C语言

第2讲：运算符和表达式

赵岩

哈尔滨工业大学软件学院

August 20, 2011

目录

目录

目录

目录

变量定义

- 变量定义语句：
  - <数据类型名> <变量名列表>;
  - 例1：`int` i, j, k;
  - 例2：`long` l1, l2;
  - 例3：`int` i=3, j=5, k=i+j;
- 变量名：其命名规则同标识符。

变量定义

- 变量定义语句：
  - <数据类型名> <变量名列表>;
  - 例1：`int` i, j, k;
  - 例2：`long` l1, l2;
  - 例3：`int` i=3, j=5, k=i+j;
- 变量名：其命名规则同标识符。

变量定义说明

- 变量必须“先定义，后使用”
  - 所有变量必须在第一条可执行语句前定义（C语言）
  - 定义的顺序无关紧要
  - 一条定义语句可定义若干个**同类型**的变量，变量名之间用逗号分隔
  - 变量定义后，即占用内存，可随时向其存入数据，并可通过变量名使用数据
- 定义变量时，是初始化变量的最好时机
  - 不被初始化的变量，其值为危险的垃圾(随机)数

```
1  int i,j;
2  printf("%d",i);  /*i的值到底为多少*/
```

变量定义说明

- 变量必须“先定义，后使用”
  - 所有变量必须在第一条可执行语句前定义（C语言）
  - 定义的顺序无关紧要
  - 一条定义语句可定义若干个同类型的变量，变量名之间用逗号分隔
  - 变量定义后，即占用内存，可随时向其存入数据，并可通过变量名使用数据
- 定义变量时，是初始化变量的最好时机
  - 不被初始化的变量，其值为危险的垃圾(随机)数

```
1  int i,j;
2  printf("%d",i);  /*i的值到底为多少*/
```

常见常量

- 整型常量
  - 18、-31
  - `long int`型常量123l、123L、123456l、123456L
  - `unsigned int`型常量123u、123U
- 浮点常量
  - 十进制小数形式123.45、456.78
  - 指数形式1e-2、4.5e3
  - `float`型常量123.45f、456.78F、1e-2f、4.5e3F
  - `long double`型常量123.45l、456.78L、4.5e3L
  - 缺省为`double`
  - 因为字母l和数字1容易混淆，所以当用l做后缀时，常使用大写形式

常见常量

- 整型常量
  - 18、-31
  - `long int`型常量123l、123L、123456l、123456L
  - `unsigned int`型常量123u、123U
- 浮点常量
  - 十进制小数形式123.45、456.78
  - 指数形式1e-2、4.5e3
  - `float`型常量123.45f、456.78F、1e-2f、4.5e3F
  - `long double`型常量123.45l、456.78L、4.5e3L
  - 缺省为`double`
  - 因为字母l和数字1容易混淆，所以当用l做后缀时，常使用大写形式

整型常量的进制和后缀

- 以数字“0”开始(不是字母“O”)的整型常数是八进制数
  - 022、-037
  - 010和10大小不一样
  - 因为八进制并不常用，所以此种表示法比较少见
- 以“0x”或者“0X”开始的整型常数是十六进制
  - A-F和a-f用来表示十进制的10-15
  - 十六进制的形式比较常用
  - 0x12、-0x1F, -0x1f
  - 0XFUL 这是一个什么数？

整型常量的进制和后缀

- 以数字“0”开始(不是字母“O”)的整型常数是八进制数
  - 022、-037
  - 010和10大小不一样
  - 因为八进制并不常用，所以此种表示法比较少见
- 以“0x”或者“0X”开始的整型常数是十六进制
  - A-F和a-f用来表示十进制的10-15
  - 十六进制的形式比较常用
  - 0x12、-0x1F, -0x1f
  - 0XFUL 这是一个什么数？

常量后缀的应用

- 整型后缀
- 浮点型后缀

```
1  main()
2  {
3      float r;
4      r = 1.0f+0.2f;
5      r = r-1.2f;
6      printf("%22.16f\n",r);
7  }
```

- 去掉f后缀后，发生什么？

常量后缀的应用

- 整型后缀
- 浮点型后缀

```
1  main()
2  {
3      float r;
4      r = 1.0f+0.2f;
5      r = r-1.2f;
6      printf("%22.16f\n",r);
7  }
```

