
 - * 增 1 和减 1
 - * 类型强转
 - * 关系运算符
 - * 逻辑运算符
 - * 位运算符.....

算术运算
密切相关



3.1 C运算符和表达式

- 何谓运算符(Operator)和操作数(Operand) ?

Example:

W + Z

操作数

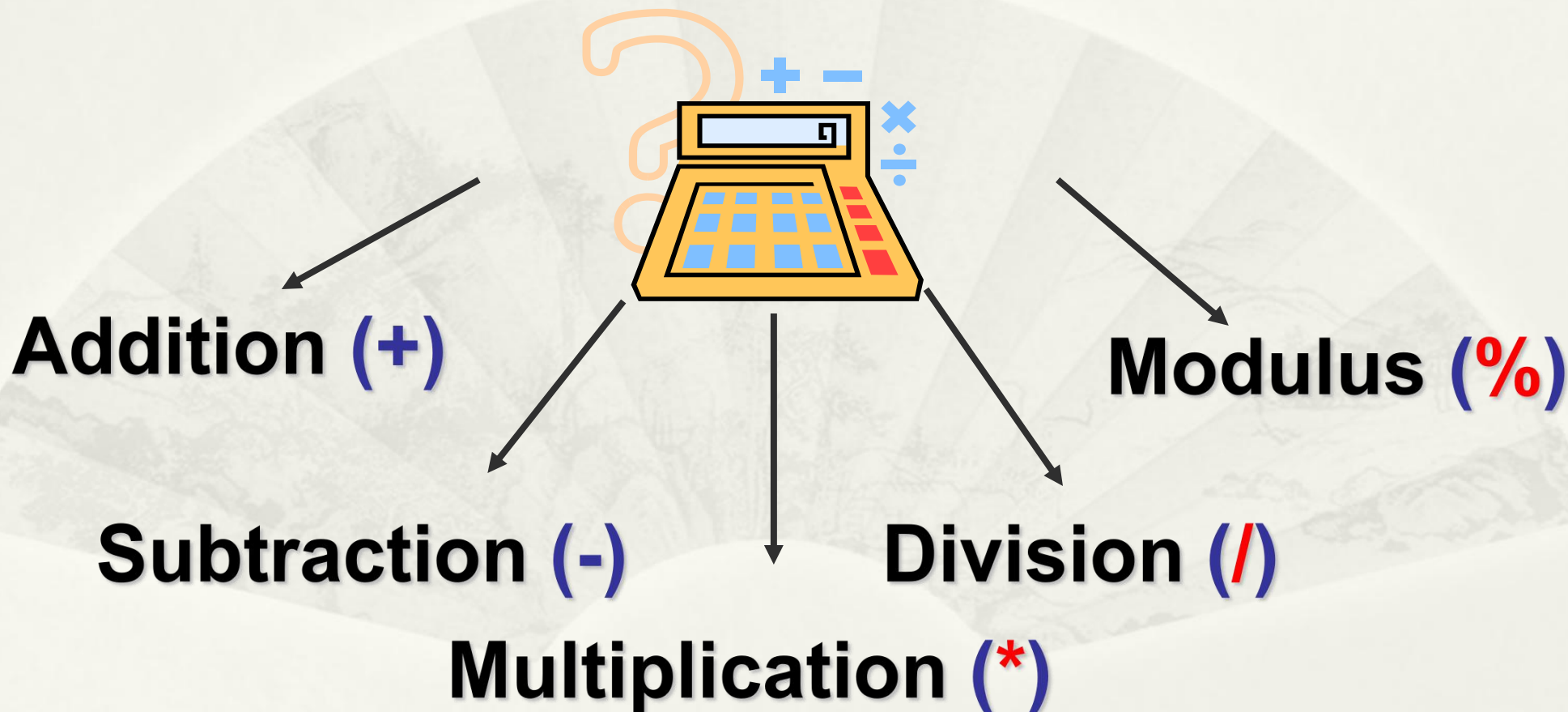
运算符

操作数



3.1.1 算术运算符和表达式

Arithmetic Operators



除法 (Division)

Example:

W / Z

$$11 / 5 = 2$$

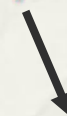


整数除法

(Integer Division)

■ **W** and **Z** are integers

$$11.0 / 5 = 2.2$$



浮点数除法

(Floating Division)

