西北工業大學

NORTHWESTERN POLYTECHNICAL UNIVERSITY

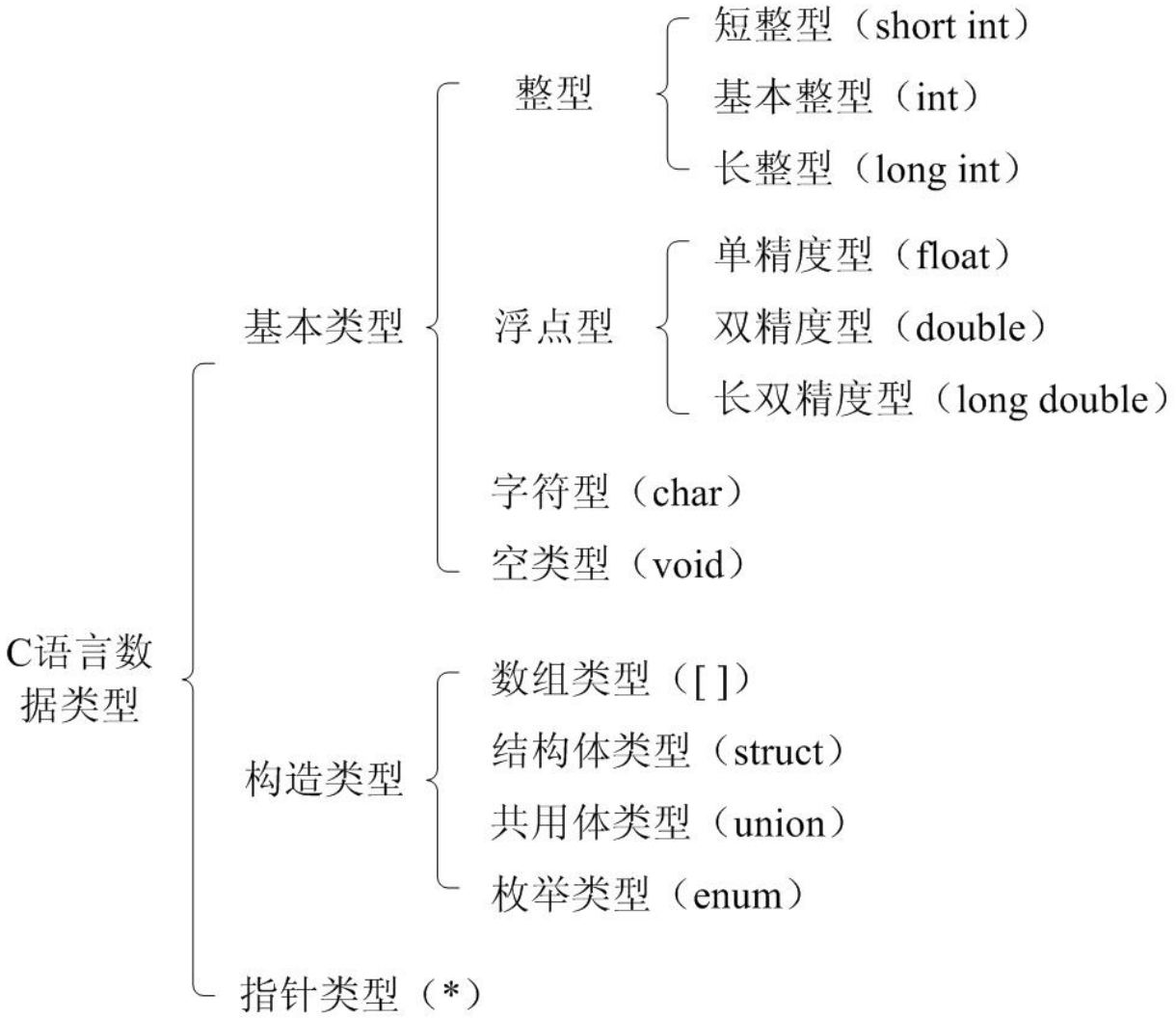
程序设计基础
Programming in C++

U10G13027/U10G13015

主讲：魏英，计算机学院

- 2.1 数据类型
- 2.2 常量
- 2.3 变量
- 2.4 运算符与表达式
- 2.5 类型转换

C++语言内置数据类型


C语言数据类型
基本类型
整型
短整型（short int）
基本整型（int）
长整型（long int）
浮点型
单精度型（float）
双精度型（double）
长双精度型（long double）
字符型（char）
空类型（void）
构造类型
数组类型（[ ]）
结构体类型（struct）
共用体类型（union）
枚举类型（enum）
指针类型（*）

2.1 数据类型

表2-1 基本类型数据的内存长度和数值范围

类型	类型标识符	内存长度	数值范围	精度
整型	[signed] int	4	-2147483648～+2147483647	
无符号整型	unsigned [int]	4	0～4294967295	
短整型	[signed] short [int]	2	-32768～+32767	
无符号短整型	unsigned short [int]	2	0～65535	
长整型	[signed] long [int]	4	-2147483648～+2147483647	
无符号长整型	unsigned long [int]	4	0～4294967295	
字符型	[signed] char	1	-128～+127	
无符号字符型	unsigned char	1	0～255	
单精度型	float	4	3.4×10^{-38}～3.4×10^{38}	7
双精度型	double	8	1.7×10^{-308}～1.7×10^{308}	16
长双精度型	long double	同上/12	同上/	19

2.1.1 整型

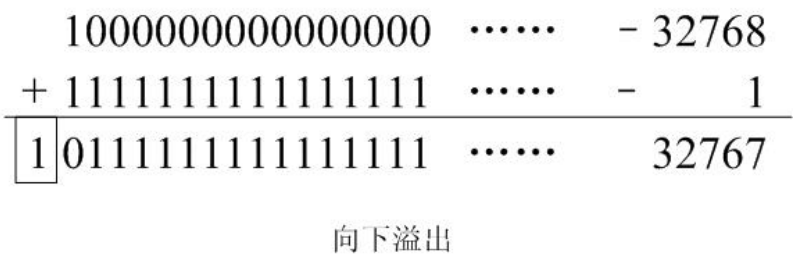
- C++语言整型分为长整型（long int）、基本整型（int）和短整型（short int）
- long int可以简写为long、short int可以简写为short。
