第3章 表达式和语句

目录

学习目标

- 理解表达式和语句的概念
- 掌握运算符的功能与特点
- 理解运算符的优先级和结合性
- 理解解决问题的逐步描述方式
- 理解自顶向下、逐步求精的方法
- 能够使用if和if/else语句来选择动作
- 能够使用表达式来描述语句和操作

3.1 表达式

- 表达式概述
- 左值和右值
- 优先级和结合性
- 语句和块

表达式概述

- 表达式：一个序列,含操作符、操作数,完成一个计算
- 可以嵌套

  2+3+(6*sizeof(int))/235
- 操作符和操作数必须匹配,但对编译器有额外约定

  例如：float a=3.2;

  int b = a+12; //float与int做+操作
- 表达式中操作数需约定求值次序ch3.9
- 操作符有优先级,级别高者先操作

  例如：2+3*6 中*优先级高于+
- 操作符有结合性,决定同优先级的操作次序

  例如：b=5;

  a=b=3; //先做b=3,再做a=b

左值和右值

- 针对赋值表达式而言
- 等号左边为左值表达式,等号右边为右值表达式
- 左值表达式需要具备可以改写的存储空间（注意与一般书中描述的差别）
  - 变量可以为左值
  - 常量、表达式、临时变量不能为左值
- 右值表达式只需要读取值,可为直接数或来自CPU的直接计算结果

运算符—优先级和结合性

- 表3-1所示
- 操作符即运算符
- 操作符有单目，双目，三目之分
- 第2级都是单目运算符（优先级同级）
- 第14级是三目运算符（只有一个）
- 第15级是赋值运算符
- 第2,14,15级为右结合，其余都是左结合

语句和块

- 表达式加上分号构成语句

  例如：a=a+2;

- If,switch,do…while,for,while是语句单位，因为其内部含有别的语句，称为复合语句
- 单独的花括号对{ }包起来的语句是语句块,语句块也是语句
- 复合语句含自身,称为嵌套

例如：
```
if(a>3)   //if语句
{
   if(b<a)  c=a+b;   //含if语句
}
else  c=d+6;
```

3.2 算术运算和赋值

- 操作符种类
- 赋值缩写
- 溢出

算术运算符:

C 的操作	算术运算符	代数表达式	C 表达式
加法	+	$f + 7$	f + 7
减法	-	$p - c$	p - c
乘法	*	bm	b * m
除法	/	x / y	x / y
求模	%	$r \bmod s$	r % s

运算优先关系:

运算符	操作	计算顺序（优先次序）
（）	小括号	先求值，若表达式里面还有小括号，则先求里面小括号的值。如果同级有几个小括号，则从左到右一一求之。
*，/，%	乘法，除法，求模	优先顺序没有括号高，如果同级有几个乘除操作，则从左到右一一求之。例如，8/2(2+2)=8/2*4=4*4=16，第一步先括号，所以得8/2*4，第二步乘除法优先级同，从左到右做，所以得4*4，第三步得16，不是8/2(2+2)=8/2*4=8/8=1
+,-	加法，减法	优先级没有乘除法高，如果同级有几个加减操作，则从左到右一一求之。

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72;` *(Last operation—place* ^{72} *in* ^{y}*)*

运算符

- 算术操作符
  - 单目： + - ++ --
  - 双目： + - * / %
  - 注意！
    - % 模(求余) ,必须是整型数据

      5%6＝5，9%4＝1，100%4＝0
    - / 整数除整数，得整数；如有一个是实数，得实数

      1/2＝0，9/4＝2
    - 双目运算符两侧操作数的类型要相同，如果不同也必须通过类型转换成为相同类型的操作数。

赋值运算符和赋值表达式

- 赋值符号“=”就是赋值运算符，作用是将一个数据赋给一个变量。
- 如“a=3”的作用是执行一次赋值操作（或称赋值运算），把字面值3赋给变量a。
- 也可以将一个表达式的值赋给一个变量。如“a=3+5-8%5”，相当于把结果值5赋给变量a（“a=5”）。

- 由赋值运算符将一个变量和一个表达式连接起来的式子称为“赋值表达式”。
- 一般形式为：<变量> <赋值运算符> <表达式>
- 赋值表达式求解过程：将赋值运算符右侧“表达式”的值赋给左侧的变量。表达式的值就是被赋值的变量的值。
- 上述一般形式中的“表达式”还可以是一个赋值表达式。如：a=(b=5)。所以，C/C++语言允许连续赋值。
- 赋值操作符的计算顺序是从右到左（与加减乘除操作不同），即遇到相同赋值操作符时，先做右边，再做左边，例如，a=b=5即a=(b=5),所以右边的括号去掉不影响计算。