- 去掉f后缀后，发生什么？

常量后缀的应用

- 整型后缀
- 浮点型后缀

```
1  main()
2  {
3      float r;
4      r = 1.0f+0.2f;
5      r = r-1.2f;
6      printf("%22.16f\n",r);
7  }
```

- 去掉f后缀后，发生什么？

字符型常量

- 字符常数的表示方法
  - 'a'，'A'，'5'，' '
  - 单引号内只能有一个字符，除非用“\”开头
- 字符常量就是一个普通整数，也可以参与各种数学运算
  - 每个字符有一个0-255之间的值，可从ASCII表查出
  - 注意：'5'和整数5的区别
- 用“\”开头的字符为转义字符
  - 例如，"\n"，代表1个回车字符

字符型常量

- 字符常数的表示方法
  - 'a'，'A'，'5'，' '
  - 单引号内只能有一个字符，除非用“\”开头
- 字符常量就是一个普通整数，也可以参与各种数学运算
  - 每个字符有一个0-255之间的值，可从ASCII表查出
  - 注意：'5'和整数5的区别
- 用“\”开头的字符为转义字符
  - 例如，"\n"，代表1个回车字符

字符型常量

- 字符常数的表示方法
  - 'a'，'A'，'5'，' '
  - 单引号内只能有一个字符，除非用“\”开头
- 字符常量就是一个普通整数，也可以参与各种数学运算
  - 每个字符有一个0-255之间的值，可从ASCII表查出
  - 注意：'5'和整数5的区别
- 用“\”开头的字符为转义字符
  - 例如，"\n"，代表1个回车字符

转义字符

\a	响铃符	\\	反斜杠
\b	回退符	\?	问号
\f	换页符	\'	单引号
\n	换行符	\"	双引号
\r	回车符	\ooo	八进制数
\t	横向制表符	\xhh	十六进制数
\v	纵向制表符		

字符串常量

- 用双引号括住的由0个或多个字符组成的字符序列
  - “I am a string”
  - “”表示空字符串
  - 转义字符也可以在字符串中使用
  - 引号只作为字符串开始和结束的标志
  - 除注释外，是唯一可以出现中文的地方
- “x”和’x’是不同的
  - C语言内部用’\0’表示字符串的结束
- <string.h>里定义了一系列专门的字符串处理函数
  - strlen, strcpy

字符串常量

- 用双引号括住的由0个或多个字符组成的字符序列
  - “I am a string”
  - “”表示空字符串
  - 转义字符也可以在字符串中使用
  - 引号只作为字符串开始和结束的标志
  - 除注释外，是唯一可以出现中文的地方
- “x”和’x’是不同的
  - C语言内部用’\0’表示字符串的结束
- <string.h>里定义了一系列专门的字符串处理函数
  - strlen, strcpy

字符串常量

- 用双引号括住的由0个或多个字符组成的字符序列
  - “I am a string”
  - “”表示空字符串
  - 转义字符也可以在字符串中使用
  - 引号只作为字符串开始和结束的标志
  - 除注释外，是唯一可以出现中文的地方
- “x”和’x’是不同的
  - C语言内部用’\0’表示字符串的结束
- <string.h>里定义了一系列专门的字符串处理函数
  - strlen, strcpy

宏常量

- 宏常量
- #define　标识符　字符串

Demo 1: A simple example about macro

```
1 #include <stdio.h>
2 #define PI 3.1415926
3 main(){
4       float r = 2.0f;
5       printf("area = %f\n",PI*r);
6 }
```

const 常量

- const float PI = 3.1415926f;
- 一旦被const修饰，在程序中就无法修改了。

Demo 2: A simple example about const

```
1 #include <stdio.h>
2 const float PI = 3.1415926f;
3 main()
4 {
5       float r = 2.0f;
6       PI = 3.14f; /*- compile error*/
7       printf("area = %f\n",PI*r);
8 }
```

枚举常量

Demo 3: A simple example about enumation

```
1  enum weeks {
2      MON, TUE, WED, THU, FRI, SAT, SUN
3  };
4  enum weeks today, tomorrow;
5
6  today = MON;
7  tomorrow = today + 1;
8  if (tomorrow == TUE)
9      printf("Tomorrow is Tuesday.\n");
10 else
11     printf("Tomorrow is NOT Tuesday.\n");
```

使用常量注意事项

- 避免使用“幻数”
  - 程序的可读性变差
  - 容易发生书写错误
  - 修改麻烦
- 用以下三种方式来代替
  - 宏常量
  - const常量
  - 枚举常量
- 建议使用后两者。