- short ≤ int ≤ long

2.1.1 整型

- 不同的数据类型规定了不同的机器数长度，决定了对应数据的数值范围，当一个整数超出此范围时计算机会将其转换为在数值范围内所允许的一个数，称为整型数据的溢出处理。

- 一般地，超过最大值的有符号整型数值会向上溢出变成负数，超过最小值的数据会向下溢出变成正数。

图2.2 short型的溢出



2.1.2 浮点型

- C++语言浮点型又称实型，分为单精度（float）、双精度（double）和长双精度（long double）3种。
- 在VC中规定float型在内存中占用4个字节，提供7位有效数字；
- double型和long double型在内存中占用8个字节，提供16位有效数字。

2.1.2 浮点型

例2.1

```
a=111111.1093750000000000 , b=111111.1093750000000000
c=111111.1110067800000000 , d=111111.1110012300100000
e=1000000000000000110000.000000
```

```
1  #include <iostre
2  #include <iomani
3  using namespace
4  int main()
5  {
6      float  a=0.00000678f, b=0.00000123f;
7      double c=0.00000678 , d=0.00000123;
       double e=10000000000000000000.0;
8      a = a + 111111.111f;
9      b = b + 111111.111f;
10     c = c + 111111.111;
11     d = d + 111111.111;
12     e = e + 111111.111;
13     cout<<setiosflags(ios::fixed)<<setprecision(16);
14     cout<< “a=” <<a<< “,b=” <<b<<endl;
15     cout<< “c=” <<c<< “,d=” <<d<<endl;
16     cout<< “e=” <<e<<endl;
17     return 0;
18 }
```