■ **W** or **Z** or both are floats

求余 (Modulus)

- 返回操作数相除之后的**余数 (Remainder)**
- 规则：
 - * 操作数必须是整数，**特别要注意负数参加的求余**

$$11 \% 5 = 1$$

Handwritten long division of 11 by 5. The divisor 5 is on the left. The dividend 11 is inside the division bar. The quotient 2 is written above the bar. The product 10 is written below 11. A horizontal line is drawn under 10, and the remainder 1 is written below it. An arrow points from the remainder 1 to the equation $11 \% 5 = 1$ above.

$$11 \% (-5) = 1$$

Handwritten long division of 11 by -5. The divisor -5 is on the left. The dividend 11 is inside the division bar. The quotient -2 is written above the bar. The product 10 is written below 11. A horizontal line is drawn under 10, and the remainder 1 is written below it. An arrow points from the remainder 1 to the equation $11 \% (-5) = 1$ above.

$$(-11) \% 5 = -1$$

Handwritten long division of -11 by 5. The divisor 5 is on the left. The dividend -11 is inside the division bar. The quotient -2 is written above the bar. The product -10 is written below -11. A horizontal line is drawn under -10, and the remainder -1 is written below it. An arrow points from the remainder -1 to the equation $(-11) \% 5 = -1$ above.

3.1.1 算术运算符和表达式

- **问题：**当**算术表达式 (Arithmetic Expression)**包含**两个或两个以上的运算符**时，根据什么确定**运算顺序**呢？
 - * **运算符的优先级 (Order of Precedence)**
 - * **不同优先级时——从高到低运算**
 - * **相同优先级时——二元算术运算符为左结合 (从左到右)**

3.1.1 算术运算符和表达式

运算符	含义	操作数个数	优先级	结合性
-	取相反数 (Opposite number)	1个 (一元)	最高	从右向左
* / %	乘法 (Multiplication) 除法 (Division) 求余 (Modulus)	2个 (二元)	较低	从左向右
+ -	加法 (Addition) 减法 (Subtraction)	2个 (二元)	最低	从左向右

3.1.1 算术运算符和表达式

`a = - 3 * 2 - 1 + 3;`



`a = ((- 3) * 2) - 1 + 3;`



`a = (- 6) - 1 + 3;`



`a = (- 7) + 3;`



`a = - 4;`

`a = -(3 * 2 - 1 + 3);`



`a = -(6 - 1 + 3);`



`a = -(5 + 3);`



`a = - 8;`

巧妙使用**圆括号**改变运算顺序

——从内往外运算

【例3.1】 计算并输出一个三位整数的 个位、十位和百位数字之和

问题的关键是什么？

——如何分离个位、十位、百位数字？

$$153/100 = 1$$

$$153\%10 = 3$$

$$153 - 1*100 - 5*10 = 3$$

$$153 - 1*100 = 53$$

$$153\%100 = 53$$

$$53/10 = 5$$

$$153/10\%10 = 5$$



3.1.2 复合的赋值运算符

- 三种赋值形式:

- 1. 简单赋值 (Simple Assignment)

变量 = 表达式 ;

- 2. 多重赋值 (Multiple Assignment)

变量1 = 变量2 = 表达式 ;

