第5章

循环结构

编程先驱

何积丰(图 5-0),1943 年 8 月 5 日出生于上海市,计算机软件专家,中国科学院院士,上海华科智谷人工智能研究院院长,华东师范大学软件学院原院长、教授、博士生导师,他在计算机领域有着卓越的贡献。

图 5-0　何积丰

何积丰与图灵奖获得者 Hoare 教授创造性地提出了软件的程序统一理论,解决了程序语义的一致性问题,奠定了软件语义元理论基础,开创了程序统一理论学派,出版了英文专著*Unifying Theories of Programming*,该文献他引超过 800 次。

针对软件开发各阶段模型正确性问题,何积丰创建了数据精化完备理论,首次提出了数据精化的“程序分解算子”与“上下仿真映照对”方法,将规范语言与程序语言看成同一类数学对象,采用“关系代数”作为程序和软件规范的统一数学模型,在此框架中建立了求解规范方程的演算法则。该成果被国际计算机科学界誉为“面向模型软件的开发的一个里程碑”。

何积丰创造性地开拓和发展了基于模型的可信软件的开发与验证研究领域,建立了正确性系统的可证理论与方法,解决了可信嵌入式系统构造与验证技术的若干关键问题,并应用于轨道交通、汽车电子、航天控制等安全攸关行业,推动了相关产业发展。

引言

自动化指的是利用各种机械、设备、技术来代替大脑和双手进行操作,在没有人工参与的情况下,按照人们预想的方式来完成人们不能做或很难完成的工作。随着自动化技术的不断普及和发展,各种机器人、机械臂不仅将人类从繁重的体力劳动、部分脑力劳动以及恶劣、危险的工作环境中解放出来,而且还极大地提高了劳动生产率,增强了人类认识世界和改造世界的能力。

工厂中随处可见日夜不停工作的机械臂,它们往往都不断重复着各自独特的工作,有的专门负责组装,有的专门负责喷涂。如果只使用前几章学习过的知识点,是无法实现这类重复不停工作的机械臂的控制的。本章学习的循环结构将解决在编程时遇到需要大量重复操作的问题。

前置知识

1. 增量和减量运算符(++、--)

增量和减量运算符用于将变量值加 1 或减 1。两种运算符都有前置写法和后置写法。

```
将变量 x 的值增加 1          将变量 x 的值减少 1
++x       x++               --x       x--
```

以增量运算符为例,上述两种写法在将变量的值增加 1 的方面作用相同,但它们作为表达式求出的值不同:前置写法++x 求出的值是 x 加 1 以后的值;后置写法 x++的值是 x 加 1 操作之前的值。减量操作情况也类似。请看下面语句序列在计算中的情况。

```
x = 2;
y = 2 + ++x;     /* x 值变为 3,y 置为 5,因为++x 的值是加 1 之后的值 */
z = 3 + x++;     /* x 值变为 4,z 取得值 6,因为 x++ 的值为 3,是加 1 之前的值 */
```

增量和减量运算符常用于循环变量更新。另请注意上面第二个语句的写法,我们在增量运算符和加运算符之间写了空格,这是非常必要的。这里前后出现了三个加符号,插入空格可以保证编译系统对这个表达式的分析不出现错误。

2. 二元运算符操作的赋值运算符

程序里常常需要“sum=sum+n*n;”形式的赋值语句做变量更新,其中用到一个二元运算符,从变量原有值出发,通过与另一表达式运算得到新值再赋给变量。这种操作在程序中很典型。为了能更方便地描述这类操作,C 语言为许多二元运算符提供了对应的赋值运算符。每个算术运算符都有对应的赋值运算符,分别是:

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

这些运算符的优先级与简单赋值运算符相同,同样采用从右向左的结合方式。它们的计算结果就是变量的最后更新值,类型与变量类型相同。写在这些赋值运算符左边的必须是变量,右边可以是任何表达式。

下面是一些例子,每行中左边的语句在效用上与右边语句相同。

```
x += 3.5;                    x = x + 3.5;
sum += n * n;                sum = sum + n * n;
res *= x;                    res = res * x;
x += y += 3;                 x = x + (y = y + 3);
```

