一起++　第 7 弹

循环（下）

作者：@孤言　　原审校：@Alex Cui

Loop, loop, 再 loop.　　鸽子，鸽子，鸽子 again.

第 1 节　for 循环

1.从重复执行 x 次谈起

Scratch 里常会用到计次循环。比如我们想要把 1-20 这 20 个数添加到列表中，可以像图 1 这样做。这个样子的循环有什么特征呢？那就是循环的次数和循环体中的语句密切相关。我们把这里的加入列表变成打印数字，利用 while 循环也可以实现重复 20 次的效果：

图 1　把 1-20 这 20 个数添加到列表中

```
int i=1;
while(i<=20)
{
    cout << i << endl;
    i++;
}
```

但是，这样的程序会带来麻烦，因为这里变量 i 的初始化是在 while 语句的外部，而我们以后常要声明一个只用于该循环的变量。此外这样的代码可读性也很差。

2.for 循环的结构

基于此，我将给出另一种循环语句——for 循环。它的语法格式是这样的：

```
for (初始化表达式; 条件表达式; 更新表达式)
{
    循环体;
}
```

图 2　狗头保命（来自网络）

看这个结构估计又要难倒一堆小朋友了。其实表达式的名字无所谓辣！我们用刚才的例子就可以说明道理了！

刚才的 while 循环可以改写成如下 for 循环：

```
for (int i = 1; i <= 20; i++)
{
    cout << i << endl;
}
```

初始化表达式：在 for 循环的循环体执行前先执行的语句，用于 for 循环参数的初始化，只在一开始执行一次，如这里的 `int i = 1`

条件表达式：循环体继续循环所需要满足的条件。条件成立则执行一次循环，否则跳出循环。如这里的 `i <= 20`

更新表达式：在每次循环体执行后执行一次的语句，常用于参数的加减。如这里的 `i++`

注意 请务必注意，for 循环的括号内，前两个表达式后要用分号来分隔，而更新表达式后没有分号！

关于大括号的省略和 for 循环的嵌套使用，与之前的 while 循环相同，就不再多说了。要补充的是，我们可以把 for，if 等整体看作“一条语句”。比如，对于下面这种结构：

```
for(int i = 1; i <= 10; i++)
{
    for(int j = 1; j <= 10; j++)
    {
        cout<<"Hello,";
        cout<<"world";
    }
    //这个位置没有多的语句了
}
```

这里，外层 for 循环的循环体只有一个内层 for 循环，而没有其它语句。此时我们可以将内层 for 循环看作一条语句，也适用省略的规则，也就可以省略为：

```
for (int i = 1; i <= 10; i++)
    for (int j = 1; j <= 10; j++)
    {
        cout << "Hello,";
        cout << "world";
    }
```

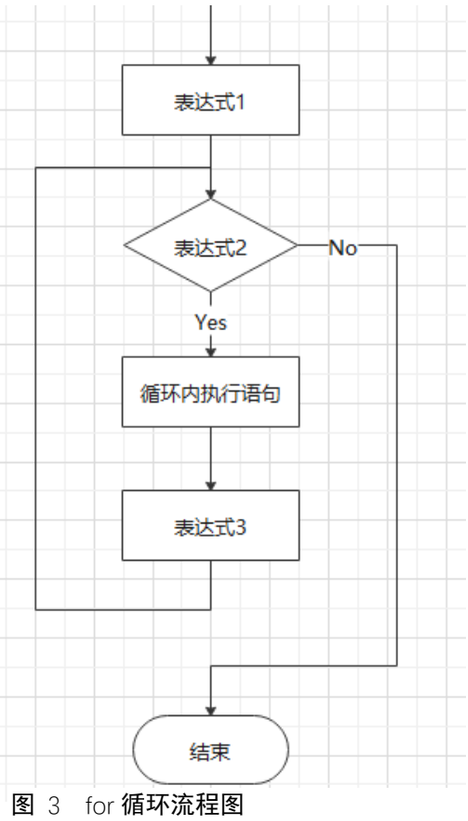

图 3　for 循环流程图

3.for 循环的执行方式

一般来说，for 循环是这样执行的：

①执行初始化表达式，使得控制变量得到一个值。

②判断条件表达式，若满足则执行一遍循环体，否则跳出 for 循环，执行其后面的语句。

③执行更新表达式一次，计算出控制变量所得到的新值。

④跳转到第②步。

流程图如右图的图 3 所示，其中的表达式 1-3 分别对应上述的 3 种表达式。

4.for 循环表达式省略 [拓展]