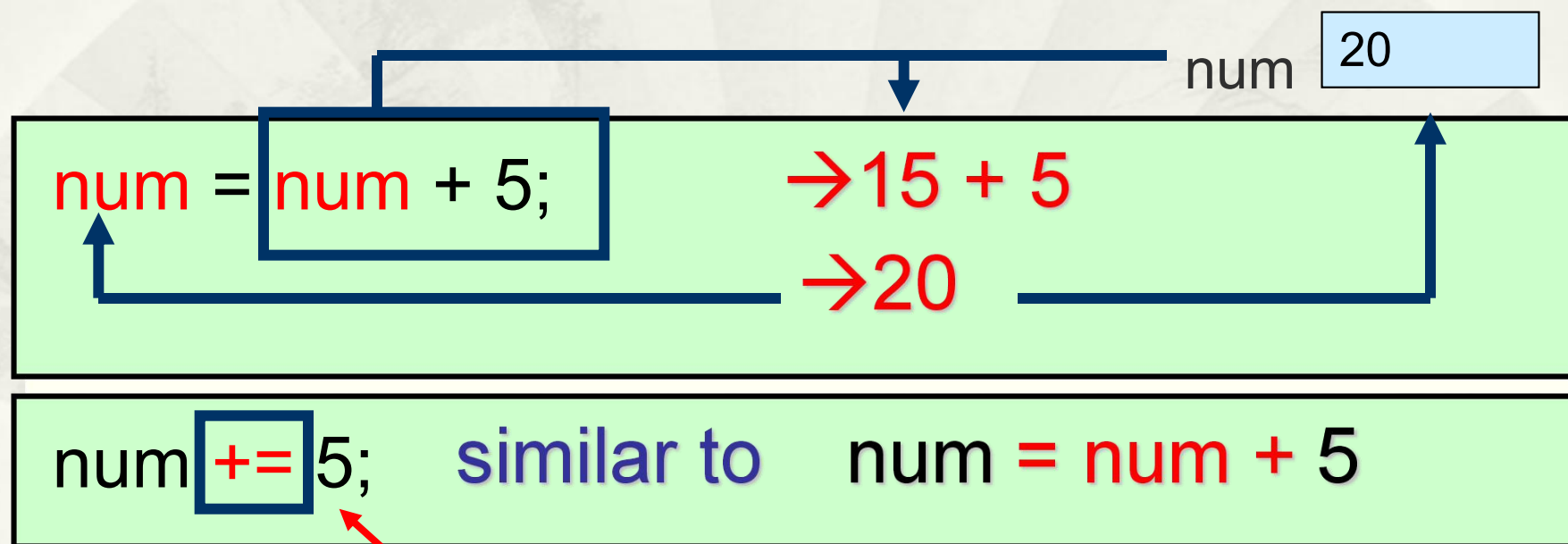
* 赋值运算的结合性是右结合

- 3. 复合的赋值 (Combined Assignment)

3.1.2 复合的赋值运算符

变量 x 运算符 op = 表达式 ;

变量 x = 变量 x 运算符 op 表达式 ;



Shorthand assignment operator

3.1.2 复合的赋值运算符

变量 **x** 运算符 **op** = 表达式 ;

注: **op=** 中间没有空格

简写 (Shorthand) 形式更直观, 且执行效率也更高一些

Operation	Examples of expression	Description
+=	<code>num += 5;</code>	<code>num = num + 5;</code>
-=	<code>num -= 5;</code>	<code>num = num - 5;</code>
*=	<code>num *= 5;</code>	<code>num = num * 5;</code>
/=	<code>num /= 5;</code>	<code>num = num / 5;</code>
%=	<code>num %= 5;</code>	<code>num = num % 5;</code>

已知 `int a = 3;`

执行 `a += a -= a * a` 后，变量a的值？

`a += a -= a * a`

`a += a -= 9`

`a += -6`

`a = -12`

-12

执行 `a += a -= a *= a` 后，变量a的值？

`a += a -= a *= a`

`a += a -= 9`

`a += 0`

`a = 0`

0



3.1.3 增1和减1运算符

■ 增1和减1运算符(Increment and Decrement)

■ $n++$, $n--$, $++n$, $--n$

* $++$ 让参与运算的变量加1, $--$ 让参与运算的变量减1

* 作为前缀(prefix)运算符时, 先加/减1, 然后取 n 的值

$m = ++n;$



$n = n + 1;$

$m = n;$

3.1.3 增1和减1运算符

■ 增1和减1运算符(Increment and Decrement)

■ $n++$, $n--$, $++n$, $--n$

* $++$ 让参与运算的变量加1, $--$ 让参与运算的变量减1

* 作为**后缀(postfix)**运算符时, 先取 n 的值, 然后加/减1

$m = n++;$



$m = n;$

$n = n + 1;$

3.1.3 增1和减1运算符

Prefix Example:

```
j = ++i - 2;
```

similar to

→

```
i = i + 1;
```

→

```
j = i - 2;
```

i 6

j 4

3.1.3 增1和减1运算符

Postfix Example:

