第 3 章语句控制结构

某某某

中国地质大学（武汉）

2018 年 6 月 20 日

目录

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

```
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
}
```

3.2 分支结构

——if 语句

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){
    /*...*/
}
```

上面的 if 语句中，只要 i 的值不是 0，就执行其作用域里面的语句

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。

示例：

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

示例：

```
#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，如：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

```
#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

嵌套使用示例：

求一元二次方程 $ax^2 + bx + c = 0$ 的根。