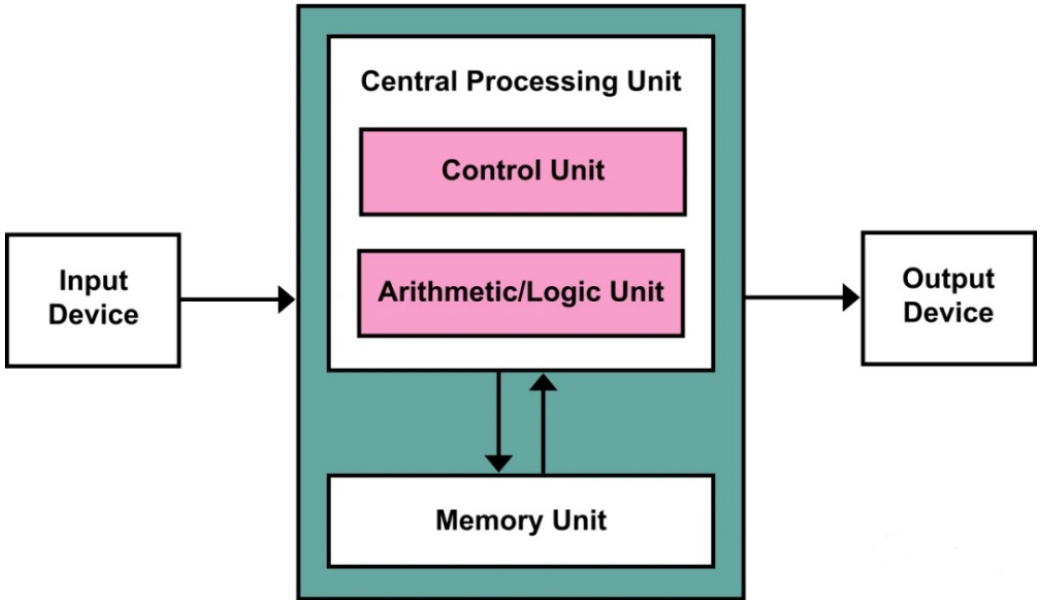
05 运算符，表达式和语句

内容提要

- 关键字：while，typedef
- 运算符：= - * / %++ --（typedef）
- C 的各种各样的运算符，其中包括用于普通数学运算的的运算符。
- 运算符的优先级以及术语“语句”和“表达式”的含义。
- 简单的 while 循环
- 复合语句，自动类型转换和类型指派
- 如何编写带有参数的函数



循环简介

1 循环简介

5.1 shoes1.c，5.2 shoes2.c

- C提供多种方法做重复计算，引入while循环
- while循环的原理
  - 当程序第1次到达while循环时，会检查圆括号中的条件是否为真

```c
/* shoes1.c -把鞋码转换成英寸 */
#include <stdio.h>
#define ADJUST 7.31     //常量
int main(void){
    const double SCALE = 0.333;  //另一种常量
    double shoe, foot;
    shoe = 9.0;
    foot = SCALE * shoe + ADJUST;
    printf("Shoe size (men's)    foot length\n");
    printf("%10.1f %15.2f inches\n", shoe, foot);
    return 0;
}
```

```c
/* shoes2.c -计算多个不同鞋码对应的脚长 */
#include <stdio.h>
#define ADJUST 7.31  //字符常量
int main(void)
{
    const double SCALE = 0.333;  // const 变量
    double shoe, foot;

    printf("Shoe size (men's) foot length\n");
    shoe = 3.0;
    while (shoe < 18.5)/*starting the while loop*/
    {                   /* start of block */
        foot = SCALE * shoe + ADJUST;
        printf("%10.1f %15.2f inches\n", shoe, foot);
        shoe = shoe + 1.0;
    }                  /* end of block */
    printf("If the shoe fits, wear it.\n");

    return 0;
}
```