下面是使用赋值运算符的例子。

```
for (sum = 0, i = 1; i <= 100; i++)
  sum += n * n;
```

这些赋值运算符也有与增量、减量运算符类似的问题。因此上面说的效用等价并不准确。这里也有一次计算或两次计算的问题等,也可能有实现效率问题。

本章知识点

5.1　循 环 结 构

在 C 语言中，共有以下三大常用的程序结构。

（1）顺序结构：代码从前往后执行，函数中的第一个语句先执行，接着是第二个语句，以此类推。

（2）选择结构：也叫分支结构，重点要掌握 if…else、switch 以及条件运算符。

（3）循环结构：重复执行同一段代码。

前面介绍了程序中常用到的顺序结构和选择结构，但是只有这两种结构是不够的，还需要用到循环结构（或称重复结构）。因为在日常生活中或是在程序所处理的问题中常常遇到需要重复处理的问题，例如，要计算 1＋2＋3＋…＋99＋100 的值，就要重复进行 99 次加法运算。最原始的方法是先编写求一次加法的程序段，然后再重复写 98 个相同的程序段。这种方法虽然可以实现要求，但是显然是不可取的，因为工作量大，程序冗长、重复，难以阅读和维护。实际上，几乎每一种计算机高级语言都提供了循环控制，用来处理需要进行的重复操作。

在 C 语言中，可以使用以下循环语句来处理上面的问题。

```
#include <stdio.h>
int main(){
    int i=1, sum=0;
    while(i<=100){
        sum+=i;
        i++;
    }
    printf("%d\n",sum);
    return 0;
}
```

可以看到：用一个循环语句（while 语句），就把需要重复执行 99 次程序段的问题解决了。一个 while 语句实现了一个循环结构。我们将在 5.2 节对其执行过程进行学习。

大多数的应用程序都会包含循环结构。循环结构和顺序结构、选择结构是结构化程序设计的三种基本结构，它们是各种复杂程序的基本构成单元。因此熟练掌握选择结构和循环结构的概念及使用是进行程序设计最基本的要求。

5.2　while 语句

while 循环的一般形式为

```
while(表达式){
  语句块
}
```

意思是,先计算“表达式”的值,当值为真(非 0)时,执行“语句块”;执行完“语句块”,再次计算表达式的值,如果为真,继续执行“语句块”……这个过程会一直重复,直到表达式的值为假(0),就退出循环,执行 while 后面的代码。

通常将“表达式”称为循环条件,把“语句块”称为循环体,整个循环的过程就是不停判断循环条件并执行循环体代码的过程。

下面以 5.1 节中的代码为例进行分析。

(1) 程序运行到 while 时,因为 i=1,i<=100 成立,所以会执行循环体;执行结束后 i 的值变为 2,sum 的值变为 1。

(2) 接下来会继续判断 i<=100 是否成立,因为此时 i=2,i<=100 成立,所以继续执行循环体;执行结束后 i 的值变为 3,sum 的值变为 3。

(3) 重复执行步骤(2)。

(4) 当循环进行到第 100 次,i 的值变为 101,sum 的值变为 5050;因为此时 i<=100 不再成立,所以就退出循环,不再执行循环体,转而执行 while 循环后面的代码。

while 循环的整体思路是:设置一个带有变量的循环条件,也即一个带有变量的表达式;在循环体中额外添加一条语句,让它能够改变循环条件中变量的值。这样,随着循环的不断执行,循环条件中变量的值也会不断变化,终有一个时刻,循环条件不再成立,整个循环就结束了。

如果循环条件中不包含变量,会发生什么情况呢?

(1) 循环条件成立时,while 循环会一直执行下去,永不结束,成为“死循环”。例如:

```
#include <stdio.h>
int main(){
    while(1){
        printf("1");
    }
    return 0;
}
```

运行程序,会不停地输出“1”,直到用户强制关闭。

(2) 循环条件不成立的话,while 循环就一次也不会执行。例如:

```
#include <stdio.h>
int main(){
    while(0){
        printf("1");
    }
    return 0;
}
```

运行程序,什么也不会输出。

(3) 再看一个例子,统计从键盘输入的一行字符的个数。

```
#include <stdio.h>
int main(){
    int n=0;
```

```
    printf("Input a string:");
    while(getchar()!='\n') n++;
    printf("Number of characters: %d\n", n);
    return 0;
}
```

运行结果：

```
Input a string:c.biancheng.net↙
Number of characters: 15
```

本例程序中的循环条件为 getchar()!='\n',其意义是,只要从键盘输入的字符不是 Enter 键就继续循环。循环体中的 n++;完成对输入字符个数计数。

5.3　do…while 语句

除了 while 语句以外,C 语言还提供了 do…while 语句来实现循环结构。例如：

```
int i=1;
do
  {
    printf("%d",i++);
  }
  while(i<=100);
```

它的作用是：执行(用 do 表示“做”)printf 语句,然后在 while 后面的括号内的表达式中检查 i 的值,当 i 小于或等于 100 时,就返回再执行一次循环体(printf 语句),直到 i 大于 100 为止。执行此 do…while 语句的结果是输出 1～100,共 100 个数。请注意分析 printf()函数中的输出项 i++的作用：先输出当前 i 的值,然后再使 i 的值加 1。如果改为 printf("%d",++i),则是先使 i 的值加 1,然后输出 i 的新值。若在执行 printf()函数之前,i 的值为 1,则 printf()函数的输出是 i 的新值 2。在本例中 do 下面的一对花括号其实不是必要的,因为花括号内只有一个语句。可以写成

```
do
  printf("%d",i++);
while(i<=100);
```