使用常量注意事项

- 避免使用“幻数”
  - 程序的可读性变差
  - 容易发生书写错误
  - 修改麻烦
- 用以下三种方式来代替
  - 宏常量
  - const常量
  - 枚举常量
- 建议使用后两者。

使用常量注意事项

- 避免使用“幻数”
  - 程序的可读性变差
  - 容易发生书写错误
  - 修改麻烦
- 用以下三种方式来代替
  - 宏常量
  - const常量
  - 枚举常量
- 建议使用后两者。

算术运算符

- +, -, *, /
  - 加、减、乘、除运算，先乘除，后加减
  - 从左向右计算（左结合性）
- %
  - 求余(取模)，**符号与被除数相同**
  - 求余只能用于整型数据类型
- 数学库函数

常用的标准数学函数

函数名	功 能	函数名	功 能
sqrt(x)	计算x的平方根，x应大于等于0	exp(x)	计算e^x的值
fabs(x)	计算x的绝对值	pow(x,y)	计算x^y的值
log(x)	计算lnx的值	sin(x)	计算sinx的值，x为弧度值
log10(x)	计算lgx的值	cos(x)	计算cosx的值，x为弧度值

关系运算符

- >，>=，<，<=，==，!=
  - 大于，大于等于，小于，小于等于，等于，不等于
  - 关系运算符运算出的结果为0和1
    - 0，表示假，即该关系不成立
    - 1，表示真，即该关系成立
- 在所有涉及到真假判断的地方，0表示假，非0表示真

```
1  int a=0;
2  if (a=0) /* if(0=a) */
3       printf("a equal 0");
4  else
5       printf("a not equal 0");
```

逻辑运算符

- 逻辑运算也被称为布尔（Boolean）运算，运算结果也是1和0
- &&
  - 与运算
  - (a > b && b > c)；a大于b，并且b大于c
- ||
  - 或运算
  - (a > b || b > c)；a大于b，或者b大于c
- !
  - 求反
  - (!a)；如果a是0，结果是1；如果a是非0，结果是0
  - 并不改变a的值

逻辑运算规则

&&	1	0
1	1	0
0	0	0

||	1	0
1	1	1
0	1	0

!	1	0
	0	1

&&和||运算的一个特点

- 只算一半
- &&
  - 如左边的表达式值为0，则运算结果必为0
  - 所以，如左边为0，右边的表达式就不再求值
- ||
  - 如左边的表达式值为非0，则运算结果必为1
  - 所以，如左边为非0，右边的表达式就不再求值
- 此特点有时可以用来提高程序效率，有的时候也是错误的根源

```
1  if(a==0 && b=b+1) /*- error 1 */
2
3  if( (a!=2) || (a!=3) ) /*- error 2 */
```

逻辑运算符实例

- 判断ch是否为英文大写字母
- 判断某一年year是否是闰年的条件是满足下列两个条件之一
  - 能被4整除，但不能被100整除；
  - 或者能被400整除；

```
1       if((ch>='A') && (ch<='Z'))
2
3       if ( ( (year%4==0)&&(year%100)!=0)
4            || (year%400 == 0) )
```

增1和减1操作符使用事项

- 运算符为后缀，运算结果是加/减1之前的值
- 运算符为前缀，运算结果是加/减1之后的值
- 在一行语句中，使用++或--的变量最好只出现一次
- 否则，不仅可读性差，而且因为编译器实现的方法不同，容易导致不同编译器运行效果不一样，贻害无穷

```
1  int i=3; int j=3;
2  printf("%d,%d",j+(++i),i); /*output 7,4*/
3  printf("%d,%d",j+(i++),i);   /*output 6,4*/
4  printf("%d,%d",i+(++i),i); /*output ?*/
5  k=(i++)+(++i)+(i++); /*-bad style, not kiss*/
6  (i+j)++; /*-error, only variable*/
```

位运算符

- <<
  - 按位左移运算
  - 等价于乘2操作
- >>
  - 按位右移运算
  - 分为算术移位和逻辑移位
- ~
  - 按位求反
  - ~(~0<<3)
- &
  - 按位与运算
  - 将特定位置0
- |
  - 按位或运算
  - 将特定位置1
- ∧
  - 按位异或运算