基本运算符

2 基本运算符

- C用运算符(operator)表示算术运算
- 【运算符是程序设计里最基础的处理信息的操作！】

2.1 赋值运算符： =

- 赋值运算符
  - 右往左
- `i = i + 1;`
  - 在数学上，该语句没有任何意义
  - 找到名字为 i 的变量的值，对那个值加 1，然后将这个新值赋给名字为 i 的变量
- `2002 = bmw; //无效`
  - 赋值语句左侧只能是变量！

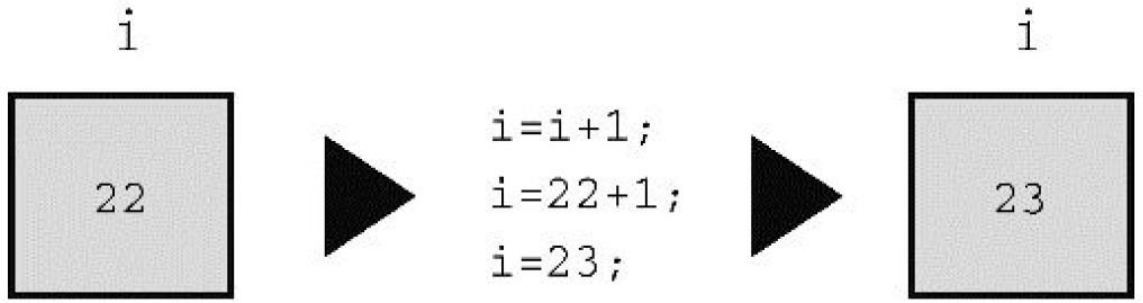


图5.1 语句i = i + 1;

几个术语：数据对象、左值、右值和运算符

- 数据对象【数据存储区，程序中存储/表达信息的实体】
  - 泛指数据存储区的术语，用于存储值的数据存储区域
  - 使用变量名是标识对象的一种方法
  - 还可以指定数组的元素、结构的成员，或者使用指针表达式(指针中存储的是它所指向对象的地址)
- 左值
  - 用于标识特定数据对象的名称或表达式
  - 它指定一个对象【用变量名标识】，可以引用内存中的地址【地址可以为表达式】
  - 可出现在赋值运算符的左边。可修改左值：可修改该对象的值
- 右值
  - 能赋值给可修改左值的量，且本身不是左值
  - 右值可以是常量，变量或者任何可以产生一个值的表达式
- 操作数
  - 被称为“项”(如，赋值运算符左侧的项)的就是运算对象(operand)。运算对象是运算符操作的对象

赋值运算符

- 5.3 golf.c
- 赋值运算符结果本身是一个值
- 特别的赋值语句，三重赋值
  - 赋值的顺序是从右往左
  - 首先把68赋给jane，然后再赋给tarzan，最后赋给cheeta

```c
/* golf.c -- golf tournament scorecard */
#include <stdio.h>
int main(void)
{
    int jane, tarzan, cheeta;

    cheeta = tarzan = jane = 68;
    printf("  cheeta   tarzan    jane\n");
    printf("First round score %4d %8d %8d\n", cheeta, tarzan, jane);

    return 0;
}
```

2.2/2.3 加法运算符 ： +/-

- 加法运算符(addition operator)用于加法运算，使其两侧的值相加
- 减法运算符(subtraction operator)用于减法运算，使其左侧的数减去右侧的数

2.4 符号运算符： - 和 +

- 一元运算符(unary operator)