但这样写,容易使人在看到第 2 行末尾的分号后误认为整个语句结束了。为了使程序清晰、易读,建议把循环体用花括号括起来。

do…while 循环的一般形式为

```
do{
    语句块
}while(表达式);
```

do…while 循环与 while 循环的不同在于：它会先执行“语句块”,然后再判断表达式是

不是真,如果为真则继续循环;如果为假,则终止循环。因此,do…while 循环至少要执行一次“语句块”。

5.2 节中求解 1+2+3+…+99+100 的值问题也可以使用 do…while 语句。

```
#include <stdio.h>
int main(){
    int i=1, sum=0;
    do{
        sum+=i;
        i++;
    }while(i<=100);
    printf("%d\n", sum);
    return 0;
}
```

注意:while(i<=100);最后的分号必须要有。while 循环和 do…while 各有特点,可以适当选择,实际编程中使用 while 循环较多。

5.4 for 语句

5.4.1 用 for 语句实现循环结构

除了 while 循环,C 语言中还有 for 循环,它的使用更加灵活,完全可以取代 while 循环。5.2 节使用 while 循环来计算从 1 加到 100 的值,代码如下。

```
#include <stdio.h>
int main(){
    int i, sum=0;
    i = 1;                  //语句①
    while(i<=100 /*语句②*/){
        sum+=i;
        i++;                //语句③
    }
    printf("%d\n",sum);
    return 0;
}
```

可以看到,语句①②③被放到了不同的地方,代码结构较为松散。为了让程序更加紧凑,可以使用 for 循环来代替,如下。

```
#include <stdio.h>
int main(){
    int i, sum=0;
    for(i=1/*语句①*/; i<=100/*语句②*/; i++/*语句③*/){
        sum+=i;
    }
    printf("%d\n",sum);
    return 0;
}
```

在 for 循环中，语句①②③被集中到了一起，代码结构一目了然。

for 循环的一般形式为

```
for(表达式 1; 表达式 2; 表达式 3){
    语句块
}
```

它的运行过程如下。

(1) 先执行“表达式 1”。

(2) 再执行“表达式 2”，如果它的值为真(非 0)，则执行循环体，否则结束循环。

(3) 执行完循环体后再执行“表达式 3”。

(4) 重复执行步骤(2)和(3)，直到“表达式 2”的值为假，就结束循环。

上面的步骤中，(2)和(3)是一次循环，会重复执行，for 语句的主要作用就是不断执行步骤(2)和(3)。

“表达式 1”仅在第一次循环时执行，以后都不会再执行了，可以认为这是一个初始化语句。“表达式 2”一般是一个关系表达式，决定了是否还要继续下次循环，称为“循环条件”。“表达式 3”在很多情况下是一个带有自增或自减操作的表达式，以使循环条件逐渐变得“不成立”。

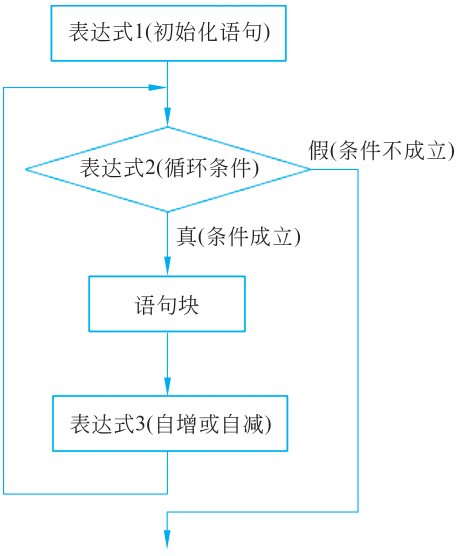

图 5-1　for 循环的执行过程

for 循环的执行过程可用图 5-1 表示。

再来分析一下“计算从 1 加到 100 的和”的代码。

```
#include <stdio.h>
int main(){
    int i, sum=0;
    for(i=1; i<=100; i++){
        sum+=i;
    }
    printf("%d\n",sum);
    return 0;
}
```

运行结果：

```
5050
```

代码分析：

(1) 执行到 for 语句时，先给 i 赋初值 1，判断 i<＝100 是否成立；因为此时 i＝1，i<＝100 成立，所以执行循环体。循环体执行结束后(sum 的值为 1)，再计算 i＋＋。