2.1.2 浮点型

- 因为浮点型数据长度和精度是有限的，所以浮点数存在计算误差。虽然浮点数精度越高计算结果越精确，但其处理时间也长。

- 一个较大的浮点数与一个很小的浮点数做加法时，由于精度限制使得很小的浮点数被忽略了，从而使得这样的加法无意义。

2.1.3 字符型

- C++语言字符型分为有符号（signed char）和无符号（unsigned char）两种，其中signed书写时可以省略。

- 字符型数据在内存中占用1个字节，采用二进制形式存储。

2.1.3 字符型

- 字符型数据可以存储单字节字符，如ASCII码，此时在内存中的数据是字符的ASCII码值。例如字符'A'在内存中的存储形式为

0	1	0	0	0	0	0	1

'A'的ASCII码值

- 在C++语言中字符型数据和整型数据之间可以通用。一个字符数据可以赋给整型变量，一个整型数据也可以赋给字符型变量，还可以对字符型数据进行算术运算。

- 一般地，单字节字符和小范围的整型，如月份、日期、逻辑值、性别等使用字符型。

2.1.3 字符型

例2.2

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int  i, j;
6      char c1,c2;
7      c1='a';      //字符数据赋值给字符型
8      c2=98;       //整数数据赋值给字符型
9      i='A';       //字符数据赋值给整型
10    j=66;        //整数数据赋值给整型
11     cout<< "i=" <<i<< ",j=" <<j<< ",c1=" <<c1<< ",c2=" <<c2<<endl;
12     cout<< "c1-32=" <<c1-32<<endl;     //字符型可以进行减法运算
13     return 0;
14 }
```

```
i=65 , j=66 , c1=a , c2=b
c1-32=65
```

- 常量（constant）是指程序中其值不能被修改的数据，分为**字面常量**和**符号常量**。
- 从字面形式即可识别的常量称为字面常量（literal constant），例如64、3.1415926和‘A’等。
- 每个字面常量都具有数据类型，由它的书写形式和值来决定。

- 一个整型常量可以用3种不同的方式表示：
  - （1）十进制整数。
    - 以非零十进制数1～9组成的整数，例如13579,-24680等。
  - （2）八进制整数。
    - 以0开头和八进制数0～7组成的整数，例如0，012，0177等。
  - （3）十六进制整数。
    - 以0x或0X开头和十六进制数0～9、A～F或a～f组成的整数（字母大小写任意），例如0x1234，0xab，0xCF等。

2.2.2 浮点型常量

- 一个浮点型常量可以用两种不同的方式表示：
  - （1）小数形式。由小数点、十进制整数和小数组成的浮点数。整数和小数可以省略其中之一，但不能省略小数点。

    如： .123、123. 、0.0等。
  - （2）指数形式，又称科学记数法表示。以f E n或 f e n格式组成的浮点数，其中E或e表示以10为低的幂，n为指数且必须是整型，f可以是整数或小数。

    如： 0.31415926e+1、314.15926E-2等。
  - 浮点型常量默认为double型。若在浮点数后面加一个字母f或F，则它是float型。

1. 用字面常量表示字符常量

- 以一对单引号（ ‘ ’ ）括起来的一个字符表示字符常量，如： ‘A’、 ‘0’、 ‘&’、’ab’ 错误！
- 字符常量表示的是一个字符，存储的是该字符的ASCII码值。例如 ‘A’表示英文字符A，数据值是65； ‘2’表示数字字符2，数据值是50。
- 注意区别字符’2’和整数2的写法。

2. 用转义字符表示字符常量

- 以反斜线（\）开头，后跟一个或几个字符序列表示的字符称为转义字符，如\n表示换行符。
- 采用转义字符可以表示ASCII字符集中不可打印或不方便输入的控制字符和其他特定功能的字符。

2.2.3 字符常量

表2-2 转义字符及其含义

转义字符形式	含 义	ASCII码值
\a	响铃符	7
\b	退格符	8
\f	进纸符，将光标位置移到下页开头	12
\n	换行符，将光标位置移到下一行开头	10
\r	回车符，将光标位置移到本行开头	13
\t	水平制表符，光标跳到下一个TAB位置	9
\v	垂直制表符	11
\'	单引号	39
\"	双引号	34
\\	反斜线	92
\?	问号	63
\0	空字符	0
\ooo	用1～3位八进制数ooo为码值所对应的字符	ooo（八进制）
\xhh	用1～2位十六进制数hh为码值所对应的字符	hh（十六进制）

\ooo和\xhh称为通用转义字符，其中ooo表示可以用1至3位八进制数作为码值表示一个ASCII字符，hh表示可以用1至2位十六进制数作为码值表示ASCII字符。

如：\1234 ➡ “\123和4”， \128➡ “\12和8”，\19 ➡ “\1和9”，

\9 错误！ ，\0xFE 错误！

由于字符型数据在内存中只占用1个字节，即使按无符号处理其最大值也仅是255（八进制为377），因此ooo的数值范围为0～377（八进制），其他值使得字符型数据溢出。同理，hh的数值范围为0～FF。

2.2.3 字符常量

例2.3

