一起++ 第6弹

循环（上）、作用域

作者：@孤言　　原审校：@Alex Cui

循环结构，执行一遍又一遍~

知识点1　变量的作用域

1.作用域

在开始学习循环结构之前呢，我们先来了解一下变量的**作用域**。

作用域是变量、函数等可访问的一个区域。也就是说，对于某些变量（或函数等），在某些地方可以使用，在另一些地方就不能用了。

不同的变量声明方式可能导致变量的作用域不同。之前的几弹中，孤言一直让大家把变量的声明写在开头，正是因为写在其它地方，这个变量的作用域就不同了。

2.全局变量

在所有函数外部声明的变量，称为**全局变量**。之前我们紧跟着头文件、`using namespace std;`等的后面，在 main 函数外写的那些变量声明，其实都是全局变量声明。

scratch 创建变量的时候就可以选择全局变量。聪明的你一定知道，Scratch 里全局变量一旦创建，在整个程序中都是可用的，没有角色的限制。C++里也一样，可以说，全局变量的作用域是全局的。全局变量的值可以被该程序的所有对象或函数等使用。

3.局部变量

在函数或一个代码块内部声明的变量，称为**局部变量**。它们只能被函数内部或者代码块内部的语句使用。

通俗但不是很严谨的说法就是，C++中，你在某一个大括号里声明了一个变量，那么它只能在这个大括号括起来的部分起作用。在这个大括号外的地方不能使用它。如图 1,2 所示，在 if 外使用局部变量 b 时会报错。

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

int main()
{
    if (1 == 1)
    {
        int b; //声明局部变量b
    }
    cout << b;

    system("pause");
    return 0;
}
```

```
输出  终端  调试控制台  问题 1        3: 任务 - g++
error: 'b' was not declared in this scope
cout << b;
```

图 1　局部变量错误使用（在 VScode 中）

```
#include <iostream>
using namespace std;
int a=0; // 全局变量声明
int main ()
{
    while(a<1)
    {
        a++;
        int b=1;  //局部变量声明并初始化
    }
cout<<b;
return 0;
}
```

11　7　[Error] 'b' was not declared in this scope

图 2　局部变量错误使用（在 Dev C++中）

一般地，局部变量（或数组等）在声明时的初始值为原来内存中的值，（如 4248811）你无法事先确定。而全局变量（或数组等）在声明时会初始化，比如，int 型变量初始值为 0.

4.关于“同名”

局部变量和全局变量的作用域不同，那么可否声明 2 个名字相同的变量呢？完全可以哒！而且声明的这两个变量是完全不相干的两个变量，占据不同的内存空间。形象地说，这两个变量就是两个仅是名字相同的人，存放的值是不互通的。“同名”的变量在各自的作用域发挥自己的作用。

但这里就有矛盾了，如果某个域同时属于两个同名变量的作用域呢？这时候就要比比谁的“权利”大了。实际上，局部变量的作用域更大一些，我们来看下面这个例子。

示例 6-1　变量的作用域

```cpp
#include<iostream>

using namespace std;

int a=1;//声明全局变量 a

int main()
{
    cout<<a<<endl;
    if(true)
    {
        int a=2;//声明局部变量 a，这两个变量 a 不同
        cout<<a<<endl;
    }
    cout<<a<<endl;

    return 0;
}
```

输出：

```
1
2
1
```

可见，if 语句中的局部变量 a“覆盖”了全局变量的 a.

知识点 2　while 语句

1.从重复执行直到谈起

在 Scratch 里，我们经常会用到重复执行、重复执行直到这些重复执行某段指令的积木。类似于这样的程序设计结构称为循环结构，这一点我们在上一弹已经谈到了。现在，我们重点来康康图 3 的这个重复执行直到积木。积木告诉计算机，**满足条件 a>b 时就停下来。**

图 3　Scratch 积木

而 C++中的 while 语句不是这样的。while 是“在……时”的意思，也就是说，当满足条件时不要停下来，也就是**不满足条件时才停下来。**

所以说，while 语句的条件和重复执行直到的条件表达式是完全相反的。为了帮助大家顺利完成过度，孤言总结以下规则：

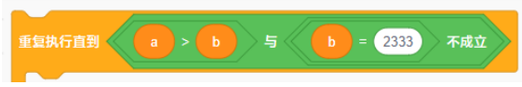
等于改成**不等于**，**和**改成**或**，**大于**改成**小于**，**小于**改成**大于。**

当然，这里的规则并不很全面，但相信你已经 get 到了精髓。

这里再举几个例子。

1. 

2. 

3. 


```cpp
while(a!=50)  //1
while(a<b)  //2
while((a<=b)||(b==2333))  //3
```

1.Scratch 中是 a==50，取其“反面”为 a!=50

2.Scratch 中是 a>b 或 a==b 也就是 a>=b，那么“反面”为 a<b

3.Scratch 中是(a>b)&&(b!=2333)，那么根据规则，“反面”为(a<=b)||(b=2333)

当然，以上的转换只是一个为过渡阶段的同学提供一条道路，即使看不懂以上的转换，也没有关系，只要明确一点就可以了——while 语句只要当条件成立时，才继续循环。

2.while 语句

while 语句的语法格式为：

```
while(条件表达式)
{
    //循环体（是语句块）
}
```

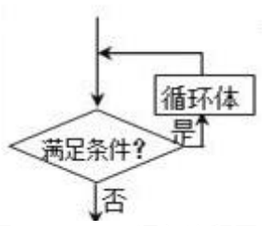


图 4　while 语句流程图

它是怎么运作的呢？可以用流程图 4 来表示。

while 语句先判断条件表达式真假，如果满足条件，则执行循环体（花括号里的语句块）一遍，然后继续判断条件，直到条件不满足时，跳出循环，执行后续语句。

注意　while 语句先判断条件，再执行循环体，循环次数可能为 0 次。

例如，对于下面的代码段：

```
int a=5;
while(a<=4)
{
    a-=1;
}
```

因为一开始就没有满足 a<=4 这个条件，所以循环体根本没有执行，a 的值还是 5.

算法表述的常见方法

1.自然语言：用“讲话”的方式直接描述算法，成为**自然语言**。

2.流程图：以特定的图形符号加上说明，表示算法的图，称为**流程图**。

3. 伪代码：**伪代码**是一种类似于英语结构的，用于描述算法结构的方式（如图 5）。

```
将移动n个盘从A，通过B，移到C
{
  如果n等于1
    直接将盘子从A移到C；
  否则
  {
      将上面的n-1个盘从A，通过C，移到B；
      将最底下的一个盘从A移到C；
      将B上的n-1个盘从B，通过A，移到C；
  }
}
```

图 5　伪代码

下面，我们隆重请出 6 佬来表演跳舞！

//被本教程坑害的大佬还少吗 

-6 总共要跳满 4 个小时的舞蹈，但中间需要休息。他第一次跳 2 个小时，第二次为第一次的 3/4，也就是 90 分钟，以此类推，那么至少要跳几次呢？以下程序展示了求解方式。（也可以使用等比数列求和公式来求和，这里不作引申）。

示例 6-2　6 佬的舞会

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
    double t=120,s=0;  //时间增量（分钟），总用时（分钟）
    int n=0;  //次数

    while(s<5*60) //在 s 没有达到 5h 时
    {
        s+=t;
        t*=2.0/3;  //注意这里的 3.0，用于隐式类型转换
        n+=1;
    }

    cout<<n<<endl;

    system("pause");
    return 0;
}
```

与 for 语句类似，当循环体中仅有一条语句时，大括号可以省略不写。

知识点 3　do-while 语句

1.来做个猜数游戏！

while 语句是先判断、再执行，但有的时候，我们还会遇到先执行一遍、再判断的情况。例如，你想编写一个让别人猜随机数的程序。

随机数函数

C++ 中有一个名为 rand() 的函数，每次调用该函数时都将返回一个非负整数（比如 1804289383）。要使用 rand() 函数，必须在程序中包含 cstdlib 头文件：

```
#include <cstdlib>
```

但要注意，这个函数产生的随机数是伪随机数，也就是用一些确定的计算公示计算出来自的随机数，rand 函数通常在启动计算机以后，生成的随机数就固定不变了。当然也有办法改变这种情况，这里就不深入了。

这样一个随机数显然不能满足你的欲望。我们想要的应该是像图 6 这样的特定范围内的随机数。

图 6

方法也很容易想到，我们以较简单的生成 1~10 的随机整数为例。因为 rand()对 10 取模的结果是在 0~9 间的整数，因此对其+1 也就得到了 1~10 的随机整数。即：

```
rand()%10+1
```

下面我们就来编写这个猜数游戏。

示例 6-3　猜数游戏

```
1   #include <iostream>
2   #include <cstdlib>
3
4   using namespace std;
5
6   int main()
7   {
8       int n = rand() % 10 + 1;
9       int ans;
10
11      do
12      {
13          cout << "Guess the number: ";
14          cin >> ans;
15
16          if (ans > n)
17              cout << "Too large!" << endl;
18          else if (ans < n)
19              cout << "Too small!" << endl;
20      } while (ans != n);
21
22      cout << "U GOT IT!!" << endl;
23
24      system("pause");
25      return 0;
26  }
```

示例 I/O：

```
Guess the number: 3
Too large!
Guess the number: 1
Too small!
Guess the number: 2
U GOT IT!!
```

2.do-while 语句

do-while 语句基本格式为：

```
do
{
    //循环体
} while (条件表达式);
```

注意　while 后面有分号。

它与 while 的不同之处是，会先执行一次循环体，然后再判断条件，如果条件成立则继续循环，否则退出循环。因此循环体至少执行一次。do-while 语句的流程图如图 7 所示。因此即使 Ln 9 改为：

```
int ans=n;
```

也并不影响程序运行，因为第一次的回答已经将会赋给 ans.

图 7　do-while 语句流程图

再如，对于图 8 的两个代码段，代码段 1 输出为 0，代码段 2 输出为 1. 原因很容易想到。

```cpp
//代码段1
#include <iostream>

using namespace std;

int main()
{
    int i = 0;
    while ((i >= 5) && (i <= 10))
        i++;

    cout << i;

    return 0;
}
```

```cpp
//代码段2
#include <iostream>

using namespace std;

int main()
{
    int i = 0;
    do
    {
        i++;
    } while ((i >= 5) && (i <= 10));

    cout << i;

    return 0;
}
```

图 8　while 和 do-while 的比较

根据运算符优先级，&&两侧的子条件表达式的括号**原则上**可以不写，但加上以后可以提高代码可读性。

和分支语句一样，循环语句也可以嵌套使用。

此外，当循环体只有一句语句时，大括号可以省略不写。例如上面代码段 1 的 Ln 11. 代码段 2 中 do-while 语句的大括号也可以省略。当然有的时候，大括号不省略，层次更加清晰一些。

循环还没讲完，所以本弹不设有代码练习。更加丰富的练习将在下一弹呈现。

发布平台： GitHub | 阿儿法营 | 编程猫 | 有道卡搭

第一版日期：2019.12.5
第二版日期：2021.2.4