右移中的算数移位和逻辑移位

- 算术移位和逻辑移位
- 最右面的如果是零，一定是逻辑移位。最右面的如果是1，就要考虑数据的符号型了。

```
1 int i; /* unsigned int i; what happen? */
2 i = (~0)<<4;
3 printf("%d\n",i);
4 i = i>>2;
5 printf("%d\n",i);
```

位运算符实例

- 快速乘除

```
1  int i=100;
2  i<<2; /* multiply 4 */
3  i>>3; /* divided by 8 */
```

- 取一个数的从右端开始的4-7位

```
1  i>>4;
2  i&~((~0)<<4);
```

赋值运算和复合赋值运算符

- 赋值运算的结果是被赋值变量赋值后的值
  - $a = b = c = 0;$
  - $a = (b = (c = 0));$
- 复合赋值运算符
  - $i = i + 2;$
  - $i \mathrel{+}= 2;$
  - +、-、*、/、%、<<、>>、&、|运算符都可以按此种方式处理
- 这种形式看起来更直观，而且执行效率一般也能更高一些

逗号运算符

- 表达式1, 表达式2, ..., 表达式n
- 从第一个开始，依次计算每个表达式的值，最后一个表达式的值即为逗号表达式的值
- 主要用在循环语句中，同时对多个变量赋初值等

```
1 for (i = 0 , j = 0; i < j; i++, j++)
2
3 x = (a=2,b=5,b++,a+b) /*x equals ? */
```

