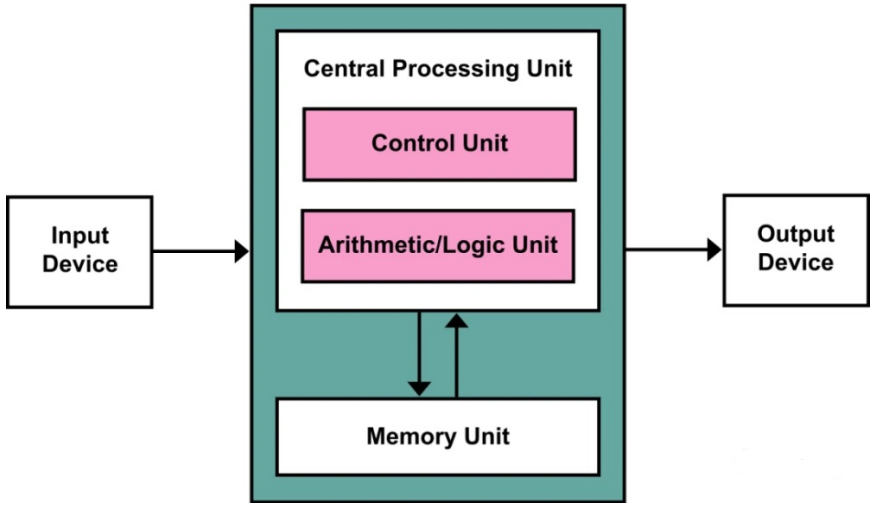
06 C控制语句-循环

内容提要

- 关键字： for while do while
- 运算符： < > >= <= != == += *= -= /= &=
- 函数： fabs()
- C 的三种循环结构：while ， for 和 do while
- 使用关系运算符构建控制循环的表达式
- 其他一些运算符
- 循环中常用的数组
- 编写具有返回值的函数



再探while循环

1 再探while循环

6.1 summing.c

- 对用户键盘输入整数求和
- 伪代码(pseudocode)

```
把 sum 初始化为 0
提示用户输入数据
读取用户输入的数据
当输入的数据为整数时，
    输入添加给 sum，
    提示用户进行输入，
    然后读取下一个输入
输入完成后，打印 sum 的值
```

- long类型以存储更大的整数
- sum初始化为0L(long类型的0)

```
1.  #include <stdio.h>
2.  int main(void){
3.      long num;
4.      long sum = 0L;      // initialize sum to zero
5.      int status;
6.
7.      printf("Please enter an integer to be summed ");
8.      printf("(q to quit): ");
9.      status = scanf("%ld", &num);
10.     while (status == 1) // == means "is equal to"
11.     {
12.         sum = sum + num;
13.         printf("Enter next integer (q to quit): ");
14.         status = scanf("%ld", &num);
15.     }
16.     printf("Those integers sum to %ld.\n", sum);
17.     return 0;
18. }
```

1.1 程序注解

- 循环的判断条件
  - while (status == 1)
  - ==，相等运算符(equality operator)，判断status是否等于1
  - status = 1，把1赋给status
- 伪代码(pseudocode)
  - 一种用简单英语表示程序的方法，与计算机语言的形式对应
  - 伪代码有助于设计程序的逻辑
    - 确认逻辑正确后，再将伪代码翻译成实际编程代码
  - 提示用户输入数据
  - 读取用户输入的数据
  - 当输入的数据为整数时，
    - 输入添加给sum，
    - 提示用户进行输入，
    - 然后读取下一个输入
  - 输入完成后，打印sum的值

```c
#include <stdio.h>
int main(void){
    long num;
    long sum = 0L;      // initialize sum to zero
    int status;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);
    while (status == 1) // == means "is equal to"
    {
        sum = sum + num;
        printf(“Enter next integer (q to quit): ");
        status = scanf("%ld", &num);
    }
    printf("Those integers sum to %ld.\n", sum);
    return 0;
}
```

