東南大學 SCHOOL OF INTEGRATED CIRCUITS, SEU

集成电路学院

计算机科学基础I

——常量和表达式

东南大学 集成电路学院 朱彬武

E-mail: bwzhu@seu.edu.cn

字面量和常量

字面量


```
(pound + ounce / 16) * 0.453592
```

- **在程序设计中，固定不变的数，是常数。如果直接将其写在程序里，我们称其为字面量（literal）**

字面量

- 布尔型常量（默认bool类型）
  - true， false
- 整型常量（默认int类型（32位））
  - 123，-123
- 实型常量（默认double类型）
  - 1.2，12e-3，.23，-.1e-3
- 进制前缀
  - 十六进制：0X或0x开头，0xBAD
  - 八进制：0开头，0123
  - 二进制（C++14）：0B或0b开头，0b101
- 范围、精度后缀
  - U、L（64位）、UL、F（单精度） （不区分大小写）

```
// 布尔型常量（默认类型：bool）
bool b1 = true;   // 表示真

// 实型常量（默认类型：double）
double d1 = 12e-3;     // = 12 * 10^-3 = 0.012
double d2 = .23;       // 省略整数部分，相当于 0.23

// 进制前缀
int hexVal = 0xBAD;    // 十六进制（0x 或 0X 开头）

// 范围、精度后缀
unsigned int ui = 123U;    // U 或 u：无符号
long l = 123L;             // L 或 l：long
```

让人困惑的程序


```
double pound, ounce;
cout << "请分别输入包裹的英镑和盎司，如输入\"8 5\"表示8英镑5盎司:" << endl;
cin >> pound >> ounce;

cout << "质量是" << (pound + ounce / 16) * 0.453592 << "千克"  << endl;
```

但是，别人看这段程序的时候第一个反应：16和0.453592是什么？人们管这种莫名其妙的数字叫魔法数（Magic Number）

常量


```
(pound + ounce / 16) * 0.453592
```

- **好的编程规范要求程序中避免出现这种魔法数，魔法数会提高代码的理解难度**
- **更好的方式是定义一个常量**

```
const int OUNCE_PER_POUND = 16;
const double KG_PER_POUND = 0.453592;
```

const修饰符

- const是一个修饰符，加在变量类型（int，double）前面，用来给这个变量加上一个const（不变的）的属性。这个const的属性表示这个变量的值一旦初始化，就不能再修改了。

const <类型名称> <变量名称> = <初始值>

- 如果试图对常量做修改，把它放在赋值运算符的左边，就会被编译器发现，指出为一个错误。

```
const int AREA = 100;    //定义了一个常量AREA，初始值为100
AREA = 20;  //尝试给常量赋值，编译器会报error
```

加入常量后的代码


```
const int OUNCE_PER_POUND = 16;
const double KG_PER_POUND = 0.453592;
double pound, ounce;
cout << "请分别输入包裹的英镑和盎司，如输入\"8 5\"表示8英镑5盎司:" << endl;
cin >> pound >> ounce;
cout << "质量是" << (pound + ounce / OUNCE_PER_POUND) * KG_PER_POUND << "千克"  << endl;
```

编程规范

- **良好的格式与排版：缩进统一、花括号风格一致**

```
int main(){
    cout << "Hello World" << endl;return 0;
}
```

```
int main(){

}
```

```
int main()
{

}
```

两种风格都很常见，但要在代码里保持统一

- **命名规范：驼峰命名，下划线命名（参考上周PPT）**
- **避免魔法数字**
- **注释**
- **……**

注释

注释

注释（comment）是插入在程序代码中用来向读者（自己）提供解释信息。它们对于程序的功能没有任务影响，但是往往能使得程序更容易被人类读者理解。

注释的两种写法

- 单行注释——两个斜杠开头
- 多行连续注释——由“/*”开始，“*/”结束

注释写法


```
// 这是一个单行注释
int a = 10; // 变量a等于10
```

```
/* 这是多行注释
可以写很多内容 */
int a = 10;
```

- 单行注释也可以用/* */

```
/*这是一个单行注释*/
int a = 10; /*变量a等于10*/
```

- 多行注释也可以每行加一个//

```
// 这是多行注释
//可以写很多内容
int a = 10;
```

Visual C++快捷键：默认ctrl+k ctrl+c（注释），ctrl+k ctrl+u（撤销注释），可以修改

VS Code快捷键：ctrl+/ （注释，撤销注释）

注释的更多功能

- 解释变量/函数用途
- 描述复杂逻辑
- 标记TODO/BUG
- 暂时屏蔽代码

如何写好的注释？供参考：
https://zh-google-styleguide.readthedocs.io/en/latest/google-cpp-styleguide/comments.html