`j = i++ - 2;`

similar to

→ `j = i - 2;`

→ `i = i + 1;`

i	6
j	3

3.1.3增1和减1运算符

```
int a = 3;  
printf("%d", -a++);
```

similar to

→

```
printf("%d", -a);
```

→

```
a = a + 1;
```

a 4

- 良好的程序设计风格提倡：在一行语句中，一个变量只出现一次增1或减1运算

* 问题：过多的增1和减1运算混合会产生什么结果？

计算圆的周长和面积

```
#include <stdio.h>
main()
{
    printf("area = %f\n", 3.14159*5.3*5.3);
    printf("circumference = %f\n", 2*3.14159*5.3);
}
```

- 在程序中直接使用的常数，称为**幻数(Magic Number)**
- 问题：使用幻数存在什么问题？
 - * 程序的可读性变差，容易发生书写错误
 - * 当常数需要改变时，要修改所有引用它的代码，还可能有遗漏
- 问题：如何避免在程序中使用幻数？
 - * 把幻数定义为常量（**宏常量、const常量.....**）



3.2宏常量与宏替换

- **宏常量 (Macro Constant)**
 - * 也称**符号常量 (Symbolic Constant)**
- **宏定义**

`#define` **标识符** **字符串**

`#define` **PI** **3.14159**

- * **标识符——宏名 (Macro Name)**
- * **一般采用全大写字母表示**

计算圆的周长和面积

```
#include <stdio.h>
#define PI 3.14159
#define R 5.3
```

```
main()
{
    printf("area = %f\n", PI * R * R);
    printf("circumference = %f\n", 2 * PI * R);
}
```

宏定义是
语句吗？




相当于执行

```
#include <stdio.h>
main()
{
    printf("area = %f\n", 3.14159 * 5.3 * 5.3);
    printf("circumference = %f\n", 2 * 3.14159 * 5.3);
}
```

宏替换

计算圆的周长和面积

```
#include <stdio.h>
#define PI 3.14159;
#define R 5.3;
main()
{
    printf("area = %f\n", PI * R * R);
    printf("circumference = %f\n", 2 * PI * R);
}
```



相当于执行

```
#include <stdio.h>
main()
{
    printf("area = %f\n", 3.14159 * 5.3 * 5.3);
    printf("circumference = %f\n", 2 * 3.14159 * 5.3);
}
```

语法错误

3.3 const常量

```
#include <stdio.h>
main()
{
    const double pi = 3.14159;
    const double r = 5.3;
    printf("area = %f\n", pi * r * r);
    printf("circumference = %f\n", 2 * pi * r);
}
```

- 问题：const常量与宏常量相比的优点是什么？
 - const常量有数据类型
 - 某些集成化调试工具可以对const常量进行调试

3.4 自动类型转换与强制类型转换

■ 算术表达式中

- * 问题：相同类型数据的运算结果的类型是什么？
- * 还是该类型
- * 例如，整数除法

$$11 / 5 = 2$$



3.4 自动类型转换与强制类型转换

■ 算术表达式中

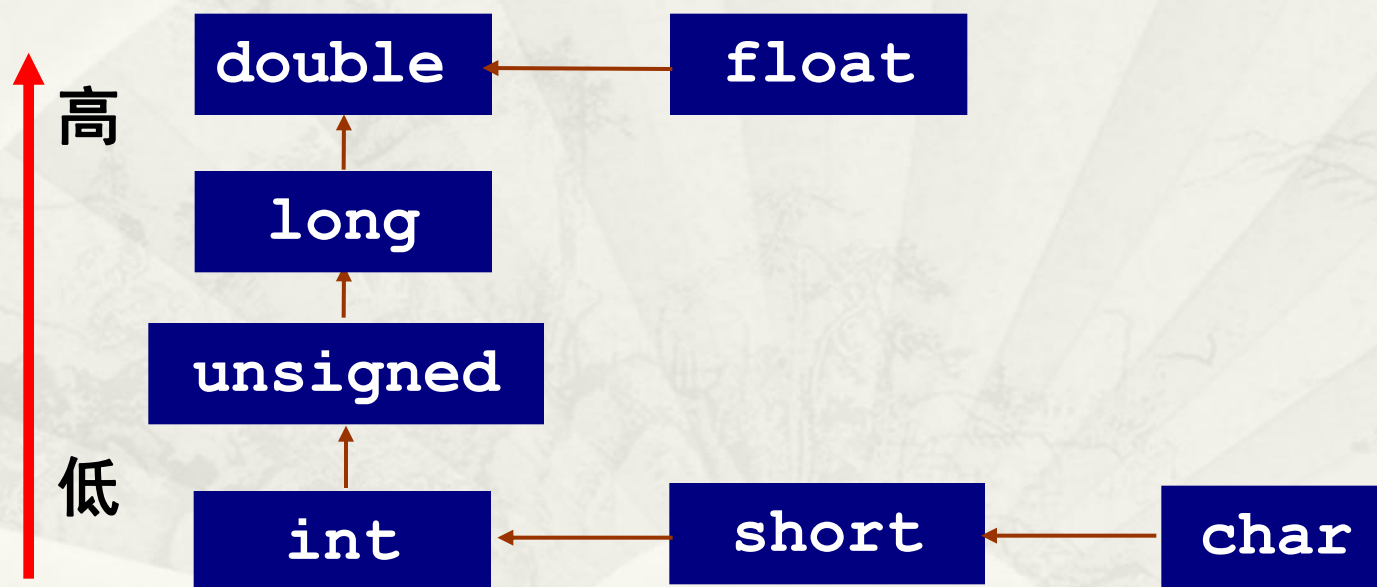
- * 问题：不同类型数据的运算结果的类型是什么？
- * 取值范围较大的那种类型
- * C编译器将所有操作数都转换成占内存字节数最大的操作数的类型，称为类型提升（Type Promotion）
- * 例如，浮点数除法