```
1  #include<iostream>
2  using namespace std;
3  int main ( )
4  {   cout<<“ab c\t  de\rf\tg”<<endl;
5      cout<<“h\ti\b\bj k\n123\’\”\\\x41\102CDE”<<endl;
6      return 0;
7  }
```

```
f␣ ␣ ␣ ␣ ␣ ␣ ␣gde
h␣ ␣ ␣ ␣ ␣ ␣ j ␣k
123'"\ABCDE
```

2.2.4 字符串常量

- 以一对双引号（“ ”）括起来的零个或多个字符组成的字符序列称为字符串常量，ASCII字符集或多字节字符集（如汉字、日韩文字等）都可以组成字符串。

- 双引号是字符串常量的边界符，不是字符串的一部分，如果在字符串中要出现双引号应使用转义字符（\“）。

```
""    //空字符串（0个字符）
" "   //包含一个空格的字符串
"Hello,World\n"    //包含Hello,World 和 换行符的字符串
"xyz\101\x42"    //包含x y z AB的字符串
"\\\'\"\n"    //包含反斜线（\\） 单引号（\'）和双引号（\"）的字符串
"\"a/b\" isn\'t a\\b"    //字符串"a/b" isn't a\b
```

2.2.5 符号常量

符号常量定义形式为：

```
#define 标识符 常量
```

其中#define是宏定义命令，作用是将标识符定义为常量值，在程序中所有出现该标识符的地方均用常量替换。

2.2.5 符号常量

例2.4

```
1  #include<iostream>
2  using namespace std;
3  #define PI 3.1415926     //3.1415926即为圆周率π
4  int main()
5  {
6      double r=5.0;
7      cout<< “L=” <<2*PI*r<< “,S=” <<PI*r*r<<endl;     //PI替换为3.1415926
8      return 0;
9  }
```

在程序运行期间其值可以改变的量称为变量（variable）。

变量实际上就是计算机中的一个内存单元。

2.3.1 变量的概念

C++语言规定变量应该有一个名字，用变量名代表内存单元。

C++语言通过定义变量时指定其数据类型来确定内存单元的大小，不同的数据类型有不同的数据形式和存储形式，需要一定数量（单位为字节）的内存单元。

C++语言变量必须“**先定义，后使用**”，定义变量的一般形式是

变量类型 变量名列表;

```
double a , b , c , d; //定义实型变量
int i , j , k; //一次定义多个int型变量
int a, char c; //错误
```

2.3.3 使用变量

- 变量定义后，变量值是未确定的（除了第4章的静态存储情形），即变量值是随机的。直接使用此时的变量参与运算，运算结果也是随机的。

- 在变量定义的同时给变量一个初值，称为变量初始化（initialized），一般形式为：

变量类型 变量名=初值;

变量类型 变量名1=初值1, 变量名2=初值2, … ;

2.3.3 使用变量