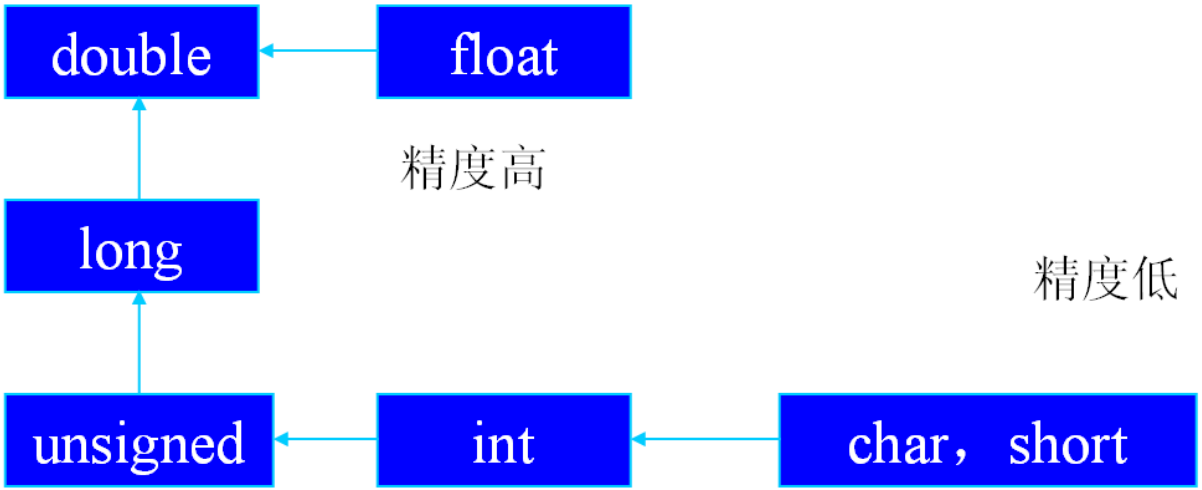
赋值中类型转换

- 数据赋值时
  - 从小类型到大类型，顺利转换
  - 从大到小，发出警告（好的编译器会给出），结果可能错误
  - 要知道自己在做什么

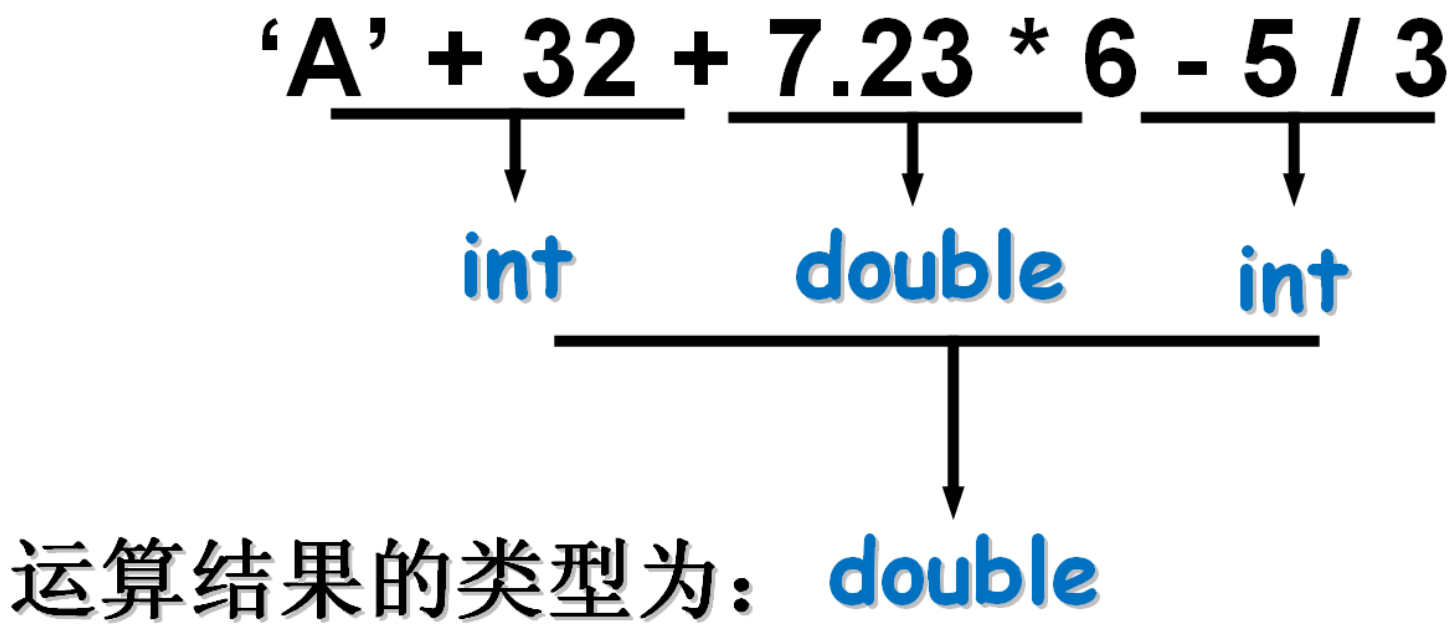
```
1 int i;
2 float f = 2.8f;
3 i = f; /*隐式转换*/
4 i = f+0.5f; /*四舍五入*/
```

表达式中自动类型转换

- 混合运算
  - 当操作数1和操作数2的类型不同时，即为混合运算
  - 在运算时要求类型应相同。即先转换，再运算，运算结果的类型同操作数的类型。



表达式类型转换实例



字符串与数值类型之间的转换

- int i = “123”
  - 这样用是不行滴
- atof()，atoi()，atol()
  - 把字符串转为double，int和long
  - 定义在stdlib.h中
- sprintf()
  - 可以用来把各种类型的数值转为字符串
  - 定义在stdio.h中

字符串与数值类型之间的转换实例

Demo 4: conversion of number and string

```
1   float f = 1234.5678f; double d;
2   char numstr[10] = "765.4321";
3   char str[10];
4
5   /* change number to string*/
6   sprintf(str, "%9.4f",f);
7   printf("%s\n",str);
8   /* change string to number*/
9   d = atof(numstr);
10  printf("%f\n",d);
```

类型强制转换

- 可以通过“(类型名)操作数”的方式把表达式的值转为任意类型
  - 类型名必须用括号括起
  - 操作数：可以是常量、变量、函数、表达式
  - 例如：(float)(5/3)，(float)5/3
- 强转时，你必须知道你在做什么！
  - 强转与指针，并称C语言两大神器，用好了可以呼风唤雨，用坏了就损兵折将

条件表达式

- 把a和b中的最大值放入z中
- 此种表达式切忌用得过于繁杂

```
1  if (a > b)
2      z = a;
3  else
4      z = b;
5
6  z = (a > b) ? a : b;
```

运算符优先级

- 单目→算术→关系→逻辑→赋值→逗号
- 能背下完整优先级表的人凤毛麟角
  - 脑细胞太宝贵了，不能用来死记硬背
- 用**括号**或者**分步书写**来控制运算顺序更直观、方便，并减少出错的概率
  - 先算乘除，后算加减，有括号就先算括号里的
  - 注意用空格做好分隔
  - 实在不行就拆分表达式