$$11.0 / 5 = 11.0 / 5.0 = 2.2$$



3.4 自动类型转换与强制类型转换

- 问题：类型提升的规则是什么？为什么这样设计？



3.4 自动类型转换与强制类型转换

- 把表达式的值转为任意类型——**类型强转 (Casting)**
(类型) 表达式

Example:

```
int    x = 10;
```

```
float  y;
```

```
y = (float)x;
```

→ (float)10

→ 10.000000

不改变x

x

10

y

10.000000

3.4 自动类型转换与强制类型转换

整除的结果还是整数，
不是浮点数

Example:

```
int total, number;  
float average;
```

...

```
average = total / number;
```

→ 15 / 2

→ 7

total 15

number 2

average 7.000000

3.4 自动类型转换与强制类型转换

Example:

```
int total, number;  
float average;
```

...

```
average = (float) total / number;
```

total 15

→ 15.000000 / 2

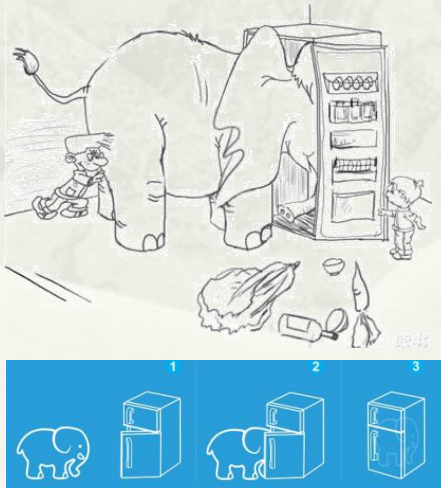
number 2

→ 7.500000

average 7.500000

**3.4 自动类型转换与强制类型转换

- 问题：在不同类型数据间赋值，是安全的吗？
 - * 取值范围小的类型赋值给取值范围大的类型是安全的，反之是不安全的
 - * 数值溢出(Overflow)
 - * 把大象放到冰箱里



数据类型	所占字节数 (bytes)	取值范围
char		
signed char	1	-128~127
unsigned char	1	0~255
short int		
signed short int	2	-32768~32767
unsigned short int	2	0~65535
unsigned int	4	0~4294967295
int		
signed int	4	-2147483648~2147483647
unsigned long int	4	0~4294967295
long int		
signed long int	4	-2147483648~2147483647
float	4	$3.4 \times 10^{-38} \sim 3.4 \times 10^{38}$
double	8	$-1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$
long double	10	$-1.2 \times 10^{-4932} \sim 1.2 \times 10^{4932}$

3.4 自动类型转换与强制类型转换

```
#include <stdio.h>
int main()
{
    short a;
    int b = 65537;
    a = b;
    printf("%d,%d\n", a, b);
    return 0;
}
```