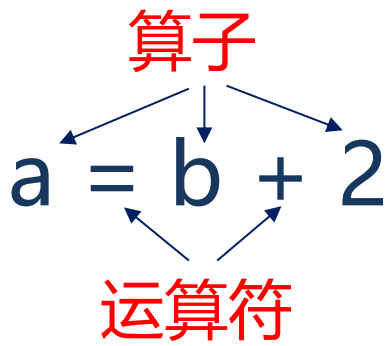
表达式

表达式

- 一个表达式是一系列运算符和操作数的集合，用来计算一个值
  - 算术表达式 `a + b`
  - 自增/自减表达式: `i++`
  - 关系表达式 `a > b`
  - 逻辑表达式 `a > 0 && b > 0`
  - 赋值表达式 `x = 10`
  - 组合表达式 `(a + b) * c - 5`
  - 条件运算符表达式 `max = (a > b) ? a : b`
  - 函数调用表达式 `sqrt(16)`

表达式基本组成

- 运算符（operator）是指进行运算的动作，比如加法运算符“+”，减法运算符“-”
- 操作数（operand）是指参与运算的值，这个值可能是常数，可能是变量，还可能是一个函数（方法）的返回值




```
c = sqrt(pow(a, 2) + pow(b, 2))
```

运算符优先级

优先级	名称	运算符	结合关系	例子
1	括号运算符	()	自左向右	

2	自增、自减	++，--	自右向左	int i = 0, j = 0;
				j = ++i;
				j = i++;

2	逻辑非	!		int i = !(1==0)
2	正号，负号	+，-		int i = 2, j =3; int k = i *(-j)
3	乘，除，取余	*，/，%	自左向右	
4	加、减	+，-	自左向右	
5	左移，右移	<<，>>	自左向右	
6	小于，小于等于，大于，大于等于	<，<=，>，>=	自左向右	
7	等于，不等于	==，!=	自左向右	int i = 1==1 int j = 0!=1

结合关系指的是有相同优先级的运算符放在一起，按照指定顺序运算

运算符优先级

优先级	名称	运算符	结合关系	例子
8	按位与	&	自左向右	101 & 110
9	按位异或	^	自左向右	101 ^ 110
10	按位或	|	自左向右	101 | 110
11	逻辑且	&&	自左向右	(1!=1)&&(1==1)
12	逻辑或	||	自左向右	(1==1)||(0!=1)
13	条件运算符	?:	自右向左	int a=1>0?2:3;
14	赋值运算符	=	自右向左	int a, b, c = 0 b=a=2 c=b+=2
14	复合赋值运算符	+=，-=，*=，/=	自右向左	
15	逗号表达式	,	自左向右	int i=0 , j=0 j=2, 3

要掌握运算符之间的优先级关系，分类记忆，大体上看是单目>算术>关系>逻辑>赋值

自增（减）运算符

- 自增运算符：++，操作数必须是变量，不能是++2
- i++/++i
  - i = i+1
- 两种形式
  - 前置自增：++i，表达式的值是加1以后的值
  - 后置自增：i++，表达式的值是i的值
- 使用场景：（1）常用于循环（2）简化变量加1操作
- 例：int i= 0；i++；++i；表达式i++的值为（0），表达式++i的值为（2）

关系运算符

- 关系运算符：计算两个值之间的关系

运算符	含义
==	相等
!=	不相等
>	大于
>=	大于等于
<	小于
<=	小于等于

- 关系运算的结果：当两个值的关系符合关系运算符的预期时，关系运算的结果为true，否则为false

- 使用场景：判断、循环

```
cout << (5==3) << endl;
cout << (5>3) << endl;
cout << (5<=3) << endl;
```

赋值运算符

- 赋值也是运算，也有结果
- a=6的结果是a被赋予的值，也就是6
- a=b=6等价于a=(b=6)，因为赋值运算符的结合关系是自右向左
- 例：int a = 6; int b; int c = 1 + (b=a)；c的值是（7）。

复合赋值运算符

- 5个算术运算符，+ - * / %都可以和赋值运算符=结合起来，形成复合赋值运算符：+=，-=，*=，/=，%=

- sum += 5
  - sum = sum + 5

- total += (sum+100)/2
  - total = total + (sum+100)/2

- total *= sum+12
  - total = total * (sum+12)

逗号运算符

- 逗号运算符：，
- 作用：依次执行多个表达式，结果为最后一个表达式的值
- 语法形式：表达式1，表达式2，…，表达式n
  - 从左到右依次执行
  - 整个逗号表达式的值等于最后一个表达式
- 使用场景：for循环
- 例题：int i= 0, j = 0；j = ++i, i++，表达式的值为（1），i的值为（2），j的值为（1）

考试爱考

1. int x=0,y=2,z=3; 则表达式 x!=x ||(y=y^y)&&(z=z+1)执行后，说法正确的是 ( )。

A. x 的值为 1 B. y 的值为 2 C. z 的值为 4 D. 表达式值为 0

2.若定义float x=1;int y=2;则表达式!y && y^3 ? y-x : y++ 的值为( )。

A. 2 B. 0 C. 1 D. 3

实际编程和考试是两回事

可以说，考试喜欢出的复杂表达式都属于不规范的编程，不利于阅读和理解，容易造成读程序时候的误解。所以要避免写出这种复杂表达式。将复杂表达式拆开来，按照你的预期顺序写出来。