1.2 C风格的读循环

- 两种scanf()方式

- 第二种形式同时使用scanf()的两种不同的特性
  - 如函数调用成功，scanf()会把一个值存入num
  - scanf()的返回值(0或1，非num的值)控制while循环

```
1.  status = scanf("%ld", &num);
2.  while (status == 1)
3.  {
4.      /* 循环行为 */
5.      status = scanf("%ld", &num);
6.  }

7.  while (scanf("%ld", &num) == 1)
8.  {
9.      /*循环行为*/
10. }
```

while语句

2 while语句

```
while(expression)
        statement
```

- 如果 expression 为真，那么就执行一次 statement 部分，然后再次判断 expression
  - 在expression变为假(0)之前，重复这个判断和执行的循环
- 每次循环称为一次迭代(iteration)
- statement 部分可以是一个带有分号的简单语句，也可以是花括号的一个复合语句

2.1-2.3 终止 while 循环

- 构造 while 循环时
  - 循环中必须包含能改变判断表达式值的语句
    - 使得判断表达式的值最终变为假
  - 否则，循环永远不会终止
  - 特别的，break和 if 语句来终止循环

- while循环：使用入口条件的有条件循环
  - “有条件”指：循环体的执行取决于是否满足入口条件(entry condition)
    - 如(n < 7)
    - 满足条件才能进入循环体

- 程序清单6.2 when.c

```
1.  // when.c -- when a loop quits
2.  #include <stdio.h>
3.  int main(void)
4.  {
5.      int n = 5;
6.
7.      while (n < 7)                       // line 7
8.      {
9.          printf("n = %d\n", n);
10.         n++;                            // line 10
11.         printf("Now n = %d\n", n);      // line 11
12.     }
13.     printf("The loop has finished.\n");
14.
15.     return 0;
16. }
```

2.4 语法要点

- 程序清单6.3　while1.c
- 程序清单6.4　while2.c
- 语句10并不是循环体语句，虽然缩进
  - 导致无限循环（infinite loop）
  - 需要括号
- 语句8，单独的分号表示空语句

```c
/* while2.c -- watch your semicolons */
#include <stdio.h>
int main(void){
    int n = 0;
    while (n++ < 3);          /* line 7 */
    printf("n is %d\n", n);   /* line 8 */
    printf("That's all this program does.\n");
    return 0;
}
```

```c
/* while1.c -- watch your braces       */
/* bad coding creates an infinite loop */
#include <stdio.h>
int main(void)
{
    int n = 0;

    while (n < 3)
        printf("n is %d\n", n);
        n++;
    printf("That's all this program does\n");

    return 0;
}
```

比较大小：使用关系运算符和表达式

3 比较大小：使用关系运算符和表达式

- 6.5 cmpflt.c
  - 浮点数比较
- 关系表达式（relational expression）
  - 比较大小关系的表达式
  - 逻辑表达式的一种
- 关系运算符（relational operator）

Table 6.1 Relational Operators

Operator	Meaning
<	Is less than
<=	Is less than or equal to
==	Is equal to
>=	Is greater than or equal to
>	Is greater than
!=	Is not equal to

```c
// cmpflt.c -- floating-point comparisons
#include <math.h>
#include <stdio.h>
int main(void){
    const double ANSWER = 3.14159;
    double response;
    printf("What is the value of pi?\n");
    scanf("%lf", &response);
    while (fabs(response - ANSWER) > 0.0001)
    {
        printf("Try again!\n");
        scanf("%lf", &response);
    }
    printf("Close enough!\n");

    return 0;
}
```

3.1 什么是真？

- 6.6 t_and_t.c
- 对 C的逻辑表达式
  - 真表达式的值为 1
  - 假表达式的值为 0

```c
/* t_and_f.c -- true and false values in C */
#include <stdio.h>
int main(void)
{
    int true_val, false_val;

    true_val = (10 > 2);    // a true relationship
    false_val = (10 == 2);  // a false relationship
    printf("true = %d; false = %d \n", true_val,
false_val);

    return 0;
}
```

3.2 还有什么是真

- 6.7 truth.c
- 数值转换成真假
  - 所有的非零值都被认为是真
  - 只有 0 被认为是假

```c
// truth.c -- what values are true?
#include <stdio.h>
int main(void)
{
    int n = 3;

    while (n)
        printf("%2d is true\n", n--);
    printf("%2d is false\n", n);

    n = -3;
    while (n)
        printf("%2d is true\n", n++);
    printf("%2d is false\n", n);

    return 0;
}
```