1, 65537

a

0000 0000 0000 0001

b

0000 0000 0000 0001 0000 0000 0000 0001

1

```
#include <stdio.h>
int main()
{
    short a;
    int b = 32768;
    a = b;
    printf("%d,%d\n", a, b);
    return 0;
}
```

-32768, 32768

a

1000 0000 0000 0000

-32768

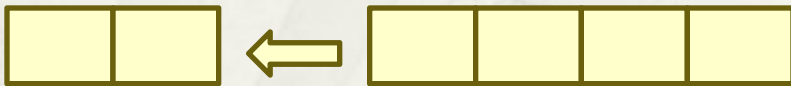
b

0000 0000 0000 0000 1000 0000 0000 0000

32768

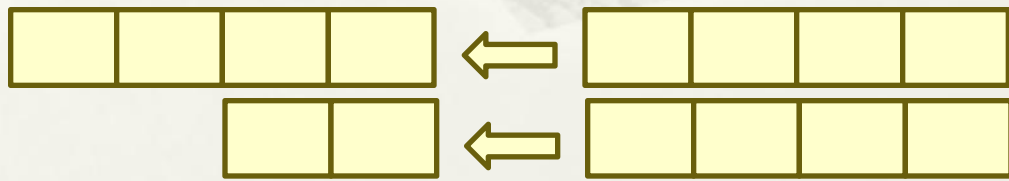
3.4 自动类型转换与强制类型转换

- 问题：从高精度向低精度转换时，损失什么信息？
 - * 因低精度的数据位数比高精度的少，容纳不下高精度的所有信息，就会丢失信息，出现舍入（Round），也称截断（Truncation）



```
warning: '=': conversion from 'float' to 'int', possible loss of data
```

- * 浮点数转为整数，会丢失小数部分，某些情况下整数部分精度也会损失

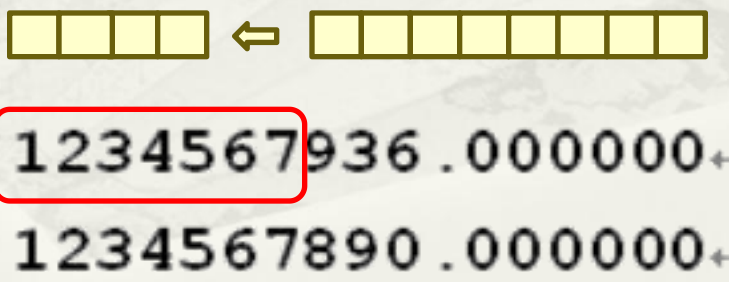


- 问题：双精度浮点数转为单精度，结果会怎样？
 - 可能因有效数字位数不够而出现精度损失
 - 尾数所占位数决定实数的精度，不同系统下实数精度不同
 - Visual C++中，float类型的有效位只有6~7位，double类型的有效位只有15~16位，有效位后的数字不精确

```
#include <stdio.h>
main()
{
    float    a = 123456.789e4;
    double   b = 123456.789e4;
    printf("%f\n%f\n", a, b);
}
```

double常量

【例 2.3】



warning: 'initializing': truncation from 'const double' to 'float'

```
*C:\Users\908pc\Desktop\c\Debug...  
1! = 1  
2! = 2  
3! = 6  
4! = 24  
5! = 120  
6! = 720  
7! = 5040  
8! = 40320  
9! = 362880  
10! = 3628800  
11! = 39916800  
12! = 479001600  
13! = 6227020800  
14! = 87178291200  
15! = 1307674368000  
16! = 20922789888000  
17! = 355687428096000  
18! = 6402373705728000  
19! = 121645100408832000  
20! = 2432902008176640000  
21! = 51090942171709440000  
22! = 112400072777607680000  
23! = 25852016738884976640000  
24! = 620448401733239439360000  
25! = 1551121004330985984000000  
26! = 403291461126605635584000000  
27! = 10888869450418352160768000000  
28! = 304888344611713860501504000000  
29! = 8841761993739701954543616000000  
30! = 265252859812191058636308480000000
```

```
*C:\Users\908pc\Desktop\c\Debug\mai...  
1! = 1  
2! = 2  
3! = 6  
4! = 24  
5! = 120  
6! = 720  
7! = 5040  
8! = 40320  
9! = 362880  
10! = 3628800  
11! = 39916800  
12! = 479001600  
13! = 6227020800  
14! = 87178291200  
15! = 1307674368000  
16! = 20922789888000  
17! = 355687428096000  
18! = 6402373705728000  
19! = 121645100408832000  
20! = 2432902008176640000  
21! = 51090942171709440000  
22! = 112400072777607700000  
23! = 25852016738884978000000  
24! = 620448401733239410000000  
25! = 1551121004330986000000000  
26! = 40329146112660565000000000  
27! = 1088886945041835200000000000  
28! = 30488834461171384000000000000  
29! = 884176199373970080000000000000  
30! = 26525285981219103000000000000000
```


