西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

赋值和类型转换

1、赋值和类型转换

2、隐式类型转换

2.4.8 赋值运算符

表2-10 赋值运算符

运算符	功能	目	结合性	用法
=	赋值	双目	自右向左	lvalue = expr
+= -= *= /= %=	复合赋值	双目	自右向左	lvalue+=expr lvalue-=expr lvalue*=expr lvalue/=expr lvalue%=expr
&= ^= |= <<= >>=				lvalue&=expr lvalue^=expr lvalue|=expr lvalue<<=expr lvalue>>=expr

赋值运算符的优先级在所有运算符中较低，仅高于逗号运算符。其作用是将运算符右侧的expr表达式的值赋值给左侧的lvalue变量，并且将该值作为整个表达式的值。

2.4.8 赋值运算符

表2-11 复合赋值运算符的含义

复合赋值	等价表达式	含义
lvalue+=expr	lvalue=lvalue+(expr)	先执行加法运算，然后执行赋值
lvalue-=expr	lvalue=lvalue-(expr)	先执行减法运算，然后执行赋值
lvalue*=expr	lvalue=lvalue*(expr)	先执行乘法运算，然后执行赋值
lvalue/=expr	lvalue=lvalue/(expr)	先执行除法运算，然后执行赋值
lvalue%=expr	lvalue=lvalue%(expr)	先执行求余运算，然后执行赋值
lvalue&=expr	lvalue=lvalue&(expr)	先执行按位与运算，然后执行赋值
lvalue^=expr	lvalue=lvalue^(expr)	先执行按位异或运算，然后执行赋值
lvalue|=expr	lvalue=lvalue|(expr)	先执行按位或运算，然后执行赋值
lvalue<<=expr	lvalue=lvalue<<(expr)	先执行左移位运算，然后执行赋值
lvalue>>=expr	lvalue=lvalue>>(expr)	先执行右移位运算，然后执行赋值

2.4.8 赋值运算符

- 示例

```
int a=6,c=10,m=21,n=32;
a=a-1; //正确，a减1后再赋值给a，等价于--a
c*=m+n; //正确，等价于c=c*(m+n)，不是c=c*m+n
```

2.4.8 赋值运算符

- （1）赋值运算符与运算对象构成的式子称为赋值表达式。赋值表达式除完成赋值功能外，它本身也可以当作一个普通的表达式项参与运算。
- 示例

```
int a,b,c,m=25;
c = (a=12) % (b=5);
//正确，运算结果a为12，b为5，c为2，等价于a=12,b=5,c=a%b
m = m+ = m*= m-=15; //正确，运算结果m为200
```

- 但表达式中包含过多的赋值表达式时会降低程序的可读性，理解困难，故不宜提倡。

2.4.8 赋值运算符

- （2）赋值运算符要求运算对象expr的类型应与lvalue类型相同，如果不相同会自动将expr的类型转换成lvalue的类型再赋值。转换过程中可能会产生精度丢失、数据错误等。
- 示例

```
char a;
a = 4.2; //a为4，精度丢失，发生在浮点型转换成整型时
a = 400; //a为-112，数据错误，发生在数据溢出时
```

2.4.8 赋值运算符

- 2．左值与右值
- **左值**（lvalue）是指可以出现在赋值表达式左边或右边的表达式
- **右值**（rvalue）是指只能出现在赋值表达式右边而不能出现在左边的表达式。

2.4.8 赋值运算符

- C语言中只有变量、下标运算（[ ]）和间接引用运算（*）才能作为左值，下标运算和间接引用运算将在后面章节讲到，在此之前我们可以将左值简单地理解为变量。

2.4.8 赋值运算符

- 常量、const变量、表达式、函数调用均不能作为左值，而右值可以是常量、变量、函数调用或表达式。
- 示例

```
int k=95,a=6,b=101;
const int n=6;
b-a=k;    //错误，区别b-a==k的写法为合法的关系运算
5=b-a;    //错误
n=b-a*k; //错误
```

2.5 类型转换

- C语言表达式是否合法，以及合法表达式的含义是由运算对象的数据类型决定的。不同类型的数据混合运算时需要进行类型转换（conversion），即将不同类型的数据转换成相同类型的数据后再进行计算。

