第 3 章 语句控制结构

2018 年 8 月 26 日

目录

3.1 语句

学习目标

- 掌握基本语句控制结构的语法和特点；
- 学会运用基本控制结构解决简单问题；
- 理解并能够运用递推法和穷举法解决实际应用问题。

3.1 语句

语句

表达式后面加上分号就变成了一个表达式语句（expression statement）。
如：

```
counter + 1;    //一条没有实际意义的表达式语句
counter += 1;  //一条有用的复合赋值语句
```

空语句

- 只有一个分号构成的语句，如：

```
;    //空语句
```

- 空语句不会执行任何操作，如：

```
counter += 1;;    //第二个分号不会影响该语句的执行
```

是否可以随意使用分号？

3.1 语句

右边的程序段有问题吗？

```
counter = 0;
while(counter < 10 );
++counter;
```

3.1 语句

复合语句

- 复合语句（compound statement）指用花括号括起来的语句和声明序列，也被称作语句块。
- 在块内引入的名字只在块内可见，如：

```
{ //语句块开始
    int sum = 0; // 定义一个对象
    /*...*/
} //语句块结束
```

控制结构语句作用域

- 下面 while 语句的作用域是什么？

```
while(i > 10 )
    a = i;
    b += i;
```

- 可用花括号扩展其作用域，如：

```
while(counter < 10 ){   //while 作用域从这里开始
    ++counter;
    sum += counter;
}  //while 作用域到这里结束
```

3.2 分支结构

分支结构通过条件控制语句实现。C++ 提供了两种分支形式：`if` 语句和 `switch` 语句

3.2 分支结构

——if 语句

if 分支结构格式：

if 语句的语法格式：

```
if (expr) {  //条件表达式

    statement; //语句
}
```

else 分支结构格式：

```
if (expr) {
statement1;  //分支语句1
}
else {
statement2;  //分支语句2
}
```

圆括号里面的表达式也可以是一个初始化了的对象的定义，如：

```
if (int i = 10){
    /*...*/
}
```

建议：尽量使用花括号改善程序的可读性

花括号一般可以省去，但如果 `if` 或 `else` 的作用域里有一条以上的语句，那么花括号是必须要的，建议使用花括号显式地将语句的作用域标出，改善程度的可读性以及避免一些难以察觉的错误。
这个建议同样适用于 `switch`、 `while` 和 `for` 语句。

3.2 分支结构

——if 语句

例 3.1：

判断一个整数是否大于 0 且是 3 的倍数。

3.2 分支结构

——if 语句

例 3.1：

```
#include<iostream>
using namespace std;
int main() {
    int n;
    cout << "请输入一个整数n:";
    cin >> n;
    if (n > 0 && n % 3 == 0) { //n大于0且被3整除
        cout << "Yes" << endl;
    }
    else {
        cout << "No" << endl;
    }
    return 0;
}
```

嵌套的 if 语句

- 有两个以上分支时，选用嵌套的 if 语句结构
- 内嵌 if 语句既可以嵌套在 if 语句中，也可以嵌套在 else 语句中

例 3.2：
将百分制的成绩转换成五级制，如果成绩在 90 分到 100 分范围内（包括 90 分和 100 分），则转换成 A ，80 分到 90 分为 B （包括 80 分不包括 90 分），依次类推，60 分以下为 F。

3.2 分支结构

——if 语句

例 3.2：

```
#include<iostream>
using namespace std;
int main() {
    unsigned score;
    cout << "请输入一个分数:";
    cin >> score;
    if (score < 60) {
        cout << "F" << endl;
    }
    else if (score < 70) {
        cout << "D" << endl;
    }
    else if (score < 80) {
        cout << "C" << endl;
    }
    else if (score < 90) {
        cout << "B" << endl;
    }
    else {
        cout << "A" << endl;
    }
    return 0;
}
```

避免悬垂 else

- 上例中 if 和 else 语句个数相同，若 if 语句数目多于 else 语句数目[a]，就会出现 else 和 if 匹配的问题，也称悬垂 else（dangling else）。
- C++ 规定 else 和离它最近的尚未匹配的 if 匹配，如：

```
if(n % 2 == 0)     //n 被 2 整除
    if(n %3 ==0)     //n 被 3 整除
        cout << "n 是 6 的倍数";
    else     //n 被 2 整除但不能被 3 整除
        cout << "n 是 2 的倍数不是 3 的倍数";
```

[a] else 语句数目不能够多于 if 语句数目，这是因为 else 语句是可以省略的，而 if 是不能省略的

避免悬垂 else

- 根据原则，上例中 else 会和第二个 if 匹配，若我们的本意是 else 和第一个 if 匹配，则相应代码如下：

```
if(n % 2 == 0){
    if(n %3 ==0)
        cout << "n 是 6 的倍数";
}else     //n 不能被 2 整除
        cout << "n 不是 2 的倍数";
```

建议:

显式地标明每个 if 和 else 的作用域，再利用代码编辑器（IDE）提供的缩进功能来进一步改善代码的可读性

例 3.3：

求一元二次方程 $ax^2 + bx + c = 0$ 的根。

3.2 分支结构

——if 语句

例 3.3：

```
#include<iostream>
#include<cmath>    //用于求平方根函数sqrt，第12行代码
using namespace std;[]
int main() {
    double a,b,c;   //创建3个double类型对象存放三个系数值
    cout << "请输入a,b,c:";
    cin >> a >> b >> c;
    if (a != 0) {;
        double delta = b*b - 4 * a*c;
        if (delta > 0) {
            double x1, x2;      //需要时创建对象
            delta = sqrt(delta);   //求delta的平方根
            x1 = (-b + delta) / (2 * a);
            x2 = (-b - delta) / (2 * a);
            cout << "方程有两个实根：" << x1 << ", " << x2 << endl;
        }
        else if (delta < 0) {
            cout << "方程无实根" << endl;
        }
        else {
            cout << "方程有两个相同的实根：" << -b / (2 * a) << endl;
        }
    }
    else {  //二次项系数不能为0
        cout << "a不能为0" << endl;
    }
    return 0;
}
```

3.2 分支结构

——switch 语句

switch 分支结构格式：

```
/*...*/
switch(score/10){
case 9:
    cout << "A" << endl;
    break;
case 8:
    cout << "B" << endl;
    break;
case 7:
    cout << "C" << endl;
    break;
...
default;
cout << "F" << endl;
}
/*...*/
```

思考：

如果缺少 break 会出现什么情况？

使用 switch 语句解决例 3.2

```
#include<iostream>
using namespace std;
int main() {
    int score;
    cout << "请输入一个分数:";
    cin >> score;
    switch (score/10){    //整型值表达式
    case 9:case 10:
        cout << "A" << endl;
        break;
    case 8:  //常量标签值后面紧跟冒号
        cout << "B" << endl;
        break;
    case 7:
        cout << "C" << endl;
    break;
    case 6:
        cout << "D" << endl;
        break;
    default:
        cout << "F" << endl;
        break;
    }
    return 0;
}
```

3.3 循环结构

重复执行某些语句称为循环，C++ 提供了三种循环语句来实现循环结构：while 语句、do while 语句和for 语句。下面将分别介绍它们的用法。

while 语句语法格式

```
while (expr){      //条件表达式
statement;         //循环体语句
}
```

while 语句执行流程

首先判断圆括号里面的条件表达式 expr 的值，只要为真就执行循环体语句，直到其值变成假为止。

例 3.4：

根据以下公式利用迭代法求 π 的近似值，最后一项小于或等于 $1.0E-10$ 时停止。

$$\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{1}{3} \times \frac{2}{5} + \frac{1}{3} \times \frac{2}{5} \times \frac{3}{7} + \ldots + x_i, \quad x_i = x_{i-1} \times \frac{i-1}{2i-1}$$

3.3 循环结构

——while 语句

例 3.4：

```
#include<iostream>
#include <iomanip>     //库函数setprecision
using namespace std;
int main() {
    double sum=0,x=1; //sum存放数列前i项的和，x存放当前项的值，注意初始值
    int i = 1;        //求解第i项
    while (x > 1.0E-10) {
        sum += x;
        ++i;          //在当前项基础上计算下一项
        x *= (i - 1.) / (2 * i - 1);//注意1后面的小数点
    }                 //fixed和setprecision用于控制输出精度
    cout << "pi=" << fixed << setprecision(10) << 2 * sum << endl;
                      //输出结果pi=3.1415926533
    return 0;
}
```

3.3 循环结构

——do while 语句

do while 语句语法格式

```
do{
statement;      //循环体语句
}while (expr); //条件表达式，注意以分号结束
```

do while 语句执行流程

首先执行循环体再判断圆括号里面的条件表达式 expr 的值。

3.2 循环结构

——do while 语句

例 3.5：

输入一段文本，统计数字字符个数。

提示：计数器 cnt，判断输入字符 x 是否是数字字符（'0'≤ x ≤'9'）

3.3 循环结构

——do while 语句

例 3.5：

```
#include<iostream>
using namespace std;
int main() {
    int cnt = 0;
    char x;
    do {
        x = cin.get();  //获取终端输入的任意一个字符
        if (x >= '0'&&x <= '9') ++cnt;
    } while (x != EOF); //EOF为输入流结束标志，组合键Ctrl+z
    cout << "数字字符个数为：  " << cnt << endl;
    return 0;
}
```

3.3 循环结构

——for 语句

for 语句语法格式

```
for(expr1; expr2; expr3){
statement;
}
```

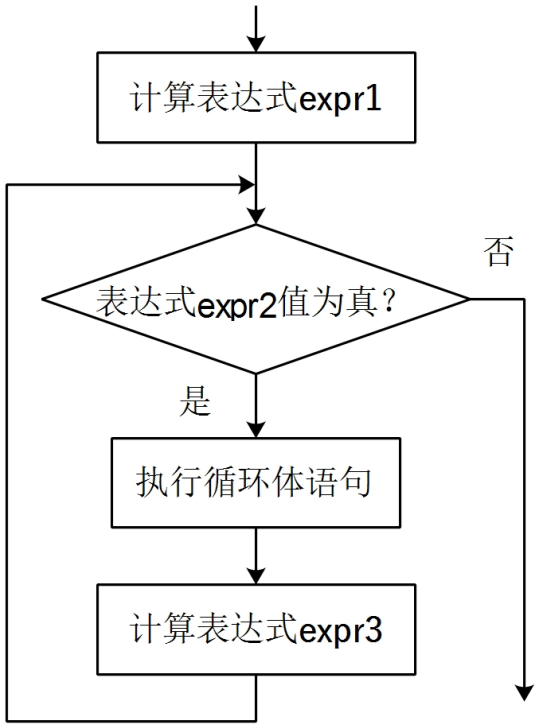

图: for 语句执行流程

3.3 循环结构

——for 语句

思考：

for 语句中表达式不足三个会出现什么情况？

3.3 循环结构

——for 语句

for 非常灵活，可以有多种形式

- 可以省略任意一个表达式，但分号不能省略
- 表达式 expr1 可以定义多个对象，表达式 expr3 可以是任意表达式

注意：

虽然上述表达式可以省略，但是需要在合适的位置添加相应功能的语句。比如，省略表达式 expr1，需要在 for 之前实现对象 i 的初始化；省略表达式 expr3，需要在循环体内部有修改对象 i 的语句；同样省略表达式 expr2，意味着循环条件永远为真，因此需要在循环体里面使用 break 语句结束 for 循环的执行。

3.3 循环结构

循环语句的选择原则

如果循环次数是确定的，一般选择 `for` 语句，否则选择 `while` 或 `do while` 语句。至于 `while` 和 `do while` 语句，它们之间的差别是细微的，如果循环体至少执行一次，请选择 `do while` 语句，如果可能一次也不执行，请选择 `while` 语句。

3.2 循环结构

例 3.6：

猜数字游戏。程序随机选择一个 0-100 之间的一个数，玩家来猜测程序选择的数，如果猜对了，游戏结束，否则玩家继续猜测，直到猜中为止。对于玩家的每一次猜测，需要给出相应的提示信息：猜对了、猜大了或猜小了。

提示：分析题目要求，选择合适的循环语句

3.3 循环结构

例 3.6：

```
#include <cstdlib>                    //使用函数srand和rand
#include <ctime>                      //使用函数time
#include <iostream>
using namespace std;
int main(){
    srand(time(0));                   //系统当前时间作为随机数发生器的种子
    int target = rand() % 100;//获取一个0-100内的随机数
    int guess;
    cout << "'请猜0—100之内的数'" << endl;
    do {
        cin >> guess;
        if (guess < target) {
            cout << "猜小了" << endl;
        }
        else if(guess > target) {
            cout << "猜大了" << endl;
        }
        else {
            cout << "恭喜！ 猜对了！ " << endl;
        }
    } while (guess != target); //猜中时游戏结束}
    return 0;
}
```

3.4 跳转语句

跳转语句用于中断当前的执行顺序，前面已经接触了`break`和`return`语句，C++ 还提供了`continue`和`goto`语句，其中 `return` 语句用来返回到函数的调用处。

3.4 跳转语句

——break 语句

break 语句

break 语句只能用于 switch 语句或循环语句中，用来跳出离它最近的 switch 语句或终止循环的执行，它的作用域仅限离它最近的 switch 语句或循环语句。

示例：

将例 3.6 改造成由 while 内嵌一个 switch 结构来说明 break 语句的用法

3.4 跳转语句

——break 语句

例 3.6：

```
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;
int main(){
    //系统当前时间作为随机数发生器的种子
    srand(time(0));
    //获取一个0-100内的随机数
    int target = rand() % 100;
    int guess;
    cout << "'请猜0—100之内的数'" << endl;
    while(1) {
        cin >> guess;
        int val = (guess > target) - (guess < target);
        //将guess和target的大小关系转化为三个数
        switch (val){
        case -1:
            cout << "猜小了" << endl;
            break;  //跳出switch
        case 1:
            cout << "猜大了" << endl;
            break;  //跳出switch
        default:
            cout << "恭喜！  猜对了！" << endl;
            //跳出switch
            break;
        }//switch结束
        if (val == 0)
        //跳出while，游戏结束
            break;
    }
    }while结束
    return 0;
    }
```

3.4 跳转语句

——continue 语句

continue 语句

continue 语句只在循环结构中有作用，用来终止当前操作，进入下一次循环，下一次循环是否被执行取决于循环条件是否成立。与 break 语句类似，continue 语句的作用域也仅作用于离它最近的循环

例如：

```
//当 i 为奇数，输出 *#，为偶数时无输出
for (int i = 0; i < 5; ++i) {      if (i % 2) {
        cout << "*";  //打印符号 *，然后再打印符号 #
    }
    else {
        continue;    //结束当前迭代，跳转到 for 语句，执行 ++i
    }
    cout << "#";
}
```

3.5 嵌套结构和应用实例

对于大多数问题，单一的语句结构很难解决，往往需要语句结构的嵌套，比如例 3.6 中的循环结构和分支结构的嵌套。如何去设计合理的语句控制结构，需要根据具体问题来分析。

例 3.7：

将坐标系顺时针旋转 90 度，画出 $\sin(x)$ 在 $x \in [0, 2\pi]$ 之间的曲线，如下图所示。

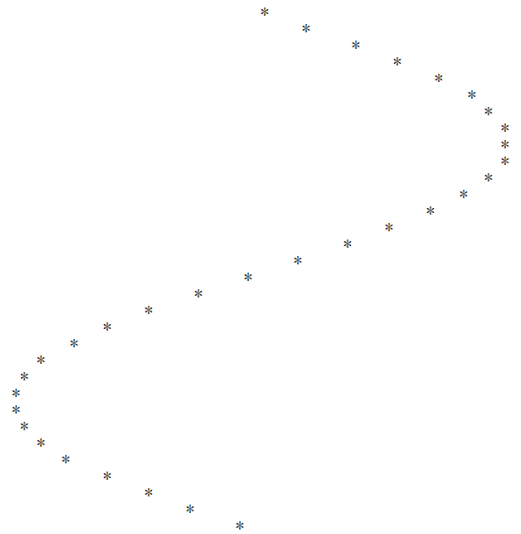

图: sin(x) 曲线

3.5 嵌套结构和应用实例

例 3.7：

```
#include<iostream>
#include<cmath>
using namespace std;

int main() {
    double step = 0.2;                    //x增加的步长
    double x = 0;                         //x从0开始
    while (x < 6.28) {                    //画一个周期的曲线
        int val = 30*(sin(x)+1);          //计算sin(x)左侧的空格数
        for (int i = 0; i < val; ++i) //画出所有空格}
            cout << " ";
        }
        cout << "*" << endl;              //在相应的位置打印*
        x += step;                        //处理下一个x
    }
    return 0;
}
```

例 3.8：石头剪刀布游戏

玩家和电脑出法相减结果如右图所示。

提示：利用双层嵌套循环来实现此游戏，内层循环处理某一局游戏，当已分胜负或者玩家出错时，该局游戏结束。当内层循环结束时，外层循环询问用户是否还要继续玩。

玩家 \ 电脑	石头/0	剪刀/1	布/2
石头/0	0	-1	-2
剪刀/1	1	0	-1
布/2	2	1	0

图: 玩家和电脑出法编号相减结果分布

3.5 嵌套结构和应用实例

例 3.8：

```
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;
int main() {
    //系统当前时间作为随机数发生器的种子
    srand(time(0));
    while (1) {
        int computer(0),you(0);
        do {
            cout << "你好！ 石头=0, 剪刀=1, 布=2";
            //电脑随机选一种出法
            computer = rand() % 3;
            cout << "请出手：";
            //输入0、1或2，不要搞错
            cin >> you;
            switch (you - computer) {
            case 0:
                cout << "平手！" << endl;
                break;
            case 1: case -2:
                cout << "你输了！" << endl;
                break;
            case -1: case 2:
                cout << "你赢了！" << endl;
                break;
            default:
                cout << "你出错了！" << endl;
            }
            //电脑和你出的一样，双方继续出
        } while (computer == you);
        cout << "还要玩吗？Y/N:";
        char play;
        cin >> play;
        if (play == 'N' || play == 'n') break;
    }
    return 0;
}
```

3.5 嵌套结构和应用实例

例 3.9：

公元前五世纪，我国古代数学家张丘建在《算经》一书中提出了（百钱百鸡）：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

提示：穷举法

3.5 嵌套结构和应用实例

例 3.9：

```
#include <iostream>
using namespace std;
int main() {
    int max_rst = 100 / 5, max_hen = 100 / 3;  //公鸡、母鸡最大数目
    for (int i = 0; i < max_rst; ++i) {
        for (int j = 0; j < max_hen; ++j) {
            int k = 100 - i - j;     //小鸡数目
            if (k % 3) continue;     //跳过不能被3整除的数，执行流程跳转到++j
            if (5 * i + 3 * j + k / 3 == 100) {
                cout<<"公鸡："<<i<<" 母鸡："<<j<<" 小鸡："<<k<<endl;
            }
        }
    }
    return 0;
}
```