3.5常用的标准数学函数

■ `#include <math.h>`

函 数 名	功 能
<code>sqrt(x)</code>	计算 x 的平方根, x 应大于等于 0
<code>fabs(x)</code>	计算 x 的绝对值
<code>log(x)</code>	计算 $\ln x$ 的值, x 应大于 0
<code>log10(x)</code>	计算 $\lg x$ 的值, x 应大于 0

函 数 名	功 能
<code>exp(x)</code>	计算 e^x 的值
<code>pow(x,y)</code>	计算 x^y 的值
<code>sin(x)</code>	计算 $\sin x$ 的值, x 为弧度值, 而非角度值
<code>cos(x)</code>	计算 $\cos x$ 的值, x 为弧度值, 而非角度值

如何进行更复杂的数学运算呢?



【例3.8】计算三角形面积

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

$$s = \frac{1}{2}(a + b + c)$$

假设a,b,c定义为int型

```
area = sqrt(s * (s - a) * (s - b) * (s - c))
```

```
area = sqrt(s(s-a)(s-b)(s-c))
```

```
s = 0.5 * (a + b + c)
```

```
s = 1.0/2 * (a + b + c)
```

```
s = (a + b + c) / 2.0
```

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

```
s = 1/2 * (a + b + c)
```

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



【例3.8】计算三角形面积

```
1  #include <stdio.h>
2  #include <math.h>
3  main()
4  {
5      float a, b, c, s, area;
6      printf("Input a,b,c:");
7      scanf("%f,%f,%f", &a, &b, &c);
8      s = (float)(a + b + c) / 2;
9      area = sqrt(s * (s - a) * (s - b) * (s - c));
10     printf("area = %f\n", area);
11 }
```

Input a,b,c:3,4,5✓

area = 6.000000

问题：若输入的三边不能构成三角形，那么会怎样？



小结

- 简单的算术运算
 - * 算术运算符
 - * 增 1 和减 1 运算符
 - * 类型强转运算符
- 较为复杂的数学运算
 - * 常用的标准数学函数
- 宏常量与 `const` 常量



作业

- 第3章 3.1-3.4, 背表3-5（就是下面这个）
- 第四章（自学）作业4.2, 4.3;

函 数 名	功 能
sqrt(x)	计算 x 的平方根, x 应大于等于 0
fabs(x)	计算 x 的绝对值
log(x)	计算 $\ln x$ 的值, x 应大于 0
log10(x)	计算 $\lg x$ 的值, x 应大于 0

函 数 名	功 能
exp(x)	计算 e^x 的值
pow(x,y)	计算 x^y 的值
sin(x)	计算 $\sin x$ 的值, x 为弧度值, 而非角度值
cos(x)	计算 $\cos x$ 的值, x 为弧度值, 而非角度值

常见错误实例	使用力
$1.0/2.0+[a-b]/(a+b)$	表达式运算顺序
$\sin x$	使用数学函数运算时,未将参数用圆括号括起来 且未注意其定义域要求和参数的单位
$3.5 \% 0.5$	对浮点数执行求余运算
$1/2$	误将浮点数除法当做整数除法
$\text{float}(m)/2$	强转表达式中的类型名未用圆括号括起来
—	误以为 $(\text{float})m$ 这种强制运算可以改变变量 的类型和数值
—	误以为用双引号括起来的字符串中与宏名相同的 字符也被宏替换,误以为宏替换时可以做语法检查
—	误以为三角函数中的角的单位是角度
$\#define \text{PI} = 3.14159;$	将宏定义当做 C 语句来使用,在行末加上了分号, 或者在宏后加上了“=”
$+=, -=, *=, /=, \%=$	将复合的赋值运算符 $+=, -=, *=, /=, \%=$ 的两个字 符中间加入了空格
$(a+b)++$	对一个算术表达式使用增 1 或者减 1 运算