运算符优先级实例

```
1  x *= y+1; /* How to understand?*/
2
3  char c;
4  while(c=fgetc(STDIN)!=EOF)
5  {
6      ...
7  }
8  while((c=fgetc(STDIN))!=EOF)
9  {
10     ...
11 }
```

运算符的结合性

- 当运算符在一个级别的时候，要考虑结合性
- 双目运算符基本都是左结合性
- 大部分单目运算符是右结合性（附录D）
  - 负号运算符-
  - 自增减运算符++
  - 按位取反

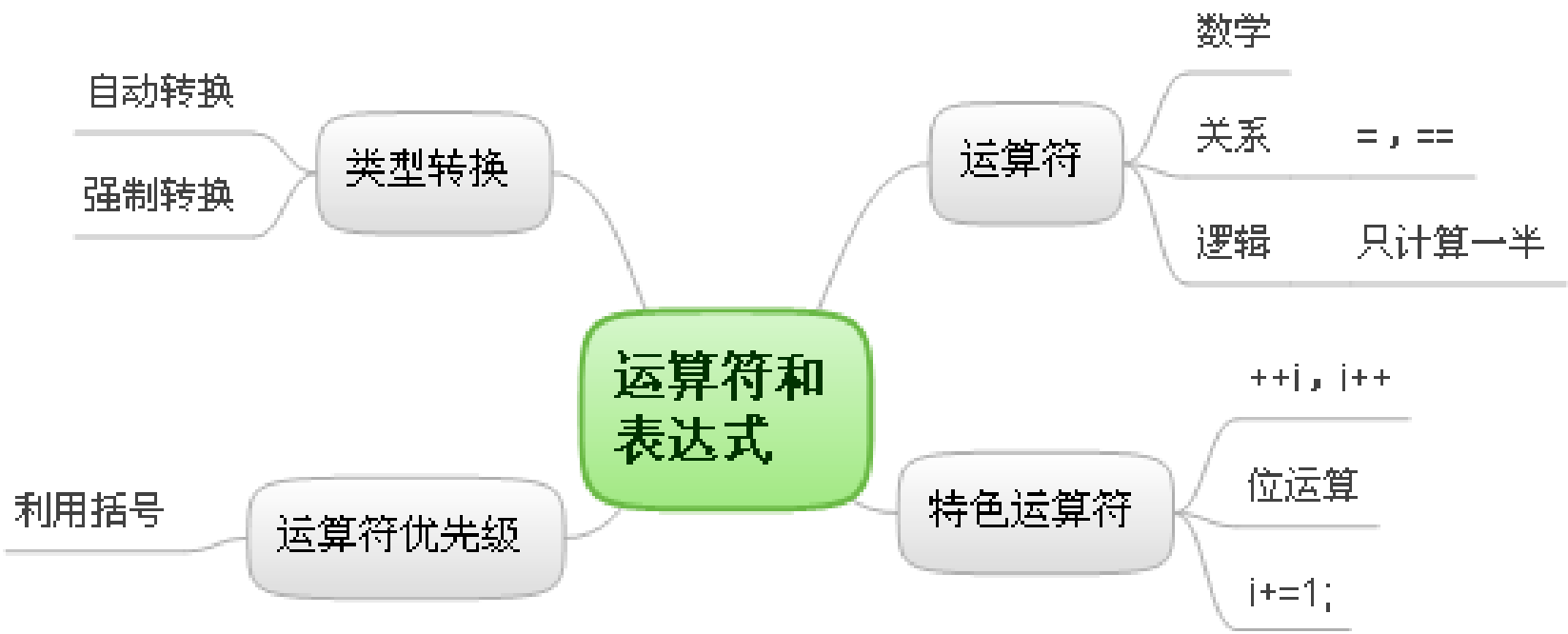
```
1  4/2*4     /* 8 or 0.5? 左结合*/
2  x=y=z+1;  /* y=z+1; x=y; 右结合*/
3  m=-n++;   /*  m=-n ;  n++;  */
```

运算的顺序

- 运算顺序并不固定
- **赋值运算符**和**自增减运算符**最好单独一个语句

```
1  int m = 1,n;
2  n = m-(m=7);
3  printf("%d",n);
```

总结


运算符和表达式
类型转换
自动转换
强制转换
运算符
数学
关系 =，==
逻辑 只计算一半
特色运算符
++i，i++
位运算
i+=1;
运算符优先级
利用括号