```
double pi=3.1415926; //正确，初始化pi为3.1415926
int x , y , k=10; //正确，可以只对部分变量初始化
int a=1 , b=1 , c=1; //正确，可以同时初始化多个变量
int d=a , e=a+b; //错误，初值不能是变量或表达式
```

2.3.3 使用变量

- 定义变量后，可以通过赋值语句为变量赋予新的数据，一般形式为：

变量名=表达式;

- 赋值后，无论变量原来的值是多少，都将被新值替代。

```
int k;
k=5; //给k赋值5
..... //k保持不变
k=10; //重新给k赋值10，k已改变不再是5
```

- 1. const限定
  - 在变量定义前加上const修饰，这样的变量称为**只读变量**（read-only variable）或**常变量**（constant variable）
  - 它在程序运行期间的值不能被修改。其定义的一般形式为

const 变量类型 变量名列表;

```
int x;
const int i=6 , j=10;
x=i+1; //正确，可以使用const变量
i=10; //错误，不可以给const变量赋值
j++; //错误，不可以修改const变量
```

2.3.5 类型限定

```
const int i=6; //正确
const int m; //错误
```

- const限定过的变量在编译过程中若发现有修改的操作时会报编译错误，从而“阻止”对变量的修改。
- 使用const限定强制实现对象最低访问权限，是现代软件开发的设计原则之一。

2.3.5 类型限定

例1.14

```
1  #include <iostream> //标准输入输出函数库
2  #include <cmath> //数学函数库
3  using namespace std;
4  double root(const double x, const double y) //root函数求x-y的平方根
5  {   if (x>=y) return sqrt(x-y); //只有在x大于等于y时计算x-y的平方根
6      else return 0; //否则返回0
7  }
8  int main() //主函数
9  {   double a, b; //定义两个浮点型变量
10     cin>>a>>b; //输入两个数
11     cout<<root(a,b)<<endl; //输出a-b的平方根
12     return 0; //主函数正常结束返回0
13 }
```

求解问题的基本处理是运算。

通过C++语言丰富的运算符及其表达式构成实现算法的基本步骤，在不同程序结构的控制下有机地组织在一起形成程序。

2.4.1 运算符与表达式的概念

- 1. 运算对象的数目
  - 运算符所连接的运算对象的数目称为**运算符的目**
  - （1）单目运算符（unary operator）
  - （2）双目运算符（binary operator）
  - （3）三目运算符（ternary operator）

- 2. 运算符的优先级
  - 同一个式子中不同的运算符进行计算时，其运算次序存在先后之分，称为运算符的优先级（precedence）。
  - 运算时先处理优先级高的运算符，再处理优先级低的运算符。

- 3. 运算符的结合性
  - 在一个式子中如果有两个以上同一优先级的运算符，其运算次序是按运算符的结合性（associativity）来处理的。
  - C++语言运算符分为左结合（方向）和右结合（方向）。
- 4. 表达式
  - 由运算符和运算对象组成的式子称为表达式（expression。

2.4.2 算术运算符

表2-3 算术运算符

运算符	功能	目	结合性	用法
+、-	取正值取负值	单目 单目	自右向左 自右向左	+expr、-expr
*、/、%	乘法除法整数求余/模数运算	双目 双目 双目	自左向右 自左向右 自左向右	expr1 * expr2、expr1 / expr2、expr1 % expr2
+、-	加法减法	双目 双目	自左向右 自左向右	expr1 + expr2、expr1 - expr2

```
35 % 6 //结果为5
35%7 //结果为0
8.5 % 3 //错误，因为求余运算符的两个操作数都必须是整数
```

【例2.5】 已知int x=1234，求x的千位、百位、十位、个位数。

解：

x/1000为千位数，x%10为个位数，x/10%10为十位数，x/100%10为百位数。

2.4.3 自增自减运算符

表2-4 自增自减运算符

运算符	功能	目	结合性	用法
++、--	后置自增后置自减	单目 单目	自右向左 自右向左	lvalue++、lvalue--
++、--	前置自增前置自减	单目 单目	自右向左 自右向左	++lvalue、--lvalue