习 题 3

3.1 分析并写出下列程序的运行结果。

(1)

```

1  #include <stdio.h>
2  main()
3  {
4      int a = 12, b=3;
5      float x = 18.5, y = 4.6;
6      printf("%f\n", (float)(a*b) / 2);
7      printf("%d\n", (int)x *(int)y);
8  }
```

(2)

```

1  #include <stdio.h>
2  main()
3  {
```



```

4   int x = 32, y = 81, p, q;
5   p = x++;
6   q = --y;
7   printf("%d %d\n", p, q);
8   printf("%d %d\n", x, y);
9   }

```

3.2 参考例 3.1 程序,从键盘任意输入一个 3 位整数,编程计算并输出它的逆序数(忽略整数前的正负号)。例如,输入 -123,则忽略负号,由 123 分离出其百位 1、十位 2、个位 3,然后计算 $3 * 100 + 2 * 10 + 1 = 321$,并输出 321。

3.3 设银行定期存款的年利率 *rate* 为 2.25%,已知存款期为 *n* 年,存款本金为 *capital* 元,试编程计算并输出 *n* 年后的本利之和 *deposit*。

3.4 编程计算并输出一元二次方程 $ax^2 + bx + c = 0$ 的两个实根, $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, 其中 *a*、*b*、*c* 的值由用户从键盘输入,假设 *a*、*b*、*c* 的值能保证方程有两个不相等的实根(即 $b^2 - 4ac > 0$)。

本章实验题

参考例 3.4 和例 3.5 程序,分别使用宏定义和 `const` 常量定义 π 的值,编程计算并输出球的体积和表面积,球的半径 *r* 的值由用户从键盘输入。

实验目的:熟悉简单的算术运算、宏定义和 `const` 常量的使用。

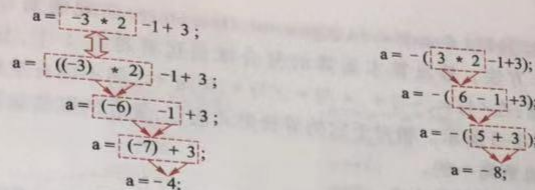


图 3-2 算术混合运算示意图

【例 3.1】计算并输出一个三位整数的个位、十位和百位数字之和。

【问题求解方法分析】要计算一个三位整数的个位、十位和百位数字值，首先必须从一个三位整数中分离出它的个位、十位和百位数字，而巧妙利用整数除法和求余运算可以解决这个问题。

例如，整数 153 的个位、十位和百位数字应该分别是 3、5、1。其中，个位数字 3 刚好是 153 对 10 求余的余数，即 $153 \% 10 = 3$ ，因此可用对 10 求余的方法求出个位数字 3；百位数字 1 说明在 153 中只有 1 个 100，由于在 C 语言中整数除法的结果仍为整数，即 $153 / 100 = 1$ ，因此可用对 100 整除的方法求得百位数字；中间的十位数字既可通过将其变换为最高位后再对 10 整除的方法得到，即 $(153 - 1 * 100) / 10 = 53 / 10 = 5$ ，也可通过将其变换为最低位再对 10 求余的方法得到，即 $(153 / 10) \% 10 = 15 \% 10 = 5$ 。根据上述分析，可编写程序如下：

```
1 #include <stdio.h>
2 main()
3 {
4     int x = 153, b0, b1, b2, sum;
5     b2 = x / 100;           /* 计算百位数字 */
6     b1 = (x - b2 * 100) / 10; /* 计算十位数字 */
7     b0 = x % 10;           /* 计算个位数字 */
8     sum = b2 + b1 + b0;
9     printf("b2 = %d, b1 = %d, b0 = %d, sum = %d\n", b2, b1, b0, sum);
10 }
```

程序的运行结果如下：

b2 = 1, b1 = 5, b0 = 3, sum = 9

由于算术运算符 *、/、% 的优先级高于 +、-，因此为了保证减法运算先于除法运算，程序第 6 行语句中的圆括号是必不可少的。

【思考题】本例程序还可以利用 $b0 = x - b2 * 100 - b1 * 10$ ；来计算个位数字 b0，请重新编写例 3.1 的程序，观察运行结果，并分析其原理。