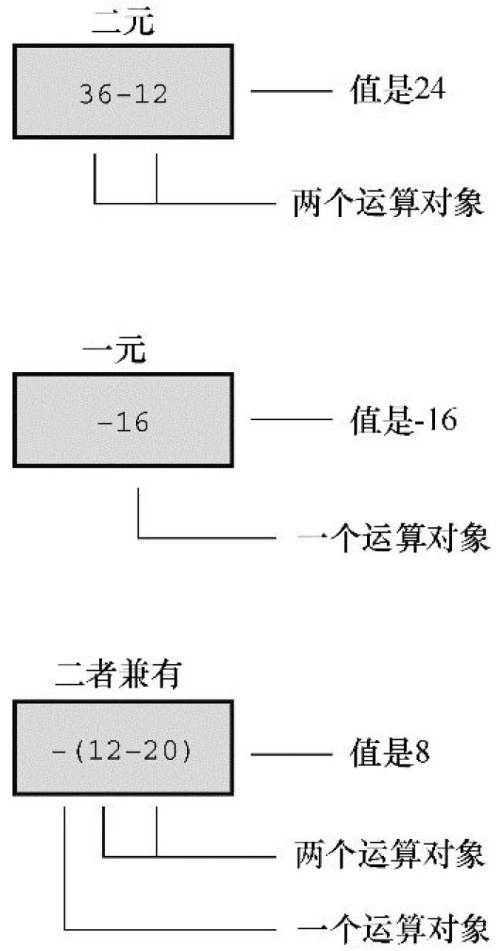


图5.2 一元和二元运算符

2.5 乘法运算符： *

5.4 squares.c

- C没有平方函数，使用乘法来计算平方

```c
/* squares.c --计算1 20 的平方 */
#include <stdio.h>
int main(void)
{
    int num = 1;

    while (num < 21)
    {
        printf("%4d %6d\n", num, num * num);
        num = num + 1;
    }

    return 0;
}
```

2.5 乘法运算符：*

- 程序清单5.5 wheat.c
- 演示指数增长的现象
  - 棋盘格放小麦，指数增长

```
1.  #define SQUARES 64  // squares on a checkerboard
2.  int main(void){
3.      const double CROP = 2E16; //world wheat … grains
4.      double current, total;
5.      int count = 1;
6.      total = current = 1.0; // start with one grain
7.      printf("%4d %13.2e %12.2e %12.2e\n", count,
    current, total, total/CROP);
8.      while (count < SQUARES) {
9.          count = count + 1;
10.         current = 2.0 * current;
11.         /* double grains on next square */
12.         total = total + current; /* update total */
13.         printf("%4d %13.2e %12.2e %12.2e\n", count,
    current, total, total/CROP);
14.     }
15.     return 0;
16. }
```

2.6 除法运算符：/

- 5.6 divide.c
- 整数除法和浮点数除法不同
  - 浮点数除法的结果是浮点数
  - 整数除法的结果是整数，没有小数部分
- 整数除法结果的小数部分被丢弃，这一过程被称为截断(truncation)

```c
/* divide.c -- divisions we have known */
#include <stdio.h>
int main(void)
{
    printf("integer division:  5/4   is %d \n", 5/4);
    printf("integer division:  6/3   is %d \n", 6/3);
    printf("integer division:  7/4   is %d \n", 7/4);
    printf("floating division: 7./4. is %1.2f \n", 7./4.);
    printf("mixed division:    7./4  is %1.2f \n", 7./4);

    return 0;
}
```

2.7 运算符的优先级

- 执行操作的顺序：运算符优先级决定
  - 每个运算符都有自己的优先级
  - 先优先级高，再优先级低
  - 如果两个运算符的优先级相同，则根据它们在语句中出现的顺序来执行
    - 对大多数运算符，按从左到右的顺序进行
    - =运算符除外
- 用表达式树(expression tree)来表示求值的顺序
- 表5.2

```
SCALE=2;
n=6;
butter=25.0+60.0*n/ SCALE;
```

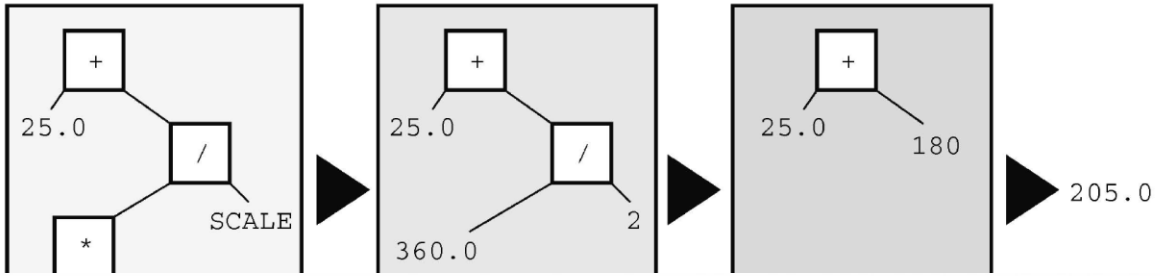


表 5.2 按优先级递减顺序排列的运算符

运 算 符	结 合 性
()	从左到右
+-（一元运算符）	从右到左
*/	从左到右
+-（二元运算符）	从左到右
=	从右到左

2.8 优先级和求值顺序

- 5.7 rules.c
- 当运算符共享一个运算对象时，优先级决定了求值顺序

```c
/* rules.c -- precedence test */
#include <stdio.h>
int main(void)
{
    int top, score;

    top = score = -(2 + 5) * 6 + (4 + 3 * (2 + 3));
    printf("top = %d, score = %d\n", top, score);

    return 0;
}
```

其他运算符

3 其他运算符

3.1 sizeof运算符和size_t类型

- 程序清单5.8 sizeof.c
- sizeof 运算符以字节为单位返回其操作数的大小

```
1.  // sizeof.c -- uses sizeof operator
2.  // C99 %z modifier - try %u or %lu if you lack %zd
3.  #include <stdio.h>
4.  int main(void)
5.  {
6.      int n = 0;
7.      size_t intsize;
8.
9.      intsize = sizeof (int);
10.     printf("n = %d, n has %zd bytes; all ints have %zd
    bytes.\n",
11.            n, sizeof n, intsize );
12.
13.     return 0;
14. }
```

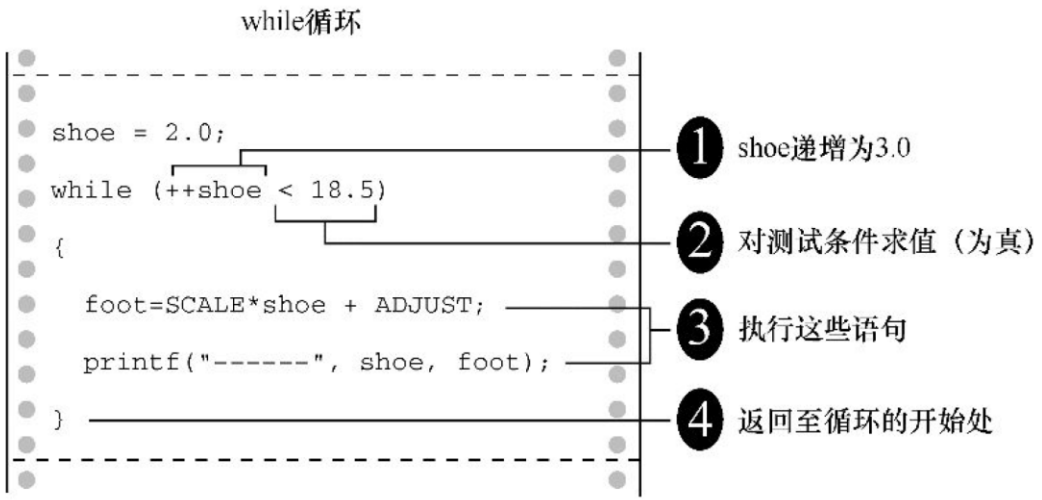
3.2 取模运算符%

- 程序清单5.9 min_sec.c
- 求模运算符(modulus operator)用于整数运算
- 判定整除，取模为0
- 无论何种情况，只要a和b都是整数值，便可通过a - (a/b)*b来计算a%b
  - 辗转相除法

```
1.  #define SEC_PER_MIN 60   // seconds in a minute
2.  int main(void){
3.      int sec, min, left;
4.      printf("Convert seconds!\n");
5.      printf("Number of seconds (<=0 to quit):\n");
6.      scanf("%d", &sec); // read number of seconds
7.      while (sec > 0){
8.          min = sec / SEC_PER_MIN;  // truncated minutes
9.          left = sec % SEC_PER_MIN; // seconds left over
10.         printf("%d : %d m, %d s.\n", sec,  min, left);
11.         printf("Enter next value (<=0 to quit):\n");
12.         scanf("%d", &sec);
13.     }
14.     printf("Done!\n");
15.     return 0;
16. }
```

3.3/3.4 增量和减量运算符： ++ 和 --

- 程序清单5.10 add_one.c
- 递增运算符(increment operator)将其运算对象递增1
  - 前缀(prefix)模式
  - 后缀(postrix)模式
- 紧凑结构的代码让程序更简洁，可读性更高




```c
/* add_one.c -- incrementing: prefix and postfix */
#include <stdio.h>
int main(void)
{
    int ultra = 0, super = 0;

    while (super < 5)
    {
        super++;
        ++ultra;
        printf("s = %d, u = %d \n", super, ultra);
    }

    return 0;
}
```

3.3/3.4 增量和减量运算符： ++ 和 --

➢ 5.11 post_pre .c

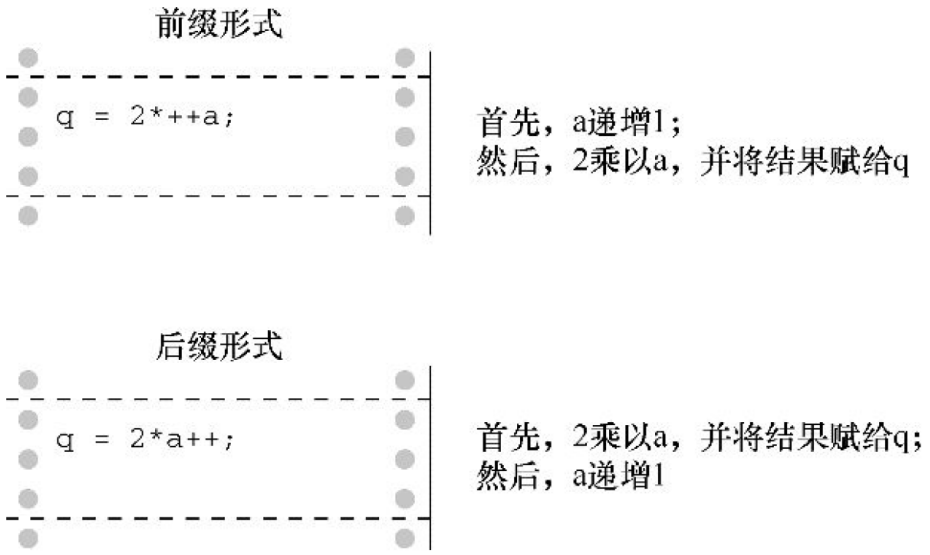


图5.5　前缀和后缀

```c
/* post_pre.c -- postfix vs prefix */
#include <stdio.h>
int main(void)
{
    int a = 1, b = 1;
    int a_post, pre_b;

    a_post = a++;  // value of a++
    pre_b = ++b;   // value of ++b
    printf("a  a_post   b   pre_b \n");
    printf("%1d %5d %5d %5d\n", a, a_post, b, pre_b);

    return 0;
}
```

3.3/3.4 增量和减量运算符： ++ 和 --

➢ 5.12 bottles.c(循环)