3.3 真值的问题

- 程序清单6.8 trouble.c
- == 和 =
- 13行：不要在应该使用 == 的地方使用 =
- 如果有一个常量，可把它放在表达式的左边
  - 1==status
  - 1=status会报错

```
1.  // will cause infinite loop
2.  #include <stdio.h>
3.  int main(void){
4.      long num;
5.      long sum = 0L;
6.      int status;
7.
8.      printf("Please enter an integer to be summed ");
9.      printf("(q to quit): ");
10.     status = scanf("%ld", &num);
11.     while (status = 1){
12.         sum = sum + num;
13.         printf("Enter next integer (q to quit): ");
14.         status = scanf("%ld", &num);
15.     }
16.     printf("Those integers sum to %ld.\n", sum);
17.
18.     return 0;
19. }
```

3.4 新的 _Bool 类型【选】

- _Bool类型的变量只能储存1(真)或0(假)
  - 如果把其他非零数值赋给_Bool类型的变量，该变量会被设置为1
- 6.9 boolean.c

```c
#include <stdio.h>
int main(void){
    long num;
    long sum = 0L;
    _Bool input_is_good;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    input_is_good = (scanf("%ld", &num) == 1);
    while (input_is_good)    {
        sum = sum + num;
        printf(“Enter next integer (q to quit): ");
        input_is_good = (scanf("%ld", &num) == 1);
    }
    printf("Those integers sum to %ld.\n", sum);

    return 0;
}
```

3.5 关系运算符的优先级

- 关系运算符的优先级比算术运算符（包括+和-）低，比赋值运算符高
- x > y + 2　　也同时等于　x >（y + 2）
- 关系运算符从左到右进行结合

表 6.2　运算符优先级

运算符（优先级从高至低）	结合律
()	从左往右
- + ++ -- sizeof	从右往左
* / %	从左往右
+ -	从左往右
< > <= >=	从左往右
== !=	从左往右
=	从右往左

不确定循环与计数循环

4 不确定循环与计数循环

- 不确定循环(indefinite loop)
  - 【在测试表达式为假之前】不能预先判断循环次数
  - 6.10 sweetie1.c
- 计数循环(counting loop)
  - 循环前知道重复执行次数