实际上对于 for 循环来说，以上 3 个表达式都可以不写。产生的效果是

①初始化表达式不写：那就不初始化呗，这个时候如果有循环变量 i，就需要提前在 for 的外部声明了。

②判断条件表达式不写：默认成 true，即条件表达式永远成立（虽然原理并非如此）。

③更新表达式不写：那就不更新呗。

注意 表达式虽然可以不写，但是分号还是不能省。

例如对于一个三个表达式都不写的 for 循环，可以写成：

```
for (;;)
{
    //一些东西
}
```

这里的两个分号是必要的。

5.for 循环的两个实例

以下展示几个 for 循环的实例。

（1）整数和问题：计算 1-2021 这 2021 个整数的和。当然，我们可以利用等差数列的计算公式来高效地计算，但这里用 for 循环来算一遍。

等差数列的求和

从第二项起，每一项与它前一项的差等于同一个常数的数列成为**等差数列**。这个常数叫做公差，用字母 d 表示。例如数列 1, 2, 3, 4, ... （公差为 1）或数列 5, 10, 15, 20, ...（公差为 5） 都是等差数列。

学过小学奥数或高中数学的同学们都知道，等差数列可以用公式求和。

求和公式一： ((首项 + 末项) * 项数) / 2

求和公示二：$na_1+\frac{n(n-1)}{2}d$ (n 为项数，d 为公差)

例如求 1~100 这 100 个整数的和就可以利用公式一：((1+100)*100)/2=5050

示例 7-1　1~2021 求和

```
#include <iostream>

using namespace std;

int main()
{
    int sum;
    for (int i = 1; i <= 2021; i++)
        sum += i;
    cout << sum;

    return 0;
}
```

用 i 来计次，对于每一个年份，将 i 累加到 sum，即可得到 1+2+3+...+2019+2020+2021 的和。

（2）图形打印：例如打印一个 n 行，每行 n 个星号的平行四边形的程序。

示例 7-2　图形打印

```
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;

    for (int i = 1; i <= n; i++) //i 控制行数
    {
        //每一行由空格和星星组成，空格的个数与行数 i 有关系
        for (int j = 1; j < i; j++) //控制每行的空格数，j<i 相当于 j<=i-1
            cout << "  ";
        for (int j = 1; j <= n; j++) //控制每行的星号数
            cout << "* ";
        //注意这里前后的两个变量 j 作用域不同。

        cout << endl;
    }

    return 0;
}
```

I/O：

```
5
* * * * *
  * * * * *
    * * * * *
      * * * * *
        * * * * *
```

程序由两层 for 循环嵌套构成，其中里层有两个 for 循环。

外层循环变量为 i，一直循环到 n，用于控制每一行的操作。

里层的第一个 for 循环控制在该行开头输出多少个空格，应该是 i-1 个，所以可以写成 j<=i-1，而更简洁的写法是 j<i，二者均可。里层的第二个 for 循环控制在该行打印 n 个星号。

这样的 for 循环嵌套初次学习有一定难度，需要认真领会。

第 2 节　循环控制语句

循环是个任性的家伙，不肯本本分分地做好枯燥的循环工作。有的时候啊，得闹点小情绪。这里就来向你展现“老朋友”循环的任性新面貌。

1.break 语句

图 4

循环跑累了，要来场大罢工，而 break 语句就充当了这一角色。通俗来说，当循环体中执行到 break 语句时，**会立即退出当前一层的循环，让该层循环提前结束。**

例如，如图 4 所示的这一循环中，“退出循环”积木充当了 break 语句的角色，程序会正常执行语句 1，然后如果来到了 break 语句处，则无论循环的条件表达式结果如何，也无论 break 语句后面还有没语句，都会直接跳出本层循环，而不执行语句 2。

注意 在循环嵌套中，break 语句也只跳出自己所在的那一层循环（以大括号为界），不会跳出多层循环。

算法竞赛中，break 语句通常会和 if 结合使用。实践中，通常在执行过程中遇到异常，或在某些特殊的循环中满足了某种条件时，我们需要使用 break 语句提前结束循环。

2.continue 语句

continue 在英文里是“继续”的意思。它的作用是，**立即停止当次循环，并开始下一次循环。**

图 5

简单地说，break 语句是某项作业做到一半不想做，出去玩了，而 continue 语句是某项作业做到一半不想做，我换个作业从头开始继续做。

因此，对于图 5 所示的结构，会执行语句 1，如果遇到 continue，则立即结束本次（而非本层）循环，不执行语句 2，开启下一次循环，即继续执行语句 1.

对于 for 循环来说，遇到 continue 语句后会结束当次循环，并且执行一次更新表达式，然后再进入下一次循环。

continue 语句什么时候用呢？孤言也说不上来，有用的时候自然有用辣！

第 3 节　特殊的条件表达式

讲到这里呢，循环也就学得差不多了。但是，还有一些比较零碎的东西要在这里碎碎念。

任何一个表达式都有对应的值。而对于条件表达式，它的值只有两种，即 true 和 false. 所以，对于一个循环语句，条件表达式的值是 true 就继续循环，是 false 就结束循环。

但在这里有一些特殊（调皮捣蛋）的成分充当条件表达式，我们要知道它们的值。

可以通过下面的程序来验证某个条件表达式的值（bool 型忘记了的话可以去看前面的教程）。

```
bool a;
a=(条件表达式);
cout << a;
```

程序会输出 1 表示 true，输出 0 表示 false.

通过尝试我们发现，当条件表达式为 0 时，输出为 0. 当条件表达式由一个非 0 数字组成时，输出为 1. 可见，在 C++中， 0 为 false，非 0 数字为 true.

因此，对于我们可以写出这样的结构：

```
while(1)
{

}
```

来使程序产生“重复执行”的效果，然后在内部通过添加 break 语句的方式来结束循环。

这种循环本身我们叫作**死循环**。

代码练习 4

Q1　欢乐 2021

输出 1-2021 这 2021 个整数，用空格隔开。

*注：在不作特殊说明的情况下，输出结尾包含多余的空白字符（空格、空行等）通常是被允许的情况。

Q2　您头没了

题目描述

某位用户统计了n天A营群中每天出现万能头的次数。假设这n天中万能头平均出现次数大于等于3.5次，就会被Alex发现，然后你头没了.

请编写程序判断你的头还在不在，在输出yes，否则输出no.

输入格式

输入第一行为一个整数n，代表天数，接下来的n行，每行一个整数，代表该天万能头出现的次数.

输出格式

输出共一行，为yes或no.

输入输出样例

输入 #1

```
3
2
3
3
```

输出 #1

```
yes
```

输入 #2

```
5
9
2
5
13
6
```

输出 #2

```
no
```

说明/提示

样例1：(2+3+3)/3<3.5，头还在.

样例2：(9+2+5+13+6)>=3.5，头没了.

对于100%的数据，不超过int型.

万能头

学习 OI 的同学应该对**万能头**并不陌生。由于评论区对它的讨论实在太多，所以这里就顺带提一下。

它的写法是：

```
#include <bits/stdc++.h>
```

仅此一行，可包含 C++中“所有”的头文件，如

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cmath>
...
```

但是，万能头并不属于 C++语言标准，为保证教程严谨科学，本教程中**绝对不会使用万能头来代替常规的包含头文件。**

万能头一定程度上会降低编译效率。在某些算法竞赛中不被允许使用。

Q3 输出三角形

输入格式

一个整数n.

输出格式

共n行，是一个如样例所示的三角形.

输入输出样例

输入 #1

```
3
```

输出 #1

```
@
@@
@@@
```

输入 #2

```
5
```

输出 #2

```
@
@@
@@@
@@@@
@@@@@
```

说明/提示

对于100%的数据，n<=50.

Q4　菲波那切数列

题目背景

斐波那契数列（Fibonacci数列），又称黄金分割数列，因数学家莱昂纳多·斐波那契以兔子繁殖为例子而引入，故又称兔子数列。在现代物理、准晶体结构、化学等领域，斐波纳契数列都有直接的应用。

斐波那契数列指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、……我们通常将0作为数列的第0项，第一个1作为数列的第1项。从第二项起，该数列的每一项都是前两项的和。

题目描述

计算菲波那切数列的第n项

输入格式

一个正整数n

输出格式

一个整数，表示菲波那切数列的第n项

输入输出样例

输入 #1

```
1
```

输出 #1

```
1
```

输入 #2

```
6
```

输出 #2

```
8
```

说明/提示

对于100%的数据，$n \leq 100$

Q5　亲爱的质数

判断某正整数是否为质数，是则输出 Y，否则输出 N.（质数是啥不知道的可自行询问度娘）

答案

Q1 一道大水题。

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

using namespace std;

int main()
{
    for(int i=1;i<=2021;i++)
    {
        cout<<i;
        cout<<" ";
    }

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

Q2 请大家不要使用万能头。

```cpp
#include <bits/stdc++.h> //看不见，看不见~

using namespace std;

//n 为天数，sum 为总出现次数
int a, n, sum;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) //输入接下来的 n 行
    {
        cin >> a; //用 a 临时记录第 i 天的出现数
        sum += a;
    }

    /*判断头还在不在，注意这里是两个 int 型作除法，
     要用 1.0 作显式的类型转换，不然结果是一个整数*/
    if (1.0 * sum / n < 3.5)
        cout << "yes";
    else
        cout << "no";

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

Q3 外层循环控制控制行，内层循环控制每行的@打印，请注意两个 for 循环中条件表达式的区别。内层 for 循环循环体只有一句，所以这里省略了大括号，第二个输出语句不属于内层循环。

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

using namespace std;
```

```
//n 为天数，sum 为总出现次数
int n;

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) //输入接下来的 n 行
    {
        for (int j = 1; j <= i; j++) //第 i 行输出 j 个@
            cout << "@";//注意@作字符串，要使用引号引起
        cout << endl; //换行
    }

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

Q4 对于初学者来说有点难了。如果写不出来也没关系，可以先体会一下答案。用 a,b 来记录数列中相邻的两项，先初始化为第 1 和 2 项，每次循环可以更新一项，也就是第一次循环变为第 2 项和第 3 项，依次类推。n<=2 时不循环，直接输出初始化好的 b，其值恰好为 1.

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

using namespace std;

int main()
{
    int a, b, c, n;
    a = 1;
    b = 1; //a,b 初始值赋为数列的第 1、2 项

    cin >> n;

    /*c 储存第 i 项，每个数都等于前面 2 个数之和
      注意临时变量 c 起到的交换作用*/
    for (int i = 3; i <= n; ++i)
    {
        c = a + b;
        a = b;
        b = c;
    }

    cout << b;

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

以后我们还将使用“递归”的方法更高效、更直观地来解决这个问题。

此外，菲波那切数列存在通项公式，可以直接用公示计算得到。

Q5 本题可参考算法微波系列教程的质数判断算法。（该系列教程在发稿时暂未发布，还在筹划中，这里做个宣传 qwq）判断质数其实有多种方法，这里采用一种比较直观（同时意味着效率比较低）的算法。

思路是将这个数从 2 开始试除，一直除到这个数减一，如果其中任意一个数可以除得尽它，那么不是质数，否则就是质数了。

比如对于数字 35，

35%2，不为 0

35%3，不为 0
35%4，不为 0
35%5，为 0，判断出 35 不是质数，退出循环。
再如数字 13，
13%2，不为 0
13%3，不为 0
13%4，不为 0
……
13%11，不为 0
13%12，不为 0
因此可知 13 是质数。
由此，代码如下

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

using namespace std;

int main()
{
    int n;
    cin >> n;

    //变量 isPrime 用于记录是否为质数，先默认是质数
    bool isPrime = true;

    for (int i = 2; i < n - 1; i++) //从 2 到 n-1 开始试除
    {
        if (n % i == 0)
        {
            isPrime = false;
            break; //发现不是质数，没有继续循环的意义了，跳出循环
        }
    }

    if (isPrime)      //将布尔变量直接作为条件表达式
        cout << "Y"; //注意要用引号引起
    else
        cout << "N";

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

实际上，把 for 循环的条件表达式改成 i<=sqrt(n)，并包含头文件 cmath，可以提高运行效率。你可以想一想为什么可以这样做。

第一版日期：2020.1.20
第二版日期：2021.2.13

发布平台：GitHub | 阿儿法营 | 编程猫