```
int m=4, n;
① n = ++m; //m先增1，m为5，然后表达式使用m的值，赋值给n，n为5
② n = --m; //m先减1，m为4，然后表达式使用m的值，赋值给n，n为4
③ n = m++; //表达式先使用m的值，赋值给n，n为4，然后m增1，m为5
④ n = m--; //表达式先使用m的值，赋值给n，n为5，然后m减1，m为4
```

2.4.3 自增自减运算符

```
int n=4 , m=4;
n++; //运算后n为5
++m; //运算后m为5
const int k=6;
5++; //错误
--(a+b); //错误
k++; //错误
max(a,b)--; //错误
```

注意：自增自减运算符只能用于变量，而不能用于常量和表达式

2.4.4 关系运算符

表2-5 关系运算符

运算符	功能	目	结合性	用法
<、<=、>、>=	小于比较 小于等于比较 大于比较 大于等于比较	双目 双目 双目 双目	自左向右 自左向右 自左向右 自左向右	**expr1 < expr2、expr1 <= expr2、expr1 > expr2、expr1 >= expr2**
==、!=	相等比较 不等比较	双目 双目	自左向右 自左向右	**expr1 == expr2、expr1 != expr2**

```
int a=5,b=6,c=6,k;
3>4      //结果为假
a<b      //结果为真
k= b!=c      //k为0
k= b>=c      //k为1
```

```
若a=5 ， b=0 ， c=-5
a>b>c      //表达式为真
若a=5 ， b=0 ， c=2
a>b>c      //表达式为假
```

2.4.5 逻辑运算符

表2-6 逻辑运算符

运算符	功能	目	结合性	用法
!	逻辑非	单目	自右向左	!expr
&&	逻辑与	双目	自左向右	expr1 && expr2
||	逻辑或	双目	自左向右	expr1 || expr2

表2-7 真值表

expr1	expr2	expr1 && expr2	expr1 || expr2	!expr1	!expr2
假（0）	假（0）	假（0）	假（0）	真（1）	真（1）
假（0）	真（非0）	假（0）	真（1）	真（1）	假（0）
真（非0）	假（0）	假（0）	真（1）	假（0）	真（1）
真（非0）	真（非0）	真（1）	真（1）	假（0）	假（0）

注意：

① 在给出一个逻辑运算或关系运算结果时，以“0”代表“假”，以“1”代表“真”，在判断一个量为真假时，以“0”代表“假”，以“非0”代表“真”。

例：求解表达式 ! a && b || x+y > c

设 a=0,b=0,c=5,x=3,y=1,

运算符优先次序：

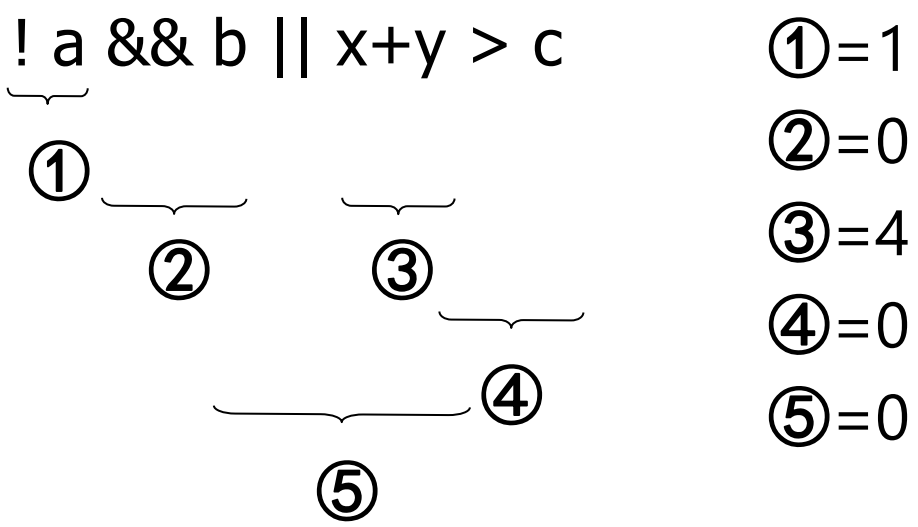
! （非）

算术运算符

关系运算符

&&和||

赋值运算符



② C++在逻辑表达式求解时，并不一定是所有的运算都被执行，当刚开始求解或求解的中途就可以确定整个逻辑表达式的值时，其余的运算将不再进行。

对&&运算，左边运算量为0，则不判断右边

对||运算，左边运算量为1，则不判断右边

例：a=1, b=2, c=3, d=4, m=n=1;

(m=a>b) && (n=c>d)

运算结果m的值为0，n的值为1。

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
=、+=、 -=、 *=、 /=、 %=、 &=、 ^=、 |=、<<=、 >>=	赋值复合赋值	双目 双目	自右向左 自右向左	**lvalue = exprl、value+=exprl 、value-=exprl 、value*=exprl、value/=exprl、value%=exprl、value&=exprl、value^=exprl 、value|=exprl、value<<=exprl、value>>=exprl**

2.4.8 赋值运算符

```
int a=6,c=10,m=21,n=32;
a=a-1; //正确，a减1后再赋值给a
```

```
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
```

```
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
```

```
int k=95,a=6,b=101;
const int n=6;
b-a=k; //错误
5=b-a; //错误
n=b-a*k; //错误
```

注意：“=”的作用

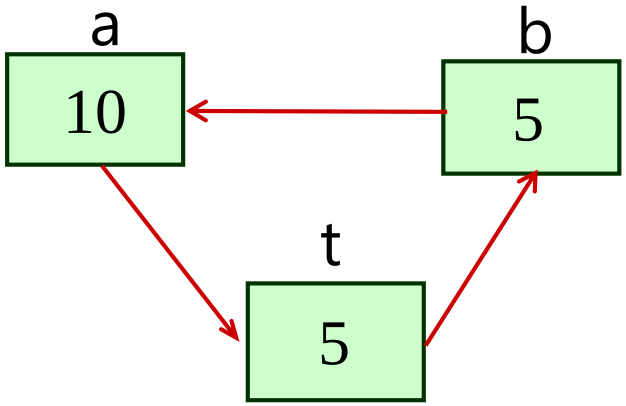
将“=”右端的值赋给左端的变量，不表示两端量相等。如：x=x+1结果是将x的值加1后赋给x，不是x与x+1相等。赋值后，“=”左端变量原值不存在。

又如： 要实现a、b交换，不能直接赋值。

应该使用中间变量实现：

```
int a=5,b=10,t;

t=a,a=b,b=t;
```



2.5 类型转换

- 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。

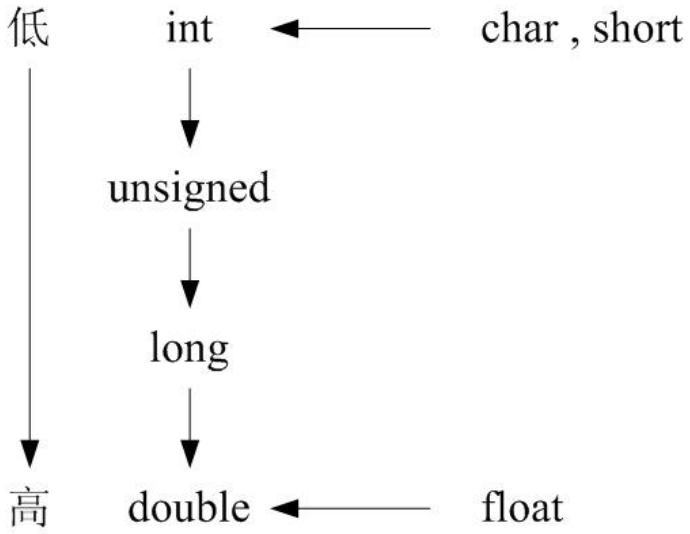
2.5.1 隐式类型转换

- 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。

- 1．何时进行隐式类型转换
  - （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。
  - （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。
  - （3）调用函数的实参被转换为函数形参的数据类型。

2. 混合运算中的隐式类型转换

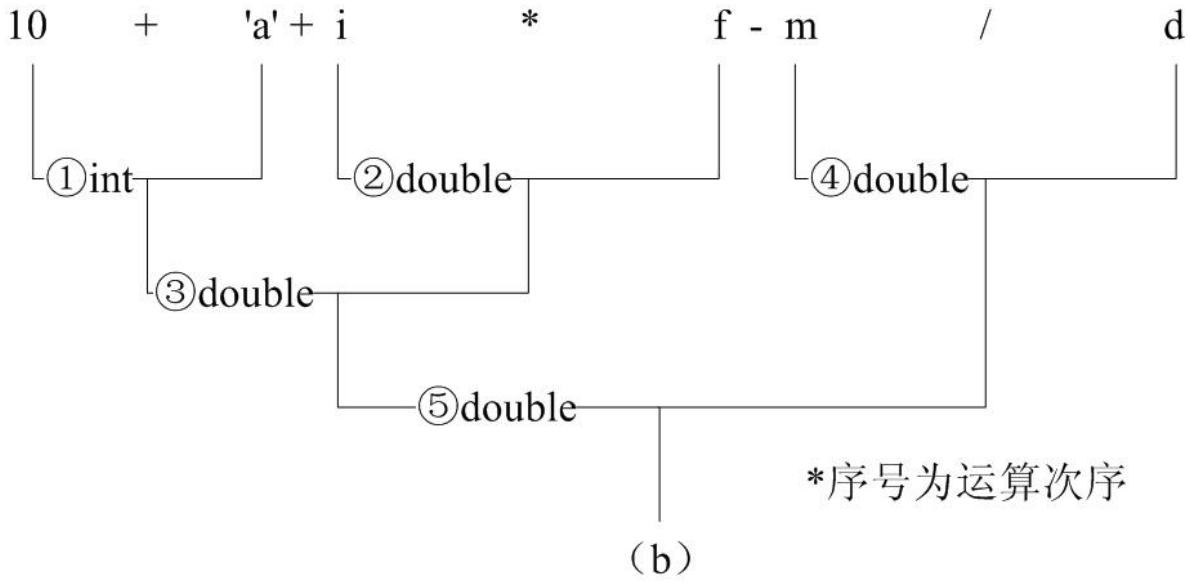
图2.4 混合运算类型转换



（a）

已知
int i; float f; double d; long m;

表达式10 + ‘a' + i * f - m / d的运算步骤如图所示。



- 3．赋值运算中的隐式类型转换
  - （1）将浮点型数据赋给整型变量时，舍弃浮点数的小数部分。如n是整型变量，n=6.18的结果是n的值为6。
  - （2）将整型数据赋给浮点型变量时，数值不变，但以浮点数形式存储到变量中。如78按78.0处理（根据浮点类型分别有7位或16位有效数字）。
  - （3）将double型数据赋给float变量时，截取前面7位有效数字存储到float变量。将float型数据赋给double变量时，数值不变，有效数字扩展到16位。

2.5.2 显式类型转换

表2-15 显式类型转换运算符

运算符	功能	目	结合性	用法
(type)	显式类型转换	单目	自右向左	**(type)expr**

```
(int)x+y //将x转换成整型
(int)(x+y) //将x+y转换成整型
(int)x%3 //x的类型和数据值不变，表达式引用转换成int后的x值
```

【例2.11】 将一个浮点型变量d保留两位小数（四舍五入）。

解

```
 (int)(d*100+0.5)/100.0
//d=1.2356, d*100=123.56+0.5=124.06, (int)124.06=124/100.0=1.24
```


程序设计