- 建立一个重复执行固定次数的循环
  - 初始化一个计数器
  - 循环条件测试：计数器与某个值进行比较
  - 每次循环迭代：计数器更新（递增/减）

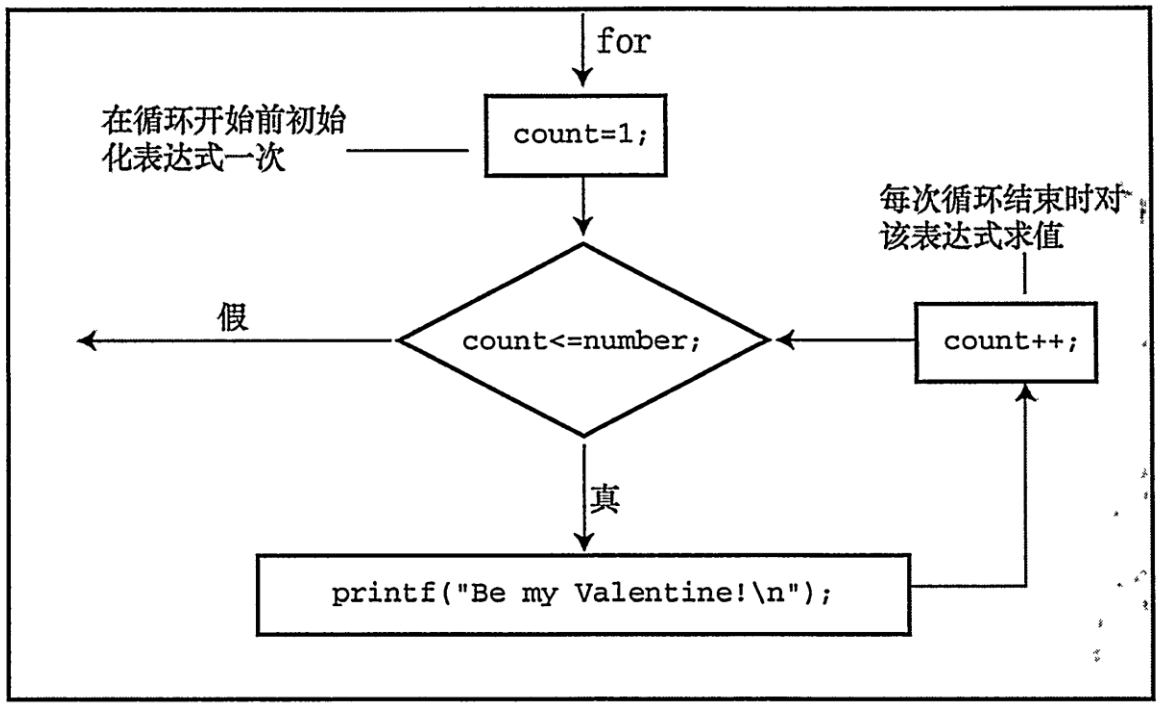
```c
// sweetie1.c -- a counting loop
#include <stdio.h>
int main(void)
{
    const int NUMBER = 22;
    int count = 1; // initialization

    while (count <= NUMBER) // test
    {
        printf("Be my Valentine!\n");  // action
        count++; // update count
    }

    return 0;
}
```

for循环

5 for循环

- for循环：三种动作(初始化，测试，更新)一起

- 6.11 sweetie2.c




```c
// sweetie2.c -- a counting loop using for
#include <stdio.h>
int main(void)
{
    const int NUMBER = 22;
    int count;

    for (count = 1; count <= NUMBER; count++)
        printf("Be my Valentine!\n");

    return 0;
}
```

for循环

```
for ( initialize; test; update)
    statement;
```

- initialize进行初始化
  - for 循环开始的时候执行一次
- test是判断条件
  - 每次循环之前都要对它进行求值。当表达式为假（count 大于 NUMBER)时，循环结束
- update进行改变或称为更新
  - 每次循环结束时进行计算

for的灵活性

- 6.12 for_cube.c
  - 使用for 循环创建一个立方表
- for_down.c
  - 可以使用递减运算符来递减计数器：
- for_13s.c
  - 可以让计数器递增2 、10 等
- for_char.c
  - 可以用字符代替数字计数
- for_geo.c
  - 可以让递增的量几何增长，而不是算术增长
- for_wild.c
  - 第3个表达式可以使用任意合法的表达式
- for_none.c
  - 可以省略一个或多个表达式（但不能省略分号）
- for_show.c
  - 第3 个表达式可以使用任意合法的表达式

更多赋值运算符：+=　-=　*=　/=　%=

6 更多赋值运算符+= -= *= /= %=

- +=
- -=
- *=
- /=
- %=

```
scores += 20      与    scores = scores + 20      相同
dimes -= 2        与    dimes = dimes - 2         相同
bunnies *= 2      与    bunnies = bunnies * 2     相同
time /= 2.73      与    time = time / 2.73        相同
reduce %= 3       与    reduce = reduce % 3       相同
```

逗号运算符

7 逗号运算符

- 程序清单6.13 postage.c
- exp1，exp2
  - 循环中使用多个初始化或更新表达式
  - 两个属性
    - 表达式按从左到右的次序进行计算
    - 整个逗号表达式的值是，右边成员的值
- 常见作用
  - 逻辑上相近的多个表达式组合成一条语句

```
1.  // postage.c -- first-class postage rates
2.  #include <stdio.h>
3.  int main(void)
4.  {
5.      const int FIRST_OZ = 46; // 2013 rate
6.      const int NEXT_OZ = 20;  // 2013 rate
7.      int ounces, cost;
8.
9.      printf(" ounces  cost\n");
10.     for (ounces=1, cost=FIRST_OZ; ounces <= 16;
    ounces++, cost += NEXT_OZ)
11.         printf("%5d $%4.2f\n", ounces, cost/100.0);
12.
13.     return 0;
14. }
```

Zeno

- 希腊哲学家Zeno ： 箭永远不会达到它的目标
  - 箭要到达目标距离的一半，然后再达到剩余距离的一半，然后继续到达剩余距离的一半，这样就无穷无尽
  - Zeno 认为箭的飞行过程有无数个部分，所以要花费无数时间才能结束这一过程
  - 1 + 1/2 + 1/4 + 1/8 + 1/16 +.....
- 6.14 zeno.c
- 该程序演示了在表达式中可以使用多个逗号运算符

```c
/* zeno.c -- series sum */
#include <stdio.h>
int main(void)
{
    int t_ct;       // term count
    double time, power_of_2;
    int limit;

    printf("Enter the number of terms you want: ");
    scanf("%d", &limit);
    for (time=0, power_of_2=1, t_ct=1; t_ct <= limit; t_ct++, power_of_2 *= 2.0) {
        time += 1.0/power_of_2;
        printf("time = %f, terms = %d.\n", time, t_ct);
    }

    return 0;
}
```

退出条件循环：do while

8 退出条件循环：do while

```
do{
 statement;
}while(exit-condition)
```

- do while 循环，一种出口条件循环(exit-condition loop)
  - 执行循环之后，进行条件判断。循环体中的语句至少被执行一次
- while 循环和 for 循环都是入口条件循环，在每次执行循环之前先检查判断条件，这样循环中的语句就有可能一次也不执行。

6.15 do_while.c

6.16 entry.c

选择哪种循环

9 选择哪种循环

- 首先确定入口条件循环还是退出条件循环
- 如果在循环开始的地方进行循环判断，程序的可读性更强；for，while
- 在很多应用中，如果一开始就不满足判断符，while

- L1和L3等价
- L5-10和L11-12等价

- 假定需要一个入口条件循环，应该使用 for 还是 while 循环？
  - 这是个人爱好的问题

```
1.  for ( ; test ; )
2.
3.  while ( test )
4.
5.  初始化;
6.  while ( 测试 )
7.  {
8.      其他语句
9.      更新语句
10. }

11. for ( 初始化; 测试; 更新 )
12.     其他语句
```

嵌套循环

10 嵌套循环

- 嵌套循环(nested loop)
  - 循环体之内，包含了其它循环
- 6.17　rows1.c

- 10行开始的for循环：外层循环（outer loop）
- 12行开始的for循环：内层循环（inner loop）
- 嵌套循环中的内层循环在每次外层循环迭代时都执行完所有的循环

```
1.  /* rows1.c -- uses nested loops */
2.  #include <stdio.h>
3.  #define ROWS  6
4.  #define CHARS 10
5.  int main(void)
6.  {
7.      int row;
8.      char ch;
9.
10.     for (row = 0; row < ROWS; row++)//line 10
11.     {
12.         for (ch = 'A'; ch < ('A' + CHARS); ch++)
13.             printf("%c", ch);
14.         printf("\n");
15.     }
16.
17.     return 0;
18. }
```

10.2 嵌套变化

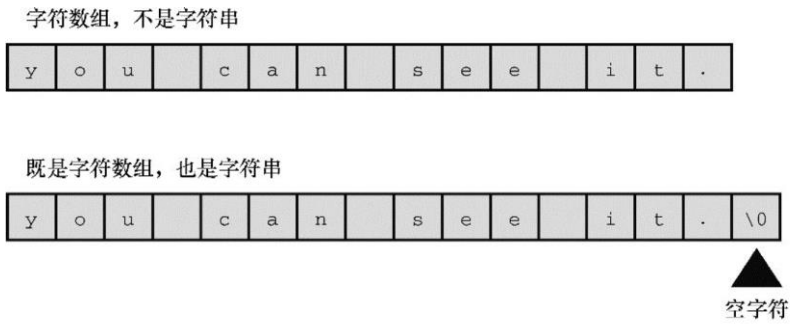
- 6.18 rows2.c
- 内层循环取决于外层循环

```c
// rows2.c -- using dependent nested loops
#include <stdio.h>
int main(void)
{
    const int ROWS = 6;
    const int CHARS = 6;
    int row;
    char ch;

    for (row = 0; row < ROWS; row++)
    {
        for (ch = ('A' + row); ch < ('A' + CHARS); ch++)
            printf("%c", ch);
        printf("\n");
    }

    return 0;
}
```

数组

11 数组

- 数组(array)：按顺序存储的一系列类型相同的值
  - 整个数组有一个数组名
  - 通过整数下标访问数组中的项或元素（element）
  - float debts[20];
  - debts[1] = 0.0f;
- 考虑执行速度，C编译器不检查数组的下标是否正确【越界】
  - debts[20] = 88.32; // 该元素不存在，编译器不管
- 数组



- 用于识别数组元素的数字被称为下标(subscript)、索引(indice)或偏移量(offset)
  - 下标必须是整数，从0开始计数
  - 数组的元素被依次存储在内存中相邻的位置

在for循环中使用数组

- 6.19 scores_in.c
- 在 for 循环中使用数组
- 读取10个高尔夫分数，稍后进行处理。程序打印总分、平均分、差点（handicap，它是平均分与标准分的差值）

```c
#define SIZE 10
#define PAR 72
int main(void){
    int index, score[SIZE];
    int sum = 0;
    float average;
    printf("Enter %d golf scores:\n", SIZE);
    for (index = 0; index < SIZE; index++)
        scanf("%d", &score[index]);  // read scores
    printf("The scores read in are as follows:\n");
    for (index = 0; index < SIZE; index++)
        printf("%5d", score[index]); // verify input
    for (index = 0; index < SIZE; index++)
        sum += score[index];     // add them up
    average = (float) sum / SIZE;
    printf("Sum = %d, avr = %.2f\n", sum, average);
    printf("Handicap of %.0f.\n", average - PAR);
    return 0;
}
```

使用函数返回值的循环例子

12 使用函数返回值的循环例子

- 用一个函数计算数的整数次幂，累乘
- 6.20 power.c

- 编写一个有返回值的函数：
  - 定义函数时，确定函数的返回类型
  - 关键字return表明待返回的值

- main()是一个驱动程序（driver），即被设计用来测试函数的小程序

- power()函数在程序中出现了3次
  - 原型，函数调用，函数定义

```
1.  double power(double n, int p); // ANSI prototype
2.  int main(void){
3.      double x, xpow;
4.      int exp;
5.      printf("Enter a number and + integer power to
    which\nthe number will be raised. Q to quit.\n");
6.      while (scanf("%lf%d", &x, &exp) == 2){
7.          xpow = power(x,exp);   // function call
8.          printf("%.3g to the power %d is %.5g\n", x,
    exp, xpow);
9.          printf(“Next pair or q to quit.\n");
10.     }
11.     return 0;
12. }
13. double power(double n, int p){
14.     double pow = 1;
15.     for (int i = 1; i <= p; i++) pow *= n;
16.     return pow; // return the value of pow
17. }
```

12.2 使用具有返回值的函数

- 声明函数，调用函数，定义函数，使用 return 关键字
  - 定义并使用具有返回值的函数的基本要素

- 必须通过前置声明（forward declaration）预先说明函数的返回类型。前置声明告诉编译器，power()定义在别处，其返回类型为double

- 函数经常放在单独的文件中，所以向前声明是必不可少的
- stdio.h 头文件中含有 scanf(), printf()以及其他一些 I/O 函数的函数声明

13 关键概念

- 在创建循环时，要特别注意以下3个方面
  - 注意循环的测试条件要能使循环结束
  - 确保循环测试中的值在首次使用之前已初始化
  - 确保循环在每次迭代都更新测试的值
- C通过求值来处理测试条件，结果为0表示假，非0表示真
- 数组由相邻的内存位置组成，只存储相同类型的数据
  - 数组元素的编号从0开始，所有数组最后一个元素的下标一定比元素数目少1。C编译器不会检查数组下标值是否有效
- 使用函数涉及3个步骤
  - 通过函数原型声明函数
  - 在程序中通过函数调用使用函数
  - 定义函数

14 总结