- 类型转换有两种：隐式类型转换和显式类型转换。

2.5.1 隐式类型转换

- 隐式类型转换（implicit type conversion）又称自动类型转换，它是由编译器自动进行的。

- 编译器根据需要，在算术运算、赋值、函数调用过程中将一种数据类型的数据自动转换成另一种数据类型的数据。

2.5.1 隐式类型转换

- 1. 何时进行隐式类型转换
- 编译器在必要时将类型转换规则应用到C语言内置数据类型的对象上，在下列情况下，将发生隐式类型转换。

2.5.1 隐式类型转换

- （1）在混合类型的算术运算、比较运算、逻辑运算表达式中，运算对象被转换成相同的数据类型。如：

```
int m=10,n=20;
m=m*1.5+n*2.7; //发生隐式类型转换
m+n>30.5       //发生隐式类型转换
```

2.5.1 隐式类型转换

- （2）用表达式初始化变量时，或赋值给变量时，该表达式被转换为该变量的数据类型。如：

```
int x=3.1415926; //发生隐式类型转换
x=7.8;           //发生隐式类型转换
```

2.5.1 隐式类型转换

- （3）调用函数的实参被转换为函数形参的数据类型。如：

```
c=max(5.6,6.7); //若函数原型参数为整型，则发生隐式类型转换
```

2.5.1 隐式类型转换

- 2．混合运算中的隐式类型转换
- 在表达式中经常有不同类型数据之间的运算，称为混合运算。
- 如：

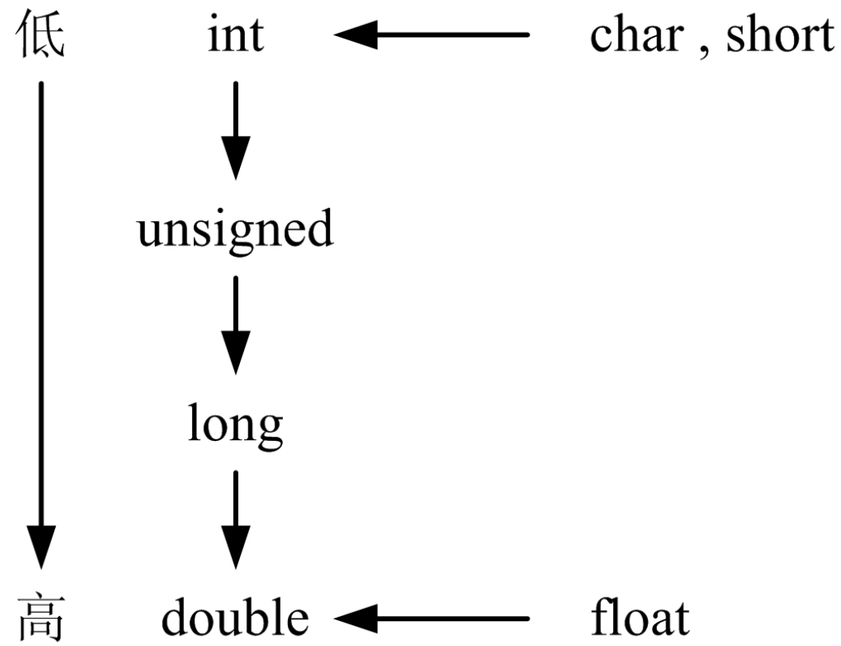
```
10 + 'a' + 150 / 1.5 - 3.7 * 'b'
```

2.5.1 隐式类型转换

- 算术运算中，如果运算符的两个运算对象是不同的类型，C语言会在计算表达式之前自动转换成同一种类型后才进行运算。
- 转换的规则按“存储空间提升原则”进行，即存储空间小的类型转换成存储空间大的类型，或精度低的类型转换成精度高的类型，以保证运算结果尽可能精确。

2.5.1 隐式类型转换

- 如图所示，数值型数据间的混合运算规则为：



（a）

2.5.1 隐式类型转换

- ①整型数据中字符型（char）和短整型（short）转换成基本整型（int），基本整型（int）转换成长整型（long），有符号（signed）转换成无符号（unsigned）；
- ②浮点型数据中单精度（float）转换成双精度（double）；
- ③整型数据与浮点型数据运算时，都转换成双精度（double）；
- ④类型转换是按步骤执行的，即运算到哪步就转换哪步。