(2) 第二次循环时，i 的值为 2，i<＝100 成立，继续执行循环体。循环体执行结束后(sum 的值为 3)，再计算 i＋＋。

(3) 重复执行步骤(2),直到第 101 次循环,此时 i 的值为 101,i<=100 不成立,所以结束循环。

由此可以总结出 for 循环的简单形式:

```
for(初始化语句; 循环条件; 循环变量增值){
    语句块
}
```

5.4.2 for 循环中的三个表达式

for 循环中的"表达式 1(初始化条件)"、"表达式 2(循环条件)"和"表达式 3(循环变量增值)"都是可选项,都可以省略(但分号;必须保留)。

(1) 修改"从 1 加到 100 的和"的代码,省略"表达式 1(初始化条件)"。

```
int i = 1, sum = 0;
for(; i<=100; i++){
    sum+=i;
}
```

可以看到,将 i=1 移到了 for 循环的外面。

(2) 省略了"表达式 2(循环条件)",如果不做其他处理就会成为死循环。例如:

```
for(i=1;; i++) sum=sum+i;
```

相当于:

```
i=1;
while(1){
    sum=sum+i;
    i++;
}
```

(3) 省略了"表达式 3(循环变量增值)",就不会修改"表达式 2(循环条件)"中的变量,这时可在循环体中加入修改变量的语句。例如:

```
for(i=1; i<=100;){
    sum=sum+i;
    i++;
}
```

(4) 省略了"表达式 1(初始化语句)"和"表达式 3(循环变量增值)"。例如:

```
for(; i<=100;){
    sum=sum+i;
    i++;
}
```

相当于：

```
while(i<=100){
    sum=sum+i;
    i++;
}
```

(5) 三个表达式可以同时省略。例如：

```
for(; ;)  语句
```

相当于：

```
while(1)  语句
```

(6)“表达式 1”可以是初始化语句，也可以是其他语句。例如：

```
for(sum=0; i<=100; i++ )  sum=sum+i;
```

(7)“表达式 1”和“表达式 3”可以是一个简单表达式，也可以是逗号表达式。例如：

```
for(sum=0,i=1; i<=100; i++ )  sum=sum+i;
```

或：

```
for(i=0,j=100; i<=100; i++,j-- )  k=i+j;
```

(8)“表达式 2”一般是关系表达式或逻辑表达式，但也可以是数值或字符，只要其值非零，就执行循环体。例如：

```
for(i=0; (c=getchar())!='\n'; i+=c);
```

又如：

```
for(; (c=getchar())!='\n';)
    printf("%c",c);
```

5.4.3　几种循环的比较

(1) 三种循环都可以用来处理同一问题，一般情况下它们可以互相代替。

(2) 在 while 循环和 do… while 循环中，只在 while 后面的括号内指定循环条件，因此为了使循环能正常结束，应在循环体中包含使循环趋于结束的语句(如 i++或 i=i+1 等)。for 循环可以在表达式 3 中包含使循环趋于结束的操作，甚至可以将循环体中的操作全部放到表达式 3 中。因此 for 语句的功能更强，凡用 while 循环能完成的，用 for 循环都能实现。

(3) 用 while 和 do…while 循环时，循环变量初始化的操作应在 while 和 do…while 语句之前完成。而 for 语句可以在表达式 1 中实现循环变量的初始化。

5.5 改变循环执行的状态

以上介绍的都是根据事先指定的循环条件正常执行和终止的循环。但有时在某些情况下需要提前结束正在执行的循环操作。此时可以使用 break 语句或 continue 语句。

5.5.1 break 语句

在 4.3 节中讲到了 break,用它来跳出 switch 语句。当 break 关键字用于 while、for 循环时,会终止循环而执行整个循环语句后面的代码。break 关键字通常和 if 语句一起使用,即满足条件时便跳出循环。

使用 while 循环计算从 1 加到 100 的值:

```
#include <stdio.h>
int main(){
    int i=1, sum=0;
    while(1){          //此循环条件导致死循环
        sum+=i;
        i++;
        if(i>100) break;
    }
    printf("%d\n", sum);
    return 0;
}
```

while 循环条件为 1,是一个死循环。当执行到第 100 次循环的时候,计算完 i++;后 i 的值为 101,此时 if 语句的条件 i>100 成立,执行 break;语句,结束循环。

在多层循环中,一条 break 语句只向外跳一层。例如,输出一个 4×4 的整数矩阵:

```
#include <stdio.h>
int main(){
    int i=1, j;
    while(1){                          //外层循环
        j=1;
        while(1){                      //内层循环
            printf("%-4d", i*j);
            j++;
            if(j>4) break;             //跳出内层循环
        }
        printf("\n");
        i++;
        if(i>4) break;                 //跳出外层循环
    }
    return 0;
}
```

运行结果: