It makes C more powerful than maths.

C 程 序 设 计

C Programming

5

操作符、表达式和语句

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 煒

(特色化示范性软件学院) 博士,副教授

School of Informatics Wei Huang

知识框架

- 利用循环解决问题
- 操作符
  - 赋值、四则运算（加减乘除模）、复合（四则与赋值结合）
  - 强制类型转换、尺寸、增减量
  - 优先级
- 表达式
  - 序列点
- 语句

内容纲要

```c
/* shoes1.c -- converts a shoe size to inches */
#include <stdio.h>
#define ADJUST 7.31                    // one kind of symbolic
constant
int main(void)
{
    const double SCALE = 0.333;  // another kind of symbolic
constant
    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;
}
```

```
Shoe size (men's)    foot length
       9.0              10.31 inches
```

实例

- 题目
  - 输出3~18号的男鞋足长
- 样例输入：无
- 样例输出

```
Shoe size (men's)     foot length
        3.0             8.31 inches
        4.0             8.64 inches
（此处忽略一部分）
       18.0            13.30 inches
If the shoe fits, wear it.
```

- 最直接的想法
  - 尺寸从3开始，若小于等于18，依次完成
    - 根据号码计算足长；号码增加1步；回到循环

```c
/* shoes2.c -- calculates foot lengths for several sizes */
#include <stdio.h>
#define ADJUST 7.31              // one kind of symbolic constant
int main(void)
{
    const double SCALE = 0.333; // another kind of symbolic constant
    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;
    }
    printf("If the shoe fits
    return 0;
}
```

有经验的程序员不会在此写18.5这样令人不解的数

指定宽度可以使得输出更美观

```
Shoe size (men's)    foot length
       3.0              8.31 inches
       4.0              8.64 inches
（此处忽略一部分）
      18.0             13.30 inches
If the shoe fits, wear it.
```

内容纲要

1	利用循环解决问题
2	操作符
3	表达式
4	语句
5	带参量的函数

一元和二元操作符

- 一元操作符（ unary operators ）

```
<操作符> <操作数>
```

  - 例如： -16 +7.5 &op *pointer

- 二元操作符（ binary operators ）

```
<操作数1> <操作符> <操作数2>
```

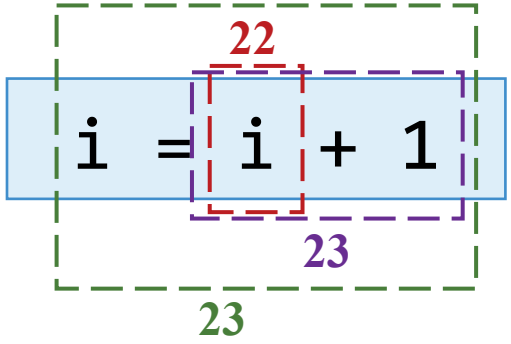
  - 赋值操作符是一个二元操作符
  - 例如： 5-3 8+17 3*9 2.5/0.3

赋值操作符（=）

- 赋值操作符常用于赋值表达式中
  - 赋值表达式的格式：<变量> = <值>

Times字体的尖括号表示该内容可选但不可忽略

    - 赋值操作符左侧称为左值，赋值操作符右侧称为右值。
- 作用：计算右值，用其作为左值的新值
  - A=B 不表示“A等于B”，而是“令A等于B的值”
- 示例：
  - 原来 i 的值为22，运行后，i的新值为23



赋值操作符（=）

- 格式
  - 赋值符号左值应为变量名，使用常量将出现编译错误

```
2018=a;
```

    - 提示

```
m.c: In function ‘main’:
m.c:6:3: error: lvalue required as left operand of assignment
```

- 赋值表达式的值为右值的值。
  - 多重赋值

```
a=b=5
```

    - 作用：先计算b=5，再计算a=（b=5的值，即5）。

```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;
}
```

等价于：

```c
jane=68;
tarzan=jane;
cheeta=tarzan;
```

```
                      cheeta   tarzan    jane
First round score       68       68       68
```

四则操作符

- 四则运算
  - 作用：与算术课相同
  - 整除
    - 商和余数的值按被除数和除数绝对值相除
    - 商的符号按被除数与除数是否同号，余数符号同被除数

名称	加	减	乘	除	模(求余)
操作符	+	-	*	/	%

- 算术转换
  - 当两个操作数的数据类型相同，结果的数据类型不变
  - 当两个操作数的数据类型不同，操作数应做转换
    - 如果其中一个操作数更高，则另一个操作数先转换为该类型再计算

优先级顺序	long double	double	float	long long int	int	short	char

```c
/* squares.c -- produces a table of first 20 squares */
#include <stdio.h>
int main(void)
{
    int num = 1;

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

    return 0;
}
```

```
   1      1
   2      4
   3      9
（此处忽略一部分）
  20    400
```

麦子与国王问题

- **题目**

  相传古印度舍罕王的宰相达依尔是国际象棋的发明者。国王因其贡献，问他想要什么奖励。达依尔说：“请在国际象棋棋盘上按规律摆上麦子：第1格1粒，第2格2粒，……，后面一格的麦子总是前一格麦子数的两倍，摆满整个棋盘赐我，我就感恩不尽了。”国王一想，这还不容易，就答应他了。其实，整个印度全国的麦子都给他也不够！

- **归纳规律**

$$n = \sum_{i=1}^{64} 2^{i-1}$$

```
total = current = 1.0;
while (count < SQUARES)
{
    count = count + 1;
    current = 2.0 * current;
    total = total + current;
}
```

```c
/* wheat.c -- exponential growth */
#include <stdio.h>
#define SQUARES 64                  // squares on a checkerboard
int main(void)
{
    const double CROP = 2E16;  // world wheat production in
wheat grains
    double current, total;
    int count = 1;

    printf("square     grains        total     ");
    printf("fraction of \n");
    printf("             added       grains    ");
    printf("world total\n");
    total = current = 1.0; /* start with one grain   */
```

进入循环之前，应检查初始状态是否设置正确。
如乘法的初始值为0将出错。

```
    printf("%4d %13.2e %12.2e %12.2e\n", count, current,
total, total/CROP);
    while (count < SQUARES)
    {
        count = count + 1;
        current = 2.0 * current;
        /* double grains on next square */
        total = total + current;      /* update total */
        printf("%4d %13.2e %12.2e %12.2e\n", count, current,
              total, total/CROP);
    }
    printf("That's all.\n");

    return 0;
}
```

循环的执行条件

循环条件的变化
（否则可能永不退出循环）

相同内容的并排输出要对齐，增加可读性。一般通过格式字符串中的宽度来控制。

```
square      grains          total       fraction of
            added           grains      world total
   1      1.00e+000      1.00e+000       5.00e-017
   2      2.00e+000      3.00e+000       1.50e-016
   3      4.00e+000      7.00e+000       3.50e-016
   4      8.00e+000      1.50e+001       7.50e-016
（此处忽略一部分）
  63      4.61e+018      9.22e+018       4.61e+002
  64      9.22e+018      1.84e+019       9.22e+002
That's all.
```

继续循环，将出现溢出

```
 126      4.25e+037      8.51e+037       4.25e+021
 127      8.51e+037      1.70e+038       8.51e+021
 128      1.70e+038      1.#Je+000       1.#Je+000
 129      1.#Je+000      1.#Je+000       1.#Je+000
```

1.#INF0e+000 配合 %13.2e 输出 1.#Je+000

```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;
}
```

这里的1不是整数部分长度

7 是整型常量，
7.是双精度型常量

```
integer division:  5/4   is 1
integer division:  6/3   is 2
integer division:  7/4   is 1
floating division: 7./4. is 1.75
mixed division:    7./4  is 1.75
```

```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;
}
```

```
integer division:  5/4   is 1
integer division:  6/3   is 2
integer division:  -7/4   is -1
floating division: -7./4. is -1.75
mixed division:    7./4  is 1.75
```

7 是整型常量，
7.是双精度型常量

```c
/* convert.c -- automatic type conversions */
#include <stdio.h>
int main(void)
{
    char ch;
    int i;
    float fl;
    fl = i = ch = 'C';
    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;
}
```

```
convert.c: In function ‘main’:
convert.c:15:5: warning: overflow in implicit
constant conversion [-Woverflow]
     ch = 1107;
     ^
```

```
ch = C, i = 67, fl = 67.00
ch = D, i = 203, fl = 339.00
Now ch = S
Now ch = P
```

强制类型转换

- 格式 (<类型>) <表达式>
  - 作用：将表达式的值从原有类型转换到另一种类型
    - 一般地，不进行强制类型转换结果也正确的，不必强制类型转换
  - 例如：mice为double类型。

表达式语句	运行后mice的值	说明
`mice = 1.6 + 1.7 + 1.8;`	5.1	直接相加。
`mice = (int)1.6 + 1.7 + (int)1.8;`	3.7	第1、3项值为1。
`mice = (int)(1.6 + 1.7) + (int)1.8;`	4.0	第1项值为3。
`mice = int(1.6 + 1.7) + int(1.8);`	编译错误	这是C++的语法，不是C的语法。

复合赋值：自增减乘除模等

- 格式

<变量名><二元操作符>=<表达式>

  - 第一个加数或乘数，或被减数、被除数与左值相同时简化

自增	等价
op+=val;	op=op+val;
op-=val;	op=op-val;
op*=val;	op=op*val;
op/=val;	op=op/val;
op%=val;	op=op%val;

- 注意
  - 左值应为变量名

```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;
}
```

```
top = -23, score = -23
```

计算过程

- 表达式求值过程

步骤	表达式	变动
0	top = score = -(2 + 5) * 6 + (4 + 3 * (2 + 3))	
1	top = score = -7 * 6 + (4 + 3 * (2 + 3))	
2	top = score = -7 * 6 + (4 + 3 * 5)	
3	top = score = -7 * 6 + (4 + 15)	
4	top = score = -7 * 6 + 19	
5	top = score = -7 * 6 + 19	
6	top = score = -42 + 19	
7	top = score = -23	
8	top = -23	score: -23
9	-23	top: -23

尺寸操作符

- 格式

```
sizeof(<表达式|类型>)
```

格式中的竖线指两侧选一

  - sizeof后为类型名或表达式，其中表达式并不计算

sizeof后	需要括号	示例	示例
类型名	必须加	sizeof(int)	sizeof(double)
表达式	可加可不加	sizeof(a)	sizeof 'A'

- 类型：size_t
  - size_t长度与操作系统架构有关
    - 操作系统架构是32位时，size_t是unsigned int型
    - 操作系统架构是64位时，size_t是unsigned __int64型

```c
// sizeof.c -- uses sizeof operator
// uses C99 %z modifier -- try %u or %lu if you lack %zd
#include <stdio.h>
int main(void)
{
    int n = 0;
    size_t intsize;

    intsize = sizeof (int);
    printf("n = %d, n has %zd bytes; all ints have %zd bytes.\n",
           n, sizeof n, intsize );

    return 0;
}
```

有经验的程序员不会对同一类量使用不同的实现风格。

```
n = 0, n has 4 bytes; all ints have 4 bytes.
```

```c
// min_sec.c -- converts seconds to minutes and seconds
#include <stdio.h>
#define SEC_PER_MIN 60              // seconds in a minute
int main(void)
{
    int sec, min, left;

    printf("Convert seconds to minutes and seconds!\n");
    printf("Enter the number of seconds (<=0 to quit):\n");
    scanf("%d", &sec);              // read number of seconds
    while (sec > 0)
    {
        min = sec / SEC_PER_MIN;  // truncated number of minutes
        left = sec % SEC_PER_MIN; // number of seconds left over
        printf("%d seconds is %d minutes, %d seconds.\n", sec,
               min, left);
```

```
        printf("Enter next value (<=0 to quit):\n");
        scanf("%d", &sec);
    }
    printf("Done!\n");
    return 0;
}
```

```
Convert seconds to minutes and seconds!
Enter the number of seconds (<=0 to quit):
255↲
255 seconds is 4 minutes, 15 seconds.
Enter next value (<=0 to quit):
53↲
53 seconds is 0 minutes, 53 seconds.
Enter next value (<=0 to quit):
15↲
15 seconds is 0 minutes, 15 seconds.
Enter next value (<=0 to quit):
-3↲
Done!
```

增量减量操作符

- 格式（ op必须是变量名 ）

整个表达式的值和op的值不是一回事

名称	表达式	表达式的值	运行后op的值
前缀增量操作符	++op	op+1的值	op+1的值
后缀增量操作符	op++	op的值	op+1的值
前缀减量操作符	--op	op-1的值	op-1的值
后缀减量操作符	op--	op的值	op-1的值

– 前缀先自增减再求值；后缀先求值再自增减

```
printf("%d, %d\n", ++pre, post--);
```

```
pre = pre + 1;
printf("%d, %d\n", pre, post);
post = post - 1;
```

```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("super = %d, ultra = %d \n", super, ultra);
    }

    return 0;
}
```

```
super = 1, ultra = 1
super = 2, ultra = 2
super = 3, ultra = 3
super = 4, ultra = 4
super = 5, ultra = 5
```

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

    return 0;
}
```

a的值是1，所以a++的值是1，
a_post的值是1，运行后a的值是2

b的值是1，所以++b的值是2，
pre_b的值是2，运行后b是2

```
a  a_post   b   pre_b
2     1     2     2
```

增量减量操作符

- 增量减量操作符与循环语句混合使用可控制循环次数

```
int i = 0;              // initialized value
while (i < 5)    // loop condition
{
   // statements here ...
   i++;          // count 1 by 1 in loop
}
```

  - 等价于（for循环）

```
for (i = 0; i < 5; i++) // init. value; loop condition; count
{
    // statements here ...
}
```

```c
/* bottles.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);
```

```
（此处省略数行）
2 bottles of spring water on the wall, 2 bottles of spring water!
Take one down and pass it around,
1 bottles of spring water!

1 bottles of spring water on the wall, 1 bottles of spring water!
Take one down and pass it around,
0 bottles of spring water!
```

```c
}
```

运算符的优先级

- 下表为与本节相关的优先级顺序
  - 一元优先于二元，算术优先，赋值最低
  - 尊重常用：以常用方法不加括号为先。

序号	符号	说明	序号	符号	说明
1 →	后缀++ --	后缀增减量	3	* / %	算术运算：乘除模
	( )	函数调用	4	+ -	算术运算：加减
	[ ]	数组下标	14 ←	=	赋值
2 ←	前缀++ --	前缀增减量		+= -= *= /= %=	自增自减自乘自除自模
	+ -	正负号			
	(type)	强制类型转换			
	sizeof	存储空间			

不要自作聪明

- 有的考试喜欢考一些钻牛角尖的问题
- 但程序员编程时对增量减量的使用应适可而止
  - 程序员有义务让读者看懂（除非你要保护技术秘密）
  - 适当合并增量可以简化代码，易于理解
  - 过度合并增量降低代码质量，队友头疼

```
ans /= num / 2 + 6 * ( 1 + num++ + ++num ); // ans=7, num=3
```

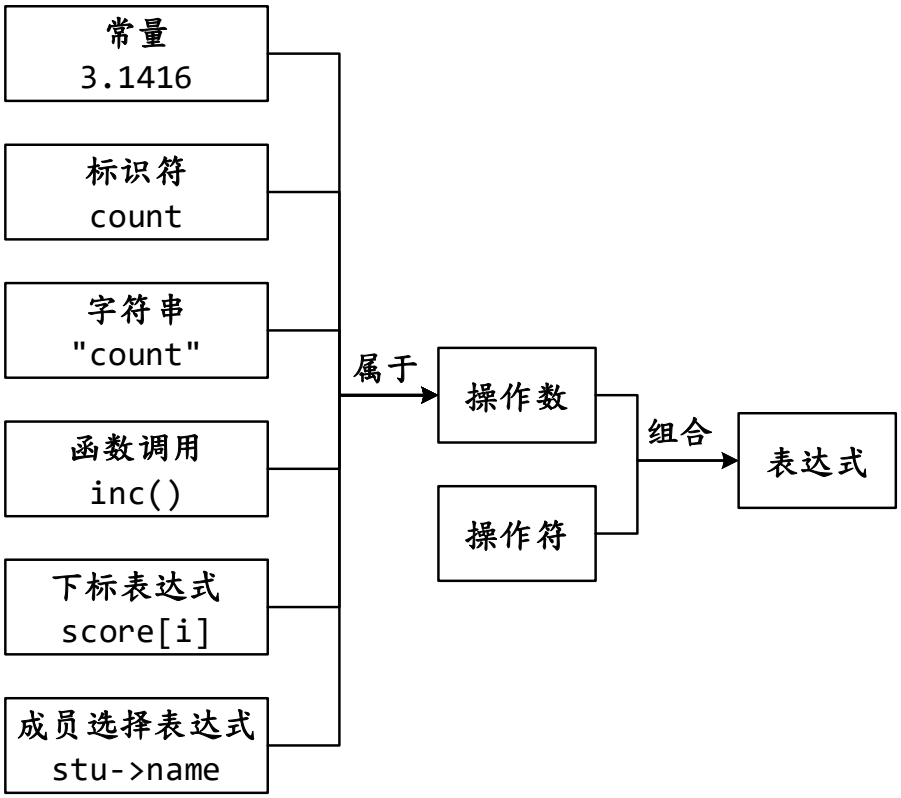
内容纲要

1	利用循环解决问题
2	操作符
3	表达式
4	语句
5	带参量的函数

表达式

- 表达式是一系列的操作符和操作数的任意组合，执行
  - 主要作用：计算值
  - 主要作用：指定对象或函数
  - 副作用：可能改变别人的值

表达式	值
-4+6	2
c=3+8	11
5>3	1
6+(c=3+8)	17



内容纲要

语句

- 定义
  - C 语句由标记、表达式和其他语句组成。构成另一个语句的组成部分的语句称为封闭语句的“体”。
- 分类
  - 声明语句、标记语句、复合语句、表达式语句
  - 选择语句、循环语句、跳转语句

语句示例

- 声明语句

```
int count;
int inc(int a);
```

- 标记语句

```
    if (i == 5)
       goto stop;
    ...
stop:
     printf_s( "Jumped to stop. i = %d\n", i );
```

- 表达式语句

```
y = sqrt(x);
length = 5;
```

语句示例

- 复合语句
  - 以花括号为界限，自上而下执行

```
{
    line[i] = x;
    x*=2;
    i++;
}
```

- 选择语句

```
if ( i > 0 )
    y = x / i;
else
    y = f( x );
```

语句示例

- 循环语句

```
for (i=0; i<10; i++)
{
    line[i] = x;
    x*=2;
}
```

- 跳转语句

```
for (i=0; i<10; i++)
{
    if (x>20)
        break;
    line[i] = x;
    x*=2;
}
```

语句示例

- 空语句

```
for ( i = 0; i < 10; line[i++] = 0 )
    ;
```

- 返回语句

```
void draw( int i, long long ll )
{
    printf( "i = %d, ll = %lld\n", i, ll );
    return;
}
```

- 其它语句

```c
/* addemup.c -- five kinds of statements */
#include <stdio.h>
int main(void)          /* finds sum of first 20 integers */
{
    int count, sum;             /* declaration statement    */

    count = 0;                  /* assignment statement     */
    sum = 0;                    /* ditto                    */
    while (count++ < 20)        /* while                    */
        sum = sum + count;      /*     statement            */
    printf("sum = %d\n", sum);/* function statement         */

    return 0;                   /* return statement         */
}
```

```
sum = 210
```

副作用（side effect）

- 赋值表达式的副作用是对数据对象或文件的修改
  - 表达式的主要作用是对表达式求值
  - 造成length的改变是求值过程中的副作用。

```
length = 5;
```

序列点（sequence points）

- 序列点是要求前项表达式求值与副作用执行完毕的点

```
for (i = 0, j = i; i <= j; i++)
```

这里i的值是多少？前次赋值i=0在哪里生效？

  - 序列点间多次修改对象的值，执行顺序不确定。

```
for (i = 0, j = ++i + i++; i <= j; i++)
```

在 VS 中，i为2，j为2；
在GCC中，i为2，j为3。

- C语言表达式求值的原则
  - 根据操作符的优先级和结合性确定表达式的求值顺序，
  - 但求值时需保证序列点左边的操作数先于其右边求值。

序列点

序列点	说明
逻辑与	如果左操作数的结果为false(零)，则不计算另一个操作数。
逻辑或	如果左操作数的结果为true(非零)，则不计算另一个操作数。
逗号操作符	始终计算逗号操作符的两个操作数。
函数调用	计算函数的所有参数，并在输入函数前完成所有副作用。未指定参数之间的计算顺序。
条件操作符	计算条件操作符的第一个操作数，在继续前完成所有副作用。
初始化表达式	声明语句中的初始化的末尾
表达式	副作用计算该表达式，并且此计算后面有一个序列点。
选择语句中的控制表达式	完全计算选择（if 或 switch）语句中的控制表达式，并在执行依赖于选择的代码之前完成所有副作用。
循环语句中的控制表达式	完全计算选择（for，while 或 do-while）语句中的控制表达式，并在下一次迭代中的任何语句执行前完成所有副作用。
返回语句中的表达式	完全计算该表达式，并在控制返回调用函数之前完成所有副作用。

内容纲要

函数

- 将可重复使用的代码封装为函数，变化量为参量

```c
#include <stdio.h>

int main(void)
{
    printf("Shift: %6.3lf.\n", 2.3 * 6 + 1.0 / 2 * 1.2 * 6 * 6);
    printf("Shift: %6.3lf.\n", 3 * 1.5 + 1.0 / 2 * 3 * 1.5 * 1.5);
    printf("Shift: %6.3lf.\n", 1 * 0.8 + 1.0 / 2 * 5 * 0.8 * 0.8);
    return 0;
}
```

```c
double shift(double v0, double a, double t) {
    return v0 * t + 1.0 / 2 * a * t * t;
}
```

```c
printf("Shift: %6.3lf.\n", shift(2.3, 1.2, 6));
printf("Shift: %6.3lf.\n", shift(3, 3, 1.5));
printf("Shift: %6.3lf.\n", shift(1, 5, 0.8));
```

带参数的函数

- 函数需要先声明，有定义，再使用
  - 声明：函数原型

```
double shift(double v0, double a, double t);
```

    - 原型的形式参量(formal parameters)的名字可以省略，但不建议。
    - 如果函数体书写在调用之前，则原型可以省略，但不建议。
  - 定义：函数的实现代码
    - 形式参量：用来接收调用函数时传递的参数。
    - 函数中改变形式参量的值不影响调用该函数的函数中的变量值。

```
double shift(double v0, double a, double t) {
    return v0 * t + 1.0 / 2 * a * t * t;
}
```

```c
/* pound.c -- defines a function with an argument   */
#include <stdio.h>
void pound(int n);    // ANSI function prototype declaration
int main(void)
{
    int times = 5;
    char ch = '!';    // ASCII code is 33
    float f = 6.0f;
    pound(times);     // int argument
    pound(ch);        // same as pound((int)ch);
    pound(f);         // same as pound((int)f);
    return 0;
}
```

函数的原型声明，一般以全局形式出现

函数的调用。其中，times, ch, f 分别为三次调用时的实际参数。

```
void pound(int n)     // ANSI-style function header
{                     // says takes one int argument
    while (n-- > 0)
        printf("#");
    printf("\n");
}
```

函数的定义

```
#####
#####################################
######
```

```c
/* pound.c -- defines a function with an argument   */
#include <stdio.h>

void pound(int n) {  // ANSI-style function header
    while (n-- > 0)  // says takes one int argument
        printf("#");
    printf("\n");
}

int main(void) {
    int times = 5;
    char ch = '!';   // ASCII code is 33
    float f = 6.0f;
    pound(times);    // int argument
    pound(ch);       // same as pound((int)ch);
    pound(f);        // same as pound((int)f);
    return 0;
}
```

如果函数体书写在调用之前，则原型可以省略，但不建议。这样容易出错。

```
#####
#####################################
######
```

```c
// running.c -- A useful program for runners
#include <stdio.h>
const int S_PER_M = 60;          // seconds in a minute
const int S_PER_H = 3600;        // seconds in an hour
const double M_PER_K = 0.62137; // miles in a kilometer
int main(void) {
    double distk, distm;  // distance run in km and in miles
    double rate;          // average speed in mph
    int min, sec;         // minutes and seconds of running time
    int time;             // running time in seconds only
    double mtime;         // time in seconds for one mile
    int mmin, msec;       // minutes and seconds for one mile
    printf("This program converts your time for a metric race\n");
    printf("speed in miles per hour.\n");
    printf("Please enter, in kilometers, the distance run.\n");
    scanf("%lf", &distk);  // %lf for type double
    printf("Next enter the time in minutes and seconds.\n");
    printf("Begin by entering the minutes.\n");
```

```
    scanf("%d", &min);
    printf("Now enter the seconds.\n");
    scanf("%d", &sec);
    // converts time to pure seconds
    time = S_PER_M * min + sec;
    // converts kilometers to miles
    distm = M_PER_K * distk;
    // miles per sec x sec per hour = mph
    rate = distm / time * S_PER_H;
    // time/distance = time per mile
    mtime = (double) time / distm;
    mmin = (int) mtime / S_PER_M; // find whole minutes
    msec = (int) mtime % S_PER_M; // find remaining seconds
    printf("You ran %1.2f km (%1.2f miles) in %d min, %d sec.\n",
           distk, distm, min, sec);
    printf("That pace corresponds to running a mile in %d min, ", mmin);
    printf("%d sec.\nYour average speed was %1.2f mph.\n",msec, rate);
    return 0;
}
```

内容纲要

C 程 序 设 计

C Programming

5

谢谢观看

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 煒

(特色化示范性软件学院) 博士,副教授

School of Informatics Wei Huang