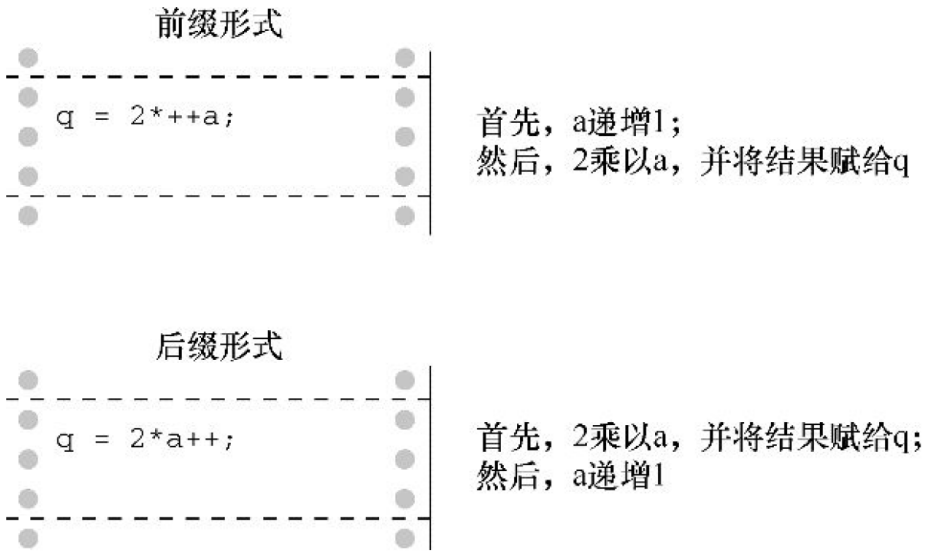


图5.5 前缀和后缀

```c
#include <stdio.h>
#define MAX 100
int main(void)
{
    int count = MAX + 1;

    while (--count > 0) {
        printf("%d bottles of spring water on the wall, %d bottles of spring water!\n", count, count);
        printf("Take one down and pass it around,\n");
        printf("%d bottles of spring water!\n\n", count - 1);
    }

    return 0;
}
```

3.5 优先级

- 递增运算符和递减运算符都有很高的结合优先级，只有圆括号的优先级比它们高
  - x*y++表示的是(x)*(y++)，而不是(x*y)++

```
y = 2;
n = 3;
nextnum = (y + n++)*6;
```

当n++是表达式的一部分时，表示 “先使用n，然后将它的值增加

当++n是表达式的一部分时，表示“先将n的值增加，然后再使用它

3.6 不要自作聪明

- 在C语言中，编译器可以自行选择先对函数中的哪个参数求值
  - 这样提高了编译器的效率，但如果在函数的参数中使用了递增运算符，就会出现问题

```
1. while (num < 21)
2. {
3.     printf("%10d %10d\n", num, num*num++);
4. }
```

- 打印num，然后计算num*num得到平方值，最后把num递增1。只能在某些系统上能正常运行。该程序的问题
  - 当printf()获取待打印的值时，可能先对最后一个参数(num*num++)求值，这样在获取其他参数的值之前就递增了num。
    - 6 25
  - 它甚至可能从右往左执行，对最右边的num(++作用的num)使用5，对第2个num和最左边的num使用6
    - 6 30

表达式和语句

4 表达式和语句

- 表达式(expression)和语句(statement)

4.1 表达式

- 表达式(expression)
  - 由运算符和运算对象组成(运算对象是运算符操作的对象)
- 运算对象
  - 可以是常量、变量或二者的组合。一些表达式由子表达式(subexpression)组成(子表达式即较小的表达式)
- 每一个表达式都有一个值
  - 赋值运算符构成

表 5.3 一些表达式和它们的值

表 达 式	值
-4+6	2
c=3+8	11
5>3	1
6+（c=3+8）	17

4.2 语句

- 程序(program)是一系列带有某种必需的标点的语句集合。一个语句是一条完整的计算机指令
  - 在 C 中，语句用结束处的一个分号标识。
- 语句(statement)是构造程序的基本成分
  - 一条语句相当于一条完整的计算机指令
  - 在C中，大部分语句都以分号结尾
  - 最简单的语句是空语句
- 语句中的子表达式y = 5是一条完整的指令，但是它只是语句的一部分。因为一条完整的指令不一定是一条语句，所以分号用于识别在这种情况下的语句(即，简单语句)

```
1.  legs = 4

2.  legs = 4;

3.  ; //空语句

4.  //没有实质性作用的语句
5.  8;
6.  3 + 4;

7.  //语句可以改变值或调用函数
8.  x = 25;
9.  ++x;
10. y = sqrt(x);

11. x = 6 + (y = 5);
```