变量赋初值

注意：int a=3;
相当于：int a; //指定a为整型变量
a=3; //赋值语句，将3赋给a

- 程序中常需要对一些变量预先设置初值。
- C/C++语言允许在定义变量的同时使变量初始化。
- int a=3; /*指定a为整型变量，初值为3*/
  float f=3.56; /*指定f为实型变量，初值为3.56*/
  char c='a'; /*指定c为字符变量，初值为 'a'*/
- 也可以使被定义的变量的一部分赋初值。
  int a,b,c=5; /*指定a,b,c为整型变量，只对c初始化*/
- 如果对几个变量赋予初值3，应写为:int a=3,b=3,c=3;
  不能写成：int a=b=c=3;

复合赋值运算符

- 在赋值符“=”之前加上其他运算符，可以构成复合赋值运算符。如：a+=3等价于a=a+3，x%=3等价于x=x%3。
- 注意：赋值符“=”右边看作一项。
  t=t*(s+u);//看作t*=(s+u);而非t*=s+u;
- 凡是二元运算符，都可以与赋值符一起组成复合赋值符。C/C++规定了10种复合赋值运算符：+=，-=，*=，/=，%=，<<=,>>=,&=,^=,|=

例如:

```
int main()
{
   int i=3;
   i+=2;
   cout<<i<<"\n";
}
```

思考:

```
i/=2;
i-=2;
i*=2
i%=2;
```

结果如何？

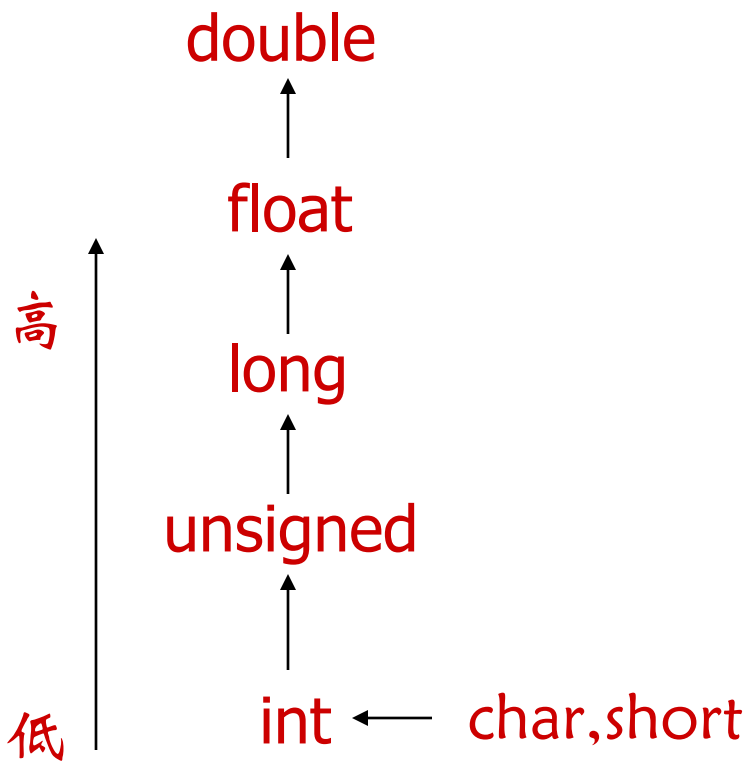
假设: `int c = 3, d = 5, e = 4, f = 6, g = 12;`

赋值运算符	表达式示例	解释	赋值
+=	`c += 7`	`c = c + 7`	10 to c
-=	`d -= 4`	`d = d - 4`	1 to d
*=	`e *= 5`	`e = e * 5`	20 to e
/=	`f /= 3`	`f = f / 3`	2 to f
%=	`g %= 9`	`g = g % 9`	3 to g

Fig. 算术赋值运算符

3.3 算术类型转换

- 整型（包括char,int,short,long）和实型（包括float,double）数据可以混合运算。
- 例如：10+‘a’+1.5-8765.1234*‘b’
- 在进行运算时，不同类型的数据要先转换成同一类型，然后进行运算。转换的规则如下图。



向左向上的箭头表示表达式计算时，不同类型转换的方向。例如，整型数加上float型数，则先将整型数转换成float型数。注意：float型在运算时先转换成双精度型，以提高运算精度（在32位C++编译器中，CPU中浮点数长度是64位，所以即使两个float型数据相加，也都先化成double型，结果再化成float型）。

3.4自增与自减

- 有前置和后置两种形式：

  前置形式： ＋＋变量名 ， －－变量名

  后置形式： 变量名++ , 变量名－－

- 功能：对变量的值进行自加1(自减1)的运算。
- 例：语句++a； 相当于执行a=a+1;
- 前置形式运算规则：先对表达式中含有前置自增(减)运算符的变量进行自加(减)1，然后用这些变量的新值参与表达式运算。
- 后置形式运算规则：先用变量原值参与表达式运算，然后对含有后置自加(减)运算符的变量进行自加(减)1运算。

运算符	示例表达式n	解释
++	++a	a 加 1 ， 然后， 在a所在的表达式中使用它的新值
++	a++	在 a 的表达式中使用它的当前值, 然后 a 加1.
--	--b	b 减1 ， 然后， 在b所在的表达式中使用它的新值.
--	b--	在 b 的表达式中使用它的当前值, 然后 b减1.
Fig. 3.12 增量运算符与减量运算符		

相当于 i=i+1;j=i

```
int main()
{
   int j,i=3;
   j=++i;
   cout<<i<<",”<<j;
}
```

运行结果：4,4

相当于 j=i; i=i+1

```
int  main()
{
   int j,i=3;
   j=i++;
   cout<<i<<",”<<j;
}
```

运行结果：4,3

(1) 自增运算符(++)和自减运算符(--)，只能用于变量，不能用于常量或表达式；例如：3++;错！

(2) 前++为左值，后++为右值。例如：a为5,则++a+=3;得a为9;a+++=3;错！

(3) 自增自减是一元运算符，比二元运算符优先级高。例如：a为5,则++a*=5;应理解为(++a)*=5;得30，而非++(a*=5);得26。

思考：若i的初值为3，那么cout<<-i++;的结果为多少呢？

3.5 关系与逻辑运算

- C/C++的执行语句要么会执行一些动作（计算或数据的输入或输出），要么会进行判断。
- 例如：我们可以在程序中作出判断，以判断一个人在考试中的评分是否大于等于60，如果大于等于60，那么就显示“Congratulation you passed”这样的信息。

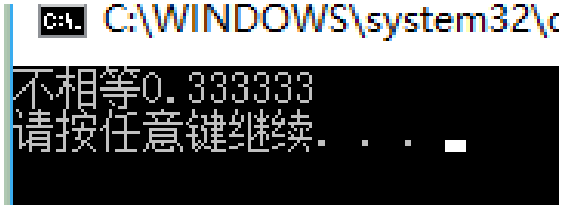
标准代数相等运算符或关系运算符	C 相等或关系运算符	C条件示例	C条件含义
相等运算符			
=	==	x == y	x等于y
≠	!=	x != y	x 不等于 y
关系运算符			
>	>	x > y	x 大于y
<	<	x < y	x 小于 y
>=	>=	x >= y	x 大于等于 y
<=	<=	x <= y	x 小于等于 y

逻辑运算符的说明

- 逻辑运算的短路求值：
  - (1)a&&b&&c：只有a为真时，才需要判断b的值，只有a和b都为真时，才需要判断c的值。
  - (2)a||b||c：只要a为真，就不必计算b和C。
  - 例如：
    - int a = 1, b = 2, c = 3, d = 4;
    - int m = 1, n = 1;
    - (m = a>b) && (n = c>d)： m和n的值？

浮点数的比较

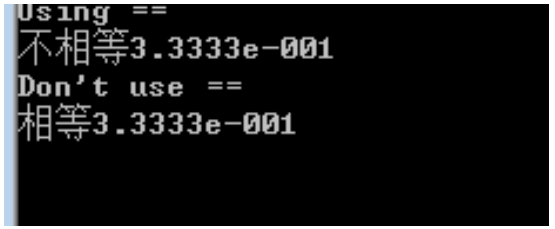
```
float x = 1.0 / 3;
if (x == 0.333333f)
    std::cout << "相等" << x<< std::endl;
else
    std::cout << "不相等" << x<<std::endl;
```



注意：

浮点数进行相等(==)和不相等(!=)比较操作一般都有问题，所以，浮点数的比较一般总是使用两个浮点数相减的结果是否落在0的某个领域内来判断！

```
float x = 1.0 / 3;
if ((x-0.33333f)<10e-6)
   std::cout<< “相等<<x<<std::endl;
else
   std::cout<<"不相等<<x<<std::endl;
```



运算符的几点说明

① =和==的区别：
- 前者是赋值操作符，后者是关系操作符，用于比较两个数是否相等。
- 例如：
  - x=0　和 x == 0：前者是赋值，后者是比较

② 数学表达式和C++表达式的区别：
- $s(s-a)(s-b)(s-c)$ ⟶ `s*(s-a)*(s-b)*(s-c)`
- $(x+2)e^{2x}$ ⟶ `(x+2)*exp(2*x)`

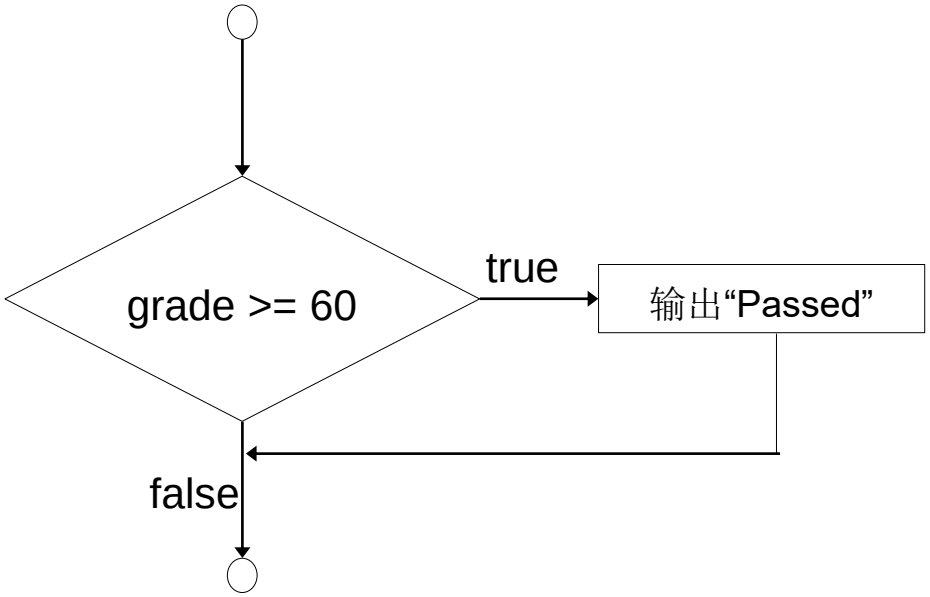
③ 不等式连写的错误：
- int a = -1, b=8,c=6;
- a<b<c：结果为true
- (a<b) && (b<c): 结果为false

3.6 if 语句

- 选择结构用于在可选择的几个动作之间进行选择。
- 例如：假定一次考试中的及格分数是60分。
- 伪代码为：
- **if 学生分数大于或等于60**
  输出“passed”
- **如果条件为真，那么就输出“passed”，然后按照顺序，下一个伪代码语句就会被“执行”，如果条件为假，那么就会忽略操作，按照顺序执行下一条语句**

前面的语义可以写成C/C++语句的形式：

```
if(grade >= 60)
   cout<<"Passed\n";
```



菱形表示对任何表达式的判定.

0 代表 false

非0 代表 true

例如:

3 和 -4 为 true

if语句的嵌套和匹配

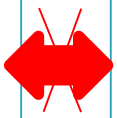
- if 语句：

形式1：
```
if (x > y)
    y = x;
```

形式2：
```
if (x > y)
    max = x;
else
    max = y;
```

形式3：
```
if (x > 100)
    x *= 0.7;
else if (x > 90)
    x *= 0.8;
else if (x > 80)
    x *= 0.9;
```

嵌套形式：
```
if (x > y)
{
    max = x;
    if (max < z)
        max = z
}
else
    max = y;
```



嵌套形式：
```
if (x > y)
    max = x;
    if (max < z)
        max = z
else
    max = y;
```

注意：

if语句的嵌套中，else与其前面最近的未配对的if配对

3.7 条件运算符

- 条件运算符?:是C/C++中唯一一个三元运算符，其?:这两个字符隔开三个操作数，构成条件运算表达式。
- 形式：x ? y : z //x是条件表达式，y是条件表达式为true时的表达式(值)，z是条件表达式为false时的值。
- 例如：
- **int** a=3,b=5;
- **int** x=a<b?a:b; //因a<b，故x得表达式a的值3

3.8 逗号表达式

- 用逗号隔开若干表达式，构成大表达式。从左到右挨个计算表达式值，以最后的子表达式作为大表达式的值。例如：

```
int a,b,c;
c=(a=1,b=a+2,b+3); //a为1,b为3,c为6
```

3.9 求值次序与副作用

- 不同的编译器求值顺序不同
- 求值顺序使交换律失去作用
- 求值顺序使括号失去作用
- 消除副作用

3.9 求值次序与副作用

取数操作若涉及内存中的变量访问和修改，例如：

```
int a=3,b=5,c;
c=a*b + ++b;   //取++b时顺便会修改b
```

- a*b与++b的结果分别取来用以做+计算，但是先取前者还是后者，C++标准并不规定（不同的编译器求值顺序不同）
  - vs编译器： c=24　　GNU编译器：c=21
- 先取前操作数或后操作数，都将使c=a*b+++b;与c=++b+a*b;中的c结果不同（求值顺序使交换律失去作用）
  - c=++b+a*b // vs编译器： c=24　　GNU编译器：c=24
- 即使括号也无法幸免（求值顺序使括号失去作用）,例如c=++b*(a+b);与c=(a+b)*++b;括号操作同样有先后操作的差别
- 消除副作用的方法：直接拆开

```
int a=3,b=5,c;
++b; c=a*b+b;    //或   c=a*b+b; ++b;
```