2.5.1 隐式类型转换

例题分析

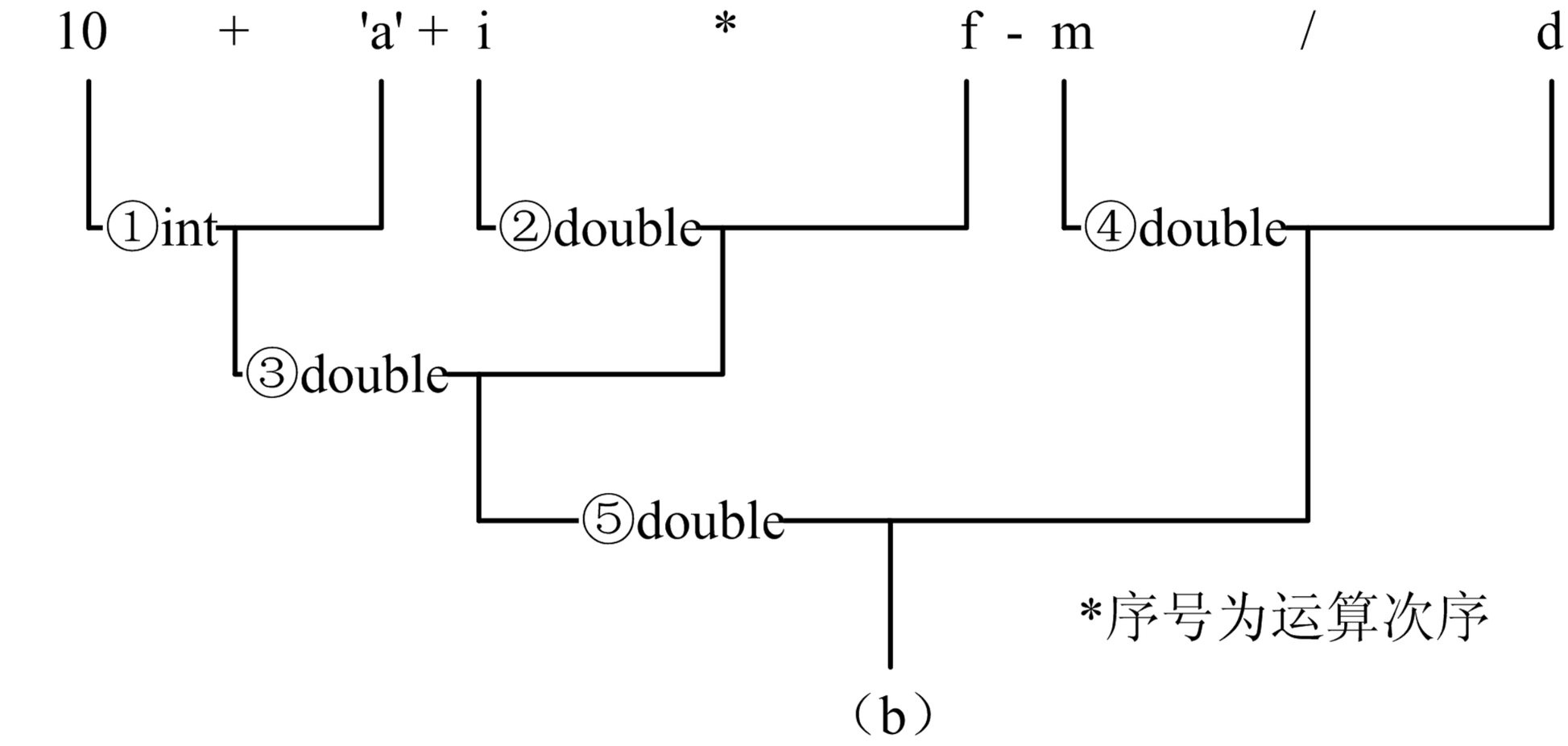
已知

int i; float f; double d; long m;

表达式32 + 'A' + i * f - m / d的运算步骤

2.5.1 隐式类型转换

图2.4 混合运算类型转换



2.5.1 隐式类型转换

- 3．位运算时的类型转换
- 如果两个不同类型的数据进行位运算时，会按“存储空间提升原则”进行隐式类型转换，即按右端对齐长度小的数据左边进行扩展。当此数为正数时，左边扩展位补满0；当此数为负数时，左边扩展位补满1；如果是无符号整数，左边扩展位补满0。

2.5.1 隐式类型转换

- 例如：已知

```
unsigned short A=7; //A是两个字节，16位
unsigned char B=9;  //B是一个字节，8位
char C=-7 ;         //C是一个字节，8位
```

- A&B的运算过程为：

00000000 00000111 …… A=7

& 00000000 00001001 …… B=9（符号扩展）

00000000 00000001 …… 1

2.5.1 隐式类型转换

- A|C的运算过程为：

00000000 00000111 …… A=7

11111111 11111001 …… C=-7（符号扩展）

11111111 11111111 …… 65535


程序设计