语句

- 5.13 addemup.c
- 声明创建了名称和类型，并为其分配内存位置
  - 注意，声明不是表达式语句
- 赋值表达式语句在程序中很常用：它为变量分配一个值
- 赋值表达式语句的结构
  - 一个变量名，后面是一个赋值运算符，再跟着一个表达式，最后以分号结尾
- 函数表达式语句会引起函数调用

```
1.  /* addemup.c -- five kinds of statements */
2.  #include <stdio.h>
3.  int main(void) /* finds sum of first 20 ntegers */
4.  {
5.      int count, sum;  /* declaration statement */
6.
7.      count = 0;  /* assignment statement */
8.      sum = 0; /* ditto */
9.      while (count++ < 20)      /* while */
10.         sum = sum + count;    /* statement */
11.     printf("sum = %d\n", sum);// function statement
12.
13.     return 0;                 /* return statement */
14. }
```

语句

- 副作用(side effect)
  - 对数据对象或文件的修改
- 序列点(sequence point)
  - 程序执行的点，在该点上，所有的副作用都在进入下一步之前发生
- 完整表达式(full expression)
  - 指这个表达式不是另一个更大表达式的子表达式

4.3 复合语句（代码块）

- 复合语句(compound statement)
  - 使用花括号组织起来的两个或更多的语句，也被称为一个代码块(block)

- 风格提示{}
  - 使用缩进可以为读者指明程序的结构
  - 对编译器而言，这两种风格完全相同

- 表达式由运算符和运算对象组成。最简单的表达式是不带运算符的一个常量或变量
- 语句可分为简单语句和复合语句。简单语句以一个分号结尾
- 复合语句(或块)由花括号括起来的一条或多条语句组成

类型转换

5 类型转换

- 通常，语句和表达式通常应该只使用一种类型的变量和常量，否则，需要使用类型转换
  - 程序清单5.14 convert.c

```c
#include <stdio.h>
int main(void){
    char ch;
    int i;
    float fl;

    fl = i = ch = 'C'; /* line 9  */
    printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl);
    ch = ch + 1;
    i = fl + 2 * ch;
    fl = 2.0 * ch + i;
    printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl);
    ch = 1107;
    printf("Now ch = %c\n", ch);
    ch = 80.89;
    printf("Now ch = %c\n", ch);

    return 0;
}
```

提升

- 提升(promotion)
  - 有符号和无符号的char和short类型都将自动被转换为int
  - float将被自动转换为double
  - 在包含两种数据类型的任何运算里，两个值都被转换成两种类型里较高的级别
- 类型级别从高到低的顺序是 long double，double，float，unsigned long long，long long，unsigned long，long，unsigned int和int 。
- 如果待转换的值与目标类型不匹配
  - 目标类型是无符号整型，且待赋的值是整数时，额外的位将被忽略
  - 如果目标类型是一个有符号整型，且待赋的值是整数，结果因实现而异
  - 如果目标类型是一个整型，且待赋的值是浮点数，该行为是未定义的
  - 程序清单5.14 convert.c

降级

- 在赋值语句里，计算的最后结果被转换成将要被赋予值的那个变量的类型
- 当作为函数的参数被传递时，char和short会被转换为int，float会被转换为double

- 提升通常是一个平滑的无损害的过程，但是降级可能导致真正的问题

指派运算符

- 通常，应该避免自动类型转换，尤其是类型降级

- 强制类型转换(cast)
  - 在某个量的前面放置用圆括号括起来的类型名，该类型名即是希望转换成的目标类型。圆括号和它括起来的类型名构成了强制类型转换运算符(cast operator)
  - `mice =(int)1.6 + (int)1.7;`

- C语言的原则是避免给程序员设置障碍，但是程序员必须承担使用的风险和责任

带有参数的函数

6 带有参数的函数

- 5.15 pound.c
- 功能：打印指定数量的#号
- 声明参数就创建了被称为形式参数(formal argument或formal parameter，简称形参)的变量
  - 形式参数是int类型的变量n。像pound(10)这样的函数调用会把10赋给n。在该程序中，调用pound(times)就是把times的值(5)赋给n
- 称函数调用传递的值为实际参数(actual argument或actual parameter)，简称实参
  - 函数调用pound(10)把实际参数10传递给函数，然后该函数把10赋给形式参数(变量n)。也就是说，main()中的变量times的值被拷贝给pound()中的新变量n

```
1.  void pound(int n);//ANSI function prototype
    declaration
2.  int main(void){
3.      int times = 5;
4.      char ch = '!';   // ASCII code is 33
5.      float f = 6.0f;
6.
7.      pound(times);    // int argument
8.      pound(ch);       // same as pound((int)ch);
9.      pound(f);        // same as pound((int)f);
10.
11.     return 0;
12. }

13. void pound(int n)    // ANSI-style function header
14. {                    // says takes one int argument
15.     while (n-- > 0)
16.         printf("#");
17.     printf("\n");
18. }
```

一个示例程序

7 一个示例程序

5.16 running.c，使用了min_sec程序(程序清单5.9)中的方法把时间转换成分钟和秒。

```
1.  const int S_PER_M = 60;   // seconds in a minute
2.  const int S_PER_H = 3600; // seconds in an hour
3.  const double M_PER_K = 0.62137; // miles(kilometer)
4.  int main(void){
5.      double distk, distm; // distance, km and in miles
6.      double rate;     // average speed in mph
7.      int min, sec;    // minutes and seconds
8.      int time;        // running time in seconds only
9.      double mtime;    // time in seconds for one mile
10.     int mmin, msec; // times for one mile
11.     printf(“Converts your time for a metric race\n");
12.     printf(“Enter,the distance run.\n");
13.     scanf("%lf", &distk);  // %lf for type double
14.     printf(“Enter the time.\n");
15.     scanf("%d %d", &min, &sec);
16.     // converts time to pure seconds
17.     time = S_PER_M * min + sec;
18.     distm = M_PER_K * distk; // converts kilometers
19.     // miles per sec x sec per hour = mph
20.     rate = distm / time * S_PER_H;
21.     // time/distance = time per mile
22.     mtime = (double) time / distm;
23.     mmin = (int) mtime / S_PER_M;
24.     msec = (int) mtime % S_PER_M;
25.     printf("You ran %1.2f km (%1.2f miles) in %d min,
    %d sec.\n", distk, distm, min, sec);
26.     printf(“Running a mile in %d min, ", mmin);
27.     printf("%d sec.\nYour average speed was %1.2f
    mph.\n",msec, rate);
28.
29.     return 0;
30. }
```

关键概念

8 关键概念

- C通过运算符提供多种操作。每个运算符的特性包括运算对象的数量、优先级和结合律。当两个运算符共享一个运算对象时，优先级和结合律决定了先进行哪项运算
- 每个C表达式都有一个值
- 虽然C允许编写混合数值类型的表达式，但是算术运算要求运算对象都是相同的类型。因此，C会进行自动类型转换
- 不要养成依赖自动类型转换的习惯，应该显式选择合适的类型或使用强制类型转换。
  - 避免出现不必要的自动类型转换

9 总结

- 运算符需要一个或多个运算对象才能完成运算生成一个值
  - 只需要一个运算对象的运算符(如负号和sizeof)称为一元运算符
  - 需要两个运算对象的运算符(如加法运算符和乘法运算符)称为二元运算符
- 表达式由运算符和运算对象组成
  - 在C语言中，每个表达式都有一个值，包括赋值表达式和比较表达式
  - 运算符优先级规则决定了表达式中各项的求值顺序。当两个运算符共享一个运算对象时，先进行优先级高的运算。如果运算符的优先级相等，由结合律(从左往右或从右往左)决定求值顺序
- 大部分语句都以分号结尾
  - 最常用的语句是表达式语句
  - 用花括号括起来的一条或多条语句构成了复合语句(或称为块)
  - while语句是一种迭代语句，只要测试条件为真，就重复执行循环体中的语句
- 在C语言中，许多类型转换都是自动进行的
  - 在混合类型的运算中，较小类型会被转换成较大类型
- 定义带一个参数的函数时，便在函数定义中声明了一个变量，或称为形式参数
  - 然后，在函数调用中传入的值会被赋给这个变量